MASTER'S THESIS

Sending QR-codes as a replacement for scanning

Dijkhof, R.

Award date:
2021

SENDING QR-CODES AS A REPLACEMENT FOR SCANNING

by

Robin Dijkhof

in partial fulfillment of the requirements for the degree of

Master of Science
in Software Engineering

at the Open University, Faculty of Science
Master Software Engineering
to be defended publicly on 07-09-2021 at 15:30.

Student number:

Course code:	IM9906	
Thesis committee:	Dr. Fabian van den Broek,	Open University
	Dr. Greg Alpár,	Open University

Open Universiteit
de beste! www.ou.nl

SENDING QR-CODES AS A REPLACEMENT FOR SCANNING

in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering (IM9906).

Thesis committee: Dr. Fabian van den Broek, Open University and Dr. Greg Alpár, Open University

Faculty of Science
Open University of the Netherlands
Graduation Assignment

CONTENTS

1

SUMMARY

In this research, a QR-code sending system is created to omit the scanning of a QR-code with a mobile device. One such app that uses QR-codes is IRMA. The IRMA app is an app that stores personal attributes. Examples of such attributes are name, address, date of birth and email address. These attributes are only stored on the mobile device and are never stored on a server. This is to provide privacy and security.

Attributes can be used on various websites. For example, you can use your name and date of birth attributes to provide your identity. There are two different flows to provide the attributes to websites; a mobile flow and a desktop flow. With the mobile flow, the user browses to a page on their mobile device. A *login with IRMA* button is present on which the user clicks. The IRMA app opens and the user can provide their attributes to the web page. With the desktop flow, the user browses to a page on their desktop. Upon clicking on the *login with IRMA* button, a QR-code is presented. This QR-code should then be scanned with the IRMA app. After scanning the QR-code, the user can provide their attributes just like with the mobile flow. These flows are common and also occur in other apps such as *De Rabobank* app and the *DigiD* app.

The desktop flow has a few disadvantages. The first disadvantage is that it can be slow. Being a bit slow is not a big problem for a system that you would use one or two times a day, but would quickly become annoying when you had to use it every time you want to log in. A second disadvantage is that the desktop flow cannot be used by everyone. Some people are simply unable to use their phone to scan a QR-code due to a disability. People who are blind for example must know where on the screen the QR-code is present and cannot see whether a QR-code is obstructed or not. Some wheelchair users have their phone fixed in a phone holder and also cannot easily scan a QR-code.

To create the QR-code sending system, this research looks at how IRMA uses QR-codes. Next, what are the required components to build the system. Next, what is the impact on privacy and security. Last, an effort is made to generalize the solution. The final solution uses push notifications to send the QR-code to the mobile device. The system consists of a browser plugin, trusted server, push notification service and a mobile app. The system is built without affecting the state of security and privacy of the IRMA app. The system is built in a generic way so it supports not only IRMA QR-codes but also any QR-code that contains a URI. That means developers can allow their users to use the QR-code sending system by simply placing a universal link in their QR-code.

2

INTRODUCTION

The IRMA app is an app that allows a user to share personal attributes in a secure and privacy friendly way. Examples of such attributes are name, address, date of birth and email address. Attributes can be used for different purposes. These attributes are only stored on the mobile device and are never stored on a server. This is to provide privacy and security.

A user can provide their attributes to various websites. For example, you can use your name and date of birth attributes to provide your identity. Other examples are to prove you are above a certain age without providing your date of birth or to authenticate yourself.

In contrast to other methods of providing information, such as an online form, with IRMA it is possible guarantee to correctness of an attribute. An attribute is digitally signed by the provider of this attribute. The website requesting your information can verify the authenticity of the provided attributes with the digital signature and the attribute provider its public key.

Another way in which IRMA works privacy-friendly is that your attributes are directly sent to the website requesting them. This in contrast to other methods such as Sign in with Facebook or Sign in with Google. Using the latter methods, you first sign in to Facebook or Google. Google or Facebook then tellR the website who you are. Google and Facebook know exactly when and where you are signing in to.

Providing your attributes with IRMA is rather simple. There are two scenarios. In the first scenario, the user is on their phone. The user browses a website where there is support for IRMA login. When the user presses the login button, the IRMA app opens. After unlocking the app by entering the PIN and agreeing to disclose the attributes, the user is returned to the website and is now logged in.

In the second scenario, the user uses a non mobile device. A desktop or laptop for example. When the user presses the login button, a QR-code is displayed. The user should now take their phone, probably unlock it and open the IRMA app. After unlocking the IRMA app, the user presses the button to scan a QR-code; scans the QR-code on the website and agrees to disclose the attribute. From this point on, the flow is equal to the first scenario. Scanning the QR-code is necessary to set up a connection between the IRMA app and the IRMA server. In the first scenario the connection can be setup directly since the user is already browsing on their mobile device.

Though the steps in the second scenario are simple, they can be tedious. Authentication methods that are too complex or take too much time, discourage users from using them.

Steven Furnell states users do not always have an authentication method enabled on their devices. The reason is that it is inconvenient to unlock the device every time. Especially when users only want to do something small such as checking an appointment or sending a message[Furnell, 2016]. It can be concluded that user experience is important. The way IRMA currently works could lead to a worse user experience, because it takes many steps, which could hinder the adoption of IRMA. Steven Furnell concludes that usability has started to receive more attention than before. According to Steven Furnell, developers understand that for security to be effective, it must not only be present but also used.

It is also worth noting that some people are simply unable to use their phone to scan a QR-code due to a disability. People who are blind, for example, must know where on the screen the QR-code is present and cannot see whether a QR-code is obstructed or not. Some wheelchair users have their phone fixed in a phone holder and also cannot easily scan a QR-code. Also, it is not possible to scan a QR-code with a broken camera, whether it is a hardware or software problem.

Ideally, from a user perspective, the user should be able to provide their attributes with as few steps as possible. Though, in some cases it might be desirable to have a user perform an explicit action. Instead of using the phone to scan the QR-code, it might be possible to create a browser plugin that scans the QR-code and sends its content to the IRMA app on a mobile phone. Instead of picking up the phone; unlocking it; opening the IRMA app; pressing the button to scan a QR-code and scanning the QR-code, the user unlocks their phone and the IRMA app is already open. This removes several steps, which improves usability. Yet, security cannot be forgotten. The solution must not be at the expense of security and privacy. For Example, when a user leaves their computer and or mobile phone unattended, it should not be possible for a malicious user to abuse one of its attributes. In this research, a solution will be created to omit the QR-code scanning step when using IRMA on a separate computer.

3

RESEARCH QUESTIONS AND METHOD

3.1. RESEARCH QUESTIONS

The goal of the research is to create a browser extension for the IRMA app. Because the IRMA app focuses on privacy and security, so should the browser extension. Ignoring the importance of privacy and security would render the plugin useless. Therefore, the main research question that should be answered is: *In what way can an IRMA browser extension, which replaces the QR-code scanning step, be realised without affecting the state of security and privacy of the app?*

The system should in some way be able to set up communication between the browser and mobile device. By looking at other systems we can learn how they facilitate communication or synchronisation between a browser and mobile device. The first research question therefore is: *What techniques do other systems use to facilitate communication or synchronisation between a browser and mobile device?* This section is also the related work section.

To answer the main research question, we should know how the IRMA app works. The second research question(RQ2) is: *How does the process of connecting an IRMA app to an IRMA server, using QR codes, work?* This information is important because it will tell us what information should be sent from the browser extension to the app. It is not necessary to fully understand every part of the IRMA app. For example, it is not necessary to know what kind of encryption is used, or how the app communicates with the server. However, it is necessary to understand how to QR-code part works and what data is sent to the app.

All systems consist of one or more components. In our case, some are obvious such as a browser plugin, a phone and a mobile app. Other components are less obvious. We might need a server, a push notification service, an email server or Bluetooth. A component does not necessarily need to be physical. Components can also be software libraries. For example, a library might be required to perform encryption. The third research question(RQ3) therefore is: *What are the required components for realising the IRMA browser plugin?* A set of components should be able to provide communication between the browser plugin and the mobile phone. It should also be able to link and unlink a specific mobile device to a specific plugin. There is not a single solution to this answer. There are multiple solutions of which each solution has its benefits and downsides. In order to pick a solution, it should be clear what the advantages and disadvantages of each solution are.

The fourth research question(RQ4) is: *What is the impact on privacy and security?* To assess the impact on privacy and security, we should be aware of the information flows. Different solutions require different components and therefore, different information flows. For example, before sending push notifications, a user needs to register with their unique ID.

The last research question(RQ5) is *How can the QR-sending system for IRMA be extended to also work for other QR-codes?* It might be possible to build the system in such a way, it is easy to support other QR-codes as well.

3.2. Research method

For RQ1, we can look into apps and browser plugins that have somewhat the same functionality as the IRMA app or as an IRMA browser plugin should have. We can search in app and plugin stores for the same functionality. We can also use existing literature about the subject.

IRMA has some great documentation, both functional and technical. This should probably give an answer to RQ2. Since IRMA is open source, we could simply analyse the source code if the documentation appears to be insufficient.

For RQ3, we can look at RQ1. This section will contain some examples of browser plugins communicating with mobile phones in some way.

To find an answer to RQ4, a component diagram and a data flow diagram(DFD) will be created. Using these, a security analysis will be carried out using the STRIDE method[Microsoft, 2020]. STRIDE is a method to identify security threats. It is used to help reason and find threats related to Spoofing, Tampering, Repudiation, Information disclosure Denial of Service and Elevation of privileges. Therefore the name STRIDE. The software will be created using an agile method with several increments. The component diagram, DFD and the security analysis will be created or updated after each increment is completed. Components and data flows can change for each deliverable so the answer to RQ4 could change as well. Analysing the components and data flow after each deliverable is completed is to catch (design) errors in an early phase.

The last research question, RQ5, can be answered by looking into how other apps use QR-codes and what is in the QR-codes. Examples of these apps are the DigiD and the Rabobank app as stated in the results of RQ1.

With these five research questions, we can give an answer to the main research question.

3.3. Validation

First of all, there should be a plugin that works, that means it should be functionally correct. This can easily be tested by testing one of the demos on the IRMA web page.

With the help of the component diagram, DFD and the security analysis, which are created or updated after completing each deliverable, it is tried not to affect the state of security and privacy. The chances are some issues will be overlooked, no threat modelling techniques can guarantee all the possible attacks will be found. Therefore, the solution, DFD diagram and analysis will be discussed with fellow students.

4

RESULTS

4.1. WHAT TECHNIQUES DO OTHER SYSTEMS USE TO FACILITATE COMMUNICATION OR SYNCHRONISATION BETWEEN A BROWSER AND MOBILE DEVICE?

By looking at other systems we can learn how they facilitate communication or synchronisation between a browser and mobile device. We chose to look into apps and browser plugins that have somewhat the same functionality as the IRMA app or as an IRMA browser plugin should have. The mention of apps that are similar to IRMA is not exhaustive. There are probably more apps that work in a similar way. The mentioned browser plugins are found on the Chrome extension store. This section is also the related work section.

4.1.1. TIQR

Rijswijk et al. state one-factor authentication such as username password is not safe enough. They also state that current two-factor authentication methods are not ideal. Current two-factor authentication methods are hardware dependent, software dependent, not secure enough, too expensive, not open, not easy to use, or a combination of factors.[van Rijswijk and van Dijk, 2011]

Rijswijk et al. provide a new solution called *Tiqr*. Tiqr is based on a mobile app that is open source and does not have the disadvantages stated above. Tiqr and IRMA are very similar in behaviour. IRMA also has non of the disadvantages above. When browsing to a page that requires a login, the user sees a Tiqr QR-code. Just as with IRMA there are two scenarios. When the user is on their laptop or desktop the user opens the Tiqr app on their mobile phone and scans the QR-code. When the user visits a page on the mobile phone, they simply click on the QR-code. After scanning or clicking on the QR-code, the user selects an identity and confirms it wants to log in to the website using that identity. After confirming the identity, the user enters their PIN code or uses the fingerprint sensor. Last, a confirmation is shown in the app and on the website and the user is redirected to the protected content. When there is no internet connection, the app shows a one-time password as a fall-back. It is unclear what the consequences are of losing the phone and what a user should do in that case.

Yet, there are also differences between IRMA and Tiqr. With IRMA you can provide your attributes which are digitally signed by the provider. With Tiqr you can only log in to web

pages, it is used to perform challenge/response authentication. Tiqr also requires you to set up an account for each website.

4.1.2. DigiD App

DigiD is an authentication method that can be used by Dutch citizens to authenticate themselves to governmental organisations or organisations that are carrying out government tasks [DigiD, 2020]. One of the DigiD methods is using the app. The user logins into the app only once. After that, the user can use the DigiD app in almost the same way as the IRMA app. On desktops and laptops, a QR-code is shown which can be scanned with the app. This QR-code is not immediately presented. The user first has to enter a linking code that is presented by the app. On mobile, the app pops up. Before confirming, the user also has to provide a PIN code.

4.1.3. Rabobank app

The Rabobank app is a mobile banking app. It also uses QR-codes. When clicking on the payment button on a website on a mobile device, the app is opened. When on a non-mobile device, a QR-code is presented. This QR-code can then be scanned with the app. Before opening the app and after scanning the QR-code, the user should enter the correct PIN code. It is worth noting QR-code cannot always be used. On some websites the option to pay with a QR-code is simply not present.

4.1.4. Okta Verify Push and Google Prompt

Okta Verify Push[Okta, 2020] and Google Prompt[Google, 2020] allow two-factor authentication by sending a push notification to a mobile device app. First, the user signs in using a username and a password. When the entered credentials are correct, a push notification is sent to the app. The user can now click *Accept* when it recognises the login attempt or *Decline* when it does not recognise the attempt. Both solutions are closed source and do not have a fallback method. When the phone is lost, users can simply choose another 2FA method since no passwords are stored and it is just a simple confirm button.

It would be desirable for IRMA to have a somewhat similar functionality. When browsing on a laptop or desktop, instead of scanning the QR-code, a push notification is sent to the mobile device. On receiving the push-notification, the IRMA app is opened. When the user turns on their devices, the IRMA app is already present and the user can directly select the attributes that should be provided. The big difference is that Google Prompt already knows which phone belongs to which user. The user is already signed in with their Google account on their phone. This is not the case with IRMA. The QR-code can be scanned with any instance of the IRMA app.

4.1.5. Bitwarden and Browser sync

Bitwarden is one of many password managers[Bitwarden, 2020]. It is freemium, open-source and offers 2FA (two-factor authentication) with email, authenticator apps and other services. Bitwarden is available on multiple platforms, including browsers with the use of a plugin. All data is stored in the cloud and fetched after a login. When offline, you can still use Bitwarden using the latest synced version of your passwords. To prevent password leakage should the data get exposed, the passwords are encrypted using AES-CBC 256-bit

encryption [Bitwarden-FAQ, 2020]. Browser sync works in a similar way. Some browsers such as Chrome[Help, 2021] and Firefox[Mozilla, 2021c] include the same functionality as Bitwarden. These browser have an option to store and sync information such as passwords, tabs and bookmarks. Information is synced over other instances where the user is logged in.

What is interesting about Bitwarden is that it comes with a browser plugin. This plugin can detect when there is a login screen present for almost any website. Next, the user can select an entry to fill in the username and password field. In part, we want to achieve similar behaviour, only we want to detect the IRMA QR-code and send it directly to the app.

4.1.6. Securing Instant Messages With Hardware-Based Cryptography and Authentication in Browser Extension

Rodrigues et al. propose a method to secure Instant Messages With Hardware-Based Cryptography and Authentication in a Browser Extension[Pimenta Rodrigues et al., 2020]. Rodrigues et al. use *converse.js* which is a free and open-source XMPP JavaScript chat client that can be opened in a browser. This client connects to an XMPP server. The communication between the client and the server should be secure for which encryption is used. To achieve true randomness for encryption, Rodrigues et al. use Hardware-Based Cryptography. Rodrigues et al. created a browser plugin which reads(decrypt) from and writes(encrypt) to the *converse.js* client. To communicate from the plugin to the hardware cryptography services, Rodriques et al. use the *Native Messaging Host(NMH)*. The NMH communicates with an application that is installed on the user its client. This application actually communicates with hardware cryptography services.

Rodrigues et al. created a browser plugin that not just enhanced the functionality of the client, but also overrides it. Rodrigues et al. describe a number of encountered difficulties, related to creating the browser plugin. This can be very informative when realising the IRMA browser plugin.

4.1.7. Mightytext and FingerKey

Mightytext and FingerKey are browser plugins that communicate with mobile devices. Mightytext [Mightytext, 2020] allows you to send Text messages from a browser plugin, or other devices, through your mobile phone. By signing in to the client and your phone, you pair the devices. FingerKey [FingerKeyApp, 2020]is a password manager that works somewhat like IRMA. Passwords are stored encrypted on an iPhone. The iPhone is connected to a macOS device using Bluetooth, WiFi or Push notifications. A browser plugin can be installed to fill in usernames and passwords. Like IRMA, data is stored on a mobile device and never stored on a server. However, backups can be made and stored on iCloud or Google Drive. Unfortunately, there is not much to find about these two extensions since they are closed source.

Both Mighttext and FingerKey are browser plugins that enhance browser functionality by providing functionality from a mobile device. In the same way, we could provide IRMA app functionality to the IRMA login page in a browser.

4.1.8. SUMMARY

Tiqr is somewhat similar to IRMA. The big difference is that Tiqr is about identity and IRMA about attributes. Solutions such as Okta Verify Push and Google Prompt still require a username and password for an account stored on a server, it is nothing more than a simple confirm button on your phone. Bitwarden is a password manager that also requires a username and password to access your data on their server. IRMA does not require an account on their server.

Rodrigues et al. created an interesting browser plugin that encrypts and decrypts browser client messages. The IRMA browser plugin should detect the QR code, just like Rodrigues et al. detect client messages. Mightytext and FingerKey both provide access to your phone functionality using a browser plugin. Unfortunately, it is unclear how these work. Table 4.1 contains the listed applications and their used methods to communicate between a desktop and mobile.

Table 4.1: Used methods in other applications

	Push notifications	Direct connect (Wi-Fi/Bluetooth)	QR-codes	Other / none	Unknown
Tiqr			x		
DigiD			x		
Rabobank			x		
Okta Verify Push	x				
Google Promt	x				
Bitwarden				x	
Browser sync				x	
Mightytext					x
FingerKey	x	x			

4.2. HOW DOES THE PROCESS OF CONNECTING AN IRMA APP TO AN IRMA SERVER, USING QR CODES, WORK?

To answer the main research question, we should know how the IRMA app works. This information is important because it will tell what information should be sent from the browser extension to the app. Of course, it is not necessary to fully understand every part of the IRMA app. For example, it is unnecessary to know what kind of encryption is used or how the app communicates with the server. However, it is necessary to understand how to QR-code part works and what data is sent to the app.

An IRMA session starts with the user performing some action on a website [IRMA, 2021]. For example, this could be the user clicking on a "Log in with IRMA" button. This website (also called the requestor) requests a session from the IRMA server. The IRMA server sends back the QR-code content, which consists of a session token and an URL to the server. This QR code is presented by the *irma-frontend* and can be scanned with the IRMA app. Upon scanning, the IRMA app requests the session from the IRMA server. The IRMA server returns the session. This session contains the requested attributes, the issued attributes or a message to be signed. Finally, the app presents the session information, at which point the user can accept the request.

As stated above, the requestor requests a session from the IRMA server. This can be the *irma-frontend*, but also a backend. *Irma-frontend* provides a set of JavaScript packages that together communicate to the IRMA server. The purpose is to have a flexible IRMA client in the web browser. The frontend client can either communicate directly to the IRMA server or proxy through a backend.

Since the IRMA QR-code is just a regular QR-code, it can be scanned with any QR app. When scanning a QR-code generated by the IRMA demos with any QR-code app, the same pattern occurs. The QR-code content consists of two properties, *"u"* which is the URL and *"irmaqr"* which is the QR-code type. The types found were *disclosing*, *issuing* and *signing*. An example result is displayed below (Listing 1).

```
{
    "u": "https://privacybydesign.foundation/backend/irma/
            session/wICkh6QehT8sJdd6dYcV",
    "irmaqr": "disclosing"
}
```

Listing 1: IRMA QR-code example

When using one of the IRMA demos on a mobile phone, the QR-code part is omitted. Instead, the user is directly taken to the IRMA app. So, for example, the user clicks on the "Log in with IRMA" button and the IRMA app opens. When faking a desktop browser to be a mobile Android device, the IRMA demo tries to open the IRMA app. This obviously fails because there is no Android IRMA app installed on a desktop. The browser throws an error in the console showing what the browser is trying to navigate to. This error contains the same JSON object and what app to open it with. An example is displayed below (Listing 2).

```
intent://qr/json/{"u":"https://privacybydesign.foundation/backend/irma/
session/j7MweFY3mYzAGWKKYjEI","irmaqr":"disclosing"}#Intent;package=
org.irmacard.cardemu;scheme=cardemu;l.timestamp=1612109193199;
S.browser_fallback_url=
https://play.google.com/store/apps/details?id=org.irmacard.cardemu;end
```

Listing 2: IRMA Android browser link example

The same thing happens when faking a desktop browser to be a mobile iOS device. Again, though slightly different, we can see the same JSON object. An example is displayed below.

The process of connecting an IRMA app to an IRMA server using QR codes is quite simple. Upon a user action, the requestor requests a session from the IRMA server. This can be either from the frontend directly or through the backend as a proxy. The IRMA server returns a session token and an URL to the server. These are used to create the QR-code. The QR-code presented on the screen is just a simple JSON object with two attributes. When

```
irma://qr/json/{"u":"https://privacybydesign.foundation/backend/irma/
session/xhrLz52zaVrpsIwf7QLJ","irmaqr":"disclosing"}
```

Listing 3: IRMA iOS browser link example

faking a browser to be a mobile device and performing a user action (login for example), the same JSON object is passed to the app.

4.3. WHAT ARE THE REQUIRED COMPONENTS FOR REALISING THE IRMA BROWSER PLUGIN?

In order to omit the scanning of a QR-code, another way is needed to get that QR-code information to the IRMA app. In order to do so, a set of components is needed. Two of them are pretty obvious. The first is a browser plugin, also called a web extension. This would make it able to inject extra functionality into a webpage or the browser. The second component is an app. This app would somehow receive the QR-code information. The app does not necessarily have to be a new app. It could also be a modification to the current IRMA app. However, with a separate app would be possible to create a generic solution.

4.3.1. DETECTING QR-CODES

The most important task of the browser plugin is to detect the actual QR-code. A browser plugin is suitable for this task because it injects functionality to the webpage on which the QR-code is displayed. A native program could probably also do the task but would require to be developed for multiple platforms while browser plugins are mostly platform-independent [Mozilla, 2021d]. There are three possible solutions to detect QR-code, which are discussed below.

Intercepting network traffic and check if it contains session format

For RQ2, we found that the IRMA server sends information to the frontend, which is used to create the QR-code. This can actually be seen by inspecting network traffic. When inspecting the network traffic when using the IRMA email verification, we can see the following GET request is made.

```
https://privacybydesign.foundation/demo-en/start_session.php?type=gmail
&lang=en
```

Listing 4: IRMA demo session request

A session is started and the following response is returned.
The response contains the content of the Qr code, which is in a specific format.
Other IRMA demos do return the same response format. The same applies to other

```
{"sessionPtr":{"u":"https://privacybydesign.foundation/backend/irma/
session/dbeV4OBAK63rYjjP3j4c","irmaqr":"disclosing"},
"token":"4SJ2iPMgN2VWVnMbkKFP"}
```

Listing 5: IRMA demo session response

```
{"u": ..., "irmaqr": ...}
```

Listing 6: IRMA QR-code format

websites that use IRMA [Privacy by Design Foundation, 2021]. However, clients are free to use another format.

Intercepting HTTP responses can easily be achieved in Firefox with the webRequest filterResponseData() API [Mozilla, 2021e]. However, FireFox has only a small market share of a few per cent Netmarketshare, 2020; Statcounter, 2020; W3schools, 2020. Chrome, which has the biggest market share, does not have such an API. In fact, it is a desired feature for more than 6 years [Chromium.org, 2021].

Though not supported by the Chrome API, there is a way to intercept some of the network traffic. This solution is based on overriding the browser its default *XMLHttpRequest Prototype* [Stackoverflow, 2021]. The *XMLHttpRequest* object is used to fetch data from an URL. It can be used to fetch all kinds of data types, not just XML as the name suggests [Mozilla, 2021f]. So by overriding the *XMLHttpRequest* object we can let the webpage use our own version of the *XMLHttpRequest* object.

To understand how we can override the default *XMLHttpRequest* object, it is necessary to have a better understanding of browser plugins, also called Web Extensions. A browser plugin consists, *inter alia*, of background scripts and content scripts [Mozilla, 2021a]. Background scripts are loaded when the plugin is installed and stay active till the plugin is disabled or removed. Background scripts can use the special apis provides by the browser given they have the right permissions. Background scripts have their own context and therefore cannot access the same *window* object as the webpage.

Content scripts are part of the webpage context. Just as regular scripts, content scripts can modify the DOM. However, content scripts cannot see variables defined by page script. Therefore, we cannot just override the default *XMLHttpRequest* because build-in DOM properties are not shared between the content script and the page script. However, we can work around this problem.

As stated above, content scripts can modify the DOM. That means we can append a new script to the *head* of the page. This new script becomes part of the webpage so it becomes a pagescript. The new script can now override the default *XMLHttpRequest* object with our own behaviour. Communication between the new script and the plugin can take place with a simple *eventlistener* that passes text.

I was successful in overriding the default *XMLHttpRequest* object with my own behaviour. Using that, it was possible to log the responses on the IRMA demos and other IRMA webpages. Though successful, this method relies on the IRMA client using *XML-*

HttpRequest. When using WebSocket for example, overriding the default *XMLHttpRequest* is not useful. To my knowledge, there currently are no clients that use WebSocket. However, clients are free to use WebSockets. IRMA does not require to use *XMLHttpRequest*.

Though interception network traffic will work for current IMRA implementation, there is no guarantee it will work for future implementations. Clients are free to use something else than *XMLHttpRequest* and another format than specified above. Listing 6

Detect QR-code in the Document Object Model (DOM)

The Document Object Model (DOM) contains almost everything present on a webpage. Almost every image, *div* link and button is contained in the DOM. This allows searching for a specific element if you know how to find it. If we know how to find it, we can get access to the IRMA QR-code.

Analysing the IRMA demos and other pages that use IRMA [Privacy by Design Foundation, 2021], we find that there is no uniform way to detect the QR-code. Both the HTML *img* and the *canvas* tags are used and also ids and or classes differ. That said, there is no way to know what we are looking for. The table below 4.2 contains the different ids, classes and elements found.

Table 4.2: QR-code specification on webpages

IRMA page	QR id	QR element
https://www.irma-meet.nl/	class="irma-web-qr-canvas"	canvas
https://qrona.info/	class="irma-web-qr-canvas"	canvas
https://helder.health/		canvas
https://qrona.info/	class="irma-web-qr-canvas"	canvas
https://login.ivido.nl	class="irma-web-qr-canvas"	canvas
https://medipark.hix365.nl/		img
https://huisartsenpraktijkde1elijn.hix365.nl/		img
https://mijn.huisartsenpraktijksnelder.nl/		img
https://www.030irma.nl/	id="modal-irmaqr"	canvas
https://privacybydesign.foundation/	id="modal-irmaqr"	canvas
https://id.amsterdam.nl/	id="irma-qr"	canvas

Taking screenshots and scanning for the QR-code

Another solution would be to simply take a screenshot of the webpage and scan for the QR-code. On the *chrome web store*, there are multiple extensions that can take screenshots, so this should be possible for the IRMA plugin as well. There happens to be a webextension api that takes screenshots Mozilla [2021b]. Next, the captured screenshot should be processed and the QR-code content should be extracted. Since QR-codes are not something new, there are a lot of JavaScript libraries that can scan and decode QR-codes.

There is already an open-source plugin available that does everything described above [Beizhedenglongdev, 2021]. As expected, that plugin uses the webextension api to take a screenshot. This screenshot is then processed by an open-source QR-code reader, which is included in the plugin. The plugin is perfectly able to scan a QR-code on the IRMA demo page. Since it is an open-source plugin, it can be modified to fit our needs. An example is provided in figure 4.1.

Figure 4.1: QR Code Reader plugin

An advantage of taking screenshots and scanning for the QR-code is that it is not tied to IRMA. It could scan any QR-code and sent that to a mobile device. For example, I was able to scan a QR-code generated by the DigiD app. This solution does work very well. Basically, it does the same as a regular user would.

4.3.2. Sending the QR-code content to the mobile device

Now that there is access to the QR-code content, there should be a way to send that content to the mobile device. A possible solution is to use push notifications like Okta Verify Push[Okta, 2020] and Google Prompt[Google, 2020]. Another option is to store a copy of the IRMA app database in the browser plugin just as Bitwarden does[Bitwarden, 2020]. FingerKey [FingerKeyApp, 2020] keeps a persistent connection to the mobile device using Wi-Fi or Bluetooth which is also a possible solution to send the QR-code content to the mobile device.

Push notification

Both Android(Firebase Cloud Messaging[Firebase, 2021]) and iOS(Apple Push Notification[Apple, 2021b]) support push notifications. Firebase Cloud Messaging(FCM) even supports push notifications on multiple platforms. Using push notification it is possible to notify a client app. For example, notify a mail app that there are new emails. The advantage of push notifications is that the client does not have to poll for new mails anymore, but gets notified instead. It is also possible to send a small payload, which can be useful. In the case of the mail app example, it would be possible to send the subjects as payload so the new subjects can be displayed without fetching all the new mails.

Push notifications work by first registering as a client, an mail app for example, to the

push notification service. Upon registering, a unique id is received. A push notification is first composed by a trusted client, a backend for example. This notification is then sent to the notification service backend. The notification service backend generates a message-id, possible other metadata and sends it to the platform-specific transport layer.

A push notification can be used to send the QR-code content to the mobile device. The mobile device registers to the push notification service and receives a unique id. The browser plugin should then use this id to send a notification to the device that is registered with the id. Since the push notification should be composed by a trusted client, a backend is needed. The plugin sends the id and the QR-code to the backend. The backend then composes the push notification and sends the notification to the push notification backend service. Last, the notification is sent to the device.

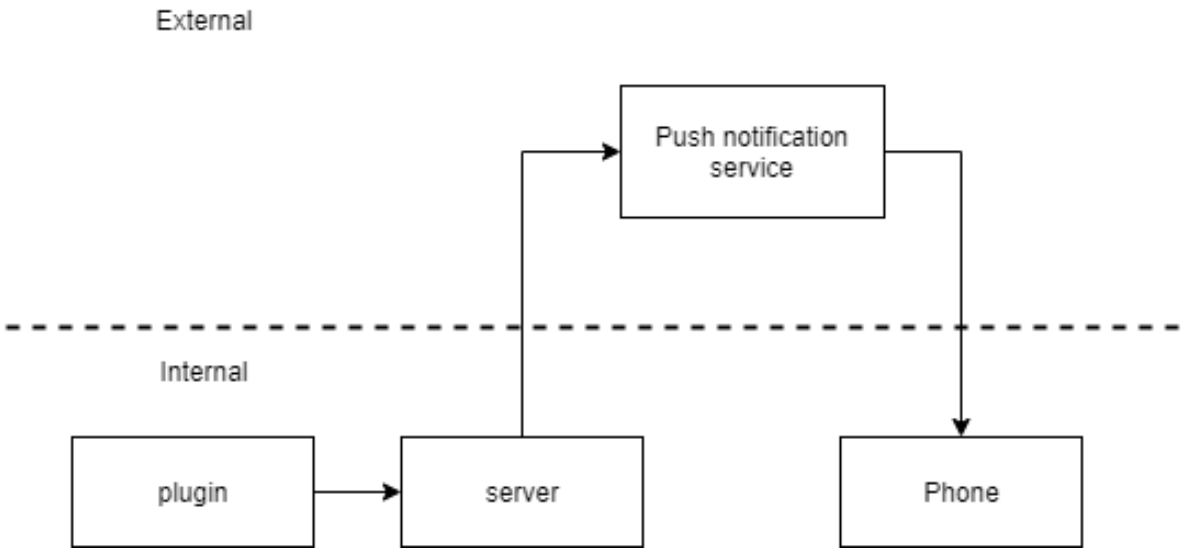

Figure 4.2: Push notification

Storing a copy of the IRMA app database

Bitwarden keeps a database, which contains usernames and passwords, stored and synced on multiple devices. This way, Bitwarden can be used on all kinds of platforms at the same time. There is a Bitwarden plugin for all major browser, apps for mobile devices and webpage. A big difference between IRMA and Bitwarden is that IRMA stores attributes on the mobile device and not on a server. Bitwarden stores data on a server. Currently, IRMA does not have an export option. Attributes can not be transferred from one device to the other. The reason for this is that it involves some delicate security issues. Attributes are person-related, an export would probably allow a user to share attributes with others.

Simply having access to the IRMA app database is not enough. The database is encrypted to prevent leaking attributes. The database can only be unlocked with a special key. This key is obtained from the server by sending the salted and hashed PIN. Separately, the app and the server can not access attributes. Both have to cooperate. That means the plugin and the server will have to cooperate. Basically, the browser plugin will become a fully new IRMA client comparable to the mobile app.

Some attributes are further secured with key splitting. When using such an attribute the user must enter their PIN in order for the session to complete. This is forced by the server

and not by the client. To allow the use of these credentials in the plugin, the key splitting mechanism should also be supported.

Keep a persistent connection to the mobile device using Wi-Fi or Bluetooth
Fingerkey is a password manager that stores passwords and usernames on a mobile device. These passwords and usernames are then made available to a computer using a Bluetooth or Wi-Fi connection. Though generally available on a mobile device, not every computer does have Bluetooth. Using the GsmArena Phone Finder, we found that since 2010 6607 of the 6768 phones have Bluetooth[GSMARENA, 2021]. There is not any hard data on what percentage of computers has Bluetooth. So to get an idea of how common Bluetooth is in a computer we checked how many motherboards did have Bluetooth on a computer component webpage. On Tweakers.net 250 of the 1250 motherboards do have Bluetooth and on Alternate.nl it is only 2 of 200. So it is safe to assume that computers generally do not have Bluetooth. That makes Bluetooth not very suitable.

Another option is Wi-Fi. Two devices can connect with Wi-Fi direct. Wi-Fi Direct was introduced by the Wi-Fi Alliance in 2010 [Khan et al., 2017]. It enables two Wi-Fi devices to connect directly without an access point (AP). The Fingerkey webpage notes an interesting detail in the FAQ. It says Fingerkey is slower over Wi-Fi. This is because on iOS apps are put to sleep when they are in the background for more than three minutes. In order to make Fingerkey work after the app goes to sleep, it uses push notifications, which are a bit slower. It is unfortunate that Wi-Fi cannot be used on its own. Since we are pushing data to the app(QR-code content) and not requesting data from the app, using Wi-Fi in addition to push notifications does not add any value. Using only push notifications on its own is sufficient.

According to the Apple docs on background execution, an app has 5 seconds to perform background task when it is put in the background. This can be extended, but it is unclear by how much. A few scenarios in which background execution is allowed are audio communications, location services, communication with an external (Bluetooth LE) accessory, regular updates from a server, and push notifications[Apple, 2021a].

The problem on iOS devices described above does not occur on Android. Though Android limits background execution, it makes a distinction between so-called background services and foreground services. The name foreground service is a bit confusing since it is not really in the foreground. It is running in the background but noticeable by a user by an icon in the top bar. The operating system does not limit these foreground services.

The Web Extension API does not really support both Bluetooth and Wi-Fi Direct. Bluetooth is somewhat supported, but functionality differs for each browser. Though not supported by the Web Extension API, it would still be possible to use Bluetooth or Wi-Fi. Rodrigues et al. use the *Native Messaging Host(NMH)* to communicate with a native application installed on the user its client[Pimenta Rodrigues et al., 2020]. The browser plugin will communicate with the natively installed application. The native application will then use Wi-Fi or Bluetooth to connect to the mobile device.

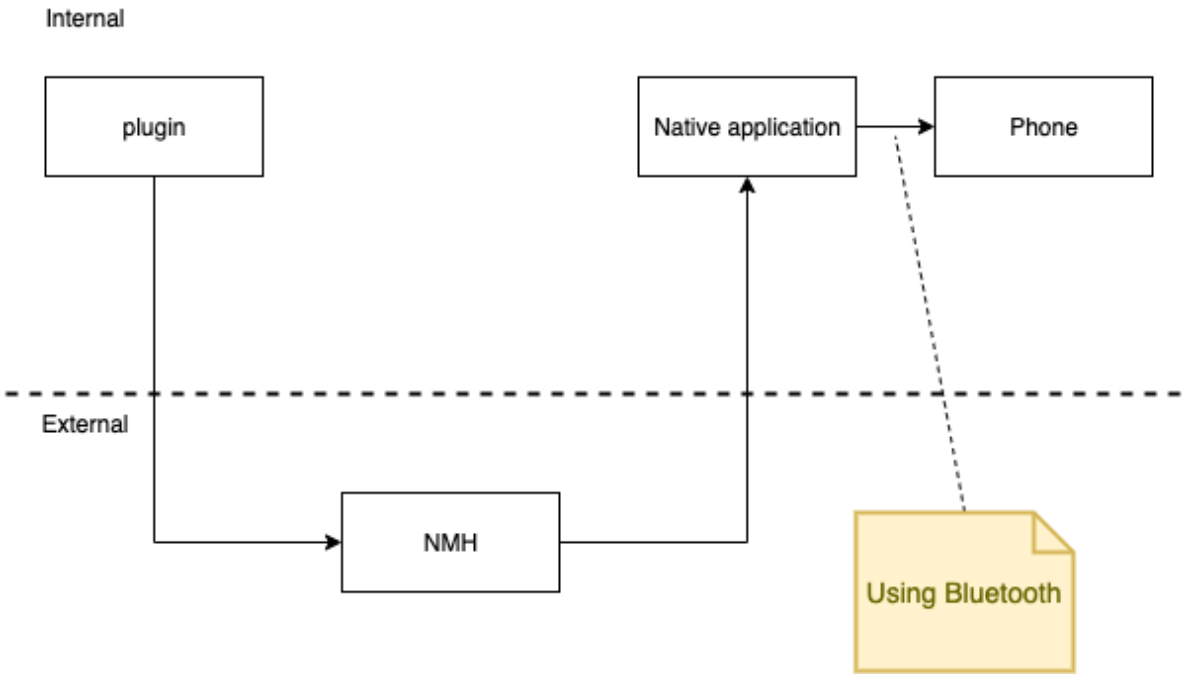

Figure 4.3: Bluetooth

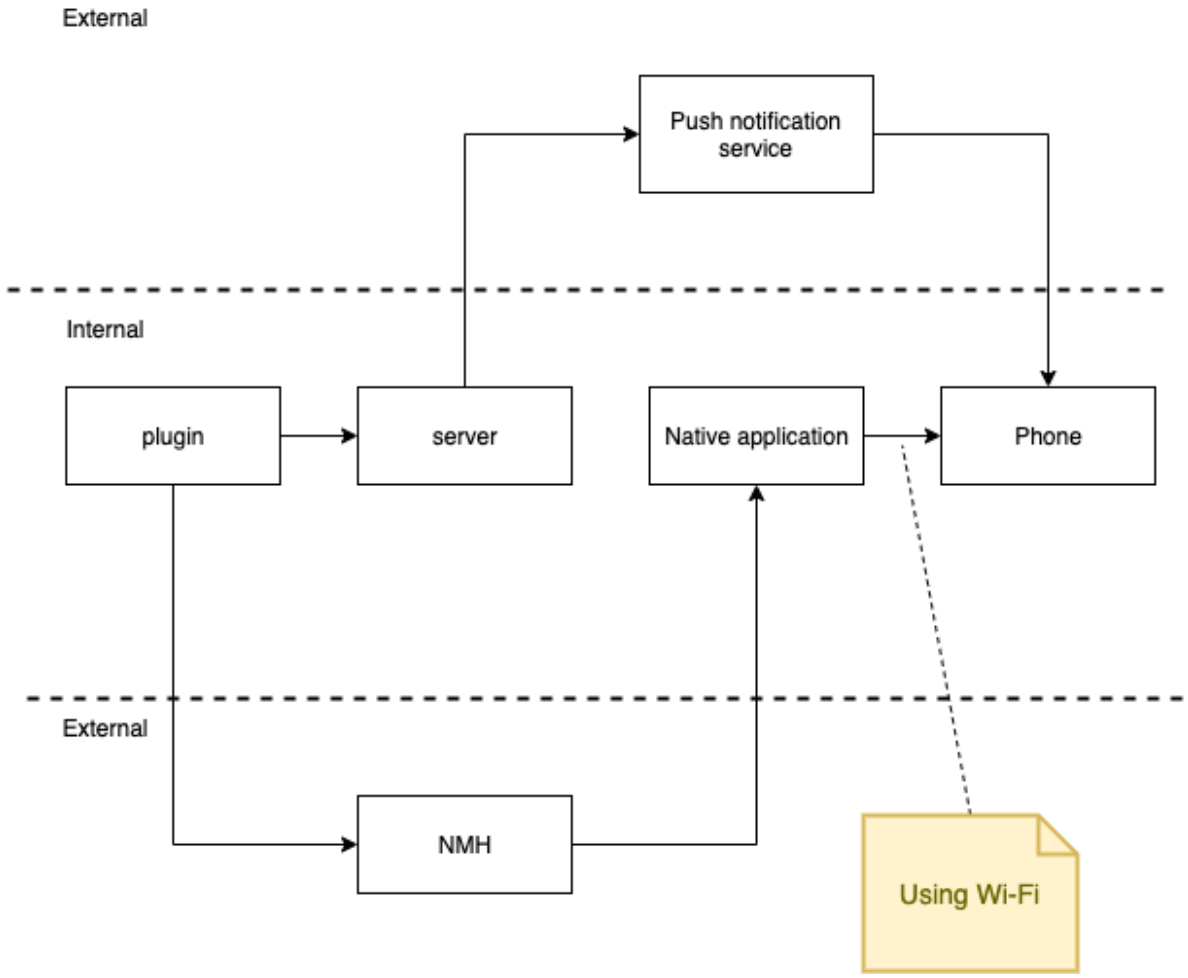

Figure 4.4: Wi-Fi

After detecting the QR-code and sending the content to the mobile phone, it can be used to open the IRMA app. To prevent modification to the IRMA app, a separate app is needed for now to receive the content and open the IRMA app.

4.4. What is the impact on privacy and security?

This section addresses the impact on privacy and security. This section is divided into a pre-development part and a post-development part. The pre-development part describes the impact on privacy and security based on the components described in RQ3. The post-development part describes the impact on security and privacy based on the developed system.

4.4.1. Pre-development

Only two of the three methods described to decode the QR-code are viable. Both of those two options have somewhat of an impact on privacy. The first option is to scan all the network traffic and scan for the IRMA session object. This is not an ideal solution since we are really only interested in the QR-code content. Another point is that there is no *network inspection* permission. It is just a modification to the *XMLHttpRequest* object, for which the *all_urls* permission is required. Upon installation or run time, the permission is requested

as shown in figure 4.5. Also, the statement: "This plugin scans all the network traffic to find the QR-code" does not feel very privacy-friendly.

The second option is to take a screenshot and process the screenshot with an open-source QR-code scanner. There is no dedicated permission to make a screenshot, though the *all_urls* permission is required making it somewhat transparent. An offline QR-code scanner is preferred to make sure no data is leaked to other parties Mozilla [2021b].

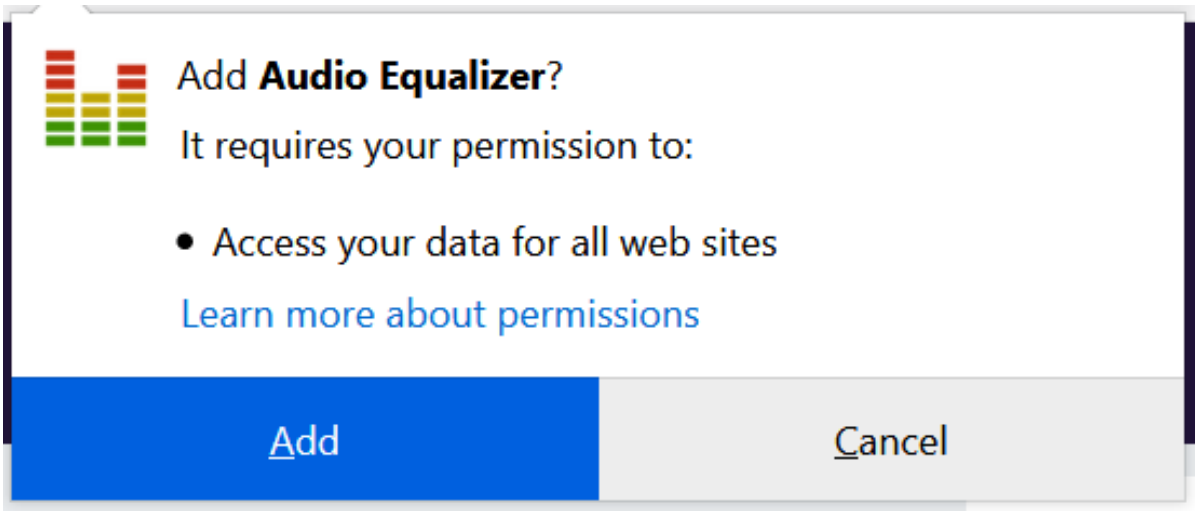

Figure 4.5: Plugin permission

Usually, a browser plugin can not make an HTTP request. Making a request would require a *host permission* for the host the request would be sent to. That way the user would be aware that the browser is accessing a particular website. Unfortunately, both methods to detect the QR-code content require the *all_urls* permission. Therefore, the browser plugin automatically can make requests to any page. It would be a nice enhancement if there would be a dedicated permission to make a screenshot.

An advantage of the screenshot method is that a single screenshot is sufficient. As soon as the QR-code a present on the screen, a single screenshot can be processed to read the QR-code. Taking that screenshot can be controlled by the user clicking on a button. In contrast, the network scanning option has to be enabled all the time. It cannot be enabled by the user clicking on a button after the screenshot is present because that is simply too late. The request for obtaining the QR-code data has already passed.

The disadvantage of push notifications is that it requires a third party service. From both a privacy and a security perspective, this is not ideal. Since the third party basically relays the message, it has access to the content of the message, knows to who the message is sent, when the message is sent and to what app. The content of the QR-code also contains the URL to the IRMA server, so it is likely the third party will also know what page the user is visiting. Since the third party is not within our sphere of influence we cannot control their security. If the third party gets compromised, so does our system. The issues related to security and privacy above can partially be solved by encrypting the content which is sent in the message. The content can be encrypted by the browser plugin using a passphrase. This same passphrase can then be used to decrypt the message on the phone. Using encryption, the identity and the integrity of the message can be assured. However, the third party would still be able to know when a message has been sent, to whom and to what app.

By storing a copy of the IRMA app database in the browser plugin there is little to zero impact on privacy. Since it is basically a new client, there is almost no difference between the plugin and the app. However, there is a huge potential security risk. Any mistake can compromise the browser plugin client. To create this new client, it would be required to understand every aspect of the IRMA app, which seems like a lot of effort for the problem we are trying to resolve. Second, the IRMA app consists of approximately 28000 lines of dart code. Assuming the same number of lines is required for the new client, it is highly unlikely there will not be any bugs.

Using a persistent Wi-Fi connections suffers from the same privacy and security issues as push notifications. Since the background apps are put to sleep and the connection as well, push notifications have to be used in addition. That means the persistent Wi-Fi connection inherits the same problems stated above. Also, other points of attacks are introduced. The plugin should use the NMH to communicate with a native application installed on the user's machine. This native application should then communicate with the mobile device over Wi-Fi. The NMH, native application and communication over Wi-Fi are all possible points of attack. Since it is a native application, it should be developed for multiple platforms, meaning all possible points of attack should be multiplied by the number of supported platforms.

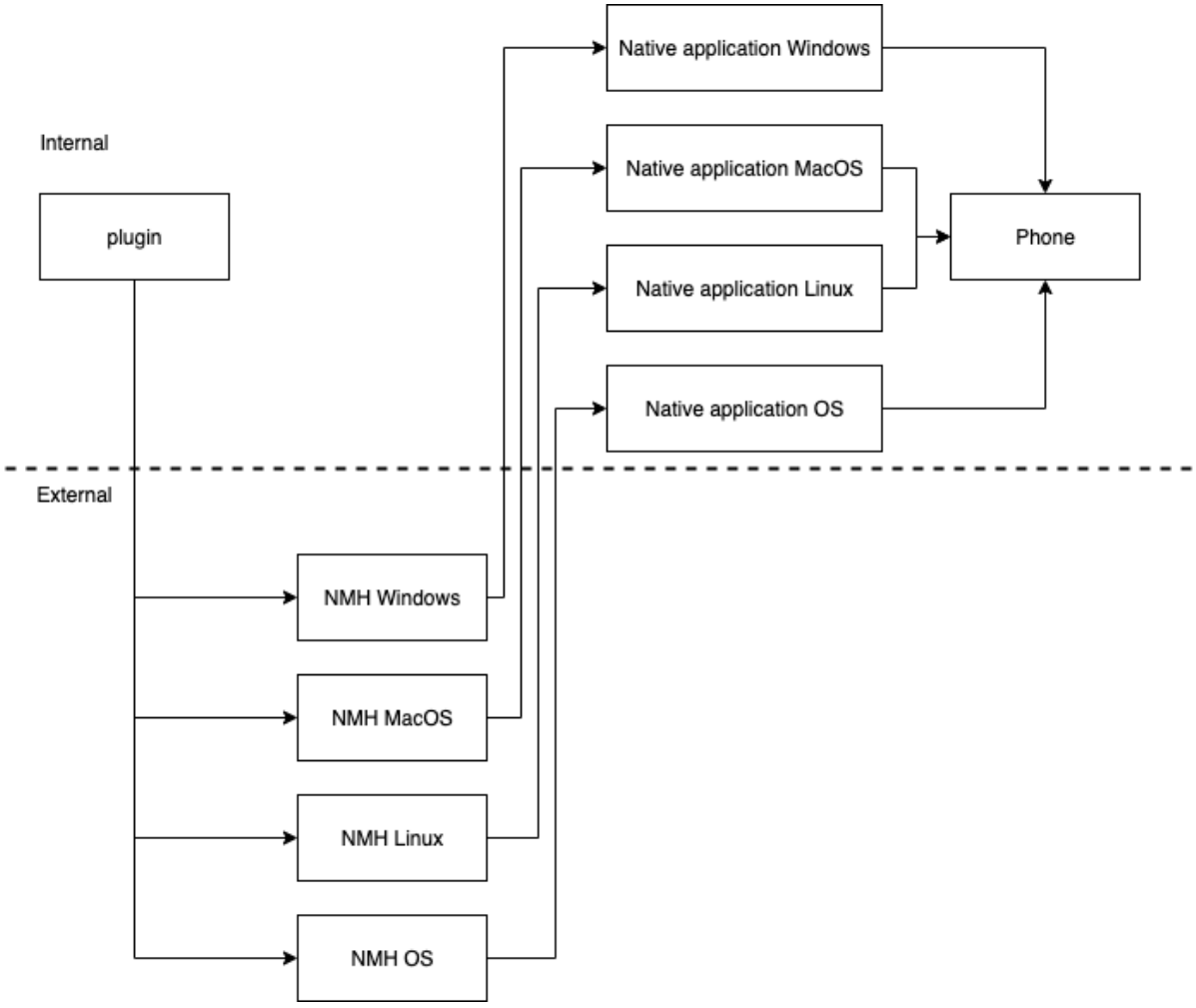

Figure 4.6: NMH

When using Bluetooth to communicate with the phone, it is not necessary to use push notifications. Because it also uses the NMH it also has many points of attack.

4.4.2. POST-DEVELOPMENT

When choosing the most viable option, which includes a browser plugin that takes a screenshot of the QR-code, processes it and sends it to the mobile device, a dataflow diagram and component diagram can be created 4.7, 4.8.

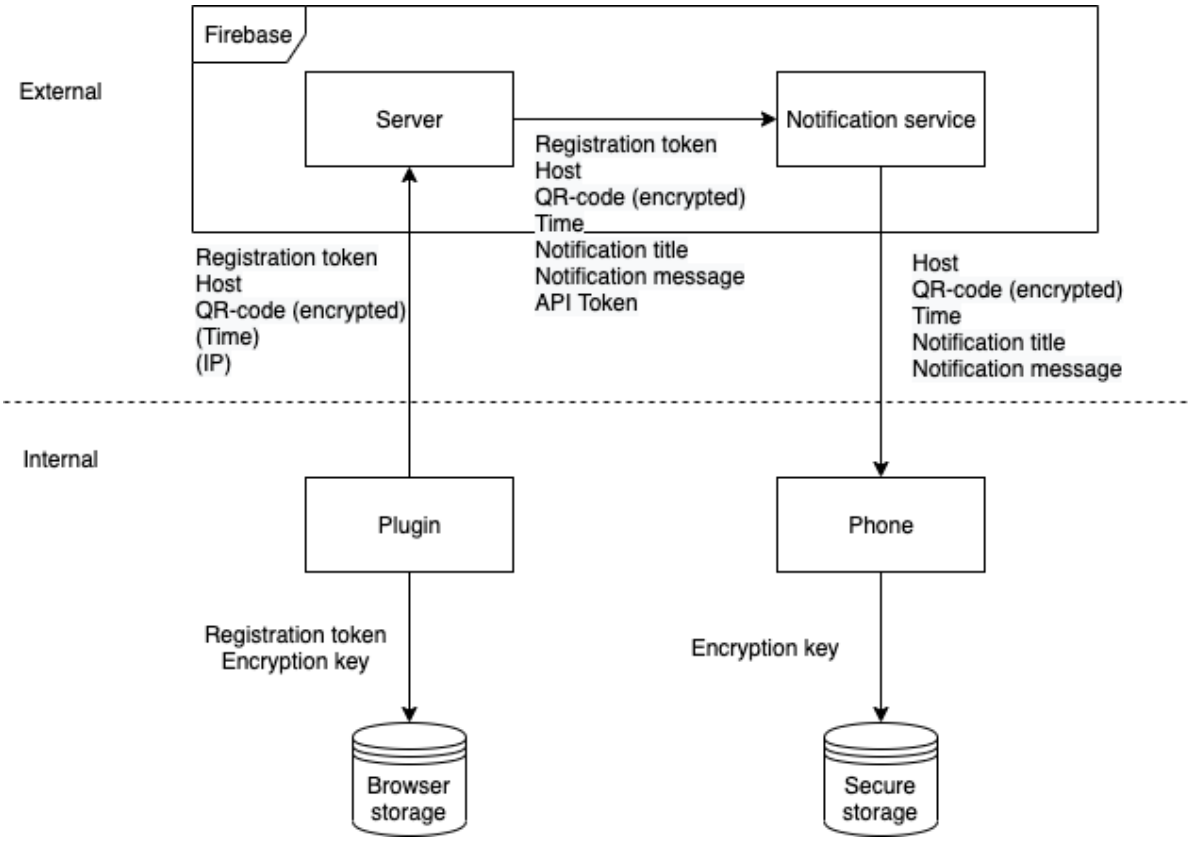

Figure 4.7: Dataflow 1

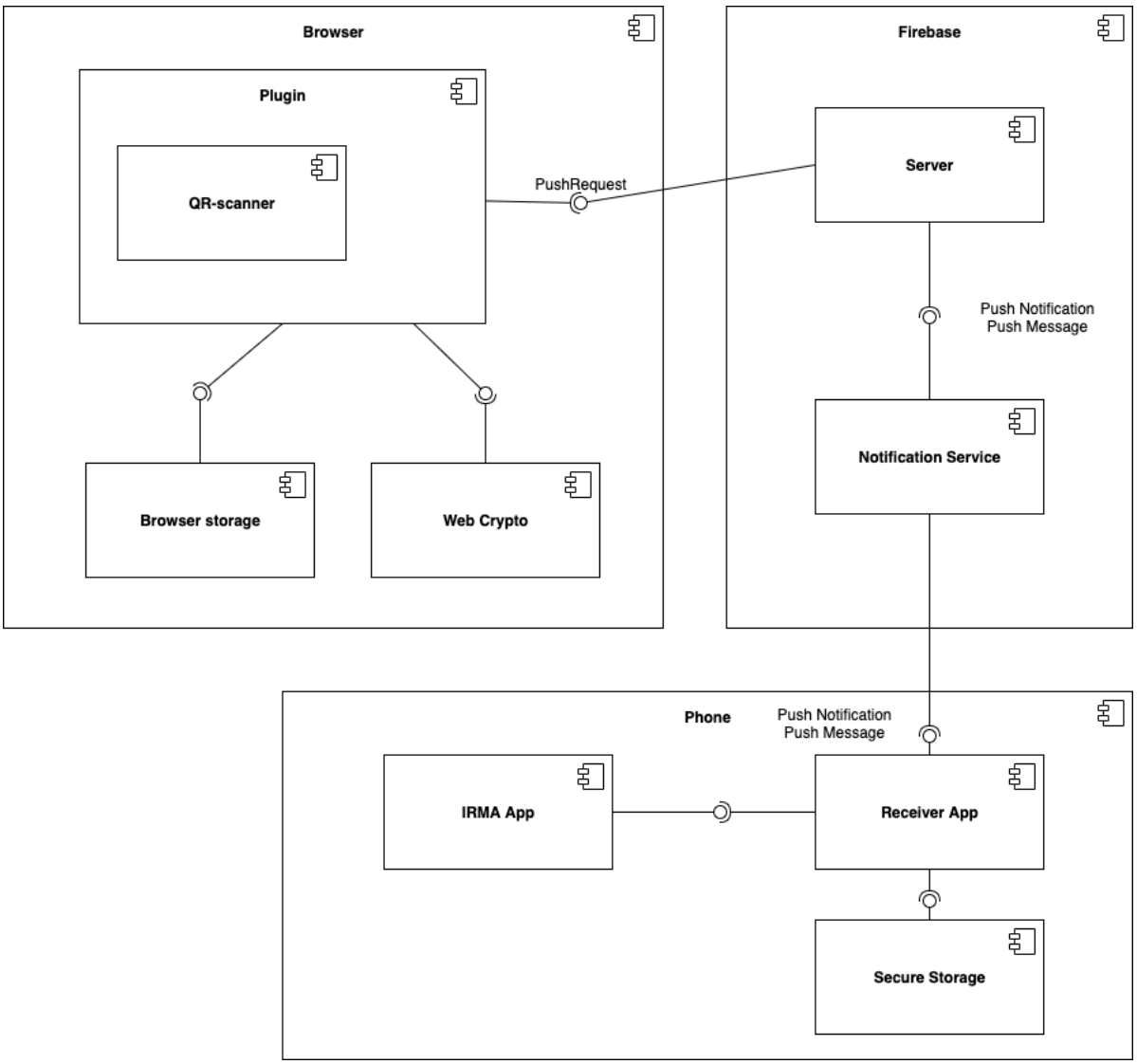

Figure 4.8: Component 1

The plugin scans the QR-code and sends it to the mobile device. The QR-code is scanned using an offline scanner. That means it does not require an internet connection and can scan the QR-code by itself. In order to send a push notification to a specific device, a registration token is required. This is a unique token that can change though. This token is too long to enter every time, so it is stored in the browser storage.

The plugin sends the QR-code, registration token and the host from which to QR-code was presented to the server. These attributes are part of the *PushRequest*. It is worth noting that this is an external server that runs on Firebase. Therefore, Firebase also knows the IP address and the time and date of the request.

The server now creates the push notification. This notification includes a *Push Message* which contains the host, QR-code and the current time. The server also composes a notification title and message which are presented to the user. All these fields are sent to the phone with the use of the registration token and the notification service API token. This API token is stored on the server.

The advantage of this system is that no user data is stored on some server. All the neces-

sary data, which is only the encryption key and registration token, are stored on the user's device. There is simply no user account. Disenrolment is achieved by simply deleting the app and browser plugin. The disadvantage of the system that it is a bit tedious to set up. After the user has installed the plugin and the app, the encryption key should be entered in both the plugin and the app. As described in the findings section, the registration token is required in the plugin. This is a long token, so it is not very convenient to enter it every time. Therefore, we store it in the plugin. Even entering it only one time would be annoying so it can be sent from the phone in an email. From a user perspective, enrollment would probably be easier with a single sign-on method such as *Sign in with Google*. This account could then be used to store the encryption key and registration token.

4.4.3. Security and privacy analysis

Using the STRIDE method [Microsoft, 2020], the system can be analysed in a systematic way. The STRIDE method works by looking at the system and each component and look for threats that fall into the Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service or Elevation of privilege category. The STRIDE method is performed on a very abstract version of the system. That means a system where almost no security and privacy-related measures are taken. The results are presented in table 4.3. If applicable, an explanation of how the possible attack is mitigated is present in the table.

Table 4.3: System threats

Threat	**Type**	**Mitigation**
An attacker would get access to the account of another user.	Information disclosure	The system is to be used without the requirement of a user account
An attacker would get access to the registration token of a legitimate user and use it to send QR-codes to the user.	Spoofing	To open the notification, authentication is required.
An attacker would get access to a legitimate user's registration token and encryption key and use it to send QR-codes to the user.	Spoofing	
An attacker would get access to the push notification server key and send QR-codes to users.	Spoofing	To open the notification, the QR-code content must be encrypted with the correct key.
An attacker would use a man in the middle attack to get access to the QR-code.	Spoofing, Information disclosure	The QR-code is encrypted
An attacker would use a man in the middle attack to replace the QR-code.	Spoofing, Tampering	The QR-code is encrypted using AES-GCM. Modification of the QR-code without the correct key will be noticed.

Continued on next page

Table 4.3 – *Continued from previous page*

Threat	**Type**	**Mitigation**
An attacker would use a man in the middle attack to get access to the hostname, notification title, notification message, timestamp or registration token.	Spoofing, Information disclosure	
An attacker would use a man in the middle attack to modify the hostname, notification title, notification message, timestamp or registration token.	Spoofing, Information disclosure	
An attacker would deny access to the server or Notification Service by flooding it with TCP/IP packets.	Denial of service	Both are protected by Firebase
An attacker would get access to the registration token and flood the device with push notifications	Denial of service	
An attacker would modify the registration token stored in the plugin to send QR-codes to their own device.	Tampering, Information disclosure	
An attacker would get access to the user their mobile device and read the encryption key and registration token.	Information disclosure	If supported by the device, the settings page is protected with a fingerprint or pin code.
A user claims unknown push notifications were sent by the system or an attacker.	Repudiation	A log shows when QR-codes have been received.
An attacker makes an unnoticed modification to any of the used dependencies (supply chain attack)	Spoofing, Information disclosure, Elevation of privilege	
An attacker gains access to the plugin and sends QR-codes.	Spoofing, Elevation of privilege	To open the QR-code, the user always has to click on the notification. Even when the receiver app is already open.
An attacker places a QR-code on a popular website that is scanned by a legitimate user, without the user knowing it.	Spoofing	The QR-code is never scanned and sent automatically. The user has to press the scan button in the browser plugin
An attacker hides a second QR-code on a popular website that is scanned by a legitimate user while the user tries to scan a legitimate QR-code.	Spoofing, Information disclosure	An error is displayed in the plugin if multiple QR-codes are present

Continued on next page

Table 4.3 – *Continued from previous page*

Threat	**Type**	**Mitigation**
An attacker creates a malicious copy of the app or plugin and tricks the user into installing it.	Spoofing, Information disclosure	The plugin and app should contain a download link to each other.

From the component and dataflow diagram it can be noticed that Firebase has access to a lot of data. It does not use or need the data, it simply forwards the data to the mobile device. Access to the data is reduced by encrypting it using a text or a passphrase. This text or passphrase should then be stored in the plugin to encrypt and in the phone to decrypt.

Though Firebase only forwards the hostname, it can not be encrypted. The hostname is used to create a notification title that says:*QR-code received from somehost.com*. As describes in the findings section, notification messages are intercepted and presented by the operating system. Therefore, it is not possible to first decrypt an encrypted notification title. This would be possible with data messages. Unfortunately, data messages are less reliable than notification messages. Delivery is not guaranteed. Because not all data is encrypted, it would be possible for the QR-code sending system or the push notification services to create a some user profile. The registration token would be the unique identifier and it could be track on what website the QR-code sending system is used.

There are more benefits from encryption than just Firebase not being able to access the data. If the registration token leaks out, an attacker cannot send a meaningful message to the phone. The passphrase would be required. If an attacker would use a man in the middle attack to intercept or modify the QR-code the attacker would be unable to do so. With the correct type of encryption, it is possible to guarantee the authenticity, integrity and confidentiality of the QR-code. Authenticity is required to make sure only the user can send and receive the messages. Integrity is required to make sure no one has changed the data. Last, confidentiality is required to make sure no one can read the message.

Though the chances of the registration token leaking out are low, the token is sent through the whole chain. The plugin stores it and sends it to the server, the server sends it to the notification server and the notification server uses it to send a message to the phone. The token could leak anywhere in the chain. The same applies to the QR-code. Though the chances of tampering and leaking are low, it passes through the whole chain, which is why confidentially and integrity are desirable.

The chosen encryption method is AES-GCM with PBKDF2 to generate a 256 bit key. AES is well known and since it is symmetric the user can use the same readable encryption key on the plugin and the app. AES GCM also provides message authentication using a *Message Authentication Code (MAC)* [McGrew and Viega, 2005]. Another reason to choose AES GCM is the availability of libraries to perform the encryption and decryption for JavaScript and Dart Flutter as described in the findings section.

A man in the middle attack can be used to get access to most data. Information disclosure can be partly prevented. Public-key cryptography could be used between the plugin and the server. All data would be encrypted on the plugin using a public key. On the server, the data would be decrypted using the private key. This form of encryption would only partly solve the problem of information disclosure. For example, it cannot be done between the server and the push notification service because the push notification service is out of our control. Therefore, it does not really solve the issue of information disclosure in

the case of a man in the middle attack.

In essence, the QR-code sending system does exactly what normally a user has to do. It scans a QR-code and opens an app with it. That means barely anything changes for systems that use QR-code such as IRMA. For example, IRMA attributes will remain to be not stored on a server. The QR-code sending system is completely unrelated to IRMA. The only thing that changes is that it is less certain a user is actually behind the computer screen. Previously, you could send a screenshot of a QR-code, print it and scan it with your phone, but that is tedious. That could now be bypassed by using the QR-code system.

As stated before, the plugin has access to a lot of browser features due to the nature of the permission system. Not only that, but data such as host names and IP addresses are theoretically accessible by the QR-code system. Just as with any system, the user has to trust the system.

Based on the list of attacks, an attacker can perform four interesting actions. Those are intercepting notification data, intercepting QR-codes 4.9, sending QR-codes 4.10 and sending notifications 4.11. For three of these actions, an attack tree is created.

The two most dangerous actions for the user are intercepting QR-codes and sending QR-codes. In these cases an attacker could get access to confidential data. Less dangerous is the ability to intercept notification data. In this case an attacker will know the website what website the QR-code was scanned from. The least dangerous is the ability to send notifications. In this case, an attacker is able to spam the user with notifications. Note that the attacker in this case is unable to include a QR-code.

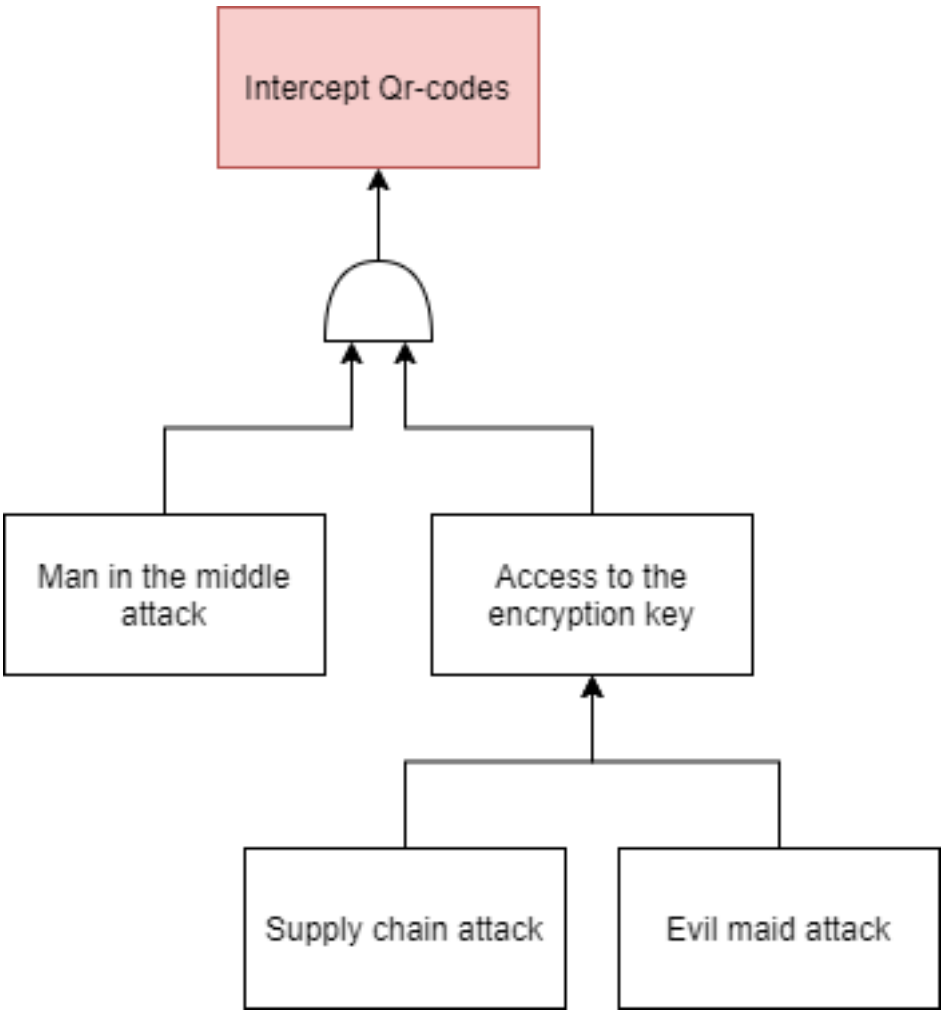

Figure 4.9: Attack tree intercepting QR-codes

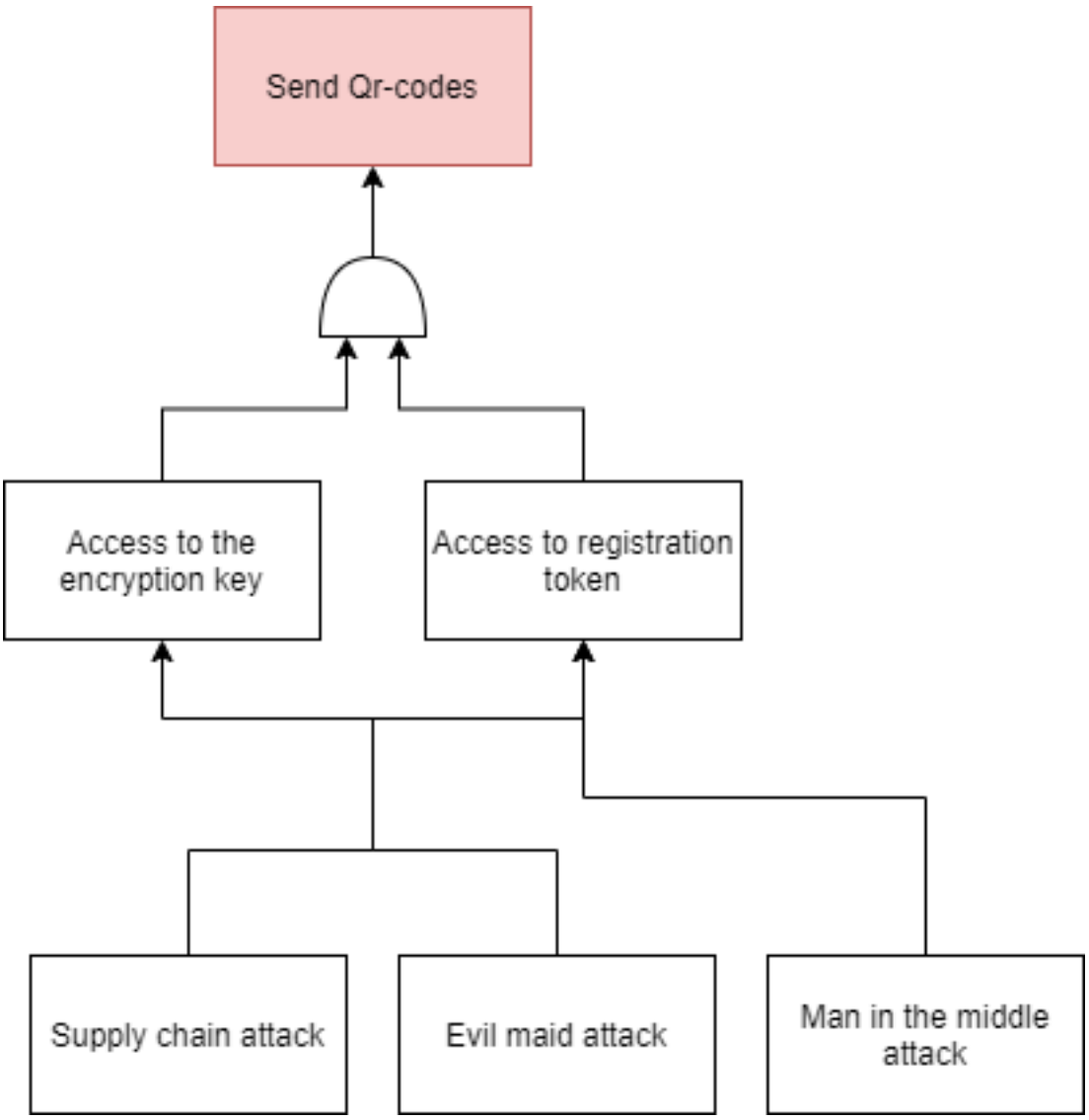

Figure 4.10: Attack tree sending QR-codes

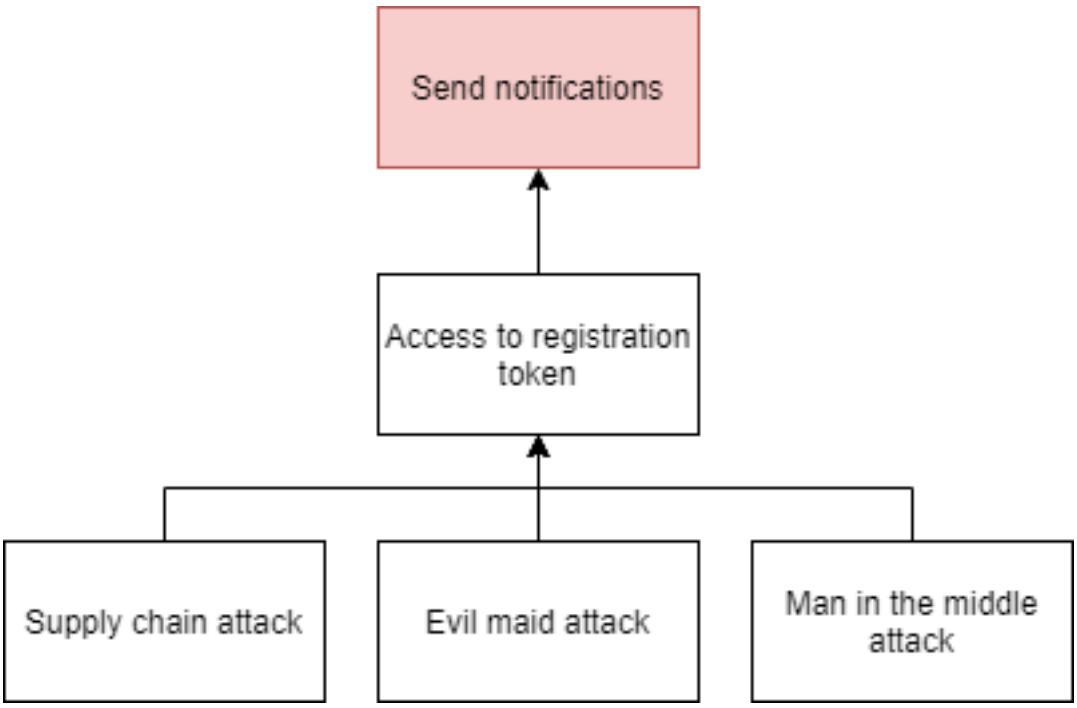

Figure 4.11: Attack tree sending notifications

4.5. How can the QR-sending system for IRMA be extended to also work for other QR-codes?

In this section we will look into how the QR-code sending system can be extended for other apps. We will look into how other apps use QR-codes and how other apps can be opened.

4.5.1. QR-codes

Not all apps work in the same way as IRMA does. For example, the Rabobank app, which is a banking app works differently. Using the Rabobank app, a *Betaalverzoek or Tikkie* can be sent, which is basically a payment request. This payment request is a link that can be opened in the browser. On a mobile device, the Rabobank app is opened and on a computer, a QR-code is presented. On the surface, it seems this process works in the same way as IRMA does. However, when analysing the QR-code and the process in the same way as above, there is no resemblance between the link that opens the app and the QR-code content.

When scanning the QR-code with a regular QR-code scanner, the following content is presented.

```
nl.rabobank.app:sign:78ANOeKEs-K77aCa5qiyLAubbyDPP0kJ2g-v1rRhjAS
:jiQHey9faBC
```

Listing 7: Rabobank QR-code content

When faking the browser to be a mobile device, in the same way as described for IRMA,

the following link is tried to be opened.

```
nl.rabobank.ideal://ideal-payment?trxid=20005022577860&random=
a086c7cc5f91e6630289f72826e9d0e7d2bacc1e1e464fb8cb3c3b3561f7fc0d
```

Listing 8: Rabobank Intent link

As can be seen from the two examples, there is no shared part. Without more information about these links, how they work and how the Rabobank app interprets these, it is impossible to open the app using the QR-code sending system.

Another app that uses a somewhat similar mechanism is the *DigiD* app, as discussed in RQ1. In addition to the QR-code, the user first has to enter a *linking code* on the computer, which is presented by the app. After entering the code, the QR-code is presented, which then can be scanned.

When scanning the QR-code with a regular QR-code scanner, the following content is presented.

```
digid-app-auth://app_session_id=40c5f44f-067e-4746-85b6-cd55f44a6a83
&lb=!YW7VYcncp0jDyF0zHRJoPG9/HBZPo3kSXdGvTOWAJdWx2JE3P0AsZCXUI+3Ro4c
DNvYZGcp0f+mbzhT4+SXjkQoxE/kf3/bkR7CD1qs=
&at=1619353659&verification_code=XLGX
```

Listing 9: DigiD QR-code content

On a mobile device, the following link is opened.

```
intent://#Intent;scheme=digid;package=nl.rijksoverheid.digid.pub;
S.data=digid-app-auth://app_session_id=bf4c8785-5595-5f08-9125-
36a622a72963
&lb=!YW7YVcnpn0jDyF0zHRJoDG9/ZBHPo3kSXdGvOTWAJdWx2JE3P0AsZCZ
UI+3Ro4cDNvYZGcp0f+mbxzT4+SXjkQoxE/kf3/bkR7CD1qs=
&at=1619353766;end
```

Listing 10: DigiD Intent link

Just as with IRMA, the QR-code and the mobile link are very much the same. In both cases, there is an *app_session_id* and an *lb* attribute. Both cases also include an *at* attribute, which is probably a timestamp. The QR-code also contains a *verification_code* which is the *linking code*. A mobile link can be transformed into a QR-code by simply appending the *verification code* to it. This code is visible in the DigiD app.

IRMA, Rabobank and DigiD all use a QR-code mechanism. However, all are slightly different. Of these three, IRMA is the most simple. The QR-code is embedded in the mobile

link. The same goes for the DigiD app. However, a verification code is added to the QR-code. The Rabobank app is different because there are no similarities between the QR-code and the mobile link. It is unclear how this works.

4.5.2. OPENING OTHER APPS

Apps can be opened by the OS by providing a special link. This is called *deep linking* or *universal links*. Apps can be registered to handle these special URIs. For example, the following code on a mobile webpage will open an app that is able to make phone calls.

```
window.open("tel:+4842566");
```

Listing 11: Phone number link

In this case, the app tells the OS it can handle the *tel:* URI. Other examples are *sms*, *mailto*, *http* and *https*. These are examples of standard URIs (RFC 5724,RFC 6068 and RFC 7230).

Apps can also define their own unofficial URI. This is the case with the IRMA app, as can be seen in RQ2. The IRMA app tells the OS it can handle URLs that start with *irma:*

```
irma://qr/json/{"u":"https://privacybydesign.foundation/backend/irma/
session/xhrLz52zaVrpsIwf7QLJ","irmaqr":"disclosing"}
```

Listing 12: IRMA iOS browser link example

On Android, the app registers the URIs in the app manifest. As can be seen from the example below, the IRMA app registers *irma*, *cardemu* and *https*.

```
<intent-filter android:autoVerify="true">
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="https" android:host="irma.app" ... />
</intent-filter>
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="irma" />
    <data android:scheme="cardemu"/>
</intent-filter>
```

Listing 13: IRMA Android manifest

In iOS, the app registers the URIs in the *Info.plist*

```
<key>CFBundleURLTypes</key>
<array>
    <dict>
        <key>CFBundleURLName</key>
        <string>foundation.privacybydesign.irmamob.alpha</string>
        <key>CFBundleURLSchemes</key>
        <array>
            <string>irma</string>
        </array>
    </dict>
</array>
```

Listing 14: IRMA iOS Info.plist

Unfortunately, it is not possible to get a list of all registered URLs. That is unfortunate because it is not possible to present a list of apps to the user to open the QR-code with. However, it is possible to hack around this problem in Android. On Android, it is possible to get access to the manifest of an installed app with the use of libraries [hsaifan, 2021]. Though there are differences between the original IRMA manifest and the extracted IRMA manifest, the part for registering URLs is identical. That means we can get all the registered URLs for a given app.

Though it is possible to find the registered URLs for an app, that is not sufficient enough. The IRMA app registers 3 URLs and the Rabobank app registers 9 URLs. It is simply unknown which of these registered URLs should be used to open the QR-code. Even if it is known which URL should be used, that is still not sufficient in some cases. This is the case with the IRMA app. One of the registered URL is *irma*, though as can be seen from the example below, the QR-code content is prefixed. This prefix "qr/json/" cannot be known by inspecting the Android manifest. In the case of IRMA, this prefix could be found by faking the browser to be a mobile device as described for RQ2 and above in this subsection.

```
irma://qr/json/QRCODE_CONTENT
```

Listing 15: IRMA iOS browser link template

Apps can register specific URLs which the app can handle. There is no universal way to figure out which app supports which URLs. On Android, is it possible to find all the registered URLs for a given app, though this is not an official API or method. Even with this information, it is not possible to simply open an app with the QR-code content. For each app, specific information is required to open it with the QR-code.

ANDROID

On Android it is possible to open apps with extra information using the Chrome browser [Developers, 2021]. Chrome uses a specific scheme which can be used by web pages to open apps. The basic format for an intent-based URI is as follows:

```
intent:
   HOST/URI-path // Optional host
   #Intent;
      package=\[string\];
      action=\[string\];
      category=\[string\];
      component=\[string\];
      scheme=\[string\];
   end;
```

Listing 16: Chrome intent template

Web pages can open a scheme with a specific app. That is a bit different from universal link, because with universal link a scheme is opened, not a specific app. With the Chrome intent it is possible to add extra information such as a timestamp. On IRMA the link looks like this:

```
intent:
    //qr/json/{"u":"https://privacybydesign.foundation/backend/irma/
        session/j7MweFY3mYzAGWKKYjEI","irmaqr":"disclosing"}
    #Intent;
        package=org.irmacard.cardemu;
        scheme=cardemu;
        l.timestamp=1612109193199;
        S.browser_fallback_url=https://play.google.com/store/apps/
            details?id=org.irmacard.cardemu;
    end
```

Listing 17: IRMA Android browser link example 2

It opens *//qr/json/DATA* with the package *org.irmacard.cardemu* with the scheme *cardemu*. The package is the package name of the IRMA app and the scheme is the scheme registered by the IRMA app. Extra information is added, such as a timestamp and a fallback url in case the irma app is not installed.

QR-codes that consist of a URI can easily be supported. Those can be passed to the system to open. The system will try to find an app that is installed that can open the URI.

IRMA QR-codes do not contain a URI. IRMA QR-codes contain JSON objects. These JSON objects always contain the keys *"u"* and *"irmaqr"*. Using pattern recognition, it can

be determined whether the QR-code content is an IRMA QR-code or not. If it is indeed an IRMA QR-code, a deep linking URI can be created. This URI can then be passed to the operating system. For this to work, it should be known how the data in the QR-code should be used to create a deep link. For IRMA, this is pretty simple. The link is as follows: *irma://qr/json/QR-CODE-CONTENT*.

Unfortunately, it is not possible to open Rabobank QR-codes. It is not possible to create a deep link from the QR-code. This is because the QR-code and the deep link which is used on mobile devices do not have a shared part. Theoretically, it would be possible to support the DigiD app because the content of the QR-code is a deep link. However, on Android the scheme in the QR-code is not registered by DigiD. The scheme supported by DigiD is *digid*. This was found using an app decompiler. Using this scheme will start the DigiD app. The app immediately crashes however. It is not known whether DigiD QR-codes work on an iOS device.

In summary, the system supports two kind of QR-codes. The first one is IRMA QR-codes which are supported by pattern recognition. The second one is QR-codes that consist of a URI. In order to open actually open an app, there should be an app installed that registers to the *scheme* of the URI. In other words, the QR-code should consist of a deep link and an app should be installed that can be opened with the deep link.

4.6. USAGE

Overall installation and usage of the system from a user perspective is quite simple. The user can install the app and plugin from the platform associated store[1]. After installation, the user opens the app, goes to settings, enters an encryption key and mails the registration token to themself (image 4.12). Next, the user opens the plugin settings and enters the encryption key and the just received registration token (image 4.13).

The system is now set up and ready to be used. When the user wants to send a QR-code that is currently present on the screen to the phone, they simply click on the plugin button (image 4.14). A small dialog with basic info is presented to the user and the QR-code is sent to the phone. QR-codes are never scanned and sent to the user without an explicit user action. The QR-code can be opened by the user by clicking on the notification received on the phone (image 4.15). The QR-code is also never opened without an explicit user action. The receiver app tries to open an app with the QR-code. If that is not successful, an error message is presented to the user. Currently, the app supports two kinds of QR-codes. The first one is QR-codes uses for the IRMA app. The second one is any QR-code that contains a valid URI. That means any universal URL or HTTP(S) URL can be opened. Since there is no user account, all data can be deleted by removing the app and plugin.

[1]The app and plugin are currently not present on stores.

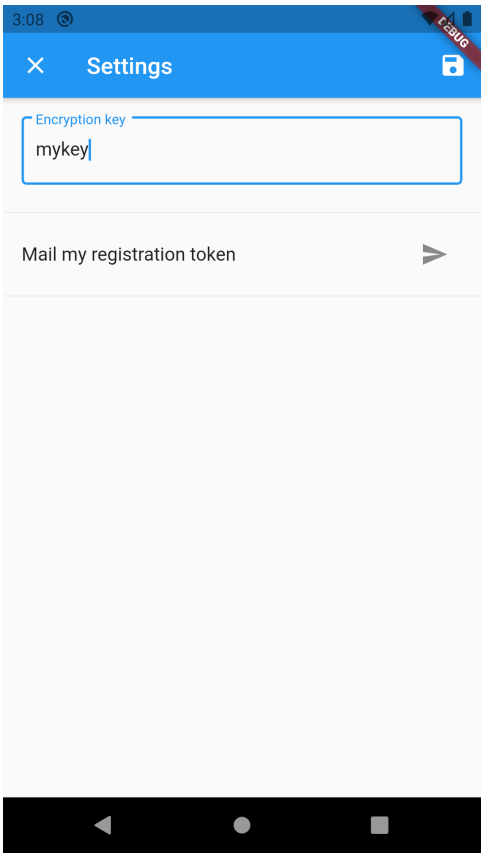

Figure 4.12: Setup app

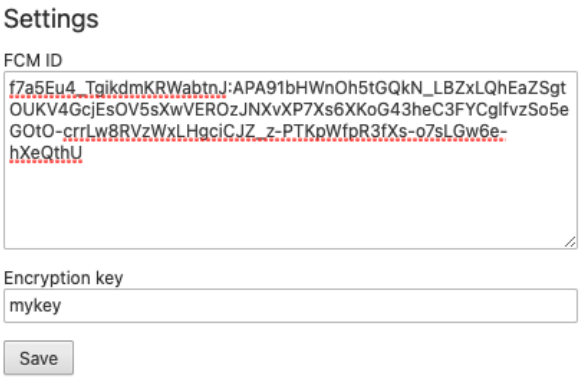

Figure 4.13: Setup plugin

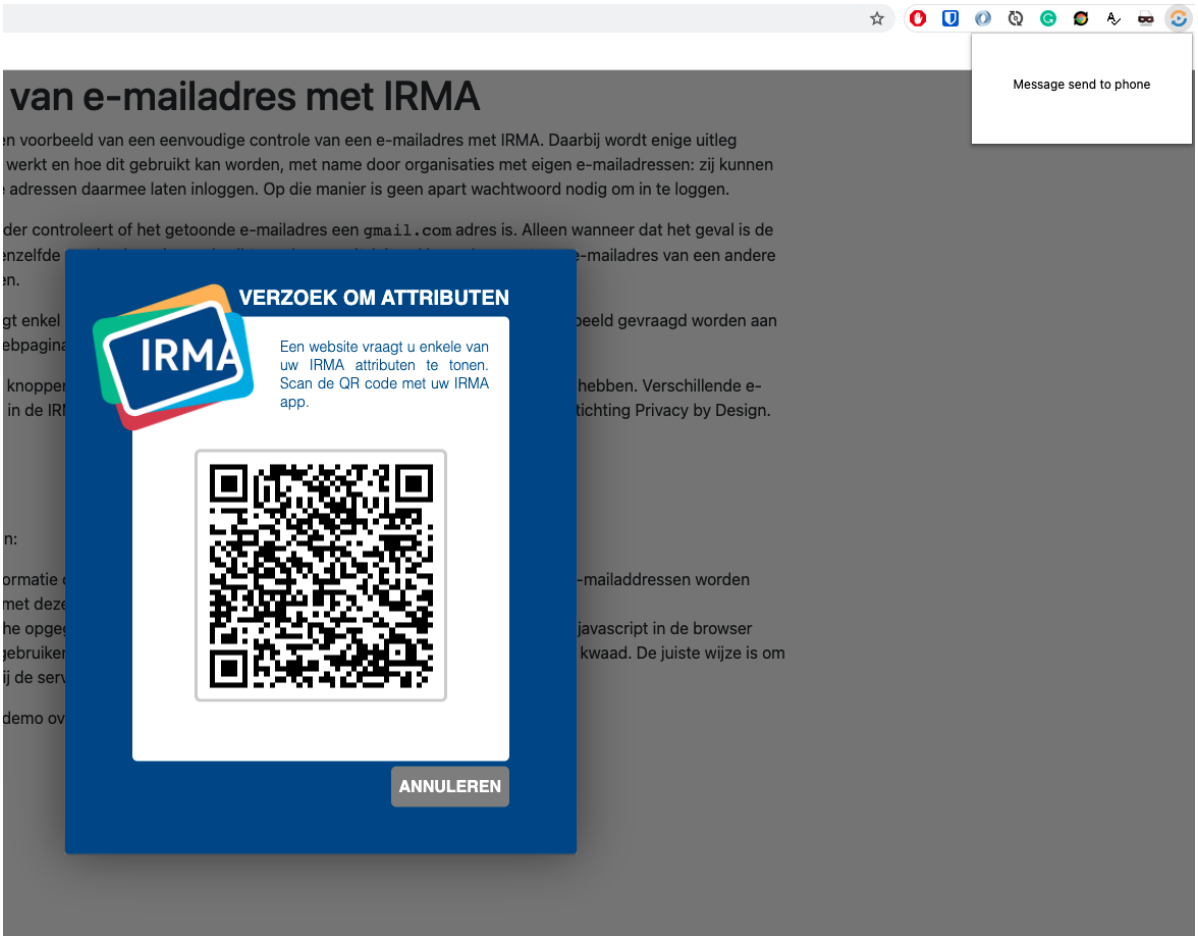

Figure 4.14: Usage plugin

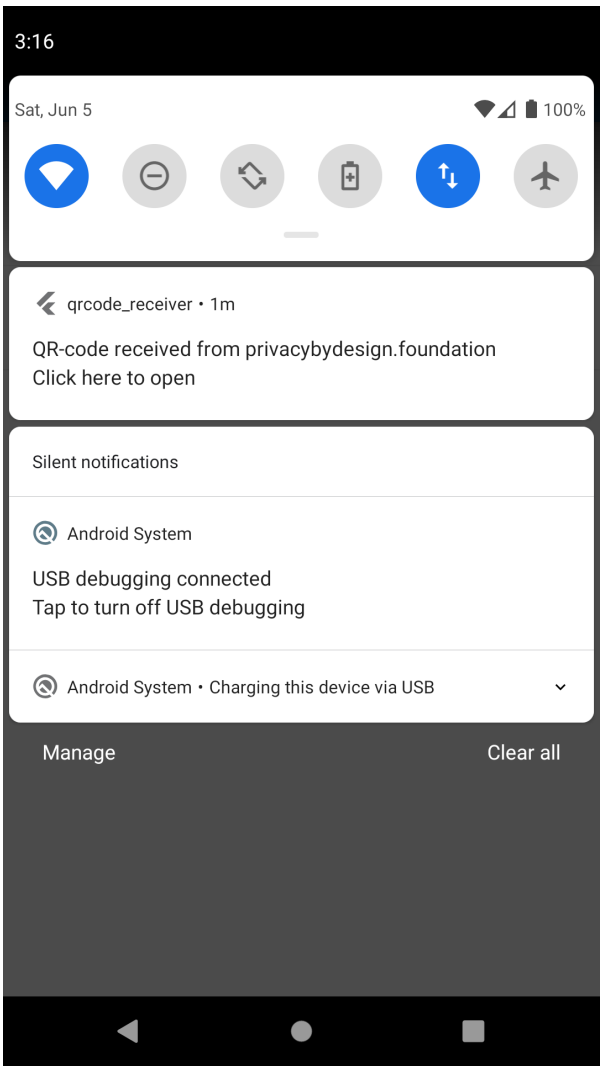

Figure 4.15: Usage app

4.6.1. User trust

Just as with any system and piece of software, the user has to trust it. For example, the user has to trust that the system will store data in a secure way. Another example is that the system will not abuse certain functionality, though this can partially be prevented with permissions. Users with the technical knowledge can inspect the source code and build the app and plugin themself.

The first thing the user should trust the QR-code sending system with is not intercepting the QR-codes. Theoretically, it would be possible for the system to intercept the QR-code. This could have far-reaching consequences depending on the content of the QR-code. For example, a QR-code containing a user session is much more interesting than a simple URL to a web page such a *google.com*. In the case of IRMA, this could potentially result in the theft of the attributes that should be issued to the user.

Next, the user should trust that plugin does not spy on the user. The user is not informed by the browser when a screenshot is taken. So it would be possible to take a screenshot every second and upload it to a server. Since the plugin is active on every tab, it would be possible to record every open tab. Since making screenshots requires the *"all_urls"* per-

mission, it would be possible to make HTTP requests to any URL, again without the user noticing. Browser manufacturers could prevent this by improving the permission system. Last, it would be possible for the plugin to inspect the page the user is visiting and make modifications to it.

The user should also trust that the system will not analyse user data. Since the hostname of the website the QR-code is presented on is not encrypted and the IP address is also known, it would be possible to analyse which web pages the user visits. This could be interesting information to advertisers for example.

4.7. OTHER FINDINGS

This section contains finding that are not directly related to the research question, but are relevant to the research as a whole.

4.7.1. PUSH NOTIFICATIONS

Though fundamentally the same, there are differences between push notifications on iOS and Android [Firebase, 2021]. The Flutter Cloud Messaging plugin tries to solve these differences as much as possible [FlutterFire, 2021]. FMC makes a distinction between *Notification Messages* and *Data Messages*. As their names imply, *Notification Messages* are presented in the form of a notification and *Data Messages* are not. Data messages are considered *low priority*. That means these messages do not wake up the device. However, the priority can be changed. This does still not guarantee delivery.

The Flutter plugin makes a distinction between three different states. These states are *Foreground*, *Background* and *Terminated*. Foreground messages arrive when the app is in the foreground. Messages arrive within the app and by default, there is no notification displayed. Background messages arrive when the app is in the background, but is still active. Because the message is received in the background, it can not do anything UI related. For example, it is not possible to bring the app to the foreground or bring another app to the foreground. When the message contains a *notification property*, The operating system will intercept the message and present a notification to the user. When clicking this notification, the app is opened. Terminated messages are similar to background messages. They occur when the app has been terminated.

Since most users will probably not run the app always in the foreground, the app should be able to handle messages in the background. Unfortunately, it is not possible to bring any app to the foreground when a background message has been received. However, it is possible to use the notification option. The operating system can show a notification with the text: "QR-code received from irma-demo.com. Click to open" When the user clicks on the notification, the app is opened, processes the QR-code content and start the IRMA app.

In order to send a message to a specific device, a registration token is required. This token is unique but can change in certain cases. For example, it changes when the user re-installs the app or deletes user data. The token is obtained in the app when it registers for push notifications. The token should then be used by the browser plugin. It would be inconvenient for a user to type in this token every time the plugin is used since it can be pretty long. The first token received was 133 characters long. Copying and pasting every time would save a bit of time, but is still more inconvenient than simply scanning a QR-code with your phone. Therefore, it is not a solution to the problem we are trying to resolve.

From a user perspective, it would be desirable to store the registration token. That way, the user would only have to enter the token the first time. Browser plugins have the ability to store a small amount of data. A browser plugin can either use *sync storage* or *local storage*. Local storage is local to the machine the extension was installed on and sync storage is synced across instances of that browser the user is signed in to. Storage is scoped to a browser plugin, so plugin A cannot access the storage of plugin B. Storage is not encrypted though, so it should not be used for confidential data.

Registration token generation

Since notifications are sent to a specific registration token, it would be great for an attacker to register their device with the token of a victim. That way, notifications are sent to the attacker instead of the victim. Therefore, it would be good to understand how the registration process works.

As stated above, registration tokens are unique but can change. This can be tested on an Android device by simply deleting the app data and check if the token changes. Wiping data three times results in three unique tokens.

```
eQxwAfbQSJufWtVoAErJn_:APA91bG-tceQuufgraGVftPTF14X2ZraBaY6SdQ1xyTwSt
76EdUNttPdO3vccKpfSrgEAJCck-rmWUtuytBzmkcG8qbrzpw8Pvn-yGUE-8dEAryIJt1
wu4RTRVpFHRBoXTmkxrdJUbaU
```

Listing 18: Example registration token

When disabling network traffic upon opening the app for the first time, it is impossible to obtain a registration token. Therefore it seems a token is generated or at least verified on the Firebase server. Using the *network profiler*, is it possible to inspect network traffic for your app. Unfortunately, not all traffic can be monitored. For example, a web socket connection will not show up. Also, traffic of other apps is not visible. The registration token does not show up in the network traffic. However, part of it does. The registration token seems to consist of two parts separated by a colon. Upon opening the app for the first time, an installation request is made with the first part of the registration token.

```
{
  "fid": "eQxwAfbQSJufWtVoAErJn_",
  "appId": "1:411431974280:android:d48de4e6730c3aeddd363d",
  "authVersion": "FIS_v2",
  "sdkVersion": "a:16.3.5"
}
```

Listing 19: Installation request (Request 1)

```
{
  "name": "projects/411431974280/installations/eQxwAfbQSJufWtVoAErJn_",
  "fid": "eQxwAfbQSJufWtVoAErJn_",
  "refreshToken": "2_B6ko6yFngBl3DVInS0O_6Ujwsorz7u_
      7EFwqWt6V4QirwT_K2VkBikYeIRxFuwSM",
  "authToken": {
    "token": "eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6IjE6NDE
        xNDMxOTc0MjgwOmFuZHJvaWQ6ZDQ4ZGU0ZTY3MzBjM2FlZGRkMzYzZCIsImV4c
        CI6MTYyMjM4MTEyMCwiZmlkIjoiZVF4d0FmYlFTSnVmV3RWb0FFckpuXyIsInB
        yb2plY3ROdW1iZXIiOjQxMTQzMTk3NDI4MH0.AB2LPV8wRAIgXW4nP48inoSfb
        Uh9ukMknj5NaN6yoJlldvXYEMdZ0goCIHkj1zVenA7knnYMK6QWjRzBDhzLVuX
        Zoz3R0qKSAjQN",
    "expiresIn": "604800s"
  }
}
```


Listing 20: Installation response (Response 1)

Repeating Request 1 will result in another *fid* returned in the response. Since not all traffic can be monitored and the source code is obfuscated, it is not possible to tell how registration tokens are generated and registered. Using a proxy, more information can be obtained. This will only work for older devices. Starting from Android Nougat proxies can only be used for apps that you control. Using this proxy, a second request can be seen that is fired immediately after receiving the one above. Since this request is not visible in the *network profiler*, it is probably fired by another app or the OS. On an older version of the OS, all traffic can be seen.

In the second request, the *fid* and token obtained from the first request are used to obtain the registration token. When executing request 1 for a second time, the authentication token obtained cannot be used to obtain the registration token.

The process of generation and registering a registration token probably works as follow. First, the client generates a *fid*. Using this *fid*, a session is started on the server and an authentication token is returned to the client. If the *fid* is already known by the server, another session is started and another *fid* is returned to the client. Using the authentication token and the *fid*, the registration token is obtained.

Since the registration token is obtained from the server, an attacker cannot simply register that token with their device. To get the registration token from the server, an authentication token and *fid* is required. This authentication token can be obtained from the server, using the *fid*, only once.

4.7.2. Cross-platform encryption

Both Flutter and browser plugins do support AES-GCM, whether through native functionality or libraries. However, there are subtle differences in implementation and terminology. The WebCryptoAPI, which is accessible by the browser plugin, uses a *salt* for the password key and an *iv (initialization vector)* to create the cyphertext. The Flutter cryptography library does not use a salt or an iv but uses a *nonce*. Also, the WebCryptoAPI uses a *tag* while

in Flutter, it is called a *MAC (Message Authentication Code)*.

A MAC or tag can be used to authenticate the message. This MAC or tag is handled differently in Flutter and in the WebCryptoAPI. When encrypting using the WebCryptoAPI, the MAC is directly appended to the ciphertext. Flutter, meanwhile, expects the mac to be a separate argument. It required the developer to strip the mac from the ciphertext before decrypting it.

4.7.3. IRMA SECURITY ISSUE

During the development of the QR-code sending system, a severe security issue was found in the IRMA app. It is possible to get access to a subset of attributes by simply connecting a cable. Using USB debugging on an Android device allows someone to see logging of the system, apps and flutter. When the IRMA app starts, that means it was killed by either the system or the user, a subset of attributes is printed to the console. Only a subset of the attributes is displayed because the logline is truncated because it is too long. To someone who has more knowledge of how Flutter logging works, it might be possible the get the full logging line and therefore, the full list of attributes.

It is interesting to see the logging occurs right after starting the IRMA app. At this point, the user has not entered their PIN, yet the app has somehow access to the attributes. Even when there was no internet connection and the system date was changed to a year later, the attributes were still printed onto the console. That means either the attributes are stored in plain text or a password to protect them are somewhere stored on the device. This logging issue was reported and it seems the issue has now been resolved[irma, 2021].

5

DISCUSSION

To omit manually scanning a QR-code with a mobile device, a system is created that detects a QR-code and sends it to to the mobile device instead. This system consists of a browser plugin, a trusted server, push notifications and an app. The push notification functionality is similar to Google Prompt and FingerKey described in RQ1.

Without using the QR-code sending system, the process is as follows for scanning an IRMA QR-code. When the user presses the login button, a QR-code is displayed. The user should now take their phone, probably unlock it, search for the IRMA app and open the IRMA app. After unlocking the IRMA app, the user presses the button to scan a QR-code; scans the QR-code on the website and agrees to disclose the attribute. The user is now logged in. With the QR-code sending system, steps are omitted. Assuming the user has set up the system correctly. When the QR-code is present, the user clicks on the browser plugin. Next, the user picks up their phone, probably unlocks it and click on the notification created by the QR-code receiver app. The IRMA app now opens and from this point on, the process is equal to the process without the QR-code sending system. In summary, the following steps have been removed: searching for the IRMA app, opening the IRMA app, clicking on the QR-code scanning button and scanning the QR-code.

Though the impact on privacy and security is minimal, there are possible improvements. The first thing that can be improved is the browser plugin permission system. Some of the current permissions are too broad. An example is the *"<all_urls>"* permission. This permission is required to allow the plugin to run on any domain, make requests to any domain and to make screenshots. As a result, if the *"<all_urls>"* permission is required to make a screenshot, the plugin can automatically run on any domain and make requests to any domain. Also, if the plugin can run on any domain, it can take screenshots and therefore spy on the user by taking a screenshot every second or so. To improve the security and privacy of browser plugins, the permission system could use an overhaul. Before that, some research needs to be done on the subject of browser plugin APIs and their permissions.

Currently, not all data is encrypted. As described in the Results section, the notification title and notification message cannot be encrypted because those are intercepted and presented by the system. A possible solution would be to use data messages instead of notification messages. According to the documentation, data messages are less reliable than notification messages and the delivery of data messages can not be guaranteed. Unfortunately, the documentation does not describe how reliable both notifications are. It would

be interesting to see how reliable both notifications are and if data messages can be used for the QR-code sending systems.

The QR-code sending system can be improved by supporting more QR-codes. Currently, the system support QR-codes which content are valid URIs and IMRA QR-codes. Obviously, not all QR-codes contain a valid URI. Examples are QR-codes for the Rabobank app, but also for the IRMA app. There are two options when it comes to supporting more QR-codes. The first option is to recognise patterns in the QR-code content. This solution is applied for IRMA QR-codes. Those QR-codes always consist of a JSON object with always the same two attributes. The second option is for system developers that use QR-code to use URIs as QR-code content. For example, if the Rabobank QR-code contained a URI, it would be supported by the QR-code sending system. A reason for Rabobank not using URIs in their QR-codes could be security. Additional research could be performed to understand why system developers choose not to use URIs and how the QR-code sending system should be modified to support QR-codes that do not contain URIs.

Another possibility to omit the scanning of QR-codes is not to use QR-codes at all. Developers could implement the same functionality as the QR-code sending system in their apps. Let us take the Rabobank app for example. When the user wants to confirm a payment, a QR-code is presented on the screen, which the user has to scan with the app. On both the client and the app, Rabobank should know the identity of the user. Obviously, you cannot confirm payments without confirming your identity. That means a user would have to sign in first. So instead of presenting a QR-code, a push notification is sent to the Rabobank app. The user clicks on the notification in the notification bar on the mobile device to confirm the payment. This is comparable to how Google Prompt works. A pin code can be used to confirm the identity of the user. However, push notifications do introduce a dependency. Depending on push notifications only might not be a rugged solution. Another option is to create a short-lived session on the server. If the user opens the app within a short time span, the payment can be confirmed without scanning a QR-code.

The functionality of both the plugin and app is limited and could be extended. Within the plugin, it is not possible to register multiple devices. This could be a useful addition. From within the app, it is not possible the see where the QR-code was sent from. It could be useful to label senders.

It is clear some steps of the scanning process are omitted by using the QR-code sending. Yet, it is not clear whether the system is more convenient to use for users with visual impairment. It would be useful to perform a case study with actual users to see if and how the accessibility of the system can be improved.

From a privacy and security aspect, little changes for the user. However, for system developers that use QR-codes, there are some changes. Previously, developers could be fairly sure that a user who scanned a QR-code was actually behind their computer screen. Of course, one could take a photo of it and send it to someone else. The latter then has to print it or display it on another screen before scanning it, which is a bit of a hassle. With the QR-code sending system, the user is not tied to their computer screen. Wherever you are, a notification is sent to your mobile device. Measures have been taken to prevent opening QR-code without the user knowing it. To scan the QR-code with the browser plugin and open it with the app, a user must perform an action. Though measures have been taken, it becomes easier to scan a QR-code without the user being behind the computer screen with the QR-code sending system.

It is interesting to see Rodrigues et al. use the NMH (native messaging host) to facilitate communication between the browser plugin and the Hardware Cryptography Server. Rodrigues et al. claim the browser plugin cannot directly communicate with the hardware cryptography server, as it is limited to the page's DOM[Pimenta Rodrigues et al., 2020]. However, we found the browser plugin can communicate with a server. As a matter of fact, the QR-code scanning plugin communicates with the trusted server to create push notifications.

6

Conclusion

In this research, a QR-code sending system is created to omit the scanning of a QR-code with the IRMA app without affecting the state of security and privacy. This system is built in a generic way so it does support not only IRMA QR-codes but also any QR-code that contains a URI.

We have seen other systems use different methods to facilitate communication between the browser and mobile device. These methods are Push notifications, QR-codes and direct connect.

The process of connecting an IRMA app to an IRMA server using QR codes is quite simple. Upon a user action, Upon a user action, a session is obtained from the IRMA server. The IRMA server returns a session token and an URL to the server. These are used to create the QR-code. The QR-code presented on the screen is just a simple JSON object with two attributes. This object is passed to the app using *Universal links* or by scanning the QR-code with the IRMA app. The QR-code sending system sends QR-codes directly to the phone without scanning the QR-code using the mobile camera.

The QR-code sending system consists of four main components. These are a browser plugin, a trusted server, a notification service and an app. The system also contains smaller components used for encryption and decoding QR-codes. The browser plugin can be used in the browser to take a screenshot. The plugin uses a QR-code decoding library to get the content of the QR-code from the screenshot. The plugin sends the QR-code and some metadata to the trusted server. The trusted server contains an API key for the notification server and sends the message to the notification server. The notification server is an external service that provides push notifications. Using a push notification, the message is pushed to the mobile device. A mobile app receives the message, decrypts it and tries to open another app with it. If the content is a valid URI, it is passed to the system to open it. If the content is an IRMA JSON object, the IRMA app is opened with it. In other cases, an error message is shown. In essence, the QR-code sending system does exactly what normally a user has to do. It scans a QR-code and opens an app with it.

The impact on privacy and security is minimal. The content of the QR-code is encrypted to ensure authenticity, integrity and confidentiality. Privacy could potentially be further improved by also encrypting the notification title and message as described in the discussion section. A STRIDE analysis has been carried out on the system. Based on the analysis, the system was further improved. Also, based on the analysis, four actions were found that

are interesting to attackers. These are: intercept notification data, intercept QR-codes, send QR-codes and send notification. The most common threat is an evil maid attack or a man in the middle attack.

The system can be extended to support more app than just the IRMA app in two ways. The first option is for the receiver app to recognise more patterns in the QR-code content. Of course, this op requires the QR-code content to have some kind op pattern and requires a modification to the receiver app. The second option is for developers of other app that use QR-codes, to make the QR-code consist of a URI. If this URI is used for *deep linking*, it will be supported by the QR-code sending system

The system as a whole works and is usable for QR-code for IRMA or any QR-code that consists of a URI. The system has a minimal impact on privacy and security. However, improvements can be made as described in the discussion section.

7

REFLECTION

I am pretty happy with subject of this thesis and the product as well. Scanning QR-codes bother me somewhat, which made it more interesting to find a solution. Therefore, I am a bit disappointed the solution does not work for every QR-code. Another thing that interested me was IRMA itself. I had read some news articles about IRMA on technology websites, but did not know much about it. I do not claim to be an export on the subject of IRMA, but I am pretty sure I know more than the average user. Though I think IRMA is not widely used, I was surprised to see my wife had to use the IRMA app to fill in some questionnaire. What I really liked was the variety of work. I have been busy with Irma, browser plugins, push notifications, mobile apps, flutter, encryption and STRIDE analyse. In my opinion, I have somewhat of a short attention span, unless something is new or exciting. In the latter case, which occurred often with the big variety of work, I have somewhat of a hyper-focus.

Between the *VAF* and *AF* is spent some time developing a browser plugin. I did this because I thought it was interesting and it solved a problem as well. During the development I learned a lot about browser plugins. Documentation and code examples is not that great. Because of this experience I was able to develop the QR-code sending plugin a lot faster.

Initially, the plan was to first do all the research and then built something. I am glad I changed that and had a healthy combination of researching and building. Research results were the input for the building and vice versa. The best example of this is the threats table 4.3 in the results section of RQ4. Threats were mitigated and as a result new threats could occur. Another advantage was that I would never have to work on something that was less interesting for very long. As with anything, some think are just more interesting than others.

Something what really helped me was the review of the threats table with a fellow student. It not only gave me some insights, but also somewhat assured me I was doing the correct thing. Besides that, it was also nice to catch up and exchange experiences.

I think writing is not my strongest skill. I am perfectly able to form clear sentences. However, I have the tendency to keep it short. As a result, details are sometimes omitted, which sometimes makes it hard to understand the whole story. It helps me to immediately write things down when they are still fresh in my mind.

What really surprised me is how powerfully browser extensions can be. Plugin have access to functionality that is not available to web pages. An example of such an API is the screenshot API. Unfortunately, that also means these APIs can be used for malicious

purposes. I was for example able to spy on network traffic. I knew there were some risks, but not this big. I think many users do not know about the right either.

BIBLIOGRAPHY

Apple. Preparing your ui to run in the background. https://developer.apple.com/documentation/uikit/app_and_environment/scenes/preparing_your_ui_to_run_in_the_background, 2021a. Accessed: 2021-03-07. 16

Apple. Notifications. https://developer.apple.com/notifications/, 2021b. Accessed: 2021-03-06. 14

Beizhedenglongdev. Qr code reader. https://chrome.google.com/webstore/detail/qr-code-reader/likadllkkidlligfcdhfnnbkjigdkmci, 2021. Accessed: 2021-02-27. 13

Bitwarden. Open source password management for you and your business. https://bitwarden.com, 2020. Accessed: 2020-08-02. 7, 14

Bitwarden-FAQ. What happens if bitwarden gets hacked? https://bitwarden.com/help/article/what-happens-if-bitwarden-is-hacked/, 2020. Accessed: 2020-09-04. 8

Chromium.org. Issue 487422: Webrequest api: allow extensions to read response body. https://bugs.chromium.org/p/chromium/issues/detail?id=487422, 2021. Accessed: 2021-02-15. 12

Chrome Developers. Android intents with chrome. https://developer.chrome.com/docs/multidevice/android/intents/, 2021. Accessed: 2021-05-23. 33

DigiD. Digid. https://www.digid.nl/, 2020. Accessed: 2020-11-28. 7

FingerKeyApp. Fingerkey. http://www.fingerkeyapp.com/, 2020. Accessed: 2020-08-15. 8, 14

Firebase. Firebase cloud messaging. https://firebase.google.com/docs/cloud-messaging/, 2021. Accessed: 2021-03-06. 14, 38

FlutterFire. Cloud messaging. https://firebase.flutter.dev/docs/messaging/usage, 2021. Accessed: 2021-03-20. 38

Steven Furnell. The usability of security – revisited. *Computer Fraud & Security*, 2016(9):5 – 11, 2016. ISSN 1361-3723. doi: https://doi.org/10.1016/S1361-3723(16)30070-7. URL http://www.sciencedirect.com/science/article/pii/S1361372316300707. 3

Google. Sign in with your phone instead of a password. https://support.google.com/accounts/answer/6361026, 2020. Accessed: 2020-08-02. 7, 14

GSMARENA. Phone finder. https://www.gsmarena.com/search.php3?, 2021. Accessed: 2021-03-07. 16

Google Chrome Help. Get your bookmarks, passwords and more on all of your devices. https://support.google.com/chrome/answer/165139, 2021. Accessed: 2021-07-11. 8

hsaifan. Apk parser. https://github.com/hsiafan/apk-parser, 2021. Accessed: 2021-04-26. 32

irma. Irma bugfix commit. https://github.com/privacybydesign/irmamobile/commit/f58dc1d27de3e2630def9367f8f7add9cf5da967, 2021. Accessed: 2021-06-06. 41

IRMA. What is irma? https://irma.app/docs/what-is-irma/, 2021. Accessed: 2021-01-31. 9

Muhammad Asif Khan, Wael Cherif, Fethi Filali, and Ridha Hamila. Wi-fi direct research - current status and future perspectives. *Journal of Network and Computer Applications*, 93:245–258, 2017. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2017.06.004. URL https://www.sciencedirect.com/science/article/pii/S1084804517302230. 16

David A. McGrew and John Viega. The security and performance of the galois/counter mode (gcm) of operation. In Anne Canteaut and Kapaleeswaran Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004*, pages 343–355, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-30556-9. 25

Microsoft. The stride threat model. https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN, 2020. Accessed: 2020-11-28. 5, 23

Mightytext. Mightytext. https://mightytext.net/, 2020. Accessed: 2020-08-15. 8

Mozilla. Browser extensions. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions, 2021a. Accessed: 2021-03-07. 12

Mozilla. tabs.capturevisibletab(). https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/tabs/captureVisibleTab, 2021b. Accessed: 2021-02-27. 13, 19

Mozilla. Sync your firefox on any device. https://www.mozilla.org/en-US/firefox/sync/, 2021c. Accessed: 2021-07-11. 8

Mozilla. Porting a google chrome extension. https://extensionworkshop.com/documentation/develop/porting-a-google-chrome-extension/, 2021d. Accessed: 2021-02-07. 11

Mozilla. webrequest.filterresponsedata(). https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/filterResponseData, 2021e. Accessed: 2021-02-15. 12

Mozilla. Xmlhttprequest. https://developer.mozilla.org/nl/docs/Web/API/XMLHttpRequest, 2021f. Accessed: 2021-02-19. 12

Netmarketshare. Browser market share. https://netmarketshare.com/browser-market-share.aspx, 2020. Accessed: 2020-08-23. 12

Okta. Okta verif. https://help.okta.com/en/prod/Content/Topics/Mobile/okta-verify-overview.htm, 2020. Accessed: 2020-08-02. 7, 14

G. A. Pimenta Rodrigues, R. D. Oliveira Albuquerque, G. D. Oliveira Alves, F. L. L. De Mendonça, W. F. Giozza, R. T. De Sousa, and A. L. Sandoval Orozco. Securing instant messages with hardware-based cryptography and authentication in browser extension. *IEEE Access*, 8:95137–95152, 2020. 8, 16, 44

Privacy by Design Foundation. Irma gebruik. https://privacybydesign.foundation/gebruik/, 2021. Accessed: 2021-02-12. 12, 13

Stackoverflow. Chrome extension to read http response. https://stackoverflow.com/a/48134114/2564847, 2021. Accessed: 2021-02-19. 12

Statcounter. Browser market share worldwide. https://gs.statcounter.com/browser-market-share, 2020. Accessed: 2020-08-23. 12

Roland M. van Rijswijk and Joost van Dijk. Tiqr: A novel take on two-factor authentication. In *25th Large Installation System Administration Conference (LISA 11)*, Boston, MA, December 2011. USENIX Association. URL https://www.usenix.org/conference/lisa11/tiqr-novel-take-two-factor-authentication. 6

W3schools. Browser statistics. https://www.w3schools.com/browsers/, 2020. Accessed: 2020-08-23. 12