Network Security E20, Final Project

Security analysis of the mobile dating app Happn

Peter Marcus Hoveling, Student No.: 201508876

Github: https://github.com/201508876PMH/netsec-final

Guidance counselor: Diego F. Aranha
Due date: January 29 2021

Department of engineering
Aarhus university
Denmark,
IHA.

Contents

1 Introduction

1.1 Introduction

For this project, we are tasked to perform and document the results of a preliminary security analysis of a mobile application. The chosen application should have a sensitive role impacting user privacy to motivate the security analysis. Meaning the application should handle user data of some sensitive nature (financial information of any kind, health records, location data, passwords and other authentication information, etc).

We will be looking at some of the techniques and concepts learned throughout the semester and try to analyse the application in four different aspects:

1. *Software security*: decompiling the application, looking for keywords (such as cryptographic algorithms) and scanning the source code using static analysers.
2. *Network security*: documenting the TLS server configuration and attempting to MITM a connection to capture sensitive traffic. With and without a self signed certificate.
3. *Authentication*: documenting what security mechanisms are used, and how secure they are.
4. *Privacy*: observing relevant privacy characteristics (trackers and integration with social networks).

1.2 Foundation

The app chosen for the project is the local dating app **Happn**. Happn was created 7 years ago, in January 2014, with the chief executive officer Didier Rappaport wanting to give people the power to seize every day opportunities that arose when randomly crossing paths with someone you liked[11]. Happn now has 70M users worldwide, 4.9M messages sent per day and an average of 1.5M new users joining every month[11].

Having such a huge user-base, using trackers for connecting potential matches and the logging of user data is the justification for choosing this application.

Looking at table 1.1, we see the proposed threat model for the Happn app. Expected are the listed policies. We as users are expecting high availability, reliability in finding matches, an insurance that personal data is kept private and the app authenticating the correct users. The way the app achieves this, is by having cryptographic algorithms to encrypt data. Having access control to validate requests and authenticate users. And having a form of cloud computing, to allow users to remotely access their accounts.
Entities who might want to attack Happn could either be competitors, black hat or white hat hackers. Competitors as a means of ruining the reputation of Happn and gaining traction themselves, black hat hackers as a means of ransom money and lastly white hat hackers for helping the creators find critical vulnerabilities.
The attack surface would either be the mobile app, the Happn database/networks, their website and/or third-party services.

Policies:	Availability, Reliability, Privacy, Confidentiality, Authenticity
Mechanisms:	Cryptography, Access control, cloud computing
Threat model:	Competitors Black and white hat hackers
Attack surface:	Mobile app Databases/networks Website Dependent 3d party services

Table 1.1: Happn threat model

From this thread model, we will take a deeper dive into the Happn app and try to draw some conclusions on the software-, network-, authentication and privacy security views.

2 | Software Security

2.1 Decompiling Happn

To get a better understanding of the inner workings of Happn, we firstly want to decompile the app. In essence the act of decompiling, is the reverse of compiling. That is, taking the object code (binary) and trying to recreate the source code from it. We are never guaranteed the original source code, since the success of decompiling depends on the artefacts being left in the object code, which are stripped at the comping process.

Happn exists in an android and OSX version, we will be decompiling the android version since android applications can easily be decompiled through a variety of tools. As a matter of preference we will be using `dex2jar`[20] reasoned its many examples and documentation.

However we firstly need to download the Happn APK file, which we will be downloading from the website `apkpure.com`. The website offers not only APK files from different apps, but offers various versions of them, see figure 2.1.

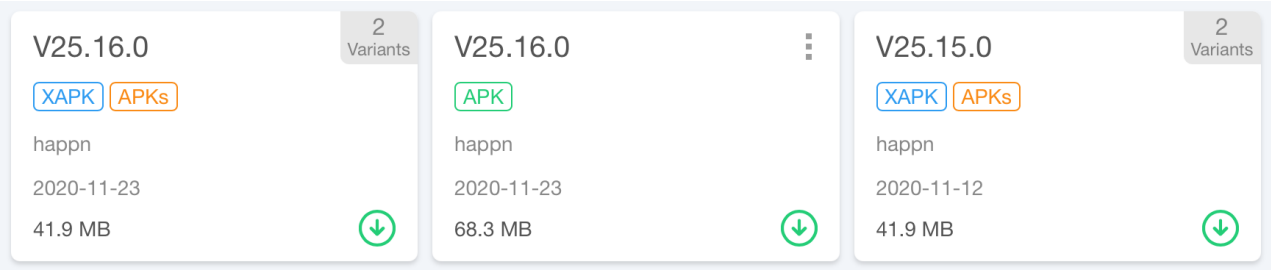


Figure 2.1: Various versions of the Happn app

When trying to decompile via dex2jar, we sadly get the following error as seen in figure 2.2. Reading the error-logs for the decompilation, we are presented with a lot *.txt files with obfuscated java functions. A theory as to why this fails, could be that some APK files include a different structure than what the dex2jar tool expects.

Figure 2.2: Error when decompiling Happn app

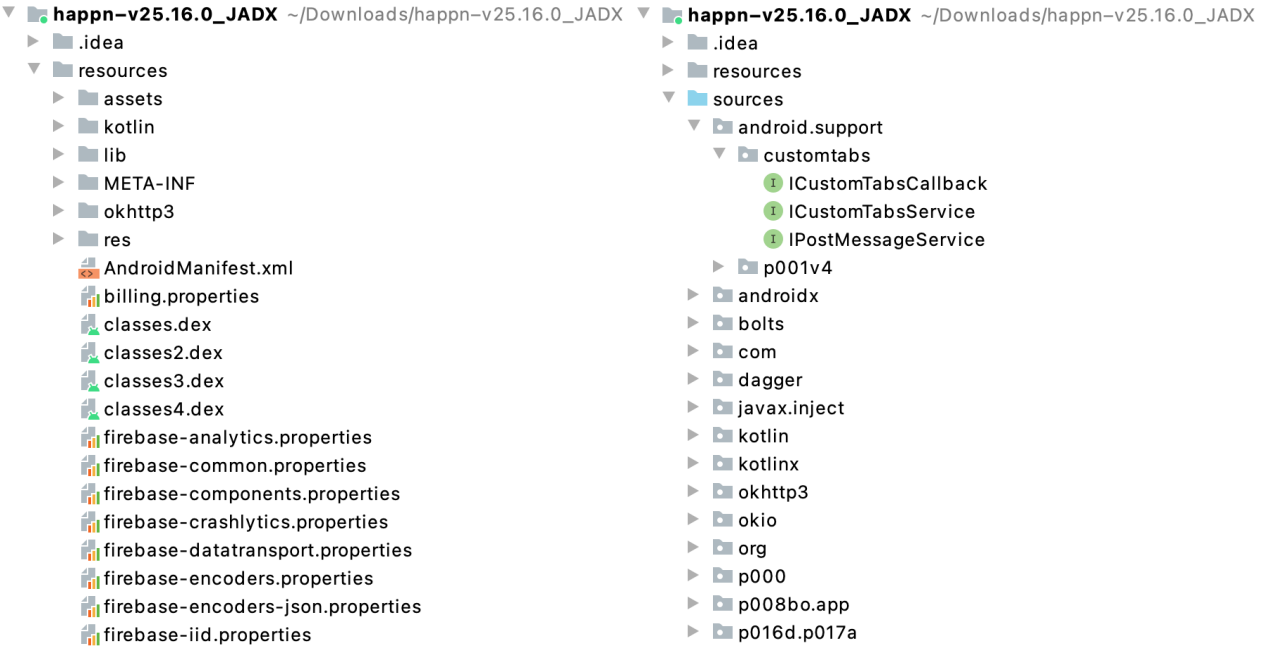
A workaround for this is to try to use the online APK decompiler from `javadecompilers.com`. The website argues that it uses the open-source APK and dex compiler called jadx[12], which is the same tool we previously tried; dex2jar.

After uploading our Happn APK file and awaiting the decompilation, we manage to get the attempted source code for the app as seen in figure 2.3.

Figure 2.3: Happn app decompiled from `javadecompilers.com`

2.2 Obfuscation and readability

Examining the decompiled APK file we see two folders; `resources` and `sources`, *see figure 2.4*. The files found in the resources folder are primarily libraries and assets; pictures, fonts and at times certificates. Whereas the files found in the sources folder, are imported java libraries and the would be original source code for the Happn app.
Taking a deeper look into the folders we see naming like `androidx` and `dagger`. These folder are official libraries and can be overlooked when trying to analyse the Happn app code.



(a) resources folder from decompiled APK

(b) sources folder from decompiled APK

Figure 2.4: Some of the contents inside the resources and sources folder

One might ask themselves, how we would find the correct folders, as many of the folder names seem to have random naming. One way is to open up the search tool and look for keywords we know should exist in the app, example `https`.

Looking at figure 2.5, we've searched for the specified keyword and found what seems to be a valid Happn connection string. Looking at which folder the function is coming from, we can see its from the folder path `sources/com/ftw_and_co/happn/core/dagger/module`. From this, we at least have a finger pointing in the right direction, as we move on to try to analyse the readability of the decompiled APK.

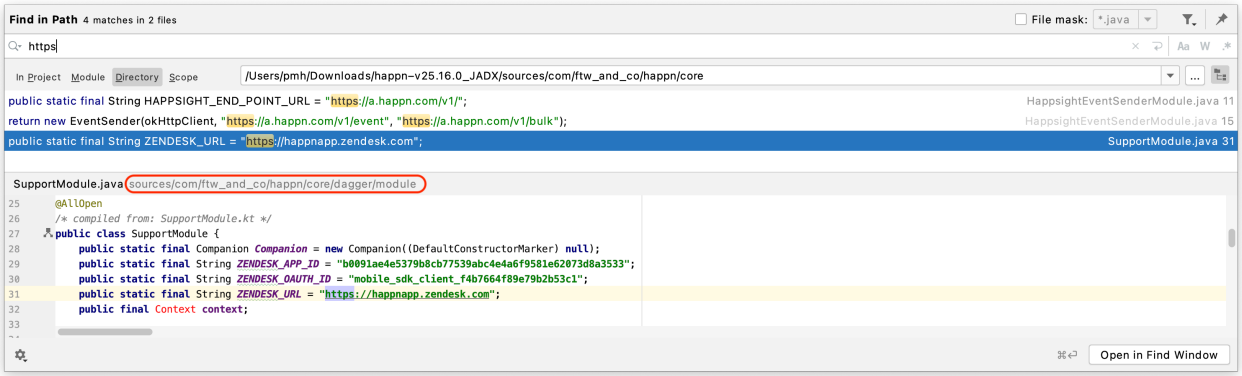


Figure 2.5: Searching project for "https" keyword

Searching more thoroughly we find evidence for code sections not obfuscated, as the one seen in figure 2.6. A simple login function taking four different strings as input, validating the parameters aren't null and setting the local variables to the match the input.

```
public LoginPasswordSignUpGenderPrefsWithSDCDelegate(@NotNull String str, @NotNull Date date,
                                                     @NotNull RegistrationTracker registrationTracker2,
                                                     @NotNull SignUpInteractions signUpInteractions) {
    Intrinsics.checkNotNullParameter(str,  str: "firstName");
    Intrinsics.checkNotNullParameter(date,  str: "birthDate");
    Intrinsics.checkNotNullParameter(registrationTracker2,  str: "registrationTracker");
    Intrinsics.checkNotNullParameter(signUpInteractions,  str: "callback");
    this.firstName = str;
    this.birthDate = date;
    this.registrationTracker = registrationTracker2;
    this.callback = signUpInteractions;
}
```

Figure 2.6: Readable login function

However, since many developers may want to hide details for their implementations, obfuscation of source code isn't hard to come by. Per default android projects have a tool (ProGuard) disabled. Enabling this will obscure and unclear the source code, whilst also shrinking the distributable app package[8]. Looking at figures 2.7, 2.8 and 2.9 we can see that there are signs of obfuscation. The three different functions, all in which have been heavily obfuscated, can be somewhat understood because of there length, however larger functions probably wouldn't.

```
/* renamed from: a */
public boolean mo11143a(C0912cf cfVar) {
    try {
        this.f242e.mo11156a((C0911ce) C0926cp.m384a(cfVar));
        return true;
    } catch (Exception e) {
        AppboyLogger.m1238w(f238a,  str2: "Failed to log location recorded event.", e);
        return false;
    }
}
```

Figure 2.7: Location function obfuscated

```
/* renamed from: a */
public synchronized void mo11424a(Gender gender) {
    if (gender == null) {
        m716c( str: "gender", (Object) null);
    } else {
        m716c( str: "gender", gender.forJsonPut());
    }
}
```

Figure 2.8: Gender function obfuscated

```
/* renamed from: e */
public synchronized void mo11442e(String str) { m716c( str: "country", str); }

/* renamed from: f */
public synchronized void mo11444f(String str) { m716c( str: "home_city", str); }

/* renamed from: g */
public synchronized void mo11445g(String str) { m716c( str: "language", str); }
```

Figure 2.9: Personal info obfuscated

2.3 Dynamic and static analysis

When trying to analyse our Happn app from a software security point of view, we can either choose to run a static analysis or a dynamic analysis. A static analysis is performed in a non-runtime environment, either manually by searching for keywords such as `AES`, `SHA1`, `passwords` etc. or by running static analysis tools. Whereas the dynamic analysis adopts the opposite approach and is executed whilst the program is running. Sniffing traffic, running applications through emulators to see logs etc.

2.3.1 Static analysis

Manually searching through the decompiled source code, we find a lot of endpoints, ranging from the use of Firebase, Facebook, Github, Happn, Bugsnag and some unknown. Some of these endpoints are perfectly acceptable, as having API keys public for client-side applications at times can't be avoided. However for most of them, we would expect limits on their use. Only being callable from an authorized referer.
Looking at at figure 2.10, we see some of the endpoints and uses of broken protocols, which should raise some eyebrows as fx. the `android.backup.api_key` on line 7.

Leaks.xml

```
1  # Google
2  <string name="default_web_client_id">1036899618040-t8viukcf1sjcb08rhtnlrrvqppq2hu8v.apps.googleusercontent.com</string>
3  <string name="google_api_key">AIzaSyCkpVZEA9jq9fgvFV4_9Vt5xj3gW15oIj4</string>
4  <string name="google_app_id">1:1036899618040:android:2b271e73545ad429</string>
5  <string name="google_crash_reporting_api_key">AIzaSyCkpVZEA9jq9fgvFV4_9Vt5xj3gW15oIj4</string>
6  <string name="google_storage_bucket">api-project-1036899618040.appspot.com</string>
7  <meta-data android:name="com.google.android.backup.api_key" android:value="AEdPqrEAAAAIUFApSPDWZgyDx2SuvGFJ62DJKx-M3Fr5-5pT1Q"/>
8  <meta-data android:name="com.google.android.gms.ads.APPLICATION_ID" android:value="ca-app-pub-9959023750977957~5969456027"/>
9  <meta-data android:name="com.google.android.maps.v2.API_KEY" android:value="AIzaSyBdrRXiJUV4PQrB4yTQeBTq25CeMtVYLLo"/>
10
11 # Facebook
12 <string name="default_facebook_app_id">247294518656661</string>
13
14 # Firebase
15 <string name="firebase_database_url">https://api-project-1036899618040.firebaseio.com</string>
16
17 # Unknown
18 <string name="gcm_defaultSenderId">1036899618040</string>
19 <string name="project_id">api-project-1036899618040</string>
20     public static final String AUTHORITY = "app.adjust.com";
21     public static final String BASE_URL = "https://app.adjust.com";
22     public static final String BASE_URL_CN = "https://app.adjust.world";
23     public static final String BASE_URL_IN = "https://app.adjust.net.in";
24     public static final String CLIENT_SDK = "android4.24.0";
25     public static final String FB_AUTH_REGEX = "^(fb|vk)[0-9]{5,}[^:]*://authorize.*access_token=.*";
26     public static final String SHA1 = "SHA-1";
27     public static final String SHA256 = "SHA-256";
28
29 # Github
30 <string name="library_roundedimageview_repositoryLink">https://github.com/vinc3m1/RoundedImageView.git</string>
31 <string name="library_roundedimageview_libraryWebsite">https://github.com/vinc3m1/RoundedImageView</string>
32 <string name="library_roundedimageview_authorWebsite">https://github.com/vinc3m1</string>
33
34 # Happn
35 <string name="popup_deletion_confirmation_happn_url">mailto:stories@happn.com</string>
36     public final String key = "https://api.happn.fr";
37     public final String secret = "FUE-idSEP-f7AqCyuMcPr2K-1iCIU_YlvK-M-im3c";
38     public final String url = "brGoHSwZsPjJ-lBk0HqEXVtb3UFu-y5l_JcOjD-Ekv";
39
40 # Bugsnag
41 <meta-data android:name="com.bugsnag.android.API_KEY" android:value="6d69443887e78067cdb26ebefd0a8faa"/>
42     public String endpoint = "https://notify.bugsnag.com";
43     public String sessionEndpoint = "https://sessions.bugsnag.com";
```

Figure 2.10: Some of the endpoints manually found from static analysis

Trying the many different endpoints and API keys is a longer process, however having experience in Google API keys, one will know that the generated key has a maximum number of calls, before requiring payments. Creating a simple website and trying a non-activated key, we get an error loading the web page, as seen in figure 2.11 & 2.12.
The Google API key found from our static analysis in figure 2.10 line 9, actually shows not to be restricted, as seen in figure 2.13. The result of this is that in theory, one could either create a lot of API requests and max out Happn' paid requests, effectively shutting down the world-map functioning, or generating a large bill to pay!

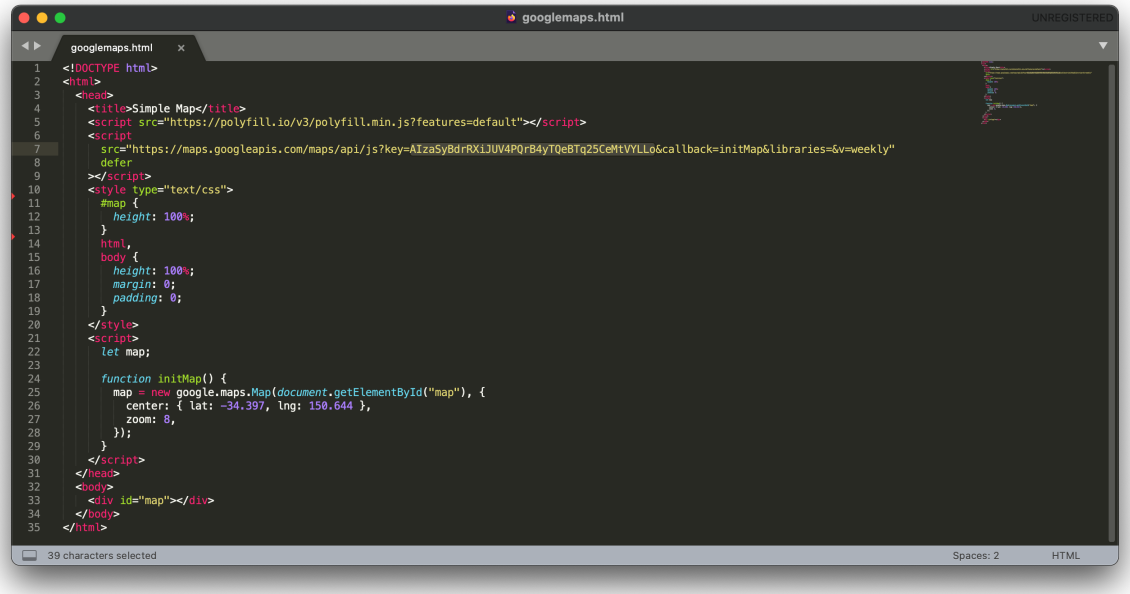


Figure 2.11: Simple website to test Google maps API key[10] (viewed on line 7)

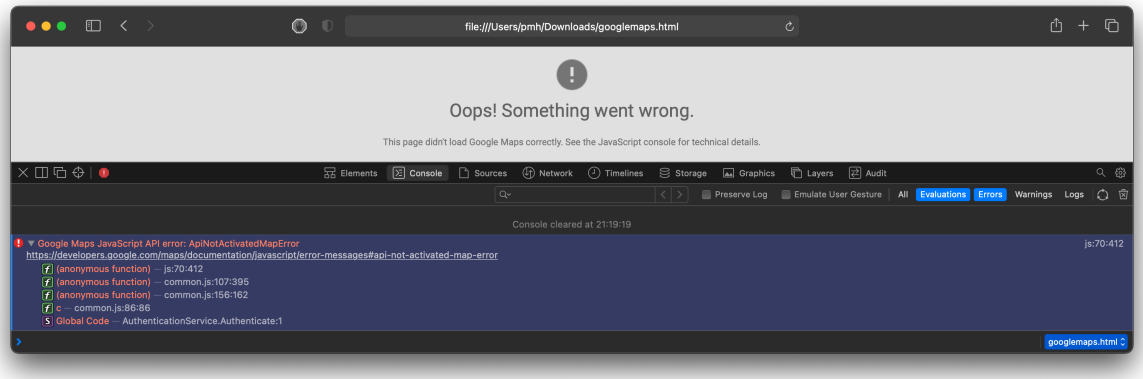


Figure 2.12: Self generated Google maps API key responding with a "not activated" error

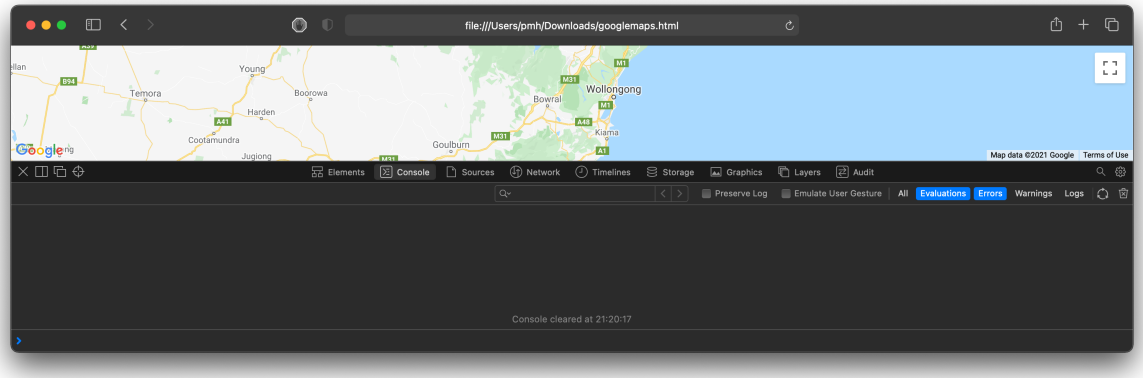


Figure 2.13: Happn API key readily available for all, not restricted to an API

Another, maybe more thorough way of statically analysing code, is through static analysis tools as Qark[15]. The Qark tool is designed to look for several security related android application vulnerabilities, either in source code or packaged APKs. Running the tool is as easy as cloning down the repository, calling the main Qark script and giving the `-java` flag, see figure 2.14.

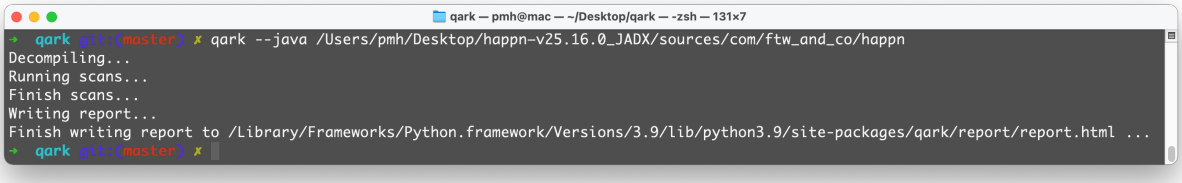


Figure 2.14: Picture of Qark analysing and producing an error report

Opening up the generated error report, we get the following as seen in figures 2.15, 2.16, 2.17 and 2.18. Some of these issues should be overlooked, as they are simply not a security risk. However using a static analysis tool is a great way to potentially find security critical flaws quickly and automated.

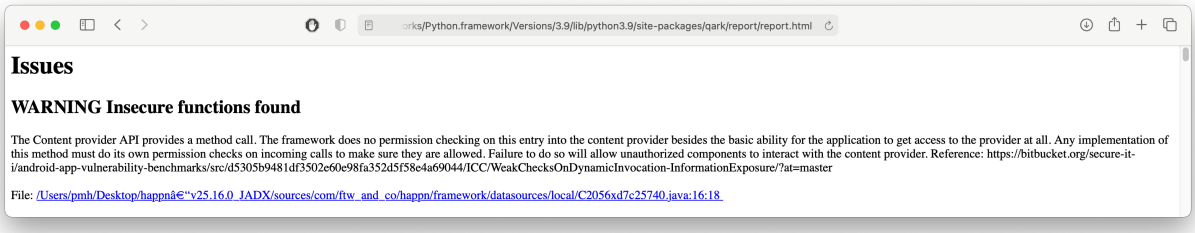


Figure 2.15: Opening the report.html

INFO Potential task hijacking

FLAG_ACTIVITY_NEW_TASK - intent flag is set. This results in activity being loaded as a part of a new task. This can be abused in the task hijacking attack. Reference: https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-ren-chuangang.pdf

File: /Users/pmh/Desktop/happnâ€“v25.16.0_JADX/sources/com/ftw_and_co/happn/p025ui/login/signup/SignUpValidationInformationActivity.java

Figure 2.16: Qark finding a potential task hijacking and pointing to the file

INFO Potential API Key found

Please confirm and investigate the API key to determine its severity.

File: /Users/pmh/Desktop/happnâ€“v25.16.0_JADX/sources/com/ftw_and_co/happn/p025ui/spotify/TrackEntry.java:67:0

Figure 2.17: One of many found API keys

VULNERABILITY Dynamic broadcast receiver found

Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter. Reference: https://developer.android.com/reference/android/content/Context.html#registerReceiver(android.content.BroadcastReceiver, android.content.IntentFilter)

File: /Users/pmh/Desktop/happnâ€“v25.16.0_JADX/sources/com/ftw_and_co/happn/connectivity/data_sources/locals/C1820x9cb861a2.java:26:28

Figure 2.18: A vulnerability found by Qark

2.3.2 Dynamic analysis

Having found security issues via manual and automatic static analysis, we turn to the dynamic analysis of the Happn app. The tool we will be using is called Mobile Security Framework, shortened MobSF[19], and is an automated, all-in-one mobile application pen-testing framework.

An easy `git clone`, running the `setup.sh` script and calling `./run 127:0.0.1:8001`, starts the application ready to analyse the Happn app. Figure 2.19 shows MobSF running in the terminal whilst figure 2.20 shows the MobSF hosted website.

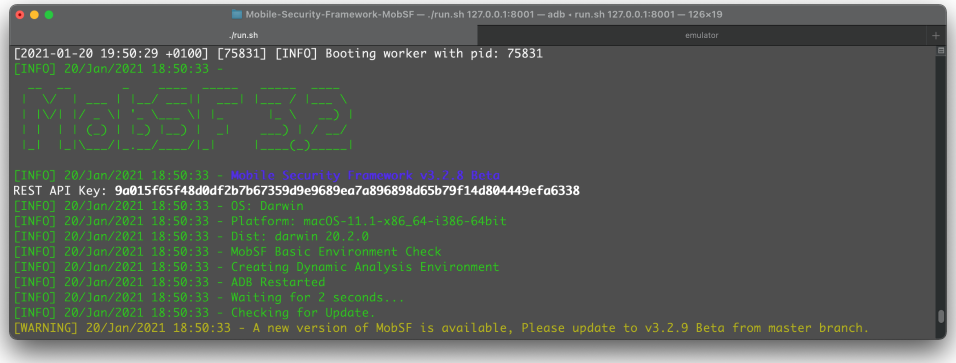


Figure 2.19: Terminal view of MobSF running

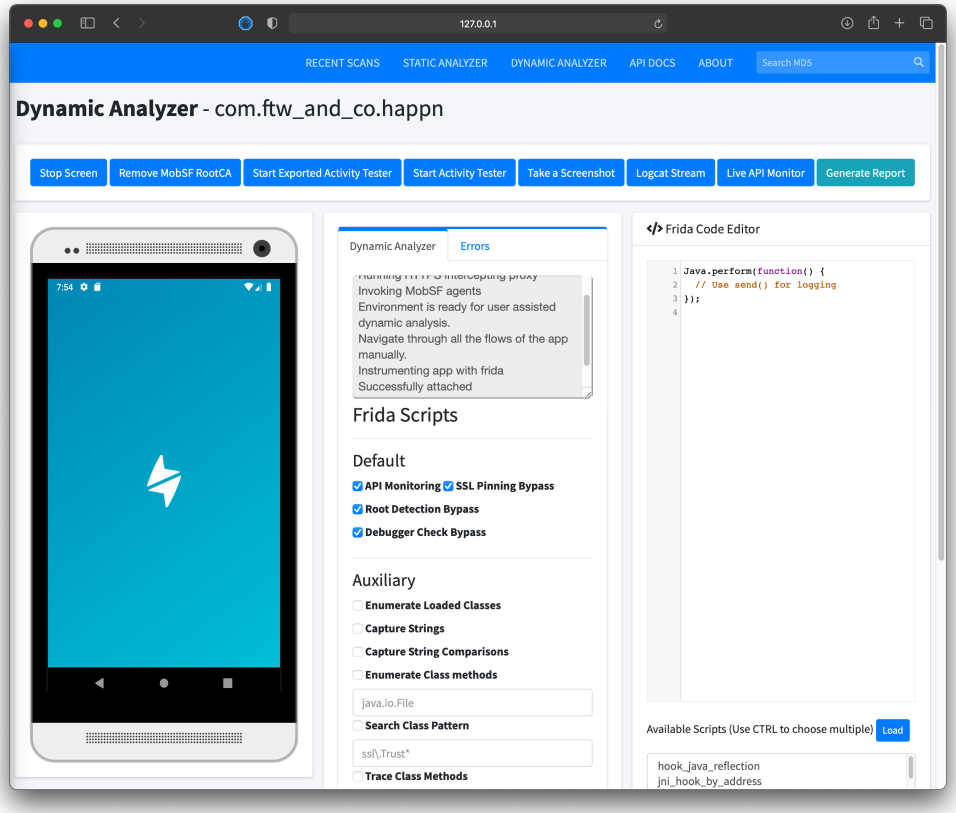


Figure 2.20: Hosted website running MobSF dynamic analyser

After having used the Happn app for some time, MobSF generates a report for all its findings. Looking through this report, the entire left column of the website is indexed in its various findings. Findings as different server locations as seen in figure 2.21, all recorded endpoints as in figure 2.22 or various logged emails as in figure 2.23.

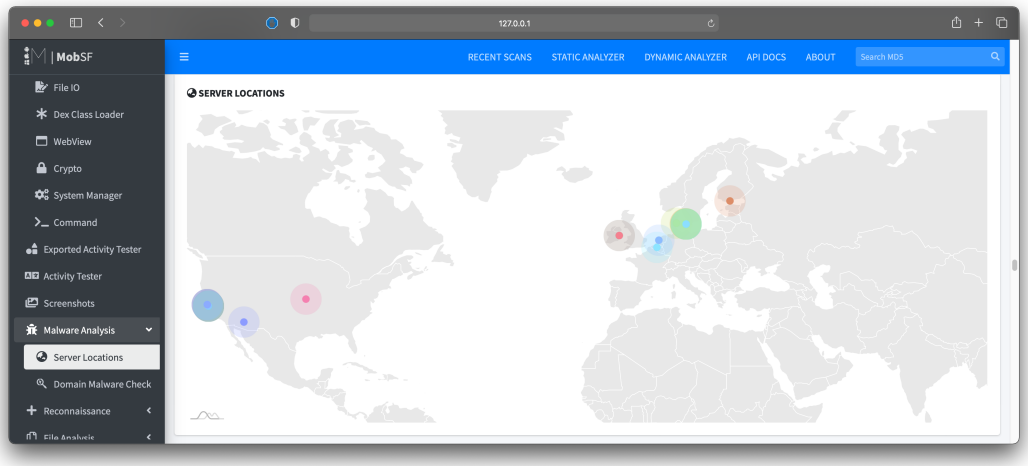


Figure 2.21: Generated MobSF report showing server locations recorded

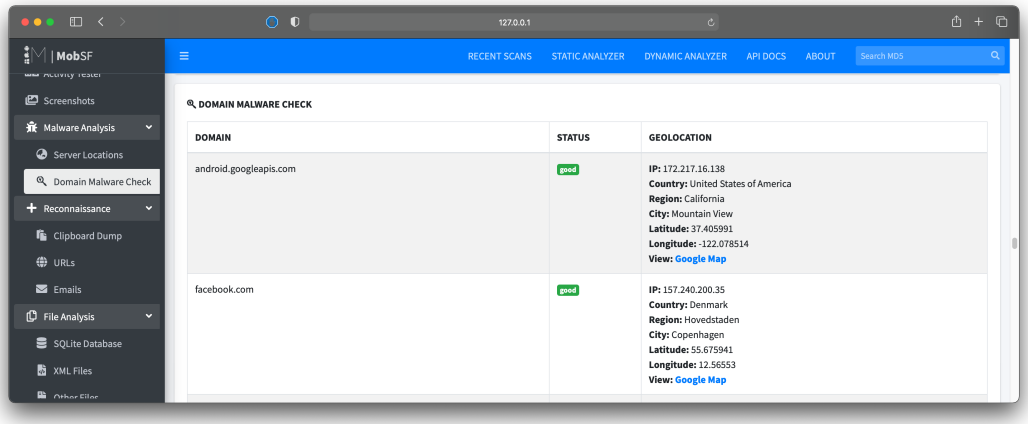


Figure 2.22: MobSF report showing all the domains and their details recorded in the analysis

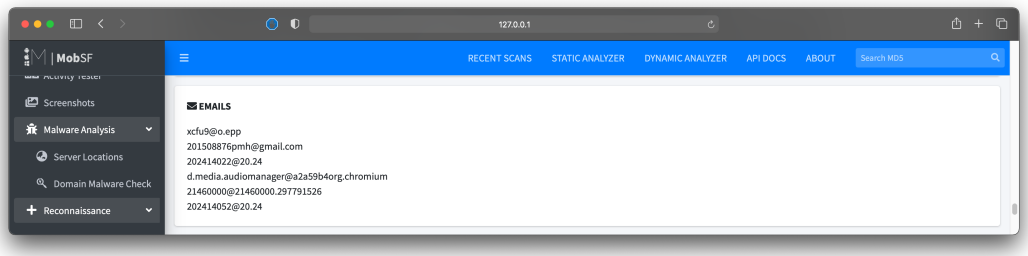


Figure 2.23: Emails logged from the dynamic analysis

For both static and dynamic analysis, one isn't better than the other, they both have their strengths and should be used in extension of each other, for a more thorough code analysis.

3 | Network Security

3.1 Hostnames and TLS configuration

An essential part of any app, is its endpoints. Most if not all apps now a days, have some form of API where data is exchanged from and to. These endpoints, which often have attached domain names, should have some form of TLS configuration as sending un-encrypted data opens up for attacks, but also reduces the likeliness of users using the app. Even though the secure socket layer protocol, is a standard for encrypting network communication, there is sadly surprisingly little attention paid to how SSL is configured, given its widespread usage[14].
A website by the name Qualys SSL Labs[13] offers a collection of documentations, tools and thoughts related to SSL. The tool enables quick but thorough analysis of the configuration of any SSL web server public to the internet. For each endpoint behind the domain name, Qualys SSL Labs tool will give a numerical score between $0 - 100$, where the different values will be given a grade, see figure 3.1.

Numerical Score	Grade
score >= 80	A
score >= 65	B
score >= 50	C
score >= 35	D
score >= 20	E
score < 20	F

Figure 3.1: Letter grade translation, from Qualys SSL Labs[14]

From the dynamic analysis we conducted in section 2.3, we managed to sniff a total of 30+ domain names. Opening up a program called Charles Proxy[21], a HTTP proxy which allows us to route traffic, we can see the domain names `api.happn.fr` and `notify.bugsnag.com` show activity, *see figure 3.2*. From this we take an educated guess and argue that these endpoints are of interest and want to analyse.

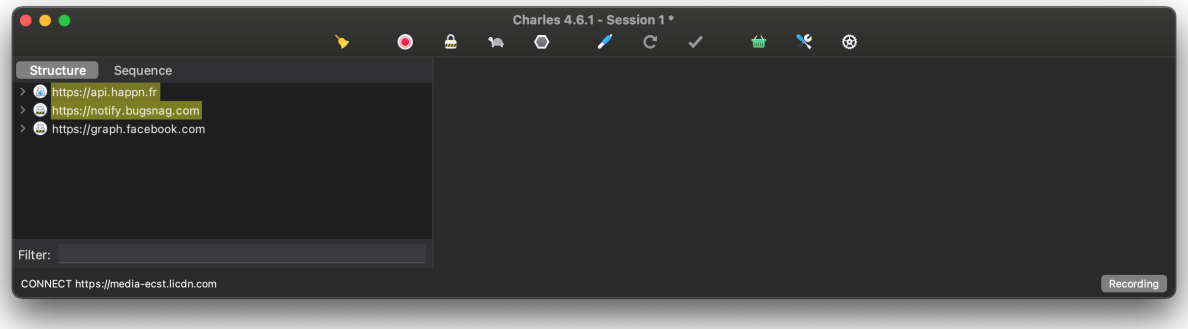


Figure 3.2: Ensuring the correct endpoint by sniffing through CharlesProxy

Trying our chosen endpoints in the Qualys SSL Labs tool, we get 10 visited IP's for `api.happn.fr` and 10 visited IP's for `notify.bugsnag.com`. Both are given a total rating of B and can be seen in figures 3.3 and 3.4.

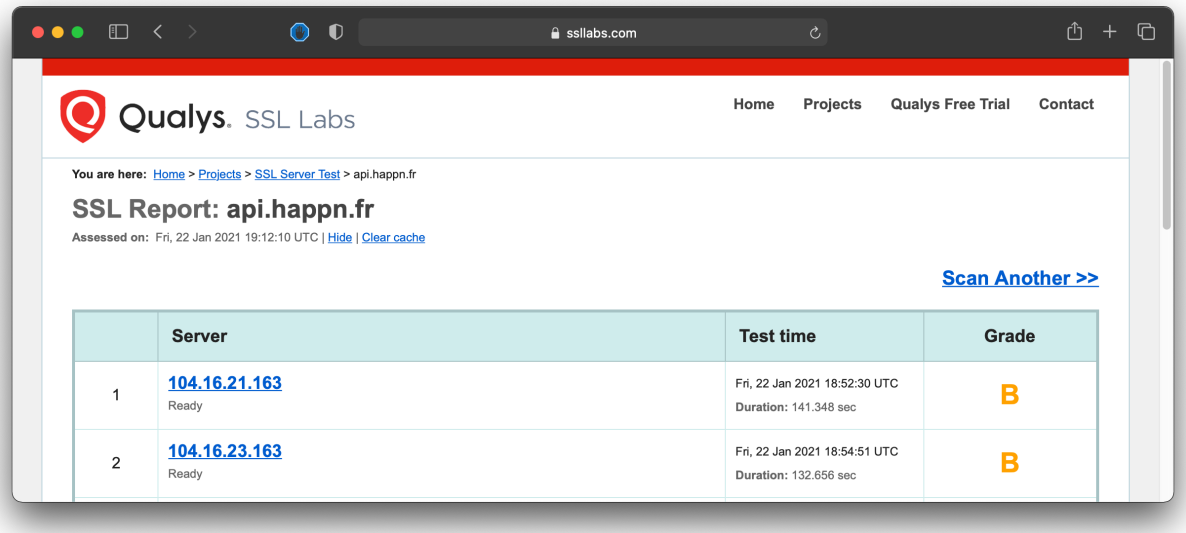


Figure 3.3: SSL Report from Happn endpoint: `api.happn.fr`

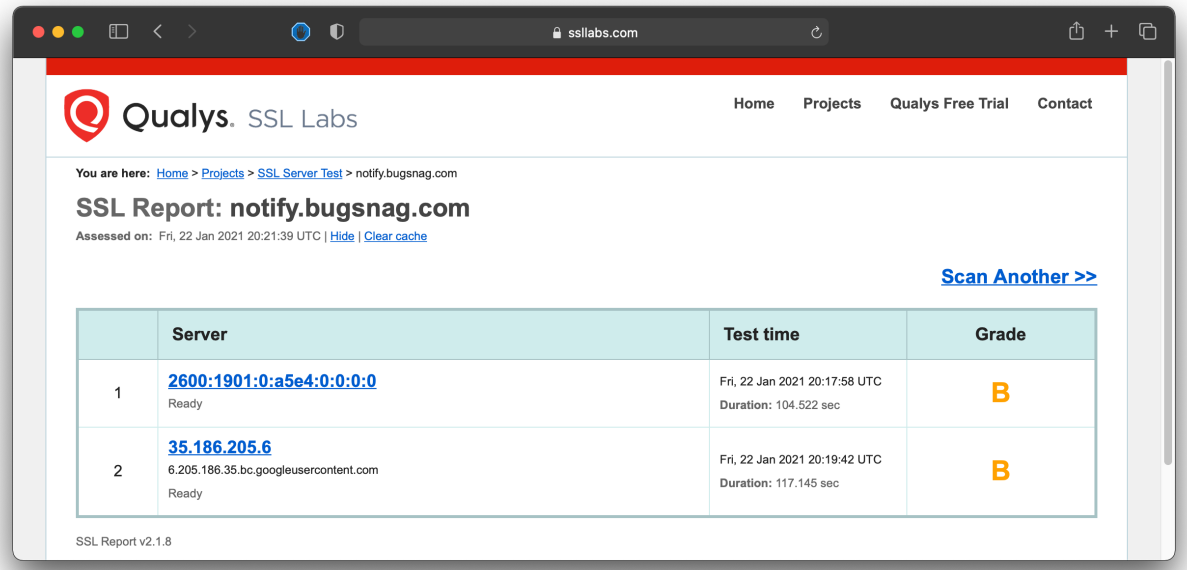


Figure 3.4: SSL report from Happn endpoint: `notify.bugsnag.com`

So what does the given rating B mean? And is it sufficient? Qualys SSL Labs argue, that the ratings are based upon a multitude of steps, as the degree on protocol, the server certificate's validity, key exchange, cipher support and more. Different web servers have different needs, and it is therefor impossible for the tool to choose any one configuration and say that it works for everyone. The Qualys SSL Labs tool, simply gives general guidance, configuration advice and tells us what never to do.
Having said that, having a good/decent SSL rating is not enough. The act of applying a TLS configuration is not an insurance for good client server communication, as TLS only deals with one aspect of security, namely the security of channel communication between the app and its users.

3.2 Packet sniffing

To fully get a grasp on how well Happn handles their data, we need to get our hands dirty and attempt to analyse the network by attacking it. We will be attempting to sniff packets, by firstly creating a spoofing attack, where we pretend to be a router by sending falsified ARP (Address Resolution Protocol) messages. We will then be trying to intercept traffic by a MITM (Man-in-the-middle) attack both with and without certificate pinning.

3.2.1 ARP Spoofing

For every given device on a given network, the `arp -a` command will display every static ARP entry. The ARP entries shows the various device IP addresses and their mac address. The idea behind an ARP spoof attack, is that if we can fool the victim device, which initially wants to send its data to the router, it can be fooled into sending its data to the malicious device.
The way this works is through the ARP (Address Resolution Protol) tables. When the client wishes to send some data, the traffic is routed to the router, which has its mac address looked up via the ARP tables. Looking at figure 3.5 we can see the router and the client device on the same network before the ARP spoofing attack. The two are distinguishable as they have different mac addresses.

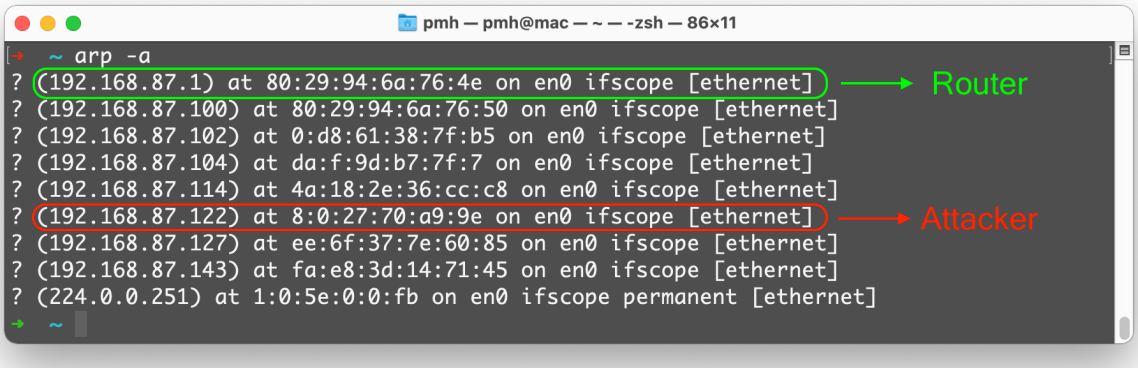


Figure 3.5: Picture of `arp -a` command from device. The red circle showing the malicious device IP and mac address

Having created a virtual machine with Kali Linux[16], we use the sniffing and spoofing tool Ettercap[9] as this supports ARP spoofing.
Looking at figure 3.6, we see the Ettercap tool having started its ARP spoof on two targets; target 1 (our router) with IP `192.168.86.1` and target 2 (our victim) with IP `192.168.87.104`. The idea is now, that we want to assign the attacker mac address to the router mac address and likewise, assign the mac address of the victim to be that of the attackers. From this, our client (victim) will open its ARP table, look for the mac address for the the router (now the attacker), send its data and we sniff it. Likewise from the routers point of view, when sending data back, it checks its ARP table, finds the mac address for the client (victim), which now is our attacker mac address, and sends it our way.

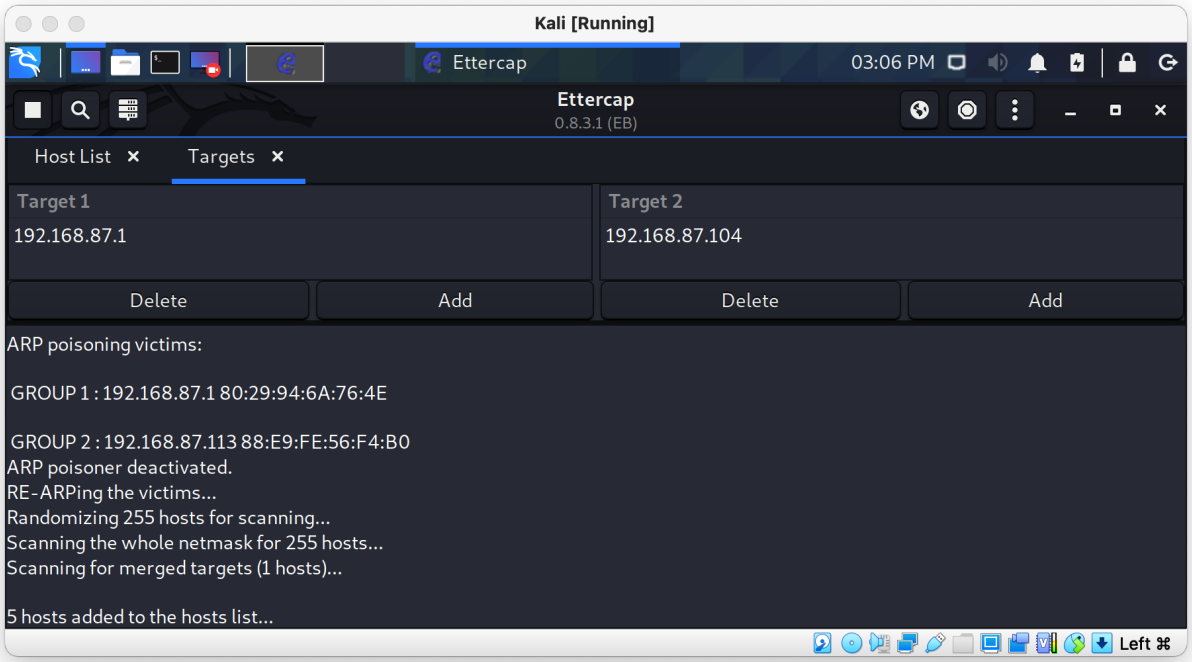


Figure 3.6: Kali Linux running Ettercap for ARP spoofing

Looking at figure 3.7, we can see how the mac address for the router has changed from our victims point of view. The mac address is now the same as the attackers mac address, meaning the traffic will be routed to the attacker. If we had the possibility to SSH into the router and call the `arp -a` command, we would see a similar result.

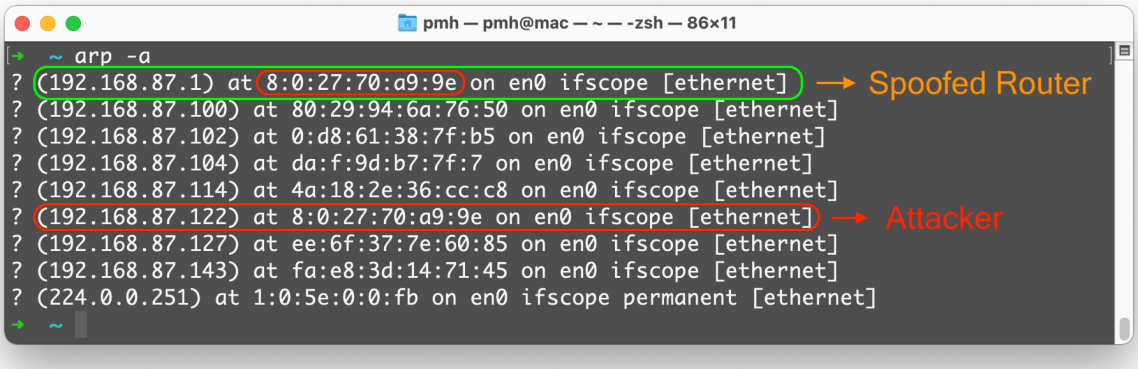


Figure 3.7: Picture of `arp -a` command from victim, showing the now spoofed router mac address

Using the website `site24x7.com`, we can find the IP addresses for one of the Happn endpoints. Looking at figure 3.8, we see a total of four IP address for the domain name. The reason for the four IP addresses could be because its behind a load balancer. The reason for all this, is WireShark needing the exact IP addresses for us to sniff the traffic.

S. No.	Domain Name	IP Address
1	api.happn.fr	pi.happn.fr./104.16.24.163
2	api.happn.fr	pi.happn.fr./104.16.21.163
3	api.happn.fr	pi.happn.fr./104.16.22.163
4	api.happn.fr	pi.happn.fr./104.16.25.163
5	api.happn.fr	pi.happn.fr./104.16.23.163

Figure 3.8: Finding IP addresses for Happn endpoint, picture from `site24x7.com`[5]

Having a look at Wireshark in figure 3.9, we see the long filter which essentially is the IP address for the victim device and all the IP addresses for the Happn domain names `api.happn.fr`. Looking at the highlighted packet No. 530, we see that the traffic is encrypted. Proving that the TLS configuration of Happn for ARP spoofing without a malicious root certificate installed, is good enough to hide packet contents.

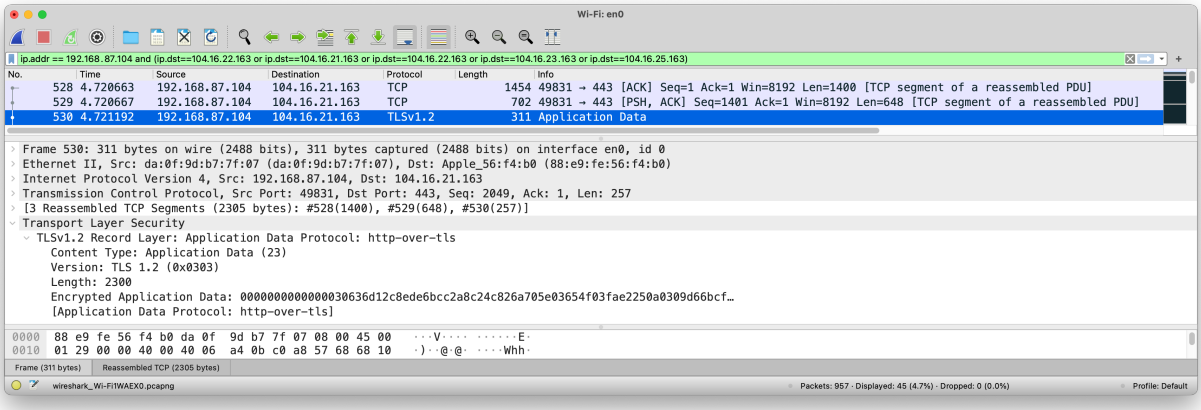


Figure 3.9: WireShark sniffing ARP spoofed packets

3.2.2 MITM Proxy Attack

The final attack we will try on our Happn app is a similar man-in-the-middle-attack as ARP spoofing, but with a program called MITM Proxy[17], enabling us to install a malicious root certificate. The idea is, that if we can install our own malicious certificate, we can decrypt encrypted traffic on the fly, enabling us to see Happn network packets.

As the name MITM Proxy suggests, it works by proxying traffic from the client to the attacker device. As a first, we must setup a manual proxy for our mobile client. We will be choosing Android Studio as our emulator, as the program offers the desired functionality. Looking at figure 3.10, we see we've set the host name to match that of the IP address for our Kali Linux machine hosting MITM Proxy. The port is given as 8080, as this is where MITM Proxy is listing.

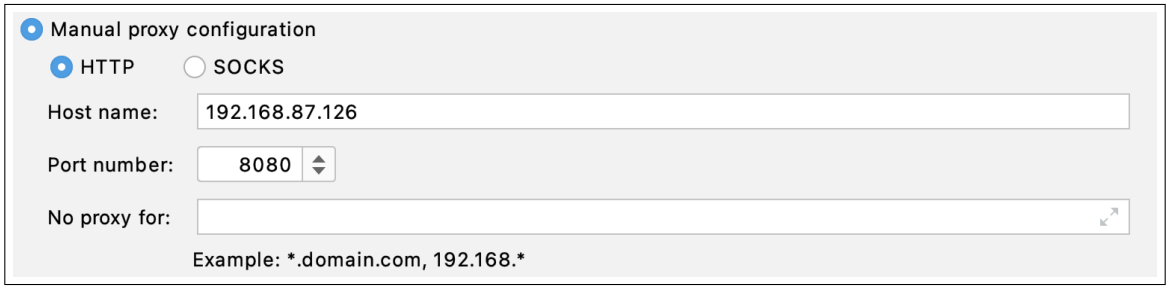


Figure 3.10: Android Studio emulator HTTP proxy routing traffic to MITM Proxy

Looking at figure 3.11, we see our Kali Linux virtual machine hosting our MITM attack. It offers a webpage to view the captured packets at `http://127.0.0.1:8081/` and showing the proxy server listening at `http://*:8080`.

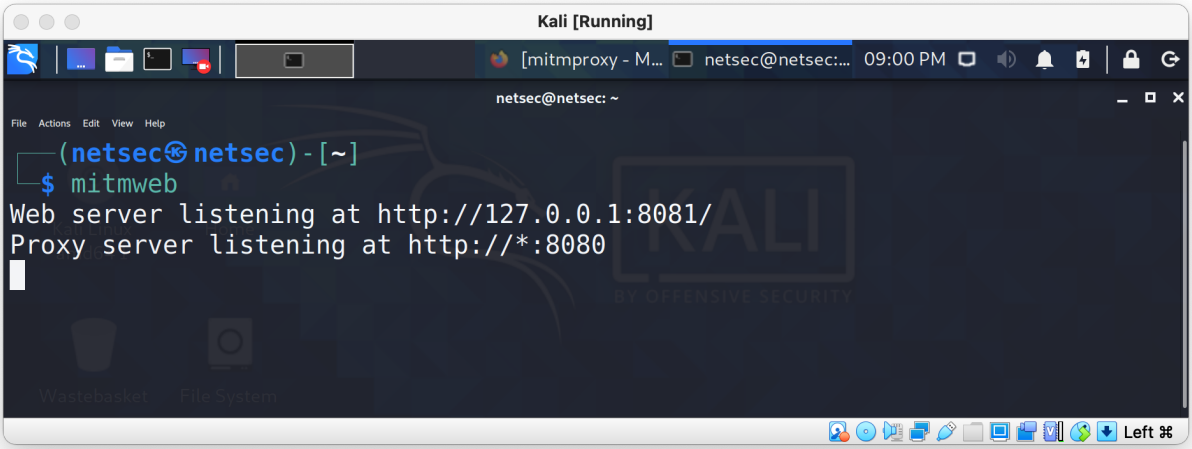


Figure 3.11: Kali Linux hosting MITM Proxy

By far the easiest way to install our malicious mitmproxy certificate, is to visit the domain `mitm.it` as seen in figure 3.12. Here we simply choose our operating system and download and install the certificate.

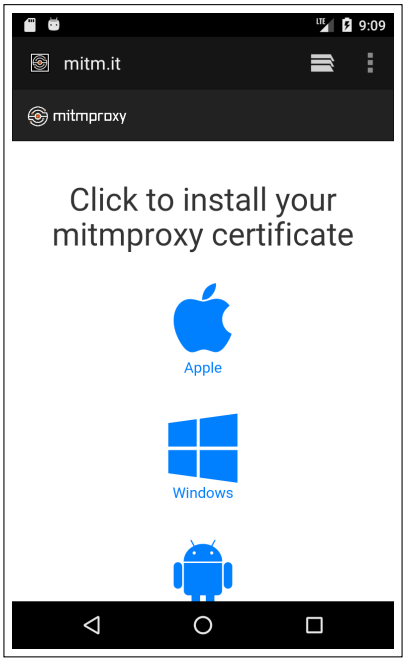


Figure 3.12: Visiting `mitm.it` domain to install mitmproxy certificates

After having set up the proxy and installed the mitmproxy certificate, we see in figure 3.13 the overwhelming amount of decrypted sniffed packets. The highlighted sniffed Firebase packet, is being inspected where an auth-token can be seen in clear text.

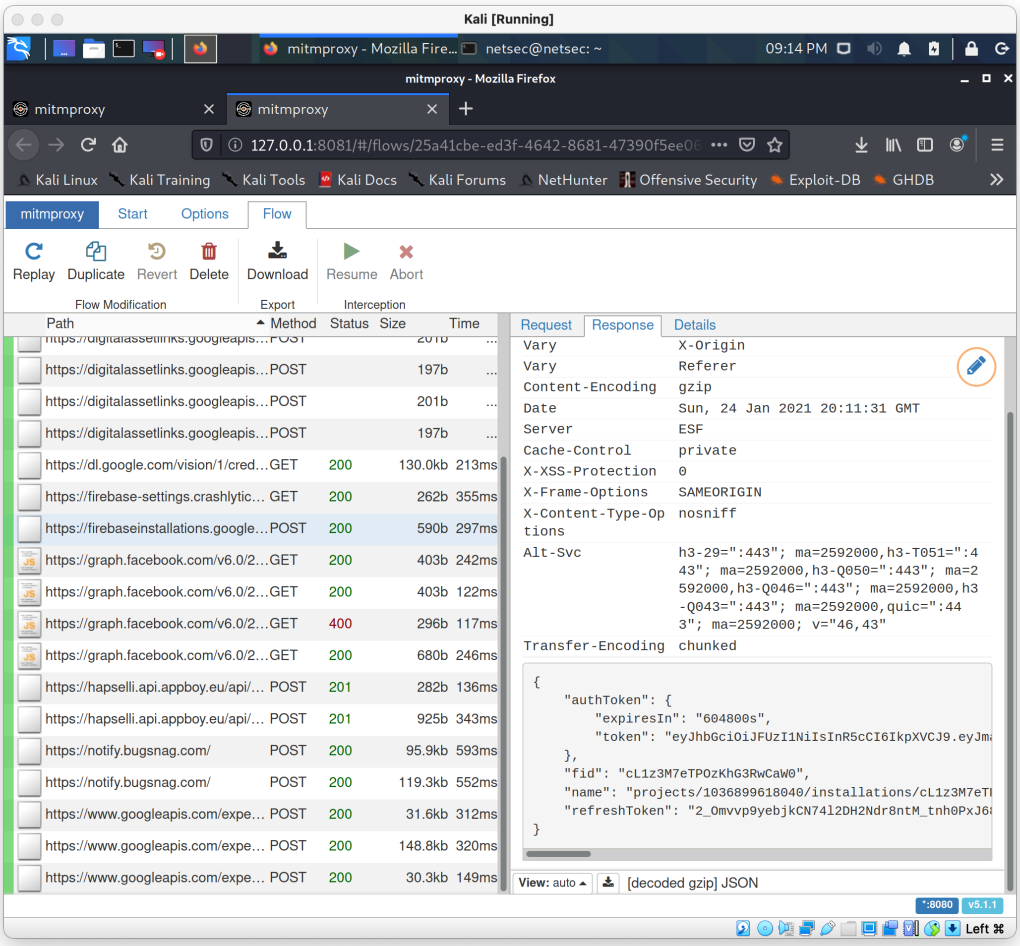


Figure 3.13: Initial readings of opening our Happn app

Going through the process of creating an account, we are asked to write our telephone number. The phone number is typed in, as seen in figure 3.14, were we in figure 3.15 can see the packet being sniffed and shown in clear text.

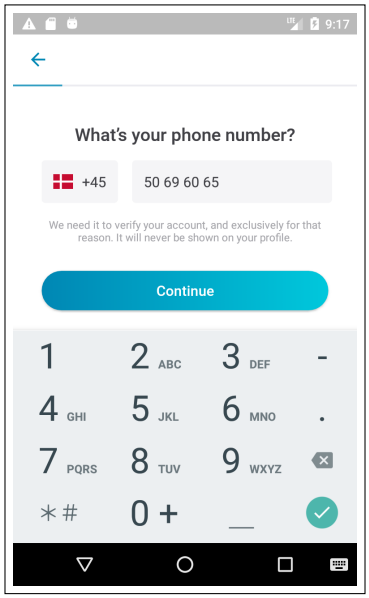


Figure 3.14: Typing in a random phone number in Happn

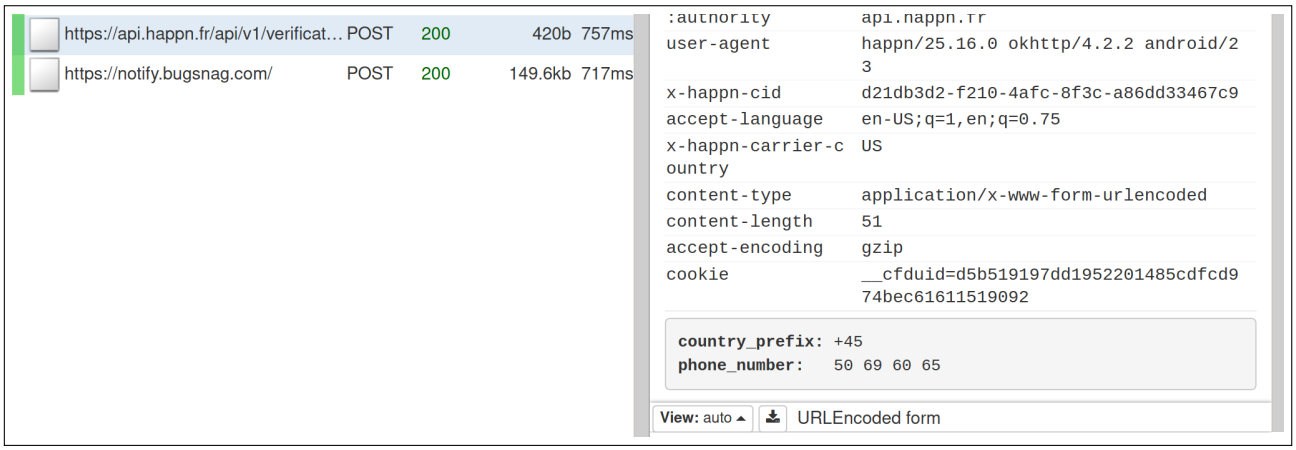


Figure 3.15: MITM Proxy sniffing the packet of our previously typed phone number

Looking at the bigger picture, both ARP spoofing and redirecting traffic to be sniffed via a proxy, is a man-in-the-middle attack. For the ARP spoof attack, we weren't required to gain access to the victims device and force him to change any settings. Whereas the mitmproxy tool required the user to manually change his/her device, to route traffic to the attacker.
For any traffic to be decrypted for the Happn app, we needed to have a malicious CA certificate installed on the target device. Some applications employ something called certificate pinning to prevent a man-in-the-middle attack. This would result in mitmproxy's certificate not being accepted by applications without modifying source code[18].
Furthermore, as of writing this report, android versions 7.0 and up wont accept custom installed certificates to route traffic from apps[1]. Marshmallow (Android version 6.0) was the version we used.

4 | Authentication

4.1 Importance of authentication

The act of authenticating is the the process of verifying the identity of a person or device, more specifically, enabling the access to a resource. As more and more interactions happen with sensitive user data, the need for good authentication is a must.
The act of authenticating is often talked about as being three factors[4]:

1. Something you **know**. Ex: password, passphrase (knowledge)
2. Something you **have**. Ex: cryptographic key/token, smart card (ownership)
3. Something you **are/do**. Ex: fingerprint, voice, iris, behaviour, trait(biometrics)

As for authentication for the Happn app, one could argue that it isn't of utmost importance, as it is a dating app. Would an attacker manage to get access to a persons Happn account, the damage would be limited to change of preferences and at worst account deletion.

4.2 Authentication mechanism in place

4.2.1 User oriented

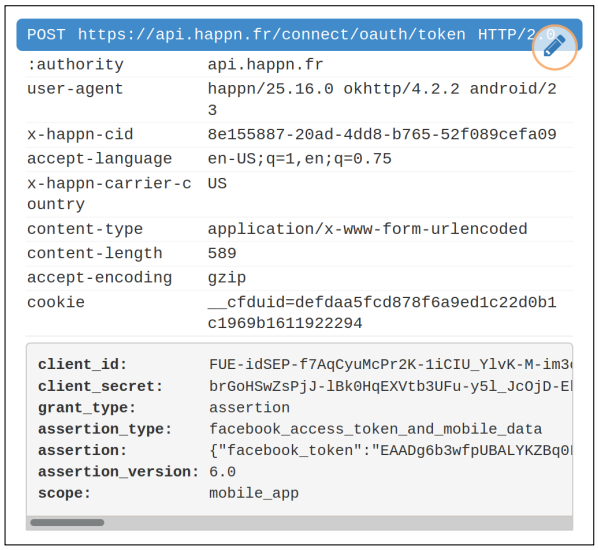
Opening the Happn app, we are offered three ways to authenticate ourselves with; login with Google, login with Facebook or create a new account with Firebase. For all of these, we end up authenticating with both the *(knowledge)* authentication factor and the *(ownership)* authentication factor. For all of the logins, we are typing ind a password of choosing and then receiving an OAuth token for us to validate future API requests. Without one or the other, the app will not work.
However, when the first login has been made, the application will no longer require the knowledge aspect of authentication, requiring no password for future use. Would this be an application as ex. MobilePay, it would be deemed as a bad security practise.

4.2.2 Device oriented

The third authenticating factor is the use to prove we are or can do. Proving by fingerprints, voice, iris, behaviour or traits (biometrics), we are to prove who we say we are. Some applications offer this third act of authentication as ex. NemID, where one can choose to authenticate themselves by fingerprint. However many applications such as Happn choose not to implement biometrics as most devices offer this themselves.
A lot of Android and Apple phones offer this as a standard feature, where both Touch ID and Face ID has become the norm. Apple writes, that for their touch ID, no images of the actual fingerprint is stored, but instead relies on mathematical representations. It wouldn't be possible for someone to reverse engineer a persons actual fingerprint image from the stored data[2].

4.3 Authentication in Happn

Having sniffed two OAuth tokens, one from the Facebook login and the other from Firebase, as seen in figure 4.1, we have in theory the authorization we need to make various http operations.
The idea for testing out the authentication behind the received OAuth tokens, is to try the limits for the given permissions. Each token should have bounded permissions, fitting for normal users. Requests for deleting other users should be strictly intolerable, as this would have massive consequences for the database of the application.



(a) OAuth token 2.0, sniffed from facebook

(b) OAuth token 2.0, sniffed from Firebase

Figure 4.1: Sniffing contents of OAuth tokens

Trying this ended up not succeeding, due to problems with the Postman program or a deeper rooted cause. The idea was, that if our mitmproxy was able to successfully make a `GET` request to the Happn API with the OAuth token, so should we.
However the concept still stands. The `GET` request shown in figure 4.2, shows the long filtered endpoint for retrieving potential matches in our specified range. One of the first operations we would try was altering the parameters and to try to retrieve users from a greater distance.

Happn.GET.Request

```
1 https://api.happn.fr/api/users/e6ce381f-15cb-4a52-99a1-72f82f87047b/conversations?view_id=pending&offset=0&limit=16
2 &with_count=true&fields=id%2Cparticipants.fields%28user.fields%28id%2Cfirst_name%2Cage%2Cis_moderator%2Ctype%
3 2Cmodification_date%2Cgender%2Cprofiles.mode%280%29.width%28112%29.height%28112%29.fields%28width%2Cheight%2Cmode%2
4 Curl%29%29%29%2Cis_read%2Ccreation_date%2Cis_disabled%2Cmodification_date%2Clast_message.fields%28creation_date%
5 2Cmessage%2Csender%29
```

Figure 4.2: Sniffed GET request for finding users

Figure 4.3, show the json object we get from our `GET` request. From this we get the users supposed unique table ID, personal information and 4 pictures. The next limit to try, would be to request a deletion of the user we just found. The theory would be, that if were able to delete a specific user, what would stop us from deleting the entire database?

Happn.GET.Response.json


```
{
   "id":"18000605602_e6ce381f-15cb-4a52-99a1-72f82f87047b_18000605602",
   "user":{
      "age":28,
      "first_name":"Mikkel",
      "gender":"male",
      "id":"18000605602",
      "is_moderator":false,
      "modification_date":"2021-01-27T10:44:14+00:00",
      "nb_photos":4,
      "profiles":[
         {
            "height":160,
            "id":"6445dbd0-266e-11eb-9e90-2d5aa6a19fd4",
            "mode":0,
            "url":"https://images.happn.fr/resizing/1562/644d0-266e-11eb-9e90-2d5aa19fd4.jpg?width=160&height=160&mode=0",
            "width":160
         },
         {
            "height":160,
            "id":"368d5b80-ee1f-11e7-9dbc-4d1402f1c7cf",
            "mode":0,
            "url":"https://images.happn.fr/resizing/1502/640-16.1_388-eef-117-9dbc-4d2c7cf.jpg?width=160&height=160&mode=0",
            "width":160
         }
      ],
      "role":"CLIENT",
      "type":"client"
   }
}
```


Figure 4.3: GET response

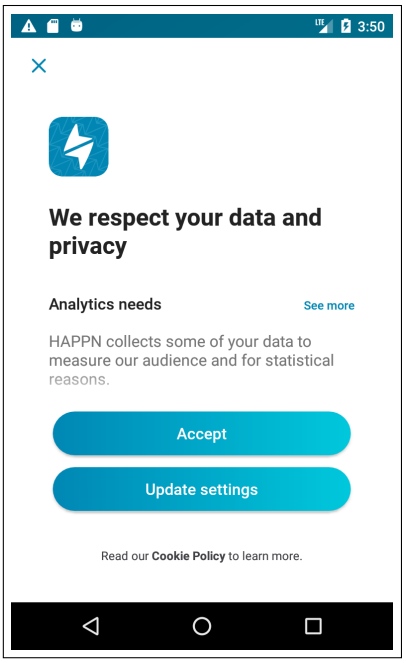
All in all for a dating application as Happn, where the primary goal is for people to match within chosen distances, the authentication methods are deemed acceptable. The application makes use of popular authentication methods such as tokens and encryption while also relying on biometrics from the device. The tokens we sniffed were only possible from the old 'Marshmellow' android 6.0 OS, so sniffing such traffic with newer phones would be more tricky. Furthermore, since we weren't able to try out our theory on testing the limits of our OAuth tokens, we could take an educated guess and argue that it probably wouldn't be the case that deleting other users was made possible.

5 | Privacy

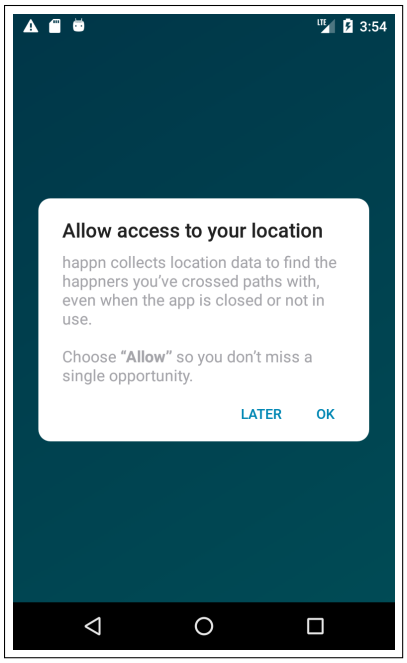
5.1 Characterization of sensitive collected data

Launching the Happn app, we are prompted with the screens shown on figure 5.1. Reading the terms of services we are explained that Happn collects and uses our data at registration and when the app is in use, in order to:

- Provide us with their services and assistance
- Provide us with promotional content, notably ads for products and services from Happn and/or its partners
- To use for optimization of how their application is used, improve it and ensure security



(a) Accepting Terms and services



(b) Asking permission for collection of location data

Figure 5.1: Privacy windows needing acceptance when opening Happn for the first time

Trying to characterise what sensitive data, the Happn app is actually asking for, we take a look into the source code and the various packets sent. Looking at the `AndroidManifest.xml` found from our decompilation, we see the entire overview for every permission we end up granting. Android apps must request permission to access sensitive user data (such as contacts and SMS), where each permission is identified by a unique label in the manifest[6].
Looking at the manifest in figure 5.2, we see the entire permission list. For some of these we have the possibility of deactivating, however many of them are a baseline requirement for Happn to function properly.

permissions.xml

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" android:maxSdkVersion="22"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.BLUETOOTH" android:required="false"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" android:required="false"/>
<uses-permission android:name="com.ftw_and_co.happn.permission.C2D_MESSAGE"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
<uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="com.android.vending.BILLING"/>
```

Figure 5.2: Happn permission list found from manifest

5.2 Third party integration

The use of cloud computing for reducing cost of computation and storage, has become mainstream for a large variety of apps. The use of third-party applications can be a valuable asset to the development process and the app as a whole...but at what cost?
The use of cloud computing requires delegating computation to non-trusted parties, which is possibly under the influence of intrusive governments or different legal jurisdictions. Even if the development team decides that the third party can be trusted, there will always be a permanent risk of operational failure and/or data leakage[3].

Reading through Happn's terms of services, we are explained what third-party services are in use and for what purposes. Table 5.1 shows this list, where category one is labeled as being "additional data", label two being for collection of personal data to measure audience and for statistical reasons, and lastly all the personalised ads showing according to age range, gender and country or city.

Categories:	Addition data:	Statistical:	Personalised ads:
Third-party services:	Adjust	Happsight	Google ad Manager
	Facebook	Bugsnag	
	Google Firebase	AppCenter	
		Facebook login	
		Zendesk	
		Braze	
		Google Firebase	
		Spotify	
		Sign in with Apple	
		Sign in with Google	
		Agora	

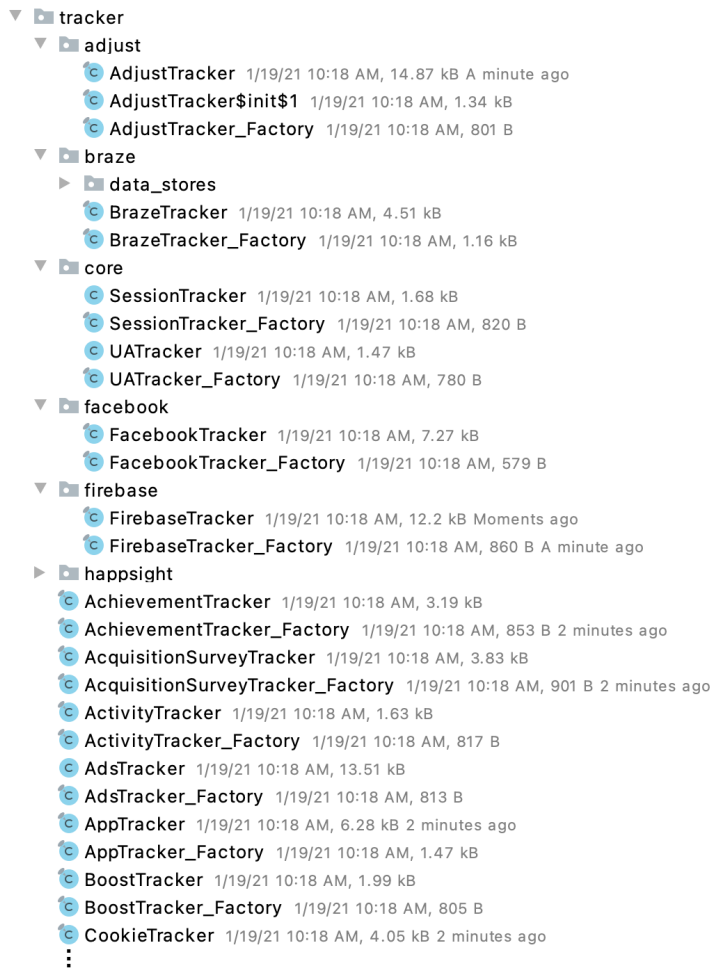
Table 5.1

The above table is derived from the Happn documentation found from reading the terms and services agreement.

5.3 Embedded trackers

Looking through the decompiled source code, we find a dedicated tracker folder with various nested trackers. Figure 5.3 shows a total of 6 folders with trackers for various aspects of the application.

Figure 5.3: Some of the many tracker classes



Looking back at the terms of service agreement, where the use of third party services where listed, we can here see in more detail, some of their uses.

As an example, reading about the Facebook tracker `com.facebook.appevents.AppEventsLogger`[7], the tracker doesn't send the sensitive user data immediately, but logs it and flushes it out to the Facebook server in a number of situations. Either when 15 seconds has passed, when the logs exceed 100 events, when the app has gone to the background and is brought back to the foreground.

Trying to analyse the traffic of some of the packages was attempted, but proved to be unsuccessful with mitmproxy. As most, if not all, of the trackers are from third party libraries, this could be caused by certificate pinning.

As a side note, looking at figure 5.4, we can see some of the trackers being injected into existence at login.

```
@Inject
public LoginTracker(@NotNull HappsightTracker happsightTracker, @NotNull UATracker uATracker,
                    @NotNull AdjustTracker adjustTracker2, @NotNull FirebaseTracker firebaseTracker2) {
    Intrinsics.checkNotNullParameter(happsightTracker, str: "happsight");
    Intrinsics.checkNotNullParameter(uATracker, str: "UATracker");
    Intrinsics.checkNotNullParameter(adjustTracker2, str: "adjustTracker");
    Intrinsics.checkNotNullParameter(firebaseTracker2, str: "firebaseTracker");
    this.happsight = happsightTracker;
    this.UATracker = uATracker;
    this.adjustTracker = adjustTracker2;
    this.firebaseTracker = firebaseTracker2;
}
```

Figure 5.4: Trackers being injected at login

Having as many different trackers as the Happn app has, where most if not all, are from third party sources, could be argued to be a security liability. Being dependent on that many third-party trackers and betting on none of them, being privacy-invasive and simultaneously following good security practises is arguably a stretch.

6 | Conclusion

All in all the mobile dating app Happn, which we've analysed through various aspects of security, has proven not to be flawless. We've seen how, from static and dynamic analysis, the Google Maps API key being non restricted and readily available for anyone to use.
The TLS configuration, which was determined to be of grade B, didn't have certificate pinning, resulting in potentially exposing sensitive user data.

The authentication used was deemed appropriate for the application context. Only requiring login at first use, but using popular and well documented third-party services.
Lastly we looked at privacy, where we saw the various integrations with trackers. Happn used trackers not only for their core functionality, but for personalised ads and statistical information. Looking back at the our threat model, it is clear that of the four assessed attack surfaces, the third-party services is one of the major ones. The Happn app builds upon the use of third-party tracking services, which all face the risk of operational failure and/or data leakage.

Bibliography

Websites

[1] android-developers. *Changes to Trusted Certificate Authorities in Android Nougat*. URL: https://android-developers.googleblog.com/2016/07/changes-to-trusted-certificate.html. (accessed: 24.01.2021).

[2] Apple. *Secure Enclave*. URL: https://support.apple.com/en-us/HT204587. (accessed: 29.01.2021).

[3] Diego F. Aranha. *PRIVACY TECHNOLOGIES*. URL: https://blackboard.au.dk/bbcswebdav/pid-2916214-dt-content-rid-10115167_1/courses/BB-Cou-UUVA-91723/NETSEC__Privacy_Technologies%281%29.pdf. (accessed: 28.01.2021).

[4] Diego F. Aranha. *Week 7: Access Control and Authentication, USER AUTHENTICATION*. URL: https://blackboard.au.dk/webapps/blackboard/content/listContent.jsp?course_id=_136793_1&content_id=_2797602_1&mode=reset. (accessed: 26.01.2021).

[5] Zoho Corporation. *Find IP Address*. URL: https://www.site24x7.com/find-ip-address-of-web-site.html. (accessed: 24.01.2021).

[6] Android developers. *App Manifest Overview*. URL: https://developer.android.com/guide/topics/manifest/manifest-intro#perms. (accessed: 28.01.2021).

[7] Facebook for developers. *com.facebook.appevents.AppEventsLogger*. URL: https://developers.facebook.com/docs/reference/androidsdk/current/facebook/com/facebook/appevents/appeventslogger.html/. (accessed: 29.01.2021).

[8] Bhavya Doshi. *Understanding the Android Compilation Process*. URL: http://www.theappguruz.com/blog/android-compilation-process. (accessed: 19.01.2021).

[9] Ettercap. *Ettercap Project*. URL: https://www.ettercap-project.org/. (accessed: 24.01.2021).

[10] Google. *SimpleMap*. URL: https://developers.google.com/maps/documentation/javascript/examples/map-simple. (accessed: 29.01.2021).

[11] Happn. *Happn-about*. URL: https://www.happn.com/en/about/. (accessed: 18.01.2021).

[12] javadecompilers. *Decompile Apk and Dex Android files to Java*. URL: http://www.javadecompilers.com/apk. (accessed: 18.01.2021).

[13] Qualys SSL Labs. *Qualys SSL Labs*. URL: https://www.ssllabs.com/index.html. (accessed: 22.01.2021).

[14] Qualys SSL Labs. *SSL Server Rating Guide*. URL: https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide. (accessed: 22.01.2021).

[15] likedin. *qark*. URL: https://github.com/linkedin/qark. (accessed: 19.01.2021).

[16] Kali Linux. *Kali Linux | Penetration Testing and ethical Hacking*. URL: https://www.kali.org/. (accessed: 24.01.2021).

[17] mitmproxy. *mitmproxy Project*. URL: https://mitmproxy.org/. (accessed: 24.01.2021).

[18] mitmproxy. *The mitmproxy certificate authority*. URL: https://docs.mitmproxy.org/stable/concepts-certificates/. (accessed: 24.01.2021).

[19] MobSF. *Mobile Security Framework (MobSF)*. URL: https://github.com/MobSF/Mobile-Security-Framework-MobSF. (accessed: 20.01.2021).

[20] pxb1988. *dex2jar-git*. URL: https://github.com/pxb1988/dex2jar. (accessed: 18.01.2021).

[21] XK72. *Charles, Web debugging proxy application*. URL: https://www.charlesproxy.com/. (accessed: 20.01.2021).