UNIVERSITY OF TARTU
Institute of Computer Science
Cybersecurity Curriculum

Mikus Teivens

Analysis of Security and Privacy Issues in Common Smart Home Products

Master's thesis (24 ECTS)

Supervisor: Arnis Paršovs, PhD

Tartu 2021

AUTHOR'S DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Mikus Teivens

14.05.2021

ANALYSIS OF SECURITY AND PRIVACY ISSUES IN COMMON SMART HOME PRODUCTS

Abstract: The smart home devices are manufactured with the idea that the house owner should be able to automate various heating, lightning, energy consumption heavy tasks. While the idea of the smart house sounds great, as with all other things, the convenience does not come without a price, and in authors opinion this price to pay is usually the privacy of the consumer. In this thesis the author will examine a few, most commonly available smart home solutions in the Baltic region and analyze what potential security and privacy issues these solutions or the applications controlling them, might introduce.

This thesis is written in English and is 66 pages long, including 8 chapters and 19 figures.

Keywords: smart home, firmware, privacy, mobile

CERCS: T120 Systems engineering, computer technology

LEVINUD NUTIKA KODU SEADMETE TURVA JA PRIVAATSUSKÜSIMUSTE ANALÜÜS

Lühikokkuvõte: Nutikodu seadmed on toodetud mõttega, et majaomanik saaks automatiseerida kütte-, valgustuse- ja energiatarbimist. Kuigi idee nutimajast tundub hea, aga nagu kõikide muude asjadega, mugavus maksab. Autori arvates, mugavuse hind tuleb tarbija privaatsuse arvelt. Selles töös uurib autor mõnda olemasolevat nutikodu lahendust Baltimaade regioonis ja analüüsib potentsiaalseid turva ja privaatsusega seonduvaid probleeme, mida kontrollitakse läbi teatud tarkvara.

See lõputöö on kirjutatud inglise keeles ja on 66 lehekülge pikk, sealhulgas 8 peatükki ja 19 joonist.

Võtmesõnad: tark kodu, tarkvara, privaatsus. mobiiltelefon

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia

LIST OF ABBREVIATIONS AND TERMS

BLE	Bluetooth Low Energy
OWASP	Open Web Application Security Project
IPC	Inter-Process Communication
EEPROM	Electrically Erasable Programmable Read-Only Memory
DEX	Dalvik Executable
GPS	Global Positioning System
ATS	App Transport Security
CA	Certificate Authority
SSL	Secure Socket Layer
TLS	Transport Layer Security
DOM	Document Object Model
USB	Universal Serial Bus
API	Application Programming Interface
JSON	JavaScript Object Notation
SDRAM	Synchronous Dynamic Access Memory
SSDP	Simple Service Discovery Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
ELF	Executable and Linkable Format
OTA	Over the Air
UUID	Universally Unique Identifier
CoAP	Constrained Application Protocol
APK	Android Package

Table of contents

List of figures

1 Introduction

There exists a lot of various smart home solutions at this moment of time and they vary from very trivial light controllers to a rather advanced chain of various devices, which control most, if not all the significant functions of a modern household, such as locks, windows blinds, tv and audio devices, various appliances, security cameras, irrigation, locks and of course the lightning. The author was always interested into this somewhat new technology and various dilemmas one needs to solve in order to automate ones living space. For example, would you rather implement a solution which phones back to manufacturers cloud, if so, what information is sent to the cloud, who can interact with it, how secure is it, what are the privacy concerns if one chooses to follow this route. On the other hand, if one has made a conscious decision to run a self-sustained solution, how would the remote controls work and where would one store the data coming from the home controller.

The author has chosen to inspect and compare three different solutions for the home automation, most popular and affordable to a local consumer in the Baltics region, at the time of the writing, and analyze how secure are the communication between the devices, consumer, and the manufacturer. The security of the remote home-automation portal if available and the mobile applications meant to provide the remote control. The author also will verify, if the manufacturer has implemented any tracking possibilities and is able to track consumers actions.

The author will only use devices he owns or has been granted access to and will only perform tests with and to accounts he owns to eliminate any interruption of the service for the other, legitimate consumers.

2 Tests and Environment

The following list contains a brief overview of aspects the author tested on each home automation solution.

Hardware devices:

- Security of communications between consumer, sensors, and manufacturer.
- Existence and security of the update mechanism.

For Mobile applications:

- Communication security.
- Privacy concerns when using provided home automation mobile applications.
- If any sensitive data are stored locally on the phone in an insecure manner.
- Security of the update mechanism.
- Usage of best practices.

Web portal and API:

- The overall security of the frontend and backend services.
- Communication security.
- Usage of best practices.

2.1 Test Network Setup

To intercept the traffic from the base stations and mobile applications, a Raspberry PI was used, since it provides a Linux environment. A Wi-Fi access point was created for the mobile devices and the Ethernet port was used for base stations which required an internet connection. The ethernet interface was added to the bridge and Hostapd [1]

responsible for managing the Wi-Fi access point was set up to use the Bridge via the following configuration:

```
country_code=LV
interface=wlan1
bridge=br0
ssid=***REDACTED***
hw_mode=g
channel=7
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=***REDACTED***
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

The DHCP server was set up to provide IP addresses on both wireless and ethernet network interfaces. All the default settings apart from creating a subnet with the following settings, from which the IP addresses were served in response to the DHCP requests:

```
subnet 192.168.2.0 netmask 255.255.255.0 {
  range 192.168.2.10 192.168.2.200;
  default-lease-time 600;
  max-lease-time 7200;
  option routers 192.168.2.1;
  option domain-name-servers 8.8.8.8;
}
```

This setup enabled the author to intercept all network traffic between mobile devices, smart home devices and the internet.

2.2 Android Test Environment

The Android application was tested with the Genymotion Android Emulator for Desktop [2], which is capable to emulate a wide range of Android versions as well as various devices by utilizing Oracle VM VirtualBox for x86 virtualization. Genymotion enables to debug Android applications in a convenient way without the need to have a physical Android phone, and if something gets broken on a way, the Android virtual machine can just be reverted back to latest snapshot or reinstalled in few minutes via few mouse clicks. For the testing of mobile application, a Google Pixel 3 virtual machine with Android 10.0 was used.

2.2.1 Android Network Traffic Interception

From the testing perspective it is an essential necessity to inspect the network traffic between the application and the backend server. The usage os plain text communications have been discouraged for years now, so it is expected that the TLS connection from application must be intercepted, otherwise the plaintext connection would be a vulnerability on its own, if sensitive data are transmitted. The TLS connection interception can be done by utilizing the Burp Suite [3], the proxy server for anything speaking HTTP and more by the help of additional modules. By observing traffic proxied by Burp [3] it was confirmed that indeed the application communicates through TLS connections, since the failures of negotiating the TLS connection with the backend were observed in Burp`s event log. Before Android version 6, this could be easily circumvented by installing the CA certificate used by the Burp proxy [3], however, since Android version 6 the certificates installed in the user certificate store are ignored by the applications. To circumvent this restriction, the CA certificate must be installed in the system certificate store, which requires to remount the system partition in read/write mode. Luckily in Genymotion VM this can be achieved with relative ease by issuing the following commands using Android Debug Bridge:

```
adb root
adb remount
```

The CA certificate is exported from the Burp proxy [3] in DER format, and must be converted to PEM:

```
$ openssl x509 -inform DER -in cacert.der -out cacert.pem
$ openssl x509 -inform PEM -subject_hash_old -in cacert.pem | head -n 1
9a5ba575
$ mv cacert.pem 9a5ba575.0
```

Then just uploading the CA certificate to the Android VM, and rebooting the VM afterwards:

```
$ adb push 9a5ba575.0 /system/etc/security/cacerts/
196 KB/s (1375 bytes in 0.006s)
$ adb shell chmod 644 /system/etc/security/cacerts/9a5ba575.0
```

After this procedure, if the application does not employ the SSL Certificate Pinning, the TLS traffic should be seen in plain text in Burp proxy [3].

2.3 iOS Test Environment

Since there are no fully functional iOS emulators to the authors knowledge, the jailbroken iPhone SE running iOS 14.4.1 was used.

2.3.1 iOS Network Traffic Interception

In contrast to Android, it is rather straightforward to install custom CA certificate in iPhone by visiting the http://burpsuite, allowing the site to download a configuration profile and then installing it from the phones “Settings” menu [4]. After the profile has been successfully installed, and, if no SSL pinning is used, the encrypted traffic can be observed in plain text via the Burp proxy [3].

2.4 Insecure Application Backgrounding

The backgrounding is a feature provided by both, Android an iOS operating systems. When user either switches between the mobile applications or minimizes any of them, the screenshot of the active view of the application is stored on the mobile phone and provides the functionality of displaying the active content of the applications, when switching in between them. This feature in itself obviously is not a security threat, however, the user has to be mindful what information gets stored in the screenshot, when mobile application gets either switched or minimized, as sometimes these screenshots can leak sensitive information. To safeguard against such leaks, a mobile application developer can instead choose to use a static picture instead of a dynamic content of the application as a screenshot displayed during the switching of the applications or set a specific flag indicated, that the particular application view contains a sensitive information and should not be screenshotted automatically, when switching in between applications. For Android devices such screenshots can be found in the following directories:

- /data/system_ce/_USER_ID_/snapshots
- /data/system/images if the File-Based Encryption is not used.

For iOS devices, these screenshots are usually stored at the following directory:

- /private/var/mobile/Containers/Data/Application/APP_UUID/Library/SplashBoard/Snapshots/

2.5 Sensitive Data Stored Locally

It is not unusual for the applications to store data on the local phone, and in most cases, it is a necessity. The best practices are to encrypt such data on the device or use some of the storage methods described in the chapter 3 Basic Mobile Application Security.

By manually inspecting the files stored by the mobile applications on the mobile device, the author verified whether the sensitive data was stored in insecure way on the devices.

3 Basic Mobile Application Security

The author did choose to include the following chapter to help explain the base security concerns of mobile applications to make the thesis topic more understandable.

Mobile applications have soared in popularity since the smart phones were introduced to the consumer and have presented unique challenges from the security standpoint, among others. One of the biggest challenges for an aspiring mobile application developer would be the need to understand, that the mobile application runs in an untrusted device from developer perspective. The fundamental difference from, for example, web applications, where most of the code is executed server side, mobile application runs most of the code client-side, unless the application is a primitive wrapper around the web browser. Vulnerabilities arising from failing to understand these differences have been rated as the most occurring vulnerabilities in latest OWASP mobile top 10 vulnerabilities list [5].

The undoubted leaders in mobile phone market are various Android based phones with around 71% and iOS with 27% global market share, however, each of those mobile operating systems have a different approach to security of applications, user data and network communications.

3.1 Basic Android Application Security

Android is a mobile operation system developed and backed by the Google. It run on Linux and uses a spinoff of Java as a main language for developing mobile applications. Compared to the Apples Appstore, Google Playstore for Android is more open, and while there exists a sort of app review before publishing, judging from history it is far easier for a malicious app to get approval on the Playstore. It is also easier for an end user to root the device and gain full control over it or install homemade apps on Android.

Similar as with iOS, Android too provides sandboxing for applications, which efficiently locks each and every application in a separate container. A huge difference from iOS is the ability to interact between applications with less restrictions if the application author chooses so. More freedom is granted for communication in between applications on the

phone as well. This data exchange or access between applications is called Inter-Process Communication or IPC and usually means either communication between applications or communication between processes in a multi-process application. To manage this communication one of the most common ways is to use so called Content Provider, which allows the communication via SQLite database, internet, or files. It manages access to images, audio and video. A content provider must be defined in application manifest and must have the **android:exported** attribute set to true, to allow other applications to access the stored data. For more interactive and feature rich communication it is recommended to use so called Intent which can manage one the following three things [6]:

- Start an Activity representing a single screen in the application.
- Start a Service which runs in the background without user interaction.
- Send or receive a Broadcast, which is considered a best practice of delivering messages in between applications.

To protect sensitive data from leaking via a backed-up copy, there exists a directive **allowBackup**, which would deny backing up the data from the phone. While it offers some level of protection, in case phone is not properly wiped, an attacker can jailbreak it and still have a full access to the files. Since the Android release version 6, all data on the phone should be encrypted by default using full-disk encryption. However, since there exists many phone manufacturers and various devices, for devices not capable of encryption, this requirement can be omitted. This issue plagues Android even to this day, since there is a huge segmentation of manufacturers compared to the Apple. Starting from Android release version 7, a different approach called file-based encryption was introduced allowing to encrypt different files with different keys, and after Android release version 10 it should be enabled by default. For storing user credentials Android offers so called System managed Keystore with two different approaches named Keychain and Android Keystore Provider. The Keychain works system vide, meaning if user consents, an app can access stored data in Keychain. Android Keystore, however, separates data allowing only the app that stored the data to access it later [7].

Since the Android release version 9, plain text HTTP connections are blocked by default. For managing traffic encryption in Android, one must use the so-called Network Security Configuration feature where application author can finetune connection properties, such

as allowing self-signed certificates, unencrypted connections, certificate pining and similar options.

A huge help for someone who wants to inspect an application without having access to the source code is the relative ease one can decompile an Android application if the code is not obfuscated. The contents of the application are stored in the .apk file, which is essentially a Zip archive containing application code compiled in dex format among other things. These dex files can then be translated into readable Java code if the obfuscation wasn't used when compiling the application.

As already stated above, there are many more features and security scenarios to consider, but in the context of evaluating the smart home solution those are not that important in the authors opinion.

3.2 Basic iOS Application Security

The iOS created by the Apple is built on top of modified BSD and use either Objective-C or lately Swift as primary languages for app development. iOS is a rather locked down ecosystem, for example, in order to get one's application on the Appstore, it must first go through a review, which ensures the basic standards for iOS application are met or the application is not outright malicious to the end user. The Apple is the only entity in charge of verifying submitted apps before publishing and there has been more than one occasion, when Apple has been accused of wrongfully disallowing the publication or removing the app from the Appstore. Once the application gets published in the Appstore, anyone, provided the application is not region restricted, can download it from the Appstore.

Each application on the phone is locked inside its own sandbox and unable to interact with data outside the sandbox with some exceptions. If app requests an access outside its sandbox, for example, to downloaded files or GPS location data, this access is managed by the iOS, and it is called the entitlement. The application must have any of such entitlements listed when the application is submitted to the Appstore, and if user consents to requested entitlement, an application can access data outside the sandbox. The applications can also interact with each other in few limited ways, the most popular being so called deeplink. After the application has declared the deeplink, any other app calling it can interact with the application via the registered deeplink, for example, by calling the

myapp://openUrl?url=example.com could redirect user to example.com via the application, which has registered the **myapp** deeplink. Another way for communication would be through the clipboard, a feature which recently made it into news when security researchers noticed a bunch of apps accessing clipboard extensively, thus raising a privacy concerns [8]. The Apple has since made a change into iOS to alert mobile user, whenever application accesses the clipboard.

Since the release of iOS version 4 there is an option for encryption of application data enabled by default if the passcode for the phone is set, named Data Protection API. It offers four levels of protection:

- No protection
- Complete until first user authentication - A default setting if the pin code is set for the phone, decrypting data once the user has unlocked the device for the first time after an reboot and keeping the data decrypted until the reboot.
- Complete unless open - Decrypting data when the application first accesses the file.
- Complete - Allowing to access data only when the device is unlocked.

For shorter data streams, for example, usernames and passwords there exists another way of storage named Keychain. The keychain provides a key-value type of storage. Each application has its own partition of space and no other applications can access the stored data unless application author has explicitly allowed it. For each record stored in the keychain one of 7 different access properties can be assigned and limit the storage only to the device or allow backing it up as well. The main difference in between the Data Protection and Keychain is that the entry in the Keychain persists after the application has been uninstalled from the phone. The preferred way for storing the sensitive data, such as usernames and passwords are Keychain since it has an ability to enforce the enabling the PIN code for the phone. Data Protection on the other hand does not enforce it, and in the event of phone without the pin code, the data stored in files is not encrypted or protected by any means.

Unless iOS application does not need to communicate via the network or transmits absolutely nothing sensitive, it is recommended to use TLS for transmitted data encryption. Since iOS 9 there is a feature named App Transport Security or ATS with the help of which it is possible to enforce all outgoing HTTP connections to get upgraded to HTTPS. It is possible to somewhat downgrade this feature by either whitelisting domains where the ATS should not be enforced or allowing to browse web in case the application has such functionality via the web browser wrapper without ATS but enforcing all other outgoing HTTP connections issued by the application to obey the ATS. To further secure the connection, it is common practice to use so called SSL Pinning to avoid scenarios when attacker is able to either issue a valid certificate or install a malicious CA on victim's phone. Usually, SSL Pinning is done by verifying the hash of the public key of the server and in case of a mismatch the connection is rejected.

There are many more features and security scenarios to consider, but in the context of evaluating the smart home solutions those are not that important in the author's opinion.

4 Basic Web Application Security

The web security is a wide topic and beyond the scope of this document. The author will try to describe only relevant parts related to the thesis topic to make the issues more understandable.

The HyperText Transfer Protocol or HTTP is the protocol driving the exchange of information between the client and the server hosting the web application. It was invented by the Tim Berners-Lee around 1990. The HyperText Markup Language or HTML is used to define how the documents containing information served via the HTTP will be formatted and displayed by the web browser, which is a program ran by the client on any device capable of browsing the Internet, be it a PC, laptop, mobile devices or tablets.

The process of issuing a HTTP request, interpreting the response and displaying it would follow the flow briefly described below:

- User issues an HTTP GET request, the web browser automatically adds information to this request, such as accepted data formats, encoding, user operational system and similar information [9].

- The web server looks up the requested file on its filesystem, and if it exists, responds with the HTTP status code 200 and the actual requested file, specifying encoding, data type etcetera.

- The web browser on user's system parses the response from the web server, requests and downloads any additional files if necessary, such as pictures, JavaScript files, and so on, and renders the requested resource in the web browser.

There exist various HTTP request methods [10], the most used ones being GET for requesting the resource, POST for submitting a chunk of information to the web server and HEAD for requesting the resource without actually downloading it from the web server. The PUT, DELETE, PATCH are somewhat more obscure methods for the end user since they are mostly used to interact with the backend API of the web application. The API or Application Programming Interface was created to help the developers to

interact with the various services. For example, a developer can issue a specific request to the API provider and ask for the current weather conditions in a specific location. The description of provided API endpoints and required formatting of the request are published by the API provider if the API is meant for public access so to say. The API responds with the data formatted in a format easily parsable for machines. The predominant data format in modern APIs are JSON, which is both, easy parsable by machines, and easy to read by humans as well.

A simple request issuing a GET request for the resource /get located at https://httpbin.org from the Firefox 84 web browser would look something like this:

```
GET /get HTTP/1.1
Host: httpbin.org
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:85.0) Gecko/20100101
Firefox/85.0
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://httpbin.org/
Connection: close
```

As can be observed, the web browser indicates, that it can accept JSON formatted data for this particular request, it can accept gzip or deflate compressed answer and the referrer address.

A response from the webserver containing JSON formatted data for the requested /get resource from the https://httpbin.org would look something like this:

```
HTTP/1.1 200 OK
Date: Fri, 26 Mar 2021 19:10:57 GMT
Content-Type: application/json
Content-Length: 458
Connection: close
Server: gunicorn/19.9.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

{
  "args": {},
  "headers": {
    "Accept": "application/json",
    "Accept-Encoding": "gzip, deflate",
    "Accept-Language": "en-US,en;q=0.5",
    "Host": "httpbin.org",
    "Referer": "https://httpbin.org/",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:85.0)
Gecko/20100101 Firefox/85.0",
    "X-Amzn-Trace-Id": "Root=1-605e31c1-15f01c3c4e9bc30225bf69c4"
  },
  "origin": "xx.xx.xx.xx",
  "url": "https://httpbin.org/get"
}
```

From the answer it can be observed that the data format is JSON, the length of the response is 458 bytes, the web server running is gunicorn/19.9.0 and some access control headers.

4.1 Tracking and Identifying User Sessions

To provide a dynamic interaction between web application and the user, there exists a mechanism, called a session which usually identifies each user by issuing a randomly generated token. By using a session token, the application can authenticate each user and respond with the appropriate data. This token can be either stored in a Cookie, a container dedicated to each site to store small chunks of data temporarily, or in the web browsers local or session storage. Server side this token is mapped to either temporary or permanent data store, storing various attributes for the unique user, for example, username, permission level, permission group, etcetera. Understandably such token should be guarded, as anyone possessing it would be able to impersonate the legitimate owner of the session. The cookie has 3 attributes developed over the years to help the safeguarding of the content of the cookie from the attackers.

- Secure [11] – If set, the cookie is allowed to be transmitted only via the HTTPS connection.

- HttpOnly [11] – If set, the access to the Cookie is forbidden from the JavaScript, to battle the stealing of the cookie by Cross-Site-Scripting attacks.

- SameSite [12] – Has 3 modes, “None”, “Lax” and “Strict”. If the “Strict” mode is set, the cookie is forbidden to be sent with the request coming from the third-party sites, limiting the chances for attacker to use Cross-Site-Request Forgery attack.

The Session Storage is a part of the Web Storage API. It provides a mechanism to store data locally until the web browser is closed. The important part here is the fact that data stored in the Session Storage is never transferred back to the web server and is accessible by the JavaScript. The Local Storage functions in the same way, the only difference is that the data stored in the local storage persists when the web browser is closed. There have been many arguments about which storage mechanism is the safest to use and there exists many articles discouraging the use of the Web Storage API for storing sensitive data, however, James Kettle, a prominent web security researcher suggests otherwise, arguing that the Web Storage API is more secure by the default [13].

The common practice for securing the web applications server side is to enable various, so called HTTP Security headers, which instructs the web browser what restrictions are or should be imposed. The following list contains the most popular headers:

- X-Frame-Options - Protection against UI redressing/Clickjacking attacks.

- Strict-Transport-Security - Protection against man-in-the-middle attacks.

- Content-Security-Policy - Protection against Cross-Site Scripting and Clickjacking attacks.

- X-Content-Type-Options - Protection against MIME-type confusion.

- X-XSS-Protection – Was supported by major browsers to enable protection against the XSS attacks, however, nowadays this header is not supported anymore, and the use of Content-Security-Policy headers are recommended as a replacement.

4.2 Same Origin Policy

One of the core concepts in modern web browsers is the so-called Same Origin Policy [14] which prevents JavaScript from interacting with the content from another "origin". The origin is defined as combination of protocol, host name and port. The JavaScript provides an easy way to interact with the HTML elements of the page via the web browser API called Document Object Model or DOM. The Single Origin Policy enforces restrictions so that the JavaScript from one browser tab cannot interact with the DOM of another tab or disallow the access to the cookies from another origin. An interesting case happens, when resources, such as the JavaScript files are loaded from another domain. Even if they are loaded from third party domain, the origin of all the resources loaded is assumed to be the one which loaded the resources. For example, if https://domain1.com loads some JavaScript from https://domain2.com/script.js, the origin of the script.js will be considered https://domain1.com.

A common misconception about the Same Origin Policy is that the origin cannot send information to another origin, however, it is not so since nobody can forbid the user to initiate requests to other origins. The content in one origin is considered to be secure, even if in reality it might not be so on every occasion. For example, if by this logic a creator of a website in one origin wants to issue request to another origin, it is assumed that there is a legitimate reason behind it. The Cross-Site Request forgery vulnerabilities exploit exactly this reasoning by forcing an unsuspecting user visiting one origin, to issue another request automatically to a different origin, where the user is authenticated, and web browser will include the authenticated session cookie, thus allowing the attacker issue arbitrary actions in the name of the user.

4.3 Cross-Origin Resource Sharing

The web applications have gotten more and more complex over the years, and there are quite a few cases, when there must be a cross origin communication, where one origin must communicate with another origin, and interpret the response from the issued request. One such example would be the API requests. To define the rules on what can be accessed in cross origin requests, a Cross Origin Resource Sharing mechanism was introduced, by adding specific headers to the issued HTTP requests and responses. When a cross origin request is initiated, a web browser automatically adds an Origins header, which cannot be

manipulated by the JavaScript, thus disabling the cases when malicious JavaScript code forges the Origin header. There exists two ways how to issue a Cross Origins requests [15]:

- Simple request – As the naming states, a simple request without the preflight check. It has somewhat limited features, but one of the gains is the not needing to send two requests instead of one.

- Preflighted requests – In case of preflighted request, an OPTIONS HTTP request is issued prior every actual request in order to determine the Cross Origin Resource Sharing rules and determine if it is safe to send the actual request.

For example, the following simple cross origins request is issued client side via the JavaScript. As can be observed, the web browser has automatically applied the Origin header:

```
GET /2/login.json?api_key=_API_TOKEN_ HTTP/1.1
Host: api.istabai.com
Origin: https://elektrumviedamaja.lv
Connection: close
Referer: https://elektrumviedamaja.lv/
[…]
```

The response from the API endpoint located at https://api.istabai.com contains Access-Control-Allow-Origins header indicating, the origins from which the requests are allowed. In the following case any origin is allowed to access as indicated by the wildcard in Access-Control-Allow-Origins header:

```
HTTP/1.1 200 OK
Server: nginx
Content-Type: application/json
Connection: close
Access-Control-Allow-Origin: *
[…]
```

In the following chapters of this work, security and privacy analysis of several smart home solutions will be analyzed by the author.

5 Istabai (Elektrum Viedā Māja)

The first home automation solution analyzed in this work is Istabai [16], which has been manufactured locally in Latvia by “Draugiem Group” and hailed as an innovative solution at the dawn of the home automation. However, in the present days it is somewhat limited with the capabilities and mostly focuses on heating and electricity control. None the less it is still a product which can compete within at least a few niches with the global leaders of the home automation business.

The Istabai home automation solution inspected by the author consisted of base station which communicates with the deployed sensors via the radio-waves on 868.MHz frequency. It was later discovered that a proprietary protocol was used instead of any of standard ones. There are some indications that Zigbee protocol was used as a reference to develop the proprietary protocol. However, at this point this is just an author’s guess. The consumer is required to install at least one base station to use Istabai home automation solution. Once the data has been gathered from the locally installed sensors by the base station, the data gets forwarded to the manufacturer’s API backend server in plain text, but encoded with, what looks like a proprietary encoding technique.

The base station hardware has a custom design board, with PIC32MX695F microcontroller, RJ-45 10/100 Base-T ethernet port, USB 2 Micro-B port for firmware updates, a RFM22 ISM transceiver module and two ST MICRO M95160 and ST MICRO M25P16 EEPROM flash memory chips. The Figure 1 contains the picture of the Istabai base station.

Figure 1: Istabai base station

The consumers have a few options to choose from on how they want to control the purchased sensors. There exists a mobile application for both iOS and Android phones, allowing remote control over the installed devices on the go. There is also a web page where the consumers can log in and perform same tasks similarly as from the mobile applications. However, they are then missing out on some of the features, like receiving alerts when base station goes offline. In either way, a consumer needs to either register an account via the mobile applications or in the web portal in order to start using the Istabai provided features.

Figure 2 represents a schematic describing how the mobile devices, webpage, a manufacturers backend server, bases station and various smart sensors are communication in between each other.

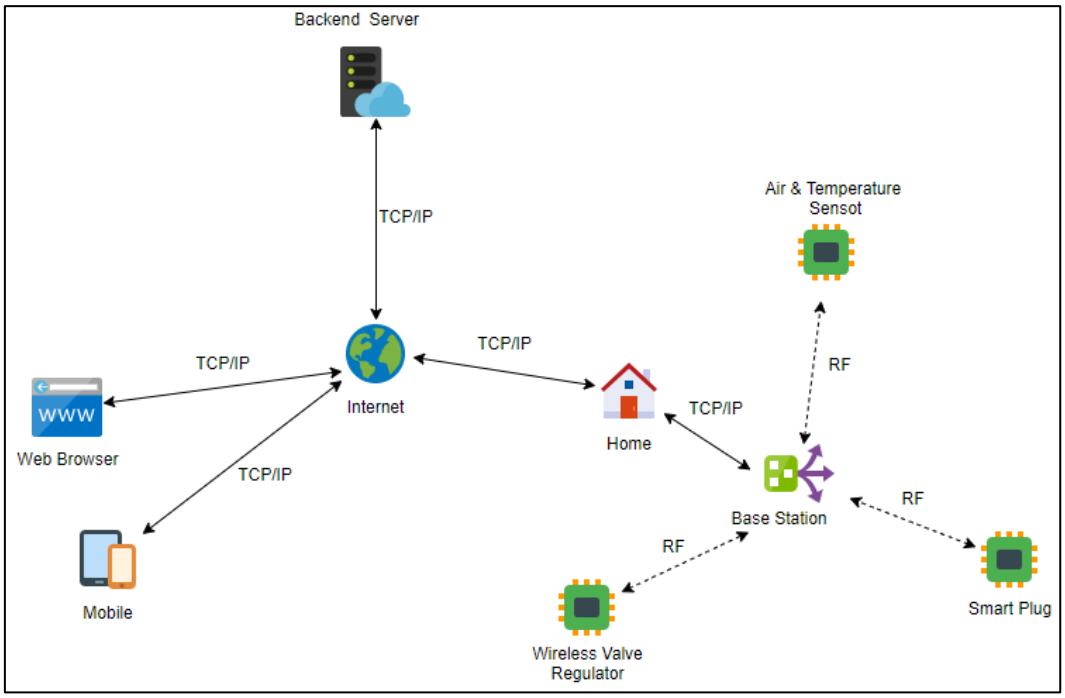

Figure 2: Communication between Istabai devices

To author's understanding, "Draugiem Group" has resold this solution to a couple of local vendors, some apparently have chosen to rebrand it completely, and integrate the mobile part functionality in vendors own mobile application. However, others have chosen to use the manufacturers infrastructure and mobile application with polished UI and the change of company's logos. The author's guess as a bystander is that the benefit for reusing the manufacturer's infrastructure is the lower price of hardware, since other resellers who use their own infrastructure sells same devices with the considerably higher price. In this paper, the author used the "Elektrum Viedā Māja" home automation solution [17], which uses the manufacturers infrastructure and has chosen to just rebrand the mobile application.

5.1 Update Process and Firmware

During the observation period (approximately three months), no update requests from the base station to the manufacturer were intercepted. Although this does not fully confirm that there is no way to upgrade the device remotely, the author believes the given time period is enough to strongly speculate about the missing update mechanism.

According to the device manual, the microUSB port found on the device is meant for physical upgrades, however, there are no detailed information about this process, nor there are any firmware images available online.

5.2 Network Communication to the Outside from Base Station

The communication between base station and the manufacturers backend server is done via the plain text. The base station connects and maintains a connection to the sensors.istabai.com. The commands can be observed in plain text, however, the payloads are encoded by a proprietary encoding mechanism (by author's guess). This ensures that a bystander cannot intercept and figure out immediately, exactly what payloads were sent. For example, the following encoded payload was observed via Tcpdump [18]:

```
~PACK.697274746E656220207973256100000FFFF#
```

Another intercepted example shows a base station reporting back the successfully executed task:

```
~TASK COMPLETE.5143281#
```

5.3 "Viedā Māja" Mobile Application

Both, iOS and Android applications communicated with the API server. The connection was secured with TLS 1.2, however, the server also supports deprecated TLS 1.0 and 1.2 versions. No user tracking was noticed by analyzing the application traffic and mobile application itself.

The Android application (draugiemgroup.istabai.elektrum) version 3.5.8 was used for testing.

The iOS application (com.istabai.istabai-elektrum), version 3.5.8 was used for testing.

5.3.1 Application Update Mechanism (Android)

During the test it was discovered that the application is a wrapper around the WebKit [19], basically displaying a web page, and the main logic is implemented via the JavaScript code stored locally on the Android device. An update check is started every time the user logs into application:

```
GET /client-
beacon/?platform=android&bundle=draugiemgroup.istabai.elektrum&version=8&env=
production HTTP/1.1
User-Agent: Dalvik/2.1.0 (Linux; U; Android 10; Google Pixel 3
Build/QQ1D.200105.002)
Host: api.istabai.com
Connection: close
Accept-Encoding: gzip, deflate
```

The response from the backend server provides the information about the latest build:

```
HTTP/1.1 200 OK
[…]

{"content":"https:\/\/api.istabai.com\/client-beacon\/payload-
elektrum\/content.zip","checksum":"ad2adcf46bcd361e6ee7de6a686187faf9ccf66857
88430ed0abc7bb7c623541","version":"3.5.8","size":1775386}
```

By intercepting and modifying the responses from the backend server, the author supplied an arbitrary download link with the incorrect SHA256 checksum, to verify if the checksum is actually compared during the update process:

```
HTTP/1.1 200 OK
Server: nginx
Content-Type: application/json
[…]

{"content":"http:\/\/10.10.10.20\/content.zip","checksum":"9c132de572816b08cc
d379f290a75348432a55bc4491af2df1f9f45968e6ea9d","version":"3.5.8","size":1776
697}
```

As observed in the terminal, a request for the modified update was successfully interpreted, and the application did issue a download request:

```
$ sudo python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
192.168.56.1 - - [21/Mar/2021 14:51:38] "GET /content.zip HTTP/1.1" 200 -
```

However, after the successful download, the application always crashes, indicating that the checksum is verified correctly before extracting the archive, but the error condition is not handled. Another way how to verify an update process would be to decompile the application and read the application code.

5.3.2 iOS Application Update Mechanism (iOS)

Similarly, as with Android application, the iOS application was discovered to be the wrapper around the WebKit [19] displaying a web page. The application logic is stored locally on iOS device and update check is done upon each login by issuing the following request:

```
GET /client-beacon/?platform=ios&bundle=com.istabai.istabai-
elektrum&env=production HTTP/1.1
Host: api.istabai.com
Accept: */*
User-Agent: Elektrum%20Vieda%20Maja/29061334 CFNetwork/978.0.7 Darwin/18.7.0
[…]
```

The responses from the backend server containing the version information were once again intercepted with the Burp [3] proxy, and arbitrary for the location of update archive and invalid checksum was injected in the response to the iOS application to verify if the checksum is validated. The modified update file was downloaded, the checksum was correctly verified and found to be invalid, and the downloaded data discarded without updating.

5.3.3 Sensitive Data Stored Locally (Android)

In the context of this particular Android application, a username and a password are used to authenticate via the web interface to the backend server. When the authentication is successful, the backend provides an API key, which is used to verify the authorization for every incoming request. It was observed that the LevelDB, a "fast key-value storage library written at Google that provides an ordered mapping from string keys to string values" [20] is used to store the API keys in clear text:

```
$ adb shell strings
/data/data/draugiemgroup.istabai.elektrum/app_webview/Local\
Storage/leveldb/000003.log

VERSION
META:file://
_file://
api_key!
***REDACTED***]dQ
META:file://
_file://
api_keySvN
META:file://
_file://
api_key!
***REDACTED***
```

The storage of the credentials in plain text on the mobile device is not a serious vulnerability, considering the fact that the device contents are encrypted by default on the latest Android versions. However, it is a missing best practice, since the Android Keystore is designed specifically for storing such sensitive information. Judging by the size of the Android version market share of the Europe in the year 2021 [21], it can be verified that

the predominant Android versions in use are 9.0 with 18.82% and 10.0 with 48.42% market share, meaning that even data stored on the devices in plain text should be encrypted by default and protect against compromise in situations if the device gets stolen or is not wiped before changing owners in most cases, provided the secure pin code is set. The author assumes that the decision to store API key outside keystore was made, since the application is basically JavaScript running in the WebKit wrapper. To the authors knowledge there is no easy way to access the keychain, a recommended mechanism to store sensitive data, from the JavaScript.

The author did verify, that the secure application backgrounding is not implemented in the Android application. The Figure 3 shows the screenshot stored by the application, which contains potentially sensitive information about the consumer.

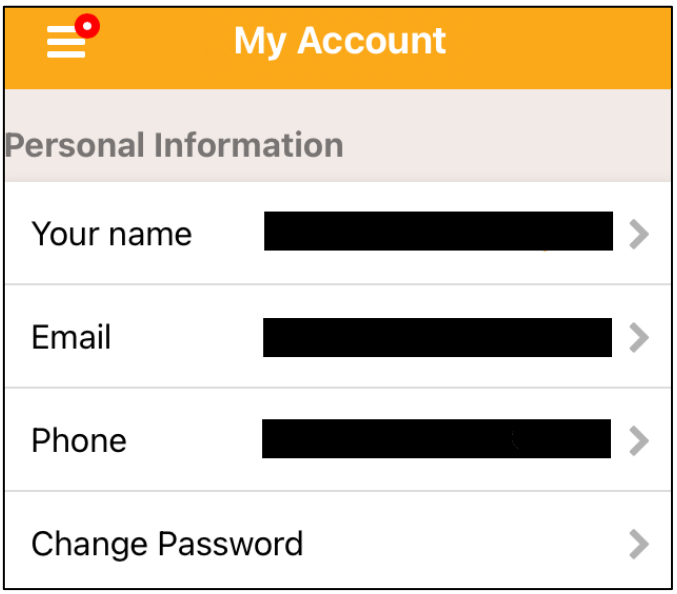

Figure 3: Insecure backgrounding in “Viedā Māja” Android application.

In author’s opinion this is a low-risk issue, since the screenshots are regenerated quite often, when the mobile application is opened or closed, more so, the potential attacker would have to have a physical access to the device, however, it is considered best practice to use a static image for the application, whenever its backgrounded.

5.3.4 Sensitive Data Stored Locally (iOS)

Similar as with Android application, the iOS application stores API keys in the plain text, on the device, using WebKit [19] local storage. The API key is stored in the:

```
/var/mobile/Containers/Data/Application/_APP_UUID_/Library/WebKit/WebsiteData
/LocalStorage/file__0.localstorage
```

The file storing the local storage is an SQLite3 database and can be viewed by either using an SQLite client or just dumping the file content as can be observed in the Figure 4.

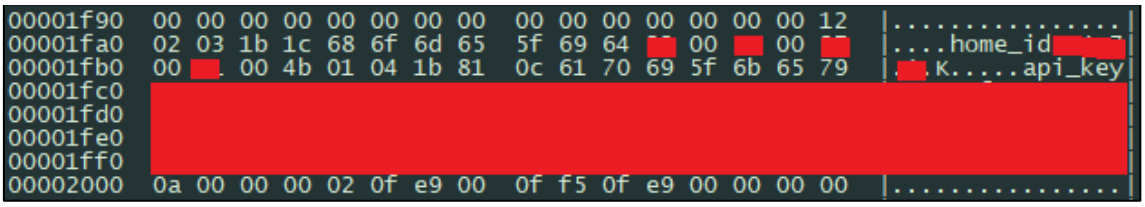

Figure 4: iOS localstorage containing Istabai API key

The same issue with insecure application backgrounding was found on the iOS application as well. For iOS devices the taken screenshots can be found at the following directory:

```
/var/mobile/Containers/Data/Application/_APP_UUID_/Library\SplashBoard\Snapsh
ots\sceneID_com.istabai.istabai-elektrum-default
```

Figure 5 shows that sometimes it is possible for the application to leak rather sensitive information, such as base station ID required for pairing the user account on the web portal with the actual device and IP address, from there the incoming data has been sent by the specific gateway.

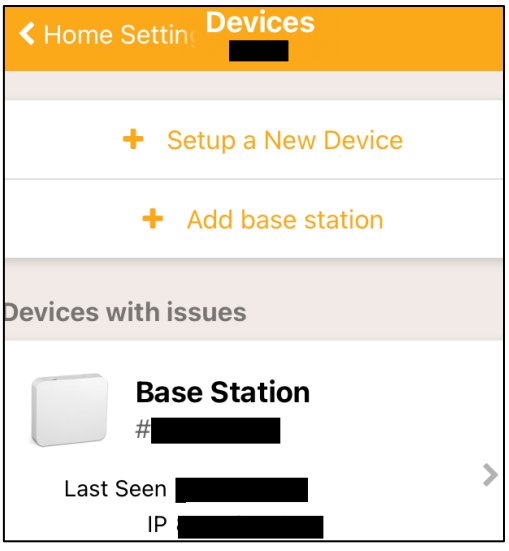

Figure 5: Insecure backgrounding in “Viedā Māja” iOS application.

5.3.5 TLS/SSL Connection security (Android)

In this case the SSL Certificate Pinning was not implemented for the Android application, and the author was free to observe the traffic in plain text from the Android application to the API backend. While the missing SSL Pinning on its own is not so easy to exploit, if the certificate validation is done correctly, the missing SSL Pinning it is still considered a serious issue.

To test if certificate validation is done correctly, a traffic from the application was directly forwarded from the application to the Burp [3] proxy listener, without setting the proxy settings in the Android emulator. The simplest way to achieve this redirect was to just add an entry in the /etc/hosts file and setting the IP address of the Burp [3] proxy to resolve as the API backend server when the Android application tries to connect. On the Burp`s [3] side, the proxy listener was set to listen to the port 443, and "Support invisible proxying" was enabled in "Request handling" tab. The certificates were set to "Generate CA-signed per-host certificates". The connections to the backend server were rejected proving that the TLS certificate verification was done correctly.

5.3.6 TLS/SSL Connection security (iOS)

The author was able to successfully view the traffic from application to the backend server and successfully verify, that the SSL Certificate Pinning is not used by the iOS application the same as with the Android "Viedā Māja" application.

The correct validation of the TLS certificates was tested differently from how it was tested for Android. Instead of adding a hardcoded DNS entry, the python code [22] responding with the single IP address to every DNS query was executed, and the IP address of the server running the code was set as an DNS server on the iPhone. To simulate an HTTPS server, another piece of Python code was used [23] which would serve HTTP requests with self-signed TLS certificate. By inspecting the traffic to simulated HTTPS server it was successfully confirmed that the application-initiated connection successfully but failed to create a full HTTPS connection indicating that the certificate validation is done correctly by the iOS application.

5.4 “Viedā Māja” Web Portal/API backend

The web application has a frontend running accessible at https://elektrumviedamaja.lv/ and the backend API accessible at https://api.istabai.com/. The web frontend is the same JavaScript running client side via the web browser as with the mobile applications. The users can view the temperature and air humidity graphs, manage paired smart plugs, view the messages about the availability of the base station, manage their home, or homes in case of multiple premises using home automation, and manage profile settings. There is also a possibility to add additional users in case the homeowner wants to share the access to the home control, supposedly to the family members.

The functionality presented by the web frontend is executed by issuing the requests to the API backend. The full documentation of the API can be found at the https://istabai.com/api/.

5.4.1 Missing HTTP Security Headers

By visiting the web application at https://elektrumviedamaja.lv/ and intercepting the initial response from the web server, a potential problem was already identified. By observing the response, it can be verified, that none of the HTTP Security headers are used by the application:

```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 16 Mar 2021 19:27:11 GMT
Content-Type: text/html
Last-Modified: Mon, 04 Jan 2021 20:04:11 GMT
Connection: close
ETag: W/"3cc374aa-2bd"
Content-Length: 1243
```

The missing headers are not a vulnerability in itself, however, it is a missing security feature weakening the security of the web application.

5.4.2 Sensitive Information in GET Requests

An authentication is one of the core parts of any application which implement a private access to each user’s data, and also one of the first one every user will be forced to face when accessing the application. The authentication for the application is done via the GET request to the API backend:

```
GET /2/login.json?email=user%40domain.com&password=_PASSWORD_ HTTP/1.1
Host: api.istabai.com
[…]
```

As can be observed in the request, there is one issue from the security viewpoint. The use of the HTTP GET method instead of POST means, that the username and password is stored in the web logs on the backend server, if the logging is enabled. The logging of the full POST requests could have been easily enabled on the backend as well, however, it is probably not feasible to store such amount of data in the production system, since POST requests tend to transmit more data. The response from the API contains a JSON formatted string:

```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 16 Mar 2021 13:17:54 GMT
Content-Type: application/json
Connection: close
Access-Control-Allow-Origin: *
Expires: Sun, 01 Jan 2014 00:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Cache-Control: post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 162

{"user":{"id":1111,"name":"Username","api_key":"***REDACTED***","lang":"LV","
homes":1},"timestamp":1615813474,"homes":[],"home_invitations":[]}
```

It can be observed that no HTTP security headers are served by the API endpoint as well (author assumes that the Access-Control-Allow-Origin header for the Cross Origins Resource Sharing is set to allow any origin, since the manufacturer did open the API access to the public, and various implementations would otherwise be unable to access the API endpoints in some cases, same applies for the manufacturers own mobile application). The returned JSON data contains a bunch of information about the logged in user together with the api_key which is stored in the browser's local storage and thus persists after the web browser has been closed. The author speculates, that from the design perspective this makes sense, since the web application is based on the JavaScript, so it does not make a difference if the api_key is stored in the cookie without httponly flag, or in the web browsers local storage. The contents of the local storage can be observed by opening the web developers' tools of the web browser, and viewing the storage tab as can be observed in Figure 6:

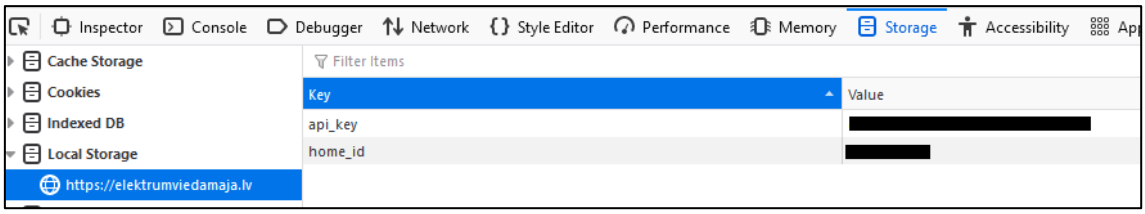

Figure 6: Istabai api_key stored in the local storage.

In addition, the API endpoint responds to the plain HTTP requests offering the same functionality as the API endpoint served over the HTTPS. To verify if this issue can be exploited together with the missing Strict-Transport-Security headers, a Man-in-the-Middle attack was executed by utilizing a Linux virtual machine acting as a router and running Bettercap [24] to perform the actual attack and Windows 10 virtual machine acting as a client. The Bettercap [24] was started on the routing network interface and the following Bettercap [24] settings were set:

```
set net.sniff.verbose false
net.sniff on
set http.proxy.sslstrip true
http.proxy on
https.proxy on
```

Since all web application requests are either redirected to the HTTPS or have HTTPS hardcoded for all the requests it was determined that the missing Strict-Transport-Security headers does not pose an immediate security concern, however, the author would suggest following the best practices and disable the serving of the API requests over the plain HTTP altogether, since someone reusing the API insecurely via the plain text HTTP would put all users using the custom implementation at risk.

5.4.3 Old API Keys Not Terminated

The biggest problem from the author's viewpoint is the fact that the old API keys are not invalidated, efficiently meaning that the logout function it the Web frontend is meaningless. The Web frontend offers the logout functionality, and when used, the user gets logged out and the api_key value gets deleted from the web browsers local storage. Upon next login, a new API key is generated. However, the old, deleted api_key token is still valid, and, if the attacker has gotten hold of the api_key, it efficiently allows an unrestricted access to the victims account for prolonged time period and victim has no way to deny the access to his account to the attacker, once one of the API keys are compromised. It was tested that the API key was still valid approximately three months

after the user has logged out, which itself would not be a vulnerability if there would exist a way to invalidate existing API keys.

5.4.4 Testing for Guessable Device IDs

Each device has a unique ID number assigned by the manufacturer, and the author assumes that each time the data is transmitted to the manufacturers cloud, these device ID numbers are transmitted as well, to correctly associate data sent. As far as the author was able to tell, the gathered results are associated with the base station ID, of base station that transmitted the data to the cloud after gathering it from the local devices in the consumers apartment. Once the consumer has registered a mandatory account with the manufacturer, he is then able to add one or more base stations, providing the IDs which are printed on the back of each base station. The base station ID consists of three alphanumeric symbols, let's assume it is "D11" and three alphanumeric characters, let's assume "A1B", which makes it easy for the potential attacker to guess the valid base station IDs. The same principle applies to all the other sensor devices, however, the prefix for each type of the device is different, and the length is longer as well. The API works in a way which forbids to add any device or base station to the users account, if it is already associated with different account in the system, so even if the attacker is able to guess a valid ID, he is unable to associate with the device if it is already associated. This, however, enables two possible attack scenarios:

- The attacker associates with all available, unassociated device IDs, thus making it impossible for the legitimate consumer to associate and gather with the devices he has purchased later.

- To enumerate all valid and already associated device IDs, by trying to associate them with attacker's account. The API responds differently when someone tries to associate with already associated device, thus enabling the attacker to gather all associated IDs by writing an automated script which will try and associate all possible device IDs.

The author did test the automated enumeration in the practice, he chose to send one hundred requests with invalid base ID and immediately after a request adding a base station with valid ID was successful. As a simple fix for this problem the author would

like to suggest the rate limiting all API calls and increase the response time of the subsequent API calls.

6 Philips HUE

The Philips HUE is a trademark for smart light solution from Philips and it was one of the first manufacturers to allow this type of functionality for the customers back in the 2012. The traditional way of controlling the lights is via the hub together with smart phone app, however, recently Philips has added an option to control lights via the Bluetooth as well, thus the customer is not forced to buy the control bridge if he does not want to. The downside to this setup is the transmit range limit for Bluetooth and inability to control lights remotely, when the customer is away from home. Philips HUE products are also compatible with all the big players in home automation today, such as Alexa, Apple HomeKit, Google Assistant, and others. The HUE bridge on the other side maintains a connection to the manufacturer via TCP/IP and enables the customer to control lights remotely. The ability to control the lights while away from home is not enabled by default, and in order to enable it, the customer has to create an account with the Philips and authorize each mobile device via OAuth to allow the remote control. The communication between the bridge and the lights is done via the Zigbee protocol. The Figure 7 represents a schematic describing how the mobile devices, the Philips HUE bridge and smart lights are communication in between each other.

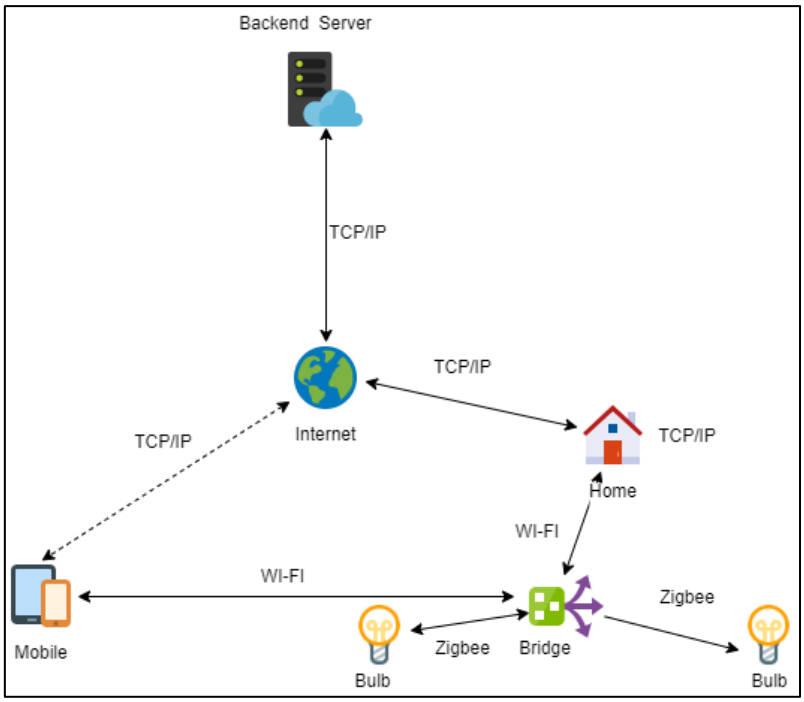

Figure 7: Communication between Philips HUE devices and the consumer

Hardware wise, the Philips HUE Bridge has Winbond W9751G6KB-25 512Mb DDR2 SDRAM, QCA4531-BL3A SOC supporting Wi-fi 802.11 b/g/n with inbuilt 650 MHz MIPS CPU, Atmel ATSAMR21B18 SOC for supporting Zigbee communications. The Figure 8 contains contain the picture of the Philips HUE bridge.

Figure 8: Philips HUE bridge

The pinout of the HUE Bridge is well documented by Colin O'Flynn [25], and it is rather trivial to gain access to the Linux shell running on the HUE Bridge. His research states, that each HUE Bridge has a unique root password, which is something that most manufacturers would not implement. Furthermore, in his research paper [26] he states that Philips has taken extra steps to overwrite the SDRAM after the update process, to make it harder for anyone to acquire the decryptions keys – "it appears that the bootloader is overwriting memory once done with it" [26]. This indicated that Philips takes security of the HUE devices seriously, and rightfully so, since HUE products are one of the most popular and mature "smart light" products in the market.

6.1 Update Process and Firmware

The firmware can be updated from the mobile application and the Philips HUE Bridge itself makes periodic connections to the update server located at to verify that the firmware is up to date:

```
http://www.ecdinterface.philips.com/DevicePortalICPRequestHandler/RequestHandler.ashx
```

The connection is via the plain text HTTP, however, the contents of the request and responses are encrypted using AES:

```
POST /DevicePortalICPRequestHandler/RequestHandler.ashx HTTP/1.1
Host: www.ecdinterface.philips.com:80
Authorization: CBAuth Type="SSO", Client="***REDACTED***", RequestNr="7",
Nonce="***REDACTED***", SSOToken="***REDACTED***",
Authentication="***REDACTED***"
Content-Length: 768
Content-Type: application/CB-Encrypted; cipher=AES
Connection: close
```

In the case of a new update is available, it is downloaded via the plain text HTTP, however, the contents of the file are encrypted judging by the filename. At the time of writing of this paper, the current firmware was downloaded from:

```
http://fds.dc1.philips.com/firmware/BSB002/1943185030/BSB002_1943185030.product.RSA_prod_01.fw2
```

By analyzing the downloaded firmware file with the Binwalk [27], the author confirmed that the file does not contain any data signatures known to the Binwalk [27]. The consistently high entropy of the contents of the firmware image observed in the Figure 9 also suggests, that the contents are encrypted.

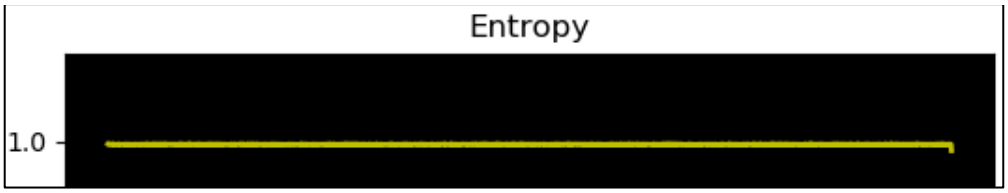

Figure 9: Entropy analysis of the Philips HUE firmware

6.2 Network Communication to the Outside from Philips HUE Bridge

Upon every system startup the Philips HUE Bridge tries to acquire the IP address via the DHCP request. In the case of success, the internet connectivity is verified by looking up Google NTP timeservers and the time is synchronized using the Google and Philips own NTP servers. A multiple HTTP POST request are issued to the:

```
http://diagnostics.meethue.com
```

hosted at the Google cloud, containing what looks like encrypted data. The traffic is sent to the diagnostics host whenever the consumer interacts with the Philips HUE devices, which raises the possibility that the analytics about every consumer action are gathered, however, these are only the speculations of the author at this point. By inspecting the Philips HUE privacy policy, it was confirmed that "Product usage and diagnostic information" are gathered without going into specifics.

The Amazon AWS IP address (34.95.78.50) is hosting the API endpoints for the Philips cloud and is contacted whenever the consumer interacts with his Philips HUE devices, by using TLS 1.2 secured HTTPS connection.

The HUE Bridge announces itself on the LAN by sending out periodic multicast Simple Service Discovery Protocol (SSDP) UDP packets. If the Philips HUE Bridge is exposed to the internet, the SSDP can be used for issuing a DDoS amplification attack [28], by exploiting the fact that UDP is a stateless protocol. By spoofing the IP address of a victim, an attacker can send a SSDP Search request to the HUE Bridge port 1900. In this case, for example, an attacker is spoofing the SSDP UDP packet coming from 192.168.6.6 (victim) to the HUE Bridge located at 192.168.2.18 as can be observed in Figure 10.

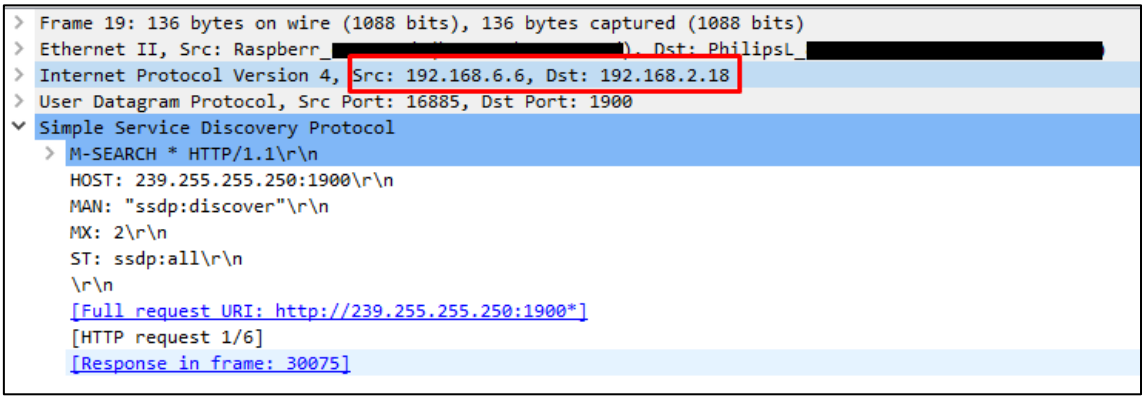

Figure 10: SSDP search request from the spoofed source

In the case the attack is successful, the HUE Bridge will respond with the SSDP message to the victim IP 192.168.6.6. The successful Amplification attack can be observed in the Figure 11:

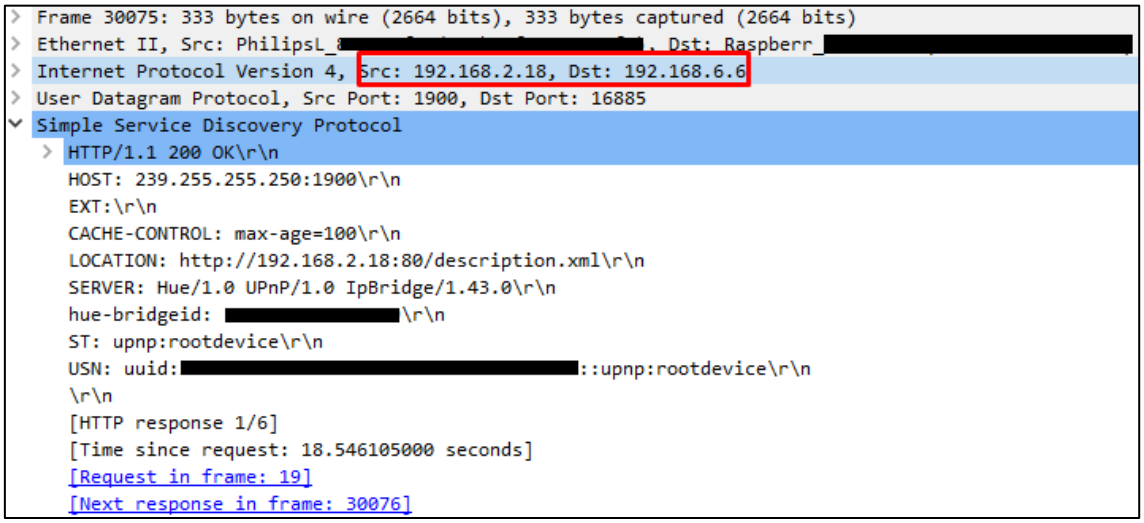

Figure 11:Successful SSDP amplification

The Figure 11 shows the HUE Bridge located at 192.168.2.18 responding with a SSDP packet to the victim 192.168.6.6. By comparing the spoofed UDP packet size which was 136 bytes with the reply to the victim, which was 333 bytes, the rough amplification is 2,4 times. To mitigate the SSDP amplification attack, one should block the UDP port 1900 or exposing the HUE Bridge to the internal network only, since the author cannot come up with cases where the exposure to the internet would be required, which obviously creates more risks for the consumer of the Philips HUE Bridge in any case.

6.3 Philips HUE Mobile Application

The communication with the gateway can either happen directly when both devices are on the same network via an TLS 1.2 encrypted connection or by issuing commands from the mobile application to the cloud, and Philips HUE Bridge then receives the commands from the cloud. The SSL pinning is correctly implemented, the author was not able to connect to the Philips HUE cloud while proxying traffic via the Burp [3] suite, even with the Burp CA installed [4] on the mobile device.

Upon first start of the application, the customer can opt out of sending telemetry data back to the manufacturer, in contrast to HUE Bridge, which will send out a periodic, encrypted streams via plain text HTTP to the:

```
http://diagnostics.meethue.com
```

The following trackers were found to be embedded in both applications which are user to gather analytics data about the consumer. A privacy seeking consumers might find such large presence of trackers disturbing:

- Amplitude
- Braze (formerly Appboy)
- Google CrashLytics
- Google Firebase Analytics
- HockeyApp

To pair a mobile or any other application with the HUE Bridge, a request to the API running on the HUE Bridge is issued. The API endpoints listen to the plain text HTTP and HTTPS as well. The Philips has stated that the support for the plain text API will be dropped eventually, however, for the time being it is still functional to enable backwards compatibility. Both mobile applications, for iOS and for the Android communicated with the HUE Bridge API endpoint (in this case 192.168.2.18) via the HTTPS endpoints apart from the initial connection to the:

```
http://192.168.2.18/api/nouser/config
```

which returns information about the HUE Bridge, containing the software version, bridge MAC address and ID, with some other configuration information:

{"name":"Philips
hue","datastoreversion":"88","swversion":"1935074050","apiversion":"1.35.0","
mac":"***REDACTED***","bridgeid":"***REDACTED***","factorynew":true,"replaces
bridgeid":null,"modelid":"BSB002","starterkitid":""}

To finish the pairing, the consumer must push a button on the Philips HUE Bridge in the thirty second window after the application has initiated the pairing request. Upon successful pairing, the API replies with the randomly generated username, which is then used by the application to issue any direct authenticated commands to the API. All the linked devices can be inspected and removed by the consumer using a non-mandatory account created at the Philips HUE cloud.

Both, iOS and Android applications requests an access to the location data even when running in the background. The justification for that is, to enable to turn on or control lights in general just before the consumer enters his apartment.

The Android application (com.philips.lighting.hue2) version 3.48.2 (7637) was tested.

The iOS application (com.philips.lighting.hue2) was tested with the application version 3.48.0 (11197).

6.3.1 Sensitive Data Stored Locally (iOS)

By inspecting the HUE applications "Info.plist" holding the configuration for the application, the author confirmed, that the App Transport Security (ATS) is enabled, meaning that plain text HTTP connections are allowed from the application. As described in the Apple iOS documentation "Set this key's value to YES to disable App Transport Security (ATS) restrictions for all domains not specified in the NSExceptionDomains dictionary. Domains you specify in that dictionary aren't affected by this key's value" [29], however, reading the documentation a bit further it is stated that "In iOS 10 and later and macOS 10.12 and later, the value of the NSAllowsArbitraryLoads key is ignored — and the default value of NO used instead—if any of the following keys are present in your app's Information Property List file: NSAllowsLocalNetworking" [29]. As the "NSAllowsLocalNetworking" is present in the "Info.plist", the ATS settings are ignored on any modern, up to date iOS device, however, the following snippet shows four domains, to whom plain-text connections are still allowed:

```
[…]
    "NSAppTransportSecurity": {
        "NSAllowsArbitraryLoads": true,
        "NSAllowsLocalNetworking": true,
        "NSExceptionDomains": {
            "huelabs-production.s3-eu-central-1.amazonaws.com": {
                "NSExceptionAllowsInsecureHTTPLoads": true
            },
            "meethue.com": {
                "NSExceptionAllowsInsecureHTTPLoads": true
            },
            "storage.googleapis.com": {
                "NSExceptionAllowsInsecureHTTPLoads": true
            },
            "www2.meethue.com": {
                "NSExceptionAllowsInsecureHTTPLoads": true
[…]
```

During the testing of the iOS Phillips HUE application, no plain text HTTP connections were made by the application.

6.3.2 Sensitive Data Stored Locally (iOS)

There are multiple analytics services embedded in the application. The file stored at:

```
/private/var/mobile/Containers/Data/Application/APP_ID/Library/com.amplitude.database
```

contains the analytics about the user actions inside the application, each record contains iOS version, phone model it runs on and the action performed. The data is stored in the SQLite version 3 formatting.

The various sensitive data, such as the API refresh tokens, HUE Bridge ID as well as unique identifiers used by some of the analytics trackers are stored securely in the keychain following the best practices as can be observed in the Figure 12: Contents of the iOS keyring created by the Philips HUE iOS application.Figure 12:

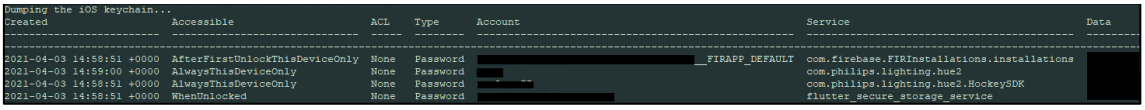

Figure 12: Contents of the iOS keyring created by the Philips HUE iOS application.

The application does not have secure backgrounding feature implemented as can be observed in Figure 13:

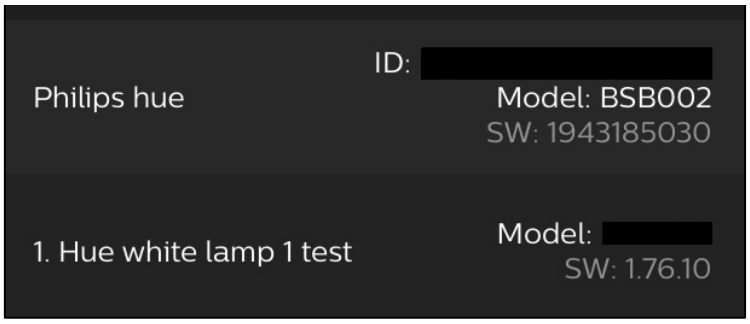

Figure 13: Insecure backgrounding in the Philips HUE iOS application.

This, however, does not pose any real issue in the authors opinion, since there was no sensitive information displayed in the application during the testing.

6.3.3 Application Configuration (Android)

By inspecting the extracted source from the applications APK archive, the author was able to confirm, that the plain text HTTP traffic is allowed to the:

```
storage.googleapis.com
```

To further verify the enabled plain text HTTP connection, the Network Security Configuration file was inspected:

```
res/xml/network_security_config.xml
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">storage.googleapis.com</domain>
    </domain-config>
</network-security-config>
```

An exported Broadcast receiver without any intent filter for one of the Analytics trackers was encountered in the “AndroidManifest.xml”:

```
<receiver android:name="com.appboy.receivers.AppboyActionReceiver"
android:enabled="true" android:exported="true" />
```

It indicates that this Broadcast receiver accepts any messages sent from any application and possibly have an uncontrolled communication.

A rather large number of “Activities” are registered in the “AndroidManifest.xml”, however, all of them have appropriate intent filters in authors opinion.

6.3.4 Extensive Application Permissions (Android)

By analyzing the “AndroidManifest.xml” file, a few requested permissions stood out:

- REQUEST_INSTALL_PACKAGES – Allows application to request an additional package installation.
- READ_EXTERNAL_STORAGE – Read an external data storage.
- WRITE_EXTERNAL_STORAGE – Write an external data storage.

The author reasons that access to the external storage might be required to access to photos which are used to customize light settings.

By analyzing the decompiled source code of the application, one of the trackers – HockeyApp was confirmed for trying to access external phone storage:

```
net/hockeyapp/android/o/c.java
[…]
    public c(Context context, String str, net.hockeyapp.android.m.a aVar) {
        this.f14569a = context;
        this.f14571c = str;
        this.f14573e = new File(context.getExternalFilesDir(null),
"Download");
        this.f14570b = aVar;
        this.f14575g = null;
    }
[…]
```

and create directories on external storage:

```
net/hockeyapp/android/a.java
[…]
    public static File a(Context context) {
        File file = new File(context.getExternalFilesDir(null), "HockeyApp");
        if (!(file.exists() || file.mkdirs())) {
            d.d("Couldn't create HockeyApp Storage dir");
        }
        return file;
    }
[…]
```

6.3.5 Sensitive Data Stored Locally (Android)

A Flutter library is used by the developers to create an UI for the Android application. As one of the features, this library offers a secure storage management by either using Keychain in iOS or in this case using AES to encrypt the sensitive data. The AES key is then encrypted using Android Keystore as can be verified in the code snippet below:

```
d\d\a\c\a.java
[…]
    private void a(Context context) {
        AlgorithmParameterSpec algorithmParameterSpec;
        Log.i("fluttersecurestorage", "Creating keys!");
        Locale locale = Locale.getDefault();
        try {
            a(Locale.ENGLISH);
            Calendar instance = Calendar.getInstance();
            Calendar instance2 = Calendar.getInstance();
            instance2.add(1, 25);
            KeyPairGenerator instance3 = KeyPairGenerator.getInstance("RSA",
"AndroidKeyStore");
            if (Build.VERSION.SDK_INT < 23) {
                KeyPairGeneratorSpec.Builder alias = new
KeyPairGeneratorSpec.Builder(context).setAlias(this.f10557a);
                algorithmParameterSpec = alias.setSubject(new
X500Principal("CN=" +
this.f10557a)).setSerialNumber(BigInteger.valueOf(1)).setStartDate(instance.g
etTime()).setEndDate(instance2.getTime()).build();
            } else {
                KeyGenParameterSpec.Builder builder = new
KeyGenParameterSpec.Builder(this.f10557a, 3);
                algorithmParameterSpec = builder.setCertificateSubject(new
X500Principal("CN=" + this.f10557a)).setDigests("SHA-
256").setBlockModes("ECB").setEncryptionPaddings("PKCS1Padding").setCertifica
teSerialNumber(BigInteger.valueOf(1)).setCertificateNotBefore(instance.getTim
e()).setCertificateNotAfter(instance2.getTime()).build();
            }
[…]
```

The directory storing configuration files for the application located at

```
/data/data/com.philips.lighting.hue2/shared_prefs
```

contains quite a lot of plain text configuration settings for various analytics trackers and application settings, however, most of them did not contain any sensitive data in authors opinion, with the exception of :

```
/data/data/com.philips.lighting.hue2/shared_prefs
/HuePlaySharedPreferenceshue_preferences.xml
```

Which contains UUID for the Philips HUE web portal and MAC address of the HUE Bridge, which could be used to identify a specific consumer.

In the same directory there are also a file containing AES encrypted information:

```
/data/data/com.philips.lighting.hue2/shared_prefs/FlutterSecureStorage.xml
```

And the file holding the RSA encrypted the AES key judging by the name of the file:

```
/data/data/com.philips.lighting.hue2/shared_prefs/FlutterSecureKeyStorage.xml
```

The SQLite3 file located at:

```
/data/data/com.philips.lighting.hue2\databases\bridge_states.sqlite
```

Contains the actual, plain text unencrypted settings of the HUE Bridge, including IP and MAC addresses.

It was also confirmed that the Android application does not have secure backgrounding feature implemented, similar as the iOS application, as can be observe in the Figure 14:

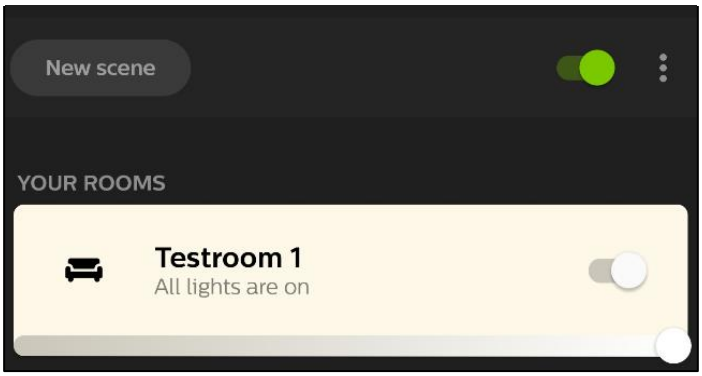

Figure 14: Insecure backgrounding in the Philips HUE Android application

This, however, does not pose any real issue in the authors opinion, since there was no sensitive information displayed in the application during the testing.

6.3.6 TLS/SSL Connection Security (iOS & Android)

The author verified that the SSL Certificate Pinning is used by both mobile applications, as the connection was terminated when trying to intercept the traffic with Burp proxy [3].

6.4 Philips HUE Web Control Panel/API backend

In order to enable the remote control over the HUE devices, the consumer must register an account at:

```
https://account.meethue.com
```

The authentication is handled by using the OAuth at:

```
https://auth.meethue.com
```

Once the authentication has been successful, the JWT Bearer token is used to authorize all further actions by the user. The web panel is minimalistic, allowing the user only to manage his account settings, any HUE Bridges linked to his account and manage the

permissions of any device, that have been authorized to interact with the HUE Bridge. The noteworthy feature is the possibility to enable two-factor authentication, which would enable the usage of one-time passwords. This significantly raises the security of the user account and is a welcome feature.

The web server serving the responses has all the relevant HTTP security headers enabled following all the best security practices, apart from using the “unsafe-inline” directive in the Content Security Policy headers:

```
HTTP/1.1 200 OK
Content-Security-Policy: default-src 'self' auth.meethue.com; connect-src
'self' auth.meethue.com https://emcm6kvdy6.execute-api.eu-west-
1.amazonaws.com/default/eloqua_unsubscribe; frame-src 'self'
auth.meethue.com; img-src 'self' auth.meethue.com data:; object-src 'none';
script-src 'self' auth.meethue.com 'unsafe-inline'; style-src 'self'
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Download-Options: noopen
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
[…]
```

This would allow a potential attacker to execute the inline JavaScript code, if he would be able to find a vulnerability in the web panel, which accepts an inline code. While this is not an immediate vulnerability, it is generally understanding, that using “unsafe-inline” directive disables most of the protection a Content Security Policy is meant to enforce, and thus it is only recommended to be used on the testing environment or together with the “strict-dynamic” directive, which will disable the ability for the JavaScript to insert script elements provided by the attacker.

7 Ikea TRÅDFRI

The TRÅDFRI product line from Ikea provides remotely controllable lights, motion detectors, smart plugs, and power relays. Some of them can be controlled by the consumer via the hardware remote control, but some are only working via the connection to the TRÅDFRI gateway, which is connected to the internet via the ethernet. However, customers cannot control their devices remotely by mobile app unless they are connected to the same local network the gateway is connected to. A communication between mobile application and the gateway is established using the Constrained Application Protocol (CoAP) [30]. The CoAP messages for now are encrypted using DTLS v1.2 [31] and are sent via the UDP protocol between client and the gateway, however, judging by the changelog [32], the DTLS was not supported in the early version. Since TRÅDFRI home automation solution is compatible with the Amazon Alexa, Apple HomeKit and Google Assistant among others, it is possible to use these solutions instead of an original TRÅDFRI gateway to enable the control of the devices from remote location over the internet. The communication between the gateway and the devices is done via the Zigbee Light Link protocol using 2.4 Ghz radio frequency. An interesting side feature of the TRÅDFRI gateway is the ability to pair with the Philips HUE lightbulbs as well and enabling the control via the bulbs from the Ikea mobile application.

The Figure 15 contains the picture of the Ikea TRÅDFRI gateway:

Figure 15: Ikea TRÅDFRI gateway

The PCB board of the gateway has a Silicon Labs EFR32MG12 Mighty Gecko Wireless SoC, BCM5241 Single-Port MII Copper/Fiber Fast Ethernet Transceiver, Type 1GC MyRata 2.4GHz & 5GHz Wi-Fi module and IS25CQ032 32Mbit Single Operating Voltage Serial Flash Memory. The author also observed, what looked like pins meant for debugging at both sides of the PCB, but during the time limit chose not to investigate them further.

Figure 16 represents a schematic describing how the mobile devices, the TRÅDFRI gateway and various smart sensors are communication in between each other.

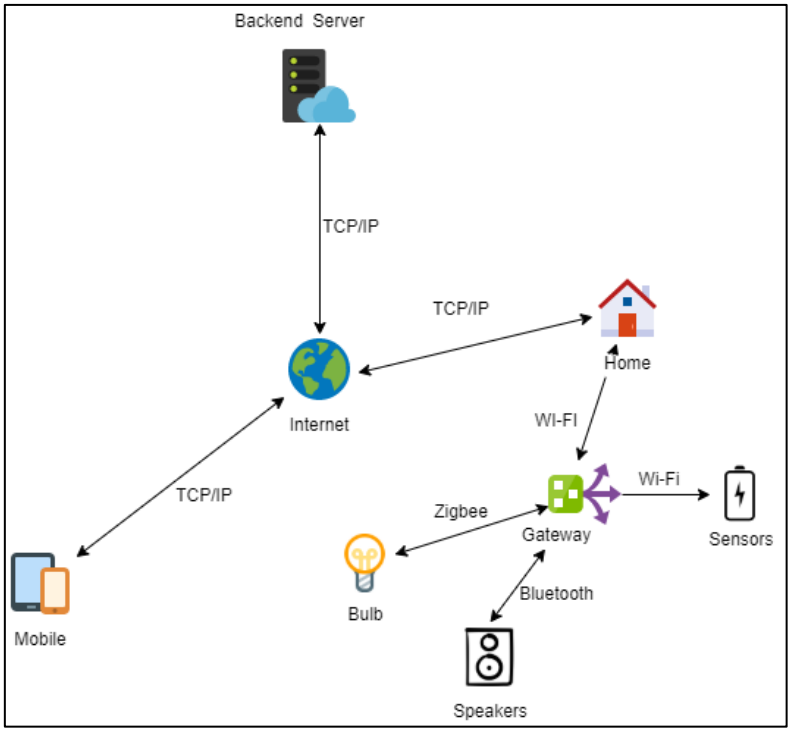

Figure 16: Communication between Ikea TRADFRI devices and the consumer

7.1 Update Process and Firmware

Upon every restart the TRÅDFRI gateway checks back home for an update, via the plain HTTP, requesting the following URL:

```
http://fw.ota.homesmart.ikea.net/feed/version_info.json
```

Which contains the metadata about the latest firmware versions for all IKEA products in the JSON format. For example, the firmware for the TRÅDFRI gateway was described by the following JSON string:

```
{"fw_binary_url":"http://fw.ota.homesmart.ikea.net/global/GW1.0/01.13.025/bin
/10032198-2.2-TRADFRI-gateway-
1.13.25.p.elf.sig.ota.signed","fw_filesize":838112,"fw_hotfix_version":25,"fw
_major_version":1,"fw_minor_version":13,"fw_req_hotfix_version":26,"fw_req_ma
jor_version":9,"fw_req_minor_version":9,"fw_type":0,"fw_update_prio":5,"fw_we
blink_relnote":"https://ww8.ikea.com/ikeahomesmart/releasenotes/releasenotes.
html"}]
```

The request from the gateway always includes a unique UUID v4 in the User-Agent string which uniquely identifies each gateway device:

```
GET /feed/version_info.json HTTP/1.0
User-Agent: HertzClient/1.0 (GW (1).(8).(25); Id 715cfcd3-7497-4aac-9a7a-
4ddc4b978024)
Host: fw.ota.homesmart.ikea.net
Connection: close
```

By analyzing the downloaded binary firmware file, the URL containing, what looks like the beta testing firmware was discovered:

```
http://fw.test.ota.homesmart.ikea.net/feed/version_info.json
```

Some of the firmware files, for example, "10005777-6.1-TRADFRI-control-outlet-2.0.024.ota.ota.signed" were formatted as Over-the-Air (OTA) [33] image which supports the signing and verification of the image. However, the firmware for the TRÅDFRI hub itself was not formatted as Over-the-Air (OTA) [33], judging by the fact that the author was not able to locate any parts of the OTA header or magic bytes "0BEEF11E" indicating the start of the OTA header. By using Binwalk [27] the author was able to confirm that the firmware image for the hub is not fully encrypted and contains a few certificates and a 32bit ARM ELF (Executable and Linkable Format) executable. There does not exists a requirement for the manufacturer to always encrypt the firmware, if the manufacturer decides that there are no sensitive information contained in the firmware, such as hardcoded passwords, access tokens. Signing of the firmware is considered a standard in the industry, otherwise a malicious actor would be able to intercept the update requests coming from the consumer devices and inject a malicious, modified version of the firmware in the response. The author wants to add that no claims from IKEA about the firmware encryption are made up to the time of writing this paper. By using a dd command, the author was able to extract the executable from the firmware image, since the start of it was identified by the ELF header starting with bytes "7F454C46". By inspecting the extracted binary, the author was able to verify, that the binary contains the binary code providing the ability to interact with the IKEA devices

via the CoAP protocol and compatibility with the devices issued by other manufacturers, for example, the Sonos wireless speakers.

7.2 Network Communication to the Outside from Gateway

The gateway synchronizes the local system time by querying one of four hardcoded NTP servers. The periodic TLS 1.2 secured connection to the following URL was observed:

```
https://webhook.logentries.com
```

The author used fakedns.py [22] to intercept and forge all requested DNS entries and redirect the request to the website with the self-signed TLS certificate, however, the TRÅDFRI gateway correctly verified the invalidity of the certificate and interrupted the connection.

As stated, on the LogEntries website, it provides “Live Log Management and Analytics”. While this might not impose the immediate concerns for those worried about their privacy, the fact that analytics and log data are sent to a centralized server, and the inability to inspect the data sent or easily opt-out from sending the data might raise a few eyebrows. More tech-savvy consumers can just block the DNS requests or any connection to the webhook.logentries.com if they choose so and opt-out of sending the data to the third-party logging service.

7.3 Mobile IKEA Home smart (TRÅDFRI) application

Every time, when starting the mobile application, the DNS requests looking for the following domains:

- app.config.homesmart.ikea.net
- meta.config.homesmart.ikea.net
- privacypolicy.config.homesmart.ikea.net
- supportdetails.config.homesmart.ikea.net

While it is not entirely clear why such requests are made, the authors guess is that these requests bootstrap some sort of periodically updated information to be displayed by the

mobile application, for example, the privacy policy, which probably have been updated more than once.

To add and communicate to a TRÅDFRI gateway, a unique Pre-Shared Secret [34] consisting of gateways MAC address and sixteen symbols long string containing alphanumeric values. Both of those are printed on the base of every device. The iOS (com.ikea.tradfri.lighting) and Android mobile applications (com.ikea.tradfri.lighting) in version 1.13.0 were used for testing.

7.3.1 Sensitive Data Stored Locally (iOS)

The files inside the:

```
/private/var/mobile/Containers/Data/Application/APP_ID/Library/Caches/logentr
ies
```

contains what looks like the information that might be sent back to the centralized analytics and log management service at logentries.com. A sample entry shows that the debug information is stored, and potentially sent to the logentries.com.

```
2021-0x-xx xx:xx:xx.xx TRADFRI[6557:985291] Unbalanced calls to begin/end
appearance transitions for <TRADFRI.TRBaseNavigationViewController:
0x106822e00>.
```

The gateways MAC address, which is part of the Pre-Shared Secret [34] used to authenticate the connection from the mobile application to the gateway is correctly stored in the iOS Keychain with the "WhenUnlockedThisDeviceOnly" attribute, forbidding the backing up of the data and only allowing the usage of it, while the device is unlocked, as can be observed in the Figure 17.

```
Dumping the iOS keychain...
Created                  Accessible                  ACL    Type      Account           Service    Data
-----------------------  --------------------------  -----  --------  ----------------  ---------  ------
2021-04-11 14:42:51 +0000  WhenUnlockedThisDeviceOnly  None   Password  [redacted]
com.ikea.tradfri.lighting on (iPhone: 14.4.1) [usb] # quit
```

Figure 17: "Home Smart" iOS keychain storing MAC address.

The file located at:

```
/private/var/mobile/Containers/Data/Application/APP_UUID/Library/Preferences/
com.ikea.tradfri.lighting.plist
```

contains various application settings, some of them might be a bit sensitive in authors opinion, such as the TRÅDFRI gateway IP address and port number, unique TRÅDFRI gateway UUID and the IP address of the Apple HomeKit gateway if there is one paired. However, none of the information is encrypted in contrast to what was encountered in the Android version of the application.

The insecure application backgrounding issue was found on the iOS application as can be observed in Figure 18. For iOS devices the taken screenshots can be found at the following directory:

```
/private/var/mobile/Containers/Data/Application/APP_UUID/Library/SplashBoard/
Snapshots/sceneID:com.ikea.tradfri.lighting-default/
```

As with all the other applications, this is a low-risk issue in authors opinion, since the serial number can be entered by scanning the QR code on the back of the device and is more convenient for customers to use.

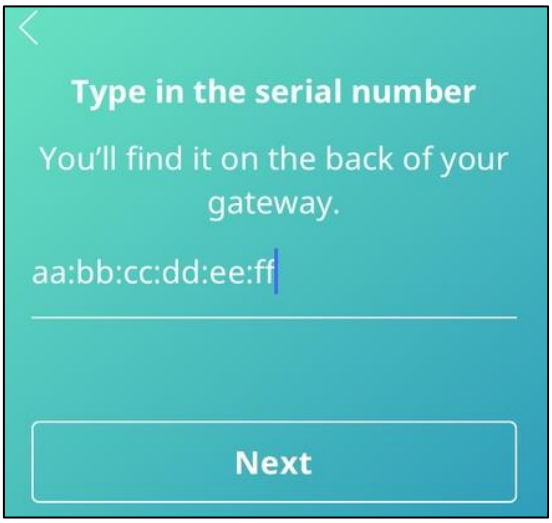

Figure 18: Insecure backgrounding in "Home Smart" iOS application

7.3.2 Sensitive Data Stored Locally (Android)

The XML file found at:

```
/data/data/com.ikea.tradfri.lighting/shared_prefs/com.ikea.tradfri.lighting_p
references.xml
```

Contains various sensitive data, such as Wi-Fi SSID, Pre-Shared Secret [34], gateway IP among other things in encrypted format. The storage of the gateway IP in encrypted form is a bit strange in authors opinion, since in the same file "CoapServer" variable can be found, where the endpoint of CoAP running on the gateway, and the port it listens to is saved in the plain text. By decompiling the APK and inspecting the Java code it was confirmed that the Android KeyStore is used to generate and store the key used for encryption of the sensitive data:

```
[…]
   public final void a() {
        boolean z2;
        try {
            if (!this.d.containsAlias("Tradfri")) {
                Locale locale = Locale.getDefault();
                String str = this.b;
                g.h(str, "createSecureKeys localeBeforeEnglishLocale " +
locale);
                if (!locale.equals(Locale.US)) {
                    d(Locale.US);
                    z2 = false;
                } else {
                    z2 = true;
                }
                Calendar instance = Calendar.getInstance();
                Calendar instance2 = Calendar.getInstance();
                instance2.add(1, 1);
                KeyPairGeneratorSpec build = new
KeyPairGeneratorSpec.Builder(this.a).setAlias("Tradfri").setSubject(new
X500Principal("CN=Sample Name, O=Android
Authority")).setSerialNumber(BigInteger.ONE).setStartDate(instance.getTime())
.setEndDate(instance2.getTime()>
                KeyPairGenerator instance3 =
KeyPairGenerator.getInstance("RSA", "AndroidKeyStore");
                instance3.initialize(build);
                instance3.generateKeyPair();
                if (!z2) {
                    d(locale);
                }
            }
[…]
```

The insecure backgrounding issue is prevalent in this version of Android application as well, as can be observed in Figure 19 showing the screenshot taken by the mobile phone during the minimization of the application.

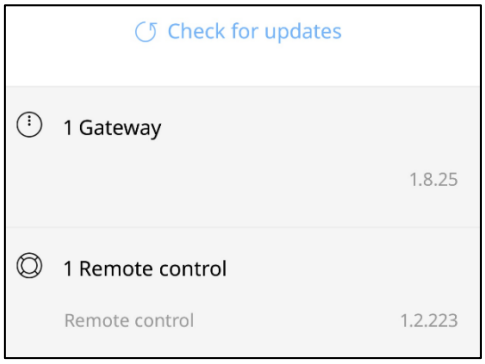

Figure 19: Insecure backgrounding in “Home Smart” Android application

7.3.3 Extensive Permissions (Android)

By inspecting the Android application manifest, the author spotted rather strange permission requirement:

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

Which indicates that the application wants to have a permission to make a call. While inspecting the application itself, the author made a guess, that this permission is used so that the customer can call the support from the "Need help?" menu within the application.

7.3.4 TLS/SSL Connection security (iOS & Android)

The official mobile application does not interact with the backend manufacturer backend servers via TLS/SSL connections. The local traffic between mobile application and the gateway is always encrypted using DTLS v1.2 [31].

8 Conclusion

The author feels that consumers are put in the tough spot at this point of time. In one hand various smart house technologies enables consumers to control various aspects of home automation, even remotely, on the other hand some of the automation solutions tracks almost every consumer action, and while the tracking is officially marketed as needed to improve the product, it still leaves a sour taste in the mouth for those who value and guard their privacy.

Some solutions, namely “Istabai” lack any remote update mechanism, which means that once a vulnerability is found in the actual hardware, there is no way for a consumer to upgrade, and thus the devices are left vulnerable. This claim is not fully confirmed, since the author was only able to monitor the devices he had in his possessions for a limited amount of time, however, in this time, he observed no update checks issued by any of “Istabai” devices, also there is no hardware update option in the mobile application, as with the other smart home solutions analyzed by the author. There is, however, a microUSB port, which is meant for the firmware updates judging by the documentation provided with the devices.

Another criticism the author has for “Istabai” devices is the fact that once consumer has acquired the devices, he is tied to the one vendor forever in the case the vendor has chosen to rebrand completely, or tied to “Istabai”, if the vendor has chosen to rebrand only the mobile application. This is due to the fact that “Istabai” has chosen to use proprietary, non-standard protocol.

Philips HUE is one of the undisputed leaders in the “smart house/lights” market, and thus has gathered a huge user base. The mobile application feels and works very smooth, however, the presence of a bunch of various analytics trackers can scare away so called “privacy nuts”. The update mechanism is present and well working, as the author had to update the HUE Switch and lightbulbs with the update patching the latest vulnerability in the HUE hardware. The usage of open industry standards enables the consumers to use a different means for the control of the HUE devices, such as Apple HomeKit, Amazon

Alexa, Google Assistant, or choose some open-source solution as long as it supports Zigbee protocol or can talk with the HUE API, which is well documented.

Ikea TRÅDFRI has taken a different approach and tries to be as least intrusive as possible in the terms of privacy. There exists no possibility for the consumer to control his TRÅDFRI devices remotely, unless he uses a third-party solution, which is possible, since TRÅDFRI use open standard, thus the consumer is not tied to use the IKEA branded mobile application if he chooses so. The update mechanism is present and works flawlessly as tested by the author. The minor concern for some of the consumers might be the fact that the analytics logs are gathered in a centralized solution by the manufacturer.

The authors conclusion and recommendation for someone looking for smart home solution would be to look for the solution using open standards, and tie them all up in Apple HomeKit, Amazon Alexa or Google Assistant. For the privacy concerned consumers, the author would recommend blocking the DNS lookups for the analytics trackers or block the outgoing connections altogether.
For more tech savvy, and privacy valuing consumers, the author would suggest using some of the opensource home automation frameworks, which are supported by all the big manufacturers, but does not try to track every action done by the consumer. This would also enable the consumer to assemble his own, isolated smart home control solution. However, this would require the technical understanding and ability to modify, run and configure various services manually.

References

[1] J. Malinen, "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator," 2019. [Online]. Available: https://w1.fi/hostapd/.

[2] Genymotion, "Genymotion Android Emulator," [Online]. Available: https://www.genymotion.com/.

[3] PortSwigger, "Burp Suite," [Online]. Available: https://portswigger.net/burp.

[4] Portswigger, "Installing Burp's CA Certificate in an iOS Device," [Online]. Available: https://portswigger.net/support/installing-burp-suites-ca-certificate-in-an-ios-device. [Accessed 20 March 2021].

[5] OWASP, "OWASP Mobile Top 10," [Online]. Available: https://owasp.org/www-project-mobile-top-10/. [Accessed 01 March 2020].

[6] Android.com, "Intents and Intent Filters," 27 January 2021. [Online]. Available: https://developer.android.com/guide/components/intents-filters. [Accessed 14 March 2020].

[7] Android.com, "Developer Guides," Google, 08 06 2020. [Online]. Available: https://developer.android.com/training/articles/keystore. [Accessed 23 02 2021].

[8] Sophos, "iOS 14 flags TikTok, 53 other apps spying on iPhone clipboards," 30 June 2020. [Online]. Available: https://nakedsecurity.sophos.com/2020/06/30/ios-14-flags-tiktok-53-other-apps-spying-on-iphone-clipboards/. [Accessed 25 February 2021].

[9] "HTTP headers," Mozilla, 20 March 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers. [Accessed 25 March 2021].

[10] "HTTP request methods," Mozilla, 04 December 2020. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods. [Accessed 35 December 2021].

[11] "Using HTTP cookies," Mozilla, 17 March 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#restrict_access_to_cookies. [Accessed 19 March 2021].

[12] "cookies, SameSite," Mozilla, 17 March 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite. [Accessed 19 March 2021].

[13] J. Kettle, "Web Storage: the lesser evil for session tokens," PortSwigger, 01 February 2021. [Online]. Available: https://portswigger.net/research/web-storage-the-lesser-evil-for-session-tokens. [Accessed 19 March 2021].

[14] M. Zalewski, "Content Isolation Logic," in *The tangled Web: a guide to securing modern Web applications*, No Starch Press, 2011, p. 320.

[15] M. P. Dafydd Stuttard, The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws, Second Edition, Indianapolis: John Wiley & Sons, Inc, 2011.

[16] Draugiem Group, "Istabai - Comfort at your fingertips," [Online]. Available: https://istabai.com/.

[17] Elektrum, "Elektrum Viedā māja," [Online]. Available: https://elektrumviedamaja.lv/. [Accessed 18 April 2021].

[18] The Tcpdump Group, "Tcpdump," [Online]. Available: https://www.tcpdump.org/.

[19] WebKit, "WebKit," [Online]. Available: https://webkit.org/.

[20] Google, "LevelDB," [Online]. Available: https://github.com/richmonkey/leveldb-android. [Accessed 03 February 2020].

[21] StatsCounter, "Android Version Market Share Europe," [Online]. Available: https://gs.statcounter.com/android-version-market-share/all/europe. [Accessed 21 03 2021].

[22] P. Hes, "Fake DNS server written in python 3," [Online]. Available: https://github.com/mikust/fakedns. [Accessed 23 January 2021].

[23] A. Dergachev, "simple-https-server.py," [Online]. Available: https://gist.github.com/mikust/6ea669f7048c764e763bfc0599ecbbda. [Accessed 2021 January 23].

[24] S. Margaritelli, "Swiss Army knife for WiFi, Bluetooth Low Energy, wireless HID hijacking and Ethernet networks reconnaissance and MITM attacks," [Online]. Available: https://www.bettercap.org/.

[25] C. O'Flynn, "Getting Root on Philips Hue Bridge 2.0," 2016. [Online]. Available: https://colinoflynn.com/2016/07/getting-root-on-philips-hue-bridge-2-0/.

[26] C. O’Flynn, "A LIGHTBULB WORM? Details of the Philips Hue Smart Lighting Design," 01 August 2016. [Online]. Available: http://colinoflynn.com/wp-content/uploads/2016/08/us-16-OFlynn-A-Lightbulb-Worm-wp.pdf.

[27] ReFirm Labs, "Binwalk," [Online]. Available: https://github.com/ReFirmLabs/binwalk.

[28] Akamai, "SSDP REFLECTION DDOS ATTACK," 2014. [Online]. Available: https://www.akamai.com/fr/fr/multimedia/documents/state-of-the-internet/ssdp-reflection-ddos-attacks-threat-advisory.pdf.

[29] Apple Inc, "NSAllowsArbitraryLoads," Apple, 2021. [Online]. Available: https://developer.apple.com/documentation/bundleresources/information_property_list/nsapptransportsecurity/nsallowsarbitraryloads.

[30] Internet Engineering Task Force, "The Constrained Application Protocol (CoAP)," 2014. [Online]. Available: https://tools.ietf.org/html/rfc7252.

[31] Internet Engineering Task Force, "Datagram Transport Layer Security Version 1.2," Internet Engineering Task Force, 2012.

[32] IKEA, "RELEASE NOTES," IKEA, [Online]. Available: https://ww8.ikea.com/ikeahomesmart/releasenotes/releasenotes.html. [Accessed 02 April 2021].

[33] Zigbee Alliance, "ZigBee Over-the-Air Upgrading Cluster," San Ramon, 2010.

[34] E. H. Tschofenig, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things," Internet Engineering Task Force (IETF), 2016. [Online]. Available: https://tools.ietf.org/html/rfc7925#section-4.2.

[35] O. O. A. L. O. K. Olaide O. Kazeem, "Comparative Study of Communication Interfaces for Sensors and Actuators in the Cloud of Internet of Things," *International Journal of Internet of Things*, pp. 9-13, 01 June 2017.

I.Appendix: License

Non-exclusive licence to reproduce thesis and make thesis public.

I, Mikus Teivens

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, **ANALYSIS OF SECURITY AND PRIVACY ISSUES IN COMMON SMART HOME PRODUCTS**
supervised by Arnis Paršovs

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Mikus Teivens
14/05/2021