

INFORME AUDITORÍA MÓVIL

AndroidApplication.apk

Contenido

1. Objetivos y alcance	5
1.1 RESOLUCIÓN DEL CASO PRÁCTICO	5
1.1.1 Obtención del código fuente de la aplicación. Detallar todo el proceso con capturas. 8	
1.1.2 Análisis del AndroidManifest (permisos y que vulnerabilidades presenta)	9
1.1.3 Análisis del código fuente de la aplicación en busca de vulnerabilidades (dos vulnerabilidades como mucho).....	13
1.1.4 Análisis de aplicación móvil en dinámico	13
1.1.4.1 Interceptación del tráfico entre aplicación y servidor.	15
1.1.4.2 Vulnerabilidades de depuración de la aplicación.	17
1.1.4.3 Almacenamiento inseguro de datos.	21
1.1.4.4 Bypass de autenticación.	22
1.1.5 Bypass de detección de root.	22
1.1.6 Bypass de detección de emulador	22
2. Resumen ejecutivo	26
3. Detalle técnico de las vulnerabilidades	28
3.1. ID_referencia del título de la vulnerabilidad	28
 Figura 1	5
Figura 2	5
Figura 3	5
Figura 4	6
Figura 5	7
Figura 6	7
Figura 7	7

Figura 8	8
Figura 9	8
Figura 10	9
Figura 11	13
Figura 12	13
Figura 13	14
Figura 14	15
Figura 15	15
Figura 16	16
Figura 17	16
Figura 18	17
Figura 19	18
Figura 20	18
Figura 21	18
Figura 22	19
Figura 23	19
Figura 24	19
Figura 25	20
Figura 26	20
Figura 27	20
Figura 28	21
Figura 29	21
Figura 30	22
Figura 31	22
Figura 32	22
Figura 33	23
Figura 34	23
Figura 35	23
Figura 36	24

Figura 37	24
Figura 38	25
Figura 39	25
Figura 40	26

1. Objetivos y alcance

1.1 RESOLUCIÓN DEL CASO PRÁCTICO

Para poder realizar esta práctica se han instalado una serie de herramientas para poder hacer tanto el análisis estático como el análisis dinámico. A continuación, aparecen descritas las aplicaciones instaladas.

- Android Debug Bridge (ADB)

- Se trata de una herramienta multiplataforma que permite la comunicación remota de un emulador o dispositivo Android conectado.

```
PS C:\Users\David\pentest> adb devices
List of devices attached
emulator-5554    device
```

Figura 1: Listado de dispositivos

- Android Studio

- Android Studio es el entorno de desarrollo integrado (IDE) oficial para crear aplicaciones Android. Android Studio proporciona herramientas especializadas y una interfaz diseñada específicamente para el desarrollo de aplicaciones en el ecosistema Android.

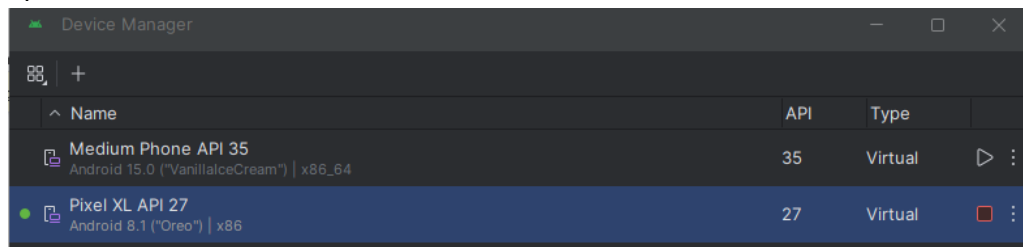


Figura 2: Android Studio

- Burpsuite Community

- Burp Suite Community Edition es una herramienta gratuita desarrollada por PortSwigger para realizar pruebas de seguridad en aplicaciones web. En este caso nos servirá de proxy para que el dispositivo virtual se pueda conectar a internet, así como para poder interceptar las peticiones que haga el dispositivo via HTTP/HTTPS.

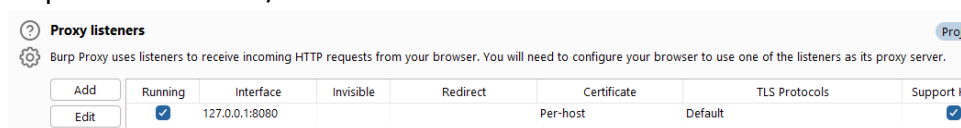


Figura 3: Bursuite Community



Figura 4: Configuración proxy Android

- **OpenJDK**
 - OpenJDK proporciona un entorno de desarrollo y tiempo de ejecución para crear y ejecutar aplicaciones y servicios escritos en el lenguaje de programación Java.
- **Jadx**
 - JADX es una herramienta de código abierto utilizada para descompilar y analizar archivos APK de Android. Convierte el código bytecode (Dalvik o ART) contenido en los archivos “.dex” de un APK en un código fuente más legible en Java o Kotlin, lo que facilita la comprensión y análisis de aplicaciones Android.
- **Apktool**
 - APKTool es una herramienta de código abierto que permite realizar ingeniería inversa de aplicaciones Android empaquetadas en archivos APK. Es ampliamente utilizada para descompilar aplicaciones, modificar recursos, realizar pruebas de seguridad y aprender sobre la estructura de las apps.
- **Máquina virtual de Linux**
 - En ella se desplegará la aplicación indicada en la práctica y se configurará el dispositivo Android para que se conecte a dicha máquina virtual.

```
(osboxes@osboxes)-[~/Modulo_6_master/Examen_practico/AppServer]
$ python app.py
The server is hosted on port: 8888
lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Figura 5: Servidor de la aplicación

```
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
link/ether 08:00:27:9f:e5:6f brd ff:ff:ff:ff:ff:ff
inet 192.168.1.77/24 brd 192.168.1.255 scope global dynamic noprefixroute eth1
    valid_lft 40525sec preferred_lft 40525sec
inet6 fe80::a00:27ff:fe9f:e56f/64 scope link noprefixroute
```

Figura 6: Configuración de red

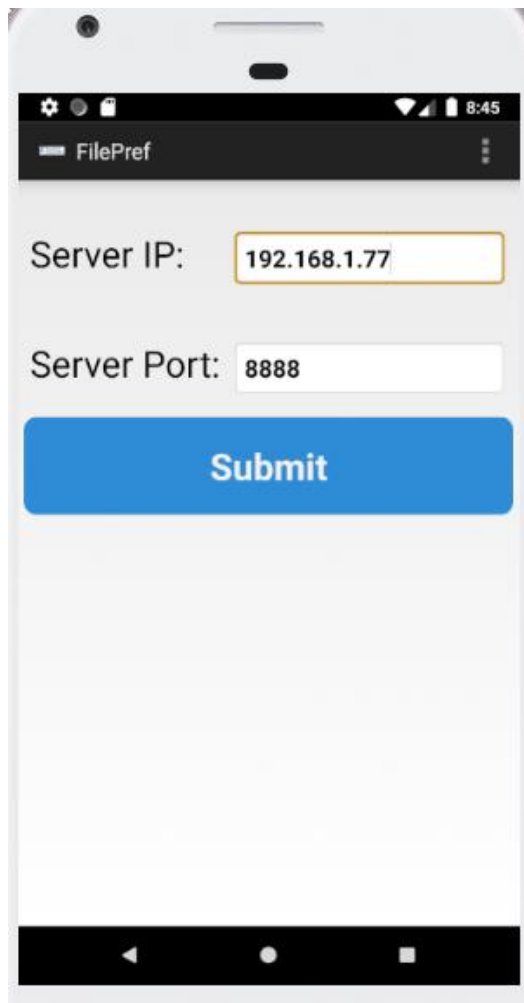


Figura 7: Configuración servidor AndroidApplication

A continuación, se resolverá el caso práctico.

1.1.1 Obtención del código fuente de la aplicación. Detallar todo el proceso con capturas.

Para obtener el código fuente de la aplicación se hará uso de la apktool que permite además de descomprimir el archivo APK, permitirá obtener el código SMALI (lenguaje similar a ensamblador).

A continuación, se puede observar el proceso para obtener el código fuente de la aplicación.

```
PS C:\Users\David\pentest> java -jar .\apktool.jar d .\AndroidApplication.apk
I: Using Apktool 2.7.0 on AndroidApplication.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\David\AppData\Local\apktool\framework\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...
```

Figura 8: Uso herramienta apktool

```
PS C:\Users\David\pentest> dir .\AndroidApplication\

Directorio: C:\Users\David\pentest\AndroidApplication

Mode                LastWriteTime         Length Name
----                -
d-----         18/01/2025   14:50             original
d-----         18/01/2025   14:50              res
d-----         18/01/2025   14:50             smali
-a-----         18/01/2025   14:50        4043 AndroidManifest.xml
-a-----         18/01/2025   14:50        442 apktool.yml
```

Figura 9: Contenido AndroidApplication

Una vez se han descomprimido los archivos de la APK, se pueden observar los archivos de configuración, el “AndroidManifest.xml”, pero no el código fuente en java.

Para obtener el código fuente, se realizará una conversión de los archivos binarios con extensión “.dex”, conocidos como Dalvik executable, con una estructura similar al código ensamblador.

Para realizar este procedimiento se utilizará la herramienta JADX.

Seleccionando la APK proporcionada en el enunciado, se obtiene lo siguiente:

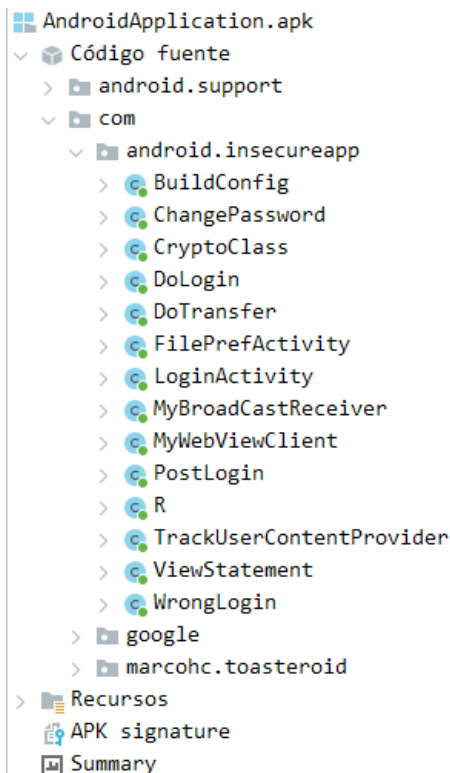


Figura 10: Contenido AndroidApplication

En primer lugar, una vez descomprimido el APK y habiendo obtenido su código fuente se analizará el AndroidManifest.xml. Dicho fichero proporciona información importante al sistema operativo Android sobre la aplicación que se está desarrollando. Define la estructura básica, permisos y componentes de la app, actuando como un puente entre el sistema y la aplicación.

1.1.2 Análisis del AndroidManifest (permisos y que vulnerabilidades presenta)

En primer lugar, se revisarán los permisos de la aplicación detallados en el AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.android.insecureapp">

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

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

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

```
<uses-permission android:name="android.permission.USE_CREDENTIALS"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.READ_PROFILE"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:maxSdkVersion="18"
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-feature android:glEsVersion="0x00020000" android:required="true"/>
<application android:allowBackup="true" android:debuggable="true"
android:icon="@mipmap/ic_launcher" android:label="@string/app_name"
android:theme="@android:style/Theme.Holo.Light.DarkActionBar">

    <activity android:label="@string/app_name"
android:name="com.android.insecureapp.LoginActivity">

        <intent-filter>

            <action android:name="android.intent.action.MAIN"/>

            <category android:name="android.intent.category.LAUNCHER"/>

        </intent-filter>

    </activity>

    <activity android:label="@string/title_activity_file_pref"
android:name="com.android.insecureapp.FilePrefActivity"
android:windowSoftInputMode="adjustNothing|stateVisible"/>

    <activity android:label="@string/title_activity_do_login"
android:name="com.android.insecureapp.DoLogin"/>

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

    <activity android:label="@string/title_activity_wrong_login"
android:name="com.android.insecureapp.WrongLogin"/>
```

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

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

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

<receiver android:exported="true"
android:name="com.android.insecureapp.MyBroadcastReceiver">

    <intent-filter>

        <action android:name="theBroadcast"/>

    </intent-filter>

</receiver>

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

<activity
android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|screenSiz
e|smallestScreenSize|uiMode" android:name="com.google.android.gms.ads.AdActivity"
android:theme="@android:style/Theme.Translucent"/>

<activity
android:name="com.google.android.gms.ads.purchase.InAppPurchaseActivity"
android:theme="@style/Theme.IAPTheme"/>

<meta-data android:name="com.google.android.gms.wallet.api.enabled"
android:value="true"/>

<receiver android:exported="false"
android:name="com.google.android.gms.wallet.EnableWalletOptimizationReceiver">

    <intent-filter>

        <action
android:name="com.google.android.gms.wallet.ENABLE_WALLET_OPTIMIZATION"/>

    </intent-filter>

</receiver>
```

```
<meta-data android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version"/>

</application>

</manifest>
```

A primera vista se pueden detectar dos vulnerabilidades, la aplicación cuenta con el atributo `android:debuggable="true"`, lo que permite que muestre información sensible de los logs durante su ejecución. A su vez, cuenta con el atributo `android:allowBackup="true"`, lo que permite que se pueda realizar un backup de la aplicación y volcarlo en un dispositivo externo.

Además, el `BroadcastReceiver` está exportado, lo que permite que aplicaciones externas envíen broadcasts para interactuar con él, permitiendo que aplicaciones maliciosas puedan enviar broadcasts falsos para enviar funcionalidades no autorizadas.

Además, tiene otra serie de permisos que puedes ser explotados:

- `android.permission.WRITE_EXTERNAL_STORAGE`: permite escribir en el almacenamiento externo, si no se valida la entrada del usuario o los archivos creados, se puede utilizar para introducir código malicioso.
- `android.permission.SEND_SMS`: permite enviar mensajes SMS, lo que podría usarse para enviar SMS sin que el usuario sea consciente, suscribiéndose a servicios de pago.
- `android.permission.USE_CREDENTIALS` y `android.permission.GET_ACCOUNTS`: permiten acceder a cuentas en el dispositivo, lo que puede resultar en la fuga de información sensible.
- `android.permission.READ_CALL_LOG`: la aplicación puede acceder a las llamadas realizadas y al contestador telefónico, realizar llamadas telefónicas, etc..
- `android.permission.READ_CONTACTS` y `android.permission.READ_PROFILE`: permite leer información de los contactos y el perfil del usuario, lo que podría usarse para realizar phishing o ingeniería social.
- `android.permission.ACCESS_COARSE_LOCATION`: permite acceder a la ubicación aproximada del dispositivo.

1.1.3 Análisis del código fuente de la aplicación en busca de vulnerabilidades (dos vulnerabilidades como mucho).

Para analizar las vulnerabilidades del código fuente de la aplicación, como se ha indicado anteriormente en la FIGURA 10 , se hará uso de jadx.

Haciendo un análisis del código fuente, se encuentran las siguientes vulnerabilidades:

- En la clase “DoLogin”, en el método SaveCreds FIGURA 11 , se almacenan los datos del usuario (usuario y contraseña) en SharedPreferences. Los datos en SharedPreferences se almacenan en texto plano en un archivo XML en el directorio que se indica en la FIGURA 12. Si un atacante accede al dispositivo o al sistema de archivos de la aplicación, podría leer estos datos sensibles.

```
private void saveCreds(String username, String password) throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException {
    SharedPreferences mySharedPreferences = DoLogin.this.getSharedPreferences("mySharedPreferences", 0);
    SharedPreferences.Editor editor = mySharedPreferences.edit();
    DoLogin doLogin = DoLogin.this;
    doLogin.rememberme_username = username;
    doLogin.rememberme_password = password;
    String base64Username = new String(Base64.encodeToString(doLogin.rememberme_username.getBytes(), 4));
    CryptoClass crypt = new CryptoClass();
    DoLogin doLogin2 = DoLogin.this;
    doLogin2.superSecurePassword = crypt.aesEncryptedString(doLogin2.rememberme_password);
    editor.putString("encryptedUsername", base64Username);
    editor.putString("superSecurePassword", DoLogin.this.superSecurePassword);
    editor.commit();
}
```

Figura 11: Función saveCreds

```
generic_x86:/data/data/com.android.insecureapp/shared_prefs # ls -latr
total 16
drwx----- 6 u0_a79 u0_a79 4096 2025-01-18 07:13 ..
-rw-rw---- 1 u0_a79 u0_a79 164 2025-01-18 07:14 com.android.insecureapp_preferences.xml
drwxrwx--x 2 u0_a79 u0_a79 4096 2025-01-18 07:14 .
generic_x86:/data/data/com.android.insecureapp/shared_prefs #
```

Figura 12: Fichero SharedPreferences

- En la clase DoLogin además como se indica en la FIGURA 11, se codifica en Base64 el nombre del usuario, el cual no es un método seguro de cifrado, si es necesario almacenar el nombre del usuario, se debería de usar un esquema seguro como AES y almacenar la clave en la Android Keystore.

1.1.4 Análisis de aplicación móvil en dinámico

Para realizar el análisis dinámico de la aplicación se utilizará Burpsuite para interceptar el tráfico HTTP/HTTPS como se indica en la FIGURA 3 y en la FIGURA 4.

Para poder realizar esto, se ha instalado el certificado de Burpsuite en el dispositivo Android. Para ello se han realizado los siguientes pasos:

- Dentro de la herramienta BurpSuite el menú “Proxy ” y posteriormente seleccionando “Proxy settings”, se exportará el certificado haciendo click en el botón “Import / export CA certificate”.

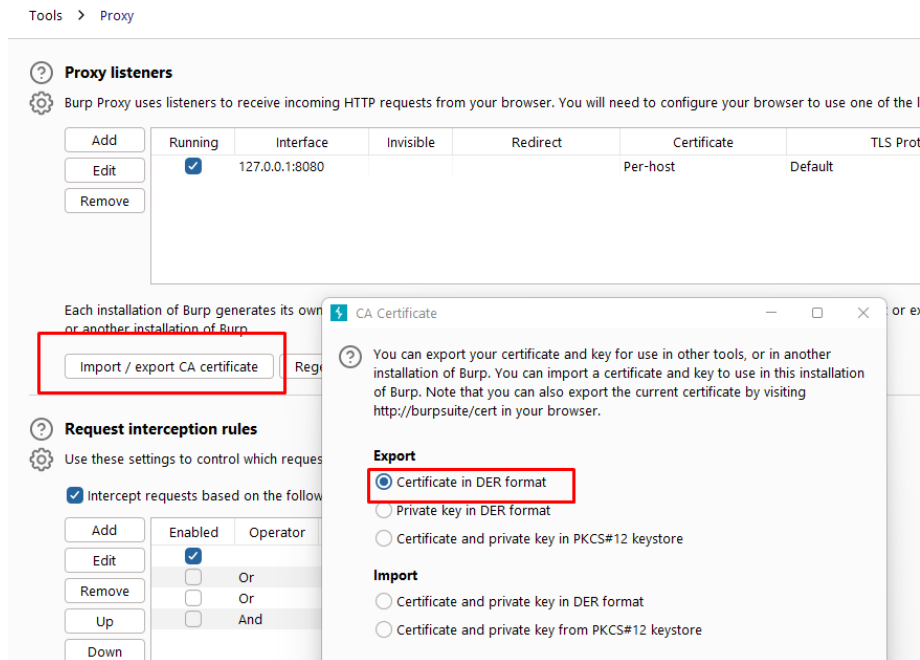


Figura 13

Una vez obtenido el certificado de la CA de Burpsuite, copiarlo a la máquina virtual de Kali Linux y se han ejecutado los siguientes comandos para convertir el certificado a formato PEM y asignarle un nombre para que pueda ser instalado dentro del dispositivo.

- openssl x509 -inform DER -in cacert.der -out cacert.pem
- openssl x509 -inform PEM -subject_hash_old -in cacert.pem | head -1
- mv cacert.pem <hash>.0

Una vez realizada la conexión se copia el certificado resultante al ordenador local y se arranca el emulador de Android en modo “system writable” con los siguientes comandos en el cmd:

- emulator -list-avds
- emulator -avd Pixel_XL_API_27 -writable-system

En otra pestaña del cmd se ejecutan los siguientes comandos para copiar el certificado al dispositivo Android.

- adb root
- adb remount
- adb push 9a5ba575.0 /sdcard

Posteriormente, desde el cmd de Windows se abre una Shell para conectarse al dispositivo Android:

- abd Shell
- cp /sdcard/9a5ba575.0 /system/etc/security/cacerts/

- `chmod 644 /system/etc/security/cacerts/9a5ba575.0`
- `touch /system/app/Superuser.apk`
- `reboot`

Una vez reiniciado el dispositivo, el certificado de Burp estará instalado, se puede comprobar accediendo a al menú “Trusted Credentials”:

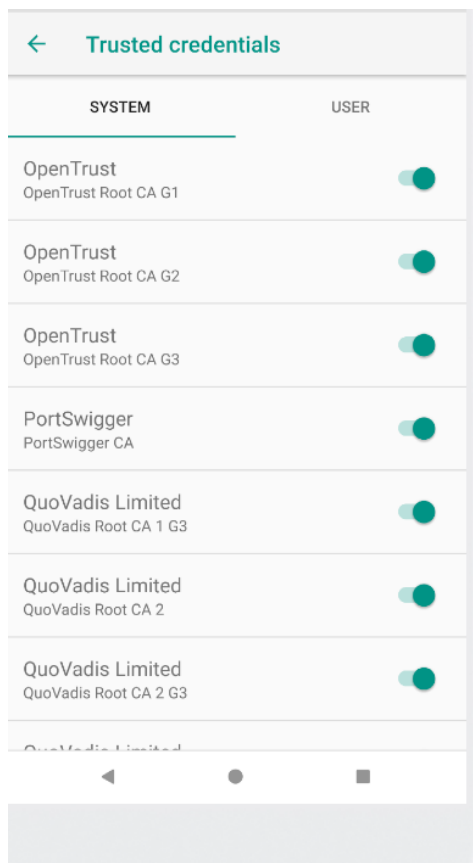


Figura 14: Listado certificados emulador

1.1.4.1 Interceptación del tráfico entre aplicación y servidor.

En primer lugar, se interceptará el tráfico entre el cliente y el servidor como se ha indicado anteriormente con Burpsuite.

El servidor se alojará en un entorno virtual en Kali Linux. Para ello se ejecutará el servidor mediante el comando indicado en el enunciado de la práctica:

```
(osboxes@osboxes) - [~/Modulo_6_master/Examen_practico/AppServer]
$ python3 app.py
The server is hosted on port: 8888
```

Figura 15: Servidor AndroidApplication

Como se indica en el enunciado se introducen las credenciales de ambos usuarios, interceptando el tráfico entre el dispositivo Android y el servidor.

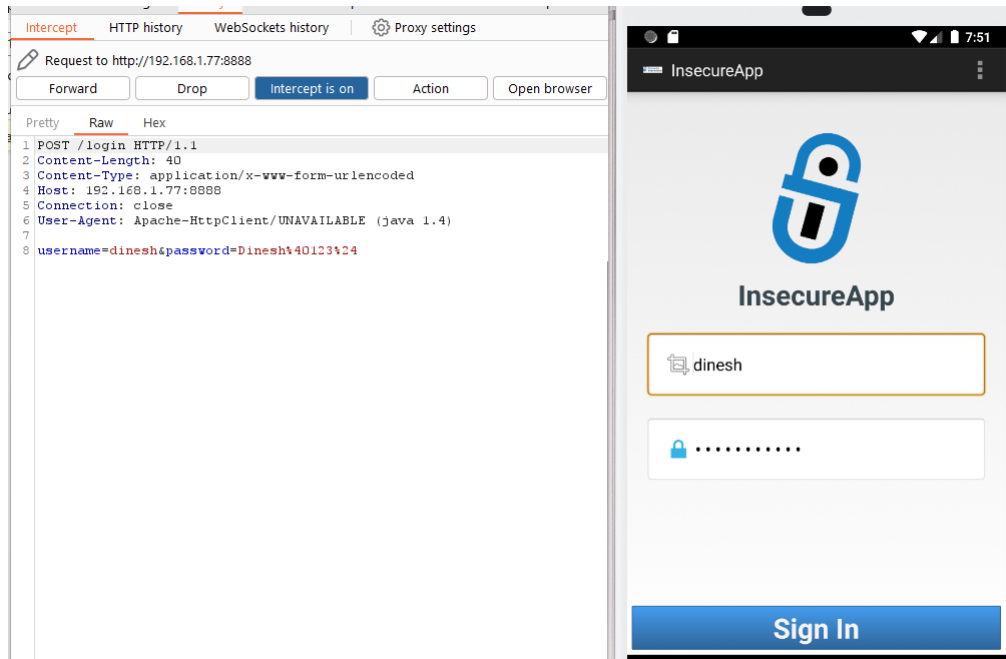


Figura 16: Login Dinesh

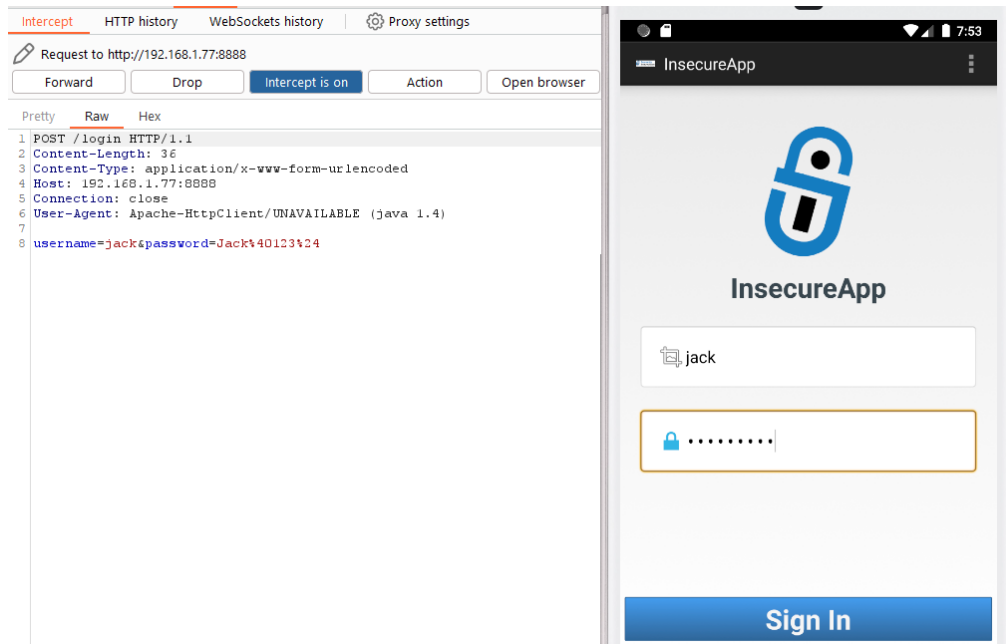


Figura 17: Login jack

Tras loguearse, aparece la siguiente pantalla principal:

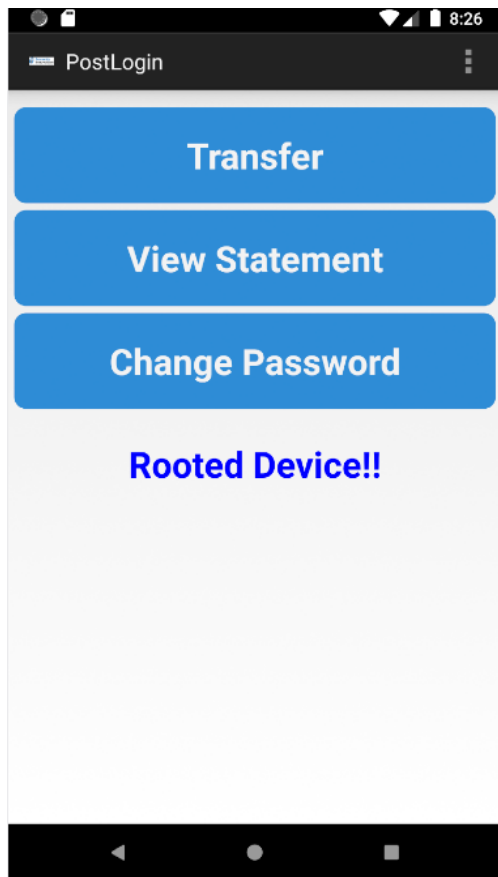


Figura 18: Pantalla principal AndroidApplication

1.1.4.2 Vulnerabilidades de depuración de la aplicación.

Para llevar a cabo el análisis dinámico de la aplicación se accederá a las diferentes vistas de la aplicación con el fin de encontrar posibles vulnerabilidades.

Se realizará una conexión por ADB al dispositivo emulado seguido de la ejecución del comando logcat para verificar la información que muestran los logs de la aplicación.

Como se muestra en la siguiente figura, se muestran los logs de la aplicación cuando se interactúa con la aplicación. Cuando se hace click en cualquiera de las opciones de la ventana principal, se generan logs relativos a la aplicación.

```
$ adb logcat | grep com.android.insecureapp
01-19 20:43:43.079 1652 2412 I ActivityManager: START u0 {cmp=com.android.insecureapp/.DoTransfer} from uid 10079
01-19 20:43:43.267 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.DoTransfer: +178ms
01-19 20:43:46.146 1652 2413 I ActivityManager: START u0 {cmp=com.android.insecureapp/.ViewStatement (has extras)} from uid 10079
01-19 20:43:46.203 1652 2413 I ActivityManager: START u0 {cmp=com.android.insecureapp/.PostLogin} from uid 10079
01-19 20:43:46.533 1450 1450 D SurfaceFlinger: duplicate layer name: changing com.android.insecureapp/com.android.insecureapp.PostLogin to com.androi
01-19 20:43:46.665 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.PostLogin: +415ms (total +488ms)
01-19 20:43:47.683 1652 2413 I ActivityManager: START u0 {cmp=com.android.insecureapp/.ChangePassword (has extras)} from uid 10079
01-19 20:43:47.964 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.ChangePassword: +271ms
01-20 19:30:57.361 1652 3906 I ActivityManager: START u0 {cmp=com.android.insecureapp/.DoTransfer} from uid 10079
01-20 19:30:57.859 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.DoTransfer: +473ms
01-20 19:31:19.020 1652 3598 I ActivityManager: START u0 {cmp=com.android.insecureapp/.ViewStatement (has extras)} from uid 10079
01-20 19:31:19.155 1652 1676 I ActivityManager: START u0 {cmp=com.android.insecureapp/.PostLogin} from uid 10079
01-20 19:31:19.308 1450 1450 D SurfaceFlinger: duplicate layer name: changing com.android.insecureapp/com.android.insecureapp.PostLogin to com.androi
01-20 19:31:19.492 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.PostLogin: +300ms (total +445ms)
01-20 19:31:22.792 1652 1676 I ActivityManager: START u0 {cmp=com.android.insecureapp/.ChangePassword (has extras)} from uid 10079
01-20 19:31:22.943 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.ChangePassword: +140ms
01-20 19:31:33.348 1652 1676 I ActivityManager: START u0 {flag=0x40000000 cmp=com.android.insecureapp/.LoginActivity} from uid 10079
01-20 19:31:33.834 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.LoginActivity: +371ms
01-20 19:31:35.404 1652 3598 I ActivityManager: START u0 {cmp=com.android.insecureapp/.DoLogin (has extras)} from uid 10079
01-20 19:31:35.542 1652 3906 I ActivityManager: START u0 {cmp=com.android.insecureapp/.PostLogin (has extras)} from uid 10079
01-20 19:31:35.542 1652 3906 W ActivityManager: startActivity called from non-Activity context; forcing Intent.FLAG_ACTIVITY_NEW_TASK for: Intent { c
mp=com.android.insecureapp/.PostLogin (has extras) }
01-20 19:31:35.695 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.PostLogin: +132ms (total +280ms)
01-20 19:31:39.242 1652 3598 I ActivityManager: START u0 {cmp=com.android.insecureapp/.DoTransfer} from uid 10079
01-20 19:31:39.398 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.DoTransfer: +134ms
01-20 19:31:39.984 1652 1676 I ActivityManager: START u0 {cmp=com.android.insecureapp/.DoTransfer} from uid 10079
01-20 19:31:40.100 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.DoTransfer: +108ms
01-20 19:31:43.342 1652 2412 I ActivityManager: START u0 {cmp=com.android.insecureapp/.ViewStatement (has extras)} from uid 10079
01-20 19:31:43.384 1652 2412 I ActivityManager: START u0 {cmp=com.android.insecureapp/.PostLogin} from uid 10079
01-20 19:31:43.551 1450 1450 D SurfaceFlinger: duplicate layer name: changing com.android.insecureapp/com.android.insecureapp.PostLogin to com.androi
01-20 19:31:43.934 1652 1674 I ActivityManager: Displayed com.android.insecureapp/.PostLogin: +525ms (total +577ms)
```

Figura 19: Logs de la aplicación

Además, como se sabe que el usuario está encriptado en base64 y que se guarda en el directorio “SharedPreferences” se puede obtener el super usuario de la aplicación como se muestra en la siguiente figura.

```
generic_x86:/data/data/com.android.insecureapp/shared_prefs # cat mySharedPreferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="superSecurePassword">v/sJpihDCo2ckDmLwSUwiw==&#10; </string>
  <string name="EncryptedUsername">amFjaw==&#13;&#10; </string>
</map>
generic_x86:/data/data/com.android.insecureapp/shared_prefs # echo "amFjaw==&#13;&#10;" | base64 -d && echo
jack
generic_x86:/data/data/com.android.insecureapp/shared_prefs #
```

Figura 20: Archivo mySharedPreferences

Accediendo a la vista principal de login y registrándose como el usuario devadmin se puede acceder sin contraseña a la aplicación.

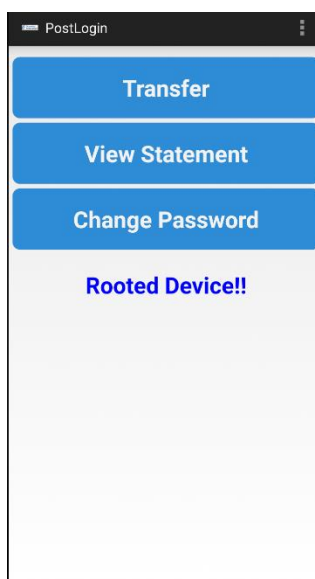


Figura 21: Login devamin

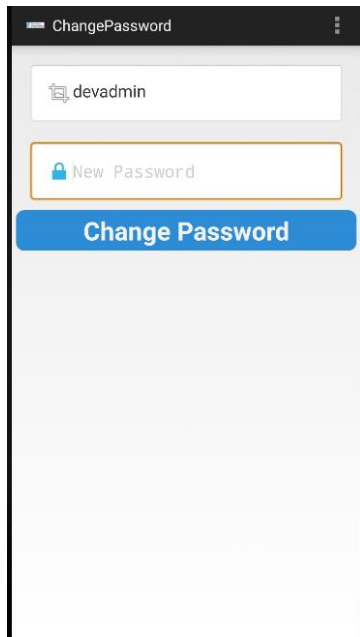


Figura 22: Cambio contraseña devadmin

Además, se puede ejecutar una actividad de una aplicación de forma unitaria si cuenta con un intent (o acción) con el atributo exported en true.

```
David@david MINGW64 ~/pentest
$ adb shell
generic_x86:/ $ am start -n com.android.insecureapp/.PostLogin
Starting: Intent { cmp=com.android.insecureapp/.PostLogin }
generic_x86:/ $
```

Figura 23: Ejecución intent PostLogin

De esta manera, nos saltamos el logging de la aplicación, no necesitamos hacer login con el usuario y contraseña.

Además, para poder realizar un análisis dinámico de la aplicación se hará uso de Mobsf para buscar vulnerabilidades.

```
PS C:\Users\David\pentest\Mobile-Security-Framework-MobSF> .\run.bat
Running MobSF on "0.0.0.0:8000 [::]:8000"
[INFO] 25/Jan/2025 11:07:50 - Loading User config from: C:/Users/David/.MobSF/config.py
[INFO] 25/Jan/2025 11:09:20 -
[MOBSF]
[INFO] 25/Jan/2025 11:09:20 - Author: Ajin Abraham | opensecurity.in
[INFO] 25/Jan/2025 11:09:20 - Mobile Security Framework v4.2.9
REST API Key: f2ce9de127776cffe79ea649e60e8df6dc35d77577456524008f53e64f1a0994
Default Credentials: mobsf/mobsf
[INFO] 25/Jan/2025 11:09:20 - OS Environment: Windows Windows-10-10.0.22631-SP0
[INFO] 25/Jan/2025 11:09:20 - CPU Cores: 6, Threads: 12, RAM: 15.91 GB
[INFO] 25/Jan/2025 11:09:20 - MobSF Basic Environment Check
[INFO] 25/Jan/2025 11:09:20 - Checking for Update.
[WARNING] 25/Jan/2025 11:09:20 - A new version of MobSF is available, Please update to 4.3.0 from master branch.
```

Figura 24

A continuación, se introducirá la apk para su análisis estático:

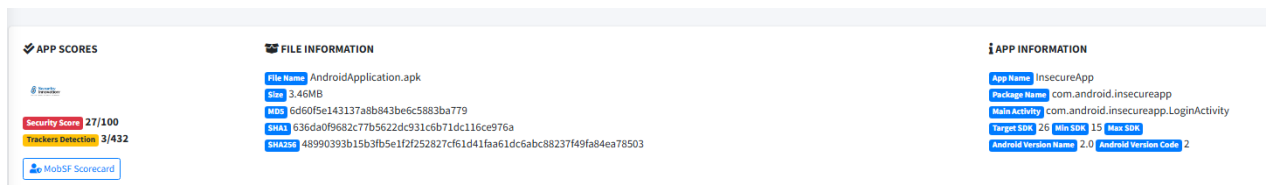


Figura 25: Análisis estático de la aplicación

Además, se podrá llevar acabo un análisis dinámico:

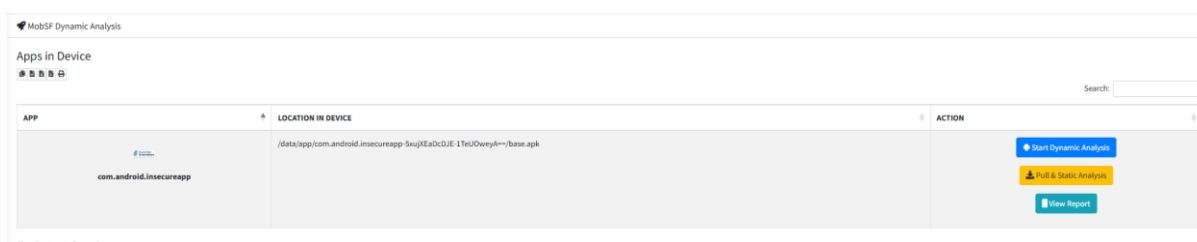


Figura 26: Listado de aplicaciones

Tras realizar un análisis dinámico se obtiene lo siguiente:

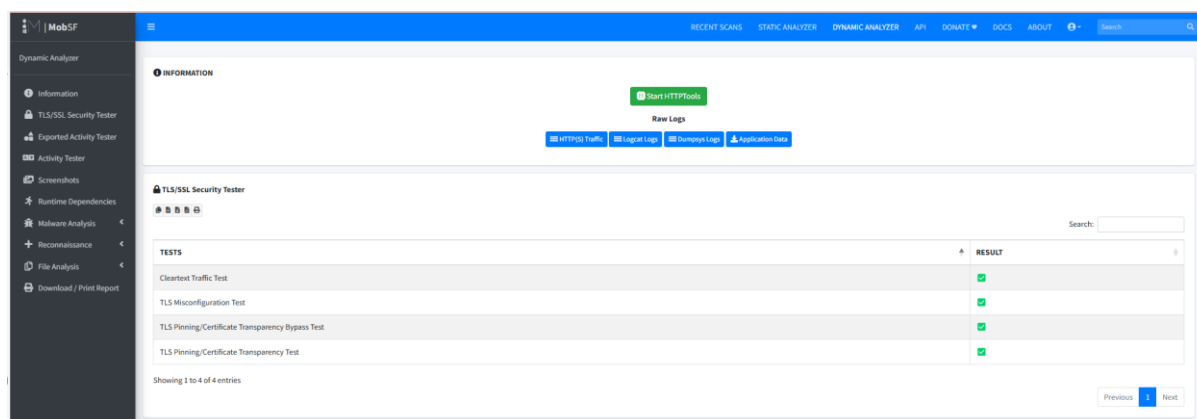


Figura 27: Análisis dinámico

URLS

https://ats-wrapper.privacymanager.io/ats-modules/a95fc332-885d-40c0-aa11-3c7c55aa0d7d/ats.js
https://static.as.com/dist/resources/js/3b7505b1018ec6cf9188608bd905f7bb21a37dd/default.js
http://www.google.com/gen_204
https://ak-ads-ns.prismad.com/slot/as_arc/slot.js
https://semanticlocation-pa.googleapis.com/userlocation/semanticlocation/getdeviceplacementmodel
https://consent.google.es/ml/continue=https://www.google.es/search%3f%3das%26oq%3das%26aq%3dchrome.0.0j5j0.237j0j7%26client%3dms-unknown%26sourceid%3dchrome-mobile%26ie%3dutf-8&gl=es&m=1&pc=srp&xe=none&cm=2&hl=en&src=1
www.google.com,
https://static.as.com/dist/resources/js/3b7505b1018ec6cf9188608bd905f7bb21a37dd/home.js
https://img.asmedia.epimg.net/resizer/v2/cg2cp0612chihagokbyzye.jpg?auth=8301707b26248fe846a96d8a5cc0710a4075f5360e63da75807f6ec36617b49&width=360&height=203&focal=1395%2c385
https://static.as.com/dist/resources/js/3b7505b1018ec6cf9188608bd905f7bb21a37dd/video.js
https://as.com/
https://as01.epimg.net/img/comunes/fotos/fichas/equipos/large/4698.png
https://player.prisamedia.com/core/js/coreplayer.min.js
https://www.google.com/generate_204
https://as01.epimg.net/img/comunes/fotos/fichas/equipos/large/4697.png
https://static.as.com/dist/resources/js/3b7505b1018ec6cf9188608bd905f7bb21a37dd/playlist.js
https://sdk.mf.io/static/marfeel-sdk.js?id=2224
https://android.googleapis.com/checkin
https://static.as.com/dist/resources/js/3b7505b1018ec6cf9188608bd905f7bb21a37dd/audio.js
https://player.prisamedia.com/componentes/reels/reels.lib.js
https://www.gstatic.com/android/sms/sticker/sticker_sets_list_5.xml
https://www.gstatic.com/android/keyboard/dictionarypack/gmon-normal/metadata.json
https://www.gstatic.com/android/keyboard/theme/dulcitone/prod/theme_index_update_info.json
http://connectivitycheck.gstatic.com/generate_204
https://as01.epimg.net/img/comunes/fotos/fichas/equipos/small/2x/4698.png
https://as.com/pf/resources/manifests/site.webmanifest?id=617
https://localhost.sensic.net:54325/?m=asweb&r=as.com&p=es1&instanceid=173756571880444d667d59d31b4ca5daef185765fda702a04310807&redirect=manual
https://fonts.gstatic.com/s/a/directory017.pb
http://play.googleapis.com/generate_204
www.google.com
https://as.com/
https://as01.epimg.net/img/comunes/fotos/fichas/equipos/small/2x/4697.png

Figura 28: Urls de la aplicación

En dicho análisis, se ha obtenido información relativa a las distintas actividades, se ha testeado la conexión TLS/SSL, se ha realizado un análisis de malware, un análisis de los distintos ficheros, así como un reconocimiento de las peticiones https o los distintos emails, entre otras pruebas. Para ello se ha iniciado un análisis dinámico en el Mobsf mientras se navegada por las distintas vistas de la aplicación para testear todos los casos de uso.

Por último, destacar que Burpsuit también hace un reporte de las vulnerabilidades encontradas en dicha aplicación.











Issue type	Host	
 Suspicious input transformation (reflected)	http://insecure-bank.com	/url-shorten
 SMTP header injection	http://insecure-website.c...	/contact-us
 Serialized object in HTTP message	http://insecure-bank.com	/blog
 Cross-site scripting (DOM-based)	https://insecure-bank.com	/
 XML external entity injection	https://vulnerable-websit...	/product/stock
 External service interaction (HTTP)	https://insecure-website....	/product
 Web cache poisoning	http://insecure-bank.com	/contact-us
 Server-side template injection	http://insecure-bank.com	/user-homepage
 SQL injection	https://vulnerable-websit...	/
 OS command injection	https://insecure-website....	/feedback/submit

Figura 29: Análisis de la aplicación de Burpsuite

1.1.4.3 Almacenamiento inseguro de datos.

Como se indica en la FIGURA 20, las credenciales del usuario se guardan en el directorio “SharedPreferences”, por lo que cualquier aplicación puede acceder a dicho fichero.

1.1.4.4 Bypass de autenticación.

Como se indica en la FIGURA 23, se puede hacer un bypass del login, iniciando la aplicación con el intent de PostLogin.

1.1.5 Bypass de detección de root.

Como se indica en el código de la clase PostLogin, para comprobar si está rooteado el dispositivo, se comprueba la existencia de “Superuser.apk” en “/system/data”. Para hacer un bypass de dicha detección se ha renombrado el fichero, de esta manera la apk ya no va a detectar que el dispositivo no está rooteado.

```
private boolean doesSuperuserApkExist(String s) {
    File rootFile = new File("/system/app/Superuser.apk");
    Boolean doesexist = Boolean.valueOf(rootFile.exists());
    return doesexist.booleanValue();
}
```

Figura 30: Función AndroidApplication

```
generic_x86:/system/app # mv Superuser.apk Superuser0.apk
```

Figura 31: Comando

Una vez volvemos a entrar en la aplicación nos indica que ya no está rooteado el dispositivo.

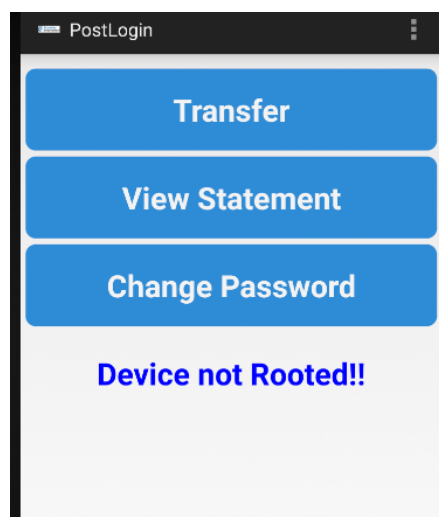


Figura 32: Login AndroidApplication

1.1.6 Bypass de detección de emulador

En primer lugar, se revisará el código fuente de la apk para saber en que punto se verifica si la aplicación está siendo ejecutada en un dispositivo emulado o no.

Como se puede comprobar en la siguiente Figura, en la clase “PostLogin Activity”, se comprueba si el dispositivo es un emulador o no.

```
private void checkEmulatorStatus() {
    Boolean isEmulator = checkIfDeviceIsEmulator();
    if (isEmulator.booleanValue()) {
        Toastroid.show(this, "Application running on Emulator", Toastroid.STYLES.ERROR, 1);
    } else {
        Toastroid.show(this, "Application running on Real device", Toastroid.STYLES.SUCCESS, 1);
    }
}

private Boolean checkIfDeviceIsEmulator() {
    if (Build.FINGERPRINT.startsWith("generic") || Build.FINGERPRINT.startsWith("unknown") || Build.MODEL.contains("google_sdk") || Build.MODEL.contains("Emulator") || Build.MODEL.contains("Android SD
    return true;
    return false;
}
```

Figura 33: Función checkEmulatorStatus

Para poder hacer un bypass de esta detección, se usará la herramienta apktool para hacer una decompilación de una apk que le pasemos para posteriormente modificar dicho comportamiento y volver a compilarla con el objetivo de cambiar el comportamiento de la aplicación generando una nueva apk.

```
PS C:\Users\David\pentest> .\apktool.jar d -r -f .\AndroidApplication.apk
```

Figura 34: Decompilación AndroidApplication

Como se puede observar se obtiene el código smali, que es un pseudocódigo que se utiliza mostrar el código fuente de la aplicación.

```
David@david MINGW64 ~/pentest/AndroidApplication
$ ls -altr
total 465
drwxr-xr-x 1 David 197609    0 Jan 24 14:55 ../
-rw-r--r-- 1 David 197609 448336 Jan 24 14:55 resources.arsc
-rw-r--r-- 1 David 197609  7316 Jan 24 14:55 AndroidManifest.xml
drwxr-xr-x 1 David 197609    0 Jan 24 14:55 res/
drwxr-xr-x 1 David 197609    0 Jan 24 14:55 smali/
drwxr-xr-x 1 David 197609    0 Jan 24 14:55 original/
drwxr-xr-x 1 David 197609    0 Jan 24 14:55 ./
-rw-r--r-- 1 David 197609  312 Jan 24 14:55 apktool.yml

David@david MINGW64 ~/pentest/AndroidApplication
```

Figura 35: Estructura de carpetas AndroidApplication

Como se puede apreciar en el código smali, en la función “checkIfDeviceIsEmulator()” devuelve v0,0x1 en caso de devolver un “true” en java y devuelve un v0,0x0 en caso de ser “false” en java, si modificamos la función para que siempre devuelva “false”, no detectará que la aplicación está siendo ejecutada en un dispositivo emulado.

```

150     const-string v1, "generic"
151
152     invoke-virtual {v0, v1}, Ljava/lang/String;->startsWith(Ljava/lang/String;)Z
153
154     move-result v0
155
156     if-nez v0, :cond_2
157
158     :cond_0
159     const-string v0, "google_sdk"
160
161     sget-object v1, Landroid/os/Build;->PRODUCT:Ljava/lang/String;
162
163     .line 104
164     invoke-virtual {v0, v1}, Ljava/lang/String;=>equals(Ljava/lang/Object;)Z
165
166     move-result v0
167
168     if-eqz v0, :cond_1
169
170     goto :goto_0
171
172     .line 108
173     :cond_1
174     const/4 v0, 0x0
175
176     invoke-static {v0}, Ljava/lang/Boolean;=>valueOf(Z)Ljava/lang/Boolean;
177
178     move-result-object v0
179
180     return-object v0
181
182     .line 106
183     :cond_2
184     :goto_0
185     const/4 v0, 0x1
186
187     invoke-static {v0}, Ljava/lang/Boolean;=>valueOf(Z)Ljava/lang/Boolean;
188
189     move-result-object v0
190
191     return-object v0
192 .end method

```

Figura 36: Código smali de AndroidApplication

Tras realizar el cambio, se compilará el código resultante para incluir la nueva funcionalidad en la aplicación.

```

PS C:\Users\David\pentest> .\apktool.jar b .\AndroidApplication\ -o .\AndroidApplication_david.apk
PS C:\Users\David\pentest> dir

Directorio: C:\Users\David\pentest

Mode                LastWriteTime         Length Name
----                -
d-----          24/01/2025   15:20             AndroidApplication
d-----          18/01/2025   16:57             AndroidApplication.apk.cache
d-----          16/01/2025   18:27             jadx-gui-1.4.6-with-jre-win
d-----          18/01/2025   22:01             Mobile-Security-Framework-MobSF
-a-----          16/01/2025   18:49             1326 9a5ba575.0
-a-----          06/04/2021   16:17          3627039 AndroidApplication.apk
-a-----          24/01/2025   15:21          3409663 AndroidApplication_david.apk
-a-----          16/01/2025   18:30             1196 apktool.bat
-a-----          16/01/2025   18:24          23171720 apktool.jar
-a-----          16/01/2025   18:45             939 cacert.der
-a-----          24/01/2025   14:40             69 comandos.txt
-a-----          24/01/2025   15:20             77535 hs_err_pid20452.log
-a-----          24/01/2025   15:20             326665 replay_pid20452.log
-a-----          16/01/2025   18:24          3208194 uber-apk-signer-1.3.0.jar

```

Figura 37: Compilación apk

Posteriormente se inspeccionará el código fuente de la apk para ver si se ha modificado su contenido.

Haciendo uso de la herramienta jadx se puede visualizar el cambio que se ha hecho previamente en el código smali.

```
private Boolean checkIfDeviceIsEmulator() {  
    if (Build.FINGERPRINT.startsWith("generic") || Build.FINGERPRINT.startsWith("unknown") || Build.MODEL.contains("google_sdk") || Build.MODEL.contains("Emulator") || Build.MODEL.contains("Android 50  
        return false;  
    }  
    return false;  
}
```

Figura 38: Función checkIfDeviceIsEmulator

Como se puede observar dicho método siempre devolverá “false” por lo tanto, no reconocerá que la aplicación se ejecuta en un entorno emulado.

Para probar dicha funcionalidad en el dispositivo, firmaremos la apk para posteriormente probar la funcionalidad en el dispositivo, para ello primero desinstalaremos la versión anterior de la aplicación.

```
PS C:\Users\David\pentest> java -jar .\uber-apk-signer-1.3.0.jar -a .\AndroidApplication_david.apk  
source:  
    C:\Users\David\pentest  
binary-lib/windows-33_0_2/libwinpthread-1.dll  
C:\Users\David\AppData\Local\Temp\uapksigner-2879031524715150786  
zipalign location: BUILT-IN  
    C:\Users\David\AppData\Local\Temp\uapksigner-2879031524715150786\win-zipalign_33_0_2.exe4369051806676041119.tmp  
keystore:  
    [0] 161a0018 C:\Users\David\AppData\Local\Temp\temp_15816420524660640540_debug.keystore (DEBUG_EMBEDDED)  
  
01. AndroidApplication_david.apk  
  
    SIGN  
    file: C:\Users\David\pentest\AndroidApplication_david.apk (3.25 MiB)  
    checksum: 55422f2f23ebdac11550da9eedb69b9345511daeea373098e3f63e2ac05bd6e4 (sha256)  
    - zipalign success  
    - sign success  
  
    VERIFY  
    file: C:\Users\David\pentest\AndroidApplication_david-aligned-debugSigned.apk (3.29 MiB)  
    checksum: d4d80384afca3cea3302514289a5debafc55f3fa50e6c92693871f19a37ddf13 (sha256)  
    - zipalign verified  
    - signature verified [v1, v2, v3]  
      Subject: CN=Android Debug, OU=Android, O=US, L=US, ST=US, C=US  
      SHA256: 1e08a903aef9c3a721510b64ec764d01d3d094eb954161b62544ea8f187b5953 / SHA256withRSA  
      Expires: Thu Mar 10 21:10:05 CET 2044  
  
[Fri Jan 24 15:31:53 CET 2025][v1.3.0]  
Successfully processed 1 APKs and 0 errors in 1.65 seconds.
```

Figura 39: Firma de la aplicación

Como se puede observar la aplicación indica que está siendo ejecutada sobre un dispositivo físico:

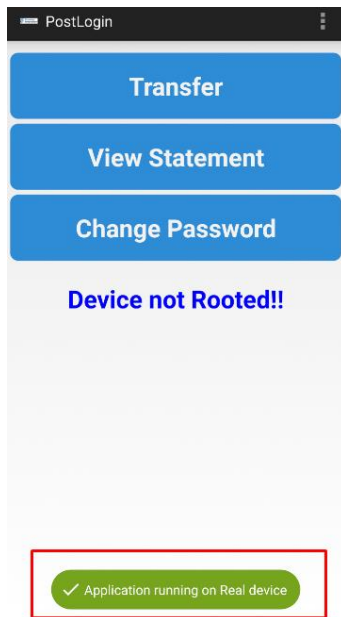


Figura 40: PostLogin aplicación

2. Resumen ejecutivo

ID vulnerabilidad	Título	Severidad
1	Almacén inseguro de datos	Alta
2	Codificación criptográfica insegura	Alta
3	Envíos SMS por parte de la aplicación	Media
4	Información de ejecución con ADB	Media

3. Detalle técnico de las vulnerabilidades

3.1. ID_referencia del título de la vulnerabilidad

ID REFERENCIA	
VULNERABILIDAD Almacén inseguro de datos.	RIESGO: ALTO
Activo afectado: AndroidApplication.apk	
DESCRIPCIÓN	
<p>En la clase “DoLogin”, en el método SaveCreds FIGURA 11 , se almacenan los datos del usuario (usuario y contraseña) en SharedPreferences. Los datos en SharedPreferences se almacenan en texto plano en un archivo XML en el directorio que se indica en la FIGURA 12. Si un atacante accede al dispositivo o al sistema de archivos de la aplicación, podría leer estos datos sensibles.</p> <pre> private void saveCreds(String username, String password) throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException { SharedPreferences mySharedPreferences = DoLogin.this.getSharedPreferences("mySharedPreferences", 0); SharedPreferences.Editor editor = mySharedPreferences.edit(); DoLogin doLogin = DoLogin.this; doLogin.rememberme_username = username; doLogin.rememberme_password = password; String base64Username = new String(Base64.encodeToString(doLogin.rememberme_username.getBytes(), 4)); CryptoClass crypt = new CryptoClass(); DoLogin doLogin2 = DoLogin.this; doLogin2.superSecurePassword = crypt.aesEncryptedString(doLogin2.rememberme_password); editor.putString("encryptedUsername", base64Username); editor.putString("superSecurePassword", DoLogin.this.superSecurePassword); editor.commit(); } </pre> <pre> generic_x86:/data/data/com.android.insecureapp/shared_prefs # ls -latr total 16 drwx----- 6 u0_a79 u0_a79 4096 2025-01-18 07:13 . -rw-rw---- 1 u0_a79 u0_a79 164 2025-01-18 07:14 com.android.insecureapp_preferences.xml drwxrwx--x 2 u0_a79 u0_a79 4096 2025-01-18 07:14 . generic_x86:/data/data/com.android.insecureapp/shared_prefs # </pre>	
RECOMENDACIÓN	
<p>Las credenciales del usuario no deberán de ser expuestas al resto de aplicaciones por lo que deberían de ser guardadas en el Android Keystore System o en su defecto, si se guardan en el SharedPreferences deberían de guardarse de forma encriptada.</p>	

ID REFERENCIA

VULNERABILIDAD

Codificación criptográfica insegura

RIESGO: ALTO

Activo afectado: AndroidApplication.apk

DESCRIPCIÓN

En la clase DoLogin además como se indica en la FIGURA 11, se codifica en Base64 el nombre del usuario, el cual no es un método seguro de cifrado

```
generic_x86:/data/data/com.android.insecureapp/shared_prefs # cat mySharedPreferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="superSecurePassword">v/sJpihDCo2ckDmLW5Uwiw==&#10; </string>
  <string name="EncryptedUsername">amFjaw==&#13;&#10; </string>
</map>
generic_x86:/data/data/com.android.insecureapp/shared_prefs # echo "amFjaw==&#13;&#10;" | base64 -d && echo
jack
generic_x86:/data/data/com.android.insecureapp/shared_prefs # |
```

RECOMENDACIÓN

Si es necesario almacenar el nombre del usuario, se debería de usar un esquema seguro como AES y almacenar la clave en la Android Keystore.

ID REFERENCIA	
VULNERABILIDAD Envíos SMS por parte de la aplicación	RIESGO: MEDIA
Activo afectado: AndroidApplication.apk	
DESCRIPCIÓN	<p>En el AndroidManifest.xml aparece la propiedad android.permission.SEND_SMS la cual permite enviar mensajes SMS, lo que podría usarse para enviar SMS sin que el usuario sea consciente, suscribiéndose a servicios de pago.</p>
RECOMENDACIÓN	<p>Quitar dicha propiedad del AndroidManifest.xml</p>
ID REFERENCIA	
VULNERABILIDAD Interacción de los componentes Android:exported	RIESGO:ALTO
Activo afectado: AndroidApplication.apk	
DESCRIPCIÓN	<p>Ocurre cuando una aplicación Android configura de forma incorrecta la exportación de sus componentes (como actividades, servicios o receptores de difusión) al declararlos en el archivo AndroidManifest.xml. Esto puede permitir que otras aplicaciones maliciosas interactúen con esos componentes de manera no deseada, lo que podría resultar en fugas de datos, ejecución de código no autorizado o escalamiento de privilegios.</p>
RECOMENDACIÓN	<ul style="list-style-type: none"> - Verificar siempre el origen del Intent dentro del código. - Validar y sanitizar cualquier dato recibido. - Para componentes que deben ser accesibles solo para aplicaciones autorizadas, declara permisos personalizados

ID REFERENCIA**VULNERABILIDAD**

Información de ejecución con ADB

RIESGO: MEDIA

Activo afectado: AndroidApplication.apk

DESCRIPCIÓN

La vulnerabilidad de información de ejecución con ADB (Android Debug Bridge) se produce cuando un atacante puede usar comandos de ADB para interactuar con aplicaciones o componentes de un dispositivo Android debido a una configuración insegura o permisos mal gestionados.

RECOMENDACIÓN

Asegurarse de desactivar el modo depuración: adb shell settings put global adb_enabled 0