

Ciberseguridad en
dispositivos móviles

Práctica 3

Luis López Cuerva
Pablo Alcarria Lozano



Introducción	2
Ejercicio 1	2
Ejercicio 2	4
Ejercicio 3	4
Ejercicio 4	6
Ejercicio 5	7
Ejercicio 6	8
Ejercicio 7	10
Ejercicio 8	11
Referencias	11

Introducción

El conjunto de prácticas de la asignatura “Ciberseguridad en Dispositivos móviles” la estamos realizando en conjunto Luis López Cuerva y Pablo Alcarria Lozano. En esta tercera práctica hemos decidido seguir una configuración de emulador genymotion junto a una máquina virtual Kali sobre Windows 10.


Ejercicio 1

El nombre del paquete es com.android.insecurebankv2. Dentro de ella nos encontramos con diversos componentes, entre ellos 10 actividades (4 de ellas exportadas), 2 receivers (uno de ellos exportado) y 1 provider. Según el informe, la actividad LoginActivity es la actividad principal de la aplicación, ya que en el manifiesto de la aplicación así se indica que es la actividad principal cuando se declara la actividad LoginActivity.

Gracias al análisis estático realizado por MobSF podemos ver que la aplicación está firmada de forma que contiene una vulnerabilidad conocida como Janus, que permite a un atacante modificar el código de la aplicación sin que afecte a su firma, por lo que se puede introducir código malicioso bajo la firma hecha por los desarrolladores legítimos.

Respecto a los permisos solicitados, nos encontramos con que se pide un número elevado de permisos, en total 9. De estos 9, MobSF sólo califica como “normales” o “no peligrosos” únicamente dos, que son los que dan acceso a ver el estado de las redes (ACCESS_NETWORK_STATE) y a internet (INTERNET). Como peligrosos, nos encontramos con los siguientes:

1. ACCESS_COARSE_LOCATION da acceso a determinar una ubicación aproximada para determinar en qué sitio está el usuario.
2. GET_ACCOUNTS da acceso a la lista de cuentas en el servicio de cuentas, por lo que la aplicación puede conocer las cuentas de usuario de otros servicios.
3. READ_CONTACTS da permiso a leer la agenda de contactos del usuario por completo.
4. SEND_SMS da permiso a enviar SMS, lo que le puede costar dinero al usuario. Más adelante veremos el uso que se le da al envío de SMS.
5. USE_CREDENTIALS permite a la aplicación pedir tokens de autenticación.
6. WRITE_EXTERNAL_STORAGE permite a la aplicación a escribir en el almacenamiento externo.



En el análisis del manifiesto nos encontramos con que MobSF indica que existen 8 problemas. La mayoría de ellos indican que las actividades no están protegidas y se pueden exportar ya que están declaradas con el parámetro [android:exported=true]. También podemos ver que está indicado que se permite hacer backups de los datos de la aplicación con [android:allowBackup=true], lo cual resulta peligroso ya que los datos de la aplicación, seguramente incluyendo nombre de usuario o alguna información sensible, se pueden recuperar y visualizar desde otro teléfono o desde un emulador.

Respecto a los trackers, en el análisis estático proporcionado por MobSF nos encontramos con 3 trackers, propiedad de google: Google AdMob, utilizado para la monetización de aplicaciones Android, Google Analytics para generar estadísticas y Google Tag Manager que permite medir el ROI de la publicidad de las aplicaciones y generar un seguimiento dentro de la aplicación mediante etiquetas.

Hablando sobre los “secretos posibles”, en el análisis estático también podemos encontrar posibles secretos “hardcodeados”, es decir, que se pueden leer directamente del código y no tienen ningún tipo de protección, como un posible usuario para hacer pruebas en la fase de pruebas de la aplicación que no fue eliminado del código. En nuestro caso, MobSF encuentra dentro del código “loginscreen_password” y “loginscreen_username”, dándonos un falso positivo ya que al encontrar las palabras clave “password” y “username” durante el análisis estático, se muestran por si tuviesen información sensible.

El nivel de riesgo asociado a la apk analizada se califica como bajo, y consigue los 100 puntos en la puntuación de seguridad. Al ver este dato no nos cuadró, ya que esta aplicación está exclusivamente diseñada para encontrar sus debilidades existentes en un entorno controlado. Entendemos que el hecho de que sea en un entorno controlado que no contenga en el código acciones realmente maliciosas contra el teléfono o la máquina del usuario pueda hacer que no se le considere insegura y se pueda analizar con tranquilidad. Este “CVSS security score”[1] es calculado en base a tres métricas:

1. Grupo de métricas base, que se refiere a los aspectos invariables en el entorno.
2. Grupo de métricas temporales, que refleja las características variables en el tiempo pero no en el entorno del usuario.
3. Grupo de métricas de entorno, que tiene en cuenta las dos anteriores métricas además de aplicar la implementación específica de la tecnología vulnerable.

Ejercicio 2

Para baipasear la pantalla del login, es decir, la actividad principal de la aplicación, mediante el uso de ADB podemos hacer una llamada a la actividad “PostLogin”, de forma que no se nos pidan credenciales, ya que se evita todo ese proceso. Este proceso no es sólo evitar esa pantalla, sino las llamadas posteriores como la consulta del usuario a la base de datos del servidor que contiene los usuarios que tienen acceso. En este caso, ni siquiera se hace la llamada al servidor. En un primer momento se intentó evadir el error producido al enviar una petición con los credenciales vacíos, pero era más rápido e interesante simplemente evitar esa pantalla mediante ADB, instalado en la máquina host. La línea para baipasear la actividad principal es:

```
adb shell am start -n com.android.insecurebankv2/com.android.insecurebankv2.PostLogin
```


Requiere que Genymotion esté ejecutando la máquina virtual que contiene la APK instalada, pero para evitar un posible fallo el teléfono no debe estar con la aplicación en primer plano.

Ejercicio 3

El método onCreate de la actividad LoginActivity contiene una comprobación que llama la atención ya que se hace un método *equals* que compara un string con “no”. Con esto podemos intuir que existe algún “modo de administrador” que al principio de la creación en la actividad comprueba si se está ejecutando de esa forma para hacer visible un botón que llama a la acción `createUser()`.

```
22 public class LoginActivity extends Activity {
23     public static final String MYPREFS = "mySharedPreferences";
24     EditText Password_Text;
25     EditText Username_Text;
26     Button createuser_buttons;
27     Button fillData_button;
28     Button login_buttons;
29     String usernameBase64ByteString;
30
31     /* access modifiers changed from: protected */
32     public void onCreate(Bundle savedInstanceState) {
33         super.onCreate(savedInstanceState);
34         setContentView(R.layout.activity_log_main);
35         if (getResources().getString(R.string.is_admin).equals("no")) {
36             findViewById(R.id.button_CreateUser).setVisibility(8);
37         }
38         this.login_buttons = (Button) findViewById(R.id.login_button);
39         this.login_buttons.setOnClickListener(new View.OnClickListener() {
40             /* class com.android.insecurebankv2.LoginActivity.AnonymousClass1 */
41
42             public void onClick(View v) {
43                 LoginActivity.this.performlogin();
44             }
45         });
46         this.createuser_buttons = (Button) findViewById(R.id.button_CreateUser);
47         this.createuser_buttons.setOnClickListener(new View.OnClickListener() {
48             /* class com.android.insecurebankv2.LoginActivity.AnonymousClass2 */
49
50             public void onClick(View v) {
51                 LoginActivity.this.createUser();
52             }
53         });
54     }
55 }
```

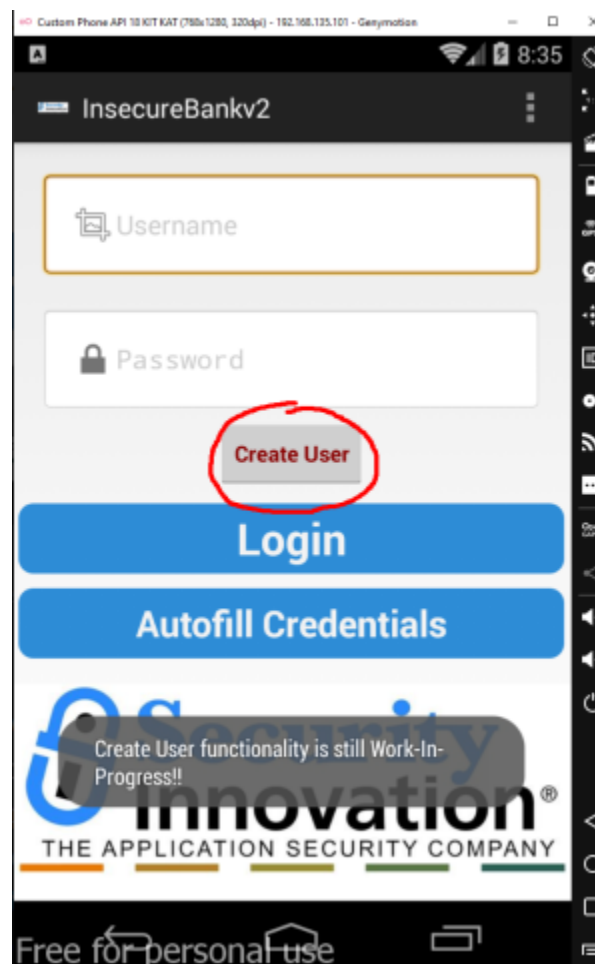
Buscando ese string que contiene el “no”, que está literalmente en el código smali de la actividad LoginActivity, si lo cambiamos a “yes” ya tenemos visible el botón creado. También se puede modificar la comprobación que se realiza, en vez de la variable que se comprueba.



```
246 const v4, 0x7f0d006e
247
248 .line 49
249 invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V
250
251 .line 50
252 const v2, 0x7f04001d
253
254 invoke-virtual {p0, v2}, Lcom/android/insecurebankv2/LoginActivity;->setContentView(I)V
255
256 .line 51
257 invoke-virtual {p0}, Lcom/android/insecurebankv2/LoginActivity;->getResources()Landroid/content/res/Resources;
258
259 move-result-object v2
260
261 const v3, 0x7f07004a
262
263 invoke-virtual {v2, v3}, Landroid/content/res/Resources;->getString(I)Ljava/lang/String;
264
265 move-result-object v1
266
267 .line 52
268 .local v1, "mess":Ljava/lang/String;
269
270 if-eqz v1, :cond_0
271 const-string v2, "yes"
272
273 invoke-virtual {v1, v2}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
274
275 move-result v2
276
277 if-eqz v2, :cond_0
278
```

Si lo comentamos y ponemos "yes" ya tenemos visible el botón, sin necesidad de modificar código java o el botón en sí

Una vez conseguido acceso al botón de crear usuario y hacer click, nos encontramos con que es una funcionalidad “que sigue en progreso”, así se posiblemente por eso por defecto esté oculta:



Ejercicio 4

Echando un primer vistazo al código decompilado de Java, nos encontramos rápidamente con la clase "DoLogin.java", encargada de realizar las peticiones HTTP al servidor donde comprueba las credenciales enviadas. Esta actividad es llamada desde la actividad principal por el método PerformLogin(), que se activa haciendo click en el botón de Login. Dentro del método postData() nos encontramos con dos clases HttpPost. Vemos que tiene un login "normal" y a parte un "devlogin". Esta segunda petición HTTP sólo se envía si el username recogido en el

formulario es igual a "devadmin". En ese caso, se hace un login en el que no se piden credenciales.

Al presionar el botón "Login" tenemos acceso a la actividad PostLogin, y en la máquina virtual de Kali en la que se está ejecutando el servidor nos indica la conexión por la terminal:

```
{"message": "Correct Credentials", "user": "devadmin"}
```

En el caso de hacer login con uno de los dos usuarios, la aplicación escrita en Python que estamos ejecutando nos indica el usuario que es, aunque no nos da ningún tipo de información sobre si tiene permisos especiales.

```
u= <User u'jack'>
```

```
{"message": "Correct Credentials", "user": "jack"}
```

En el análisis de MobSF se encontraban dos posibles "secretos" hardcodeados, pero este "devadmin" no se encuentra incluido.

Ejercicio 5

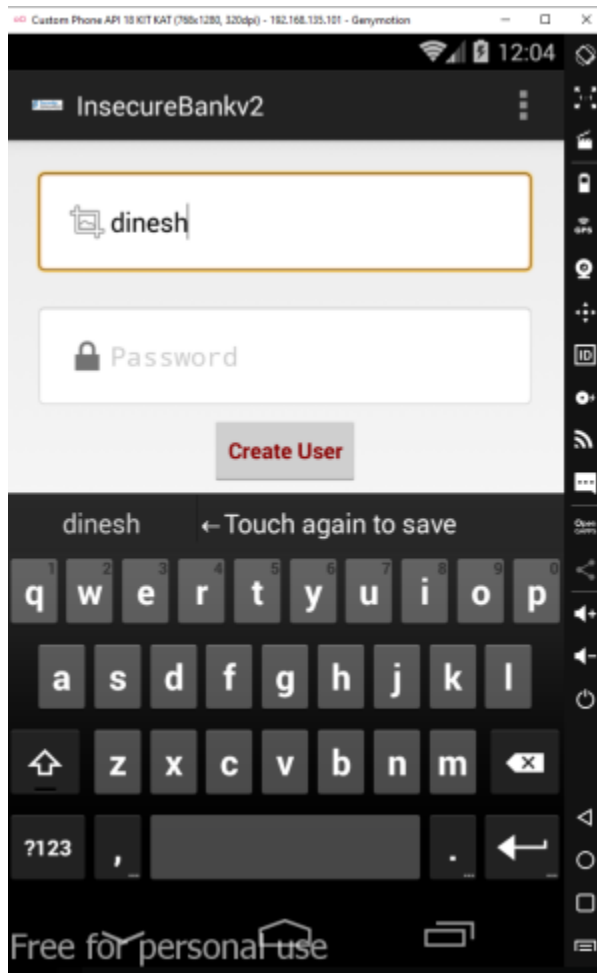
El almacenamiento de credenciales es totalmente inseguro, ya que las credenciales utilizadas se guardan en "shared preferences", un directorio que es accesible por cualquiera. Investigando en la actividad principal, encontramos que al hacer clic en "Autofill credentials", estas se recogen de SharedPreferences. En el código vemos que lo que hace es recuperar del XML "mySharedPreferences" los strings que se encuentran en él y posteriormente decodificarlos. Para la encriptación y desencriptación podemos ver que se utiliza la clase CryptoClass.java, aunque ni siquiera es necesario investigarla para saber que el nombre de usuario se encripta en base64 y la contraseña en AES.

Para el usuario, al estar en base64, que es mucho más débil, con un simple click en cualquier decodificador online tendremos el usuario decodificado.

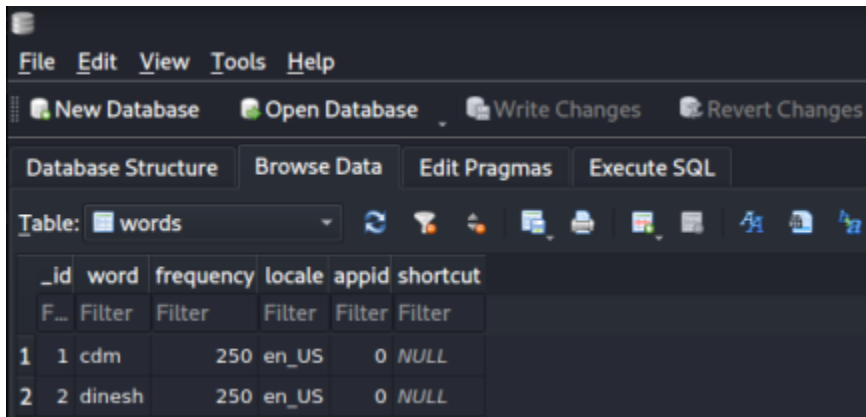
Respecto a la contraseña, dentro de la clase CryptoClass podemos leer que utiliza el modo CBC para la encriptación. En el método "saveCreds" de la clase DoLogin.java nos encontramos con la edición de "Shared Prefs", y ahí se llama al método aesEncryptedString, de la clase CryptoClass.java. Ahí se codifica en AES y una vez codificado en AES, se codifica en base64. Entones, para recuperar la contraseña, ya que en la clase java tenemos la key y el array de bytes que son necesarios para desencriptarlo. Con cualquier herramienta online como CyberChef podemos crear un flujo que introduciendo el string que se encuentra en el XML de shared prefs, codificado en base64, mostrando finalmente la contraseña.

Ejercicio 6

La caché del teclado de google puede suponer un problema de seguridad para los usuarios. En un teclado de google existe un diccionario predeterminado para cada lenguaje, de forma que palabras que estén listadas en el diccionario saldrán como sugerencias mientras se escribe. El problema de seguridad está cuando una palabra no se encuentra dentro del diccionario, ya que se permite el almacenamiento de palabras nuevas. Esto es normal ya que en el lenguaje utilizamos abreviaciones, motes y diminutivos. Sin embargo, además de estas palabras que no se encuentran en un diccionario convencional, tampoco se encuentra el nombre de usuario que se utilice en las aplicaciones. En este caso, el teclado del emulador está en inglés y al usuario "jack" no lo trata como una palabra nueva, ya que al escribirlo el propio teclado nos sugiere que pongamos la jota en mayúsculas, ya que se trata de un nombre propio. Sin embargo sí que se almacena el usuario Dinesh, o cualquier otro, en nuestro caso también escribimos repetidas veces "cdm" para ver si aparecía en esta caché.



Aquí podemos ver cómo el propio teclado nos sugiere guardarla. Este diccionario se encuentra en el espacio de almacenamiento del paquete `com.android.providers.userdictionary` bajo el nombre “`user_dict.db`”. Mediante ADB podemos hacer un pull de esta base de datos al ordenador host y ver el contenido de la base de datos. En nuestro caso, hemos pasado esa base de datos a la máquina Kali y con SQLite hemos visto el contenido:



Database Structure					
Browse Data					
Edit Pragmas					
Execute SQL					
Table: words					
_id	word	frequency	locale	appid	shortcut
F...	Filter	Filter	Filter	Filter	Filter
1	cdm	250	en_US	0	NULL
2	dinesh	250	en_US	0	NULL

Dentro de esta base de datos podemos ver todas las palabras guardadas por el usuario, por lo que puede contener información sensible, desde un apellido de algún amigo, compañero de trabajo o calle, como el nombre de usuario, correo electrónico o incluso número de teléfono si el sistema operativo no reconoce esa cadena de números como un número de teléfono y se guarda como “una palabra nueva”.

Ejercicio 7

El BroadcastReceiver exportado por la aplicación se activa al recibir cualquier broadcast, aunque para su correcto funcionamiento requiere que el broadcast recibido tenga dos parámetros extra de carácter string. Estos dos parámetros teóricamente deberían ser un número de teléfono y una contraseña nueva, aunque en la práctica no tienen porque tener estos valores ya que simplemente se comprueba que el valor del número de teléfono no sea null.

Una vez que el broadcast es activado comprueba que el valor del parámetro número de móvil no es nulo, si no es nulo recupera la usuario y su contraseña y los decodifica para posteriormente enviar un mensaje de texto en el que aparece en forma de texto plano la contraseña antigua, la contraseña nueva y el nombre de usuario.

Este broadcast no es seguro por varios motivos, el primero es que se activa al recibir cualquier broadcast y además es una actividad exportada, por lo que cualquier aplicación del teléfono móvil puede llegar a activarlo, el segundo motivo por el que no es seguro es el tratamiento que hace de datos comprometidos. Manda mediante texto plano credenciales de inicio de sesión a un número de teléfono que recibe como parámetro y sin comprobaciones de seguridad por lo que puede ser empleado para robar credenciales.

Además MobSF ya nos avisó en un primer momento de que este receiver no es seguro ya que es accesible desde otras aplicaciones.

6	Content Provider (com.android.insecurebankv2.TrackUserContentProvider) is not Protected. [android:exported=true]	high	A Content Provider is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
7	Broadcast Receiver (com.android.insecurebankv2.MyBroadCastReceiver) 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.

Ejercicio 8

Para activar el proveedor de contenidos hemos realizado la siguiente petición mediante adb:

```
adb shell content query --uri  
content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
```

Y hemos obtenido:

```
C:\Users\pablo\AndroidStudioProjects\Util\platform-tools>adb shell content query --uri content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers  
Row: 0 id=1, name=jack  
Row: 1 id=2, name=jack  
Row: 2 id=3, name=jack  
Row: 3 id=4, name=jack  
Row: 4 id=5, name=jack
```

En la imagen se puede observar que hemos obtenido un histórico de los usuarios que han hecho uso de la aplicación.

El proceso seguido para llegar a saber que uri teníamos que buscar comenzó con el análisis del archivo TrackUserContentProvider.java donde observamos lo siguiente:

```
static final String URL = "content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers";
```

Una vez leída esta URL decidimos hacer una petición para obtener su contenido y obtuvimos los resultados explicados previamente,

Referencias

[1] <https://www.nowsecure.com/blog/2017/03/28/cvss-scores-for-mobile-apps-guidelines-for-measuring-mobile-risk/>