Universität Bremen

University of Bremen
Faculty 3 - Mathematics/Computer Science

Bachelor's thesis

Security Analysis of the TRÅDFRI Smart Home System

Kelke van Lessen
matriculation number: 4485766
September 24, 2022

1. supervisor: Dr. habil. Karsten Sohr
2. supervisor: Prof. Dr. Ute Bormann

Eidesstattliche Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die selbstständige und eigenständige Anfertigung versichert an Eides statt:

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used. The independent and unaided completion of this thesis is affirmed by affidavit:

Bremen, den 24. September 2022

Kelke van Lessen

Danksagung

Ich möchte an dieser Stelle einen besonderen Dank an Dr. habil. Karsten Sohr aussprechen, der ein hervorragender Betreuer war und sich der Erstbegutachtung annimmt. Ebenfalls danke ich Prof. Dr. Ute Bormann für die Zweitbegutachtung.

Auch danke ich meinen Eltern, die mein Studium all die Jahre unterstützten und immer ein offenes Ohr für mich hatten.

Außerdem möchte ich meiner Freundin Maite danken, die mich ebenfalls während des Studiums begleitete, unterstützte und Korrekturhinweise gab.

Des Weiteren danke Jo und Jannes ebenfalls für die hilfreichen Korrekturhinweise.

Auch bei der OTARIS Interactive Services GmbH möchte ich mich herzlich bedanken, da der Umfang dieser Arbeit ohne das Smartphone, was sie mir bereitstellten, nicht möglich gewesen wäre.

Contents

List of abbreviations

APK	Android Package
CA	Certificate Authority
CBC	Cipher block chaining
DHCP	Dynamic Host Configuration Protocol
DTLS	Datagram Transport Layer Security
GPL	GNU General Public License
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
IPA	iOS/iPadOS application archive file
JSON	JavaScript Object Notation
LAN	Local area network
MAC	Media Access Control
MIT	Massachusetts Institute of Technology
MITM	Man-in-the-middle
NTP	Network Time Protocol
PKCS	Public Key Cryptography Standards
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
UDL	User Datagram Protocol
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
XML	Extensible Markup Language

Chapter 1

Introduction

The creation of the World Wide Web connected continents and people far apart. It also gave birth to a new type of device sector that links the home itself to the internet, called smart home. These smart home devices are internet-connected home appliances like blinds, lights, or even coffee machines that can be controlled via a smartphone. This internet connectivity makes a great deal of automations, remote monitoring and physical security possible.

Smart home systems are steadily gaining in popularity[1] and with an increasing number of internet connected home devices, the aspect of security is becoming more and more relevant.

A smart home alarm system is great, but only until a hacker disables it. Smart lights are convenient, but only until a hacker shuts them off. Smart home devices need to be developed with special attention to security, as they are often an integral part of a person's life.

Smart home devices are usually not sold individually, but instead grouped in a bundle that is expandable. A group of devices manufactured by one company is called a system.

The following thesis will analyze the security of IKEA's smart home system TRÅDFRI[2]. The TRÅDFRI system is best known for their lights, but does also offer blinds, air purifiers, speakers and much more.

As a similar security analysis was performed by Tristan Bruns in 2018, and the goal for this thesis is to analyze the TRÅDFRI system four years later, to evaluate changes, and possibly make new findings (Bruns, 2018).

Three main aspects of security will be looked at for this analysis. These are the app, that IKEA developed for the TRÅDFRI system, any connections to IKEA servers, and the local network traffic. What this thesis will not analyze are any hardware-level security aspects, and the implementation of ZigBee.

[1] https://statista.com/chart/24876/smart-home-security
[2] https://www.ikea.com/de/de/customer-service/product-support/smart-lighting/

Chapter 2

Technical background

This chapter will give an explanation of any terms used as part of this thesis, which are necessary for the methodology and findings of this thesis. These terms include an introduction to security's core concepts, various network terms, relevant protocols, and lastly the types of attacks that were performed during the analysis.

2.1 Security

2.1.1 Concepts

There are many concepts within IT-security, three of which are the most fundamental to this thesis; when combined, they form a secure environment to exchange data within. These three concepts are called confidentiality, integrity and availability, which can fittingly be abbreviated with the acronym CIA (Russell and Gangemi, 1991).

Confidentiality is provided, if no one except for the intended party is able to access a given set of information. This can be achieved, for example, by limiting access rights, ensuring safe passwords and encrypting data (Cabric, 2015).

Integrity is the principle of ensuring that information cannot be altered by any unauthorized party, without the alteration being detected (Cabric, 2015). An example of a breached case of integrity would be a company sending an invoice to one of its customers, but an attacker modifying the invoice's banking details, and receiving the money in its own account. A system that provides integrity would not allow the malicious invoice to reach the customer, or if it would, the customer could identify the modification.

Availability is given, if a system keeps its information accessible to all its users, and in the event of a disaster, a system with adequate availability must recover quickly and completely (Russell and Gangemi, 1991).

2.1.2 Symmetric cryptography

There are two types of cryptography. One is symmetric, the other is asymmetric. Given a scenario where two partners, Alice and Bob, want to securely exchange information, symmetric cryptography means that both partners know and use the same key for encrypting and decrypting data. This is a symmetric key, which is the intuitive way of encrypting data most people are familiar with.

A very basic symmetric key would be '$n + 1$', where 'n' is a letter, which is changed to the following letter '+1' in the alphabet. '`a`' becomes '`b`', '`d`' becomes '`e`', etc. If Alice and Bob want to communicate the word '`Hello`', it becomes '`Iffmp`', which is not readable to a non-trained eye. This symmetric key is of course very weak, and even if in some scenarios it might secure communication when literally talking from person to person, a computer would solve it in no-time. To create secure symmetric keys in real-world digital applications, standards like the Advanced Encryption Standard (AES) are used (NIST, 2001).

Symmetric cryptography does, however, have a problem: key exchange. The best symmetric key ever made does not help in securing communication, if only one of the partners knows it. Exchanging symmetric keys on an insecure channel is not an option, as it cannot be guaranteed that only the intended partner received the key. This contradicts integrity. Meeting up with the intended partner in real life and exchanging symmetric keys by hand does work, but is obviously not a sufficient solution for the ever-growing size of the internet.

2.1.3 Asymmetric cryptography

To solve the problem of exchanging keys on an insecure channel, asymmetric cryptography was invented. This type of cryptography uses two separate types of keys, public keys for encryption and private keys for decryption. In a system of asymmetric cryptography, every participant has its own set of one public and one private key, which are unique to them. The public key is openly available for everyone, and used for encrypting data directed to the key's creator. After successful delivery, the creator uses their private key, which only they have access to, for decryption.

Given a scenario where Alice wants Bob to know her location, she uses Bob's public key to encrypt that she is located in Berlin. Bob receives the encrypted message and uses his private key to decrypt the aforementioned message.

Mathematically, both public and private keys are linked, but knowledge of the public key does not infer the private key. The public key can be regenerated by the private key, but not vice-versa. In actual software applications, the Rivest–Shamir–Adleman (RSA) standard is often used (Rivest et al., 1978).

For this concept to work, it is of great importance that the private key is safe, only the intended person has access to it, and that any public keys that are used can be trusted.

This is where even asymmetric cryptography shows its problems. Firstly, the usage of an asymmetric form of encryption is more demanding on a performance level, so for systems with very restricted hardware resources this is not always possible to implement. Secondly, public keys cannot be inherently trusted, which enables man-in-the-middle attacks (Section 2.4.4). Early adoptions of asymmetric cryptography, like the Needham-Schroeder protocol, are susceptible to this type of attack (Lowe, 1995), so a similar problem as with symmetric cryptography becomes apparent; public keys need some form of validation.

2.1.4 Certificates

Certificates provide a solution for trusting public keys. The fundamental concept is that a trusted party validates that the owner of a public key is who they say they are. A limited number of certificate authorities exist, which can be considered safe. These certificate authorities issue a root-certificate, which then validates a chain of intermediary certificates, which ultimately validates an individual, website, company, etc. For browsing the web, the root-certificates of trusted certificate authorities have to be preinstalled on the many internet-connected devices that are part of our daily life (Eckert, 2008).

Certificates are commonly issued in a X509 v3[1] form. The structure of this type of certificate includes various attributes, most importantly in the scope of this thesis are the following: CommonName(CN), OrganizationalUnit(OU), Organization(O) (Boeyen et al., 2008; Oracle, 2022).

2.1.5 Certificate pinning

Certificate pinning is a security measure, which allows only specific certificates to be trusted. If more than one certificate is pinned by an application, then that application holds a *pinset* (Walton et al., 2022). Using certificate pinning is an effective way of preventing against man-in-the-middle attacks (Section 2.4.4), as any malicious certificate can safely be ignored. To be able to use certificate pinning, the specific certificate has to be present on both communication partners. For any application that intends to use certificate pinning, the specific certificate has to be bundled with the application itself.

Forward Secrecy

To understand forward secrecy, one must first understand what session keys are. Session keys are relevant in communication protocols, where they are used to encrypt data, but

[1] https://datatracker.ietf.org/doc/html/rfc5280

only for a finite amount of time. A session is started if a communication partner establishes a connection to a different partner, and ends if this communication is terminated.

Forward secrecy is provided if the knowledge of an encryption key does not reveal any future session keys. Backward secrecy provides the opposite of forward secrecy, as knowledge of one key does not reveal any prior session keys. These two concepts can be important, as without them, an attacker is theoretically able to record a communication and decrypt it later, getting access to all prior and future information that is sent along this communication channel, even though session keys were used. If an attacker compromises any encryption key in a protocol with forward secrecy in place, only the data actually encrypted with the compromised key is revealed to the attacker.

2.2 JavaScript Object Notation

The JavaScript Object Notation[2] is abbreviated with JSON, and provides a standardized format for exchanging data across different platforms. Even though JSON does have its roots in the JavaScript programming language, it would be wrong to assume that JSON is limited in any way to only being used in JavaScript (Bassett, 2015). JSON is compatible with most programming languages, and quite simple to read and to write. It uses a key-value notation, which means that every entry in a JSON-formatted object consists of a string as the identifier, and a value, or values, in the typical data-structures like integers, arrays, strings, etc. If a water bottle were to be displayed on a web-shop, it could have the following notation internally:

```
"product":{
        "name":"water bottle 500ml",
        "color":"white",
        "volume":500,
        "price":{
            "currency":"euro",
            "amount":15
        }
    }
}
```

Some developers prefer JSON over its alternative: *Extensible Markup Language (XML)*, among other reasons, for its more efficient writing style, ease of reading, and support for arrays (W3Schools, 2022).

[2] https://datatracker.ietf.org/doc/html/rfc8259

2.3 Networking

2.3.1 Local Area Network

A local area network is a network, that exists separate from the internet. Local area network is commonly abbreviated with LAN, or local network. A LAN will usually be connected to the internet, but its members can communicate with each other even if the LAN does not have internet access. Local area networks are present in most homes, where a router is the coordinator, and all phones, laptops, PCs, etc. connect to it.

2.3.2 Ethernet

Ethernet is a technology, that is used for creating local area networks, and was standardized by the Institute of Electrical and Electronics Engineers (IEEE), specifically by the IEEE 802.3[3] working group. Ethernet is most commonly used in local networks, where its members are connected with cables. In this thesis, the term *Ethernet cable* refers to the RJ45[4] type of connector, that is the most common in personal devices.

2.3.3 ZigBee

ZigBee[5] is an open wireless communication framework used for personal area networks, that is controlled by the non-profit corporation *Connectivity Standards Alliance*[6] and based on the IEEE 802.15.4[7] standard. Personal area networks have a low transmission range, and are, as the name implies, designed for use-cases where an individual wants to connect its personal devices with each other, or with the personal device of another individual. This type of network also includes wireless standards like Bluetooth[8], just like wired standards like the Universal Serial Bus (USB)[9]. ZigBee is very commonly used in smart home devices, where its key features are particularly relevant. One of these is the optimization for low power consumption, which is especially important, since a house that is fully outfitted with a smart home system can often have dozens of devices. Another useful feature of the ZigBee standard is the network-mesh its devices create; every active ZigBee device acts as a repeater, boosting the signal strength it receives. This allows ZigBee networks to stretch across multiple rooms and floors, even though a ZigBee-message sent by a single device could not reach the same distance (*Zigbee FAQs — Frequently Asked Questions* 2022). A problem of ZigBee, however, is its comparatively low interoperability, as full

[3] https://www.ieee802.org/3/
[4] https://docs.rs-online.com/a95b/0900766b8002734c.pdf
[5] https://csa-iot.org/all-solutions/zigbee/
[6] https://csa-iot.org/
[7] https://standards.ieee.org/ieee/802.15.4/7029/
[8] https://www.bluetooth.com/
[9] https://www.usb.org/

functionality is often only provided when a ZigBee network consists of devices from one-, single-, manufacturer (Danbatta and Varol, 2019).

2.3.3.1 Touchlink commissioning

Touchlink commissioning is a procedure that is used for both creating a ZigBee network, and for adding new devices into an existing network. Touchlink is one of the two commissioning modes that ZigBee 3.0 specifies (Morgner et al., 2017). Connecting a new device to an existing network is as simple as holding the new device directly next to a remote, and starting the Touchlink process. Even though Touchlink commissioning was found to have major vulnerabilities (Morgner et al., 2017), it is still used as of 2021, and still insecure (Wodtke, 2021).

2.3.3.2 Groups

Since the number of installed devices in a smart home system can be quite high, controlling them individually can become somewhat cumbersome. For this reason, smart home systems usually allow users to group devices together. A ZigBee group could, for example, consist of all lights that are controllable in a given room. This feature is certainly not exclusive to ZigBee, nor smart home systems, but especially common in this field.

2.3.4 Constrained Application Protocol

The Constrained Application Protocol (CoAP) is a machine-to-machine communication protocol, designed for large networks of small devices, such as sensors. These devices have limited processing power and tend to operate in networks of large scale, which requires this protocol to be resource efficient (Thangavel et al., 2014). CoAP is a protocol that aims to accomplish these tasks, while still providing reliability and security. CoAP does support the common request-response architecture, but a different, more unique architecture is also supported, in which a client subscribes to a certain resource, and automatically receives data, as soon as a new message is published on that resource (Thangavel et al., 2014).

2.3.5 Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP)[10] is an essential part of the internet, as it is the backbone of the World Wide Web. It allows the download, upload, and modification of data between computers that are located all around the world. The core syntax of HTTP is a simple, as commands like *GET*, *POST*, and *PUT* can be used to send and retrieve information (Mozilla, 2022a; Russell and Gangemi, 1991). HTTP requests all

[10] https://www.rfc-editor.org/rfc/rfc2616

have a Uniform Resource Locator (URL), that specifies the exact target location. This URL has a predefined composition, that can be seen in Figure 2.1. The two most relevant components of a URL, in the scope of this thesis, are the *Scheme*, and the *Domain Name*, as the domain name reveals the server, as well as the provider of the aforementioned server, and the scheme reveals the chosen protocol.

Figure 2.1: Composition of a URL (Mozilla, 2022c)

Additionally to the URL, HTTP requests are split into a header data, and body data. The header includes management data, which can, for example, be accepted data formats, preferred languages, or simply the date and time at which the network package was sent. The body of an HTTP request is reserved exclusively for content data. This content could for example be a document file, or JSON data.

For all the usefulness that HTTP provides, one major limitation, however, still has to be acknowledged: security. HTTP is an unencrypted protocol, which means that it is not suitable in scenarios where sensitive information, like payment details or login data, is transferred.

2.3.6 Transport Layer Security

The Transport Layer Security (TLS) is a protocol used very broadly for encrypting data on the internet, the most notable protocol being HTTPS, which is the encrypted and secure version of HTTP. The first versions of the protocol, we now know as TLS, were originally named SSL. The term SSL is still very commonly used to refer to the TLS protocol, even though the old SSL protocol has been deprecated. TLS uses a four-way handshake to establish a secure, encrypted connection. During the handshake procedure, the server sends its certificate, which the client uses to securely transmit a key that encrypts the following network traffic. A pre-shared key can, however, also be used for encryption. TLS is based on the underlying Transmission Control Protocol (TCP), that serves as a pillar for the Internet Protocol (IP), without which the internet would not exist.

2.3.7 Datagram Transport Layer Security

The Datagram Transport Layer Security (DTLS) protocol is a protocol that replicates the features of TLS on the User Datagram Protocol (UDP), that also serves as one of the pillars for the Internet Protocol. UDP is a faster alternative to TCP, but does not have all the features and lacks reliability. As DTLS aims to provide the same functionality that TLS does, the use of UDP requires some additional features. DTLS, just like TLS, allows

the use of a pre-shared key for encryption. This pre-shared key is of special interest for the following security analysis.

2.3.8 Subnet

A subnet is a single network that is part of a larger network structure. A subnet has members, just like a basic network does. These members, however, are confined into an isolated environment. The reason to split members of a network into subnets, in favor of one interconnected network containing all members, can be several, two of which are security and reducing network load.

A higher level of security can be achieved because a router is usually what connects a subnet to the larger network and as this router is able to manage all incoming and outgoing traffic, the subnet can be controlled, so that only certain other subnets are accessible, or none at all. In the event of a virus-infected member within the subnet, the possible spread as well as access to other members within the overall network is greatly reduced.

A reduced network load can also be achieved, as the subnet routers are able to read the source and destination addresses of the individual network packets. This means that if there is traffic between two members of a single subnet, the router is able to contain the aforementioned traffic within the subnet instead of each packet taking the longer route to a central network router and back.

2.3.9 Virtual Private Network

A virtual private network is a piece of infrastructure that is used for tunneling network traffic, and is usually abbreviated with VPN. Users of a VPN are virtually placed into an existing local network, which all their traffic is tunneled through in a secure, encrypted, way. The host of a VPN exposes its local network to any authorized users, for them to connect to, from all around the world. Using a VPN has three considerable implications. Firstly, applications running on the user's machine have no way of differentiating if they are inside or outside a local network, which can be beneficial for work-from-home scenarios, where certain services are only accessible from within a company network (Eckert, 2008). Secondly, since users of a VPN get the IP address of the network they are connected to, services on the internet expect them to physically be in the same location as the VPN host. This can allow users to access services, which are restricted in the network, or country they are currently located in. Lastly, a VPN provides privacy for users that do not trust their current network, as an active VPN tunnels all their traffic to a potentially safer network.

2.3.10 Network Proxy

A network-proxy is similar to a VPN, in the way that it also provides a point, which network traffic directed through, before it reaches a destination on the internet. This, just like a VPN, creates the effect that services on the internet can only see the IP address of the proxy, and not the address of the user. A proxy, however, lacks some features of a VPN, and is generally regarded as more basic in its functionality. One major difference is, that network traffic between the user and a proxy can, but does not have to, be encrypted. This potentially limits privacy gains, and even makes it possible for the proxy to perform malicious activities on the network traffic, but encrypted proxies do exist (Choi et al., 2020). Another major difference is, that a proxy works on the application level, as opposed to a VPN, which works on the operating system level. This has the consequence that applications can simply ignore a proxy, which is not the case for a VPN. One particular use of a proxy can, however, not be accomplished by a VPN: caching. Non-sensitive information can be cached on a proxy when a user downloads data. When the next user requests the same data as a previous user, the proxy can instantly respond with the cached data. This considerably lowers network traffic load on the provider of the original data and can be used to increase transfer speeds. As proxies redistribute data from one user, to others, they cannot be used for caching sensitive data, as that would violate confidentiality.

2.4 Attack types

2.4.1 Social engineering

In social engineering, an attacker exploits the lack of security awareness, which humans often have, to perform malicious tasks (Eckert, 2008). Uncarefully opening malicious attachments in emails was, and still is, an effective way of spreading malware. Even bypassing SMS based two-factor authentication can be as easy as asking a potential victim's telephone company for a second SIM card. In a seemingly secure environment, humans are usually the weakest link, which is why social engineering is an especially dangerous type of attack (Gupta and Sharman, 2009).

2.4.2 Penetration testing

Penetration testing, which is often abbreviated with pentesting, is a form of attacking a specific subject, and finding any security vulnerabilities they might have. Penetration testing is part of the broader spectrum of hacking. When penetration testing a company, the tester usually tries to gain as much access to servers, financial data, passwords or anything other, that is not supposed to be publicly available.

All too often, insecure passwords are still used, which can make the job of the penetration tester a lot easier. As the same passwords are then also often used on different servers and services, the control a tester can gain just by simple neglect of password strength is immense.

The difference in a penetration tester as opposed to a hacker with bad intentions, is, that a penetration tester does not take advantage of the control they gain, but they inform the affected subject about their findings.

In practice, a security research company might offer the service of penetration testing to a different company, so that tested company can fix any vulnerabilities they have, and the chance of dangerous attacks by hackers exploiting these vulnerabilities decreases.

2.4.3 Reverse Engineering

Reverse Engineering is a technique for understanding how something was made with just having access to the product itself. In the context of this thesis, reverse engineering is used to understand which security measures were implemented into the TRÅDFRI system without contacting engineers or developers, who can simply explain how it was made. Reverse engineering in general does not have to be about security, or even IT in a broader sense.

A classic example of reverse engineering in the real world is when factories find out about a well-selling product, and want to reproduce it on their own. They then get hold of the original product and try to understand how it was made. The methods used for reverse engineering a physical product, like laser-scanning, are obviously different to the methods used in software reverse engineering, but breaking apart products, to see what is inside then is actually similar on an abstract level to decompiling.

2.4.3.1 Decompilation

Decompilation is one form of reverse engineering, and refers to a process where an application that is called a decompiler is used to convert low-level code into high-level code. What this means, is that executable files, such as .exe files on Windows, .app files on macOS, or .apk files on Android, are converted into human-readable code. Doing so gives an insight into how exactly an application was made, including the code structure, if some sort of certificate pinning was used, and which servers are communicated to.

This is the opposite to a compiler, which converts written, high-level code into machine code/low-level code. A decompiler tries to undo this operation, but as one can imagine, this does not always work perfectly, and developers have some tricks up their sleeve, to make this type of reverse engineering more difficult.

One challenge in interpreting decompiled code is that any in-code comments, which could include useful documentation, are ignored by a compiler, which means that they do not end up in the outputted executable. As a result of that, comments are not accessible in decompiled code.

Another challenge in reading decompiled code is obfuscation. Developers of an application can instruct their compiler to obfuscate any classes, functions, etc. When a decompiler then tries to convert the executable file back into readable code, the meaning of code-snippets are not easily intelligible. For instance, a function which might have been originally named *requestUpdate* in its obfuscated form could be renamed to `a4`.

2.4.4 Man-in-the-middle

A *man-in-the-middle* attack, abbreviated as mitm, is an attack, where an attacker intercepts a connection and fakes that they are the original participant to both sides of the connection, which is illustrated in Figure 2.2.

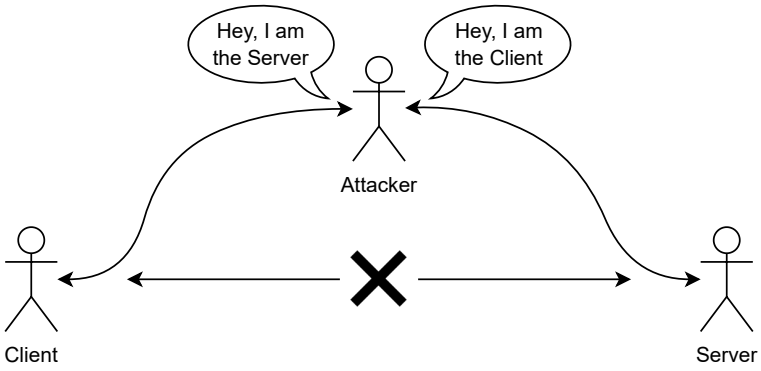

Figure 2.2: Network-setup for a man-in-the-middle attack

The following example-attack demonstrates the concept:

A client requests a file from a server, but while the response-file is supposed to reach the client, the attacker intercepts the response. With an intercepted response, the attacker can now replace the aforementioned file with a malicious file and send that to the client. This leads the client into believing the received attacker's file, was sent by the server. This in turn can potentially infect the client's system with a virus, jeopardizing confidentiality, integrity, and potentially even availability.

Another example of a man-in-the-middle attack is a client and a server exchanging encryption keys, and while the encryption itself might be secure and the cryptographically

perfect, the attacker performing the man-in-the-middle attack can intercept these encryption keys and capture all following passwords or other sensitive data

As you see in both of these examples, the attacker needs full access to the connection. This means that a man-in-the-middle attack can only be performed on connections where encryption is not already present. Even connections where asymmetric encryption is used in conjunction with certificates are prone to this type of attack, as an attacker can replace both the certificate and the public key. The preferred way of preventing man-in-the-middle attacks is with certificate pinning.

A scenario for such an attack would be the following:
As the client asks for the server, the attacker pretends to be the server towards the client. The attacker, however, simultaneously begins a connection to the genuine server and therefore acts as the original client. Using this method, the attacker is able to relay all traffic through itself, decrypt the traffic, re-encrypt the traffic and then send it to its original destination.
This type of attack is possible on HTTPS traffic only if the client's locally stored certificates are tampered with. This is not often possible in real-world conditions.

2.4.5 Replay Attack

A *replay attack* is a type of attack, where a legitimate message is recorded, and resent at a later point in time (Russell and Gangemi, 1991). In an insecure system, this allows an attacker to gain unauthorized access, for example, to open a garage door by replaying a previously recorded signal of a real garage door remote. Network protocols can have similar vulnerabilities, which is why man-in-the-middle software commonly has tools included, that are capable of replaying already sent network packages.

2.4.6 Network Sniffing

Network sniffing is, compared to a man-in-the-middle attack, more subtle, but also not actually an active attack. Rather than actively attacking a network connection, network traffic is simply recorded. This recorded traffic can give various insights into the services being used, the active clients in a network, and much more.

A very popular tool for network sniffing is called Wireshark[11]. Wireshark can see, as well as capture, all traffic that reaches a computer's network card in real-time. Using network sniffing, unencrypted traffic, which might include sensitive data, can be recorded for later use. Simply sniffing network traffic can, however, not reveal encrypted information. Wireshark does feature decryption, but the decryption key must be known. Since HTTPS, and

[11] https://www.wireshark.org/

TLS in general, have become the de facto standard for the World Wide Web, fewer people are vulnerable to network sniffing attacks.

Network sniffing can, however, even today still capture sensitive data of deprecated web services and also often recognize the network protocol being used, which is of big importance in this analysis. An attacker can also record network traffic and possibly decrypt it at a later point in time.

2.5 Phone rooting

When using a regular Android device, some settings, folders, and files are not accessible to the owner. Under daily usage conditions, this is not a problem. In some cases, however, these administrative rights are needed, for example, to decrypt and record encrypted network traffic.

When buying a usual Android phone, the user is provided with just enough system privileges to perform all the regular tasks, a user would want to do, like taking calls, capturing images, and accessing the device's camera and so on. These privileges, however, are limited, in order to protect the owner as well as the device itself and even the apps and protocols running on the device. A rooted phone, however, elevates the user's privileges and allows total system control. On a rooted phone, everything on the device is modifiable, which enables basic tasks like changing emoji styles but also more major changes like removing all ads from specific apps, modifying system files, or most importantly in the context of this thesis, importing security certificates into the key store of the device and enabling man-in-the-middle attacks. Rooting a phone is inherently difficult, and usually even detected by the operating system. A rooted device cannot be completely trusted by installed apps, which is why apps like banking apps often refuse to work. The act of rooting a device on its own is also risky as there is the potential for a step in the rooting process to fail, which can lead to the permanent, non-recoverable failure of a device. This is called hard bricking, as the device practically becomes an expensive paperweight.

Chapter 3

Methodology

The goal of this thesis was to dynamically as well as statically evaluate relevant parts of IKEA's TRÅDFRI system regarding software security. The following chapter will introduce the TRÅDFRI system, explain the two aforementioned concepts, and give an in-depth insight into how the system operates, how the analysis was prepared and which software was used.

The TRÅDFRI smart home system can be split in three parts. One are the light bulbs, air purifiers, and motion sensors and other smart devices. These use the ZigBee framework for communication. The second part of the TRÅDFRI system is the 'Home smart' app, that IKEA developed for Android and iOS. Using this app, users can control all their TRÅDFRI devices, group them together, and initiate updates. The app uses the smartphone's local area network, and internet access for all of its communication. Since the smart home devices use ZigBee, and smartphones do not support ZigBee, a third, possibly most important part of IKEA's smart home system is the TRÅDFRI gateway. This gateway serves as a middleman, accepting commands from users of the Home smart app in the local area network, and sending these commands to smart home devices using ZigBee. As the gateway serves as an integral part of the TRÅDFRI system, it is of special interest for this analysis. Remotes are also available as part of the TRÅDFRI system, and allow controlling lamps, and other devices with the simple push of a button. These remotes communicate using ZigBee, which allows remotes to control lights, even if the gateway is turned off.

While performing static analysis on the TRÅDFRI system, no code will be executed, but the Home smart app will be decompiled. This decompiled code will then be used to evaluate the security measures IKEA have taken into account while developing their app.

For dynamical analysis of the TRÅDFRI system, all of its components, including the mobile app, were tested while actively using them. This meant testing the system in the same way an ordinary user would, but also attacking it in various ways, in order to discover any possible security vulnerabilities. As part of this dynamic analysis, the local communication between the bridge and the app as well as communication from the app to external servers was analyzed.

The TRÅDFRI system used in this analysis was bought at IKEA Brinkum[1] on the 13th of November 2021 and consists of a gateway, three remotes, as well as three light bulbs, of which two are of the E14 type, and one is of the E27 type. All these parts were installed on a wooden panel to enable easy access and a nice overview of all included parts as can be seen in Figure 3.1. This installation type also enables portability of the system, as only one Ethernet and one power plug is required to power up all devices.

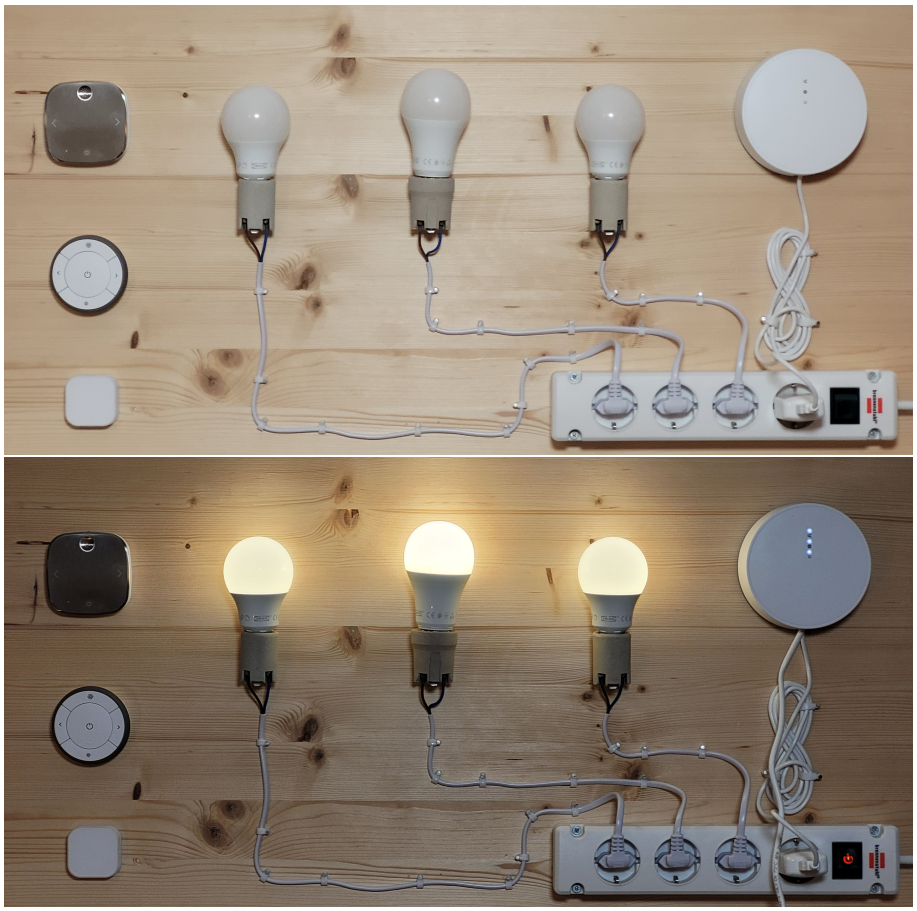

Figure 3.1: All purchased TRÅDFRI devices installed on a wooden panel

Figure 3.2: The gateway in more detail

[1] https://www.ikea.com/de/de/stores/brinkum/

Figure 3.3: The remotes in more detail

3.1 Installation

Installing the components is straight forward. The bulbs have to be screwed into their corresponding sockets, just like any traditional light bulb. The gateway has to be connected to a power socket via the included power supply, as well as connected to a network router via an Ethernet cable, which is also included.

After that, only the remotes are left to install, which can either be screwed or taped onto a wall, or in the case of this analysis, a wooden panel. The remotes have a plastic tab on the back, which has to be removed as to activate the included batteries.

With all components physically installed, we turn to software.

The only way IKEA intends the system to be configured, is via the aforementioned mobile app called *IKEA Home smart*. This app was installed on a smartphone, kindly provided by OTARIS Interactive Services GmbH, a Security research company located in Bremen, Germany. The smartphone is a rooted *Samsung Galaxy S6*, which runs version 7.0 of Android, the oldest currently supported android version the IKEA Home smart app allows, as stated within the app itself, see Figure 3.4. At this current point in time, this should thankfully not be a problem, but future updates of this app will likely not support this version of Android anymore.

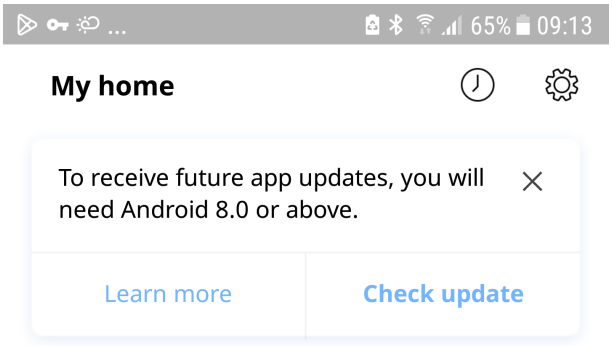

Figure 3.4: Android 8.0 will soon be required

The app was also installed on a *Huawei P30 Pro*, to evaluate differences in the way any connections are handled on a rooted, as opposed to a non-rooted Android device. This *Huawei P30 Pro* was running Android version 10.

For the installation of the app, the terms of service have to be accepted and then the phone starts searching for the gateway on the network. Given the hardware installation was done correctly, the app finds the gateway and prompts for a QR-Code, which has to be scanned by the phone camera. The contents of this QR-Code include a security code and the MAC address of the gateway. The security code is of great importance regarding confidentiality in particular, and was used in sniffing the local network traffic; more on that in Section 4.2.2. Since this is the initial setup of the gateway, no lamps have been set up yet and the home screen of the app looks empty. To add a new lamp to the gateway, two methods are available: one is connecting to the gateway directly, the other is connecting a lamp to a remote, which in itself is connected to the gateway, see Figure 3.5. These different methods of pairing lamps to the gateway actually result in different behaviors of the lamps after setup. This differentiation is unintuitive, and as will be explained further, actually somewhat buggy.

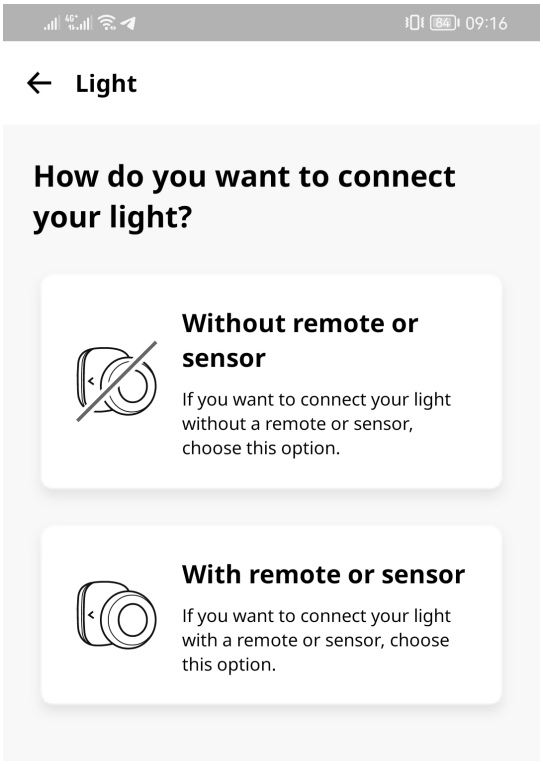

Figure 3.5: Lamp connection methods

On the first setup attempt, the round remote, seen in Figure 3.3, was used to connect the

lamps to the gateway without yet knowing the problems this would create later. After holding down the pairing button of the remote and holding it directly next to the desired lamp, the touchlink commissioning (Section 2.3.3.1) procedure started, and the lamp was automatically paired. This step was successful, but strangely enough, the new lamp did not show up in the main screen. One option that did, however, become available, was a *scene* with the name: 'All off'. A scene is a method for sending pre-defined settings of brightness and color to multiple devices at once. Controlling the newly added lamp was possible either via the remote, which worked fine, or via the app, but only via the aforementioned scene: 'All off'. This was confusing as only being able to switch off the lamp, but not being able to turn it back on, is not particularly useful, and no feedback for user input was visible within the app itself. A first, false assumption was, that lamps which were added via a remote were not phone-controllable, and only lamps added to the gateway directly were. This assumption was only found to be false after doing the setup process multiple times, and connecting lamps via a different remote, which surprisingly resulted in the lamps showing up correctly in the app. Thankfully, this made controlling the lights with the app possible as well. The cause of this bug seems to be related to the specific remote that was used first, as the second remote, one used in further testing, did not have any similar problems. This bug is seemingly linked to the creation of groups (Section 2.3.3.2). Under normal circumstances, new lights are always added to a group after successful setup, but in the case of the buggy remote, new lights were added in an ungrouped state. This state was the reason why no control, for the lights added through this particular remote, was available. Lights that were put in this special state were not easily recoverable by just changing its group, but had to actually be deleted from the list of connected devices, and re-added in a different way. Thankfully, the second remote worked fine, and all lamps that were set up via this remote were grouped correctly.

The second method for connecting new lamps is directly to the gateway. For the extent of this analysis, this method worked without problems, and never resulted in buggy lamp behavior. When selecting this method for adding new lamps, the corresponding physical light-switch of the intended light-bulb has to be turned off, and back on again for six times. This sequence activates a pairing mode, which makes the light-bulb blink slowly and softly. After a few seconds, the pairing process completes, and the new lamp is put into a group.

At this point, the setup is complete, and normal operation of the system can begin. The three lamps bought as part of the TRÅDFRI system support both dimming, and changing color temperature. The remotes, as well as the app, are able to adjust any of these values for any of the lamps. No additional problems were encountered after the setup process.

3.2 Network setup

To isolate the TRÅDFRI system from the home-network, a subnet with a dedicated router was set up. The subnet-router in question was a *FRITZ!Box 7430* which was somewhat complicatedly connected to the home-router, a *FRITZ!Box 5530 Fiber*, via the WI-FI repeater *TP-Link TL-WA850RE*. An overview of the network setup can be seen in Figure 3.6. The reason this specific network setup was chosen, was the ability of utilizing the guest-access feature on the home-router, which separates the main network into two subnets. This way, isolation from unwanted network clients, that are not involved in this analysis can be guaranteed, and the home network can still be used without any restrictions. The current *FRITZ!Box 5530 Fiber* does, however, not support guest-access via Ethernet, which was previously supported on older FRITZ!Box routers. This change means that any clients, that want to connect to this router via the guest access feature, have to use Wi-Fi. This was a big problem, because the subnet-router can only be connected to the main router via an Ethernet cable and not via WI-FI. There might have been other workarounds for isolating the network of this analysis from the home network, but the aforementioned solution was the easiest to set up, and deemed most trustworthy for separating network traffic securely. It should also be mentioned that FRITZ!Box routers are oriented more towards consumers than professionals, so features like VLAN, etc. are not supported natively.

With the WI-FI repeater in place, and connected to the home-router via WI-FI, its Ethernet port was used to connect the subnet-router to the home-router's guest-WI-FI. With this step completed, the subnet was ready to be used for analysis.

This newly created subnet was only accessible to the rooted smartphone, the TRÅDFRI gateway, the Kali machine, and at a later point, my personal phone as well. Restricting the number of active clients in a network is a key-step in minimizing background network activity for network sniffing.

For the man-in-the-middle attacks as well as for network sniffing, the Linux distribution Kali[2] was chosen. This is an operating system designed from the ground up to be the used for pentesting. It has several hundreds of tools preinstalled, that are needed for testing various aspects of security, ranging from reverse engineering, to password cracking and even to social engineering. Some famous preinstalled tools are John the Ripper, nmap, and Wireshark. While all the included tools can be installed separately on a Linux machine, and some, like the aforementioned examples, even on Windows, having the option of a ready-to-go system for all possible pentesting needs is a very welcome one. Kali Linux is developed by *Offensive Security*[3], a security research company itself, which makes it ever more apparent for which audience this operating system was designed for.

[2] https://www.kali.org/
[3] https://www.offensive-security.com/

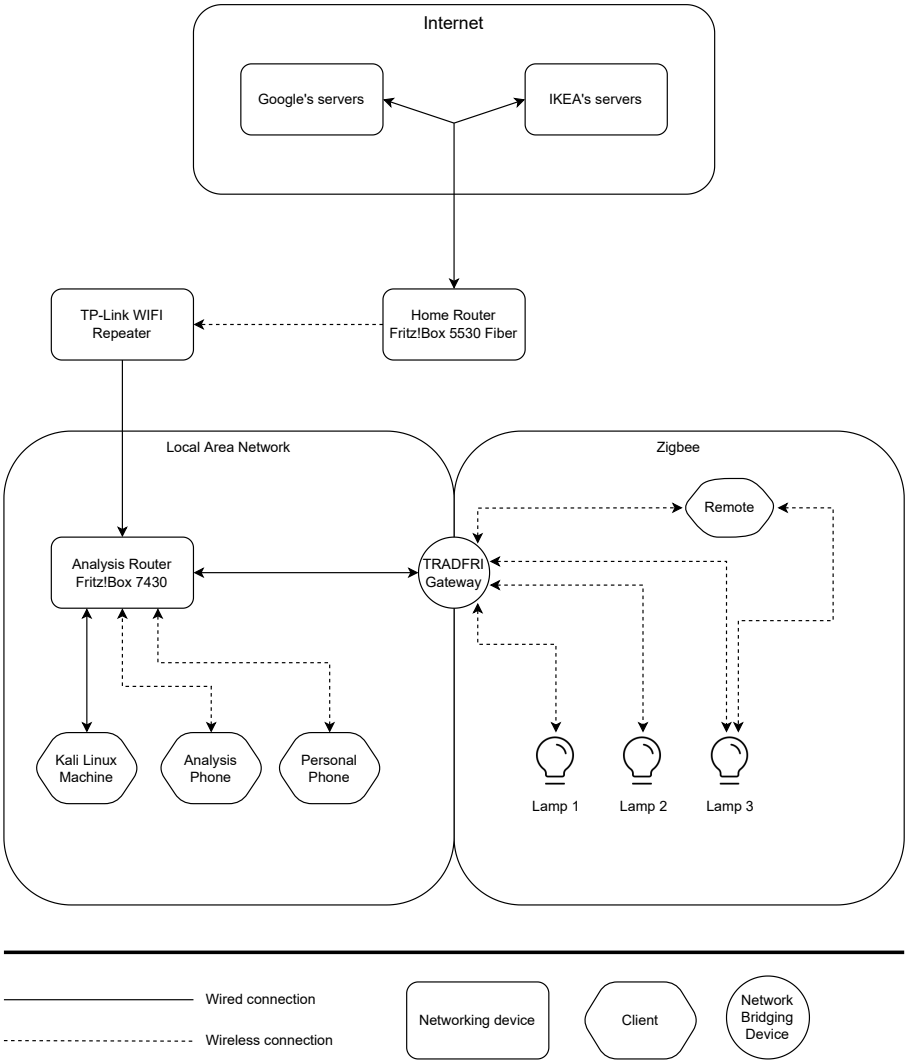

Figure 3.6: The local area network and the ZigBee network, connected with the TRÅDFRI gateway

3.3 Software

Wireshark[4] is an open source[5] network sniffing application, licensed under the GPL. It is available for all common operating systems, including various distributions of Linux, macOS, and Windows. At the time of publishing, the most current version of Wireshark, 3.6.7 was used. As part of this thesis, it was used to reverse engineer the connection between the phones and the TRÅDFRI bridge. Wireshark is not an application that is meant to be used for active attacks, such as package manipulation, which is made clear on the introductory page of the Wireshark documentation (Wireshark, 2022b). Nevertheless, Wireshark's capabilities should not be underestimated, as it can dissect 3000 different protocols, as well as decrypt traffic given the correct key, and is also able to interpret

[4] https://www.wireshark.org/
[5] https://www.wireshark.org/docs/wsug_html/#_open_source_software

unencrypted traffic without problems (Wireshark, 2022a). For many, Wireshark is the go-to application for analysis of network capture data, and the fact that it is open source makes it even more appealing.

mitmproxy[6] is an open source tool licensed under the MIT License also used for reverse-engineering network traffic, but very different to Wireshark. While Wireshark's purpose is to passively analyze any recorded traffic mitmproxy's purpose is to actively engage in it. What this means, in particular, is that network traffic, even encrypted TLS traffic, is intercepted with a man-in-the-middle (Section 2.4.4) attack. As the name implies, the way this is possible, is that the application acts as a proxy, and all traffic is relayed through the application before it can reach the internet. The decrypted network traffic is then recorded, and can be viewed most easily via mitmweb, the web server of mitmproxy. When accessing mitmweb's webpage, the captured, decrypted traffic can be viewed chronologically and all in-/outbound packages, including their content can be inspected, replayed, saved and much more. This works for all configured devices that have the IP of a running mitmproxy instance set as their proxy.

Luckily for public security, but unluckily for this analysis, it is not as easy as just setting up a proxy, and all traffic is decrypted. For mobile devices, the big hurdle is importing the self-signed mitmproxy root-certificate into the device's system-wide certificate storage. For all Android versions used as part of this analysis, this requires the phone to be rooted.

packetcapture[7] is a similar application to mitmproxy, as it serves the same role in executing a man-in-the-middle attack to access unencrypted traffic as well as decrypting encrypted traffic. The difference of mitmproxy compared to packetcapture is, for one, that packetcapture is an Android application, which means it can simply be downloaded from the Google Play Store and requires no use of the command line. The other difference is that a VPN is used for intercepting traffic instead of a proxy. Just like with mitmproxy, a self-signed root-certificate has to be imported into the root-certificate store of the Android system. Luckily, the same certificate used for mitmproxy can also be used for packetcapture if it is selected in the application's settings. When running, a fake VPN connection is established and all in-/outbound traffic of the device can be viewed inside the packetcapture app in real-time chronologically.

FRITZ!Box Packet Trace is not an application per se, but a hidden feature that is integrated into FRITZ!Box routers. What this feature enables, is capturing and saving all network traffic of any device connected to the router while the service is running. It simply captures the raw traffic, which can be encrypted, and does not perform any

[6] https://mitmproxy.org/
[7] https://play.google.com/store/apps/details?id=app.greyshirts.sslcapture

active interception to enable man-in-the-middle attacks or replay attacks, but just like with Wireshark, capturing any network traffic at all can play a central role in reverse engineering.

The feature is hidden in the sense that it cannot be found via the interface that is available at: http://fritz.box/, but only when accessing the specific site: http://fritz.box/html/capture.html. When logging in with the router's password, many options are available for capturing network traffic; these can be considered filters. Among others, each physical network port, different WI-FI options, as well as just the traffic reaching the internet can individually be captured. When stopping the capture, a file with the file type `.eth` can be downloaded, and then analyzed with an application such as Wireshark.

APK Extractor[8] is an Android application with the sole purpose of extracting the `.apk` installation files of any application that is present on the system. Although extracting installation files of applications that are already present on a phone might seem counter-intuitive, there are a variety of scenarios where this can be useful. These scenarios include preserving an application which is not officially available anymore, sharing applications with people that do not trust their internet connection, and most importantly in the scope of this thesis, having the installation file for an application allows the aforementioned file to be uploaded to an application testing framework.

jadx[9] is an open-source decompilation software, licensed under the Apache License 2.0, available for Linux, macOS, and Windows. It is designed specifically for decompiling Android applications, which results in their Java source code, the application was once written in. jadx accepts applications in either a `.dex`, or in a `.apk` file format. When the decompilation process is completed, the source code with all its classes, functions, etc. can be explored in the jadx user interface.

MobSF[10]**,** which is short for Mobile security Framework, is an open source application that is licensed under the GPL v3.0. The purpose of MobSF is to automatically analyze a specified application and finding out if any security vulnerabilities are present. Various types of applications can be analyzed with MobSF; these include Android's `.apk` and `.xapk` files, the `.ipa` file type, used in Apple's iOS operating system, as well as the `.appx` filetype, which is a *Microsoft Windows* specific application package.

When correctly installed and executed, MobSF exposes a web interface, on which both static and dynamic analysis can be performed. After uploading an application's file to the web interface, the static code gets decompiled and thoroughly tested. After about five

[8] https://play.google.com/store/apps/details?id=com.ext.ui
[9] https://github.com/skylot/jadx
[10] https://github.com/MobSF/Mobile-Security-Framework-MobSF

minutes, the results can be viewed, as well as exported as a PDF. The findings MobSF makes in the static analysis are also summarized in a score, that rates the analyzed app's overall security performance out of 100.

MobSF also features dynamic testing. For this, an Android emulator is required, on which MobSF automatically installs the provided app, and offers a suite of specific testing tools. These tools include disabling certificate pinning (Section 2.1.5), disabling root detection, and many others.

KeyStore Explorer[11] is an open source application, licensed under the GPL v3.0. The core functionality of KeyStore Explorer is its ability to open files that contain certificates, these are calles Keystores. During this analysis it was installed on Kali Linux, but it is available for all major operating systems.

[11] https://keystore-explorer.org/index.html

Chapter 4

Security analysis

The TRÅDFRI system was analyzed using both static, and dynamic analysis. The static analysis will first be discussed, with the dynamic analysis following behind.

4.1 Static analysis

For the static analysis of TRÅDFRI, the IKEA Home smart application was decompiled and evaluated. The only two sources for downloading the application are the Play Store[1] on Android, and the App Store[2] on iOS. These stores do, however, not allow downloading the applications in their original file form, but instead install them directly onto the phone. To be able to perform any static analysis, the installation file was needed in its `.apk` form, which can only be made available with third-party software. It should be said that Apple does provide a first party solution to extracting applications via iTunes, into `.ipa` files, but this will be disregarded, since no iOS device was available in the timespan of this analysis.

To get access to the aforementioned `.apk` file, the application APK Extractor was chosen, which extracted the IKEA Home smart application without any problems. The extracted application was later found to be signed by *Inter IKEA BV*, which signifies that no malicious modification was made by the APK Extractor application in the extraction process.

The first, unsuccessful method of statically analyzing the Home smart app was done with jadx. The decompilation(2.4.3.1) process worked fine, however, one problem quickly became apparent. The decompiled IKEA Home smart source code was obfuscated. Because of that, the meaning of classes and methods was not inherently obvious, as they were individually named some character a-z, followed, in some but not allcases, by a number 0-9. Even though obscurity on its own cannot provide security (Eckert, 2008), it does make the task of reverse engineering the application's code a lot more time-intensive. Since this was my first effort in decompilation with no prior experience in reading obfuscated code, a different method for static analysis was chosen.

[1] https://play.google.com/store/apps/details?id=com.ikea.tradfri.lighting&hl=de&gl=DE
[2] https://apps.apple.com/de/app/ikea-home-smart/id1195836071

More success was achieved with the Mobile Security Framework - MobSF, which in comparison to jadx not only supports static analysis, but dynamic analysis as well. For static analysis, MobSF was installed on the Kali Linux machine, but for the dynamic analysis, it was also installed on many machines as will be elaborated on in Section 4.2. Installing MobSF on many machines was a valuable decision in retrospect, as consistent results across multiple installations provided validity to these results. Results, which conflicted with previous results, where exactly the same `.apk` file was uploaded, could therefore be neglected.

The hosted version[3] was used as well, but produced an internal server error while analyzing the Home smart app, which invalidated the results. This hosted version was used again some weeks later, where no server errors were encountered, but the security score was inconsistent with all prior test results, which invalidated these results as well. The invalid scan can be viewed on the hosted MobSF page[4].

The scan results produced by the aforementioned Kali Linux Machine, using MobSF v3.5.2 Beta, were consistent with results on three different machines, and therefore assumed to be valid. This scan is what the following findings were based on.

The version of the IKEA Home smart app, that was analyzed, was v1.19.2, as this was the current-most version, available in the Google Play Store, at the time of writing. Its `.apk` is signed, and the signee is the organization(O) IKEA, the organizational unit(OU) Inter IKEA BV and the common name(CN) Lovisa Eriksson. Lovisa Eriksson is most likely an employee of IKEA, and a fitting LinkedIn profile[5], matching this description is publicly viewable. Knowledge of the person who is responsible for this certificate makes the person, as well as IKEA, vulnerable to social engineering attacks, which could be avoided by choosing an X.509v3 common name not related to any specific person.

The minimum required version of Android, as stated in the scan, is 8.0. This requirement is, however, not enforced, as determined in Section 3.1. The scan also gives an overall rating of the analyzed app, which is calculated by the number of vulnerabilities times the severity of each of them. The exact model that was used for calculating this score was neither available in the documentation of MobSF, nor specified in the report itself. It was, however, shown in a static analysis report of the Poczta Android app.(*Poczta - Android Static Analysis Report* 2020).

The IKEA Home smart app earned the Risk Rating *Medium Risk*, which translates to grade B in a rating where A is the best, and F is the worst. The app was also rated with a security score of 42/100, where 100 is the best, and 0 is the worst. IKEA should be accredited for the fact that no user/device trackers were detected, which reduces privacy

[3] https://mobsf.live/
[4] https://mobsf.live/static_analyzer/?name=Home_smart_base.apk&checksum=df5d9a2d118544a0a747a0101c2a5df2&type=apk
[5] https://se.linkedin.com/in/lovisa-eriksson-03b6854

concerns about this app. MobSF also detected that the app might be able to detect if it is executed on a rooted device, but even though this might be the case, the app was completely usable on the rooted Samsung smartphone without any restrictions. This feature was found in `x4/p.java`. The scorecard of the app analysis can be seen in Figure 4.1.

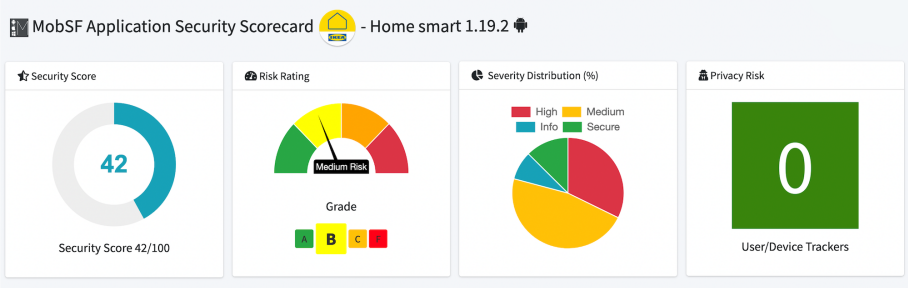

Figure 4.1: MobSF scorecard: IKEA Home smart v1.19.2

There were, however, some detected vulnerabilities, which have to be considered. Firstly, the app uses an insecure mode of encryption, specifically CBC encryption with PKCS5 padding, which is considered insecure, as described by (CWE, 2022). This insecure encryption is used in the file `a.java`, as well as `b.java`.

Secondly, there was an inconsistency with the pinning of certificates used by the app regarding external traffic, as there seemed to be two separate ways the app handled connections. One of those connections had correct certificate pinning in place, which was the connection to IKEA's cloud provider: *Google Firebase*[6]. Firebase provides a platform for building and releasing apps, but also for monitoring, messaging, and much more. The certificate pinning to Google Firebase was performed using the official Android KeyStore class (Google, 2022). This is a good practice in security, and makes it possible to detect as well as prevent man-in-the-middle attacks between the app and Google Firebase servers.

There is, however, a second way the app handles external connections, and this is where a vulnerability arises. In the file `z1/a.java` no certificate pinning is performed when establishing connections to external servers, which does, in fact, make man-in-the-middle attacks possible. All certificates, which are deemed secure by the operating system, are accepted, including self-signed certificates. This is how the attack in Section 4.2.1 was made possible. As recommended by (Walton et al., 2022), pinning should be implemented for every connection.

Even though certificates for the Google Firebase connection were implemented using the aforementioned *KeyStore* class, one Keystore file was found in the source code regardless. It was located it `res/Tbb.bks` and possible to read using *KeyStore Explorer*. Opening the Keystore prompted a password, but leaving the password empty and pressing 'OK' revealed the contents of this Keystore. These contents were two certificated named *sonos*

[6] https://firebase.google.com/

and *sonos2*, which are very likely used for the *SYMFONISK*[7] speakers, which are a product created in a collaboration of IKEA with Sonos[8] (IKEA, 2022a). These certificates are, however, not of any further interest, as no speakers were available for testing.

This concludes all major findings of the MobSF static analysis. There are findings, made by MobSF, which are not included in this thesis, as they were not seen as particularly relevant. These findings are, however, included in the full static analysis report 'MobSF-report_Home-smart_1-19-2.pdf' that can be found in the appendix, and on the website: http://bachelorappendix.kelke.de.

4.2 Dynamic analysis

Two different types of network traffic were monitored, one was local traffic, the other was external traffic. Local traffic refers to network traffic within the boundaries of the local area network. An example of local traffic is the router communicating with the TRÅDFRI bridge. External traffic refers to traffic that flows outside the local area network, into the internet. An example of this is the Home smart app receiving an update from an IKEA server. These two types of network traffic are illustrated in Figure 4.2

4.2.1 External traffic

Analyzing external traffic is an important step in the overall analysis of the TRÅDFRI system, as this traffic is the most vulnerable to being attacked by hackers. Since IKEA's servers could possibly be anywhere on earth, a secure environment for data to flow from the app, or gateway, to these servers cannot be expected. In order to protect the confidentiality and integrity of sensitive data, encryption is a necessity for external traffic. Thankfully, all the recorded internet traffic originating from the Home smart app was encrypted with the secure, and current, TLSv1.3 (Section **??**) standard as described by (Rescorla, 2018). This is a good start, but the following section will describe the findings that were made while using a man-in-the-middle attack. Two applications were used to execute this attack, of which only one was able to generate meaningful results.

The first application that was used, was mitmproxy. Setting it up was easy on Kali Linux, but required some work on Android. Setting mitmproxy up on Kali was as simple as executing the *mitmweb* command in the terminal, at which point a browser window opened, where all individual intercepted network packets were viewable. Setting up the man-in-the-middle attack on Android required two major steps.

The first step was setting the Kali Linux IP address, and the correct port as the network proxy in the WI-FI settings. In the case of this analysis the Kali Linux IP address was:

[7] https://www.ikea.com/ch/en/p/symfonisk-tradfri-gateway-kit-sound-black-white-s59318983/
[8] https://www.sonos.com/en-us/home

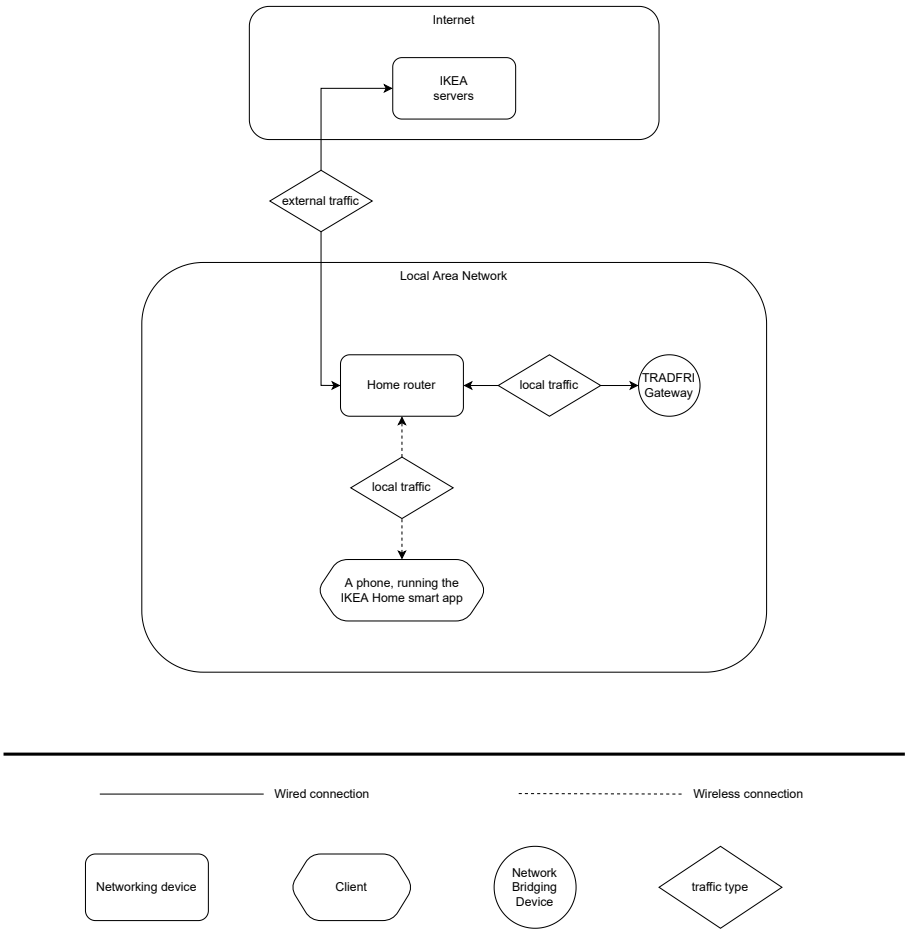

Figure 4.2: Differences in local and external network traffic

'192.168.11.21' and the port: '8080'. The second step was downloading, and importing the *mitmproxy* certificate into the device's root certificate store, which, in itself, required some additional steps. Certificates which are stored in this location are automatically fully trusted by the system. If this step is not performed, the device will never establish any TLS-secured connection while using the proxy *mitmproxy*. Downloading this certificate was possible by doing the aforementioned first step, and then accessing `mitm.it` with a web browser. Importing the downloaded certificate correctly is where some effort, as well as root privileges, were required. Firstly, the certificate had to be renamed, but the new name had to be generated back on the Kali Linux machine, with the following command:

```
1 openssl x509 -inform PEM -subject_hash_old -in certificate.pem | head -n -1
```

where the aforementioned **certificate.pem** refers to the certificate generated by *mitmproxy*. The output of this command was **c8750f0d**, which meant the downloaded certificate on the Android phone had to be renamed to **c8750f0d.0**. The **.0** extension is common for certificates, and required for Android to recognize it as such. This renamed certificate could then be copied into the folder: '**system/etc/security/cacerts**', which is the step where root access was needed for. With this step completed, the *Samsung Galaxy*

S6 needed to be rebooted, which marked the final step of the setup process. *mitmproxy* was now ready to be used (gcaillet, 2020).

While testing *mitmproxy's* functionality on various web-pages and installed apps, a major problem quickly became apparent. *mitmproxy*, is not able to strictly differentiate between the different apps that are being recorded. Since only the IKEA Home smart app is relevant for this thesis, filtering out any other, irrelevant traffic is an important feature. An app that did provide this feature was *packetcapture* (Section 3.3). Since this app runs directly on the phone itself, it is able to control network traffic before it leaves the phone. This enables a feature of selectively running a man-in-the-middle attack against only a specified app. To decrypt TLS traffic, *packetcapture*, just like *mitmproxy*, requires a root-certificate to be installed in the system-wide certificate store. Luckily, the already installed certificate, generated by *mitmproxy*, was possible to be imported in the *packetcapture* settings, which meant the *mitmproxy* certificate could be reused, simplifying the setup process. Unluckily, the results of this app were not usable, as no successful connections were recorded during the attempted man-in-the-middle attack. A user error in the setup process of the *packetcapture* app cannot be ruled out, but since all attempts in fixing this problem were unsuccessful, it was determined that *mitmproxy* would suffice.

Since no automatic feature for filtering out irrelevant traffic was available, the origin of recorded network traffic was manually determined. A major key in this process of filtering was the HTTP request header: `x-requested-with`, in some instances written as `X-Requested-With`. Not all, but most of the external network traffic produced by the Home smart app had the value `com.ikea.tradfri.lighting` written in the aforementioned header, which meant the origin of this traffic could clearly be assigned. By filtering for the above-mentioned HTTP header, and also by searching for any network activity containing the string '`ikea`', the following traffic was found to be originating from the Home smart app:

The first clearly identifiable network activity, was a request, the app made right after it was started, to Firebase servers, with the URL:

```
1 https://firebaseinstallations.googleapis.com/v1/projects/
    ihspushnotificationpocinstallations
```

This specific request is of relevance because of the response it created. The HTTP response code was not a regular `200`, which would signify a successful request, but `403`, which signifies an error due to a 'forbidden' request. The reason for this error was given in the details of the response, and reads as follows:

```
1 ...
2 "reason": "API_KEY_ANDROID_APP_BLOCKED"
3 ...
4 "message": "Requests from this Android client application com.ikea.tradfri.
    lighting are blocked.",
5 "status": "PERMISSION_DENIED"
```

The same request was recorded multiple times, but the response was the same every time. This finding coincides with the assumption made in Section 4.1, stating that certificate pinning was implemented, but not on all connections. Two different servers were also refusing connections from the Home smart app while *mitmproxy* was running; these servers were `meta.config.homesmart.ikea.net`, and `app.config.homesmart.ikea.net`. Network packages with this URL were not visible in the main window of mitmproxy, only in the console output, where the following message appeared:

```
1 192.168.11.30:50363: server connect app.config.homesmart.ikea.net:443
    (18.155.153.9:443)
2 192.168.11.30:50362: server connect meta.config.homesmart.ikea.net:443
    (52.222.191.45:443)
3 ...
4 192.168.11.30:50362: server disconnect meta.config.homesmart.ikea.net:443
    (52.222.191.45:443)
5 192.168.11.30:50363: server disconnect app.config.homesmart.ikea.net:443
    (18.155.153.9:443)
```

Knowing the existence of these connections is valuable information, but only speculation can be made about their purpose. One speculative explanation for these connections is that `app.config.homesmart.ikea.net` is used for tracking app usage, and `meta.config.homesmart.ikea.net` is used for providing updates to TRÅDFRI devices. Support for the speculative explanation, about `meta.config.homesmart.ikea.net`, is provided by the fact that the aforementioned message 'server connect meta.config.homesmart.ikea.net:443' appeared every time the button for updating TRÅDFRI devices was pushed in the Home smart app.

The servers that did accept connections from the IKEA Home smart app on the rooted Android device had the following schemes and domain names:

```
1  https://ww8.ikea.com/
2  https://www.ikea.com/
3  https://rec.ingka.com/
4  https://polyfill.ikea.net/
5  https://cdn.cookielaw.org/
6  https://cdn.optimizely.com/
7  https://errors.client.optimizely.com/
8  https://geolocation.onetrust.com/
9  https://privacyportal-eu.onetrust.com/
10 https://cloud80.realperson.de/
11 https://p11.techlab-cdn.com/
12 https://android.apis.google.com/
```

With the exception of requests directed to the google.com domain in line 12, a similarity was found in all requests containing the other eleven domain names. This similarity was the `user agent` header, which had the following value in all requests:

```
Mozilla/5.0 (Linux; Android 7.0; SM-G920F Build/NRD90M; wv) AppleWebKit
  /537.36 (KHTML, like Gecko) Version/4.0 Chrome/98.0.4758.101 Mobile Safari
  /537.36
```

A possible reason for this header is the usage of an internal web-browser that is used for handling HTTPS traffic and displaying that fetched content. On Android, this is called a WebView, and commonly used for tasks like displaying ads inside apps. Usage of a WebView was also found in the APK source code, in the files: `a8/r2.java`, `a8/a.java`, `y7/i5.java`, `y7/c2.java`, `y7/j5.java`, `y7/d2.java`, `wa/a.java`, `wa/y.java`. Most of the content of these requests was HTML, CSS and JavaScript Code, as well as images of unknown file types. HyperText Markup Language (HTML) forms the structure of a webpage, and simple webpages can be created using only HTML. Cascading Style Sheets (CSS) is a language for styling webpages and allows, among other functions, elements on a webpage to be precisely positioned, colored and animated. JavaScript is a programming language webpages use to be more interactive, and allows, for example, content to only be loaded when a user scrolls to the bottom of a webpage (Mozilla, 2022b). When opening a webpage with a browser, the webpage's JavaScript code is run in the browser of the client, which is why it can potentially be dangerous. The aforementioned network traffic, containing all HTML, CSS, and JavaScript code was generated while starting the Home smart app for the first time, at which point the terms and conditions as well as the privacy policy needed to be accepted. Since both agreements were captured in *mitmproxy*, it is safe to assume this internal web browser does not use pinned certificates. The terms and conditions were captured with the URL:

```
https://ww8.ikea.com/ikeahomesmart/tnc/android_tandc_de_de.html
```

and the privacy policy with the URL:

```
https://ww8.ikea.com/ikeahomesmart/privacy/privacy_policy_de_de.html
```

These URLs can in fact be accessed with a regular web-browser to reveal the respective agreements. These agreements are non-sensitive data, which is possibly why IKEA decided not to enforce certificate pinning. However, a threat to be considered is that JavaScript is used, and if a user is the victim of a man-in-the-middle attack, this JavaScript code can be intercepted and maliciously modified without the user noticing.

One of the above-mentioned domains 'geolocation.onetrust.com' was used to determine the user's location. A request was sent to the URL:

```
https://geolocation.onetrust.com/cookieconsentpub/v1/geo/location
```

which response contained the following JSON data:

```
{
  "city": "Bremen",
  "continent": "EU",
```

```
    "country": "DE",
    "latitude": "53.08930",
    "longitude": "8.77450",
    "state": "HB",
    "stateName": "Bremen",
    "timezone": "Europe/Berlin",
    "zipcode": "28217"
}
```

Surprisingly, this is fairly accurate. For which reason this location was requested is unclear.

Another particularly interesting connection was captured by *mitmproxy*, but not found to be originating from the IKEA Home smart app. Connections with the URL:

```
https://sessions.bugsnag.com/
```

were established in some cases where an app was closed, the phone was unlocked, and at some seemingly random interval every couple of minutes. Bugsnag[9] is an error monitoring software, which does not raise any particular concerns, but the data that was transmitted, had an interesting detail.

```
...
  "device": {
      "cpuAbi": [
          "arm64-v8a",
          "armeabi-v7a",
          "armeabi"
      ],
      "jailbroken": true,
      "manufacturer": "samsung",
      "model": "SM-G920F",
      "osName": "android",
      "osVersion": "7.0",
      "runtimeVersions": {
          "androidApiLevel": 24,
          "osBuild": "NRD90M.G920FXXU6ERF5"
      }
  },
  "notifier": {
      "name": "Android Bugsnag Notifier",
      "url": "https://bugsnag.com",
      "version": "4.22.3"
  }
  ...
```

Firstly, it is interesting that the full detail of hardware specifications is sent every time this connection is created. Secondly, in line 8 the phone reports itself as being jailbroken. Jailbreaking, on iOS, is the equivalent to rooting on Android, which means the 'Android

[9] https://www.bugsnag.com/

Bugsnag Notifier' must have root detection capabilities. It can neither be proven, nor disproven, if IKEA has access to this information.

The file 'mitmproxy-flows', that *mitmproxy* created during this analysis can be opened by any instance of *mitmproxy*, and is available in the appendix, as well as downloadable at the following website: http://bachelorappendix.kelke.de.

After using *mitmproxy* while setting up and using the Home smart app, it became apparent that an active man-in-the-middle attack did not affect the app's usability. However, internet connectivity in general seemed to only be required for reading the terms and conditions, and when performing updates. When controlling lights, or other ZigBee devices, the app, and the gateway only needed a local network connection. Being able to control a smart home system within the boundaries of a local network is not specific to TRÅDFRI, as brands, like Signify[10] with its *WiZ*[11] system, also support this feature. Where these two systems differ, is in the ability of being controlled remotely. The WiZ system does allow local control, but also intends users to use their remote service. This remote service gives users full control over all their smart home devices from anywhere in the world with an internet connection. TRÅDFRI does not have such a service, which could be seen as a drawback by some people, but a smart home system that completely decouples its functionality from access to the internet also gives hackers a considerably smaller attack surface, as intercepting a remote connection, or maliciously creating one is not possible. In Figure 4.3, the two possible ways of interacting with smart home devices can be seen. The green path signifies a connection inside a LAN, whereas the red path signifies a remote connection via the internet.

[10] https://www.signify.com/en-us
[11] https://www.wizconnected.com/en-us

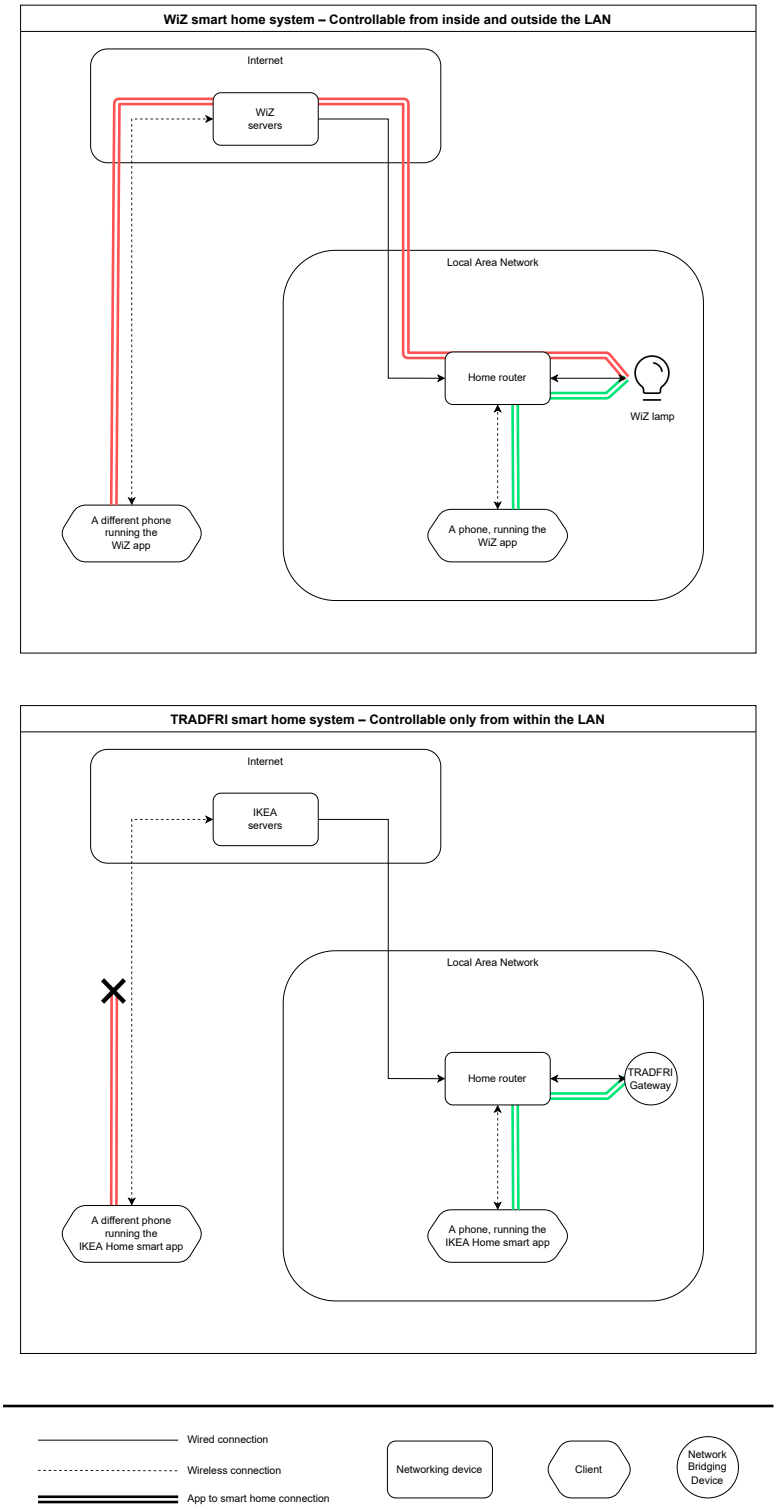

Figure 4.3: Different ways smart home systems can be controlled

IKEA's focus on local networks for the TRÅDFRI system does, however, not change the fact that this analysis clearly found network traffic with external servers. In their privacy policy, IKEA is open about the data they collect from users of the Home smart app, and

also states: "Your personal data is safe with us: we do not sell or trade your personal data to other third parties." (IKEA, 2022b). If this is true, IKEA should be praised for choosing not to follow the trend of profiting of user data, especially since devices like these found in the TRÅDFRI system are very much capable of generating valuable data (Bojanowska, 2019). But while the privacy policy in general does give the impression that IKEA cares about their user's privacy, it is outside the scope of this analysis to evaluate this policy's legal details. Accessing this policy is currently not possible using a search-engine, and even if a URL containing this policy was found during the course of this analysis, customers have no way of finding this URL on their own, and as a result of that, are confronted with both the terms and service, as well as the privacy policy only after they possibly already bought TRÅDFRI devices, or at least, already downloaded the app.

Since MobSF not only featured static analysis, but dynamic analysis as well, many attempts were made in setting up this feature. Three problems made all of them unsuccessful. MobSF uses an emulated Android system, which means a special application, like Genymotion[12], runs the Android operating system on a desktop computer. This is a performance heavy process, which was the first problem, as no already owned computer was powerful enough to meet the hardware requirements for a sufficiently high version of Android. The second problem was network related. *Amazon Web Sevices* offers a service of running Android emulation with Genymotion on their servers, which have more powerful, more current hardware. The problem with this service, however, was the distance to the TRÅDFRI gateway. Since Amazon's servers run in an entirely different network, in the city of Frankfurt, they were not able to access the gateway. The third problem was MobSF's lack of support for physical Android phones. The two above-mentioned problems would have been neglectable, if MobSF supported using a physical phone instead of emulators only. In the MobSF documentation (MobSF, 2022) usage of physical devices is considered possible, but not supported. This is why an attempt of using the rooted *Samsung* phone to connect to MobSF was made, which was also unsuccessful.

4.2.2 Local traffic

Local network traffic needs to meet different security requirements than external traffic. Participants in a local area network have to either be physically connected with an ethernet cable, or connected to WI-FI, which in most cases is protected with a password. This greatly limits an attacker's possibilities, as getting access to a victim's local area network is difficult. However, the decreased possibility of an attack does not completely rule out an attack. A LAN can get compromised if, for example, one of its members accidentally downloads malware, which is able to record network traffic. Malware-infected computers are not uncommon, which is one of the reasons LAN traffic needs to be secured as well. To

[12] https://www.genymotion.com/

evaluate the security measures IKEA implemented into the local traffic of the TRÅDFRI smart home system, two types of attacks were tested.

The first choice was network sniffing. *Wireshark* was the obvious choice for this task, as recording network traffic is its primary purpose. Using *Wireshark* on the Kali Linux machine, however, was a wrong approach. Since the Android phone running the Home smart app, and the gateway simply communicated with each other, only minimal network traffic reached the Kali Linux machine. Traffic that did reach this machine were so-called 'broadcast' messages, directed at all members of the LAN, which, of course, only gave little insight into the actual communication procedure of the app and the gateway.

The network sniffing needed to take place on the router itself. Since all traffic directed at any member of the LAN was unavoidably routed through the router, capturing traffic at this point proved to be the most effective. In the case of this analysis, the router was a *FRITZ!Box 7430*, which does have a network sniffing feature called: *Packet Trace* (Section 3.3). This feature is hidden from the default user interface at `http://fritz.box`, and can only be accessed when directly opening the URL: `http://fritz.box/html/capture.html`, which reveals an interface for recording various parts of the local traffic individually. This interface can be seen in Figure 4.4

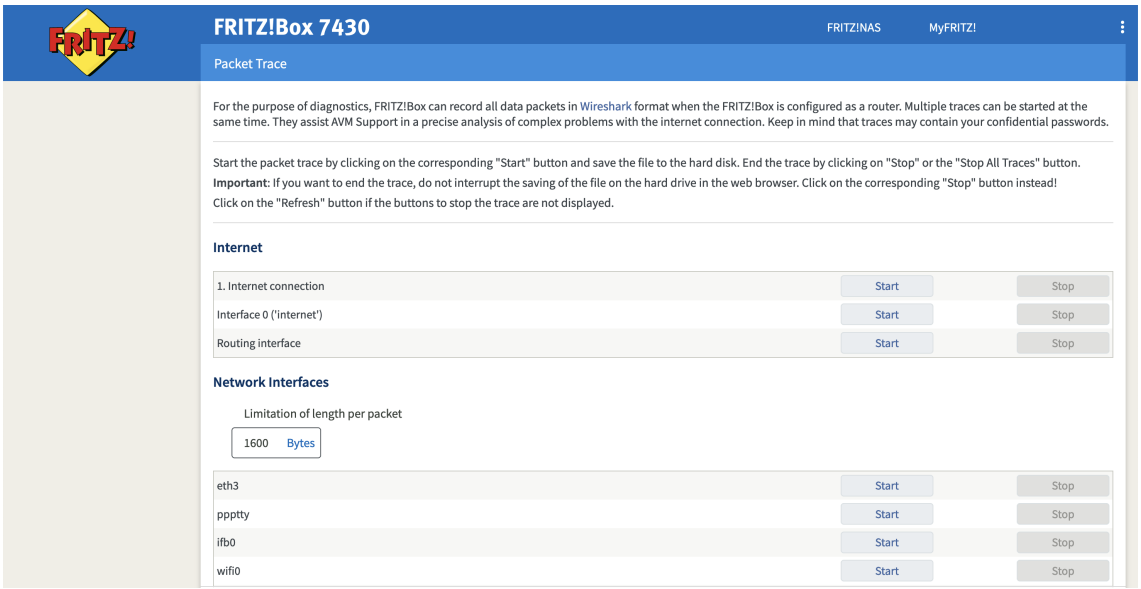

Figure 4.4: The interface for capturing network traffic on a FRITZ!Box

After starting the capture process, an `.eth` file was downloadable, the download of which stopped after stopping the capture process. For sniffing the TRÅDFRI network traffic, the following three selections were chosen:

- Interface 0 ('internet')
- eth2
- ath0

The `Interface 0` only includes network traffic with communication partners on the internet. The `eth2` selector only includes traffic flowing on the Ethernet port that is labeled 'LAN 3' on the FRITZ!Box. The physical ports 'LAN 1' up to 'LAN 4' on the FRITZ!Box equal the 'eth0' up to 'eth3' selectors on the *Packet Trace* website. 'ath0' was selected, as it contained all WI-FI traffic.

During six recording sessions, both phones were used to perform various activities, such as setting up the Home smart app, adding lamps, and controlling these lamps. The `.eth` files produced during these six recording sessions were analyzed using *Wireshark*, and form the basis of any further findings.

To analyze how the TRÅDFRI gateway operates when it is first powered on, it was reset. Resetting the gateway can be done either via the app, or by holding down the 'Reset' button of the gateway for about five seconds. After a reset, the gateway joined the LAN, which was done by performing a DHCP request to receive an IPv4 address, and an ICMPv6 solicitation to receive an IPv6 address.

After successfully obtaining valid IP addresses, the Network Time Protocol (NTP) Version 3 was used to synchronize the gateway's time with one of the two following servers:

```
1.tradfri.pool.net
3.tradfri.pool.net
```

This process was very similar to the process described by Bruns. One difference, however, was found in the way that the gateway checks for new firmware-updates. In the experiments carried out by Bruns in 2018, an unencrypted HTTP request was sent, to which the response contained update information. Since then, it seems that IKEA has changed the update checking process; in the recordings made in this analysis, a TLSv1.2 connection was established for contacting 'fw.ota.homesmart.ikea.net'. The request itself might possibly be the same, but the addition of TLS encryption ensures a safe transmission of firmware files. The file: `http://fw.ota.homesmart.ikea.net/feed/version_info.json` found by Bruns, is however, still accessible and up to date with the current firmware versions. The firmware linked in this file also still seems to be signed, as all referenced filenames end with the suffix `.signed`. See the following example:

```
    "fw_binary_url": "http://fw.ota.homesmart.ikea.net/global/GW1.0/01.19.032/
   bin/10032198-2.2-TRADFRI-gateway-1.19.32.p.elf.sig.ota.signed"
```

The firmware checking procedure was followed up by a TCPv1.2 connection to:

```
a1nvlh0fc0asuq.iot.eu-central-1.amazonaws.com
```

The domain name of this connection reveals that IKEA, in addition to Google Firebase, also uses services provided by *Amazon Web Services*[13], which is a cloud service provider as well. The aforementioned TLS connection, as well as the TLS connection to

[13] https://aws.amazon.com/

'fw.ota.homesmart.ikea.net' were not possible to decrypt. This is due to the nature of the FRITZ!Box's *Packet Trace* tool, which does not implement any man-in-the-middle features. Therefore, the content of these two connections was not readable, and certificate pinning was also not testable.

After these connections were completed, the *Huawei P30 Pro* was able to connect to the gateway. This connection required both the MAC address, and the security code of the TRÅDFRI gateway. As described in Section 3.1, these two values were encoded in the QR Code that can be seen on the back of the gateway; see Figure 4.5.

Figure 4.5: The security address, the MAC address, as well as the QR-Code can be found on the back of the TRÅDFRI gateway

During the setup process of the Home smart app, the aforementioned QR Code was scanned by the app using the phone's camera. This QR Code enabled the app to use the encoded MAC address in a mDNS query for finding the gateway. mDNS, or multicast Domain Name Service, is a protocol designed for discovering participants in smaller networks, which the LAN used in this analysis can be considered. As the gateway's MAC address was **e8-e8-b7-b9-a8-01**, the resulting mDNS query was for **TRADFRI-Gateway-e8e8b7b9a801.local**. After a successful query, the phone established a DTLSv1.2 (Section 2.3.7) connection to the gateway, which was encrypted using the security code **xVTUnHarGkMbtf94** as a symmetric key. Within this DTLS secured connection, the protocol CoAP (Section 2.3.4) was used for sending and receiving JSON-formatted objects. These JSON-formatted objects contain the commands and information, which are used to switch lights, display the current state of light bulbs, etc. DTLSv1.2 is a secure protocol, but has since been obsoleted by DTLSv1.3.

IKEA, however, decided not to use the aforementioned security code indefinitely, but instead for the single purpose of exchanging a session key in an encrypted manner. This session key is unique, and valid for the duration of a single installation of the Home smart app, which means that a different key is used for every smartphone, and every time the

app is reset. Using a session key for each device that connects to the gateway is more secure than only using one single key for all connections, but the implementation is flawed. Since the transfer of any individual key is always encrypted with the permanent security code, neither forward secrecy (Section 2.1.5) nor backward secrecy can be provided.

For the aforementioned reason, it was possible to decrypt all network traffic flowing between the gateway and its connected apps. Doing this involves the following steps: Firstly, a network recording of a phone, performing the setup process of the Home smart app, needs to be opened in *Wireshark*. One of the recordings that was captured using *Packet Trace* is available as the file **local-traffic_packet-trace.eth** both in the appedix, and downloadable at the following website:
http://bachelorappendix.kelke.de
In this `.eth` file, the IP address of the phone is '192.168.11.31', the IP address of the gateway is '192.168.20.40' and the IP address of the router is '192.168.11.1'

Secondly, the 16 character security code on the back of the gateway has to be encoded into a hexadecimal format, which can, for example, be done using an online converter[14].

Since the security code was **xVTUnHarGkMbtf94** in text-form, the resulting hexadecimal output was **785654556e486172476b4d6274663934**. This, hexadecimal, security code then has to be set as the DTLS Pre-Shared Key in *Wireshark*. The menu for setting the aforementioned Pre-Shared Key can be found under 'Edit → Preferences → Protocols → DTLS'. A screenshot of this menu can be seen in Figure 4.6.

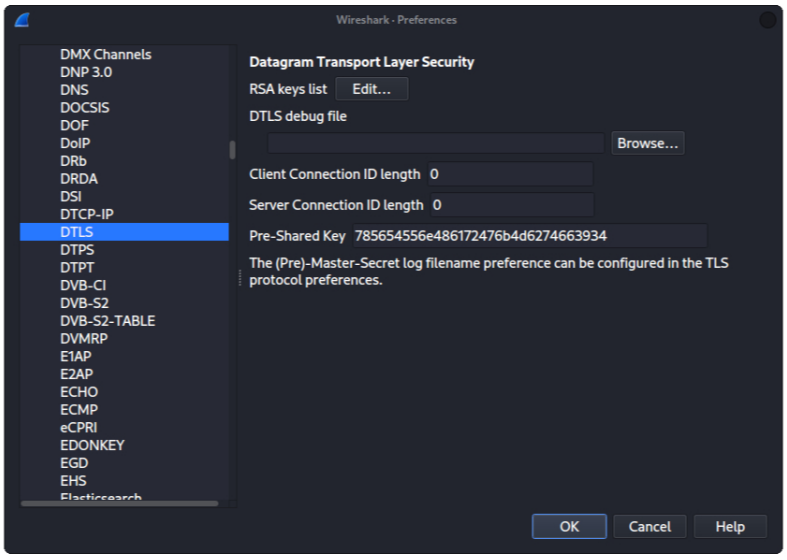

Figure 4.6: Wireshark menu for setting the gateway's security code as the key for decrypting DTLS network traffic

[14] https://codebeautify.org/string-hex-converter

After clicking 'OK', all DTLS traffic that includes prior and future session keys can be decrypted. The transfer of three session keys can be found in the provided `.eth` file by typing 'coap' into the 'display filer' filter field right under the menu bar. Six network packets appear, of which the 'CON' packets, numbered 1006, 2042 and 2787, each contain the request for a session key by the *Huawei* phone to the gateway. The 'ACK' packets, numbered '1007, 2043 and 2788, contain the actual session keys sent by the gateway to the phone. The content of these packets can be viewed by selecting a packet and clicking 'Decrypted DTLS' on the very bottom of *Wireshark*. This reveals the JSON formatted data, which reads as follows:

```
# Session-key request, packet number 1006:
{"9090":"90e2b1607c34427f85e19e445b05e8a4"}

# Session-key response, packet number 1006:
{"9091":"LKYYRuZO55HSSpWC","9029":"1.19.0032"}
```

The value of 9090 identifies the current client at the gateway, and can be considered a username. The value of 9091 contains the session key, encoded in 16 characters. The value of 9029 specifies the firmware version, the gateway is currently using. (Haan, 2022)

Since all session keys can now be read, the above process is applicable for any of the DTLS connections they encrypt. This reveals the structure of the commands sent to the bridge, which follow a similar structure of data as the packets above, and includes ZigBee group names, domain names mentioned in this Section, as well as obfuscated command names.

The steps for decrypting local network traffic are still the same as they were in the analysis made by Bruns, with the only exception being that the bug, requiring a 'RSA private key' for *Wireshark*, seems to be fixed, as no work around is required anymore.

This concludes the dynamic analysis of the TRÅDFRI smart home system. A possible solution to the aforementioned problem of unsafe key exchange can be found in Chapter 5.

Chapter 5

Results

This chapter will give an overview of the security related findings that were made while analyzing the TRÅDFRI smart home system.

While statically analyzing the IKEA Home smart app, it was very quickly discovered that its source code was obfuscated. This greatly limited the depth of information that was possible to reverse engineer manually. Obfuscation of security related information is not considered a good practice, but understandable, since it protects IKEA's intellectual property.

Thankfully, the Mobile Security Framework (MobSF) was able to perform an automatic static analysis, which revealed two vulnerabilities. The first being an unsafe type of encryption. The app uses the encryption mode AES/CBC with PKCS5Padding. This allows inputs to be modified, without the app noticing, and is called the *padding oracle attack* (CWE, 2022).

The second vulnerability that MobSF revealed, was that IKEA does perform certificate pinning, however, not on all network connections. The connection that was using certificate pinning, was found to be the connection to Google Firebase, where pinning was implemented using the official Android *Keystore* class. This is the preferred method of storing certificates, over developing an own solution. The connections made without certificate pinning were not possible to extract from the MobSF report, which is why a man-in-the-middle attack was used for discovering them.

Another finding was the common name (CN) under which the Home smart app was signed. The name, Lovisa Eriksson, coincides with the LinkedIn profile[1] of a current employee of IKEA, which makes this person vulnerable to social engineering attacks. Choosing a common name unaffiliated with any individual would be preffered.

The KeyStore `res/Tbb.bks` was also discovered, and included two certificates: 'sonos' and 'sonos2'. These are likely going to be used for IKEA's line of speakers, which are called *SYMFONISK*, as they were produced in a collaboration with the company Sonos (IKEA, 2022a).

[1] https://se.linkedin.com/in/lovisa-eriksson-03b6854

MobSF's dynamic analysis feature was also tried, but to no avail. Legacy hardware turned out to be an issue so great, that no workarounds were found for starting it. Connecting MobSF to the Genymotion offering[2] of Android emulation on *Amazon Web Services* worked, but no connection to the TRÅDFRI gateway was possible.

Using *mitmproxy* to perform a man-in-the-middle attack on a smartphone, various HTTPS connections were discovered that did not implement certificate pinning. The connection to 'ww8.ikea.com' was found to include the legal agreements of the Home smart app. The privacy policy was one part of these agreements and contained no policies of concern. The assumption made in the static analysis of connections to Google Firebase servers implementing certificate pinning was also supported by the man-in-the-middle attack, as these connections were refused.

The three tools that produced meaningful results in the context of dynamic analysis ended up being *mitmproxy*, the FRITZ!Box feature: *Packet-Trace*, and *Wireshark*. Using *mitmproxy*, the assumption, made in the static analysis section, of an irregularity in the certificate pinning process, was further supported. The connection from the Home smart app to Google's Firebase servers at 'firebaseinstallations.googleapis.com' was securely implemented and not possible to attack. For this reason, it was not possible to find out which Firebase services exactly IKEA uses, and what data is usually transmitted. All requests made by the app, to Firebase servers, resulted in the HTTP error code 403 and the following response: 'API_KEY_ANDROID_APP_BLOCKED'. Two different connections from the rooted phone were also found to be rejected; these were: 'app.config.homesmart.ikea.net' and meta.config.homesmart.ikea.net. *mitmproxy* was also able to intercept traffic by the Home smart app's embedded browser. This traffic included the terms and conditions, as well as the privacy policy. Modifying these agreements is thereby possible, but even though modifying a legal agreement's text can hardly be considered a threat to an app's security, the JavaScript code that is sent additionally to these agreements does provide an attack surface. To provide a maximum level of security, implementing certificate pinning on these connections would be preferred.

The most problematic vulnerability was discovered while sniffing local network traffic with *Packet Trace* and analyzing this traffic with *Wireshark*. The vulnerability lies in the implementation of a permanent security code, which is printed on the back of all TRÅDFRI gateways, as this security code is used as the encryption key for exchanging session keys. Since only this single code is used over the entire gateway's lifetime, an attacker with access to this code, and the local area network, can decrypt the transmission of all session keys. Using this technique, it was possible to read all the encrypted traffic of both phones, even after multiple resets. If the TRÅDFRI system were to be used in a commercial environment, getting access to the network and QR Code might very well be possible, and since the code is permanent, buying the TRÅDFRI system second

[2] https://aws.amazon.com/marketplace/pp/prodview-tz54qmys53oyo

hand can also be problematic. One possible solution to this problem is the Diffie-Hellman key exchange (Diffie and Hellman, 1976). This method for exchanging keys allows two partners to exchange a session key without the session key itself being transmitted. The gateway's security code could act as the common starting point that is required for the aforementioned key exchange to work. This revised key exchange would provide forward secrecy, as well as backward secrecy, as an attacker with access to the security code would neither be able to decrypt future, nor past transmission of session keys. Changing the key exchange procedure would even be possible without any hardware changes, and could possibly be rolled out during a software update for the Home smart app, and a firmware update for the gateway.

Chapter 6

Summary & Outlook

The security analysis performed during this thesis gives an updated look at the state of security of the TRÅDFRI smart home system. Both static and dynamic analyses have previously been performed, but the security analysis in this thesis put a special emphasis on evaluating the implementation of certificates for TLS traffic, and the key exchange of DTLS traffic. In both of these connection protocols, problems were found, and possible solutions were proposed. It seems that IKEA considered various aspects of security when developing its smart home system, with the inability of controlling smart home devices outside a local area network playing a vital role in reducing the attack surface for hackers. Some vulnerabilities, however, need to be addressed in future TRÅDFRI revisions, as to provide users with a smart home system that is as secure as possible.

Two aspects of security were not analyzed in the scope of this thesis, these are hardware level vulnerabilities, and the implementation of ZigBee. The hardware level security of the TRÅDFRI was not inspected for the reason that the focus of this analysis was mostly network traffic. The ZigBee implementation was not analyzed, since it has already been found to be insecure, and commissioning procedures have not been changed since (Wodtke, 2021).

Further research into the TRÅDFRI system is still to be made, as results from the MobSF dynamic analysis are missing, and would be of great value. This would possibly also provide access to the content of network connections that were either not established, or not possible to decrypt. Finding a way to perform a man-in-the-middle attack on the TRÅDFRI gateway would also be interesting, as connections to external servers were discovered, but also not possible to decrypt. Collecting the data sent in the aforementioned connections would also allow a comparison of the data IKEA states on collecting in their privacy policy, with the data that is exchanged in the real world.

Appendix A

The enclosed USB-Stick contains all appendices.

1. The thesis in digital form
2. Recorded local areal network traffic
3. mitmproxy capture files
4. MobSF static analysis report

All appendices can also be found online at:
http://bachelorappendix.kelke.de

List of Figures

Literature

Bassett, L. (2015). *Introduction to Javascript Object Notation: A to-the-Point Guide to JSON*. First edition. Beijing ; Sebastopol, CA: O'Reilly. ISBN: 978-1-4919-2948-3.

Boeyen, S., S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper (2008). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Request for Comments RFC 5280. Internet Engineering Task Force. DOI: 10.17487/RFC5280. URL: https://datatracker.ietf.org/doc/rfc5280 (visited on 09/09/2022).

Bojanowska, A. (2019). "Customer Data Collection with Internet of Things". In: *MATEC Web of Conferences* 252, p. 03002. DOI: 10.1051/matecconf/201925203002.

Bruns, T. (2018). "Security Analysis of a Smart Home System: The Example of IKEA TRÅDFRI". In: p. 38.

Cabric, M. (2015). *Corporate Security Management: Challenges, Risks, and Strategies*. Oxford: Butterworth-Heinemann. ISBN: 978-0-12-802934-3.

Choi, J., M. Abuhamad, A. Abusnaina, A. Anwar, S. Alshamrani, J. Park, D. Nyang, and D. Mohaisen (2020). "Understanding the Proxy Ecosystem: A Comparative Analysis of Residential and Open Proxies on the Internet". In: *IEEE Access* PP, pp. 1–1. DOI: 10.1109/ACCESS.2020.3000959.

Danbatta, S. J. and A. Varol (2019). "Comparison of Zigbee, Z-Wave, Wi-Fi, and Bluetooth Wireless Technologies Used in Home Automation". In: *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–5. DOI: 10.1109/ISDFS.2019.8757472.

Diffie, W. and M. Hellman (1976). "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6, pp. 644–654. ISSN: 1557-9654. DOI: 10.1109/TIT.1976.1055638.

Eckert, C. (2008). *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. 5., überarb. Aufl. München Wien: Oldenbourg. ISBN: 978-3-486-58270-3.

Gupta, M. and R. Sharman, eds. (2009). *Social and Human Elements of Information Security: Emerging Trends and Countermeasures*. IGI Global. ISBN: 978-1-60566-036-3 978-1-60566-037-0. DOI: 10.4018/978-1-60566-036-3. URL: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-60566-036-3 (visited on 09/10/2022).

Lowe, G. (1995). "An Attack on the Needham-Schroeder Public-Key Authentication Protocol". In: *Information Processing Letters* 56.3, pp. 131–133. ISSN: 0020-0190. DOI:

10.1016/0020-0190(95)00144-2. URL: https://www.sciencedirect.com/science/article/pii/0020019095001442 (visited on 09/13/2022).

Morgner, P., S. Mattejat, Z. Benenson, C. Müller, and F. Armknecht (2017). "Insecure to the Touch: Attacking ZigBee 3.0 via Touchlink Commissioning". In: *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '17. New York, NY, USA: Association for Computing Machinery, pp. 230–240. ISBN: 978-1-4503-5084-6. DOI: 10.1145/3098243.3098254. URL: https://doi.org/10.1145/3098243.3098254 (visited on 08/28/2022).

NIST, N. I. o. S. a. T. (2001). *Advanced Encryption Standard (AES)*. Tech. rep. Federal Information Processing Standard (FIPS) 197. U.S. Department of Commerce. DOI: 10.6028/NIST.FIPS.197. URL: https://csrc.nist.gov/publications/detail/fips/197/final (visited on 09/23/2022).

Rescorla, E. (2018). *The Transport Layer Security (TLS) Protocol Version 1.3*. Request for Comments RFC 8446. Internet Engineering Task Force. DOI: 10.17487/RFC8446. URL: https://datatracker.ietf.org/doc/rfc8446 (visited on 09/17/2022).

Rivest, R. L., A. Shamir, and L. Adleman (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM* 21.2, pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: https://doi.org/10.1145/359340.359342 (visited on 09/23/2022).

Russell, D. and G. T. Gangemi (1991). *Computer Security Basics*. Rev. ed. Sebastopol, CA: O'Reilly & Associates. ISBN: 978-0-937175-71-2.

Thangavel, D., X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan (2014). "Performance Evaluation of MQTT and CoAP via a Common Middleware". In: *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pp. 1–6. DOI: 10.1109/ISSNIP.2014.6827678.

Wodtke, C. (2021). "Sicherheitsanalyse der Implementierungen des Zigbee-Standards für drahtlose Netze". In: p. 66.

Internet sources

CWE, C. W. E. (2022). *CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking*. URL: https://cwe.mitre.org/data/definitions/649.html (visited on 08/30/2022).

gcaillet (2020). *Android : Add Cert to System Store*. URL: https://gist.github.com/pwlin/8a0d01e6428b7a96e2eb (visited on 09/03/2022).

Google (2022). *KeyStore*. URL: https://developer.android.com/reference/java/security/KeyStore (visited on 09/18/2022).

Haan, G. de (2022). *Ikea Tradfri CoAP Docs*. URL: https://github.com/glenndehaan/ikea-tradfri-coap-docs#authenticate (visited on 09/20/2022).

IKEA (2022a). *Ein SYMFONISK Duett von IKEA und Sonos*. URL: https://www.ikea.com/at/de/new/symfonisk-wifi-speaker-pubfa93e07f (visited on 09/21/2022).

— (2022b). *Privacy Statement for IKEA Home Smart Application*. URL: https://ww8.ikea.com/ikeahomesmart/privacy/privacy_policy_en_us.html (visited on 09/15/2022).

MobSF (2022). *Mobile Security Framework - MobSF Documentation*. URL: https://mobsf.github.io/docs/ (visited on 09/21/2022).

Mozilla (2022a). *HTTP Request Methods - HTTP — MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (visited on 09/17/2022).

— (2022b). *Lazy Loading - Web Performance — MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading (visited on 09/18/2022).

— (2022c). *What Is a URL? - Learn Web Development — MDN*. URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL (visited on 09/17/2022).

Oracle (2022). *Certificate Attributes*. URL: https://docs.oracle.com/cd/E24191_01/common/tutorials/authz_cert_attributes.html (visited on 08/30/2022).

Poczta - Android Static Analysis Report (2020). URL: https://zaufanatrzeciastrona.pl/wp-content/uploads/2020/04/PDF2.pdf (visited on 08/30/2022).

W3Schools (2022). *JSON vs XML*. URL: https://www.w3schools.com/js/js_json_xml.asp (visited on 09/11/2022).

Walton, J., JohnSteven, J. Manico, K. Wall, and R. Iramar (2022). *Certificate and Public Key Pinning — OWASP Foundation*. URL: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning (visited on 09/13/2022).

Wireshark (2022a). *Wireshark · Display Filter Reference: Index*. URL: https://www.wireshark.org/docs/dfref/ (visited on 09/21/2022).

— (2022b). *Wireshark User's Guide*. URL: https://www.wireshark.org/docs/wsug_html//#ChIntroNoFeatures (visited on 09/22/2022).

Zigbee FAQs — Frequently Asked Questions (2022). URL: https://csa-iot.org/all-solutions/zigbee/zigbee-faq/ (visited on 09/10/2022).