Thorough Analysis of a Danish Mobile Banking Application

Kasper Lauge Madsen

Abstract— **In this report a thorough analysis of the mobile application "Jyske Mobilbank" was done. The threat model was analysed, and several potential threats were found. Confidentiality and integrity were deemed especially important security properties for this sensitive mobile application. Confidentiality was found violated by missing use of public key pinning and an outdated server configuration. The integrity was preserved by secure two-factor authentication on transactions and similar operations. Several third-party cookies are supposedly used in the banking infrastructure, but no suspicious use was found during the network and code analysis of the application.**

I. INTRODUCTION

During the past years mobile applications take care of more operations for people than before. The mobile applications are handling more sensitive information and the tasks they are handling are important to be done in a secure and reliable fashion. Due to this development it is found important to analyse one of such mobile applications to verify the responsible way of handling important operations and sensitive information. The mobile application which is being analysed in this report is "Jyske Mobilbank" by Jyske Bank[1]. During this report the application will first be described, followed by a documentation of the threat model and important security properties of the application. After that the application will be analysed and important aspects of the application will be identified. Then an experiments and results section will follow where the application will be decompiled to be able to analyse the underlying code. The network activity will be monitored and documented to ensure proper networking done by the application. The authentication mechanisms will be analysed and discussed in detail and the privacy which the application offer will be documented. After this the results will be discussed. A conclusion will then follow.

II. DESCRIPTION OF THE MOBILE APPLICATION

The mobile application analysed in this report is "Jyske Mobilbank" by the Danish company Jyske Bank. Jyske Bank is a Danish bank, and the mobile application offers services for the customers of Jyske Bank. Through the application customers can get an overview of their banking accounts, transfer money, invest, receive important information from the bank, pay bills and similar banking operations. The services offered by Jyske Mobilbank is thereby considered very security and privacy sensitive. The application is both offered on iOS and Android platforms. In this report the Android application will be analysed due to easier access to the underlying application code and emulators.

A. Threat model

The threat model of financial services is large due to the importance of the information regarding financial activities and status. Some of the important ones are:

1) Individual people: People could be interested in stealing money from other people by exploiting vulnerabilities in the application. It would also serve as a problem if ones financial status was public as other people could discriminate based on that.

2) Stores/Shops: If companies know the financial status of a person, they could set their price according to the buying power of that individual.

3) Criminal organizations and individuals: If financial status could be public available, people with a large amount of money would by a large probability be at higher risk than other people of targeted attacks. Criminals would anticipate a larger probability of a high money outcome by attacking people which are worth more money than others.

B. Relevant security properties

Financial information is very sensitive and thereby many security properties are relevant. The security properties which are most relevant are:

1) Confidentiality: Financial status and activity must be kept confidential. Therefore, all activities performed by the application must be kept secret from outsiders. Outsiders should not be allowed to eavesdrop on non-encrypted financial activities as this could have severe consequences if attackers would be provided with this information.

2) Integrity: The financial status and the activity regarding that must keep its integrity. There must be an assurance that the information cannot be tampered with and that the financial information is truthfully.

3) Availability: Although not as important as the aforementioned properties, availability is also an important aspect for the application. Looking at only the mobile application, most financial activities are typically scheduled to be done on a remote server, making the availability less important at the app, as important transactions and such would be done even though the application was not available. Looking at the system in general, availability becomes more important. Customers of a bank should be able to pay important bills in time for services. If they do not have the possibility for this, it can have severe consequences.

4) Authenticity: To keep information secret authenticity also becomes an important security property. The application should only allow authenticated people to see their own bank account. Furthermore, users which are not authenticated should not be able to see any bank accounts.

5) Non-repudiation: Users of the mobile application, shops and the bank should be able to trust that the user in fact did the operation which was done. Therefore, the user should not be able to deny an operation done through the mobile application which the user in fact did.

6) Reliability: The application should perform in a reliable way. An example is that the user for instance should not by accident be transferring some other amount of money than the one listed by the application.

III. Analysis

To achieve the necessary security properties of the mobile application several things are expected to be done. First of all, to achieve the necessary confidentiality, TLS/HTTPS is expected to be used. This is to be done as a means of making end-to-end encryption of all communication between the mobile application and servers. This will ensure that eavesdroppers will not be able to make sense of the communication made by the application.

It is expected that the mobile application use up-to-date encryption algorithms and schemes, and up-to-date hashing functions. Several previous state-of-the-art techniques are not recommended anymore. This includes for instance DES and SHA-1. The mobile application should only be using the ones which are still recommended.

The mobile application will be using one or more backend services. These will be expected to get a high grade on Qualys SSL Labs[2] which will do a thorough SSL/TLS and Web public key infrastructure (PKI) analysis.

The application should do a decent job in preventing man-in-middle attacks. This means that the application should be communicating with services signed by trusted root certificates. Furthermore, the application should not only rely on third party Certificate Authority (CA)'s but also prevent faulty signed certificates to be used as trusted certificates. This can be overcome by public key pinning.

The authentication mechanisms of the application are expected to be done in a state-of-the-art fashion. This should ensure that only authenticated users can have access to their information. Authentication can be done in more ways. Usually it is based on either:

- Something you know (eg. a password/pin)
- Something you have (eg. a physical device displaying a code)
- Something you are (eg. fingerprint or face scan)

The current best practice is to use at least two authentication mechanisms to increase the difficulty of malicious use by attackers.

The mobile application should not introduce privacy invasive behavior which spread sensitive information to non-relevant third parties. Therefore, it is expected to have a small amount of third-party integrations, and they are expected not to carry sensitive information like account balance and similar.

IV. Experiments and Results

To investigate the security mechanisms of the mobile application, several experiments have been setup.

A. Experimental setup

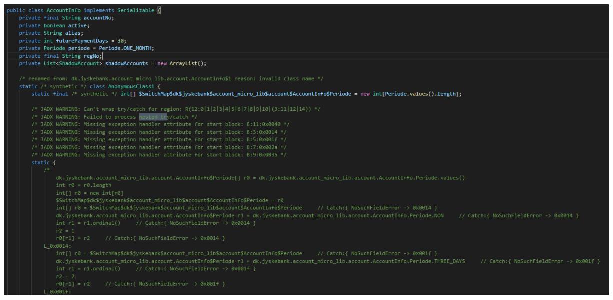
First of all, the Android application has been downloaded through Google Play to an Android device emulator. Specifically the Google Pixel 2, with the operating system Android 10.0. The application "APK extractor" is installed to retrieve the raw APK file. When the raw APK file is retrieved, most of the original Java code is retrieved using the application jadx[3]. The code is then available for analysing.

To test the network setup, a Wireshark[4] instance is run to see the different servers connected to. Furthermore, a man-in-the-middle attack is tried. This is done by first making an arpspoof between the local router and the Android emulator. Afterwards, the program mitmproxy[5] is used to get a hold of the requests made by the application. This is done on both the Android emulator running Android version 10.0, but also an emulator running Android version 6.0.

The application is then analysed qualitatively by using it to get an understanding of the authentication mechanisms and potential privacy invasive behavior.

B. Results

The first thing discovered by decompiling the APK is that no or very little obfuscating is used.



```
public class AccountInfo implements Serializable {
    private final String accountNo;
    private boolean active;
    private String alias;
    private int futurePaymentDays = 30;
    private Periode periode = Periode.ONE_MONTH;
    private final String regNo;
    private List<ShadowAccount> shadowAccounts = new ArrayList();

    /* renamed from: dk.jyskebank.account_micro_lib.account.AccountInfo$1 reason: invalid class name */
    static /* synthetic */ class AnonymousClass1 {
        static final /* synthetic */ int[] $SwitchMap$dk$jyskebank$account_micro_lib$account$AccountInfo$Periode = new int[Periode.values().length];

        /* JADX WARNING: Can't wrap try/catch for region: R(12:0|1|2|3|4|5|6|7|8|9|10|(3:11|12|14)) */
        /* JADX WARNING: Failed to process nested try/catch */
        /* JADX WARNING: Missing exception handler attribute for start block: B:11:0x0040 */
        /* JADX WARNING: Missing exception handler attribute for start block: B:3:0x0014 */
        /* JADX WARNING: Missing exception handler attribute for start block: B:5:0x001f */
        /* JADX WARNING: Missing exception handler attribute for start block: B:7:0x002a */
        /* JADX WARNING: Missing exception handler attribute for start block: B:9:0x0035 */
        static {
            /*
                dk.jyskebank.account_micro_lib.account.AccountInfo$Periode[] r0 = dk.jyskebank.account_micro_lib.account.AccountInfo.Periode.values()
                int r0 = r0.length
                int[] r0 = new int[r0]
                $SwitchMap$dk$jyskebank$account_micro_lib$account$AccountInfo$Periode = r0
                int[] r0 = $SwitchMap$dk$jyskebank$account_micro_lib$account$AccountInfo$Periode     // Catch:{ NoSuchFieldError -> 0x0014 }
                dk.jyskebank.account_micro_lib.account.AccountInfo$Periode r1 = dk.jyskebank.account_micro_lib.account.AccountInfo.Periode.NON     // Catch:{ NoSuchFieldError -> 0x0014 }
                int r1 = r1.ordinal()     // Catch:{ NoSuchFieldError -> 0x0014 }
                r2 = 1
                r0[r1] = r2     // Catch:{ NoSuchFieldError -> 0x0014 }
            L_0x0014:
                int[] r0 = $SwitchMap$dk$jyskebank$account_micro_lib$account$AccountInfo$Periode     // Catch:{ NoSuchFieldError -> 0x001f }
                dk.jyskebank.account_micro_lib.account.AccountInfo$Periode r1 = dk.jyskebank.account_micro_lib.account.AccountInfo.Periode.THREE_DAYS     // Catch:{ NoSuchFieldError -> 0x001f }
                int r1 = r1.ordinal()     // Catch:{ NoSuchFieldError -> 0x001f }
                r2 = 2
                r0[r1] = r2     // Catch:{ NoSuchFieldError -> 0x001f }
            L_0x001f:
```

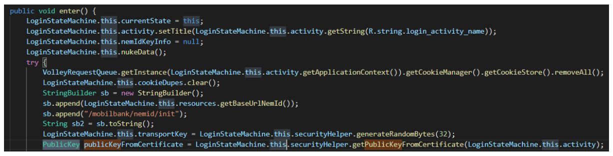
Fig. 1. A part of the decompiled code, which the jadx program could not decompile due to nested try/catch statements. The intermediate language (IL) code is available in a comment.

As seen in figure 1 some of the code cannot be decompiled to the original java code because of some nested try/catch statements. The places where this is found though seems arbitrary and not especially more security sensitive than other places. Thereby obfuscation does not seem to have been used intentionally. This could have made an attack more difficult, as all code is public based on the APK. Obfuscation is though not seen as a huge security factor as the code in theory always can be retrieved.

The next thing done with the decompiled code is searching through common cryptography keywords and analysing the use of it. The keywords searched for was:

- DES
- AES
- SHA
- RSA
- publickey (and similar)
- encrypt

What was especially notable was that extra layers of encryption were being used in the login flow of the application.



```java
public void enter() {
    LoginStateMachine.this.currentState = this;
    LoginStateMachine.this.activity.setTitle(LoginStateMachine.this.activity.getString(R.string.login_activity_name));
    LoginStateMachine.this.nemIdKeyInfo = null;
    LoginStateMachine.this.nukeData();
    try {
        VolleyRequestQueue.getInstance(LoginStateMachine.this.activity.getApplicationContext()).getCookieManager().getCookieStore().removeAll();
        LoginStateMachine.this.cookieDupes.clear();
        StringBuilder sb = new StringBuilder();
        sb.append(LoginStateMachine.this.resources.getBaseUrlNemId());
        sb.append("/mobilbank/nemid/init");
        String sb2 = sb.toString();
        LoginStateMachine.this.transportKey = LoginStateMachine.this.securityHelper.generateRandomBytes(32);
        PublicKey publicKeyFromCertificate = LoginStateMachine.this.securityHelper.getPublicKeyFromCertificate(LoginStateMachine.this.activity);
```

Fig. 2. In the login flow a randomly generated transport key is first established between the third-party bank server and the client, which is encrypted with a public RSA key. Then, AES encryption is used to transfer username and password over the connection. This ensures that even though an attacker gets hold of a man-in-the-middle connection he cannot read username and password.

In figure 4 it is seen that a public key and a certificate are hardcoded into the applications resources. This is done to ensure a secure encryption from the client to the servers. In figure 2 and figure 3 the code actually using the keys is seen. It is seen in both figures that the first thing happening is that the client generates a random 32 byte key. This key is encrypted under the public key of the server and transmitted to the server. The client then encrypts the login data under this newly established symmetric key using AES. This means that besides the TLS encryption made over the HTTP request another symmetric encryption layer is made. Even though the login flow uses these two different random keys (one for the NemID and one for Jyske Bank), the keys do not seem to be used for more than the login flow by looking at the code.

In the code two certificates besides the one hard coded in the resources were found. The reason for these certificates seemed to be for external resources like fonts, images and similar and not a counter measure for man-in-the-middle attacks against banking operations. To check the code for counter measures against man-in-the-middle attacks (eg. public key pinning), the servers which the client communicates to were identified.

In figure 6 it can be seen that there are mainly three important servers the application communicates with, which is not part of the NemID infrastructure. These servers are:

- https://mobil.bankdata.dk
- https://mobile-services.jyskebank.dk
- https://jyskebank.dk

Furthermore looking at the code more servers were identified:

- https://app-mobile-services-jyskebank-dk-t.jyskebank.dk
- https://dk-jyskebank-mobilebank.firebaseio.com

To look for public key pinning, the certificates from all of these servers were retrieved and the public keys were extracted using openssl[6]. These were then searched for in the entire code base, and were not found present anywhere. The public key and certificate used to establish the login flows did also not share any similarities with the certificates from the servers. This indicates that no public key pinning is implemented.

A man-in-the-middle attack was tried to be carried out. This was first done using the Android Emulator phone running Android version 10.0. In figure 5 the unsuccessful attempt can be seen, with the corresponding error messages. It is seen that the application fails based on the mitmproxy certificate not being in the trust chain of the operating system. This is a result of Android checking every HTTP request against the trust store in the operating system and failing the request if the certificate is not part of the chain. To test for a signed certificate, the mitmproxy certificate was installed on the emulator. The login attempt still failed with the same error though. The reason for this is that Android version 7.0 and above only checks against the original system root certificates, unless it is explicitly programmed into the application. This is a good counter measure for users unintentionally installing root certificates on their phones, but it still wont be able to handle a signed leaf certificate, which is obtainable by for instance using LetsEncrypt[7]. To test this further, an emulator using Android version 6.0 was used instead. Again the mitmproxy root certificate was installed, and the attack was performed.

Due to no man-in-the-middle attack prevention using public key pinning, the attack was successful and the result of it can be seen in figure 7. From the figure the code explained previously can be seen. First of all, the login flow with NemID is carried out. It was noted that even though the requests were unencrypted, the payload was still encrypted, due to the second layer of encryption. This was true for both login flows. This meant that login credentials could not be eavesdropped on. More severe was that the previously mentioned transport key established with Jyske Bank's own backend was not used for other than the actual login flow, leaving sensitive information open for eavesdropping. This meant that information like account balances and transactions could be seen in plaintext, which violated confidentiality of the banking information.

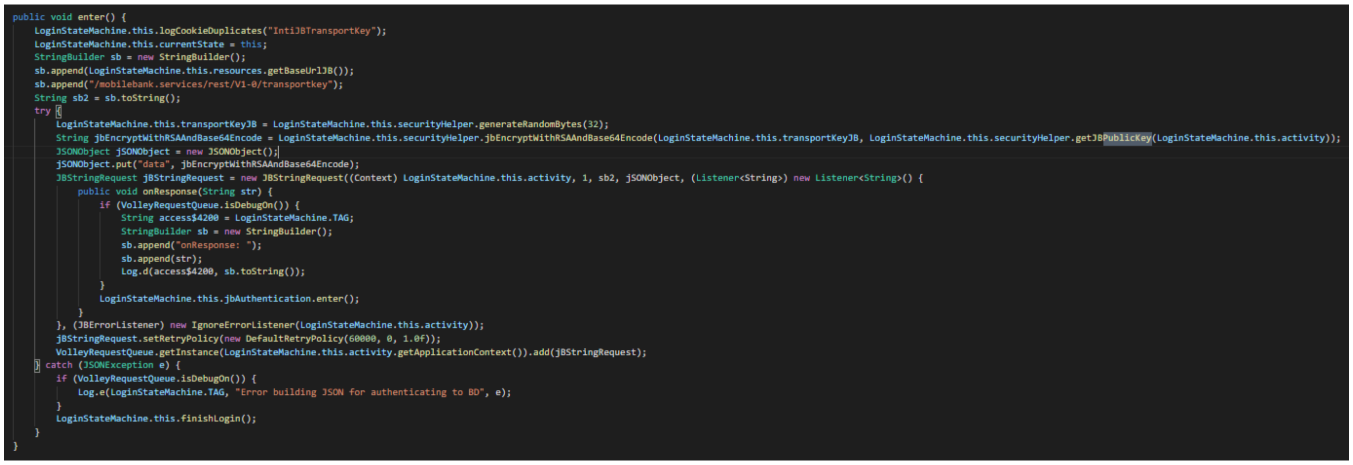
What should be noted though is that operations like actually making transactions and transferring money are protected behind NemID verification, and therefore the same two-factor authorization should be made there. This means that the integrity of operations can be assured.

After the attack was performed, a complete TLS configuration scan was done on the identified servers. This was done using Qualys SSLLabs. The results of the summary can be seen in table 1.

The most notable thing discovered was that what seems to be the main backend server of the application was vulnerable to a fairly new exploit called Zombie POODLE[8] that makes it possible to decrypt TLS packages. The summary of the TLS scan report for the server: https://mobile-services.jyskebank.dk can be seen in figure 8.

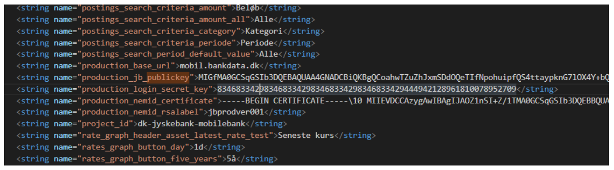
An analysis of potential privacy invasive tools is also done. Most third-party tracking tools use cookies to track information about the users, and Jyske Bank seems to be using the same approach. They have a page located at https://www.jyskebank.dk/omjyskebank/aftaler/cookies where they describe their use of third-party cookies. The ones they are using from are listed in table 2.

Most third-party cookie distributers are marketing related with the exception of "Nets DanID A/S" which are used for login purposes. Google's cookies were used to both Google



```
public void enter() {
    LoginStateMachine.this.logCookieDuplicates("IntiJBTransportKey");
    LoginStateMachine.this.currentState = this;
    StringBuilder sb = new StringBuilder();
    sb.append(LoginStateMachine.this.resources.getBaseUrlJB());
    sb.append("/mobilebank.services/rest/V1-0/transportkey");
    String sb2 = sb.toString();
    try {
        LoginStateMachine.this.transportKeyJB = LoginStateMachine.this.securityHelper.generateRandomBytes(32);
        String jbEncryptWithRSAAndBase64Encode = LoginStateMachine.this.securityHelper.jbEncryptWithRSAAndBase64Encode(LoginStateMachine.this.transportKeyJB, LoginStateMachine.this.securityHelper.getJBPublicKey(LoginStateMachine.this.activity));
        JSONObject jSONObject = new JSONObject();
        jSONObject.put("data", jbEncryptWithRSAAndBase64Encode);
        JBStringRequest jBStringRequest = new JBStringRequest((Context) LoginStateMachine.this.activity, 1, sb2, jSONObject, (Listener<String>) new Listener<String>() {
            public void onResponse(String str) {
                if (VolleyRequestQueue.isDebugOn()) {
                    String access$4200 = LoginStateMachine.TAG;
                    StringBuilder sb = new StringBuilder();
                    sb.append("onResponse: ");
                    sb.append(str);
                    Log.d(access$4200, sb.toString());
                }
                LoginStateMachine.this.jbAuthentication.enter();
            }
        }, (JBErrorListener) new IgnoreErrorListener(LoginStateMachine.this.activity));
        jBStringRequest.setRetryPolicy(new DefaultRetryPolicy(60000, 0, 1.0f));
        VolleyRequestQueue.getInstance(LoginStateMachine.this.activity.getApplicationContext()).add(jBStringRequest);
    } catch (JSONException e) {
        if (VolleyRequestQueue.isDebugOn()) {
            Log.e(LoginStateMachine.TAG, "Error building JSON for authenticating to BD", e);
        }
        LoginStateMachine.this.finishLogin();
    }
}
```

Fig. 3. In the login flow a randomly generated transport key is first established between the Jyske Bank server and the client, which is encrypted with a public RSA key. Then AES encryption is used to transfer username and password over the connection. This ensures that even though an attacker gets hold of a man-in-the-middle connection he cannot read username and password.



```
<string name="postings_search_criteria_amount">Beløb</string>
<string name="postings_search_criteria_amount_all">Alle</string>
<string name="postings_search_criteria_category">Kategori</string>
<string name="postings_search_criteria_periode">Periode</string>
<string name="postings_search_period_default_value">Alle</string>
<string name="production_base_url">mobil.bankdata.dk</string>
<string name="production_jb_publickey">MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCoahwTZuZhJxmSDdOQeTIfNpohuipfQS4ttaypknG7lOX4Y+bQ
<string name="production_login_secret_key">8346833429834683342983468334298346833429444942128961810078952709</string>
<string name="production_nemid_certificate">-----BEGIN CERTIFICATE-----\10 MIIEVDCCAzygAwIBAgIJAOZ1nSI+Z/1TMA0GCSqGSIb3DQEBBQUA
<string name="production_nemid_rsalabel">jbprodver001</string>
<string name="project_id">dk-jyskebank-mobilebank</string>
<string name="rate_graph_header_asset_latest_rate_test">Seneste kurs</string>
<string name="rates_graph_button_day">1d</string>
<string name="rates_graph_button_five_years">5å</string>
```

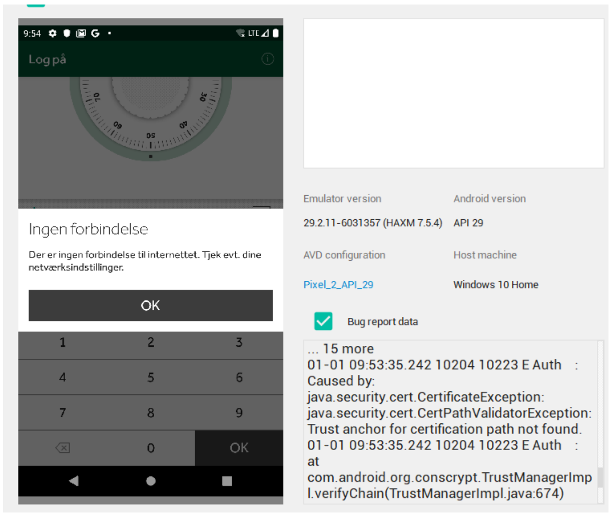
Fig. 4. The public RSA key and NemID certificate are hardcoded into the mobile application, this leaves the application unuable if the private keys becomes revoked, but leaves no options for an attacker to eavesdrop the communication.

Analytics and to crash reports. Microsoft's cookies are used for a specific site not related to the mobile application to balance the load on the server. Cookies seemed to mainly be used to track user behaviour to improve marketing. Nothing pointed towards the application sharing sensitive information like account balance.

V. DISCUSSION

The APK-file was not obfuscated, which could have made it more difficult to analyse the application code. This is not generally seen as a perfect security measure, but it could have served as another security layer. Thereby it is not seen as very severe.

It seemed a lot more severe that the application did not seem to make use of public key pinning. In the end, the end user is responsible for the network usage, and should thereby not use mobile applications or visit security sensitive web applications on an insecure network. That said, this can be difficult for an end user to actually see, being at hotels or coffee shops for instance. An attack abusing this is not very transparent to the end user. Developers already have a counter measure for man-in-the-middle attacks, by utilizing public key pinning. This pushes the responsibility towards the developers, as they can (relatively easy) provide a much safer experience to the end user. During the network traffic analyses, plain text banking information was observed. This is a clear violation of the confidentiality which the mobile application should have provided.



Der er ingen forbindelse til internettet. Tjek evt. dine netværksindstillinger.

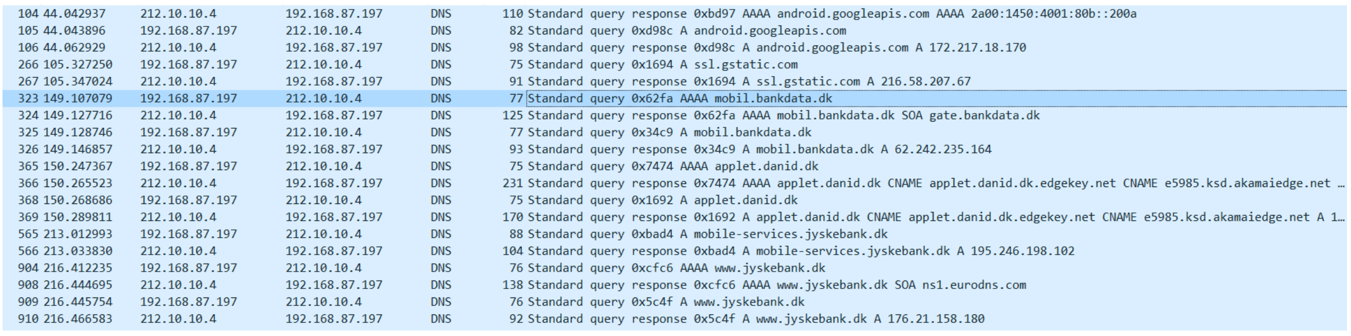
Emulator version: 29.2.11-6031357 (HAXM 7.5.4)
Android version: API 29
AVD configuration: Pixel_2_API_29
Host machine: Windows 10 Home

```
... 15 more
01-01 09:53:35.242 10204 10223 E Auth   :
Caused by:
java.security.cert.CertificateException:
java.security.cert.CertPathValidatorException:
Trust anchor for certification path not found.
01-01 09:53:35.242 10204 10223 E Auth   :
at
com.android.org.conscrypt.TrustManagerImp
l.verifyChain(TrustManagerImpl.java:674)
```

Fig. 5. Figure depicting the errors thrown by the application when the login fails due to a man-in-the-middle attack done on the phone running Android verison 10.0.

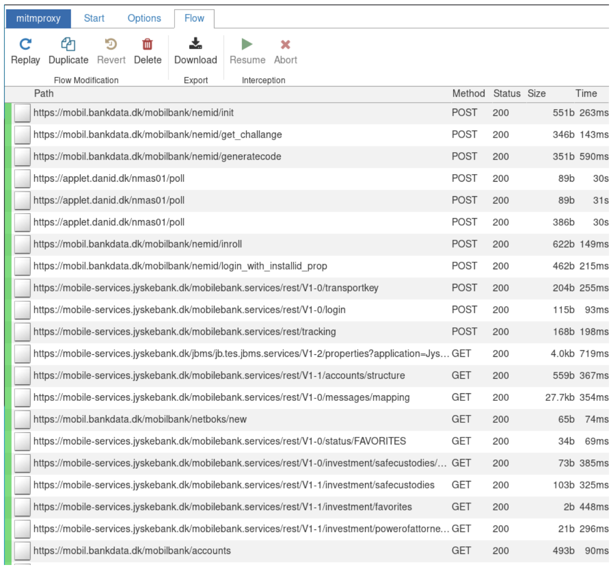
It is noted that an attacker should be in possession of a CA signed certificate to be able to carry out an attack like above mentioned, if attacking Android devices running Android OS version 7.0 or above. It is not entirely unrealistic that an attacker can achieve that, by using for instance free services like LetsEncrypt. It is also noted that operations like transferring money is still protected by two-factor authorization, double encrypted by both the server certificate and a hard coded public key within the mobile application. This assures the integrity of such operations and also somewhat the non-repudiation.

From the TLS scan the servers generally scored well, with the exception of what seemed to be the primary backend server. The reason for the one bad grade is a vulnerability to a single fairly new attack. This attack can violate the confidentiality of the users and should thereby be fixed as



```
104 44.042937   212.10.10.4     192.168.87.197  DNS   110 Standard query response 0xbd97 AAAA android.googleapis.com AAAA 2a00:1450:4001:80b::200a
105 44.043896   192.168.87.197  212.10.10.4     DNS    82 Standard query 0xd98c A android.googleapis.com
106 44.062929   212.10.10.4     192.168.87.197  DNS    98 Standard query response 0xd98c A android.googleapis.com A 172.217.18.170
266 105.327250  192.168.87.197  212.10.10.4     DNS    75 Standard query 0x1694 A ssl.gstatic.com
267 105.347024  212.10.10.4     192.168.87.197  DNS    91 Standard query response 0x1694 A ssl.gstatic.com A 216.58.207.67
323 149.107079  192.168.87.197  212.10.10.4     DNS    77 Standard query 0x62fa AAAA mobil.bankdata.dk
324 149.127716  212.10.10.4     192.168.87.197  DNS   125 Standard query response 0x62fa AAAA mobil.bankdata.dk SOA gate.bankdata.dk
325 149.128746  192.168.87.197  212.10.10.4     DNS    77 Standard query 0x34c9 A mobil.bankdata.dk
326 149.146857  212.10.10.4     192.168.87.197  DNS    93 Standard query response 0x34c9 A mobil.bankdata.dk A 62.242.235.164
365 150.247367  192.168.87.197  212.10.10.4     DNS    75 Standard query 0x7474 AAAA applet.danid.dk
366 150.265523  212.10.10.4     192.168.87.197  DNS   231 Standard query response 0x7474 AAAA applet.danid.dk CNAME applet.danid.dk.edgekey.net CNAME e5985.ksd.akamaiedge.net …
368 150.268686  192.168.87.197  212.10.10.4     DNS    75 Standard query 0x1692 A applet.danid.dk
369 150.289811  212.10.10.4     192.168.87.197  DNS   170 Standard query response 0x1692 A applet.danid.dk CNAME applet.danid.dk.edgekey.net CNAME e5985.ksd.akamaiedge.net A 1…
565 213.012993  192.168.87.197  212.10.10.4     DNS    88 Standard query 0xbad4 A mobile-services.jyskebank.dk
566 213.033830  212.10.10.4     192.168.87.197  DNS   104 Standard query response 0xbad4 A mobile-services.jyskebank.dk A 195.246.198.102
904 216.412235  192.168.87.197  212.10.10.4     DNS    76 Standard query 0xcfc6 AAAA www.jyskebank.dk
908 216.444695  212.10.10.4     192.168.87.197  DNS   138 Standard query response 0xcfc6 AAAA www.jyskebank.dk SOA ns1.eurodns.com
909 216.445754  192.168.87.197  212.10.10.4     DNS    76 Standard query 0x5c4f A www.jyskebank.dk
910 216.466583  212.10.10.4     192.168.87.197  DNS    92 Standard query response 0x5c4f A www.jyskebank.dk A 176.21.158.180
```

Fig. 6. Figure depicting a Wireshark session monitoring the network activity performed by the mobile application. The servers which the client communicates with can be seen from this. In the payload plaintext banking information was seen in more of the responses from the server



```
Path                                                                                      Method Status Size   Time
https://mobil.bankdata.dk/mobilbank/nemid/init                                            POST   200    551b   263ms
https://mobil.bankdata.dk/mobilbank/nemid/get_challange                                   POST   200    346b   143ms
https://mobil.bankdata.dk/mobilbank/nemid/generatecode                                    POST   200    351b   590ms
https://applet.danid.dk/nmas01/poll                                                       POST   200    89b    30s
https://applet.danid.dk/nmas01/poll                                                       POST   200    89b    31s
https://applet.danid.dk/nmas01/poll                                                       POST   200    386b   30s
https://mobil.bankdata.dk/mobilbank/nemid/inroll                                          POST   200    622b   149ms
https://mobil.bankdata.dk/mobilbank/nemid/login_with_installid_prop                       POST   200    462b   215ms
https://mobile-services.jyskebank.dk/mobilebank.services/rest/V1-0/transportkey           POST   200    204b   255ms
https://mobile-services.jyskebank.dk/mobilebank.services/rest/V1-0/login                  POST   200    115b   93ms
https://mobile-services.jyskebank.dk/mobilebank.services/rest/tracking                    POST   200    168b   198ms
https://mobile-services.jyskebank.dk/jbms/jb.tes.jbms.services/V1-2/properties?application=Jys… GET 200 4.0kb 719ms
https://mobile-services.jyskebank.dk/mobilebank.services/rest/V1-1/accounts/structure     GET    200    559b   367ms
https://mobile-services.jyskebank.dk/mobilebank.services/rest/V1-0/messages/mapping       GET    200    27.7kb 354ms
https://mobil.bankdata.dk/mobilbank/netboks/new                                           GET    200    65b    74ms
https://mobile-services.jyskebank.dk/mobilebank.services/rest/V1-0/status/FAVORITES       GET    200    34b    69ms
https://mobile-services.jyskebank.dk/mobilebank.services/rest/V1-0/investment/safecustodies/… GET 200   73b    385ms
https://mobile-services.jyskebank.dk/mobilebank.services/rest/V1-1/investment/safecustodies GET  200    103b   325ms
https://mobile-services.jyskebank.dk/mobilebank.services/rest/V1-1/investment/favorites   GET    200    2b     448ms
https://mobile-services.jyskebank.dk/mobilebank.services/rest/V1-1/investment/powerofattorne… GET 200   21b    296ms
https://mobil.bankdata.dk/mobilbank/accounts                                              GET    200    493b   90ms
```

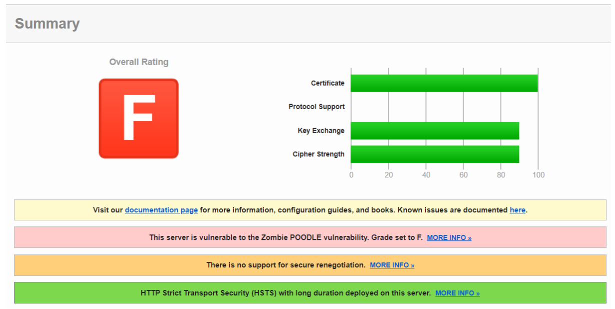
Fig. 7. In this figure it is seen that two login flows are established. One for the third-party bank server and one for the Jyske Bank server. The first login flow uses NemID, which requires a two-factor authentication process. The second flow only requires password and username.

Server	Grade	Reason
https://mobil.bankdata.dk	A+	N/A
https://mobile-services.jyskebank.dk	F	Explotable to Zombie POODLE
https://jyskebank.dk	A+	N/A
https://app-mobile-services-jyskebank-dk-t.jyskebank.dk	A-	Disabled secure renegotiation
https://dk-jyskebank-mobilebank.firebaseio.com	A+	N/A

Table 1. Table depicting the optimized results of selected classifiers in combination with eachother. The accuracy is calculated on the validation data.

quickly as possible.

Looking at the cookies which the bank claims to be using many of them seems to be marketing related third-party cookies. This can be problematic and especially if sensitive banking information is shared. From the network analysis



Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

This server is vulnerable to the Zombie POODLE vulnerability. Grade set to F. MORE INFO »

There is no support for secure renegotiation. MORE INFO »

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. MORE INFO »

Fig. 8. The summary of the result after doing a TLS scan of the server: https://mobile-services.jyskebank.dk. The server gets an F grade because it is exploitable to the attack called Zombie POODLE, which potentially makes TLS packages readable.

Company	Reason
Agillic A/S	Marketing
Facebook Inc.	Marketing
Gemalto N.V.	Flow for design of personal credit card
Google Inc.	Analytics and crash reports
LeadFamly Aps	Marketing
Microsoft Inc.	Load balancing
Nets DanID A/S	Login flow
Sleeknote Aps	Marketing
TwentyThree Aps	Marketing

Table 2. Table depicting third-party cookies which Jyske Bank uses. Some of them through the mobile application.

and the analysis of the APK code, problematic usage does not seem to be in place.

VI. CONCLUSION

During this project a thorough analysis of the mobile banking application "Jyske Mobilbank" was done. The general threat model was analysed and the application was assessed as relatively security sensitive, thereby demanding a secure development of the application. Some violation of the general confidentiality requirements was found. It was especially severe that no public key pinning was being used, which can potentially expose sensitive customer information. Furthermore, a TLS analysis of the backend servers exposed a potential vulnerability in one of them. No banking operation seemed to be vulnerable to attacks during the analysis. This pointed towards relatively secure integrity mechanisms. Many third-party cookies seem to be used in Jyske Bank's banking infrastructure. No notable use was discovered in the mobile application and during network analysis.

REFERENCES

[1] Jyske Bank A/S. Jyske bank - boligejernes bank. `https://www.jyskebank.dk/`.

[2] Inc. Qualys. Qualys ssl labs. `https://www.ssllabs.com/`.

[3] Open source. skylot/jadx: Dex to java decompiler. `https://github.com/skylot/jadx`.

[4] The Wireshark Foundation. Wireshark · go deep. `https://www.wireshark.org/`.

[5] Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. mitmproxy: A free and open source interactive HTTPS proxy, 2010–. [Version 5.0].

[6] OpenSSL Software Foundation. Openssl - cryptography and ssl/tls toolkit. `https://www.openssl.org/`.

[7] Internet Security Research Group (ISRG). Let's encrypt - free ssl/tls certificates. `https://letsencrypt.org/`.

[8] Craig Young. Zombie poodle, goldendoodle, & how tlsv1.3 can save us all. `https://i.blackhat.com/asia-19/Fri-March-29/bh-asia-Young-Zombie-Poodle-Goldendoodle-and-How-TLSv13-Can-Save-Us-All.pdf`.