Network Security Project

Henrik Buhl
Department of Computer Engineering
Aarhus University
Aarhus, Denmark
201905590@post.au.dk

1. INTRODUCTION

Mobile apps are becoming more and more advanced as time goes on. Today you can do almost anything from your mobile device, you can play games, check your bank balance, and you can also order food and other stuff online. This all shows that we are becoming a more and more digitalized society wherein all the transactions you do online ends up having to go through compact applications. Therefore it is of the utmost importance to ensure these applications are capable of securing your data such that a malicious adversary cannot use them for nefarious purposes. The mobile application which I am analyzing in this report is "Foodora" [1] which originally was called "Hungry" but ever since October 1st 2021 then it has been taken over by a Swedish firm called "Delivery Hero" [2]. The app has been made by the team Hungry Group [1] and can be downloaded for both Android and Apple smartphones. The android app has approximately 100 thousand downloads[1] and with two other versions of the app for Norway and Sweden then there is a total of approximately 2.1 million downloads. Interestingly enough then there is a mix between the ratings from the different review sites having a score of 4.4 stars in the app store[3], 2.7 stars on google play store[4] and 1.3 stars on trust pilot[5]. I am led to believe that a lot of the high reviews from the app store and google play store are coming from years ago when the app was called Hungry and that these reviews have just been "carried over" to the "new" app. This becomes more evident once you start to look at newer reviews on trust pilot for Hungry[6] where it can be seen that in the last years time then there have been 350 reviews with 295 of them giving 1 star. Thus it is clear that the customers are dissatisfied with the newer changes which have been happening ever since Delivery Hero took over. The app handles a lot of data such as the users location, phone number, contact details and more. Due to the app undergoing "changes" and a lot of sensitive data being handled by it then it is the perfect candidate for a security analysis. I will therefore detail the apps security properties by analysing its software security, network security, authentication mechanisms, and how good it is at not intruding on the users privacy.

1.1. Security Properties

1.1.1. Attack surface: The attack surface includes the app as well as the communication with the server's API over the internet. It also includes the cloud servers keeping track of the users orders, information and the delivery drivers location.

1.1.2. Threat model: The threat model of a fast food app is small because there isn't a lot of important data which needs to be handled. The most likely to attack this app is someone who is a hacktivist since the most important information which is possible to get is detailed to the users such as their location and transactions. Thus I am assuming the adversary's capabilities are rather limited to minor vulnerabilities which don't need much ground work to be exploited.

1.1.3. Security policies: The app should ensure the following security policies from the information assurance(IA) framework[7]:

Availability: The app should be available throughout the day such that the owners of the app do not lose any possible profits. However the importance of the app being up all the time is not super high since it only is roughly 100 thousand people who use it and because people are most likely to eat during sunrise, lunchtime and in the afternoon thus past midnight is a bit too much to expect restaurants to be available.

Authenticity: The app should ensure that people are who they claim to be by having users confirm their identities before gaining access to important areas such as payment details. 2-factor authentication is expected and I will go into greater details in later sections.

Confidentiality: The app should assure users that their data does not wind up in the hands of nefarious people or organizations. Highly sensitive data such as credit details and user credentials should be encrypted such that third parties who are not authorized to view the data cannot access it.

Integrity: Data must not be altered or destroyed during transmission by any unauthorized individuals.

Non-repudiation: must be enforced such that a malicious user cannot make a request and claim it didn't occur. Example we do not want it to be possible to make the old pizza prank, where someone simply places an order for someone they dislike or doesn't exist such

that the food never gets paid for and the user isn't held accountable for their actions.

2. Software security

I have decompiled the APK using MOBSF and APK-tool with dex2jar and for both instances then it was possible for me to find a lot of obfuscated filenames by opening the decompiled code in Jd-GUI (see figure 1).

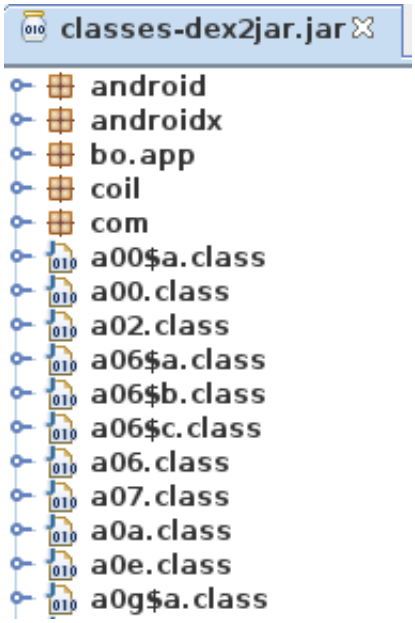

Fig. 1: Here it can clearly be seen that the app has been obfuscated by looking at how the classes are named.

I was also able to see that the contents of the files also was obfuscated as seen in listing 1.

```
import android.graphics.PointF;
import java.util.Collections;

public final class aio extends zb1<PointF,
    PointF> {
  public final PointF i = new PointF();

  public final PointF j = new PointF();

  public final zb1<Float, Float> k;

  public final zb1<Float, Float> l;

  public aue<Float> m;

  public aue<Float> n;

  public aio(yr9 paramyr91, yr9 paramyr92) {
    super(Collections.emptyList());
    this.k = paramyr91;
    this.l = paramyr92;
    j(this.d);
  }

  public final Object f() {
    return l(0.0F);
  }

  public final void j(float paramFloat) {
    this.k.j(paramFloat);
    this.l.j(paramFloat);
    this.i.set(((Float)this.k.f()).floatValue
    (), ((Float)this.l.f()).floatValue());
    for (byte b = 0; b < this.a.size(); b++)
      ((zb1.a)this.a.get(b)).a();
  }
```

Listing 1: This shows the file (aio.java) containing obfuscated code.

I could also find the following hard-coded secrets from the static-analysis which MOBSF did.

```
"google_app_id" : "1:463657480984:android:
    c932668f330bb736"
"google_api_key" : "AIzaSyC4Qis2u-
    TeEhzA_k9jz5dE21677arK83o"
"google_crash_reporting_api_key" : "
    AIzaSyC4Qis2u-TeEhzA_k9jz5dE21677arK83o"
```

Listing 2: Hardcoded secrets which MOBSF found.

I tried checking the google api key through a tool called gmapsapiscanner[8] for potential misuse and i found what is shown in figure 2.

Results	Cost Table/Reference to Exploit:
- Directions	$5 per 1000 requests
- Directions (Advanced)	$10 per 1000 requests
- Geocode	$5 per 1000 requests
- Find Place From Text	$17 per 1000 elements
- Autocomplete	$2.83 per 1000 requests
- Autocomplete Per Session	$17 per 1000 requests
- Place Details	$17 per 1000 requests
- Nearby Search-Places	$32 per 1000 requests
- Text Search-Places	$32 per 1000 requests
- Places Photo	$7 per 1000 requests

Fig. 2: This table shows how much the app-owners have to pay per 1000 requests of something. A malicious adversary could make bots do a ton of requests on Foodoras part and cost them a lot of money

I tried to see if I could find the keys manually and potentially more by looking at the file which MOBSF got the keys from, I came to the conclusion that

the file was called *R.java* and it was located in the directory *hungry/dk/android*. The file was however too big for me to load and therefore I could not search in it manually. I was however able to find a file called *fbc.java* and in it were some more potential hard coded secrets such as *google0authid*, *brazeapikey* and more.

After that then I searched the rest of the app for the keywords java.security and javax.crypto (because it would lead to any potential usage of a cryptographic algorithm) and noted some of the files down which contained something interesting, such as *l6v.java* which shows the generation of DES keys thus hinting at them using a vulnerable encryption scheme. I also found the file *ls.java* which shows *CBC/PKCS7Padding* which is vulnerable to Padding Oracle Attacks [9]. At last I found that the MOBSF reported there being a usage of some rather good security features implemented such as root detection and SSL pinning from a couple of files, here is an example of root detection, *dto.java* and SSL pinning *ca.java fzg.java*. In *ca.java* the trust manager is defined to use SSL and in *fzg.java* the trust manager is used for SSL certificate pinning to prevent MITM attacks.

From reading the MOBSF report it had noted that it would maybe be possible to inject SQL and HTML code into parts of the app however when I tried it then I could only find 1 place, the search bar for finding restaurants where it would be directly possible to test and I found that it wasn't possible because it all got caught and was stringified. I tried out a lot of injection attacks such as *¡b¿hello¡/b¿* and *SLEEP()* and I even tried to see if I could stop the stringification by trying to close the search with for example *¿¡/b¿* and more to see if it would work but it ended up not giving any fruitful results.

At last then I verified the certificate using jarsigner and got the output shown in figure 3.

3. NETWORK SECURITY

I have used Qualys SSL Labs to test some URLs which the app communicates with and they ended up getting the overall ratings shown in Table 1:

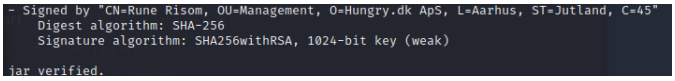

Fig. 3: As can be seen then the key is only 1024 bits long which is below the default key size and thus considered weak.

URL:	Grade:
api.paypal.com [10]	A+
foodora.com [11]	A
dk.fd-api.com[12]	A
https://disco.deliveryhero.io/ [13]	B
https://hungrydk-ab4c9.firebaseio.com [14]	B

TABLE I: Overall grades for the Qualys report on URL's associated with the app.

The ratings for A and above shows that certificates, protocol support, key exchanges and the cipher strength of any form are considered very great and that it needs a very advanced adversary to crack the TLS sessions. For *deliveryhero* and *hungrydk-ab4c9* then the reason why they only got a B is because they use TLS versions reliant on SHA-1 for older versions of android and IOS which is really insecure. Due to TLS 1.2 and above being the only used ones for *dk.fd-api.com* and *api.paypal.com* then they should ensure data integrity on information transmitted to and from them.

As I briefly explained in Section 2 then I found proof of certificate pinning and root detection which is further proven by the frida logs from my dynamic analysis, the output is shown in figure 4.
Root detection seems to happen in several places when interacting with the app thus making the tool-set I have to work with smaller since I then cannot use root access.[15]

Even though there is the possibility of doing a padding oracle attack and DES cracking on messages sent to and from the server then I have decided not to do it since it would be way above the scope of my adversary's capabilities.

I used Burpsuite to test the application for Man in the middle (MITM) attacks. I was able to intercept

Fig. 4: As can be seen then there is both SSL pinning and Root detection.

packages however since the app uses a lot of trackers then there is a lot of communication which is going on which makes it hard to narrow down the important ones such as login, place order and checkout, and when I check something then it is in the format of either clear text or encrypted and/or obfuscated text from the more important requests and responses. During the testing both with and without a root certificate then I was able to intercept the packages and for the login of a user then I was able to get what is shown in figure 5.

And the response in figure 6 shows the access token.

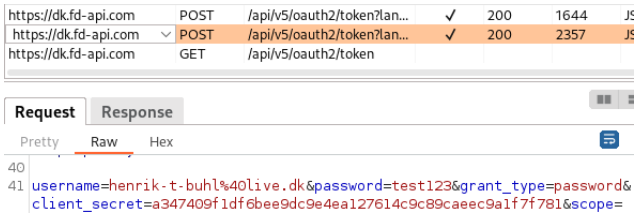

Fig. 5: As can be seen then the username and password are both unencrypted and a malicious adversary could easily obtain these.

I also tried intercepting credit card information, I found that it keeps the card bank identification number unencrypted along with the last 4 digits (probably to look it up or something), it does however seem like the rest of the information is encrypted as seen in figure 7.

This shows that the app isn't perfectly secure however you cannot steal really important data such as

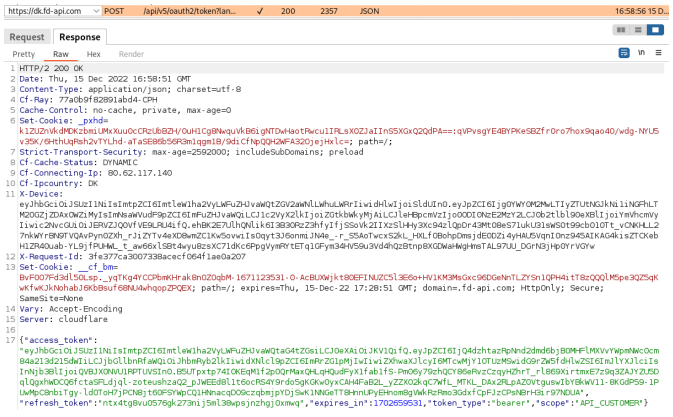

Fig. 6: As can be seen then the access-token is also up for grabs.

Fig. 7: This figure shows that the credit card information is encrypted.

credit card information. Nevertheless if an adversary could login to a users account and if their credit card information is saved (which is a feature in the app) then that could prove problematic if the app doesn't prompt the adversary for some information which only the user could have.

4. Authentication

Authentication is always used as an extra security layer to ensure that a users data is protected. Simple applications use a 1-factor authentication which mostly persists of a login username and password. However there are several more ways to authenticate a person, it can be done using something the user knows such as the already mentioned username and password, it can also be the biometric signature of the user such as a thumb print, thereby something which is part of the user, and at last then it can also be something which the user has such as a key. With more layers of authentication then it becomes more difficult for an adversary to log into another persons account. [16]

The scenario which we have on our hands is that an adversary has gotten hold of a users credentials and

can therefore login on their account. The first they will be prompted with is to login with a username and a password which the adversary can just enter without any problems, however afterwards then the adversary will be prompted to write a 4-digit random code within a time limit, with the code having already been sent to the phone number of the owners account via SMS. Here there is a possible attack which the adversary can use called SIM-swapping where they convince the telecompany that they are the victim and thus gets them to port the victim's phone number to the adversay's SIM.[17] (When I tested this I only used a card with no money on it and thus it didn't save the information for the creditcard and thus I am unsure about the saving credit card info functionality and how it works so a minor discussion will be based on assumptions.) In the case that an adversary gets into a victims account then assuming that the credit card details are saved then for it to ensure that someone can't just steal the info or use it indirectly to buy food then all payments which go through with card should prompt the user to either write the CVC-number or approve it via email. Other than using SIM-swapping which depending on the victim can be time consuming and cost some resources, then the adversary can only try and guess the pin-code where they have only around 10 tries to get it right which is in best case a 1 in 1000 chance of it being possible for them to get in before they have to wait for the next key. Thus this is really difficult to achieve and I suspect that there is a limit to the amount of queries you can do before your device gets completely banned from the app.

Since it is possible to login to another users account by doing a SIM-swapping attack but it would take a while and some resources, then maybe if we inspect the login flow we can perhaps make an illegitimate account for which we won't be held accountable for our actions on. Firstly when you open the app for the first time then you are first asked to give a location, afterwards you will be prompted with an are you a robot query, to ensure that we aren't a bot. This is great to ensure bots cannot start a bunch of sessions. To proceed to buy something then you have to login to the app using an email address, once creating your account then you can choose whether or not you want to use a password, after typing in the name and password then you have to type in a phone number to proceed to payment. Afterwards then you have to select how you want to pay for the food with the choice being either with a credit card or using mobilepay.

Otherwise if an adversary wanted to ensure an illegitimate session which circumvents non-repudiation then they would need a 10 minute mail which is easily achieve-able, a virtual phone number which cannot get linked to them which is a bit difficult to get and at last a virtual credit card to "pay for the food" which is really hard to get and again for this to not be traced back to the adversary then nothing can be linked to them. These are all needed in the case that the attacker can make a MITM attack which makes their query get approved by the server (which I haven't tested for since it would be a direct attack). Thus it is not feasible for our adversary to force query's to get approved without any form of possible ramifications.

If a user finds out their password has been leaked or forgets it then they can reset it by getting a request sent to them via email. Once they get the prompt for resetting their password over email then they can even set it to the old password which might not be a good idea in the case of there having been a leakage of passwords before.

The authentication mechanisms are in principle really good having it be 2-factor makes it tough for an intruder to get through, however I think that the login session could be better secured, with some encryption being used to mask the users email and password since an adversary could steal a password which the user also uses somewhere else such as their bank for example. Furthermore then I think that having the OTP be handled over SMS is a really bad idea since that makes SIM-swapping attacks possible and at last, the 4-digit authentication code should be set to 6 digits or higher since I believe a 0.1% of getting into an account is too high.

5. PRIVACY

A lot of sensitive data gets collected by this app and since it is a fast food app then it makes sense that it would collect some information to make the overall interactive experience easier for the user. For example then it gets personal information about the user such as their location/address, phone number (possibly also other contact details such as email) and the users interactions with the app. By looking through the AndroidManifest then I was able to find some of the more interesting permissions which have been listed below:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:maxSdkVersion="28"android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Listing 3: This listing shows all the significant permissions which the app prompts a user with.

For the location then it makes sense that the app can track the user once they are using it to order food, furthermore then contact details on said user is also rather sensible since the courier needs some way to contact the user in certain cases. However some of the other permissions are a bit privacy intrusive and doesn't make much sense in regards to the apps functionalities, such as the READ_CONTACTS permission which gives the app permission to read data about the contacts stored such as people which the user has interacted with over the device.[18] There are also the READ and WRITE to external storage which in Android versions 10 and below gives the app permission to access any file outside the app-specific directories, this is however different for newer version of android which only allow the app to have access to its own domain and target the parts of the file system which the app uses.[19] The permission for camera usage allows the application to collect images that the camera sees at any time, it is therefore intrusive and I believe it might even be unnecessary since the only time where it would make sense for a user to use it would be if they were writing a review for one of the restaurants.[20] READ_PHONE_STATE is also a bit excessive since it extracts the current cellular network information, the status of any ongoing calls, and a list of any accounts registered on the device. I think it is safe to say that the only thing which is relevant from READ_PHONE_STATE for the app to know is the users phone number which makes READ_PHONE_NUMBERS more suitable since it is less intrusive. [21]

Furthermore then there also seems to be a number of different trackers which can be seen listed below:

- Adjust
- Braze
- Facebook
- Google Analytics
- Google CrashLytics
- Google Firebase Analytics
- Google Tag Manager
- Qualtrics
- UXCAM

The trackers which collect the most data and is the most privacy intrusive is Braze and UXcam, Braze collects information about the device application usage and personally identifiable information, which it uses to make targeted ad campaigns. UXCam collects information about all the users minor interactions in the app and even records them. Then there is also Google CrashLytics which recieves and analyzes crash reports and uses the data collected from apps to advertise more efficiently on social medias. Facebook is also used which can link the user to their Facebook account and collect data on the user and the app such that it can make targeted ads on Facebook. At last then there are the minor offenders such as Adjust, Google Firebase Analytics, Google Analytics and etc, which mostly just seem to collect data for the sake of the app developer such that they can find out what the users interact with.[22]

6. Conclusion

By having tested the app and weighing heavily on the IA framework then it is safe to say that it is a rather secure app with only some potential exploits being possible albeit those would take a while to pull off and most likely wouldn't occur since it just is a fast food app. Some of the confidentiality requirements weren't met and should ideally encrypt the username and password when it gets transmitted, but luckily then this oversight isn't too severe if the assumption of a prompt showing up to view credit card details is made. Authenticity can be improved upon, such as having the sent pin-code for the user be sent through an authenticator or something alike such that SIM-swapping isn't plausible. Furthermore then the pin-code should be longer for a smaller probability of an adversary guessing their way in. Due to the fact that authenticity can't be upheld completely and a malicious adversary can in theory login to a victims account where the victim might have saved their credit card details for faster checkout, then if there is no prompt as discussed earlier then they could order a pizza on behalf of the victim and thus non-repudiation would not be upheld. Since TLS 1.2 and higher is used for the really sensitive information such as credit card details then integrity is upheld. Since there are bigger servers used for foodora in other north European countries then one can conclude that since they can maintain a larger network that they also can maintain a small one such as this. The app is rather privacy intrusive asking for way more permissions than is needed, and having several trackers which can monitor every move of the user, identify who they are and make targeted ad campaigns based on the interactions they have made in the app. For the permissions then I believe that the only ones which are needed for the app are location based and phone number associated ones. The trackers such as braze and UXCAM should be reconsidered given how much information they collect on the users.

References

[1] Rune Risom. *Foodora: Food delivered APK for Android download*. Nov. 2022. URL: https://apkpure.com/foodora-food-delivered/hungry.dk.android.

[2] *Delivery hero acquires Danish Food Delivery Service Hungry: Delivery Hero*. Oct. 2021. URL: https://www.deliveryhero.com/newsroom/delivery-hero-acquires-hungry/.

[3] Hungry.dk. *Foodora: Food delivered*. July 2013. URL: https://apps.apple.com/dk/app/foodora-food-delivered/id666759664.

[4] *Foodora: Mad til levering – apps i google play*. URL: https://play.google.com/store/apps/details?id=hungry.dk.android&hl=da&gl=US.

[5] *Foodora se er bedømt "Dårlig" med 1,3 / 5 på trustpilot*. URL: https://dk.trustpilot.com/review/foodora.se?page=2.

[6] *Hungry Er Bedømt "Middel" Med 2,8 / 5 på trustpilot*. Mar. 2013. URL: https://dk.trustpilot.com/review/www.hungry.dk?page=289.

[7] *Information assurance(ia): Definition amp; explanation*. URL: https://www.itgovernanceusa.com/information/information-assurance.

[8] Ozgur Alp. *Ozguralp/gmapsapiscanner*. URL: https://github.com/ozguralp/gmapsapiscanner.

[9] Taekeon Lee et al. "Padding Oracle Attacks on Multiple Modes of Operation". In: *Information Security and Cryptology – ICISC 2004*. Ed. by Choon-sik Park and Seongtaek Chee. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 343–351. ISBN: 978-3-540-32083-8.

[10] *SSL Labs api.paypal.com*. URL: https://www.ssllabs.com/ssltest/analyze.html?d=api.paypal.com.

[11] *SSL Labs foodora.com*. URL: https://www.ssllabs.com/ssltest/analyze.html?d=foodora.com.

[12] *SSL Labs dk.fd-api.com*. URL: https://www.ssllabs.com/ssltest/analyze.html?d=dk.fd-api.com.

[13] *SSL Labs disco.deliveryhero.io*. URL: https://www.ssllabs.com/ssltest/analyze.html?d=disco.deliveryhero.io.

[14] *SSL Labs hungrydk*. URL: https://www.ssllabs.com/ssltest/analyze.html?d=hungrydk-ab4c9.firebaseio.com.

[15] MobSF Team. *Owasp-MSTG/0x05j-testing-resiliency-against-reverse-engineering.md at master · MOBSF/Owasp-MSTG*. Nov. 2020. URL: https://github.com/MobSF/owasp-

mstg / blob / master / Document / 0x05j - Testing - Resiliency - Against - Reverse - Engineering . md # testing-root-detection-mstg-resilience-1.

[16] *Understanding authentication, authorization, and encryption*. URL: https://www.bu.edu/tech/about/security-resources/bestpractice/auth/.

[17] *Sim Swap Scam*. Nov. 2022. URL: https://en.wikipedia.org/wiki/SIM_swap_scam.

[18] Niels van Hove. *Android distribution*. URL: http://androidpermissions.com/permission/android.permission.READ_CONTACTS.

[19] *Data and File Storage Overview nbsp;: nbsp; android developers*. URL: https://developer.android.com/training/data-storage.

[20] *Manifest.permission nbsp;: nbsp; android developers*. URL: https://developer.android.com/reference/android/Manifest.permission#CAMERA.

[21] *Manifest.permission nbsp;: nbsp; android developers*. URL: https://developer.android.com/reference/android/Manifest.permission#READ_PHONE_STATE.

[22] *Foodora*. URL: https://reports.exodus-privacy.eu.org/en/reports/de.foodora.android/latest/#permissions.