



# Seguridad en Smartphones

Adán Avilés

Octubre 2021

# Índice

<b>1. Presentación del problema</b>	<b>4</b>
<b>2. Análisis de InsecureBankV2</b>	<b>5</b>
2.1. Análisis estático . . . . .	6
2.1.1. Permisos . . . . .	6
2.1.2. Actividades . . . . .	8
2.1.3. Receivers . . . . .	8
2.1.4. ContentProvider . . . . .	9
2.1.5. Almacenamiento de las credenciales . . . . .	9
2.1.6. Comunicaciones . . . . .	10
<b>3. Análisis de Instragram</b>	<b>14</b>
3.1. Análisis estático . . . . .	14
3.1.1. Permisos . . . . .	14
3.1.2. Actividades . . . . .	15
3.1.3. Receivers . . . . .	16
3.1.4. ContentProvider . . . . .	16
3.1.5. Almacenamiento de credenciales . . . . .	16
3.1.6. Comunicaciones . . . . .	16
3.2. Análisis dinámico . . . . .	17
<b>4. Análisis de MyFitnessPal</b>	<b>18</b>
4.1. Análisis estático . . . . .	18
4.1.1. Permisos . . . . .	18
4.1.2. Actividades . . . . .	19
4.1.3. Receivers . . . . .	19
4.1.4. ContentProvider . . . . .	20
4.1.5. Almacenamiento de credenciales . . . . .	21
4.2. Análisis dinámico . . . . .	21

---

<b>5. Anexos</b>	<b>22</b>
5.1. Problemas de versionado . . . . .	22
5.2. Lista de OWASP TOP 10 MOBILE RISKS . . . . .	22
5.3. Análisis de riesgos OWASP con Quixxi . . . . .	22

## 1. Presentación del problema

A lo largo de los contenidos se ha hablado de OWASP, análisis estático y análisis dinámico, vulnerabilidades, desarrollo seguro, etc. Los Smartphones no quedan libres de ataques. OWASP define un top 10 de riesgos en aplicaciones para dispositivos móviles, como podemos ver en [https://www.owasp.org/index.php/Mobile\\_Top\\_10\\_2016-Top\\_10](https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10).

A partir de los contenidos estudiados y tras leer el enunciado anterior, analiza al menos tres aplicaciones (archivos .apk) en busca de vulnerabilidades. Para ello utilizaremos

- Una apk no oficial, con vulnerabilidades deliberadamente introducidas, y que se publican en internet para prácticas de entrenamiento, **InsecureBankV2**.
- Una descargada de algún repositorio no oficial de apks, **MyFitnessPal free premium**.
- Una descargada desde una tienda oficial, **Instagram**.

Para hacer este trabajo se han usado distintas herramientas como

- MobSF
- ADB (Android Debug Bridge)
- Distintas páginas online como VirusTotal o Quixxi
- Kali Linux
- Santoku
- Genymotion

## 2. Análisis de InsecureBankV2

Cargaremos la apk en MobSF, para poder hacer los análisis:

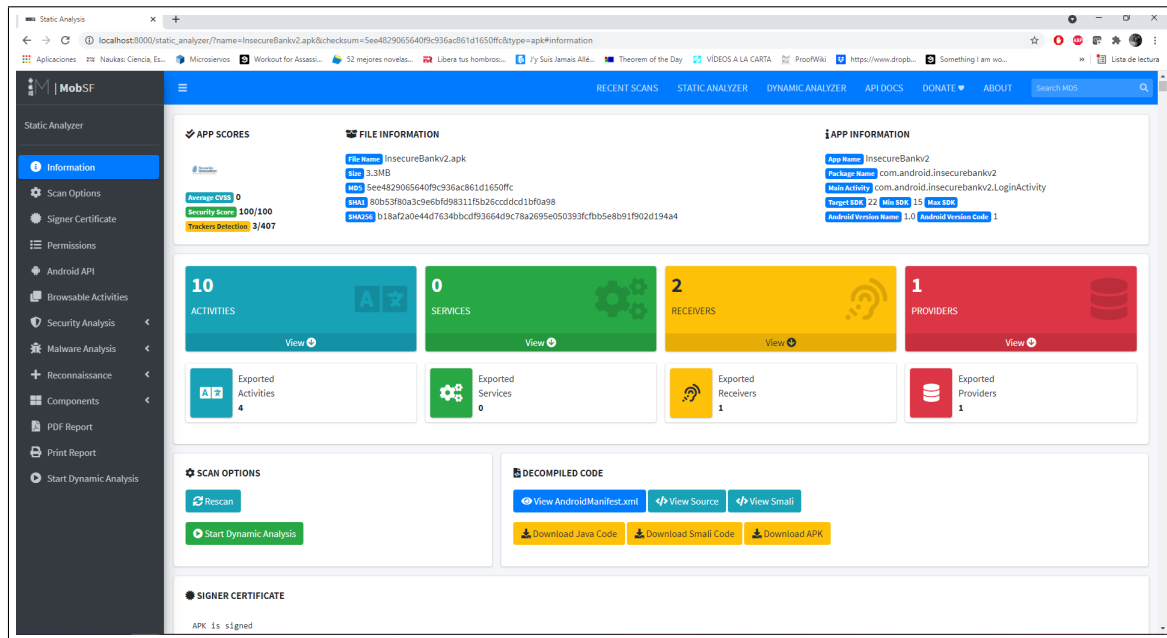


Figura 1: Imagen inicial del análisis estático de MobSF

también podemos utilizar herramientas online como *virustotal*

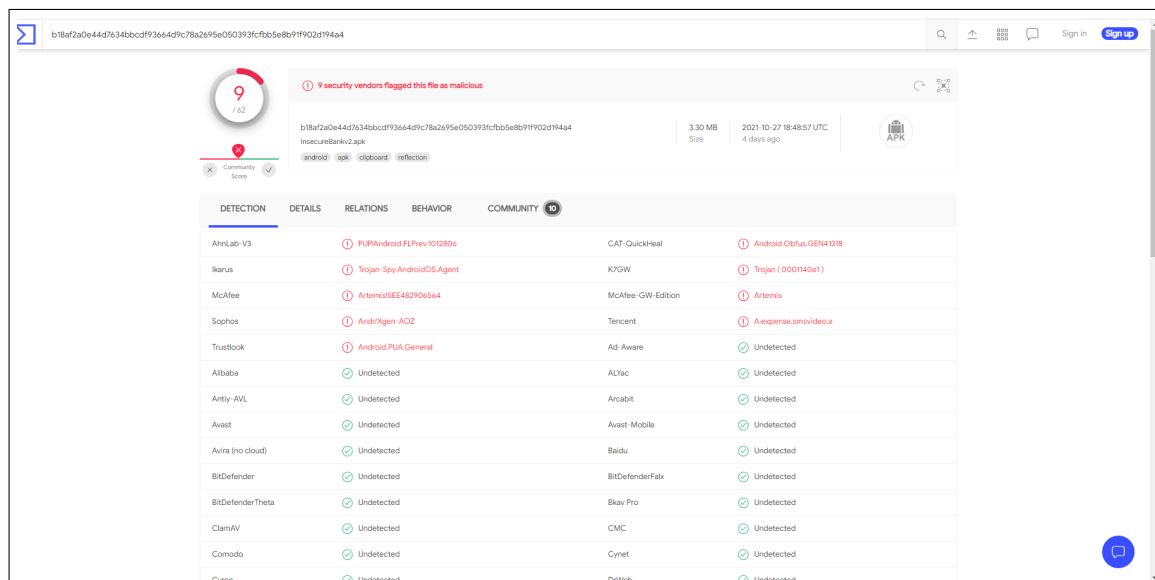


Figura 2: Análisis con VirusTotal

## Intezer

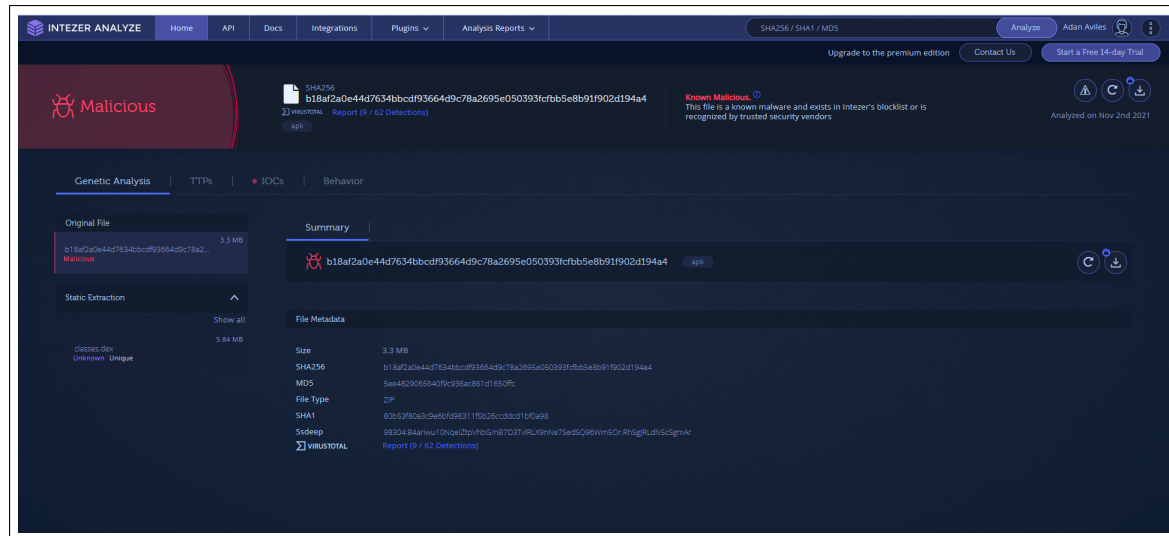


Figura 3: Análisis con Intezer

## Quixxi

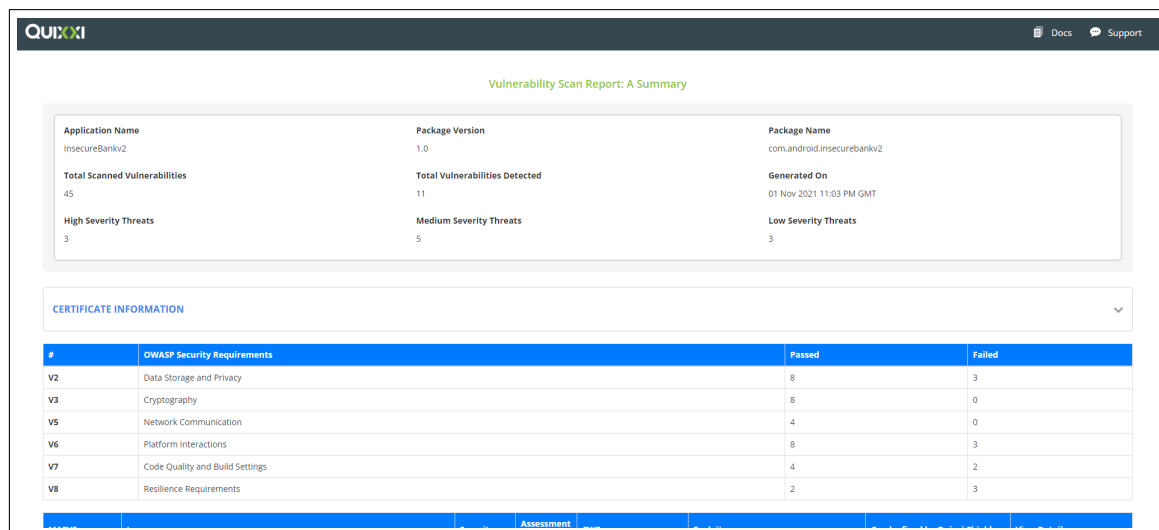


Figura 4: Análisis con Quixxi

## 2.1. Análisis estático

### 2.1.1. Permisos

Revisaremos en primer lugar el AndroidManifest.xml para ver los permisos.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest android:versionCode="1" android:versionName="1.0" package="com.android.insecurebankv2" platformSdkVersion="codeName"2" platformBuildVersionName="5.1.1" (191972"
3  xmlns:android="http://schemas.android.com/apk/res/android">
4      <uses-permission android:name="android.permission.INTERNET">
5          <!-->
6      </uses-permission android:name="android.permission.INTERNET">
7      <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE">
8          <!-->
9      </uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE">
10     <uses-permission android:name="android.permission.READ_PHONE_STATE">
11         <!-->
12     </uses-permission android:name="android.permission.READ_PHONE_STATE">
13     <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE">
14         <!-->
15     </uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE">
16     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE">
17         <!-->
18     </uses-permission android:name="android.permission.ACCESS_NETWORK_STATE">
19     <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION">
20         <!-->
21     </uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION">
22     <uses-feature android:glEsVersion="0x00000200" android:required="true">
23         <!-->
24     </uses-feature android:glEsVersion="0x00000200" android:required="true">
25     <application android:theme="@android:style/Theme.Holo.Light.DarkActionBar" android:label="@string/app_name" android:icon="@mipmap/ic_launcher" android:debuggable="true" android:allowBackup="true">
26         <activity android:label="@string/app_name" android:name="com.android.insecurebankv2.MainActivity">
27             <intent-filter>
28                 <action android:name="android.intent.action.MAIN">
29                     <!-->
30                 </action android:name="android.intent.action.MAIN">
31                 <category android:name="android.intent.category.LAUNCHER">
32                     <!-->
33                 </category android:name="android.intent.category.LAUNCHER">
34             </intent-filter>
35         </activity>
36         <activity android:label="@string/title_activity_file_picker" android:name="com.android.insecurebankv2.FilePickerActivity" android:windowSoftInputMode="adjustNothing|stateVisible">
37             <!-->
38         </activity android:label="@string/title_activity_file_picker" android:name="com.android.insecurebankv2.FilePickerActivity" android:windowSoftInputMode="adjustNothing|stateVisible">
39         <activity android:label="@string/title_activity_login" android:name="com.android.insecurebankv2.LoginActivity">
40             <!-->
41         </activity android:label="@string/title_activity_login" android:name="com.android.insecurebankv2.LoginActivity">
42         <activity android:label="@string/title_activity_register" android:name="com.android.insecurebankv2.RegisterActivity">
43             <!-->
44         </activity android:label="@string/title_activity_register" android:name="com.android.insecurebankv2.RegisterActivity">
45         <activity android:label="@string/title_activityForgotPassword" android:name="com.android.insecurebankv2.ForgotPasswordActivity">
46             <!-->
47         </activity android:label="@string/title_activityForgotPassword" android:name="com.android.insecurebankv2.ForgotPasswordActivity">
48         <provider android:name="com.android.insecurebankv2.TransactionContentProvider" android:authorities="com.android.insecurebankv2.TransactionContentProvider">
49             <!-->
50         </provider android:name="com.android.insecurebankv2.TransactionContentProvider" android:authorities="com.android.insecurebankv2.TransactionContentProvider">
51         <receiver android:name="com.android.insecurebankv2.TransactionReceiver" android:exported="true">
52             <!-->
53         </receiver android:name="com.android.insecurebankv2.TransactionReceiver" android:exported="true">
54         <intent-filter>
55             <action android:name="android.intent.action.MAIN">
56                 <!-->
57             </action android:name="android.intent.action.MAIN">
58             <category android:name="android.intent.category.LAUNCHER">
59                 <!-->
60             </category android:name="android.intent.category.LAUNCHER">
61         </intent-filter>
62     </application>
63 </manifest>

```

**Figura 5:** *AndroidManifest.xml* de *InsecureBank*

Listaremos, a continuación, los permisos que pueden ser más peligrosos:

- **SEND\_SMS:** Este permiso, le dará a la app la capacidad de enviar mensajes SMS, que pueden suponer un coste para el usuario.
- **ACCESS\_COARSE\_LOCATION:** Da permisos para obtener la ubicación aproximada a través de la ubicación de red.
- **GET\_ACCOUNTS:** Permite a la aplicación obtener las cuentas conocidas almacenadas en el teléfono, como por ejemplo cuentas creadas por otras aplicaciones, es decir, podría obtener nuestros usuarios de otras aplicaciones.
- **READ\_CONTACTS:** Podrá leer los contactos almacenados en el teléfono, llamadas, correos enviados.... Esto permitiría que pueda recabar los datos del usuario y robarlos.
- **READ\_PROFILE:** Lee los datos del usuario.
- **READ\_LOG:** Lee los datos del usuario.
- **ALLOW\_BACKUP:** Permite realizar BackUps de la aplicación
- **USE\_CREDENTIALS:** Permitirá a la aplicación solicitar tokens de autenticación.
- **WRITE\_EXTERNAL\_STORAGE:** Permite a la aplicación escribir en el almacenamiento externo.

### 2.1.2. Actividades

Sabemos que si un componente está exportamos en el `AndroidManifest.xml`, cualquier aplicación puede acceder a él si tiene los permisos adecuados. Así, hemos de tener cuidado con las actividades que exportamos, por ejemplo, **PostLogin**. Esta actividad puede ser usada en cualquier momento, así que si la ejecutamos, podemos evitar el tener que acceder con usuario y contraseña. También se puede hacer lo mismo con **DoTransfer** para acceder a las transferencias sin logging. (OWASP M6)

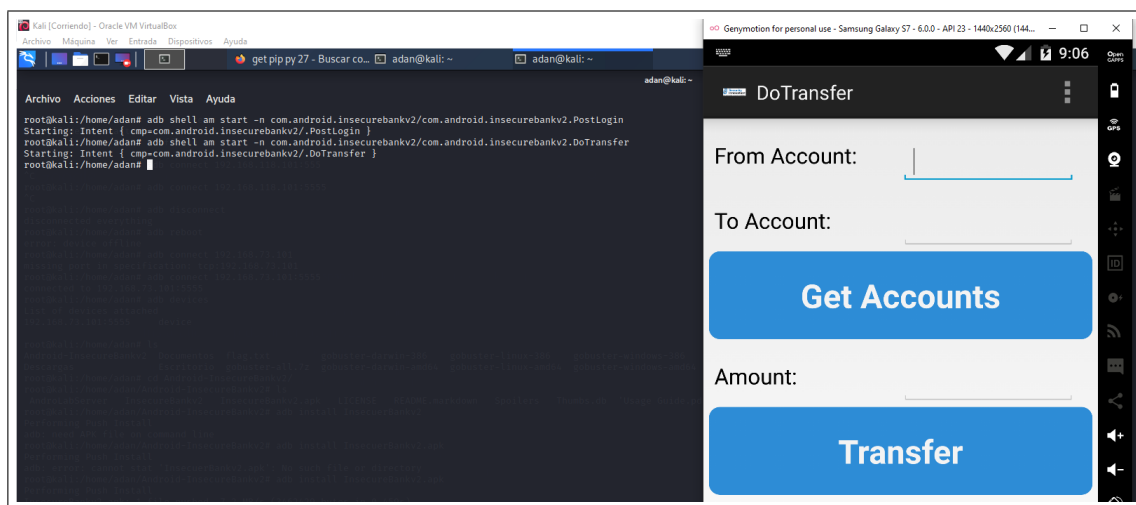


Figura 6: Acceso a transferencias

Por otro lado, y con los mismos problemas, estarían las actividades de

- `ChangePassword`
- `ViewStatement`
- `LoginActivity`

### 2.1.3. Receivers

Encontramos que **MyBroadcastReceiver** es accesible por otras aplicaciones y realiza la acción **theBroadcast**. Si revisamos el código, vemos que está relacionado con el cambio de contraseña, y que esta está encriptada en Base64 (OWASP M5), lo cual no es demasiado seguro. Además, podemos ver que su última orden es enviar un SMS, el atacante podría utilizar esto para robar dinero en forma de mensajes SMS sin el consentimiento del usuario.





```

File: MyBroadcastReceiver.java
1. package com.android.insecurebankv2;
2.
3. import android.content.BroadcastReceiver;
4. import android.content.Context;
5. import android.content.Intent;
6. import android.content.SharedPreferences;
7. import android.telephony.SmsManager;
8. import android.util.Base64;
9.
10. public class MyBroadcastReceiver extends BroadcastReceiver {
11.     public static final String MY_PREFS = "mySharedPreferences";
12.     String usernameBase64ByteString;
13.
14.     public void onReceive(Context context, Intent intent) {
15.         String phn = intent.getStringExtra("phonenumber");
16.         String newpass = intent.getStringExtra("newpass");
17.         if (phn != null) {
18.             try {
19.                 SharedPreferences settings = context.getSharedPreferences(MY_PREFS, 1);
20.                 this.usernameBase64ByteString = new String(Base64.decode(settings.getString("EncryptedUsername", null), 0), "UTF-8");
21.                 String decryptedPassword = new CryptoClass().aesDecryptedString(settings.getString("superSecurePassword", null));
22.                 String textPhoneno = phn.toString();
23.                 String textMessage = "Updated Password from: " + decryptedPassword + " to: " + newpass;
24.                 SmsManager smsManager = SmsManager.getDefault();
25.                 System.out.println("For the change password - phoneno: " + textPhoneno + " password is: " + textMessage);
26.                 smsManager.sendTextMessage(textPhoneno, null, textMessage, null, null);
27.             } catch (Exception e) {
28.                 e.printStackTrace();
29.             }
30.         } else {
31.             System.out.println("Phone number is null");
32.         }
33.     }
34. }

```

Figura 7: Código de *MyBroadcastReceiver*

#### 2.1.4. ContentProvider

El contentprovider **TrackUserContentProvider** no está protegido por ningún permiso, lo cual puede hacer que sea vulnerable a ataques de SQL injection.

#### 2.1.5. Almacenamiento de las credenciales

Este paso también se podría revisar en el análisis dinámico, pero lo haremos revisando el código. La aplicación permite la opción del *autofill* de credenciales, que estarán almacenadas en algún lugar. Buscaremos en el código de *DoLogin*.



```

/* access modifiers changed from: protected */
public void fillData() throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException {
    SharedPreferences settings = getSharedPreferences("mySharedPreferences", 0);
    String username = settings.getString("EncryptedUsername", null);
    String password = settings.getString("superSecurePassword", null);
    if (username != null && password != null) {
        try {
            this.usernameBase64ByteString = new String(Base64.decode(username, 0), "UTF-8");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        this.Username_Text = (EditText) findViewById(R.id.loginscreen_username);
        this.Password_Text = (EditText) findViewById(R.id.loginscreen_password);
        this.Username_Text.setText(this.usernameBase64ByteString);
        this.Password_Text.setText(new CryptoClass().aesDecryptedString(password));
    } else if (username == null || password == null) {
        Toast.makeText(this, "No stored credentials found!!", 1).show();
    } else {
        Toast.makeText(this, "No stored credentials found!!", 1).show();
    }
}

```

Figura 8: Código de *DoLogin*

Encontramos el método *fillData()*. Este método llama a un archivo *mySharedPreferences* (OWASP M2) que podíamos encontrar utilizando la herramienta de Android Debug

Bridge (ADB). Por otro lado, podemos desencriptar la string *SuperSecurePassword*, porque si nos vamos a la clase de *CryptoClass*, encontramos que está hardcodeda la clave de encriptado.

```
public class CryptoClass {  
    String base64Text;  
    byte[] cipherData;  
    String cipherText;  
    byte[] ivBytes = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};  
    String key = "This is the super secret key 123";  
    String plainText;
```

Figura 9: Código de *CryptoClass*

### 2.1.6. Comunicaciones

Podemos identificar las conexiones que realizará la aplicación en este paso. Revisaremos las clases que usen **postData**

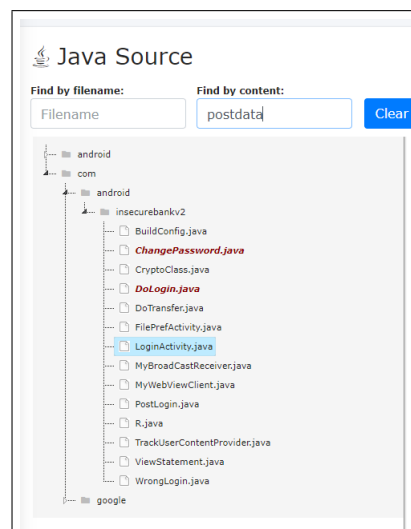


Figura 10: Búsqueda de posts

Podemos observar que en ambos casos se usa un protocolo HTTP, no seguro (OWASP M3):

```

public void postData(String valueWantToSend) throws ClientProtocolException, IOException, JSONException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException {
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost(ChangePassword.this.protocol + ChangePassword.this.serverip + ":" + ChangePassword.this.serverport + "/" + ChangePassword.this.changePassword);
    List<NameValuePair> nameValuePairs = new ArrayList<>();
    nameValuePairs.add(new BasicNameValuePair("username", ChangePassword.this.username));
    nameValuePairs.add(new BasicNameValuePair("password", ChangePassword.this.changePassword_text.getText().toString()));
    httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
    ChangePassword.this.matcher = Pattern.compile(ChangePassword.PASSWORD_PATTERN);
    ChangePassword.this.matcher = ChangePassword.this.matcher.matcher(ChangePassword.this.changePassword_text.getText().toString());
    if (ChangePassword.this.matcher.matches()) {
        InputStream in = httpClient.execute(httpPost).getEntity().getContent();
        ChangePassword.this.result = convertStreamToString(in);
        ChangePassword.this.result = ChangePassword.this.result.replace("\n", "");
        ChangePassword.this.runOnUiThread(new Runnable() {
            /* class com.android.insecurebankv2.ChangePassword.RequestChangePasswordTask$AnonymousClass */
            public void run() {
                if (ChangePassword.this.result != null && ChangePassword.this.result.indexOf("Change Password Successful") != -1) {

```

Figura 11: Código de *ChangePassword*

```

public class DoLogin extends Activity {
    public static final String MYPREFS = "mySharedPreferences";
    String password;
    String protocol = "http://";
    BufferedReader reader;
    String rememberme_password;
    String rememberme_username;
    String responseString = null;
    String result;
    SharedPreferences serverDetails;
    String serverip = "";
    String serverport = "";
    String superSecurePassword;
    String username;

```

Figura 12: Código de *DoLogin*

Esto también ocurre en la clase **DoTransfer**, que utiliza un **HttpPost** con protocolo HTTP.

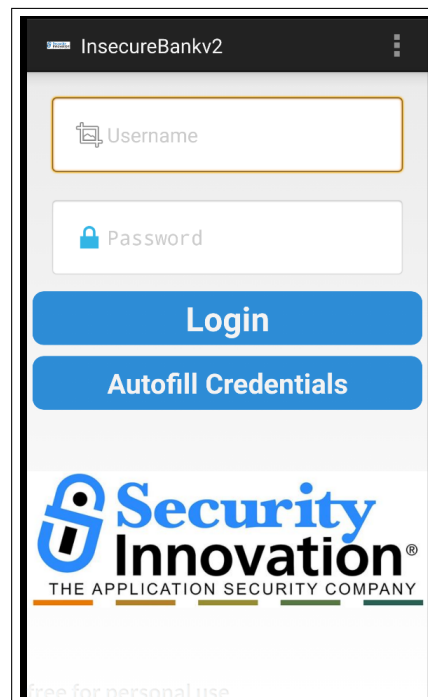
```

public String doInBackground(String... params) {
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost(DoTransfer.this.protocol + DoTransfer.this.serverip + ":" + DoTransfer.this.serverport + "/" + DoTransfer.this.getAccount());
    SharedPreferences settings = DoTransfer.this.getSharedPreferences("mySharedPreferences", 0);
    byte[] usernameBase64Byte = Base64.decode(settings.getString("EncryptedUsername", null), 0);
    try {
        DoTransfer.this.usernameBase64ByteString = new String(usernameBase64Byte, "UTF-8");
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    String password = settings.getString("superSecurePassword", null);
    try {

```

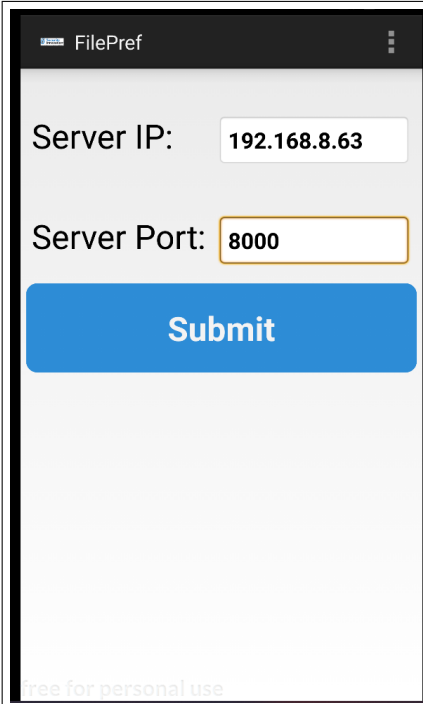
Figura 13: Código de *DoTransfer*

Análisis dinámico Si arrancamos la aplicación



**Figura 14:** Inicio de *InsecureBank*

y la configuramos para que apunte al servidor donde hemos levantado el backend de python (para más información revisar la documentación de InsecureBankv2)



FilePref

Server IP: 192.168.8.63

Server Port: 8000

Submit

free for personal use

Figura 15: Configuración de InsecureBank

Podremos analizar las comunicaciones con wireshark. En esta caso interceptamos una petición de login con el usuario *user* y contraseña por *passwd*

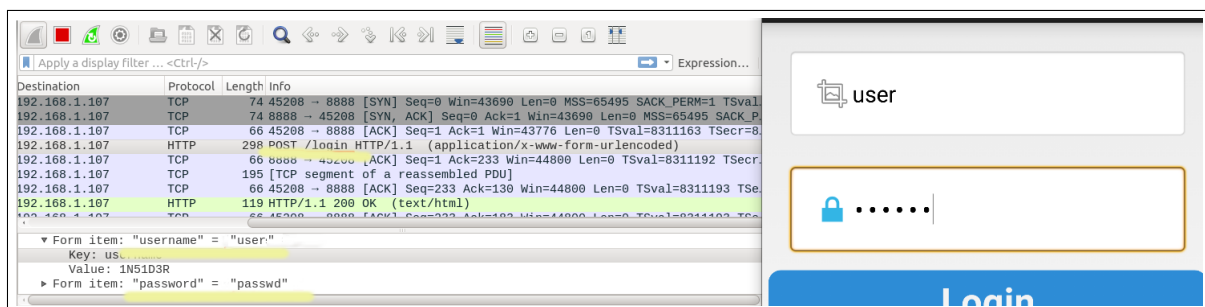


Figura 16: Configuración de InsecureBank

y podemos ver que no está cifrado, al enviarse por protocolo HTTP (OWASP M4)

### 3. Análisis de Instragram

Descargamos la apk oficial de Google de esta conocida red social. En primer lugar, hacemos el análisis con VirusTotal.

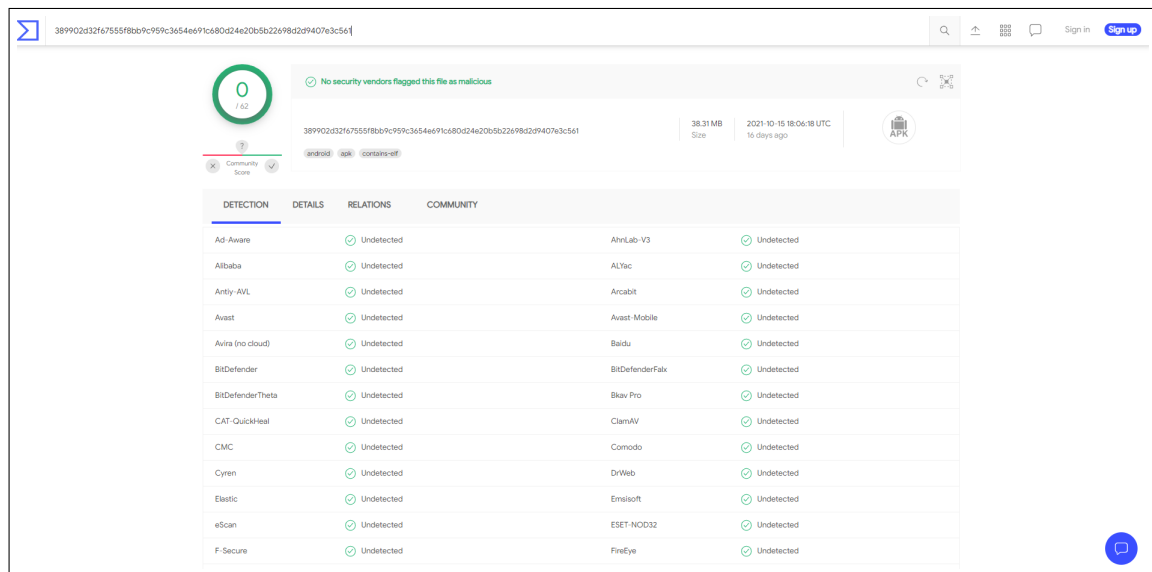


Figura 17: Análisis VirusTotal

#### 3.1. Análisis estático

##### 3.1.1. Permisos

Revisemos en primer lugar el AndroidManifest.xml en búsqueda de permisos *sospechosos*.

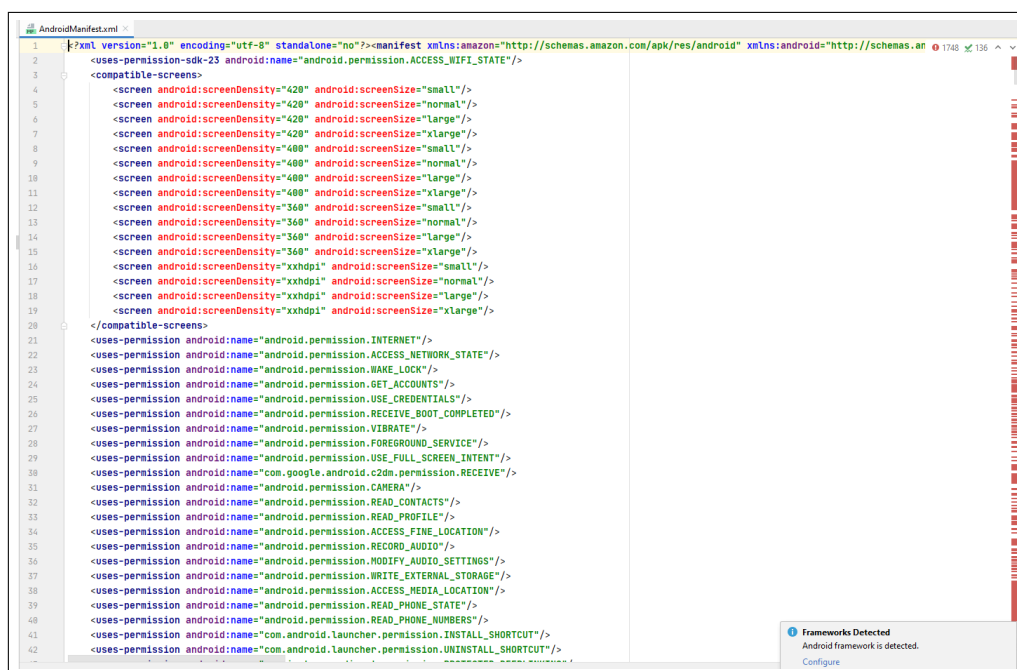


Figura 18: *AndroidManifest de Instagram*

En este caso, analizaremos la aplicación utilizando AndroidStudio, tras crear los binarios con apktool, básicamente porque MobSF, por algún motivo, no es capaz de procesarla.

- **WRITE\_EXTERNAL\_STORAGE**: Escritura en almacenamiento externo. Otras apps pueden beneficiarse de esto, sobrescribiendo o leyendo datos.
- **ACCESS\_MEDIA\_LOCATION**: Permite acceder a la localización de archivos multimedia.
- **GET\_ACCOUNTS**: Permite a la aplicación obtener las cuentas conocidas almacenadas en el teléfono, como por ejemplo cuentas creadas por otras aplicaciones, es decir, podría obtener nuestros usuarios de otras aplicaciones.
- **READ\_CONTACTS**: Podrá leer los contactos almacenados en el teléfono, llamadas, correos enviados.... Esto permitiría que pueda recabar los datos del usuario y robarlos.

### 3.1.2. Actividades

No se observan actividades peligrosas.

- `com.instagram.mainactivity.LauncherActivity`. Esta actividad está exportada, haciendo que se pueda usar por otras aplicaciones.

### 3.1.3. Receivers

No se observan *recibidores* peligrosos.

### 3.1.4. ContentProvider

No se observan ContentProviders peligrosas.

### 3.1.5. Almacenamiento de credenciales

Las credenciales se almacenan de forma segura, si revisamos el **LoginActivity**, podremos ver que Instagram funciona a través de un servidor externo de Python, al cual manda los datos de forma cifrada:

```
public class LoginActivity
    extends LoginBottomSheetActivity
{
    public static final String ACCOUNT_TYPE = "account_type";
    public static final String AUTH_TYPE = "auth_type";
    public static final String IS_ADDING_NEW_ACCOUNT = "is_adding_new_account";
    private String accountType;
    private String authType;
    private Boolean isNewAccount;

    private int getLatouyId()
    {
        return 2131493027;
    }
}
```

Figura 19: Código de Login de instagram.

### 3.1.6. Comunicaciones

Todas las comunicaciones son de la forma HTTPS.

```
<activity android:configChanges="keyboardHidden|orientation|screenSize" android:exported="true"
    android:name="com.instagram.url.UrlHandlerActivity" android:theme="@style/IgTranslucentWindow"
    <intent-filter android:autoVerify="true">
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="https"/>
        <data android:host="instagram.com"/>
        <data android:host="www.instagram.com"/>
        <data android:host="applink.instagram.com"/>
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="https"/>
        <data android:host="ig.me"/>
    </intent-filter>
```

Figura 20: Comunicación https de Instagram.



### **3.2. Análisis dinámico**

No se observan comportamientos anómalos en el análisis dinámico. Los datos no pueden ser interceptados con WireShark ni Burp. He probado a subir imágenes, pero Genymotion cierra la aplicación automáticamente.

## 4. Análisis de MyFitnessPal

En este punto, usaremos la aplicación *creackeada* descargada de un repositorio no oficial. Además, compararemos con su versión oficial. Esta aplicación sirve para contar calorías, y ver los planes nutricionales.

### 4.1. Análisis estático

En primer lugar, se observa que la versión oficial tiene un tamaño de 24MB frente a los 70MB de la versión *creackeada*. Veamos el análisis con VirusTotal de la aplicación *creackeada*.

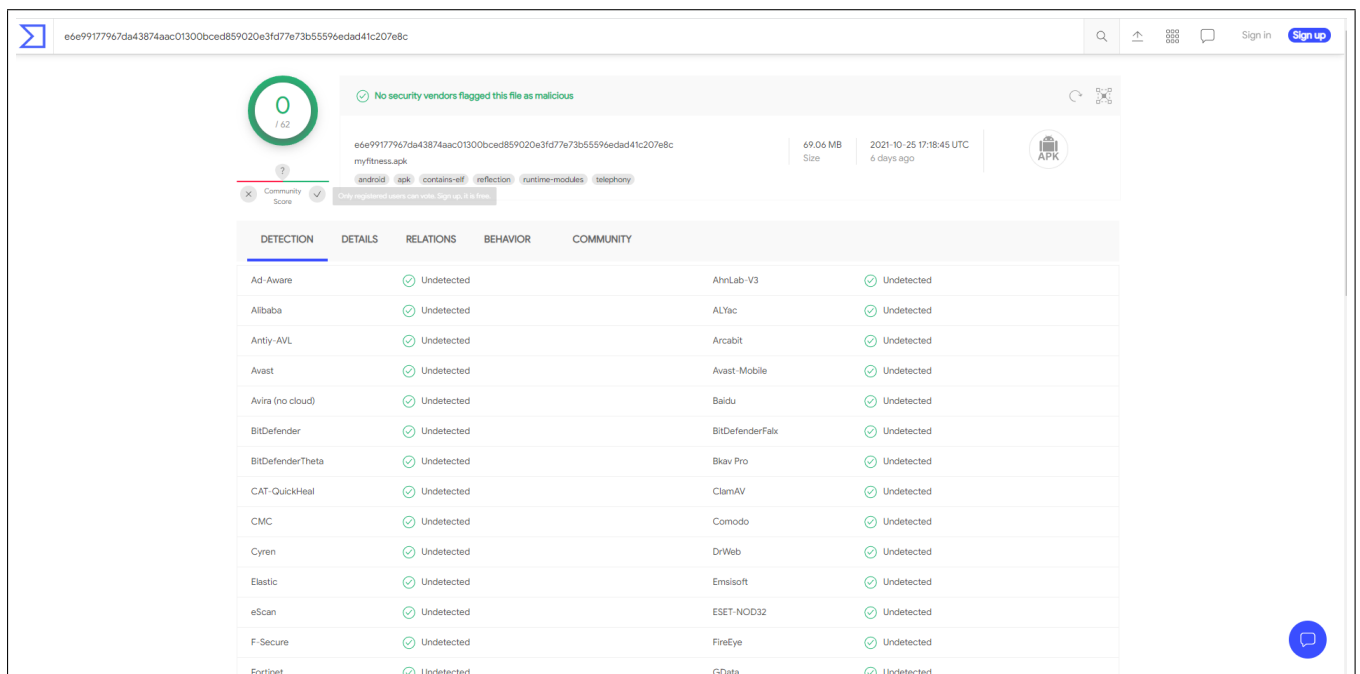


Figura 21: Comunicación https de Instagram.

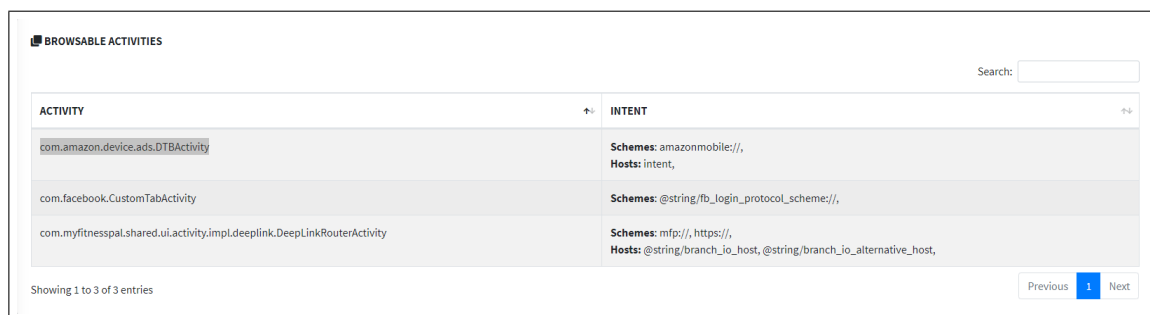
#### 4.1.1. Permisos

Ambas aplicaciones utilizan prácticamente los mismos permisos, sin embargo, la aplicación crackeada utiliza

- **android.permission.READ\_PHONE\_STATE**: Permite a la aplicación acceder al número de teléfono, número de serie, llamadas activas... Un permiso que no debería necesitar una aplicación de deporte.

### 4.1.2. Actividades

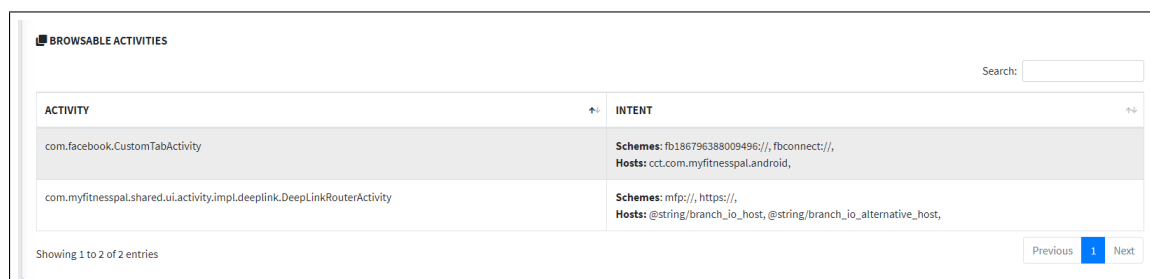
La aplicación original tiene una actividad que conecta con *Amazon Mobile Ads*, que no encontramos en la versión crackeada.



ACTIVITY	INTENT
com.amazon.device.ads.DTBActivity	Schemes: amazonmobile://, Hosts: intent,
com.facebook.CustomTabActivity	Schemes: @string/fb_login_protocol_scheme://,
com.myfitnesspal.shared.ui.activity.impl.deeplink.DeepLinkRouterActivity	Schemes: mfp://, https://, Hosts: @string/branch_io_host, @string/branch_io_alternative_host,

Showing 1 to 3 of 3 entries

**Figura 22:** Actividades de *MyFitnessPal*.



ACTIVITY	INTENT
com.facebook.CustomTabActivity	Schemes: fb186796388009496://, fbconnect://, Hosts: cct.com.myfitnesspal.android,
com.myfitnesspal.shared.ui.activity.impl.deeplink.DeepLinkRouterActivity	Schemes: mfp://, https://, Hosts: @string/branch_io_host, @string/branch_io_alternative_host,

Showing 1 to 2 of 2 entries

**Figura 23:** Actividades de *MyFitnessPal crack*.

### 4.1.3. Receivers

No se encuentran Receivers peligrosos, pero sí diferentes: la versión crackeada utiliza en Adsbynimbus, pero no hay anuncios mostrados por pantalla al utilizarla. Aunque ambas tienen distintos Receivers sin protección de permisos.

19	<b>Broadcast Receiver</b> (com.google.firebase.iid.FirebaseInstanceIdReceiver) is Protected by a permission, but the protection level of the permission should be checked. <b>Permission:</b> 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.
16	<b>Broadcast Receiver</b> (com.inmobi.commons.core.utilities.uid.ImIdShareBroadCastReceiver) 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.
11	<b>Broadcast Receiver</b> (com.myfitnesspal.shared.provider.MPFPAppWidgetProvider) 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.
13	<b>Broadcast Receiver</b> (com.myfitnesspal.shared.receiver.MfpNotificationActionReceiver) 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.

Figura 24: Receiver de MyFitnessPal

Pero no hay nada interesante si revisamos el código, solo notificaciones.

```

public class NotificationBroadcastReceiver extends BroadcastReceiver {
    public static final String EXTRA_NOTIFICATION_ID = "notification_id";
    public static final String EXTRA_NOTIFICATION_TYPE = "notification_type";
    public static final String EXTRA_NOTIFICATION_WITH_ACTION = "notification_action";
    public static final String EXTRA_RESOURCE_ID = "notification_resource_id";
    public static final String EXTRA_URL = "notification_url";
    @Inject
    public Lazy<AnalyticsService> analyticsService;
    @Inject
    public JobServiceFactory jobServiceFactory;

    private PersistableBundle convertIntentToJobParameters(@NonNull Intent intent) {
        PersistableBundle persistableBundle = new PersistableBundle();
        persistableBundle.putString(EXTRA_NOTIFICATION_TYPE, intent.getStringExtra(EXTRA_NOTIFICATION_TYPE));
        persistableBundle.putInt("notification_id", intent.getIntExtra("notification_id", 0));
        persistableBundle.putLong(EXTRA_RESOURCE_ID, intent.getLongExtra(EXTRA_RESOURCE_ID, 0));
        persistableBundle.putString(EXTRA_URL, intent.getStringExtra(EXTRA_URL));
        return persistableBundle;
    }

    private void trackEvent(@NonNull Intent intent) {
        MapUtil.Builder builder = new MapUtil.Builder();
        String stringExtra = intent.getStringExtra(EXTRA_URL);
        if (Strings.isEmpty(stringExtra)) {
            Uri parse = Uri.parse(stringExtra);
            for (String str : UriUtils.getQueryParameterNames(parse)) {
                builder.put(str, parse.getQueryParameter(str));
            }
        }
        builder.put("utm_campaign", intent.getStringExtra("utm_campaign")).put("utm_source", "mfp").put("utm_medium", MfpNotificationHandler.ANALYTIC_VALUE_UTM_MEDIUM);
        this.analyticsService.get().reportEvent(Constants.Analytics.Events.PUSH_NOTIFICATION_OPENED, builder.build());
        if (intent.getBooleanExtra("notification_action", false)) {
            this.analyticsService.get().reportEvent(Constants.Analytics.Events.PUSH_NOTIFICATION_CTA_CLICKED, builder.build());
        }
    }
}

```

Figura 25: Código del Receiver

#### 4.1.4. ContentProvider

Podemos observar que la aplicación crackeada, contiene un ContentProvider, **com.myfitnesspal.shared.provider.MPFPAppWidgetProvider**, que no está protegido y que tampoco aparece en la aplicación original. Pero revisando los códigos, vemos que simplemente es un ContentProvider con otro nombre, aunque con una dependencia menos.



## 5. Anexos

### 5.1. Problemas de versionado

En caso de querer replicar el estudio aquí hecho, hay que tener cuidado. Muchas de las herramientas utilizadas usan *python*, además, el backend de InsecureBankV2 es un archivo python también. El problema es el siguiente: las herramientas necesitan que se use de python3 en adelante y el backend de InsecureBankv2 requiere de python2, o no se ejecutará correctamente.

También da problemas similares *Androl4b* y *Genyotion*. Recomiendo encarecidamente hacer estos tests sobre una máquina Linux montada en una máquina virtual, ya que nos dará (como norma general) mayor facilidad de uso que montar nuestro lab en Windows, al menos desde mi experiencia.

### 5.2. Lista de OWASP TOP 10 MOBILE RISKS

- M1 - Improper Platform Usage
- M2 - Insecure Data Storage
- M3 - Insecure Communication
- M4 - Insecure Authentication
- M5 - Insufficient Cryptography
- M6 - Insecure Authorization
- M7 - Client Code Quality
- M8 - Code Tampering
- M9 - Reverse Engineering
- M10 - Extraneous Functionality

### 5.3. Análisis de riesgos OWASP con Quixxi

Añado los resultados de los tests que ejecuta Quixxi para determinar las vulnerabilidades. En principio esto iba a ser una subsección de cada una de las aplicaciones, pero

al ver que la aplicación de Instagram, que en un principio es segura y conocida pasaba menos test que la aplicación InsecureBank, preparada para ser insegura, he pensado que quizá no son tests demasiado fiables.

Adjunto tablas:

#	OWASP Security Requirements	Passed	Failed
V2	Data Storage and Privacy	8	3
V3	Cryptography	8	0
V5	Network Communication	4	0
V6	Platform Interactions	8	3
V7	Code Quality and Build Settings	4	2
V8	Resilience Requirements	2	3

**Figura 28:** *Tabla resumen InsecureBank*

#	OWASP Security Requirements	Passed	Failed
V2	Data Storage and Privacy	3	8
V3	Cryptography	5	3
V5	Network Communication	2	2
V6	Platform Interactions	6	5
V7	Code Quality and Build Settings	5	1
V8	Resilience Requirements	4	1

**Figura 29:** *Tabla resumen MyFitnessPal*

#	OWASP Security Requirements	Passed	Failed
V2	Data Storage and Privacy	3	8
V3	Cryptography	4	4
V5	Network Communication	3	1
V6	Platform Interactions	6	5
V7	Code Quality and Build Settings	3	3
V8	Resilience Requirements	5	0

**Figura 30:** *Tabla resumen Instagram*