1.1 - Registro Inseguro

La primera vulnerabilidad obtiene a través de un EditText una entrada de datos la cual se almacena en **`cctxr`** y por error se logea después parseado a string a través de **`Log.e()`**.

En el código básicamente se obtiene lo introducido en el textbox creando un objeto de la clase EditText, para después llamar al método de la clase **`LogActivity procesCC()`** y pasarle como parámetro lo obtenido haciendole un cast a string, ya que el método **`procesCC()`** solo admite como valor un tipo de dato String.

El método **`procesCC()`** envia una excepción, por lo tanto **`checkout()`** siempre entrara en catch() y mostrara lo almacenado en **`cctxt`** a través de **`getText()`** parseado a string:

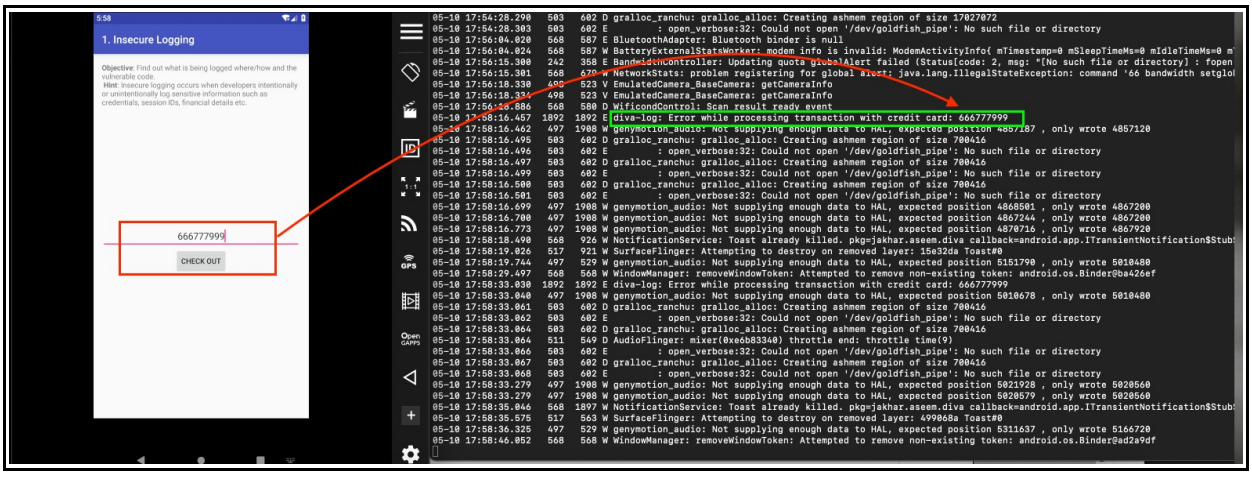
```
/* loaded from: classes.dex */
public class LogActivity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.su
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_log);
    }

    public void checkout(View view) {
        EditText cctxt = (EditText) findViewById(R.id.ccText);
        try {
            processCC(cctxt.getText().toString());
        } catch (RuntimeException e) {
            Log.e("diva-log", "Error while processing transaction with credit card: " + cctxt.getText().toStri
            Toast.makeText(this, "An error occured. Please try again later", 0).show();
        }
    }

    private void processCC(String ccstr) {
        RuntimeException e = new RuntimeException();
        throw e;
    }
}
```

Para debug y ver lo que devuelven los Log, que son una forma de detectar los errores en el desarrollo, tenemos que obtener los logs a través de adb: **`adb logcat`**

Introducimos un numero como ***"666777999"*** en la aplicación y traceamos dinámicamente



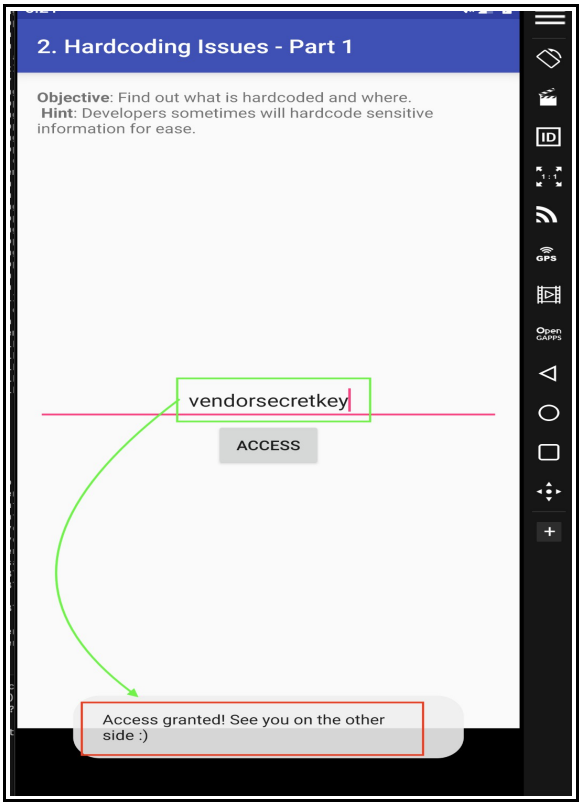
1.2 - Datos sensibles hardcodeados

En la clase **`HardcodeActivity`** se realiza un **`equals()`** a un string harcodeado en el código **`“vedorsecretkey”`**:

```
/* loaded from: classes.dex */
public class HardcodeActivity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseF
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hardcode);
    }

    public void access(View view) {
        EditText hckey = (EditText) findViewById(R.id.hcKey);
        if (hckey.getText().toString().equals("vendorsecretkey")) {
            Toast.makeText(this, "Access granted! See you on the other side :)", 0).show();
        } else {
            Toast.makeText(this, "Access denied! See you in hell :D", 0).show();
        }
    }
}
```

Si introducimos ese texto en el *`textbox`* obtendremos un Acces Granted:



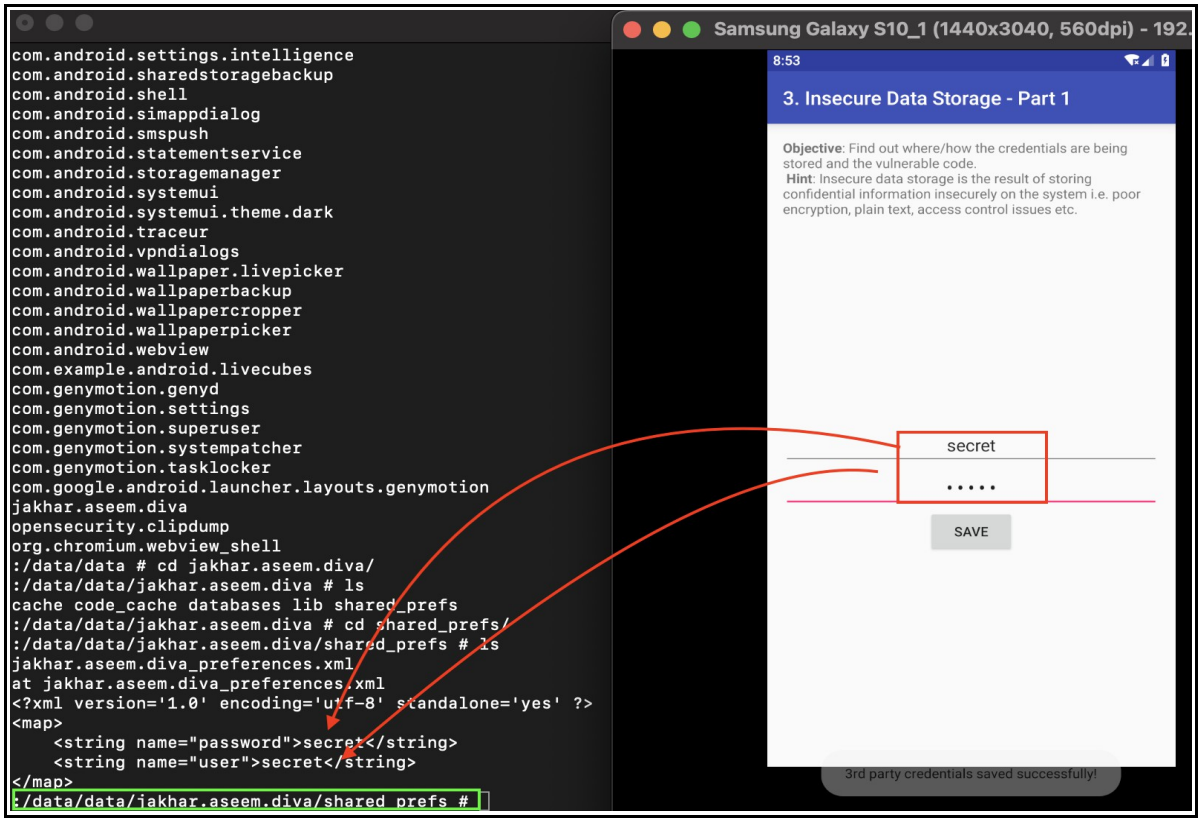
1.3 - Almacenamiento de datos inseguro

Android almacena la configuración de preferencias compartidas en un archivo .xml en la carpeta **`shared_prefs → data/data/[APP]`**.

En el código podemos ver en el método **`saveCredentials()`** como se obtiene la ruta al archivo y se edita modificando los datos introducidos en cada *`textbox`* de la APP con los del .xml

```
19      public void saveCredentials(View view) {
20          SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
21          SharedPreferences.Editor spedit = spref.edit();
22          EditText usr = (EditText) findViewById(R.id.ids1Usr);
23          EditText pwd = (EditText) findViewById(R.id.ids1Pwd);
25          spedit.putString("user", usr.getText().toString());
26          spedit.putString("password", pwd.getText().toString());
27          spedit.commit();
29          Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
      }
  }
```

Por lo tanto las credenciales que guardemos a través de ese View se almacenaran en texto plano en **`data/data/jakhar.aseem.diva/shared_prefs/jakhar.aseem.diva_preferences.xml`**



1.4 - Almacenamiento de datos inseguro 2

En el código se observa como se almacenan los campos **`ids2Usr`** y **`isd2Pwd`** en la tabla *`myuser`*, el primero en el campo user y el segundo en el campo password, ambos en texto plano.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    try {
        this.mDB = openOrCreateDatabase("ids2", 0, null);
        this.mDB.execSQL("CREATE TABLE IF NOT EXISTS myuser(user VARCHAR, password VARCHAR);");
    } catch (Exception e) {
        Log.d("Diva", "Error occurred while creating database: " + e.getMessage());
    }
    setContentView(R.layout.activity_insecure_data_storage2);
}
```

```
public void saveCredentials(View view) {
    EditText usr = (EditText) findViewById(R.id.ids2Usr);
    EditText pwd = (EditText) findViewById(R.id.ids2Pwd);
    try {
        this.mDB.execSQL("INSERT INTO myuser VALUES ('" + usr.getText().toString() + "', '" + pwd.getText().toString() + "');");
        this.mDB.close();
    } catch (Exception e) {
        Log.d("Diva", "Error occurred while inserting into database: " + e.getMessage());
    }
    Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
}
}
```

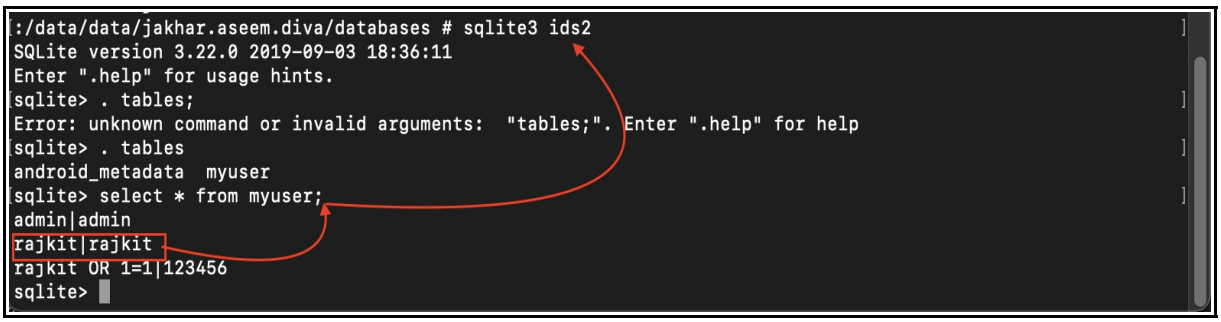
Introduciremos un user y password nuevos a través de la APP:
`user=rajkit,password=rajkit`

Para acceder a la base de datos SQLite tendremos que acceder a través de adb con una shell, para ello → adb shell

Una vez dentro vamos hasta el directorio **`/data/data/[nombre app]/databases`** y nos mostrara las BD que tiene la aplicación y comprobamos si existe la DB que nos mostraba el decompilado. **`“ids2”`**:

```
1|:/data/data/jakhar.aseem.diva/databases # ls
divanotes.db divanotes.db-shm divanotes.db-wal ids2 ids2-shm ids2-wal
:/data/data/jakhar.aseem.diva/databases #
```

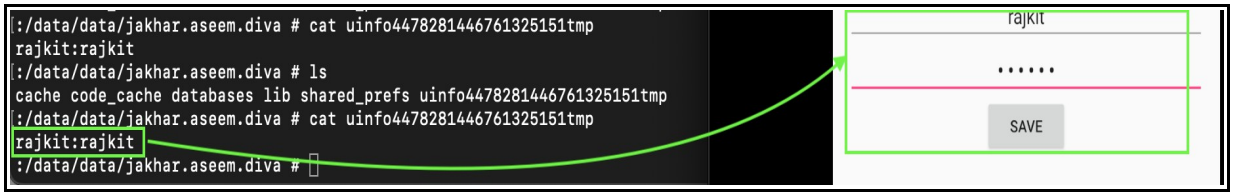
Accederemos a ella a través de sqlite3 desde la misma shell y realizaremos una consulta para que nos vuelque todos los usuarios :



1.5 - Almacenamiento de datos inseguro 3

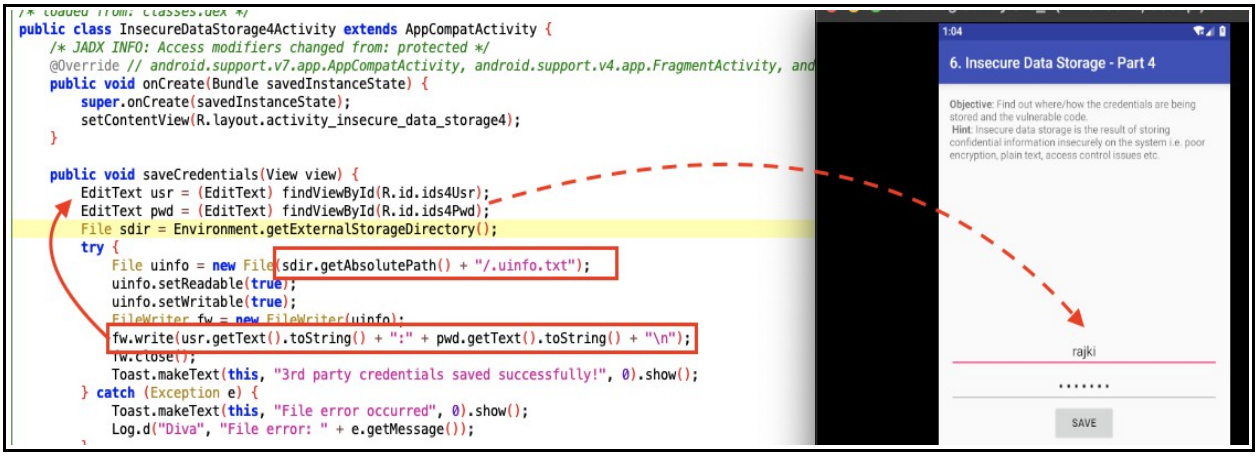
Podemos apreciar en el código que el método **`saveCredentials()`** en este caso obtiene los strings introducidos en ambos EditText y los almacena en texto plano en el directorio propio en un archivo que crea llamado **“uinfo+identificador+tmp”**

Enviamos los datos a través de la app y comprobamos si podemos obtener nuestras credenciales:

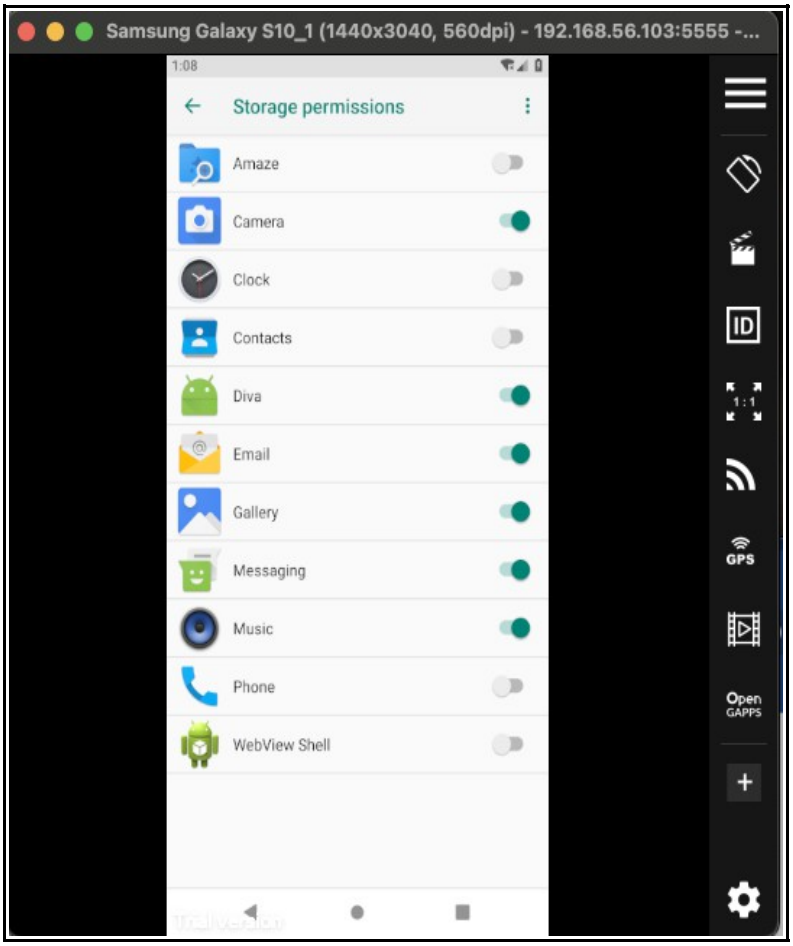


1.6 - Almacenamiento de datos inseguro 4

En este view podemos ver en el código como se obtiene el *path* del almacenamiento externo y se crea un archivo oculto con las credenciales en texto plano:



Para poder crear un archivo con los permisos de **`lectura/escritura`** fuera de los limites de la APP tenemos que darle primero los permisos desde el propio emulador.



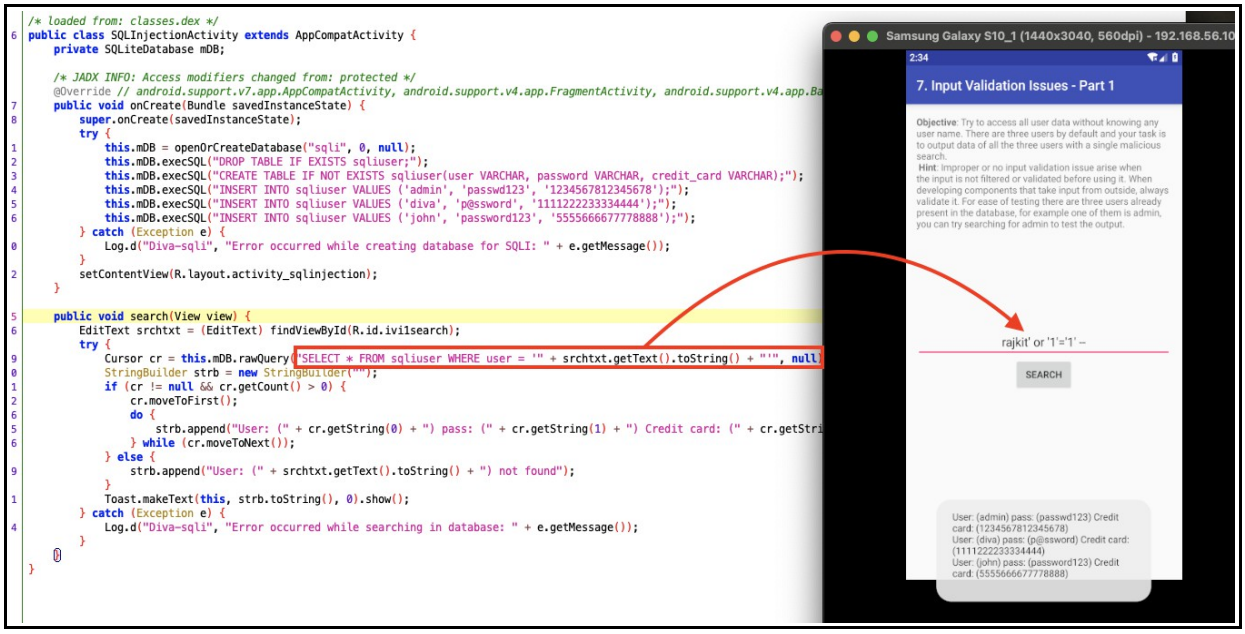
Para ver nuestra credenciales accedemos desde adb shell al directorio de la *sdcard*, y mostramos el directorio completo con ls -la:

```
:/data/data/jakhar.aseem.diva # cd /mnt/
:/mnt # ls
appfuse asec expand media_rw obb runtime sdcard secure shared user vendor
:/mnt # cd s
sdcard/  secure/  shared/
:/mnt # cd sdcard/
:/mnt/sdcard # ls
Alarms DCIM Download Movies Music Notifications Pictures Podcasts Ringtones
:/mnt/sdcard # ls -la
total 92
drwxrwx--x 11 root sdcard_rw 4096 2023-05-11 13:02 .
drwx--x--x  4 root sdcard_rw 4096 2023-05-09 15:03 ..
-rw-rw----  1 root sdcard_rw   14 2023-05-11 13:02 .uinfo.txt
drwxrwx--x  2 root sdcard_rw 4096 2023-05-09 15:03 Alarms
drwxrwx--x  2 root sdcard_rw 4096 2023-05-09 15:03 DCIM
drwxrwx--x  2 root sdcard_rw 4096 2023-05-09 15:03 Download
drwxrwx--x  2 root sdcard_rw 4096 2023-05-09 15:03 Movies
drwxrwx--x  2 root sdcard_rw 4096 2023-05-09 15:03 Music
drwxrwx--x  2 root sdcard_rw 4096 2023-05-09 15:03 Notifications
drwxrwx--x  2 root sdcard_rw 4096 2023-05-09 15:03 Pictures
drwxrwx--x  2 root sdcard_rw 4096 2023-05-09 15:03 Podcasts
drwxrwx--x  2 root sdcard_rw 4096 2023-05-09 15:03 Ringtones
:/mnt/sdcard # cat .uinfo.txt
rajki:rajkiyt
:/mnt/sdcard #
```

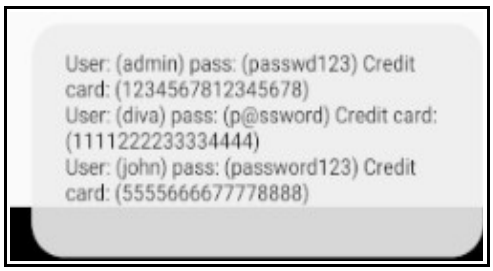
1.7 - Input validation issues 1

En este view tenemos un error en la integración de los datos recibidos desde el **`EditText`** con la creación de la consulta SQLite, básicamente no se controla los tipos de caracteres que se obtienen del **`TextView`** y los integra directamente en la consulta.

La consulta a la BD básicamente vuelca todos los datos del USER siempre y cuando coincida con alguno de la tabla, sin embargo podemos continuar esa consulta introduciendo cualquier usuario, cerramos con una comilla y extendemos con un **`OR`** seguido de **`1=1`** que equivale a a que la condición se cumpla.

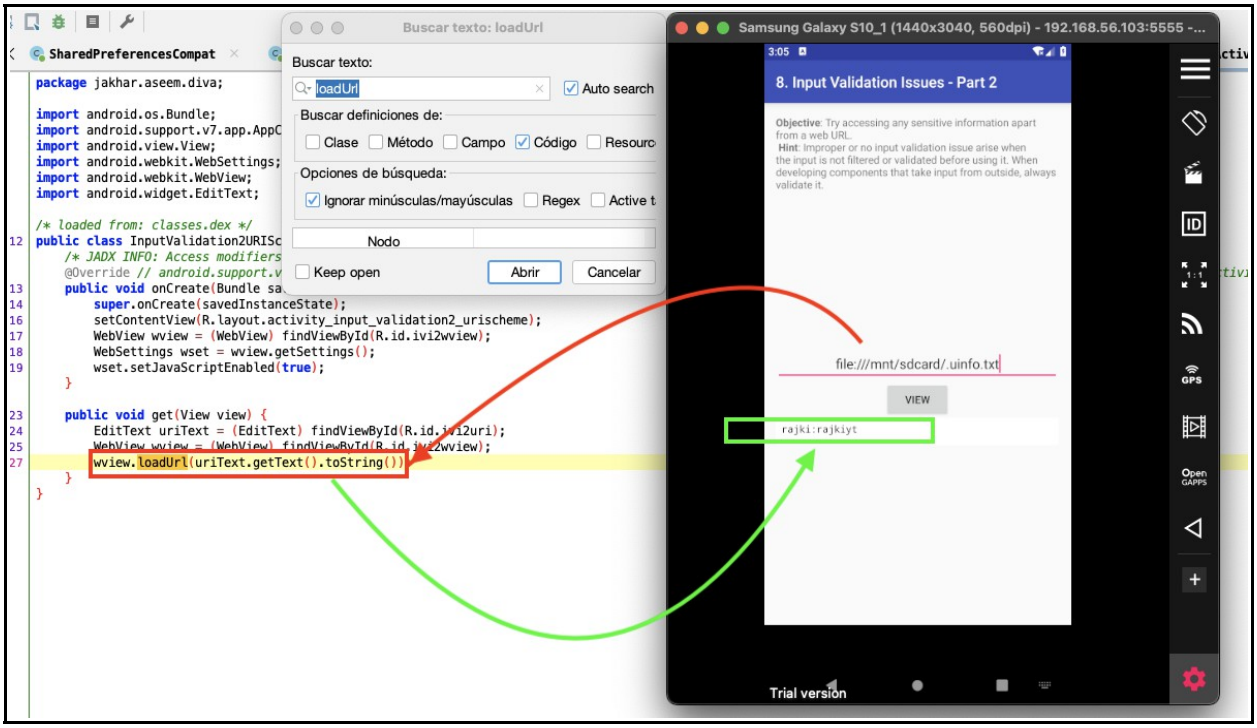


Y nos vuelca toda la tabla ***`sqliuser`***:



1.8 - Input validation issues 2

En este view método **`get()`** obtiene el string del EditText y sin ningún tipo de verificación lo pasa como parámetro a **`loadUrl()`**, por lo tanto podemos acceder a cualquier parte de la que aplicación pueda acceder dentro del teléfono, servidores FTP, etc..



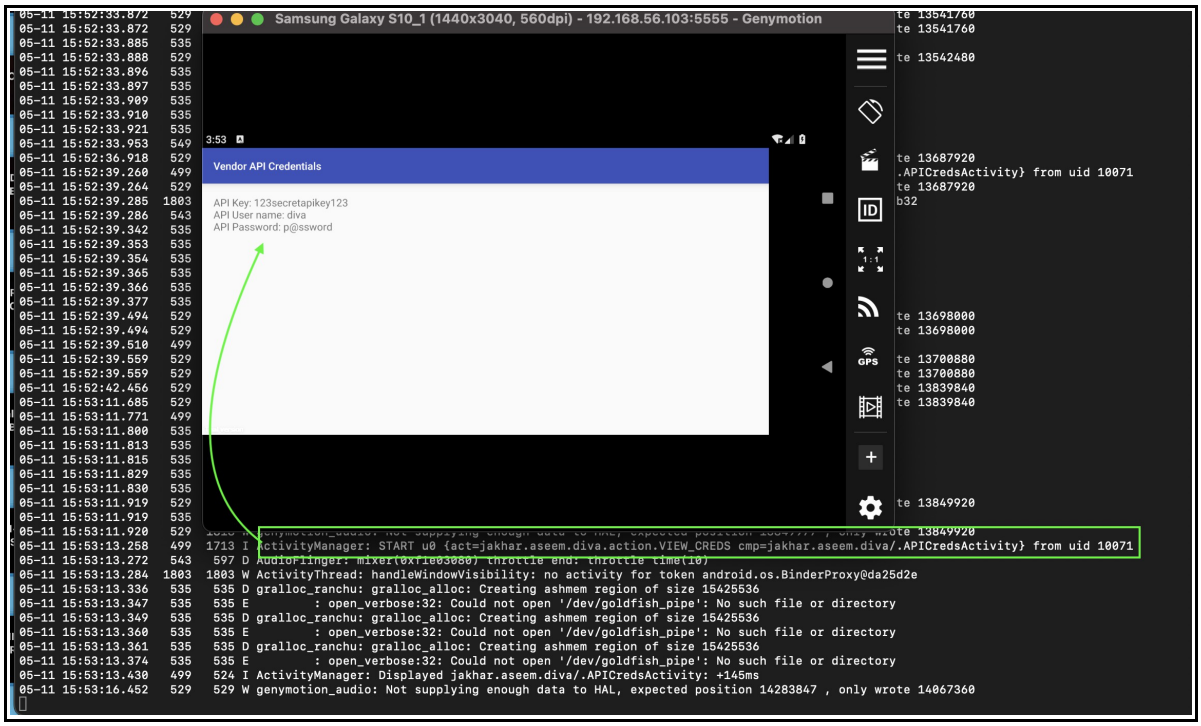
1.9 - Acces Control Issues 1

En Android los Intent son objetos que tienen información que el sistema usa para determinar que componente debe inciar.

Tenemos 2 tipo de Intent:

- **`Explicito`**, estos nos permiten invocar en nuestra APP servicios de los que no disponemos pero sin embargo se nos permite inciarla en la APP.
- **`Implicito`**, estos son los que enviamos a otra aplicación con unos parámetros determinados que necesite la otra APP para invocar startActivity()

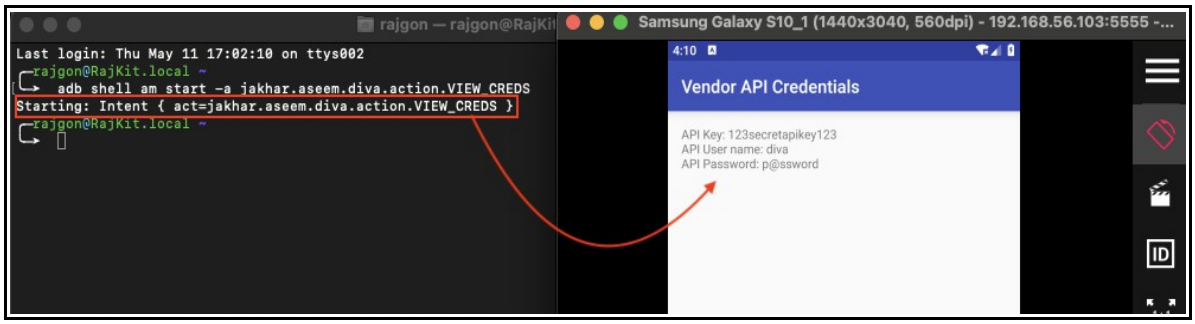
Si pulsamos en el View de la app “VIEW API CREDENTIALS” podemos observar en el logcat como se acciona el Intent al pulsar el boton, el problema es que en el código no se realiza ningún tipo de control de acceso por lo que en teoría también deberíamos de poder accionarlo desde fuera de la APP.



En el código se ve claramente que no hay ninguna estructura de control de flujo que determine si un acceso es licito o no, lo único que verifica es que no contenga un valor NULL:

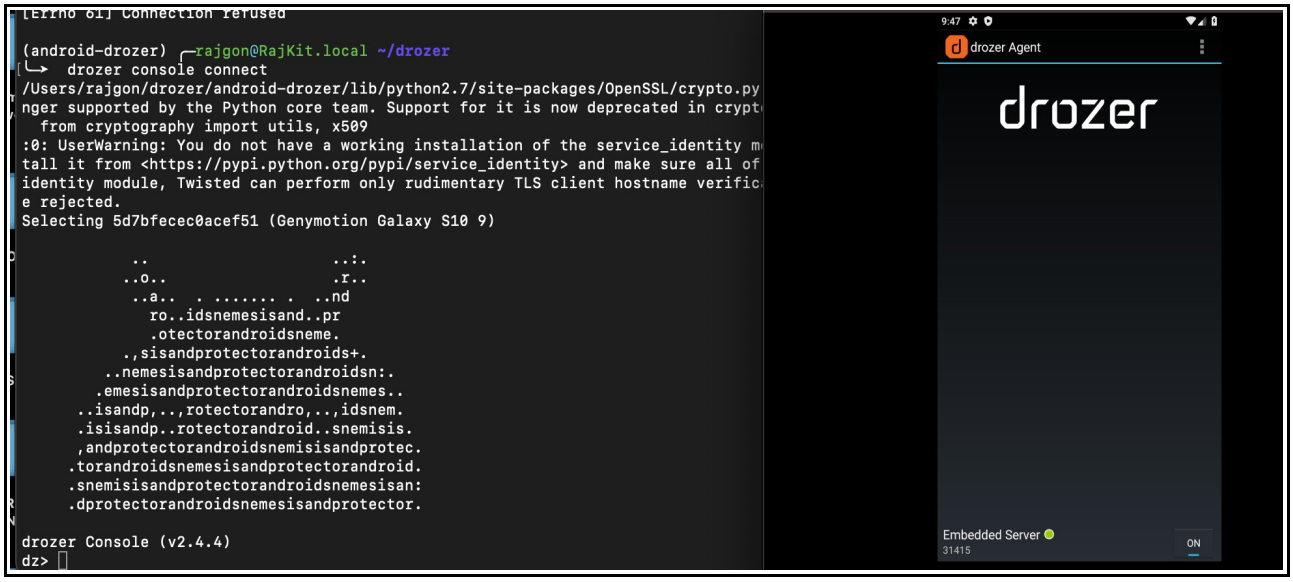
```
    public void viewAPICredentials(View view) {
        Intent i = new Intent();
        i.setAction("jakhar.aseem.diva.action.VIEW_CREDS");
        if (i.resolveActivity(getPackageManager()) != null) {
            startActivity(i);
            return;
        }
        Toast.makeText(this, "Error while getting API details", 0).show();
        Log.e("Diva-aci1", "Couldn't resolve the Intent VIEW_CREDS to our activity");
    }
}
```

Iniciaremos la activamente a través adb shell enviando **"`am start -n`"** y el nombre del componente tal y como esta definido en *`manifest.`*



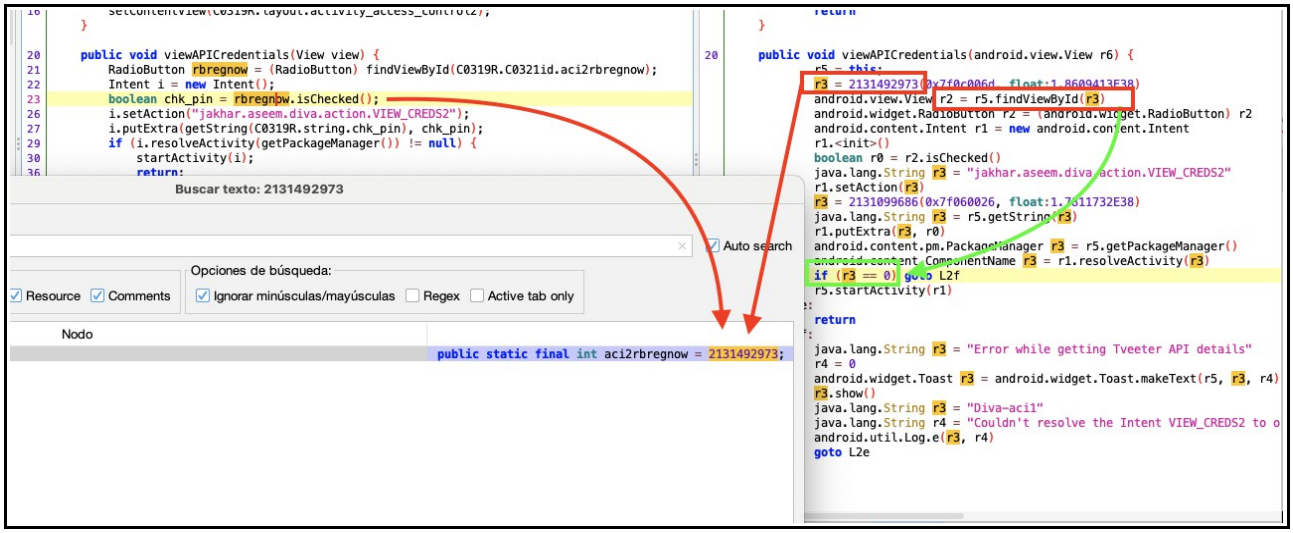
1.10 - Input Validation Issues 2

Desde el entorno virtual que e creado para las dependencias de python y con *`drozer agent`* arrancado en el emulador de android escuchando en el puerto 31415, nos conectamos:

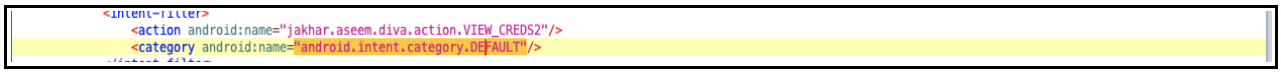


Podemos observar en el código como se va iniciar un **`Intent → startActivity()`** en función del checkbox seleccionado: esta actividad es **`APICreeds2Activity`** la cual va realizar una comprobación del radio button.

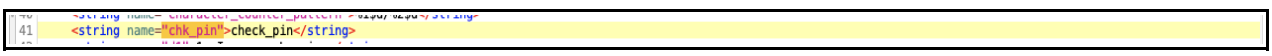
Verificara que que no es **`null`** antes de iniciar la actividad:

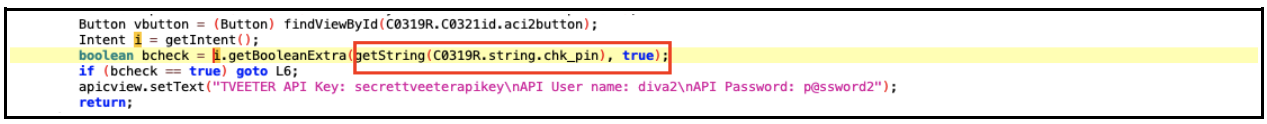


En el *`manifest`* podemos observar como esa acción esta categorizada como DEFAULT lo que significa que se le va a ocultar al usuario:



Realizamos una búsqueda de **`chk_pin`** que nos lleva a strings.xml donde obtenemos en nombre del *`value`* que debe recibir false como parámetro para validar el acceso:



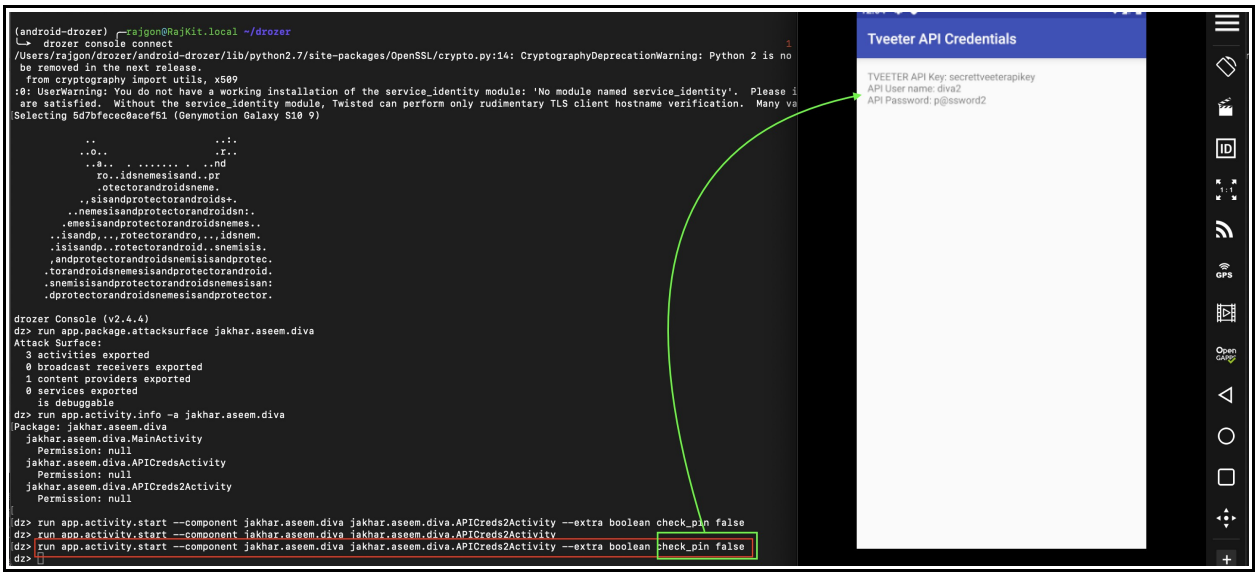


```
@Override // android.support.p003v7.app.AppCompatActivity, android.support.p000v4.app.FragmentActivity, android.support.p000v4.
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(C0319R.layout.activity_apicreds2);
    TextView apicview = (TextView) findViewById(C0319R.C0321id.apic2TextView);
    EditText pintext = (EditText) findViewById(C0319R.C0321id.aci2pinText);
    Button vbutton = (Button) findViewById(C0319R.C0321id.aci2button);
    Intent i = getIntent();
    boolean bcheck = i.getBooleanExtra(getString(C0319R.string.chk_pin), true);
    if (bcheck == true) goto L6;
    apicview.setText("TVEETER API Key: secrettveeterapikey\nAPI User name: diva2\nAPI Password: p@ssword2");
    return;
L6:
    apicview.setText("Register yourself at http://payatu.com to get your PIN and then login with that PIN!");
    pintext.setVisibility(0);
    vbutton.setVisibility(0);
}

public void viewCreds(View view) {
    Toast.makeText(this, "Invalid PIN. Please try again", 0).show();
}
}
```

bcheck ==false

Por lo tanto a partir de aquí deducimos que si enviamos a **`APICreds2Activity`** el parámetro booleano **`check_pin`** como false nos devolverá las credenciales la aplicación, para ello usaremos drozzer como inyector:



1.11 - Input Validation Issues 3

Observando el código de AC3A se puede ver al final como se realiza un intent → startActivity() de **`AccesControl2NotesActivity.class`**

```
    public void goToNotes(View view) {
        Intent i = new Intent(this, AccessControl3NotesActivity.class);
        startActivity(i);
```

Accedemos a esta clase para ver como continua el flujo de ejecución y vemos como se realiza una comparación mediante equals del pin, que de ser correcto se realiza una llamada a CONTENT_URI dentro de la clase **`NotesProvide`**r:

```
    public void accessNotes(View view) {
        EditText pinTxt = (EditText) findViewById(C0319R.C0321id.aci3notesPinText);
        Button abutton = (Button) findViewById(C0319R.C0321id.aci3naccessbutton);
        SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
        String pin = spref.getString(getString(C0319R.string.pkey), "");
        String userpin = pinTxt.getText().toString();
        if (userpin.equals(pin)) {
            ListView lview = (ListView) findViewById(C0319R.C0321id.aci3nlistView);
            Cursor cr = getContentResolver().query(NotesProvider.CONTENT_URI, new String[]{"_id", "title", "note"}, null, null, null);
            String[] columns = {"title", "note"};
            int[] fields = {C0319R.C0321id.title_entry, C0319R.C0321id.note_entry};
            SimpleCursorAdapter adapter = new SimpleCursorAdapter(this, C0319R.layout.notes_entry, cr, columns, fields, 0);
            lview.setAdapter((ListAdapter) adapter);
            pinTxt.setVisibility(4);
            abutton.setVisibility(4);
            return;
        }
        Toast.makeText(this, "Please Enter a valid pin!", 0).show();
    }
}
```

En la clase **`NotesProvide`**r vemos como esta el valor de CONTENT_URI:

```
/* loaded from: classes.dex */
public class NotesProvider extends ContentProvider {
    static final String AUTHORITY = "jakhar.aseem.diva.provider.notesprovider";
    static final String CREATE_TBL_QRY = " CREATE TABLE notes (_id INTEGER PRIMARY KEY AUTOINCREMENT, title TEXT NOT NULL, note TEXT NOT NULL);";
    static final String C_ID = "_id";
    static final String C_NOTE = "note";
    static final String C_TITLE = "title";
    static final String DBNAME = "divanotes.db";
    static final int DBVERSION = 1;
    static final String DROP_TBL_QRY = "DROP TABLE IF EXISTS notes";
    static final int PATH_ID = 2;
    static final int PATH_TABLE = 1;
    static final String TABLE = "notes";
    SQLiteDatabase mDB;
    static final Uri CONTENT_URI = Uri.parse("content://jakhar.aseem.diva.provider.notesprovider/notes");
    static final UriMatcher uriMatcher = new UriMatcher(-1);
```

Podemos realizar una consulta a través de URI con *adb* o *`drozer`*, también podríamos acceder directamente la base de datos **`divanotes.db`**:

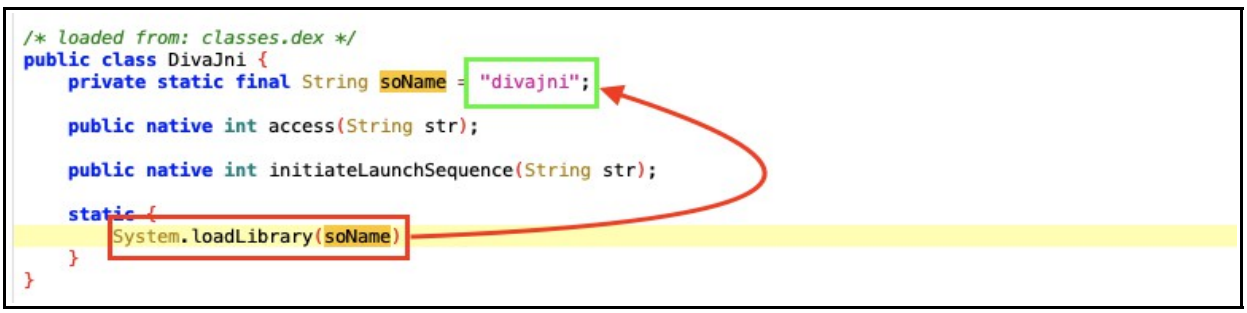
```
┌─rajgon@RajKit.local ~/drozer
└─➤  adb shell content query --uri content://jakhar.aseem.diva.provider.notesprovider/notes
Row: 0 _id=5, title=Exercise, note=Alternate days running
Row: 1 _id=4, title=Expense, note=Spent too much on home theater
Row: 2 _id=6, title=Weekend, note=b333333333333r
Row: 3 _id=3, title=holiday, note=Either Goa or Amsterdam
Row: 4 _id=2, title=home, note=Buy toys for baby, Order dinner
Row: 5 _id=1, title=office, note=10 Meetings. 5 Calls. Lunch with CEO
```

Si accedemos a **`/data/data/[nombre apk]/databases`** tenemos acceso a las bases de datos que utiliza la aplicación, simplemente elegimos la BD que obtenemos en el código de **`NoteProvider`**, realizamos una consulta de los nombre de las tablas y otra consulta con los datos que queremos que nos muestre de esa tabla:

1.12 - Hardconding Issues 2

Analizando el código observamos como se inicia una instancia de la clase **`DivaJn`**i y se realiza en el método *`acces()`* una comprobación:

Realizando un seguimiento accedemos a la clase DivaJni y observamos como el código invoca una librería con el **`string`** contenido en la variable soName:



Esto significa que automáticamente cargará la librería *`divajni.so`* almacenada en la carpeta **`Resources/lib/[arquitectura]`**

```
sh: readelf: not found
127|:/data/data/jakhar.aseem.diva/lib # ls -la
total 28
drwxr-xr-x 2 system system 4096 2023-05-10 09:38 .
drwxr-xr-x 3 system system 4096 2023-05-10 09:38 ..
-rwxr-xr-x 1 system system 5164 2015-12-31 14:37 libdivajni.so
:/data/data/jakhar.aseem.diva/lib #
```

Sabemos que este archivo es una biblioteca nativa compilada, así que la descargaremos de android y realizaremos una búsqueda de posibles datos a través de sus secciones:

```
┌─rajgon@RajKit.local ~/drozer
└─➤  objdump --section-headers libdivajni.so

libdivajni.so:  file format elf32-i386

Sections:
Idx Name                  Size     VMA      Type
  0                       00000000 00000000
  1 .dynsym               000000b0 00000114
  2 .dynstr               000000e8 000001c4
  3 .hash                 00000040 000002ac
  4 .rel.dyn              00000010 000002ec
  5 .rel.plt              00000020 000002fc
  6 .plt                  00000050 00000320 TEXT
  7 .text                 00000186 00000370 TEXT
  8 .rodata               00000014 000004f6 DATA
  9 .eh_frame             00000150 0000050c DATA
 10 .eh_frame_hdr         00000054 0000065c DATA
 11 .fini_array           00000008 00001ee0
 12 .init_array           00000004 00001ee8
 13 .dynamic              000000f8 00001eec
 14 .got                  00000000 00001fe4 DATA
 15 .got.plt              0000001c 00001fe4 DATA
 16 .data                 00000004 00002000 DATA
 17 .bss                  00000000 00002004 BSS
 18 .comment              00000010 00000000
 19 .note.gnu.gold-version 0000001c 00000000
 20 .shstrtab             000000b3 00000000
┌─rajgon@RajKit.local ~/drozer
└─➤  objdump -s -j .rodata libdivajni.so

libdivajni.so:  file format elf32-i386
Contents of section .rodata:
 04f6 6f6c7364 66676164 3b6c6800 2e646f74  olsdfgad;lh..dot
 0506 646f7400                             dot.
┌─rajgon@RajKit.local ~/drozer
└─➤
```

Sabemos que la sección **`.rodata`** se almacenan valores de constantes y se colocan datos inicializados, ademas sus datos nunca son modificados.