

Pràctica 7

Desarrollo de *Software* Seguro



Tabla de contenido

1. Objetivos	3
2. Preparación del entorno.....	4
3. Configuración correcta de componentes de la aplicación	5
4. Almacenamiento de credenciales	7
5. Eliminación de fugas de información a través de los logs	10
6. Permisos	11
7. Material a entregar para evaluación	13

1. Objetivos

En esta práctica vamos a solucionar las vulnerabilidades que se identificaron en la aplicación Android analizada en el tema anterior, InsecureBankv2.

Durante esta práctica nos vamos a centrar en tareas relacionadas con el código de la aplicación, ya que un S-SDLC (*Secure Software Development Life Cycle*) debe expandir sus actividades a lo largo de todas las tareas que componen el desarrollo de la aplicación. Para lograr nuestro objetivo, se van a realizar las siguientes tareas:

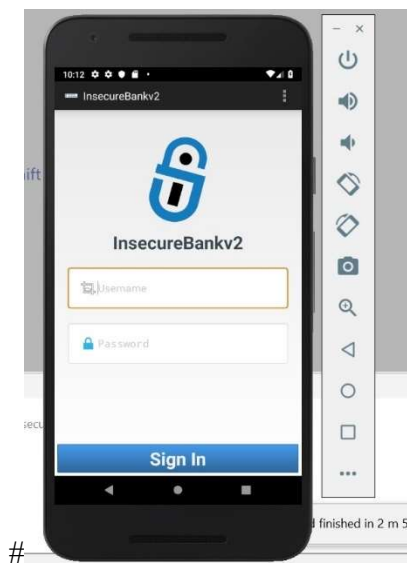
- Preparación del entorno de trabajo.
- Configuración correcta de componentes de la aplicación.
- Almacenamiento de credenciales.
- Eliminación de funcionalidad de administrador.
- Eliminación de fugas de información a través de los *logs*.
- Permisos.

2. Preparación del entorno

En primer lugar descarga el fichero `Android-InsecureBank2-master.zip` que puedes encontrar en PoliformaT -> Practicas -> P5 – Solucionar Vulnerabilidades

Para poder modificar el código de la aplicación `InsecureBankv2`, ejecutaremos el entorno de programación `Android Studio`, ejecutando desde un terminal: `ejecuta-studio`

En `Android Studio`, abriremos el proyecto `InsecureBankv2`, que se encuentra dentro del directorio `Android-InsecureBankv2`, y comprobaremos que todo se ha realizado correctamente, compilando el proyecto (en `Build->Clean Project`, y a continuación `Build->Rebuild Project`) y ejecutándolo (en `Run->Run “app”`).



Para comprobar su correcto funcionamiento pondremos en ejecución el servidor `app.py` al que debe de conectarse la aplicación móvil en la máquina virtual Kali. En primer lugar, vamos a ejecutar un script denominado `EjecutarServidor.sh` que instalará Python y todas las dependencias que necesita el servidor `app.py`

El script `EjecutarServidor.sh` se encuentra en el directorio `Android-InsecureBankv2-master/AndroLabServer/` al igual que la aplicación servidora `app.py`

Ejecútalo con la orden: `./EjecutarServidor.sh`

Al finalizar la ejecución del script veremos que el servidor está en ejecución, esperando las peticiones del cliente en el puerto 8888.

En otro terminal mediante la ejecución de la orden `ifconfig` averiguaremos la dirección IP asignada a Kali.

Con esta información configuraremos en la app `InsecureBankv2` que tenemos en ejecución en `Android` los datos del servidor. Dicha información debe introducirse en la opción *Preferences*.

3. Configuración correcta de componentes de la aplicación

En esta tarea se van a resolver todos los problemas que se detectaron en la configuración de los componentes durante el análisis de la aplicación realizado en el tema anterior.

Para ello, debemos modificar la configuración de todos los componentes de la aplicación declarados en el *manifest* para que no puedan ser utilizados por otras aplicaciones de forma maliciosa. Como **resultado** obtendremos un fichero *manifest* con la configuración de seguridad correcta para cada uno de los componentes de la aplicación.

➤ Modificaciones a realizar en el fichero *AndroidManifest.xml*:

Verifica los elementos que tienen *exported=true*.

`android:exported`

Define si la actividad puede iniciarse a través de componentes de otras aplicaciones; "true" si es posible y "false" si no lo es. Si es "false", la actividad solo se podrá iniciar a través de componentes de la misma aplicación o de aplicaciones que tengan el mismo ID de usuario.

• Actividades:

```
<activity
    android:name=".ChangePassword"
    android:exported="true"
    android:label="@string/title_activity_change_password" >
</activity>
```

```
<activity
    android:name=".ViewStatement"
    android:exported="true"
    android:label="@string/title_activity_view_statement" >
</activity>
```

```
<activity
    android:name=".DoTransfer"
    android:exported="true"
    android:label="@string/title_activity_do_transfer" >
</activity>
```

```
<activity
    android:name=".PostLogin"
    android:exported="true"
    android:label="@string/title_activity_post_login" >
</activity>
```

• Receiver:

```
<receiver
    android:name=".MyBroadCastReceiver"
    android:exported="true" >
    <intent-filter>
        <action android:name="theBroadcast" >
        </action>
    </intent-filter>
</receiver>
```

• Provider:

```
<provider
    android:name=".TrackUserContentProvider"
    android:authorities="com.android.insecurebankv2.TrackUserContentProvider"
    android:exported="true" >
</provider>
```

- Analizando la funcionalidad de las **actividades**, ninguna de ellas necesita ser accedida por otras aplicaciones por lo que se debe de eliminar el atributo `exported` de todas ellas.
- Analizando la funcionalidad del **receiver** se observa, por los comentarios y el código de este, que su objetivo es enviar un mensaje SMS con la confirmación del cambio de contraseña al usuario. Esta acción no debería ejecutarse desde el teléfono por los siguientes motivos:

- Supone un coste económico para el mismo.
- Es altamente probable que el usuario ejecute la aplicación en el mismo teléfono en el que ha configurado cuenta bancaria.
- La operación no se está realizando desde un elemento confiable del sistema (el teléfono).
- Por lo tanto, de momento, se debe de eliminar la funcionalidad de la aplicación móvil eliminando la mención al receiver del *manifest* (eliminando el código referente a receiver de AndroidManifest.xml).
- La funcionalidad del receiver se debería implementar en el servidor a través de una pasarela SMS (fuera del ámbito de esta práctica).
- También se debe eliminar el código que se encarga de ejecutar el *BroadcastIntent* una vez se ha realizado el cambio de contraseña. Este código se encuentra localizado en la Actividad *ChangePassword* (en *java->com.android.insecurebankv2->ChangePassword*), función *postData()*.

```

/*
The function that handles the SMS activity
phoneNumber: Phone number to which the confirmation SMS is to be sent
*/

broadcastChangepasswordSMS(phoneNumber, changePassword_text.getText().toString());

private void broadcastChangepasswordSMS(String phoneNumber, String pass) {

    if(TextUtils.isEmpty(phoneNumber.toString().trim())) {

        System.out.println("Phone number Invalid.");
    }
    else
    {
        Intent smsIntent = new Intent();
        smsIntent.setAction("theBroadcast");
        // String actdms = smsIntent.getAction().toString();
        // Toast.makeText(getApplicationContext(),actdms , Toast.LENGTH_LONG).show();
        smsIntent.putExtra("phoneNumber", phoneNumber);
        smsIntent.putExtra("newpass", pass);
        sendBroadcast(smsIntent);
    }
}

```

- El **provider** *TrackUserContentProvider*, como su descripción indica, se encarga de mantener un listado de los usuarios registrados en la aplicación. Sin entrar a valorar la necesidad de este tipo de *provider* en un entorno real y si la implementación es correcta (se verificará en otra tarea), se va a proteger el mismo mediante la definición de nuevos permisos para mostrar el procedimiento de protección de un *provider* mediante permisos. De esta manera las aplicaciones que quieran acceder a esta información deberán declarar alguno de los nuevos permisos.
- En primer lugar, se definen los permisos y a continuación se asignan en el *provider*.

```

<permission android:name="com.android.insecurebankv2.READ_TRACKING" />
<permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />

<uses-permission android:name="com.android.insecurebankv2.READ_TRACKING" />
<uses-permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />

<application
    android:allowBackup="true"

    . . .

    <provider
        android:name=".TrackUserContentProvider"
        android:authorities="com.android.insecurebankv2.TrackUserContentProvider"
        android:exported="true"
        android:readPermission="com.android.insecurebankv2.READ_TRACKING"
        android:writePermission="com.android.insecurebankv2.WRITE_TRACKING">
    </provider>

```

4. Almacenamiento de credenciales

En esta tarea se va a revisar el código relativo al almacenamiento de credenciales que se realiza por defecto en la actividad DoLogin. Para adecuar su seguridad, modificaremos la actividad DoLogin de forma que no almacene las credenciales del usuario de forma insegura.

Para ello abriremos el fichero DoLogin situado en *java->com.android.insecurebankv2*, y en la función *saveCreds*

```
/*
The function that saves the credentials locally for future reference
username: username entered by the user
password: password entered by the user
*/
private void saveCreds(String username, String password) throws UnsupportedEncodingException, InvalidKeyException {
    // TODO Auto-generated method stub
    SharedPreferences mySharedPreferences;
    mySharedPreferences = getSharedPreferences(MYPREFS, Activity.MODE_PRIVATE);
    SharedPreferences.Editor editor = mySharedPreferences.edit();
    rememberme_username = username;
    rememberme_password = password;
    String base64Username = new String(Base64.encodeToString(rememberme_username.getBytes(), 4));
    CryptoClass crypt = new CryptoClass();
    superSecurePassword = crypt.aesEncryptedString(rememberme_password);
    editor.putString("EncryptedUsername", base64Username);
    editor.putString("superSecurePassword", superSecurePassword);
    editor.commit();
}
```

Se puede ver que se está utilizando una clase propia para realizar el cifrado de las credenciales y luego se almacenan en un fichero de *SharedPreferences*.

Dado que las credenciales que se están almacenando son relativos a una cuenta bancaria, las credenciales no deberían ser almacenados en el dispositivo.

La primera alternativa debería ser por lo tanto la eliminación de la funcionalidad de la aplicación relativa al almacenamiento de estas credenciales, en función *postData()*.

```
if (result != null) {
    if (result.indexOf("Correct Credentials") != -1) {
        Log.d( tag: "Successful Login:", msg: "account=" + username + ":" + password);
        saveCreds(username, password);
        trackUserLogins();
        Intent pL = new Intent(getApplicationContext(), PostLogin.class);
        pL.putExtra( name: "uname", username);
        startActivity(pL);
    } else {
        Intent xi = new Intent(getApplicationContext(), WrongLogin.class);
    }
}
```

Durante la eliminación se comprueba que si el login es correcto los detalles del mismo se muestran en el *log* del dispositivo. Se procede a la eliminación de esa funcionalidad también.

```
InputStream in = responseBody.getEntity().getContent();
result = convertStreamToString( in );
result = result.replace("\n", "");
if (result != null) {
    if (result.indexOf("Correct Credentials") != -1) {
        Log.d("Successful Login:", "account=" + username + ":" + password);
        trackUserLogins();
        Intent pL = new Intent(getApplicationContext(), PostLogin.class);
        pL.putExtra("uname", username);
        startActivity(pL);
    } else {
        Intent xi = new Intent(getApplicationContext(), WrongLogin.class);
        startActivity(xi);
    }
}
```


Además, se procede también a la eliminación de los elementos del interfaz que permiten restablecer las credenciales guardadas y a los métodos que recuperan la información desde la actividad *LoginActivity* y su interfaz correspondiente.

- Botón para el rellenado de datos:

```
// The Button that allows the user to autofill the credentials,
// if the user has logged in successfully earlier
Button fillData_button;
```

- Inicialización desde:

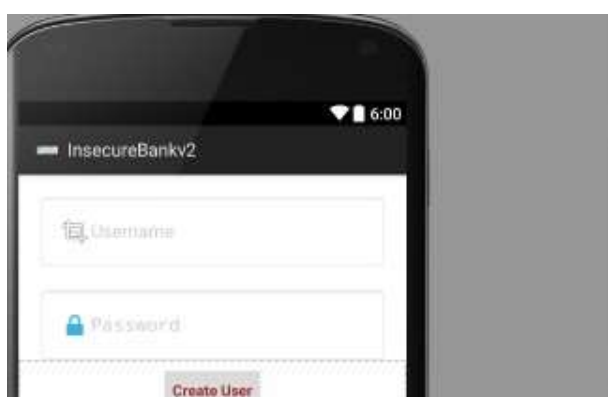
```
try {
    fillData();
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
} catch (InvalidKeyException e) {
    e.printStackTrace();
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
} catch (NoSuchPaddingException e) {
    e.printStackTrace();
} catch (InvalidAlgorithmParameterException e) {
    e.printStackTrace();
} catch (IllegalBlockSizeException e) {
    e.printStackTrace();
} catch (BadPaddingException e) {
    e.printStackTrace();
}
```

- Método fillData():

```
/*
The function that allows the user to autofill the credentials
if the user has logged in successfully atleast one earlier using
that device
*/
protected void fillData() throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException {
    // TODO Auto-generated method stub
    SharedPreferences settings = getSharedPreferences(MYPREFS, mode: 0);
    final String username = settings.getString( S: "EncryptedUsername", s1: null);
    final String password = settings.getString( S: "superSecurePassword", s1: null);

    if(username!=null && password!=null)
    {
        byte[] usernameBase64Byte = Base64.decode(username, Base64.DEFAULT);
        try {
            usernameBase64ByteString = new String(usernameBase64Byte, charsetName: "UTF-8");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
}
```

- Además, deberemos eliminar el elemento correspondiente del Layout *res/activity_log_main.xml*



Lo primero es alinear el apk sirviéndonos de *zipalign*. Esta aplicación también forma parte de las build-tools de la SDK Android que estamos utilizando y garantiza que todos los datos descomprimidos de la apk comiencen con una alineación de bytes en particular relacionada en relación con el comienzo del archivo, lo que reduce significativamente el consumo de memoria RAM de una aplicación. Se recomienda una alineación de 4 bytes. Para realizar la alineación procedemos como sigue:

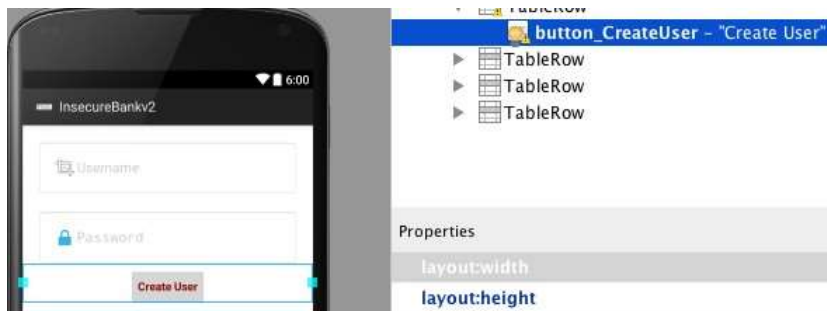
- Para incrementar la seguridad de esos datos se modifica el directorio en el que se guardan los ficheros a la *sandbox* de la aplicación.
- Sustituyendo el método `getExternalStorageDirectory()` por `getDataDirectory()`
- En la actividad *DoTransfer*:

- En la actividad *ViewState*:

Existe un botón que no se muestra por defecto en la aplicación y que permite la creación de usuarios dentro de la aplicación. Por ello, se eliminará toda la funcionalidad oculta de la actividad de login en la aplicación móvil que pueda permitir a atacantes, abusar del

servicio mediante procesos de ingeniería inversa, consiguiéndose un código de la aplicación móvil sin funcionalidades ocultas en la actividad de Login.

En el fichero `activity_log_main.xml` de *layout* se debe eliminar el botón añadido:



Aunque comprobando el código no existe ninguna funcionalidad especial añadida, se elimina el código relativo al botón en el fichero `LoginActivity.java`:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_log_main);
    String mess = "no";
    if (mess.equals("no")) {
        View button_CreateUser = findViewById(R.id.button_CreateUser);
        button_CreateUser.setVisibility(View.GONE);
    }
    login_buttons = (Button) findViewById(R.id.login_button);
    login_buttons.setOnClickListener((v) -> {
        // TODO Auto-generated method stub
        performlogin();
    });
    createuser_buttons = (Button) findViewById(R.id.button_CreateUser);
    createuser_buttons.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            createUser();
        }
    });
};
```

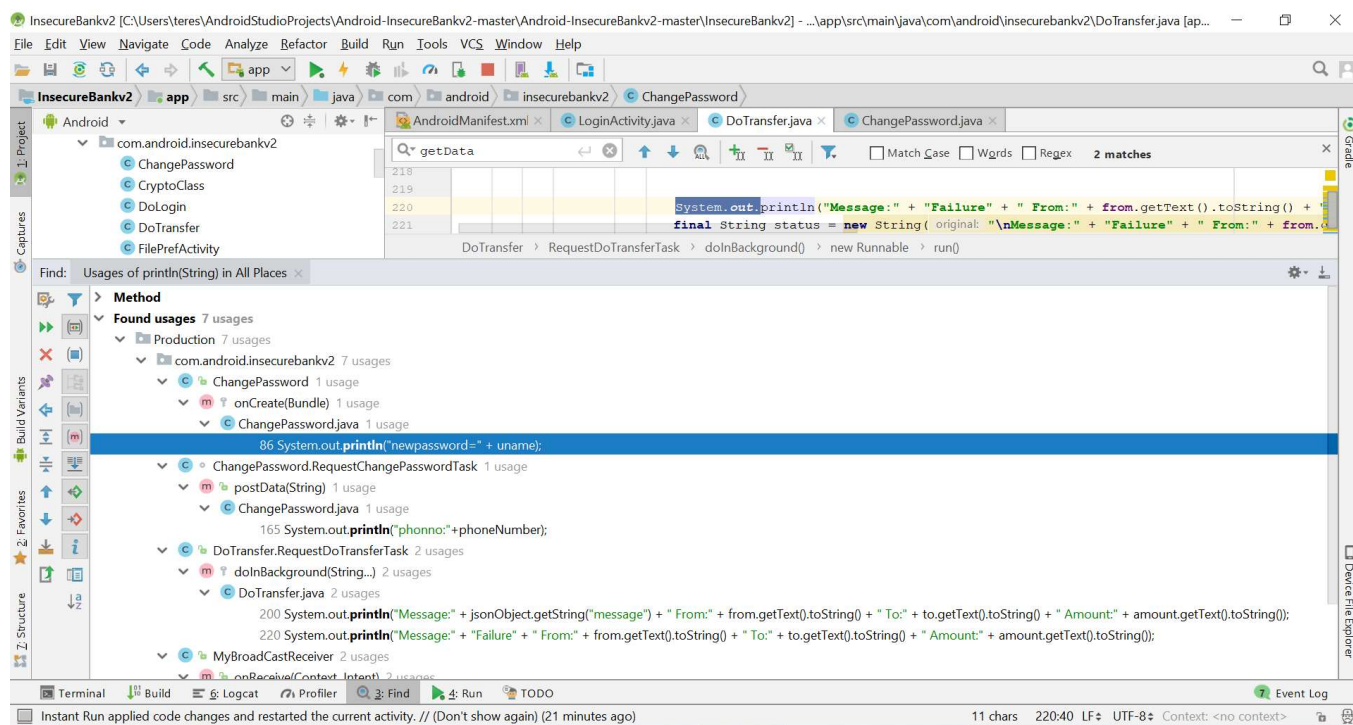
5. Eliminación de fugas de información a través de los logs

En este punto de la práctica se van a eliminar las posibles fugas de información que se produzcan por la consola del dispositivo durante la ejecución de la aplicación, consiguiéndose así un código de la aplicación sin llamadas a los *logs* o la consola con información que pueda ser considerada como sensible.

En primer lugar, procedemos a buscar todos los usos de `System.out` en la aplicación mediante la opción `Find`.

Para ellos en `Find-> Find in Path` buscaremos `System.out` y abriremos donde aparece nuestra búsqueda, y seleccionaremos el texto a buscar: `System.out`

Utilizando la función `FindUsage` que se encuentra en `Edit->Find->FindUsages`, encontraremos las demás ocurrencias de nuestra búsqueda



Analizaremos los resultados obtenidos y eliminaremos aquellos que muestran el password.

De la misma manera se realiza una búsqueda de los usos de la clase `Log` dentro de la aplicación por si hubiese alguna escritura de datos sensibles en los ficheros `Log`.

6. Permisos

Siguiendo el principio de menor número de privilegios, en esta tarea se van a eliminar aquellos permisos que la aplicación no requiera para su ejecución.

Para ello analizaremos el uso de cada uno de los permisos en la aplicación y eliminaremos aquellos que no son necesarios para el correcto funcionamiento de la misma. Esto nos permitirá obtener un *manifest* actualizado con la lista de permisos estrictamente necesaria para la ejecución de la aplicación.

Inicialmente en el fichero `AndroidManifest.xml` podemos ver los permisos que utiliza la aplicación

```

AndroidManifest.xml x
7      android:targetSdkVersion="24" />
8
9      <uses-permission android:name="android.permission.INTERNET" />
10     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
11     <uses-permission android:name="android.permission.SEND_SMS" />
12     <!--
13     To retrieve OAuth 2.0 tokens or invalidate tokens to disconnect a user. This disconnect
14     option is required to comply with the Google+ Sign-In developer policies
15     -->
16     <uses-permission android:name="android.permission.USE_CREDENTIALS" /> <!-- To retrieve the account name (email)
17     <uses-permission android:name="android.permission.GET_ACCOUNTS" /> <!-- To auto-complete the email text field
18     <uses-permission android:name="android.permission.READ_PROFILE" />
19     <uses-permission android:name="android.permission.READ_CONTACTS" />
20
21     <android:uses-permission android:name="android.permission.READ_PHONE_STATE" />
22     <android:uses-permission
23         android:name="android.permission.READ_EXTERNAL_STORAGE"
24         android:maxSdkVersion="18" />
25     <android:uses-permission android:name="android.permission.READ_CALL_LOG" />
26
27     <permission android:name="com.android.insecurebankv2.READ_TRACKING" />
28     <permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />
29
30     <uses-permission android:name="com.android.insecurebankv2.READ_TRACKING" />
31     <uses-permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />
32
33     <application
manifest > application

```

- El permiso de Internet es necesario para que la aplicación se pueda conectar al *back-end* del banco por lo que se mantiene.
- El almacenamiento externo era utilizado para almacenar los movimientos. Al haber modificado su lugar de almacenamiento se pueden eliminar los permisos de lectura y escritura de la tarjeta SD.
- El permiso de envío de SMS ya no es necesario, pues esas operaciones se deberían realizar desde una pasarela en el *back-end*.
- La aplicación en ningún momento necesita acceder a las credenciales del dispositivo. En su configuración actual las credenciales no son almacenados, y en el caso de que lo fuesen, se trataría credenciales propias de la aplicación y no se necesitaría solicitar ningún permiso adicional para su acceso. Se procede a eliminar los permisos GET_ACCOUNTS y USE_CREDENTIALS.
- Los permisos relativos al teléfono y el historial de llamadas se necesitan para que la aplicación sepa el número de teléfono al que mandarle el SMS una vez se ha modificado la contraseña. En la actividad *ChangePassword* se puede observar el siguiente código:

```

TelephonyManager phoneManager = (TelephonyManager)getApplicationContext().getSystemService(Context.TELEPHONY_SERVICE);
String phoneNumber = phoneManager.getLine1Number();
System.out.println("phonno:"+phoneNumber);

```

- Debido a que ya no son necesarios, se elimina junto a los permisos asociados:

```

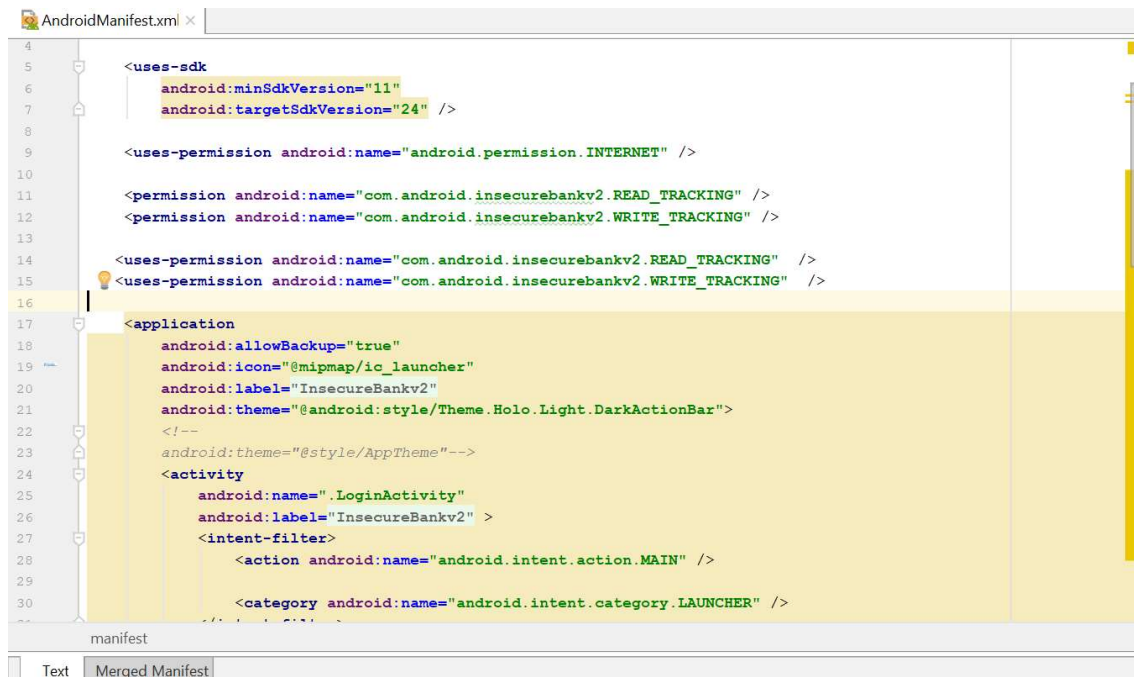
<uses-permission android:name="android.permission.READ_PROFILE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />

<android:uses-permission android:name="android.permission.READ_PHONE_STATE" />

```

- Se puede comprobar también que hay permisos mal definidos en la aplicación (empiezan en android). Se procede también a su eliminación.

- El permiso de lectura de contactos no es necesario, pues la aplicación no necesita acceder a los mismos.
- El único permiso que queda de los que inicialmente tenía la aplicación en el *manifest* es el permiso de INTERNET.



```
4
5 <uses-sdk
6     android:minSdkVersion="11"
7     android:targetSdkVersion="24" />
8
9 <uses-permission android:name="android.permission.INTERNET" />
10
11 <permission android:name="com.android.insecurebankv2.READ_TRACKING" />
12 <permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />
13
14 <uses-permission android:name="com.android.insecurebankv2.READ_TRACKING" />
15 <uses-permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />
16
17 <application
18     android:allowBackup="true"
19     android:icon="@mipmap/ic_launcher"
20     android:label="InsecureBankv2"
21     android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
22     <!--
23     android:theme="@style/AppTheme"-->
24     <activity
25         android:name=".LoginActivity"
26         android:label="InsecureBankv2" >
27         <intent-filter>
28             <action android:name="android.intent.action.MAIN" />
29
30             <category android:name="android.intent.category.LAUNCHER" />
31         </intent-filter>
32     </activity>
33 </application>
```

7. Material a entregar para evaluación

En las prácticas anteriores, mediante el análisis estático y dinámico se detectaron las vulnerabilidades que hemos corregido en los apartados anteriores.

Realiza un listado con las vulnerabilidades que has detectado en InsecureBankv2 en las prácticas anteriores.

Si hay alguna(s) vulnerabilidad(es) que hayas detectado y no se haya(n) mencionado anteriormente, indícala(s). Además, indica cual sería la forma de resolverla.

Sube este documento a tu espacio compartido para su evaluación por parte de profesor.