

# Cover page

**Exam information**  
240172E146 - Fundamentals of Computer Security

**Handed in by**  
Michael Quach  
201506330@post.au.dk

**Exam administrators**  
Ina van Gaever Lautrup Thers  
ithe@au.dk  
☎ +4541893076

**Assessors**  
Diego F. Aranha  
Examiner  
dfaranha@eng.au.dk

Rune Hylsberg Jacobsen  
Internal co-examiner  
rhj@eng.au.dk  
☎ +4541893252

## Hand-in information

# Fundamentals of Computer Security

## Final Project

Friday 24<sup>th</sup> January, 2020

Michael Quach  
*Dept. of Computer Engineering*  
*Aarhus University*  
Aarhus, Denmark  
201506330@post.au.dk

### I. INTRODUCTION

MobilePay is a payment application, developed by Danske Bank, for Android and Apple smartphones. The service launched in 2013 and has today more than four million private users and 100 000 business users. Using this application, a user can make a payment to another user by specifying only the recipient's phone number. The application stores data such as the user's credit card numbers, civil registration number (CPR), transaction history, NemID and more.

Given that MobilePay is primarily used for making payments, the motive of an adversary is probably financial gains. What may be the most apparent way of gaining money illegitimately, using this app, is by making transactions through a victim's account. Another possible way is to gain valuable information of users, and then try to sell the data. Other than financial gains, the motives could be prestige, fun, curiosity, or maybe even espionage and politics.

We can likely assume that an adversary will have perseverance and significant computing power because of the potential gains. A successful attack towards such an application could, among other things, result in theft, and damage to the MobilePay system and Danske Bank's reputation. The risks associated with the application makes it ideal to perform a security analysis of the application.

Since the application is built for mobile devices such as smartphones, it has a large attack surface by default. Smartphones can communicate in multiple ways such as bluetooth, WiFi and NFC. The smartphone has an operating system, apps, a screen, and physical ports. The application itself can use the camera for debit card scanning, QR codes, and attaching images to payments. Furthermore, it uses multiple third party libraries.

#### A. Security properties

The application should enforce strong authentication, so we can be confident that the user is who he claims to be. An adversary able to claim the identity of another user can be devastating, as they are able to make transactions or retrieve valuable information.

It is vital that the system also has high availability to serve the many million users. Though, this is mostly handled by the backend architecture. And of course, all sensitive data must be kept confidential and integrity must also be maintained. For instance, it must not be possible for an adversary to read or modify transactions to change amounts and recipients.

Another important property is non-repudiation, which assures that one can not falsely deny having been involved in a communication. If the user claims to not have made a certain transaction, the claim can be determined true or false using e.g. public key cryptography.

### II. SOFTWARE SECURITY

#### A. APK Decompilation

The MobilePay APK has been decompiled using the following tools:

- apktool 2.4.1 [1]
- dex2jar 2.0 [2]
- JD-GUI 1.6.6 [3]

Looking at the source code, it seems that most of the code has been minimized or obfuscated before being used for production. For instance, looking at lst. 1 we see what seems to be certificate pinning.

```
private final void a(h.a pa) {  
    if (gl.e()) {  
        pa.a("api.mobilepay.dk", new String[]  
            { "sha1/95MZ79/B9SD7rIVVLPLSjlq5ygs=" });  
  
        pa.a("api.mobilepay.dk", new String[]  
            { "sha1/t38+NE5/DpXNfw8VukxnF9k24cA=" });  
  
        pa.a("api.mobilepay.dk", new String[]  
            { "sha1/e/dzc3hb19Bvm/IwUKVHoB8kPLQ=" });  
    }  
}
```

Listing 1: Possible certificate pinning.

On lst. 2 the allowed protocol for the SSL socket is set to TLS 1.2 only, which is a good idea as this disables the more insecure TLS 1.1 and 1.0. Though, it may be a good idea to

support TLS 1.3 to improve security for devices that are able to do so.

```
private Socket patch(Socket socket) {
    if (socket instanceof SSLSocket)
        ((SSLSocket) socket).setEnabledProtocols(
            TLS_V12_ONLY);
    return socket;
}
```

Listing 2: Socket configuration.

The application makes use of WebSockets. An example can be seen on lst. 3. This is probably due to the need for fast communication. Furthermore, an access token is sent on each “acceptPayment” request. I was unable to find where the initialization of the access tokens takes place.

```
public void acceptPayment(@NotNull String ps1,
    @NotNull String ps2, @NotNull String ps3,
    @NotNull String ps4) {
    j.b(ps1, "accessToken");
    j.b(ps2, "paymentId");
    j.b(ps3, "cardId");
    j.b(ps4, "orderId");
    this.webSocket.acceptPayment(ps1, ps2,
        ps3, ps4);
}
```

Listing 3: Websocket and access token usage.

The Android packages shown on table I were found in the MobilePay APK. Many third party libraries are used, which increases the software attack surface. Most of the packages are for special UI elements (all except the bottom five).

The *io.card.payment* package handles credit card scanning using the built-in camera. In section V-A the four packages at the bottom of the table are briefly described.

### III. NETWORK SECURITY

Looking through the externalized strings, several API URLs have been found. These are shown on table II.

For all these domains, a quick analysis of the SSL configurations has been performed by using the Qualys SSL Server Test tool [4]. The developer API at *developer.mobilepay.dk*, has also been scanned. On table III we see the domains perform well on the SSL test.

The domain *api.qa.mobilepay.dk* could not be analysed because the domain name is unresolvable. In general, the SSL configurations allow weak cipher suites which may be a problem. This is very likely intentional as it is a tradeoff between wanting to support older devices and having very tight security.

#### A. Man-in-the-middle (MITM)

To inspect the network traffic in detail, a MITM attack was attempted using *mitmproxy* [5]. The used device is an emulated Nexus 5X on API-level 27, which was set up to use the *mitmproxy*. This allowed traffic from the browser to be seen, as shown on fig. 1.

For MobilePay the traffic could not be intercepted; shown on fig. 2. This is most likely due to the certificate pinning code in lst. 1.

TABLE I: Android packages.

Package name
fr.castorflex.android.smoothprogressbar
in.srain.cube.views.ptr
it.sephiroth.android.library.tooltip
me.iwf.photopicker
com.airbnb.lottie
com.bumptech.glide.load
com.github.rubensousa.gravitysnaphelper
com.gitonway.countzero
com.makeramen.roundedimageview
com.malinskiy.superrecyclerview
com.marshallchen.ultimateryclerview
com.mingle.widget
com.nhaarman.supertooltips
com.pnikosis.materialishprogress
com.romainpiel.shimmer
com.soundcloud.android.crop
com.squareup.picasso
com.txusballesteros.widgets
com.trifork.scanandpay.ui
io.card.payment
com.adform.adformtrackingsdk
com.appdynamics.eumagent.runtime
com.crashlytics.android
com.mixpanel.android

TABLE II: API URLs found after decompilation.

ID	URL (HTTPS)
cloud_server_prod	images.mobilepaypos.net/api
firebase_database_url	api-project-948974957432.firebaseio.com
intrepid_issuer	api.mobilepay.dk/private
intrepid_url_prod	api.mobilepay.dk
intrepid_url_qa	api.qa.mobilepay.dk
intrepid_url_sandboxprod	api.sandbox.mobilepay.dk
sg_server_public_sandboxprod	sandboxprod-possqwm.mobilepay.dk/API/P2I
sg_server_rei_sandboxprod	sandboxprod-possqwm.mobilepay.dk/API/REI
sg_server_secure_sandboxprod	sandboxprod-possqwm.mobilepay.dk/API/P2S

TABLE III: SSL ratings of domains.

URL (HTTPS)	SSL Rating
images.mobilepaypos.net/api	A
api-project-948974957432.firebaseio.com	A+
api.mobilepay.dk	A+
api.qa.mobilepay.dk	N/A
api.sandbox.mobilepay.dk	A+
sandboxprod-possqwm.mobilepay.dk	A+
developer.mobilepay.dk	A+




	https://shared.tv2.dk/t2breakingbar	GET	200	22b	43ms
	https://ping.chartbeat.net/ping?h=tv2.dk&p=%2Fnode%2F2242...	GET	200	43b	230ms
	https://ads.pubmatic.com/AdServer/js/showad.js	GET	304	0	44ms

Fig. 1: Intercepted browser traffic.

```
i 127.0.0.1:56578: clientconnect
i 127.0.0.1:56578: clientdisconnect
i 127.0.0.1:56580: clientconnect
i 127.0.0.1:56580: clientdisconnect
i 127.0.0.1:56582: clientconnect
```

Fig. 2: Snippet of mitmproxy event log after starting the MobilePay application.

#### IV. AUTHENTICATION

MobilePay uses two-factor authentication. The first factor is having access to the mobile phone, which is “something you own”, since a payment must happen from the users mobile phone. A problem with this type of authentication is that the security depends on how well the user can hide or protect the object which can authenticate them. As a second authentication factor MobilePay uses “something you know”, which requires the user to type a personal four-digit PIN code in order to access the application.

The length of the PIN code can be a problem because the application is often used in public where people next to the user, pretty easily, can see and memorize the PIN. Furthermore, an adversary may also try to randomly guess the PIN. This is not very likely to work but since there are only four numbers and the application allows up to four attempts until the user is blocked, it still has a 1/2500 probability of succeeding. Increasing the length of the PIN or allowing a bigger alphabet may make it more difficult to break in. Of course, the user must make sure to use a strong unique password for the application in case other accounts, using the same password, are compromised.

MobilePay also allows the use of a biometric authentication mechanism, “something you are”, as the second authentication factor. Instead of using a PIN code, intrinsic biometric information can be used for fingerprint scanning or face recognition. This makes it easier to use the application as no typing is involved, but is only useful security-wise if the application can recognise the user precisely and accurately enough. If this authentication mechanism gives false negatives, the user can still use a PIN code to gain access.

##### A. New phone

It is possible to login to the same account using a different device; for instance when the user has bought a new phone.

The authentication flow on the new phone is as follows [6]:

- 1) Choose login using existing user
- 2) Type phone number and PIN code
- 3) Type activation code received via SMS
- 4) Type CPR number

This may pose a problem assuming that the adversary knows the phone number, PIN code and CPR number because it is possible to take over someone’s phone number in a SIM swapping fraud, and thereby get the activation code needed. This allows authentication without having the user’s mobile phone. The adversary gains control by calling the mobile

carrier, impersonating the user using information found by methods such as using social media and sending phishing emails. By doing this, the adversary can convince the carrier to activate another SIM card which is in the possession of the scammer [7].

##### B. Forgotten password

The authentication flow for creating a new password is as follows [8]:

- 1) Press question mark
- 2) Select forgotten password option
- 3) Type activation code received via SMS
- 4) Type last four digits of card number

We see that this flow also involves an activation code via SMS. In case an adversary is able to get the user’s card number (or at least the last four digits), they can change the PIN code using SIM swapping.

##### C. Adding debit cards

When a user wants to add a new debit card to the application, a security solution named 3D-Secure is used to verify that the user actually owns the debit card. This solution basically sends an SMS with a one-time password (OTP) to the phone number registered for the debit card beforehand. Again, the OTP can be routed to the adversary by SIM swapping.

Although SIM swapping can occur, using SMS verification is still a decent layer of defense. Sending activation codes via SMS is a tradeoff between convenience and security. Using NemID instead could make MobilePay more secure but also make it too much of a bother.

#### V. PRIVACY

##### A. Trackers

Looking through the source code, several packages have been found which may be privacy-invasive:

- *com.adform.adformtrackingsdk*: allows tracking of mobile app events, such as app installs, starts and more [9].
- *com.appdynamics.eumagent.runtime*: performance monitoring of applications [10].
- *com.crashlytics.android*: functionality for lightweight crash reporting [11].
- *com.mixpanel.android*: tracking of user behaviour [12].

The traffic of these packages could not be analysed due to an SSL error by using the mitmproxy certificate, likely caused by the certificate pinning. It seems that the last package, *com.mixpanel.android*, is the worst offender though, as this package allows the people behind MobilePay to track user segments, individual user profiles and their activity, and more.

##### B. Biometric data storage

Use of fingerprints and facial data can be considered privacy intrusive, so it is important that these types of data are handled cautiously. Both Android and Apple have security features to make it much more difficult for adversaries to retrieve the biometric data.

According to Apple, the Face ID and Touch ID data is encrypted and stored locally on the mobile devices. Keys to the data are also stored locally using Secure Enclave, which is a hardware-based key manager that runs its own operating system. It should not be possible to access it even by the OS or any application running on it [13] [14] [15].

Similarly, Android has a security feature called Trusted Execution Environment (TEE). The TEE also has its own hardware and runs its own OS. The biometric data is analysed and encrypted in the TEE and stored in trusted memory that is inaccessible to the main CPU [16].

## VI. CONCLUSION

Overall the MobilePay application seems to be well secured. TLS is used for all communication with backend servers which provides confidentiality and integrity. In itself, TLS does not provide non-repudiation though, and it is difficult to see in the decompiled code where and how this is handled.

It seems to be the case that certificate pinning is used as well; making it more difficult to perform a MITM attack.

The only enabled protocol for the SSL sockets is TLS 1.2, which provides additional security by removing the possibility of using earlier weaker versions of TLS. On the other hand, it also disables the use of TLS 1.3.

MobilePay uses packages to track user behaviour, which may be privacy-invasive. Biometric information is not stored in MobilePay nor on their servers, but rather on local secured hardware according to Apple and Google.

The authentication flows often make use of activation codes sent via SMS, which is no longer recommended for security because of attacks such as SIM swapping. However, because of usability/convenience concerns, this probably won't change right away, if at all.

## REFERENCES

- [1] C. Tumbleson and R. Wiśniewski. Apk tool. [Online]. Available: <https://ibotpeaches.github.io/Apktool/install/>
- [2] pxb1988. The mnist database of handwritten digits. [Online]. Available: <https://github.com/pxb1988/dex2jar>
- [3] Java decompiler gui. [Online]. Available: <https://github.com/java-decompiler/jd-gui>
- [4] Qualys. Qualys ssl server test. [Online]. Available: <https://www.ssllabs.com/ssltest/>
- [5] A. Cortesi, M. Hils, and raumfresser. mitmproxy. [Online]. Available: <https://mitmproxy.org/>
- [6] MobilePay. Jeg har fået ny mobil. hvad gør jeg? [Online]. Available: <https://www.mobilepay.dk/hjaelp/mobilepay-til-private/kom-i-gang/ret-oplysninger/jeg-har-faaet-ny-mobil-hvad-goer-jeg>
- [7] A. Grace Johansen. Sim swap fraud explained and how to help protect yourself. [Online]. Available: <https://us.norton.com/internetsecurity-mobile-sim-swap-fraud.html>
- [8] MobilePay. Jeg har glemt min kode. hvad gør jeg? [Online]. Available: <https://www.mobilepay.dk/hjaelp/mobilepay-til-private/kom-i-gang/ret-oplysninger/jeg-har-glemt-min-kode-hvad-goer-jeg>
- [9] adform. adform-tracking-android-sdk. [Online]. Available: <https://github.com/adform/adform-tracking-android-sdk>
- [10] Appdynamics. Why appdynamics? [Online]. Available: <https://www.appdynamics.com/company/what-is-appdynamics>
- [11] fabric.io. Crashlytics (legacy). [Online]. Available: <https://fabric.io/kits/android/crashlytics/summary>
- [12] Mixpanel. Mixpanel. [Online]. Available: <https://mixpanel.com/home/>
- [13] Apple. About touch id advanced security technology. [Online]. Available: <https://support.apple.com/en-us/HT204587>
- [14] ——. About face id advanced technology. [Online]. Available: <https://support.apple.com/en-us/HT208108>
- [15] ——. Storing keys in the secure enclave. [Online]. Available: <https://www.mobilepay.dk/hjaelp/mobilepay-til-private/kom-i-gang/ret-oplysninger/jeg-har-faaet-ny-mobil-hvad-goer-jeg>
- [16] J. Hildenbrand. How does android save your fingerprints? [Online]. Available: <https://www.androidcentral.com/how-does-android-save-your-fingerprints>