Network Security
Final Project

Abhijay Hazarika
Dept. of Electrical and Computer Engineering
Aarhus University
Aarhus, Denmark
202101677@post.au.dk

Axel Søgaard Poulsen
Dept. of Electrical and Computer Engineering
Aarhus University
Aarhus, Denmark
201609062@post.au.dk

***Abstract*—In this report a preliminary security analysis of the mobile application "Oddset" was done. The threat model was analysed to provide a comprehensive analysis of the attack surfaces, and several security flaws in the implementation were found. Security, privacy and verification of authentic user-credentials were deemed as the important security properties for this mobile application. Security of the application and the user details were found to be violated by utilising man-in-the-middle attack on the application using tools such as mitmproxy. The certificate pinning was also found to be absent when analysed through the MobSF security platform. Although there was presence of two factor authentication when logging in via NemID, it was absent when the user tried to log in via normal method. Several third-party trackers were also found running in the background which may compromise the privacy of the application.**

I. Introduction

Oddset is a Sports Betting Application developed by Danske Spil A/S. This application has over 10.000 downloads on Google Play store. This application is a very popular local application and is used to bet on matches in different sports such as Tennis, Hockey, Football etc. Due to the handling of sensitive application information from the users such as banking details and NemID, this mobile application had been chosen for preliminary security analysis.

II. Threat Models for Application

Mobile applications such as Oddset are increasing popular nowadays because of the ability to integrate different API's to give review on the performance and to provide an analytical report for the overall business model such as user behaviour. Unfortunately the combination of these technologies present opportunities to bad actors a chance to exploit personal and sensitive data. API's expose application logic and sensitive data such a Personally Identifiable Information which makes it a point of target to bad attackers. If the application logic can be extracted then it gives an opportunity to hackers to launch sophisticated automated attacks on the API's.

Mobile applications are usually built under the assumption that the user is legitimate and doesn't hold any malicious intentions while he has access to the services. Another assumption is that the user is using an unaltered version of the mobile application, running on a device which is not rooted. It is also an assumption that communication with the API server is via a secure channel.

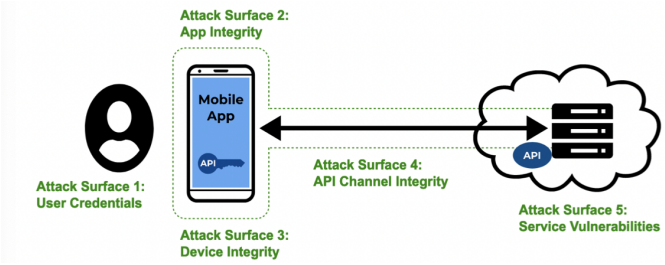


Fig. 1. Threat Surfaces

The attack surface can be defined as the number of all possible points, or attack vectors, where an unauthorized user can try to access a system and extract data. There are five potential attack surfaces which could be exploited for this application.

A. User Credentials

The authentication and authorization of the user is critical, particularly since personal data such as NemID is being accessed via the app. However valid credentials can be stolen via spoofing, phishing and have been exposed via large data breaches: username/password combinations are bought and sold on the dark web and used in credential stuffing attacks. It's not always that the adversary needs to steal the credentials to use it maliciously. Since there have been instances where users type their own credentials to a fake applications there is a possibility for users to mistakenly enter their credentials to a fake application. Finally the utilisation of the Onboarding process is often seen where anyone (including the hacker) can easily just sign up for the service.

It is much easier to use stolen user credentials if other attack surfaces have vulnerabilities. For example, if somebody's credentials are compromised then nothing can stop a bad actor from typing them into a valid instance of the app. Similarly if the user authorization token can been extracted from the mobile app by utilising Man-in-the-middle attack then the application response can be accessed from a script.

B. Application Integrity

When we are targeting the mobile application the objective is to accomplish one or both of the following tasks:

- Extract information which can be used to mount an automated attack on an API using another tool such a 'mitmproxy'. As well as gaining useful identifiers and keys this could involve inspecting the logic of the mobile app in order to reverse engineer how the API works with the aim of abusing the business logic through the API.
- Transform the app itself into a tool which can be used in an automated attack or tweak it in some other way, for example to divert payment or advertising revenue or to hijack user information for nefarious purposes.

Our goal is to understand the structure of the API calls, and investigate if any mechanism is being employed to validate the app in order to replicate it with any required secrets. The mobile app may use API keys, device ids, or a user authorization token to communicate what is making the request to the API backend, but often these identifiers are hard coded in the mobile app source code, thus they can be extracted with the use of reverse engineering techniques, for example sniffing the incoming response with Man-in-the-middle attacks. Statically reverse engineering a mobile app is one of the most common steps taken when attacking a mobile app, and it's listed in 9th position of the most recent OWASP Mobile Top 10 risks. If more sophisticated methods such as calculating the API key at runtime are used to hide these identifiers/secrets then dynamic code instrumentation at runtime using MobSF or a MitM attack can be used to extract them. Code tampering comes in the 8th position of the most recent OWASP Mobile Top 10 risks. This is done by repackaging the mobile app with removed or altered code, and if done correctly the API will see the tampered mobile app behave as the original one did. For example, if the API backend is checking the header with the mobile app binary signature hash, then the repackaged app will have code in place to deliver the same value as the original app.

C. Device Integrity

The device on which the application is running can be rooted, and this might be done for legitimate reasons like some users prefer to run customised or more recent versions of the OS and some users prefer to side-load genuine apps which may not be available in local app stores. These actions do not, in and of themselves, indicate malicious activities are going on. That said, rooting/jailbreaking is a common technique used by attackers (and pentesters such as us) to bypass security mechanisms and limitations imposed by the original version of the OS.

Rooted and jailbroken devices pose a threat to device integrity, because these actions enable the in-built security mechanisms to be compromised. By extension the threat extends to the mobile app integrity, because the mobile app is now running in an environment that cannot be trusted. Another form of code tampering is to inject code at runtime by using an instrumentation framework. Such frameworks are used to hook into the key functions which, when manipulated, will produce different app behavior than expected or will change input parameters or output results. In this way fraudsters can intercept and modify genuine user instructions.

Fig. 2. Threat Surfaces

D. API Channel Integrity

The communication channel between the application and the API can be exposed when the device is connected to the internet via public Wifi rather than using secure VPN services. Even when the latest versions of TLS standards are being used, the queries/response can be accessed by the attacker by utilising tools like 'mitmproxy' which enables the attacker to perform man-in-the-middle attack. The basic principal behind man-in-the-middle attack is to convince the client that there is a secure connection between the client and the server and there is no third-party listening in on the conversation. This is the route where usually the attacker can listen in on the traffic between the server the client. Certificate pinning actually makes the man-in-the-middle attack difficult to perform but since the mobile application can be repackaged to remove the pinning detection controls or to inject pins or certificates from the 'mitmproxy' tools it is not that hard to perform.

E. API and Service Vulnerabilities

The API and Service vulnerabilities can be exploited by bad agents using automated tools and these are three common ways to exploit the vulnerability of the system:

- Login System Attack: The user credentials can often be stolen from the dark web and can be used by the hackers to access the API services. One common way to do it is to use automated brute force attacks which would enable hackers to cross-validate the user credentials with the APIs.
- Theft of Data: Files can stolen such as photos, credit-card information and personal data from the accounts available through the API. One way of gathering the information is to perform data scrapping which would enable the attacker to gather meta data which can be used to access sensitive information.
- Denial of Service: This is one of the most common form on attack on an API which enables the attacker to slow down traffic or stop the API response on the mobile application. This would result in the user not being able to access the services provided by the API.

If a hacker is able to extract the required information from the above Attack Surfaces about the validation process and has acquired the necessary keys, then the API can be abused by an automated attack, and any vulnerabilities in the API can be exploited.

III. Software Security Analysis

Software Security Analysis is the investigation of the software for cases of malicious programming with the intent to cause fraud, breach of security. In this investigation we performed the decomplication of the APK file of the mobile application which enabled us to detect that there is no presence of obfuscation of the decompiled source code. Since the code is human readable, we were able to perform manual analysis of the source code.

Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pentesting, malware analysis and security assessment framework capable of performing static and dynamic analysis. This application has been setup on a Linux Operating System by installing the required dependencies and the output of the static analysis has been presented in a PDF format.

A. Static Analysis

Static Analysis of an Android application involves examining an application component without exploiting them by analysing the source code manually or automatically. For the purpose of decompilation MobSF has been utilised which allowed us to obtain a comprehensive report on the static analysis [**?**]. During the static analysis, the mobile application's source code was reviewed to ensure appropriate implementation of security controls. During the process of manual code review the decompiled Java source code had been scanned to seek basic keywords and calls such as 'executeQuery' and 'executeStatement' and Crypto algorithms that were implemented. MobSF also provides a list of various different API's that run in the background.

It can be seen from the static report that the application is signed with a code signing certificate which is secure and unique way of signing it. But it can be also seen that the Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android 5.0-8.0, if the certificate is signed with only v1 signature scheme.

Janus vulnerability comes from the possibility to add extra bytes to the APK and DEX files. JAR signature scheme takes into account the zip entries and ignores any extra bytes in computing or verifying application's signature. A DEX file can contain arbitrary bytes at the end, after the regular sections of strings, classes methods etc. Therefore this file can be be a valid APK and a DEX file at the same time. This give the attacker an opportunity to prepend a malicious DEX file to the APK without affecting its signature.

The report on Network Security shows that the dantotomobil.danskespil.dk domain config is insecurely configured to permit clear text traffic to this domain. This enabled us to sniff the traffic the coming in and out of the application. It can be also seen that the base config is set to trust system certificates which enabled us to install mitm certificate as a trusted system certificate and perform the mitm attack.

NETWORK SECURITY

NO	SCOPE	SEVERITY	DESCRIPTION
1	*	good	Base config is configured to disallow clear text traffic to all domains.
2	*	warning	Base config is configured to trust system certificates.
3	dantotomobil.danskespil.dk	high	Domain config is insecurely configured to permit clear text traffic to these domains in scope.
4	ds.production.loyalty-assets.s3.amazonaws.com	high	Domain config is insecurely configured to permit clear text traffic to these domains in scope.

Fig. 3. Static analysis report on Network Security

5	The App uses ECB mode in Cryptographic encryption algorithm. ECB mode is known to be weak as it results in the same ciphertext for identical blocks of plaintext.	high	CVSS V2: 5.9 (medium) CWE: CWE-327 Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-2	dk/shape/cryptokid/EncryptedStorage.java

Fig. 4. The usage of ECB mode of encryption from the static report

The static report on the Application permissions is that the application require the permission to write and read to external storage and install packages which can lead to installation of malicious software on the android device. The attacker can install malicious software which will allow him/her the permission to access sensitive information on the external storage. It also has access to the user location because the application permission has access users fine and coarse location, which can be used to pinpoint the users location.

It can viewed from the static analysis report that the application uses ECB mode for cipher. This can be viewed because we can view the source code because of the absence of obfuscation. ECB is a simple mode of operation with a block cipher that's mostly used with symmetric key encryption. In ECB each plaintext is in a block and can be defined by the corresponding plaintext but needs to have identical keys. Hence, the identical plaintext with the same keys will always encrypt the same ciphertext.

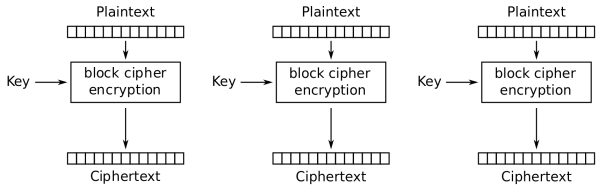
But ECB mode of encryption has the following drawbacks:

- Two identical blocks of plaintext will result in the same ciphertext hence its vulnerability is exposed to replay attacks.
- ECB is not reliable with small block sizes because the ciphertext have similar pattern from the plaintexts.

In figure 6 an example is given of a picture which is encrypted by using AES in ECB mode. It is clear that the encrypted image does show patterns that reveals information about the original image. Interestingly enough as we will see later, in subsection *Carrying out the attack*, even though a symmetric encryption algorithm is present in the code, it is not used as a measure against man in the middle attacks.

B. Dynamic Analysis

The focus of Dynamic Analysis is the testing and evaluation of apps via their real-time execution. The main objective of dynamic analysis is finding security vulnerabilities or weak



Electronic Codebook (ECB) mode encryption

Fig. 5. ECB mode of Encryption

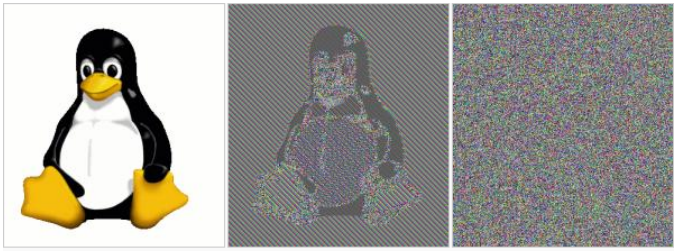


Fig. 6. Example of ECB mode of Encryption

spots in a program while it is running. Dynamic analysis is conducted both at the mobile platform layer and against the back end services and API's, where the mobile app's request and response patterns can be analyzed. Dynamic analysis is usually used to check for security mechanisms that provide sufficient protection against the most prevalent types of attack, such as disclosure of data in transit, authentication and authorization issues, and server configuration errors. [**?**]

The setup for the Dynamic Analysis had been done by setting up genymotion which is an android virtual machine which runs in the background while preforming dynamic analysis of the application. We able to dynamically analyse different test cases that we thought would be necessary to analyse the security of the application.

The Dynamic Analysis of the application has been performed by running it on MobSF and we can view that the Frida live logs and it comfirms our static analysis that the certificate is not pinned.

The live API monitor has also been tested which helped us to identify the live API requests that have been made during the running of the Application.

The MobSF framework also helped us to run TLS/SSL testing which helped us to evaluate the security of our application and network connections.

- TLS Misconfiguration Test: This test uncovers insecure configurations that allow HTTPS connections that bypass certificate errors.
- TLS Pinning Transparency test: This test evalutes the applications user certificate pinning.
- TLS Pinning Transparency bypass test: This test tries to by pass certificate pinning of there is certificate pinning.

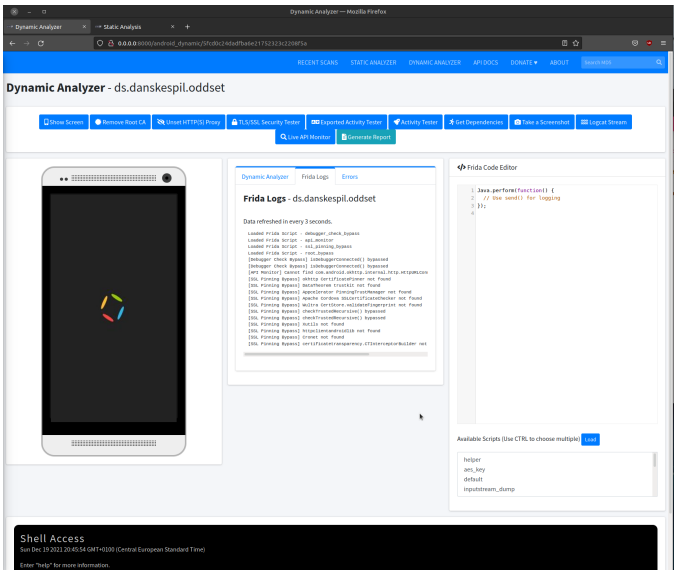


Fig. 7. Testing Certificate Pinning

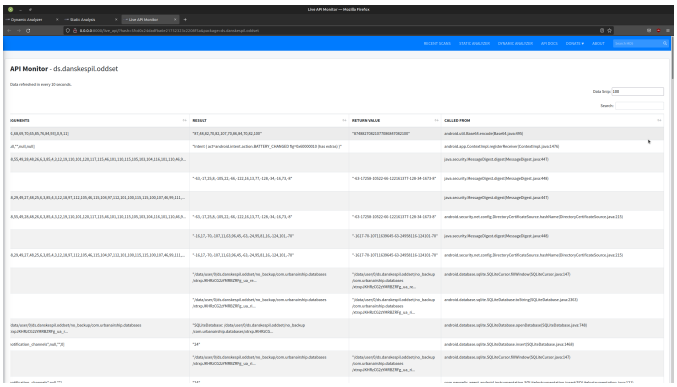


Fig. 8. Live API Monitoring

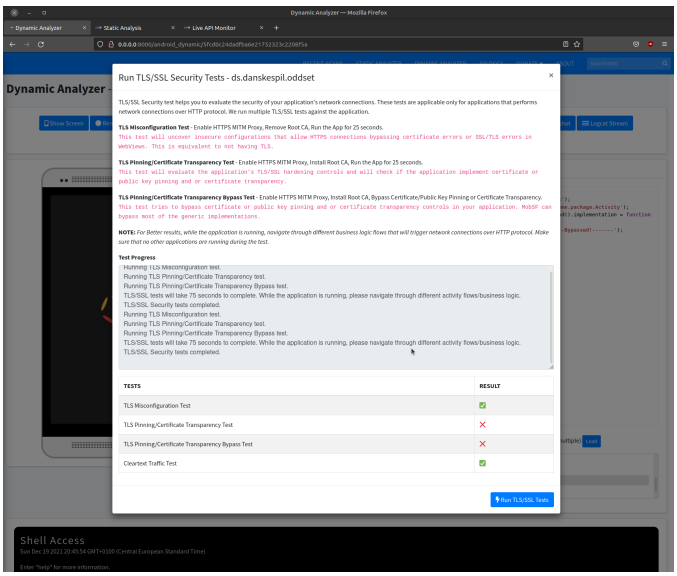


Fig. 9. SSL/TLS testing

IV. Network Security Analysis

Network security analysis is an audit designed to find the security vulnerabilities that are at risk of being exploited which could cause harm to the business model and leak private information of the users while they are logged in to the network. The purpose of this analysis to find out if there were any presence of gaping holes in the network which would enable us to access the sensitive information that passes via the communication channel of the client and the server.

A. Man in the middle attack

1) The setup: A man in the middle attack (called mitm from now on) is an attack the may be performed in the case where the user wants to connect to some kind of service.

In short, the goal of the adversary is then to place himself in the middle of the connection (hence the name) between the user and the server looking at the packets send between these two end points.

The objective of the adversary could be that the adversary would like to get hold of some kind of secret information, i.e. banking credentials.

As an example of a mitm attack, let's look at the case where a user want to connect to an online banking account. Normally that would proceed as follows:

- The user opens the browser and types in the url of the banking service.
- The browser looks up the ip address and the proceeds to set up a secure connection with the website using a ssl/tls handshake.
- The user communicates with the service until the connection is closed again

Whats important to note here is that the server probably only will accept a https (using tls/ssl) connection in order to avoid leaking the information in the packets send back and forth through the use of ecryption.

Thus the adversary will not get any sensitive information by, let's say, listening to the wifi packet being send.

This is where the *Man in the middle attack* comes in.

2) The attack: Since the information in the packets are protected by encryption, the adversary needs some other way of listening to the traffic.

In short in the mitm attack, the adversary tries to trick the user into sending all the traffic to him, which he then passes on to the real service wanted by the server. While passing the packets, he then looks at the packets when they arrive and since he is part of the connection, he'll be able to decrypt the payload.

Setting up the attack however is not that easy since he needs to trick the user into sending the traffic to him (which is then passed on). This involves setting up a proxy, installing a certificate on the user's phone (or get hold of a certificate signed by a trusted authority) and then setting the user's phone to direct the traffic at the proxy. The procedure can be described in two phases, the initial phase where the attack is set up and the attacking phase where the attack is carried out. The initial phase looks like this:

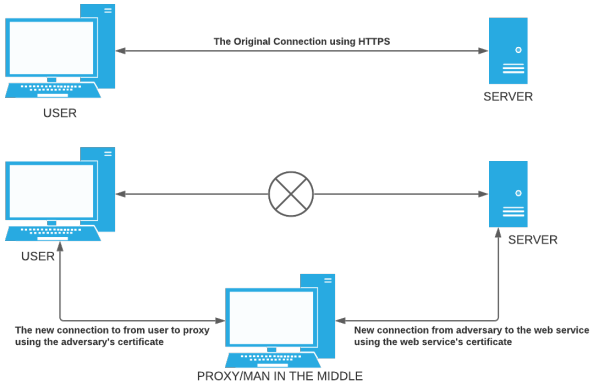


Fig. 10. Man in the middle attack setup

- The adversary sets up a proxy
- The adversary then tricks the user to install the right certificate for the proxy (to allow the encrypted traffic)
- The adversary then tricks the user into setting up the proxy on the phone

Now when the user wants to use the online service, i.e. online banking account, the attack phase proceeds as follows:

- The user opens up a web browser and types in the url of the webservice
- The browser opens a secure connection to the proxy from where a secure connection to the web service is opened.
- The user then proceeds to send packets to the server via the proxy
- While the user is communicating with the service, all information about the packets (including the payload) is recorded and possibly modified at the proxy.
- When the user is done, the connection is closed

As mentioned in the list above, the attacker might also try to alter the contents in the packet. There could be a case where the user is connected to an online banking account where he requests a money transfer. The adversary could then potentially change the information of the destination account thus sending the money to a different place.

The mitm setup can be described as in figure 10.

B. Preventing a Man in the middle attack

Even if the adversary manages to trick the user into setting up the certificate and proxy on the phone, all hope is not lost when it comes to confidentiality, integrity, authentication etc.

1) Encryption: The user and the web service might still achieve a connection that ensures the security properties (except for availability). By using e.g. a asymmetric encryption scheme with preshared keys, the payload could still be protected such that the adversary won't be able to look at the content or modify it. Of course the keys then have to be shared in advance in a secure matter such that the adversary won't be able to get hold of it (thus breaking the solution).

2) Certificate pinning: Another solution that avoids the key sharing problem all together is the use of *Certificate pinning*.

In this solution, the user is already giving the certificate that the server is going to use during the session key establishment. This serves two purposes:

Firstly it prevents the adversary at the proxy of having two separate connections (one with the server and one with the user). If the user doesn't receive the right certificate from the server, he can outright reject it (since the session keys thus would be established using a public key that is not known to be the right one). Secondly, since the adversary does not have access to the private key of the server, he won't be able to get hold of the session keys and thus won't be able to decrypt the packages sent between the user and the server.

Of course there could still be the case where the adversary could compromise availability of the web service by simply blocking the traffic. This however won't leak any confidentialities.

Lastly it needs to be mentioned that certificate pinning is not just a magic mechanism that flawlessly guarantees no mitm attacks. There might be cases where the attacker gets hold of the source code of the app (through decompiling) and is able to change the pinned certificate and then repackage the app. Then the adversary might trick the user into installing this non official version of the app which has the adversary's certificate pinned thus allowing the mitm attack to proceed.

Another practical problem of certificate pinning is that if the certificate expires, then a new one has to be generated and since the certificate is pinned in the code the whole binary needs to be recompiled and published again making the solution a bit impractical.

C. Our man in the middle attack setup

As we tested out the Oddset app from Danske Spil A/S we had to set it up on a device, for this we used an emulator provided by the AVD tool in Android Studio.

We then set up a proxy by using the mitmproxy tool and installed the certificate of this proxy in the emulated device (along with the proxy setup). This part was a bit tricky since in all Android versions above version 7, apps ignores user provided certificates (unless they are programmed to use them). We thus had to go through a few steps to set the certificate as part of the provided system certificates. These steps included:

- Generating a hash of the certificate and renaming the file to it
- Modifying the android system image such that it contains our certificate in the system certificates folder (adb root was used here)
- starting the android emulator from the terminal with our modified image.

Then the emulator was configured to acces the proxy through localhost port 8080 (the default) and thus our setup was ready for experiments. The whole setup is summarized in figure 11.

Fig. 11. Setup of mitmproxy

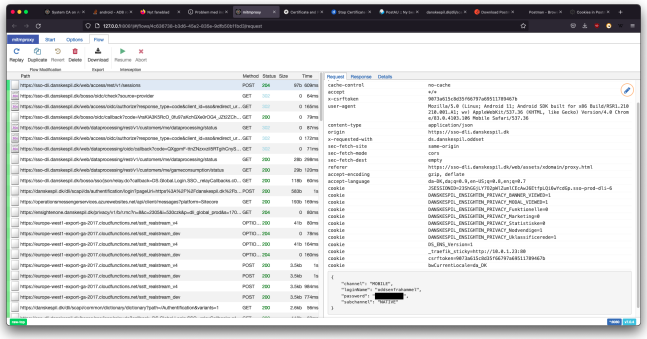


Fig. 12. Packet containing login credentials in plain text

D. Carrying out the attack

Immediately after setting up our device and proxy, we received information about packets coming to and from the device.

Opening the oddset app we imediately got information about packets as well and we were allowed to log in into the app.

This wouldn't have worked if certificate pinning was used, since in this case it should have refused the connection (because of mismatch of certificates).

Especially interesting, the oddset app offers two login options: NemID and blaa konto, where the latter is a oddset account linked to the NemID. We were testing out the app using an existing account (belonging to one in the group) and thus it was possible just logging on with the blaa konto straight away.

The interesting part here is that, after writing the login credentials and pressing the "login" button, we witnessed the login request packet in mitmproxy and to our surprise, the login credentials were sent in plaintext (of course through https), making it possible for an attacker to get access to the account. This packet can be seen in figure 12.

As the Oddset app contains an online wallet that stores the money used to place bets and the money won, we proceeded to the "Indbetal/Udbetal" page to see what happens when card

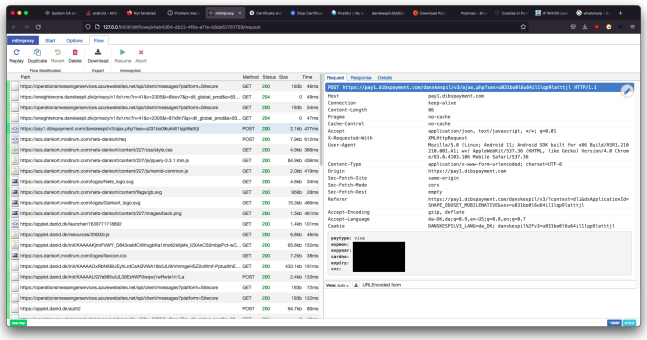


Fig. 13. Packet containing credit card credentials in plain text

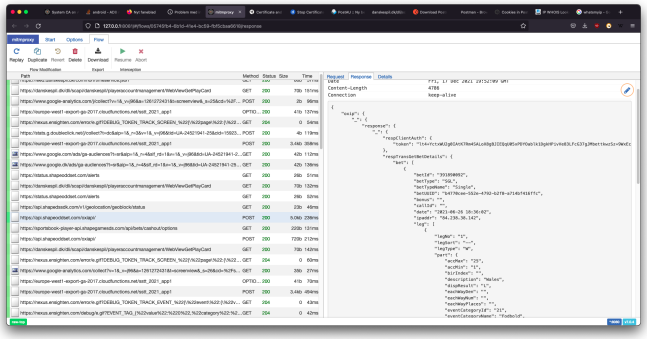


Fig. 14. Packet containing bet history information in plain text

credentials are given. We here tried to insert 70 kr (it had to be bewteen 50 and 70 kr) and we entered VisaDankort for a credit card belonging to one of the group members. Pressing the ”insert” button, we received a packet in mitmproxy.

Again interestingly enough, in this packet we were also able to see a json structure containing all the credit card information reaable in plaintext. This is a huge a risk for the user since this directly gives all the necessary information for the adversary to make transaction using the users credit card. The details of this packet can be seen in figure 13.

After this we went to the activity showing all the bets placed with this account. Going here we received a packet containing a JSON structure containing all this info as well, making all the details about each previous bets visible to the adversary. Really interesting we here found that the json structure containing the information about each bets actually contained more details than what was shown to the user.

An interesting detail in the packet was the inclusion of the IP address from which that particular bet was made. By making a whois request on these addresses, it was possible to make a rough estimate on where in the world the user was located when placing these bets. This could allow the adversary to track the location of the user back in time, which is clearly breaking privacy. Of course this is a rough estimate and the user could avoid this by using a proxy or a VPN or even just mobile data. The packet containing this information can be seen in figure 14

Looking at all these different cases of sensitive data being presented in clear text and considering the lack of certificate pinning, it is clear that the developers of this mobile application did not take into account the possibility of mitm attacks. Of course all the traffic is encrypted by https and thus some encryption are present – preventing attackers from seeing all the data just by sniffing the WiFi traffic.

One developer could try to argue that additional measures to protect against these types of attacks might not be necessary, but we would argue that a few extra measures could fix this security risk.

The most simple addition would be to encrypt all the sensitive data by using a symmetric encryption scheme and some kind of key sharing protocol. Thus, even though the adversary would have access to the connection, he wouldn't be able to see the sensitive information itself.

Of course this wouldn't prevent the adversary to compromise availability, but in this case such a prevention is not necessarily critical – the user would just not be able to spend, or retrieve any money from the account from this particular device. Even this availability risk could be solved by using certificate pinning.

V. Authentication

Authentication can be defined as a set of mechanisms to allow entities to establish identities and verify if they are communicating with other legitimate entities. There are generally three types of authentication mechanisms; one way is something that the user ”knows” like passwords; the second way of authentication is something that the user ”possesses” like cryptographic token or key; the third way of authentication is that something that the user ”does” like bio metric scans.

The user can log in the application by using two log in mechanism. One way to log in to the account is via NemID which uses two factor authentication method. NemID is a secure and standard method to log in to any account because it ensures that the user must have his application or the card ready to type in the key. The other way to log in is to create a standard user account using email and password.

The second mode of authentication is not secure because we are able to view the password and the username that we send to the server for validation by the mitmproxy. The data sent to the server by the application is not encrypted and we can view the data in plaintext format. This is a huge security flaw and can be exploited by hackers trying to sniff the traffic from an insecure router. The general workflow of the authentication method is not upto standard because it responds if the password is incorrect but does not provide a feedback via an email verification that it is not the authentic user trying to access the information.

Authentication of the user to log in to the application is vital because this application utilise the banking details and personal ID so that the user can place the bets and claim the rewards. If the user credentials are stolen then the attacker can place the bets on a some team without the prior knowledge of the user and cause the user to loose the money. This

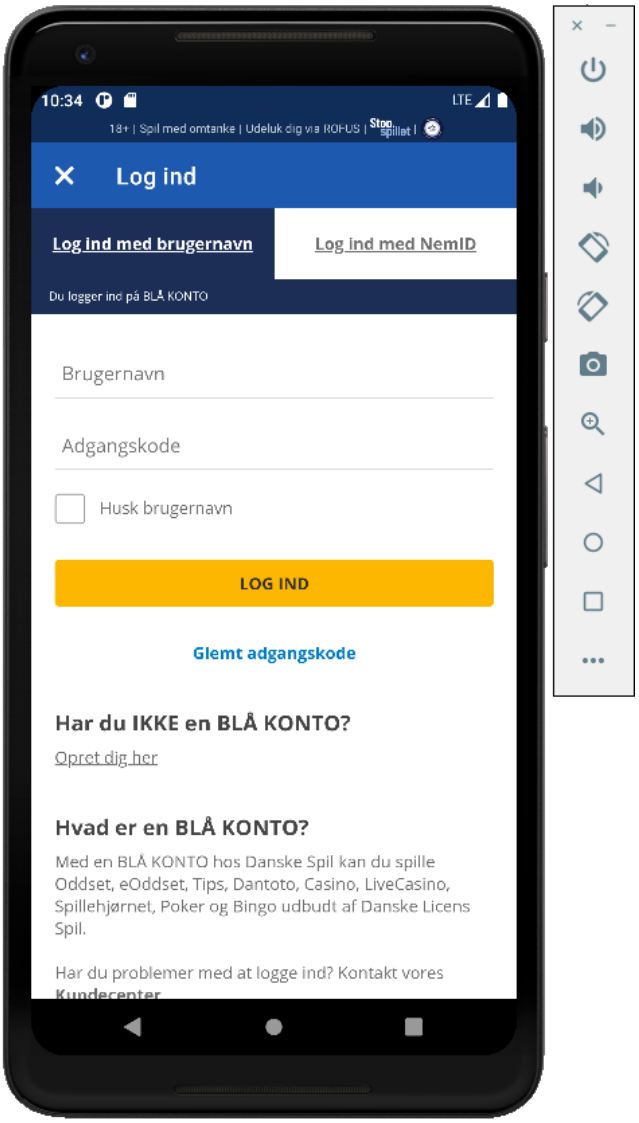


Fig. 15. Login_Screen for the application

is the reason why the application should have a two factor authentication to confirm that the bet has been placed.

For best practices the passwords should be stored with a standard cryptographic hash algorithm which cannot be reversed. The standard practice should to create the the create the hash values using Argon2id or Scrypt. The hash values should be slated with a value which is unique to that specific login credential. Hashing technologies such as MD5 or SHA1 should also be avoided as they are not secure and there is also advantege to iteratively re-hash the password multiple times. The third party verification technique by utilising NemID is also a correct way to properly authenticate the user credentials.

VI. Privacy

Privacy is one of the most important things to focus on in the 21st century when designing an application. The users should know what are the exact privacy policies and the permissions that their application requires to access. This application does provide a standard terms and conditions agreement for the users to sign on while creating an account. The nature of data that is being collected by this application is meta data in plain text format. The application uses personal data such as personal banking information, NemID, location etc. While the nature of the data collected by this application is of sensitive in nature but it is understandable since its a betting app the banking credentials are required to place the bets and claim the rewards.

android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete external storage contents	Allows an application to write to external storage.
android.permission.REQUEST_INSTALL_PACKAGES	dangerous	Allows an application to request installing packages.	Malicious applications can use this to try and trick users into installing additional malicious packages.
android.permission.ACCESS_FINE_LOCATION	dangerous	fine (GPS) location	Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.
android.permission.CAMERA	dangerous	take pictures and videos	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.

Fig. 16. Caption

The static analysis report shows the utilisation of the following crucial android permissions:

- WRITE_EXTERNAL_STORAGE: It allows application to read/modify/delete the contents of external storage.
- REQUEST_INSTALL_PACKAGES: It allows the application to request installing packages. Malicious applications can use this trick to install additional packages without the consent of the user.
- ACCESS_FINE_LOCATION: It allows access to fine location sources such as GPS on the phone where it is available. This also consumes additional battery power from the device.
- CAMERA: This allows the application to take pictures from the camera without the consent of the user.
- READ_EXTERNAL_STORAGE: This allows the application to read the contents of the external storage of the device.
- ACCESS_COARSE_LOCATION: It allows the application to access coarse locations such as mobile network database to determine an approximate phone locations when available.

The presence of 13 trackers have been detected from the static analysis report. The presence of these trackers can be considered as an invasion of privacy because the presence of Facebook and Google analytic can track the behaviour of the user and see when and how they place a bet. This can be used without the knowledge of the user because most users do not read the terms and conditions report where it is mentioned.

From the android manifest it can also bee seen that there some very high issues regarding the intrusiveness of this application.

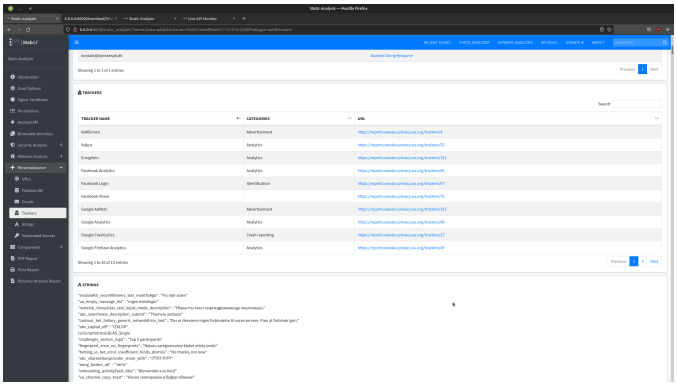


Fig. 17. The presence of trackers in the static report

			specific domains and for a specific app.
2	Activity (dk.shape.oddset.main.HierarchyNavigationActivity) is not Protected. An intent-filter exists.	high	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
3	Broadcast Receiver (dk.shape.oddset.gcm.OddsetNotificationPublisher) is not Protected. [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
4	Broadcast Receiver (com.adjust.sdk.AdjustReferrerReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.INSTALL_PACKAGES [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.
5	Broadcast Receiver (com.google.ads.conversiontracking.InstallReceiver) is not Protected. [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
6	Broadcast Receiver (com.phrase.android.sdk.LocaleChangedReceiver) is not Protected. [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.

Fig. 18. Static Analysis of Manifest Report 1

VII. Conclusion

The preliminary security analysis of the android application shows that the application is not secure. It has a very poor score on the MobSF static analysis of 25/100 which shows that the application is a high-risk for users since the data that's being sent can be seen as plain-text when sniffed through the mitmproxy tool. But it can be made more secure by implementing frameworks such as OpenID which is an open standard and decentralized authentication protocol to verify user credentials. We would also recommend the use of open source tools which would help in obfuscation of the source code. Although obfuscation will not solve the decompilation process but it will make it time consuming and expensive for the attacker to follow the code since the variables, functions and classes will be redacted. The mobile application should also employ a run-time self defences system to detect rooted devices which would prevent the attacker to access sensitive information written to the external or internal storage. Thus it can be concluded that there a lot of fundamental security flaws in the application has been detected and analysed.

7	Activity (com.facebook.CustomTabActivity) is not Protected. [android:exported=true]	high	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
8	Activity (androidx.biometric.DeviceCredentialHandlerActivity) is not Protected. [android:exported=true]	high	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
9	Broadcast Receiver (com.urbanairship.accengage.PackageUpdatedReceiver) is not Protected. An intent-filter exists.	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
10	TaskAffinity is set for Activity (com.urbanairship.push.NotificationProxyActivity)	high	If taskAffinity is set, then other application could read the Intents sent to Activities belonging to another task. Always use the default setting keeping the affinity as the package name in order to prevent sensitive information inside sent or received Intents from being read by another application.
11	Broadcast Receiver (com.google.firebase.iid.FirebaseInstanceIdReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: com.google.android.c2dm.permission.SEND [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

Fig. 19. Static Analysis of Manifest Report 2

References

[1] *The Threats to Mobile Apps and APIs*. [Online]. Available: https://www.approov.io/download/Approov-Threats-to-Mobile-Apps-and-APIs.pdf

[2] *Dynamic Analysis Using MobSF*. [Online]. Available: https://null-android-pentesting.netlify.app/src/dynamic-analysis/using-mobsf.html

[3] R. Awati, "Electronic Code Book (ECB)." [Online]. Available: https://www.techtarget.com/searchsecurity/definition/Electronic-Code-Book

[4] *MobSF Documentation*. [Online]. Available: https://mobsf.github.io/docs/#/