IDOR Attack

`Insecure Direct Object Reference (IDOR)` vulnerabilities occur when an application exposes direct access to server-side objects, based on user input. In this context, an "object" refers to any internal resource the web server manages—such as user records, files, database entries, or other identifiers tied to specific user actions. When these objects are referenced using predictable or modifiable values (e.g., user IDs or file names), attackers can manipulate input parameters to access or modify data that does not belong to them. For example, a URL like `/account?id=1001` may retrieve the account details for a specific user. If the application fails to verify the requesters level of authorization, they could then increment or change the ID (e.g., `/account?id=1002`) to access someone else's data—resulting in a classic IDOR vulnerability.

In this example, we will examine a password manager application to demonstrate how an attacker could exploit an IDOR vulnerability to access another user's sensitive data or perform unauthorized actions on their behalf. Similar to the previous section, we will use Burp Suite to intercept the application's network traffic with the remote server. This time, however, our goal is to tamper with the data exchanged between the app and server in order to uncover and exploit IDOR issues.

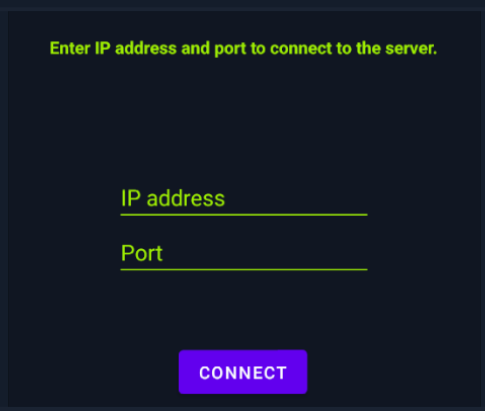
We will use an Android Virtual Device (AVD) for this exercise, although the process is compatible with any Android device, whether physical or emulated. Let's begin by connecting to the device via ADB and installing the application.

IDOR Attack

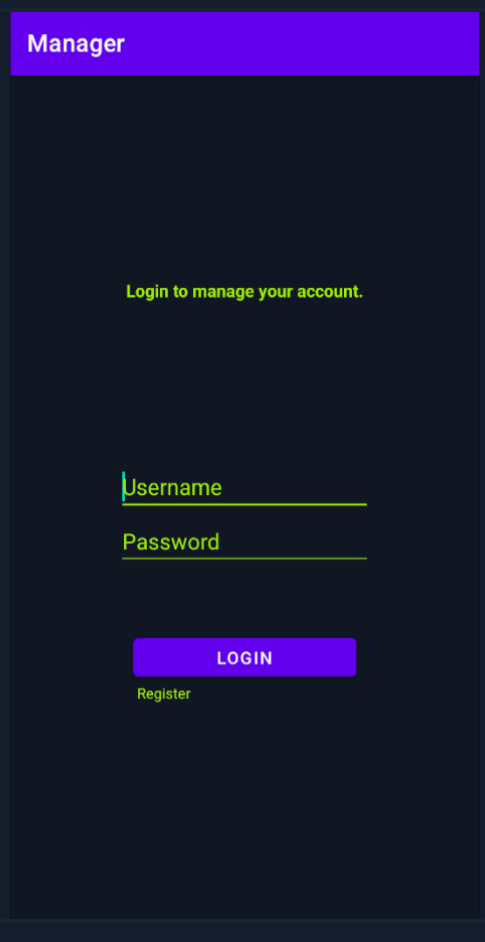
```
rl1k@htb[/htb]$ adb connect
rl1k@htb[/htb]$ adb install myapp.apk

Performing Streamed Install
Success
```

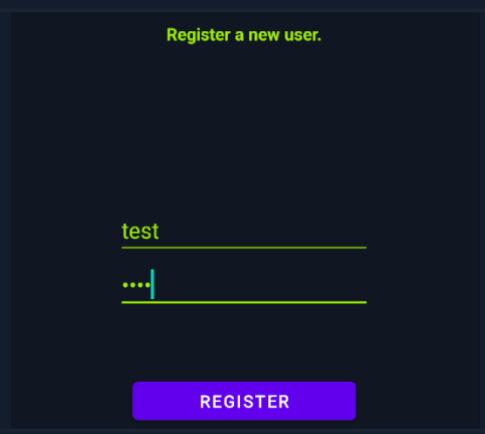
Upon launching the app, we are prompted to enter the remote server's IP and Port.



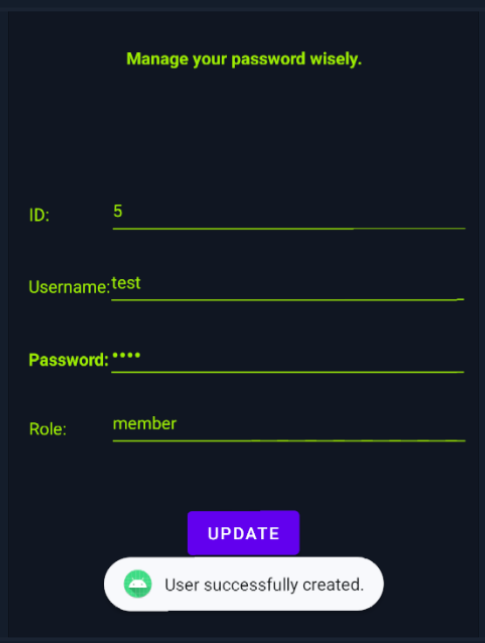
Once the IP and port are configured, a login screen is presented.



Currently, we do not have an account registed registered. Create a new one by tapping the `Register` link below the `LOGIN` button, and set the username and password as `test:test`.



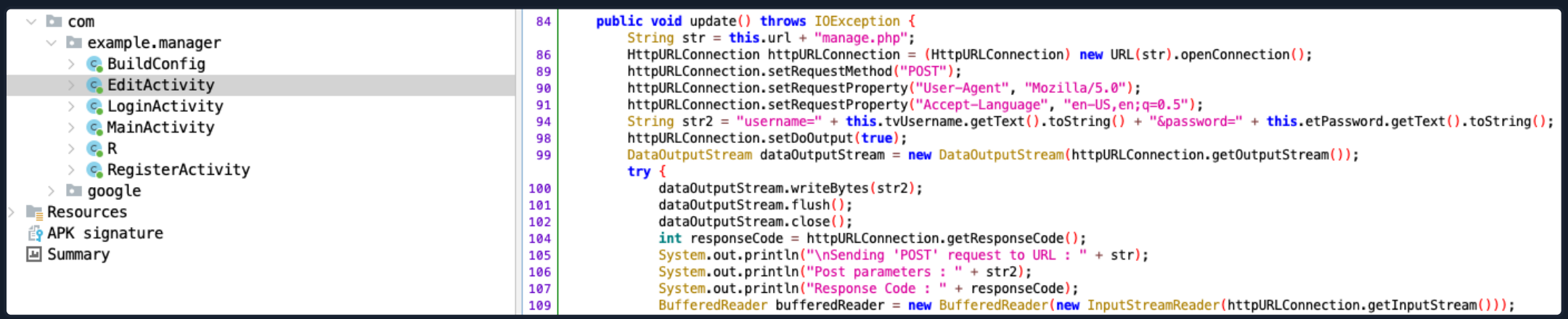
After tapping `REGISTER`, the user dashboard is displayed—along with a field for editing our password.



Let's use JADX to analyze the application's source code and understand how it handles user data when communicating with the remote server.

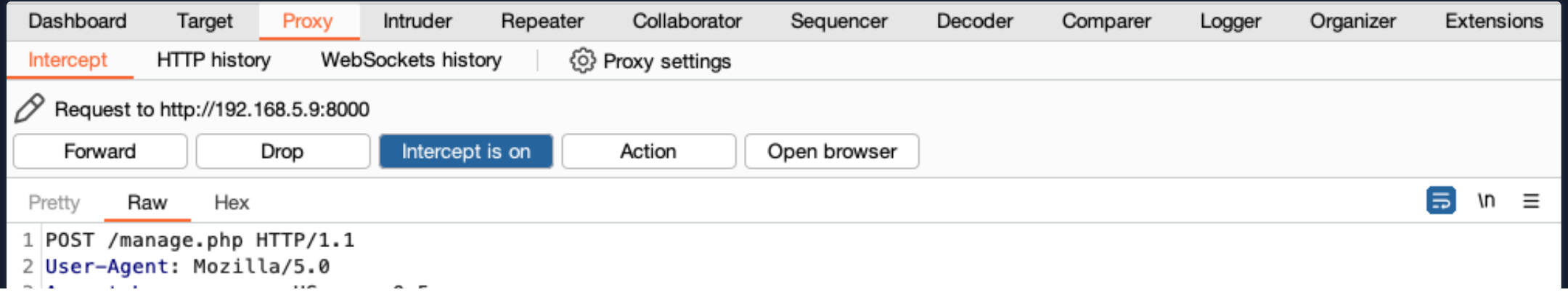
Code: bash

```bash
jadx-gui myapp.apk
```



Within the `EditActivity` class, the `update()` method shows that the app sends a POST request to `manage.php` with the parameters `"username"` and `"password"`. This suggests that the backend uses the `username` parameter to identify the account being updated. If true, this behavior introduces a serious IDOR vulnerability, as an attacker may be able change another user's password. By modifying the username field in the request, an malicious actor might reset the password for any existing user—without needing to authenticate.

To test this, we'll configure Burp Suite as a proxy to intercept the app's POST request and attempt to modify the username parameter. Setting this up involves the same steps outlined in the previous section: configuring Burp with the host machine's IP and port, updating the AVD's proxy settings, and enabling interception in Burp. Once the proxy is in place, tap the `UPDATE` button in the app. Return to Burp Suite, and you should see the captured POST request.



```
3 Accept-Language: en-US,en;q=0.5
4 Content-Type: application/x-www-form-urlencoded
5 Host: 192.168.5.9:8000
6 Connection: close
7 Accept-Encoding: gzip, deflate, br
8 Content-Length: 27
9
10 username=test&password=test
```

The request parameters appear as `username=test&password=test`. While one approach is to brute-force common usernames to discover valid accounts, we can start by testing the username `admin`. Modify the form data of the POST request to `username=admin&password=test`. If an admin user exists, this would reset their password to `test`. The modified request should look like this:

Dashboard | Target | Proxy | Intruder | Repeater | Collaborator | Sequencer | Decoder | Comparer | Logger | Organizer | Extensions

Intercept | HTTP history | WebSockets history | Proxy settings

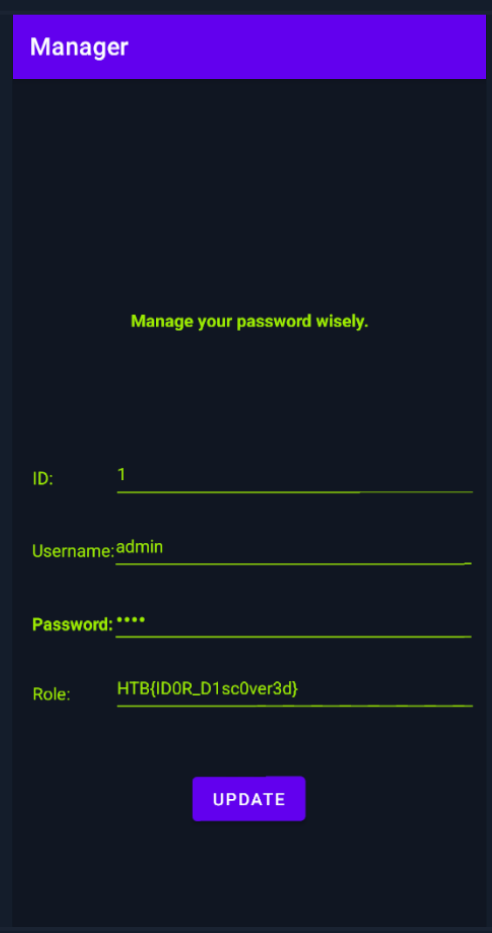
Request to http://192.168.5.9:8000

Forward | Drop | Intercept is on | Action | Open browser

Pretty | Raw | Hex

```
1 POST /manage.php HTTP/1.1
2 User-Agent: Mozilla/5.0
3 Accept-Language: en-US,en;q=0.5
4 Content-Type: application/x-www-form-urlencoded
5 Host: 192.168.5.9:8000
6 Connection: close
7 Accept-Encoding: gzip, deflate, br
8 Content-Length: 27
9
10 username=admin&password=test
```

Now, click on `Forward` to send the edited request. Back in the application, we are redirected to the login screen. Login using the credentials `admin/test`.



The attack is successful. We've gained access to the `admin` account and now have full ability to view and modify its data.

Connect to Pwnbox

Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

UK 28ms

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

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!

Target(s): Click here to spawn the target system!

+ 5 What is the Role of the admin user?

Submit your answer here...

+10 Streak pts Submit IDOR.zip

Previous Next

Cheat Sheet

Go to Questions

Table of Contents

- Hooking Java Methods
- Altering Method Values
- Hooking Native Methods
- Bypassing Detection Mechanisms
- Authentication Token Manipulation

Intercepting HTTP/HTTPS Requests

- Intercepting API Calls
- IDOR Attack
- SSL/TLS Certificate Pinning Bypass

Skills Assessments

- Skills Assessment

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left