Authentication Bypass

Application patching in Android refers to the process of modifying an app to test its resilience against unauthorized changes or to better understand its behavior under different conditions. This technique can reveal potential vulnerabilities and provide insight into how the application responds when specific components are altered. Due to the open nature of the Android platform, it is crucial for penetration testers to have a solid understanding of application patching and its role in mobile security assessments. One key benefit of this approach is that it enables testers to simulate attacks by bypassing certain functionalities—such as authentication mechanisms—to determine what unauthorized actions can be performed. This helps identify and remediate security loophooles, ultimately enhancing the overall security of the application.

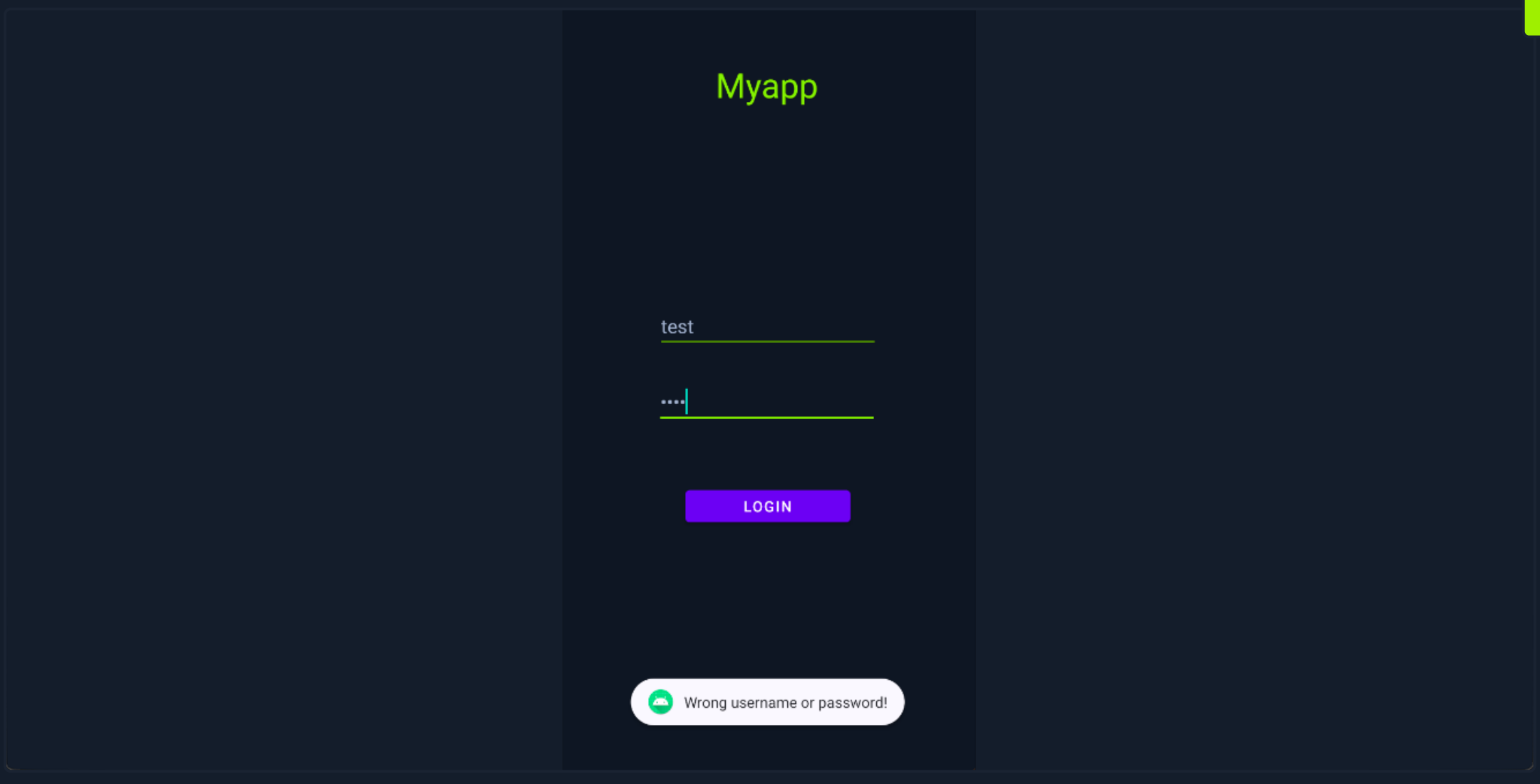
Among the various patching techniques available, this section focuses on a common method involving the decompilation of an APK file to inspect and modify its source code or resource files using tools such as JADX and APKTool. In the example below, we'll examine an application that requires user authentication before granting access to its features. We'll attempt to bypass this restriction using application patching. Although the steps will be demonstrated on an Android Virtual Device (AVD) emulator, the process is also applicable to other Android emulators or physical devices. Once the emulator is running, use the following commands to connect to the device via `ADB` (Android Debug Bridge) and install the application:

Authentication Bypass

```
rl1k@htb[/htb]$ adb connect
rl1k@htb[/htb]$ adb install myapp.apk

Performing Streamed Install
Success
```

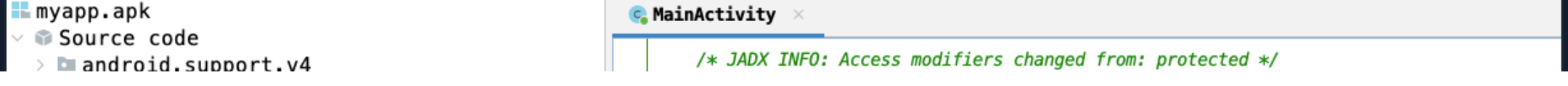
Starting the application and attempting to log in with the random credentials `test/test` will display the following message.



Now, let's use JADX to read its source.

Authentication Bypass

```
rl1k@htb[/htb]$ jadx-gui myapp.apk
```



```
androidx
com
  google
  hackthebox.myapp
    databinding
    AppDatabase
    AppDatabase_Impl
    DatabaseClient
    MainActivity
    R
    User
    UserDao
    UserDao_Impl
kotlin
```

```
    @Override
// androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx
.core.app.ComponentActivity, android.app.Activity
38      public void onCreate(Bundle savedInstanceState) {
39          super.onCreate(savedInstanceState);
40          setContentView(R.layout.activity_main);
42          final EditText editText = (EditText) findViewById(R.id.usernameEditText);
43          final EditText editText2 = (EditText) findViewById(R.id.passwordEditText);
46          ((Button) findViewById(R.id.loginButton)).setOnClickListener(new View.
OnClickListener() { // from class: com.hackthebox.myapp.MainActivity.1
                @Override // android.view.View.OnClickListener
48              public void onClick(View view) {
28                  MainActivity.this.authenticateUser(editText.getText().toString().trim(),
editText2.getText().toString().trim());
                }
            });
        }
```

On the left side of the window, we can tell from the activity names that the app might use a database to authenticate users locally. The `MainActivity` class—as seen the above snippet—indicates a call to the method `authentication()`, which takes two parameters. Double-clicking on this method shows the following snippet.

```
    /* JADX INFO: Access modifiers changed from: private */

/* JADX WARN: Type inference failed for: r0v0, types: [com.hackthebox.myapp.MainActivity
$1AuthenticateUser] */
    public void authenticateUser(final String username, final String password) {
        new AsyncTask<Void, Void, User>() {
// from class: com.hackthebox.myapp.MainActivity.1AuthenticateUser
            /* JADX INFO: Access modifiers changed from: protected */
            @Override // android.os.AsyncTask
            public User doInBackground(Void... voids) {
                return DatabaseClient.getInstance(MainActivity.this.
getApplicationContext()).getAppDatabase().userDao().getUser(username, password);
            }

            /* JADX INFO: Access modifiers changed from: protected */
            @Override // android.os.AsyncTask
            public void onPostExecute(User user) {
                super.onPostExecute((C1AuthenticateUser) user);
                if (user != null) {
                    MainActivity mainActivity = MainActivity.this;
                    Toast.makeText(mainActivity, mainActivity.stringFromJNI(), 1).show()
;
                    return;
                }
                Toast.makeText(MainActivity.this, "Wrong username or password!", 1).show
();
            }
        }.execute(new Void[0]);
    }
}
```

The method `authenticateUser` uses an asynchronous task to verify user credentials. It takes `username` and `password` as parameters and queries the database in the `doInBackground` method for a matching `User` object. If a user is found, a specific toast message is displayed. If no user is found (indicating invalid credentials), the "Wrong username or password!" message is shown. This process is executed in the background to avoid UI interruption. Let's try to patch the application and bypass this authentication mechanism. First, we need to decompile the APK file using APKTool.

Authentication Bypass

```
rl1k@htb[/htb]$ apktool d myapp.apk

I: Using Apktool 2.7.0 on myapp.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /Users/bertolis/Library/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
I: Copying META-INF/services directory
```

Decompiling the app using the APKTool will give us the Smali representation of the source code. This, apart from allowing one to analyze and understand the application's functionality and behavior, will also enable pen-testers to edit the code and change the flow of the app. Listing the content of the directory `myapp/smali/com/hackthebox/myapp/` reveals the following files.

Authentication Bypass

```
rl1k@htb[/htb]$ ls -l myapp/smali/com/hackthebox/myapp

total 208
-rw-r--r--  1 bertolis  bertolis    393 Nov 13 12:40 AppDatabase.smali
-rw-r--r--  1 bertolis  bertolis  11254 Nov 13 12:40 AppDatabase_Impl$1.smali
-rw-r--r--  1 bertolis  bertolis  10632 Nov 13 12:40 AppDatabase_Impl.smali
-rw-r--r--  1 bertolis  bertolis   2836 Nov 13 12:40 DatabaseClient.smali
-rw-r--r--  1 bertolis  bertolis   3083 Nov 13 12:40 MainActivity$1.smali
-rw-r--r--  1 bertolis  bertolis   5722 Nov 13 12:40 MainActivity$1AuthenticateUser.smali
-rw-r--r--  1 bertolis  bertolis   3474 Nov 13 12:40 MainActivity.smali
<SNIP>
-rw-r--r--  1 bertolis  bertolis    651 Nov 13 12:40 R.smali
-rw-r--r--  1 bertolis  bertolis    379 Nov 13 12:40 User.smali
-rw-r--r--  1 bertolis  bertolis    721 Nov 13 12:40 UserDao.smali
-rw-r--r--  1 bertolis  bertolis   3649 Nov 13 12:40 UserDao_Impl$1.smali
-rw-r--r--  1 bertolis  bertolis   7643 Nov 13 12:40 UserDao_Impl.smali
drwxr-xr-x  3 bertolis  bertolis     96 Nov 13 12:40 databinding
```

Among other files, `MainActivity.smali` is also listed. Let's open it with a text editor and try to locate the `if` statement that checks for our username/password in the database. The smali representation of the code contains plaintext, hardcoded strings. Therefore, a good practice that will save us some time is to search for the message `Wrong username or password!`, as it will almost certainly be included in the `if` statement. However, searching for this string inside the `MainActivity.smali` returns nothing.

Authentication Bypass

```
rl1k@htb[/htb]$ grep -Rnw './myapp/smali/com/hackthebox/myapp/MainActivity.smali' -e 'Wrong username or password!'
```

Since the method's name is `authenticateUser.smali`, we should check the content of the class `MainActivity$1AuthenticateUser.smali`.

Authentication Bypass

```
rl1k@htb[/htb]$ grep -Rnw './myapp/smali/com/hackthebox/myapp/MainActivity$1AuthenticateUser.smali' -e 'Wrong username or pas

./myapp/smali/com/hackthebox/myapp/MainActivity$1AuthenticateUser.smali:174:    const-string v1, "Wrong username or password!
```

This is successful. Opening this file using a text editor allows us to navigate to the following snippet of smalli code.

Authentication Bypass

```
rl1k@htb[/htb]$ vim myapp/smali/com/hackthebox/myapp/MainActivity\$1AuthenticateUser.smali
```

Code: smali

```
<SNIP>
    if-eqz p1, :cond_0

    .line 76
    iget-object p1, p0, Lcom/hackthebox/myapp/MainActivity$1AuthenticateUser;->this$0:Lcom/hackthebox/myapp/MainActivity;

    invoke-virtual {p1}, Lcom/hackthebox/myapp/MainActivity;->stringFromJNI()Ljava/lang/String;

    move-result-object v1

    invoke-static {p1, v1, v0}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/

    move-result-object p1

    invoke-virtual {p1}, Landroid/widget/Toast;->show()V

    goto :goto_0

    .line 78
```

```
    :cond_0
    iget-object p1, p0, Lcom/hackthebox/myapp/MainActivity$1AuthenticateUser;->this$0:Lcom/hackthebox/myapp/MainActivity;

    const-string v1, "Wrong username or password!"

    invoke-static {p1, v1, v0}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/
<SNIP>
```

Reading the snippet above, we notice the line `if-eqz p1, :cond_0`, where the `if-eqz` instruction (meaning "if equals zero") checks whether the User object (`p1`) is null. If it is, execution jumps to the label `:cond_0`, where the instruction `const-string v1, "Wrong username or password!"` is executed. This message is displayed on the screen when the user is not found. Let's try changing `if-eqz` to `if-nez` (meaning "if not equals zero"). This modification will cause the login to succeed even if the user is not found in the database. Below is the modified snippet:

Code: smali

```
<SNIP>
    if-nez p1, :cond_0

    .line 76
    iget-object p1, p0, Lcom/hackthebox/myapp/MainActivity$1AuthenticateUser;->this$0:Lcom/hackthebox/myapp/MainActivity;
<SNIP>
```

Once it's changed, recompile the APK file using the following command.

Authentication Bypass

```
rl1k@htb[/htb]$ apktool b myapp

I: Using Apktool 2.7.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs... (/lib)
I: Copying libs... (/kotlin)
I: Copying libs... (/META-INF/services)
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk into: myapp/dist/myapp.apk
```

The new APK file is located on `myapp/dist/` directory. Before installing it on the device, we must create a key and sign the app. We can do this using the following commands.

Authentication Bypass

```
rl1k@htb[/htb]$ echo -e "password\npassword\njohn doe\ntest\ntest\ntest\ntest\ntest\nyes" > params.txt
rl1k@htb[/htb]$ cat params.txt | keytool -genkey -keystore key.keystore -validity 1000 -keyalg RSA -alias john
rl1k@htb[/htb]$ zipalign -p -f -v 4 myapp/dist/myapp.apk myapp_aligned.apk
rl1k@htb[/htb]$ echo password | apksigner sign --ks key.keystore myapp_aligned.apk

Keystore password for signer #1:
```

Finally, before installing the signed APK, we need to uninstall any left over app from previous exercises. We can do this from the device, or if we know the app's package name (which can be retrueved using the command `adb shell ps | grep myapp` while the app is running), we can do it through ADB. Then, we can use ADB again to install the new one.

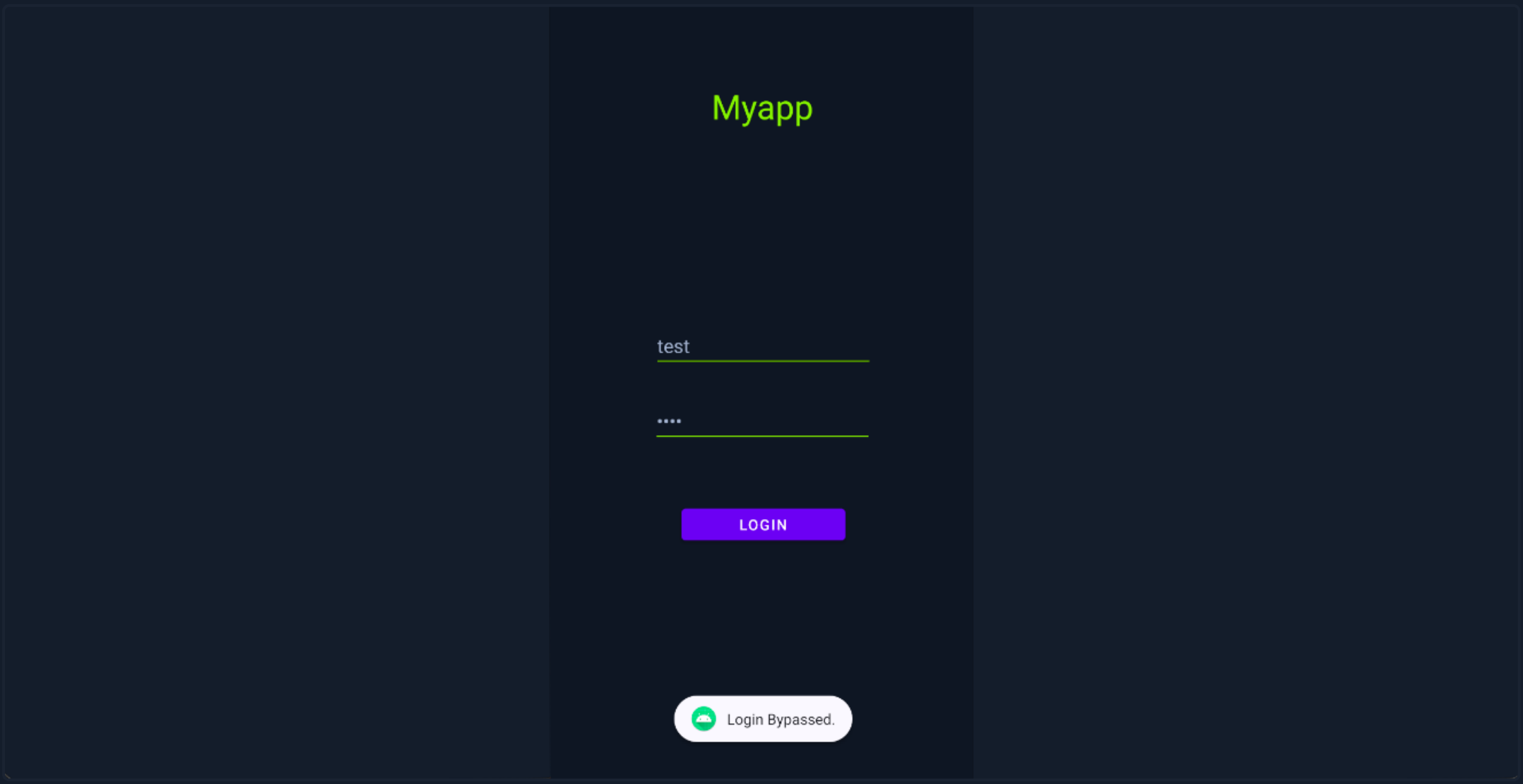
Authentication Bypass

```
rl1k@htb[/htb]$ adb uninstall com.hackthebox.myapp
rl1k@htb[/htb]$ adb install myapp_aligned.apk

Performing Incremental Install
```

```
Serving...
All files should be loaded. Notifying the device.
Success
Install command complete in 592 ms
```

Once installed, we can tap on it and try to log in using the credentials `test/test`.



The local authentication mechanism has been successfully bypassed by patching the application. Although real world applications use server-side authentication, bypassing the client-side login can directly provide access to APIs or data endpoints that are not adequately protected.

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Cheat Sheet

Answer the question(s) below to complete this Section and earn cubes!

+ 3 What is the message displayed on the screen after bypassing the login mechanism?

Submit your answer here...

+10 Streak pts Submit myapp_auth_bypass.zip

Previous Next

Cheat Sheet

Go to Questions

Table of Contents

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left