

Mobile Devices Security

Fernando García Sanz(100346043)

Francisco Manuel Colmenar Lamas (100346094)

March 26, 2019



Contents

1	Introduction	2
2	Modification of the "CredHub" app	3
2.1	Creation of "Title Screen" Activity	3
2.2	Authentication Process	3
2.2.1	Basic HTTP Authentication	5
3	Reverse Engineering and Repacking	7
3.1	Obfuscating "CreadHub"	7
3.2	Decompiling and modifying "CreadHub" using smali code	8
3.3	Repacking the modified Application	10
3.4	Resigning and installing the modified Application	11
3.5	Verifying the "new" credentials	11
3.6	Analyzing CreadHub HTTP traffic	12
4	Conclusions	15
	Bibliography	16

1 Introduction

Data privacy is a main issue nowadays. Huge information leaks are produced everyday, and this is a problem that does not stop but to grow, and most people do not seem to be conscious about it.

In this practice several tasks are going to be performed to show vulnerabilities of an android application related to credentials privacy and management.

The application developed in previous stages of the course needs to be improved by the addition of a log in activity, in which the user will need to introduce the proper credentials in order to access. Later on, the application behavior has to be modified by means of different techniques which will make possible to produce behavioral changes in the application which can be tapped by an attacker.

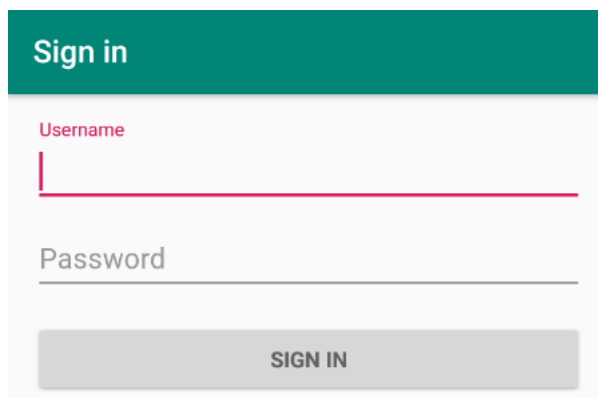
2 Modification of the "CredHub" app

2.1 Creation of "Title Screen" Activity

The first thing done has been the creation of a new activity which will allow the application user to authenticate himself, allowing the access to his credentials data.

To do so, a new activity called *LoginActivity* has been developed. It only consists of two fields, username and password, and their content must match with the stored credentials to allow the user access to the information stored in the application.

The activity design is the following:



2.2 Authentication Process

When the user introduces its credentials in the previous explained fields, the application will read them and check if they match with the application authentication credentials. To do so, the process is the following:

- First of all, six global variables are defined to store credentials and get the values from the input fields:

```
24
25     SharedPreferences sprefs;
26
27     TextView username;
28     TextView pass;
29
30     public static final String MyPreferences = "MyPrefs" ;
31     public static final String Username = "Username";
32     public static final String Password = "Password";
```

- On the *onCreate* method, the stored credentials are obtained in the global variable created before of type *SharedPreferences* *sprefs* and a listener is implemented over the sign in button, which will trigger the credentials verification once it is pressed calling the function *checkCredentials*:

```

40         sprefs = getSharedPreferences(MyPreferences, Context.MODE_PRIVATE);
41
42
43         Button submit = findViewById(R.id.sign_in_button);
44         submit.setOnClickListener((view) -> {
45             checkCredentials();
46         });
47
48
49     };

```

- The *checkCredentials* function is the one in charge of telling the system if the input credentials match with the stored ones or not, or if there was any problem during the credentials input or comparison:
 - It first obtains the input credentials and converts them to *String* type to store them into local variables.

```

username = findViewById(R.id.Username);
pass = findViewById(R.id.Password);

String name = username.getText().toString(); // Input username
String password = pass.getText().toString(); // Input password

```

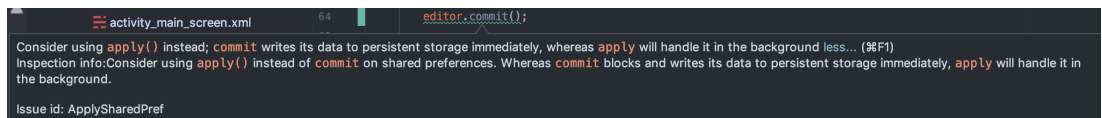
- Then, it checks if there are stored credentials. If there are not, then it checks if the user has introduced both username and password, saves them as the login credentials into internal storage, previously hashing the password, which will be later explained, and allows the user to access the main page.

```

60         if (!sprefs.contains(Username) && !sprefs.contains>Password)) {
61             // Add user and password
62
63             if(name.equals("")) {
64                 Toast.makeText(getApplicationContext(), text: "Username cannot be empty", Toast.LENGTH_LONG).show();
65             }
66
67             else if(password.equals("")) {
68                 Toast.makeText(getApplicationContext(), text: "Password cannot be empty", Toast.LENGTH_LONG).show();
69             }
70
71             else {
72                 SharedPreferences.Editor editor = sprefs.edit();
73                 /* Here the credentials are saved when no credentials saved before */
74
75                 editor.putString(Username, name);
76
77                 password = hashing(password);
78
79                 if(password != null) {
80                     editor.putString>Password, password);
81                 }

```

To save the credentials, *commit* functions is used, although it is not recommended by the IDE:



- In the case that there are stored credentials, input password is hashed and compared to the previously stored hash of the password and usernames are compared too. If they match, the user is allowed to continue, if not, a message will be displayed telling him that the credentials do not match with the stored ones.

```
98      String credName = sprefs.getString(Username, s1: "");
99      String credPass = sprefs.getString>Password, s1: "");
100
101      if(check(credName, name) && check(credPass, hashing(password))){
102          Intent unInt = new Intent( packageContext: LoginActivity.this, MainScreen.class);
103          startActivity(unInt);
104          return;
105      }
106      else {
107          Toast.makeText(getApplicationContext(), text: "Credentials do not match", Toast.LENGTH_LONG).show();
108      }
```

The comparison is done through *check* function, which simply checks if two *Strings* are equal, introducing as parameter for password comparison the function *hashing* which has as parameter the input password.

Hashing function is the one used to hash the stored password and the input one. It uses *SHA* algorithm for hashing and returns the hashed password.

```
116      public String hashing(String word) {
117
118          MessageDigest messageDigest;
119
120          try {
121              messageDigest = MessageDigest.getInstance("SHA");
122              messageDigest.update(word.getBytes());
123              byte [] messageDigestSHA = messageDigest.digest();
124              return (new String(messageDigestSHA));
125          }
126
127          catch (NoSuchAlgorithmException e){
128              e.printStackTrace();
129              return null;
130          }
131      }
```

SHA is the selected algorithm because it has demonstrated a huge resistance against diverse kinds of attacks, being even difficult to break even when some mathematical vulnerabilities were found in its version *SHA-1*.

2.2.1 Basic HTTP Authentication

According to the *HTTP Authentication*, its complexity is mainly in *TalkToServer* class. This class is the one in charge of establishing the connection with the server. Firstly, the server has to be

run using the following command:

```
MacBook-Pro-de-Fran:Server fran$ sudo java --add-modules java.se.ee -jar SDM_WebRepo.jar http+auth
```

Once that the server is running and it is waiting for a *HTTP* connection using *basic authentication*, the class *TalkToServer* has to be slightly modified. More specifically, the method *connectServer* is modified in order to introduce the authentication values in the header of the HTTP request.

```
/**
 * Perform the connection to the server adding the authentication information in the header
 */
private void connectServer() {
    // Set HTTP URL
    androidHttpTransport = new HttpTransportSE(URL);
    // Basic authentication
    headerList_basicAuth = new ArrayList<>();
    String strUserPass = Constant.BASIC_AUTH_USERNAME + ":" + Constant.BASIC_AUTH_PASSWORD; // Setting the String with the username and the password
    headerList_basicAuth.add(new HeaderProperty("Authorization", "Basic " +
        org.kobjects.base64.Base64.encode(strUserPass.getBytes()))); // Set the Header with the Authentication information
}
```

Furthermore, the variables used at *connectServer* are defined in the class *Constant*. They are set as default values in order to be able to connect with the provided server.

```
// Authentication values for server connection
static final String BASIC_AUTH_USERNAME = "sdm";
static final String BASIC_AUTH_PASSWORD = "repo4droid";
```

Once the connection is done, the client is able of accessing to the data stored in it as s/he could in the previous Assignment.

3 Reverse Engineering and Repacking

3.1 Obfuscating "CreadHub"

The aim of this section is *obfuscating* our application developed in the former section. Obfuscate is the act of making something, in our case our source code, harder to understand [1]. In order to accomplish this objective, all the variables and method names of our application, or any application obfuscated, will be converted into two character strings.

Nevertheless, this process does not make understanding the flow and logic of an application impossible, it only makes it more difficult.

There are plenty of different Java obfuscators however in this case *ProGuard* is the one which is going to be used. Moreover, there is no need to install it in order to perform the obfuscation due to the fact that it is already integrated into *Android Studio*.

Firstly, the file *build.gradle* needs to be modified. The field called *minifyEnabled* has to be set to *true*.

```
buildTypes {  
    release {  
        minifyEnabled true  
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'  
    }  
}
```

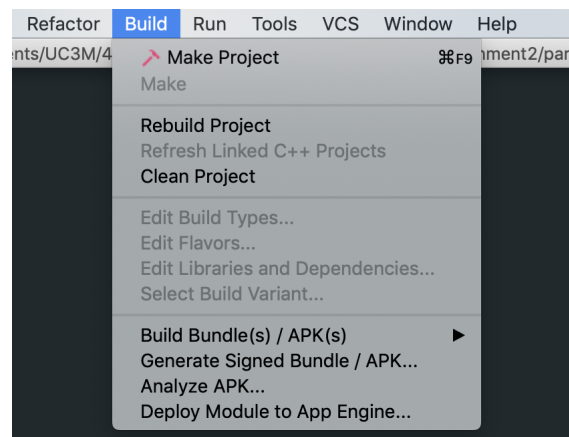
However, ProGuard find difficulties when obfuscating in some especial situations, such as when a class is referenced only in the *AndroidManifest.xml* file. As a consequence, this operation has to be done with care. In the case that any error happened during the obfuscation, such as a *ClassNotFoundException* appeared, the following code could be added into *proguard-rules.pro* in order to fix this problem.

```
-keep public class IgnoredClass # Example class
```

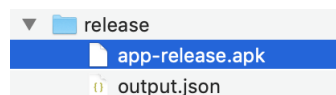
An additional advantage of obfuscating an application is the fact that the size of the obfuscated application is reduced. This is true thanks to the removal of unused classes, such as the parts of

the imported libraries which are not called in the code. Once that the process of obfuscation is done, a comparison between the size of both, the non-obfuscated and the obfuscated application will be done.

The following steep is *Rebuilding* the application and then *Generating a Signed APK*. Both options are in the tab called *Build* available from Android Studio. The process of Signing the application is not going to be described in this assignment. If further information is needed the previous assignment could be used for gaining an insight about this process.



Once this step is done, the obfuscated app is saved in the given path.



If the size of both apk are compared, the obfuscated size is of 1,957,019 bytes while the none obfuscated is of 2,301,219 bytes. Consequently, the size of the apk has been reduced by a 14.96%.

3.2 Decompiling and modifying "CreadHub" using smali code

In this section, the application previously obfuscated and signed into an apk will be decompiled and modified using smali code. Firstly, the apk needs to be decompiled. The tool which is going to be used in order to performed the decompilation is *ApkTool*. This tool decompress the apk file.

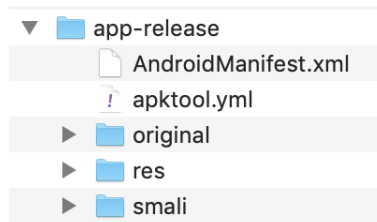
Firstly, the command to run ApkTool and decompressed the application have to be execute.

```

MacBook-Pro-de-Fran:smali code fran$ apktool decode app-release.apk
I: Using Apktool 2.4.0 on app-release.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
S: WARNING: Could not write to (/Users/fran/Library/apktool/framework), using /var/folders/pp/x97px8nx0m95rm2sc_wzvwkc0000gn/T/ instead...
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize --frame-path if the default storage directory is unavailable
I: Loading resource table from file: /var/folders/pp/x97px8nx0m95rm2sc_wzvwkc0000gn/T/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...

```

After that, the apk file is decompressed resulting in the following files.



As it can be seen, there is a folder called *smali*. In this folder is where the application translated into smali code is located. Consequently, here is the place where the code has to be modified in order to bypass the log in authentication.

In the case of this assignment, the file which contains the log in procedure is *LoginActivity.smali*. Once opened, the method or part of the code which compare the password need to be searched. In this case, the method in charge of this checking is called *checkCredentials*.

```

.method public checkCredentials()V
    .locals 7

    const v0, 0x7f080010

    invoke-virtual {p0, v0}, Lcom/example/credhub/LoginActivity;->findViewById(I)Landroid/view/View;

    move-result-object v0

    check-cast v0, Landroid/widget/TextView;

    iput-object v0, p0, Lcom/example/credhub/LoginActivity;->username:Landroid/widget/TextView;

    const v0, 0x7f08000a

    invoke-virtual {p0, v0}, Lcom/example/credhub/LoginActivity;->findViewById(I)Landroid/view/View;

    move-result-object v0

```

At first sight it might look difficult to understand. However, if the original code and the smali is compared it is not difficult to find similarities. If the bottom of this method is looked, in the smali version, the following piece of code is found.

```

205     move-result-object v4
206
207     invoke-virtual {p0, v2, v0}, Lcom/example/credhub/LoginActivity;->check(Ljava/lang/String;Ljava/lang/String;)Z
208
209     move-result v0
210
211     if-eqz v0, :cond_4
212
213     invoke-virtual {p0, v1}, Lcom/example/credhub/LoginActivity;->hashing(Ljava/lang/String;)Ljava/lang/String;
214
215     move-result-object v0
216
217     invoke-virtual {p0, v4, v0}, Lcom/example/credhub/LoginActivity;->check(Ljava/lang/String;Ljava/lang/String;)Z

```

This is the "if" statement which compares the password introduced by the user. Consequently, if the objective is bypassing the log in authentication, the only needed step to be done is deleting this check. In this case, no matter the password introduced by the user, he will have access to the application.

3.3 Repacking the modified Application

Once that the original application has been modified, the next step to be accomplish is recompiling and repacking the new application. In order to perform this action the tool *ApkTool* previously used will be used again.

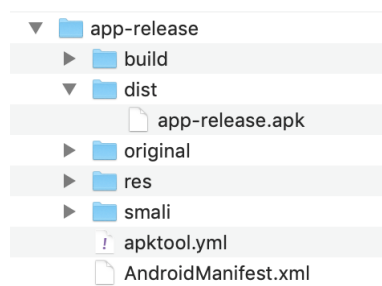
The command needed to accomplish the recompiling of the application is the following one.

```

MacBook-Pro-de-Fran:app-release fran$ apktool b
I: Using Apktool 2.4.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
W: Unknown file type, ignoring: ./smali/.DS_Store
W: Unknown file type, ignoring: ./smali/com/.DS_Store
W: Unknown file type, ignoring: ./smali/com/example/.DS_Store
I: Checking whether resources has changed...
I: Building resources...
S: WARNING: Could not write to (/Users/fran/Library/apktool/framework), using /var/folders/pp/x97px8nx0m95rm2sc_wzvwkc0000gn/T/ instead...
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize --frame-path if the default storage directory is unavailable
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...

```

Finally, the modified application recompiled is saved into the *dist* folder.



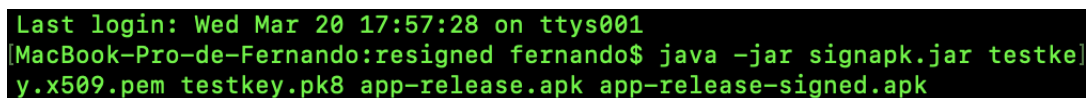
3.4 Resigning and installing the modified Application

Once the previous tasks have been done, it is time of resigning the modified application.

To do so, *SignApk tool* is used with the following command, substituting the last two parameters by the application name and the desired signed one name:

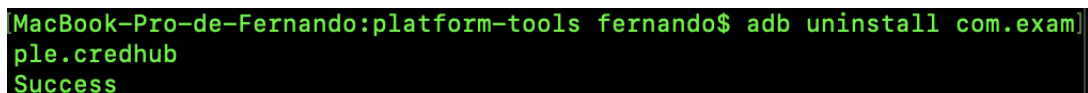
```
java -jar signapk.jar testkey.x509.pem testkey.pk8 <updated.apk><update-signed.apk>
```

This can be seen in the following picture:



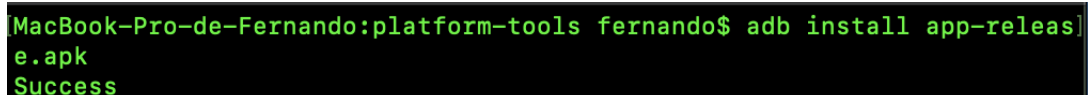
```
Last login: Wed Mar 20 17:57:28 on ttys001
[MacBook-Pro-de-Fernando:resigned fernando$ java -jar signapk.jar testkey.x509.pem testkey.pk8 app-release.apk app-release-signed.apk
```

Now it is necessary to delete the already installed application in the emulator and substitute it by the new signed one. This is done as it can be seen in the following screenshot, from inside *platform-tools* folder:



```
[MacBook-Pro-de-Fernando:platform-tools fernando$ adb uninstall com.example.credhub
Success
```

Once the application has been uninstalled, it is necessary to install the modified version, changing its name by the one the original one had.



```
[MacBook-Pro-de-Fernando:platform-tools fernando$ adb install app-release-signed.apk
Success
```

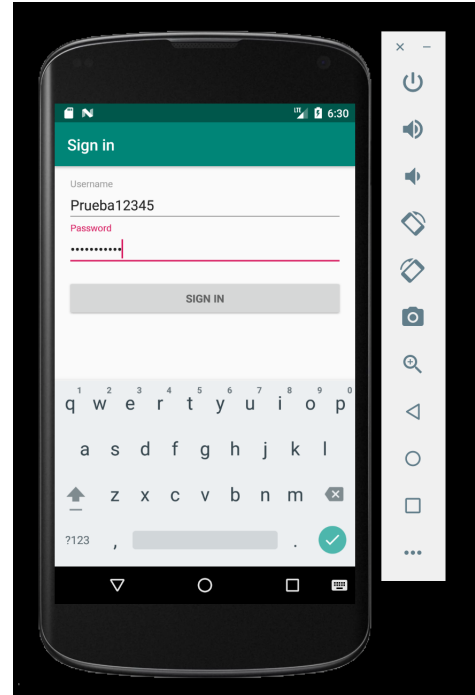
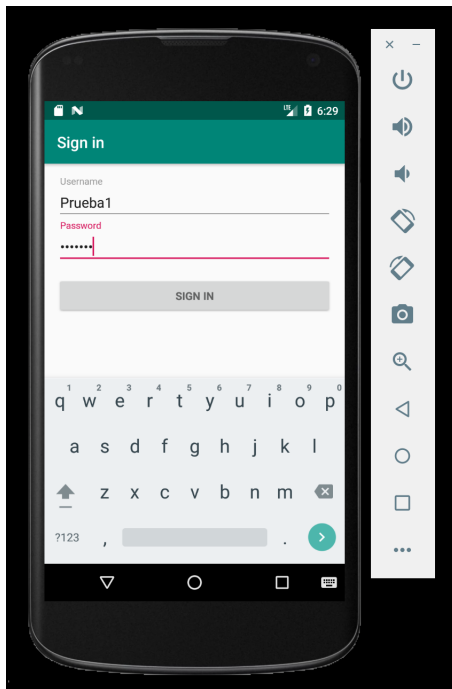
3.5 Verifying the "new" credentials

Once the modified application has been installed into the emulator, the performed changes must be checked.

The code modification allows to sign in into the application with any credentials, not the ones which were stored into the local storage of the emulator.

A successful login has been achieved using two different pairs of credentials, accessing to the main page, so it is possible to conclude that the application behavior has been correctly modified according to the exposed requirements.

This can be seen with the two different tests:



3.6 Analyzing CreadHub HTTP traffic

Finally, once the new application version has been installed and tested, we have to analyze the *HTTP* traffic of the original application using *Wireshark* application.

First, it is necessary to start the emulator properly using the following command:

```
MacBook-Pro-de-Fernando:emulator fernando$ ./emulator -list-avds
Nexus_4_API_24
MacBook-Pro-de-Fernando:emulator fernando$ ./emulator -tcpdump webtraffic.cap -avd Nexus_4_API_24
```

Once the emulator is initialized, it is also necessary to start the server so the android application can communicate with it:

```
MacBook-Pro-de-Fernando:Server fernando$ sudo java -jar SDM_WebRepo.jar http+auth
Password:
```

Once both things have been properly initialized, when a connection to the server has been performed from the application, the following data can be seen in *Wireshark*:

→	507	27.254317	127.0.0.1	127.0.0.1	HTTP/...	668	POST /SDM/WebRepo?wsdl HTTP/1.1
←	558	27.545021	127.0.0.1	127.0.0.1	HTTP/...	61	HTTP/1.1 200 OK

Analyzing these packets, the following data is extracted:

In the request message:

- ▶ Frame 507: 668 bytes on wire (5344 bits), 668 bytes captured (5344 bits) on interface 0
- ▶ Null/Loopback
- ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- ▶ Transmission Control Protocol, Src Port: 55506, Dst Port: 80, Seq: 1, Ack: 1, Len: 612

▼ Hypertext Transfer Protocol

- ▼ POST /SDM/WebRepo?wsdl HTTP/1.1\r\n
 - ▶ [Expert Info (Chat/Sequence): POST /SDM/WebRepo?wsdl HTTP/1.1\r\n]
 - Request Method: POST
 - Request URI: /SDM/WebRepo?wsdl
 - Request Version: HTTP/1.1
 - User-Agent: ksoap2-android/2.6.0+\r\n
 - SOAPAction: "http://sdm_webrepo/ListCredentials"\r\n
 - Content-Type: text/xml; charset=utf-8\r\n
 - Accept-Encoding: gzip\r\n
 - ▼ Authorization: Basic c2Rt0nJlcG80ZHJvaWQ=\r\n
 - Credentials: sdm:repo4droid
 - ▼ Content-Length: 327\r\n
 - [Content length: 327]
 - Host: 10.0.2.2\r\n
 - Connection: Keep-Alive\r\n
 - \r\n
 - [Full request URI: http://10.0.2.2/SDM/WebRepo?wsdl]
 - [HTTP request 1/1]
 - [Response in frame: 558]
 - File Data: 327 bytes

- ▶ eXtensible Markup Language

In the response message:

- ▶ Frame 558: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface 0
- ▶ Null/Loopback
- ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- ▶ Transmission Control Protocol, Src Port: 80, Dst Port: 55506, Seq: 498, Ack: 613, Len: 5
- ▶ [4 Reassembled TCP Segments (502 bytes): #552(123), #554(100), #556(274), #558(5)]

▼ Hypertext Transfer Protocol

- ▼ HTTP/1.1 200 OK\r\n
 - ▶ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
 - Request Version: HTTP/1.1
 - Status Code: 200
 - [Status Code Description: OK]
 - Response Phrase: OK
 - Date: Wed, 20 Mar 2019 17:58:42 GMT\r\n
 - Transfer-encoding: chunked\r\n
 - Content-type: text/xml; charset=utf-8\r\n
 - \r\n
 - [HTTP response 1/1]
 - [Time since request: 0.290704000 seconds]
 - [Request in frame: 507]
 - ▼ HTTP chunked response
 - ▶ Data chunk (94 octets)
 - ▶ Data chunk (267 octets)
 - ▶ End of chunked encoding
 - \r\n
 - File Data: 361 bytes

- ▶ eXtensible Markup Language

Analyzing the response, the whole list of credentials is sent in XML format:

```

▼ eXtensible Markup Language
  ▶ <?xml
  ▼ <S:Envelope
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
    ▼ <S:Body>
      ▼ <ns2:ListCredentialsResponse
        xmlns:ns2="http://sdm_webrepo/"
        ▼ <return>
          HOLZ
          </return>
        ▼ <return>
          Eee
          </return>
        ▼ <return>
          Uranus
          </return>
        ▼ <return>
          Guille
          </return>
        ▼ <return>
          Fernando
          </return>
        ▼ <return>
          Earth
          </return>
        ▼ <return>
          007
          </return>
        </ns2:ListCredentialsResponse>
      </S:Body>
    </S:Envelope>

```

If any of these credentials is clicked, then a new request is sent to the server, which will answer with the following message (in this case, the first credentials set):

```

▼ eXtensible Markup Language
  ▶ <?xml
  ▼ <S:Envelope
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
    ▼ <S:Body>
      ▼ <ns2:ImportRecordResponse
        xmlns:ns2="http://sdm_webrepo/"
        ▼ <return>
          HOLZ
          </return>
        ▼ <return>
          JA
          </return>
        ▼ <return>
          BDBD
          </return>
        </ns2:ImportRecordResponse>
      </S:Body>
    </S:Envelope>

```

So, it can be seen that analyzing the HTTP traffic generated by the application, as the messages are not encrypted, it is possible to extract all the information sent by the server, so all requested credentials could be sniffed by an attacker if this procedure continues being used in the application.

4 Conclusions

This practice has made us conscious about the different kinds of simple vulnerabilities an application can have if security is not taken into account.

Hashing all sensitive information must be a mandatory practice if we want to enhance data security and protect users' privacy in an application, specially if it performs internet activities. Furthermore, obfuscating the applications should be a mandatory step which all the application developers should accomplish in order to deliver their product to the final consumers.

Additionally, the importance of understanding and being able of writing smali code has been clearly showed in the previous sections. Thanks to this knowledge, a greater level of security could be provided as a consequence of been aware of the possible vulnerabilities which your application could suffer once it is translated into smali code.

The process of resigning an application and substituting the original one by it can be done easily if the attacker counts with the required knowledge and experience, and it can be a quite dangerous practice. Moreover, not only the application but the connections it performs to different environments must be considered, since it has been seen that even though an application is secure, if the protocols it uses to share information are not secure, it is possible to extract these data during the connection.

Summing up, security in a mobile application is a high importance topic, thus even in a small program, vulnerabilities can be anywhere.

References

- [1] C. E. Dictionary, *Meaning of obfuscate in english*, Available at <https://dictionary.cambridge.org/dictionary/english/obfuscate>.