Penetration Testing Report

Full Name : Sachin Madhumitha Sree D
Program : HCP-Penetration Testing Internship Week-2
Date : 24-02-2025

Introduction

In Week 2 of our security labs, we explored two critical web application vulnerabilities: **SQL Injection (SQLi)** and **Insecure Direct Object References (IDOR)**. These vulnerabilities pose serious risks to applications, leading to unauthorized data access, data manipulation, and privilege escalation. Understanding and exploiting these flaws in a controlled environment is essential for strengthening security defenses.

1.Objective

The objective of these labs is to identify, exploit, and understand the impact of SQL Injection and IDOR vulnerabilities. By performing hands-on exercises, we aim to enhance our penetration testing skills and learn mitigation techniques to secure web applications.

2.Scope

- **SQL Injection:** Exploiting vulnerable input fields to manipulate SQL queries and extract sensitive information.
- **IDOR:** Identifying flaws that allow unauthorized access to restricted resources by modifying parameters in requests.
- **Real-world scenarios:** Understanding how these vulnerabilities can be abused in live applications and how to prevent them.

Labs.Hacktify.in	**Insecure Direct Object References, SQL Injection.**

3.Summary

During this week's labs, we successfully exploited **12 SQL Injection vulnerabilities** that allowed us to extract login credentials and database details. Additionally, we demonstrated **IDOR attacks**, where direct object references were modified to access unauthorized user data. These exercises reinforced the importance of **input validation, least privilege principles, and secure coding practices** to mitigate such threats.

Total number of Sub-labs: 16 Sub-lab

High	Medium	Low
05	**06**	**05**

High - **Five Sub-labs with hard difficulty level.**

Medium - **Six of Sub-labs with Medium difficulty level.**

Low - **Five of Sub-labs with Easy difficulty level.**

1. Insecure Direct Object References

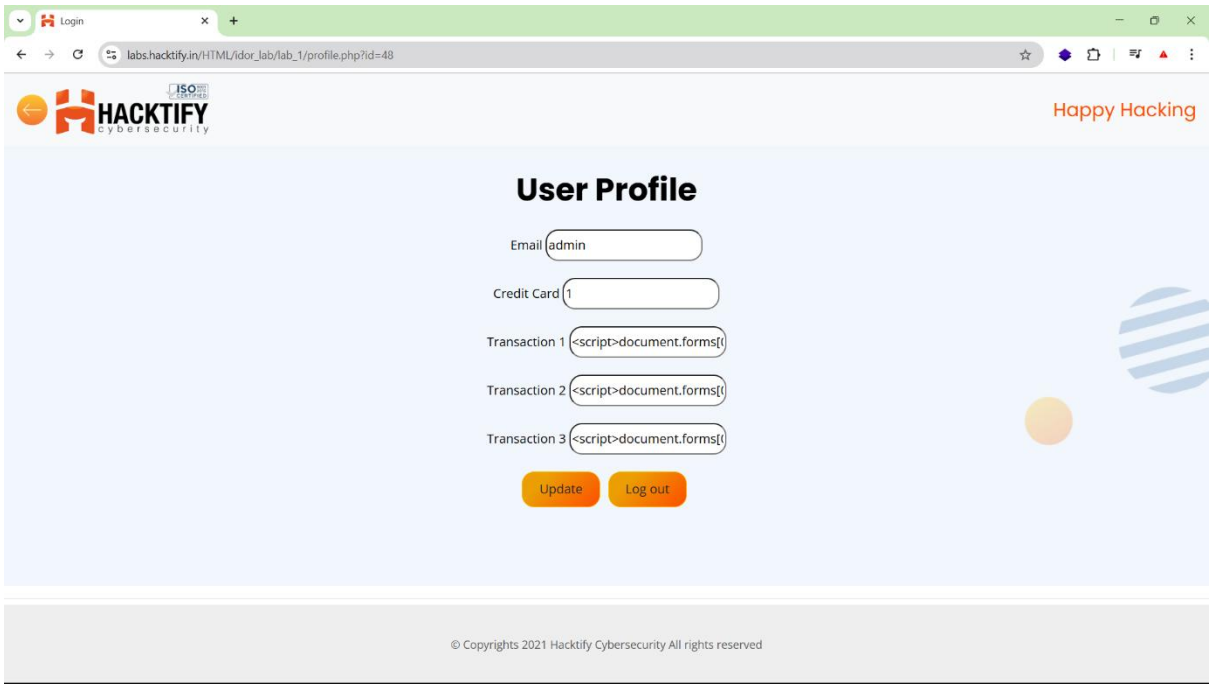
1.1. Give Me My Amount!!

Reference	**Risk Rating**
Give Me My Amount!!	**Low**
Tools Used	
Burp Suite	
Vulnerability Description	
Insecure Direct Object Reference (IDOR) allows unauthorized access to other users' data by manipulating request parameters.	
How It Was Discovered	

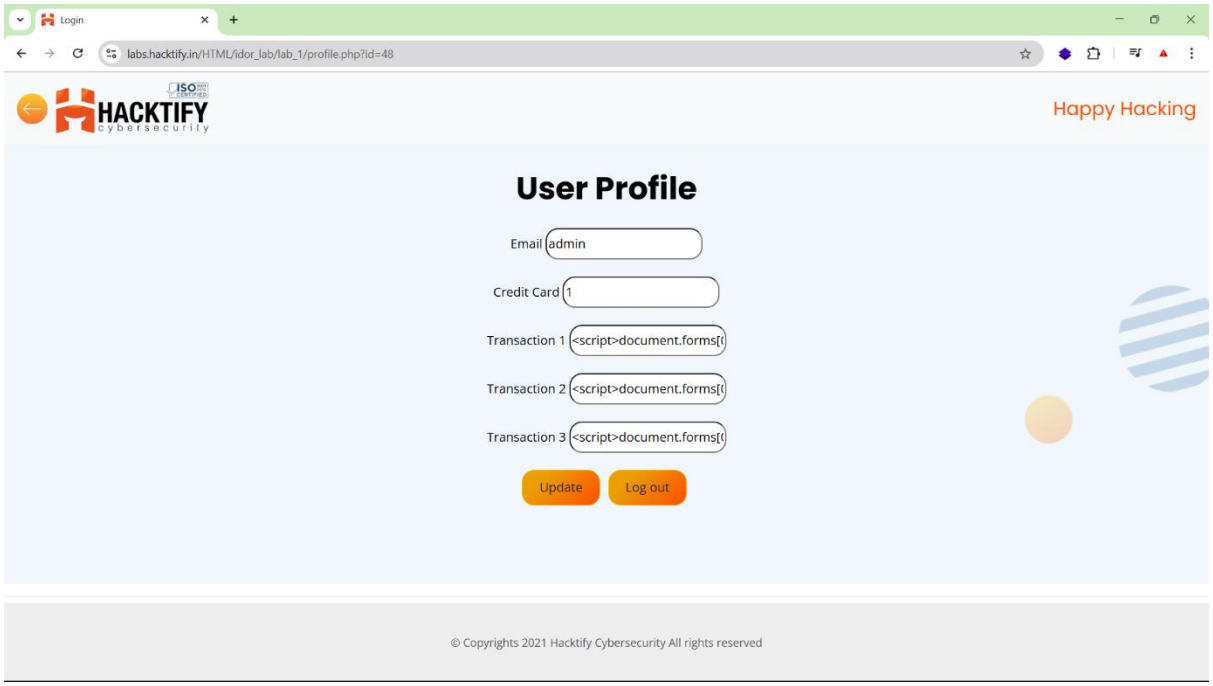
Manually changing the id parameter in the URL granted access to another user's account.
Vulnerable URLs
https://labs.hacktify.in/HTML/idor_lab/lab_1/profile.php?id=48
Consequences of not Fixing the Issue
-Unauthorized access to sensitive user information. -Potential data breaches and compliance violations.
Suggested Countermeasures
-Implement proper access control checks on the server side. -Use session-based authentication instead of exposing object references.
References
OWASP IDOR Guide.

Proof of Concept :

1.Create account and login.



2.After login I change the id parameter value into 48 then I access someone account.



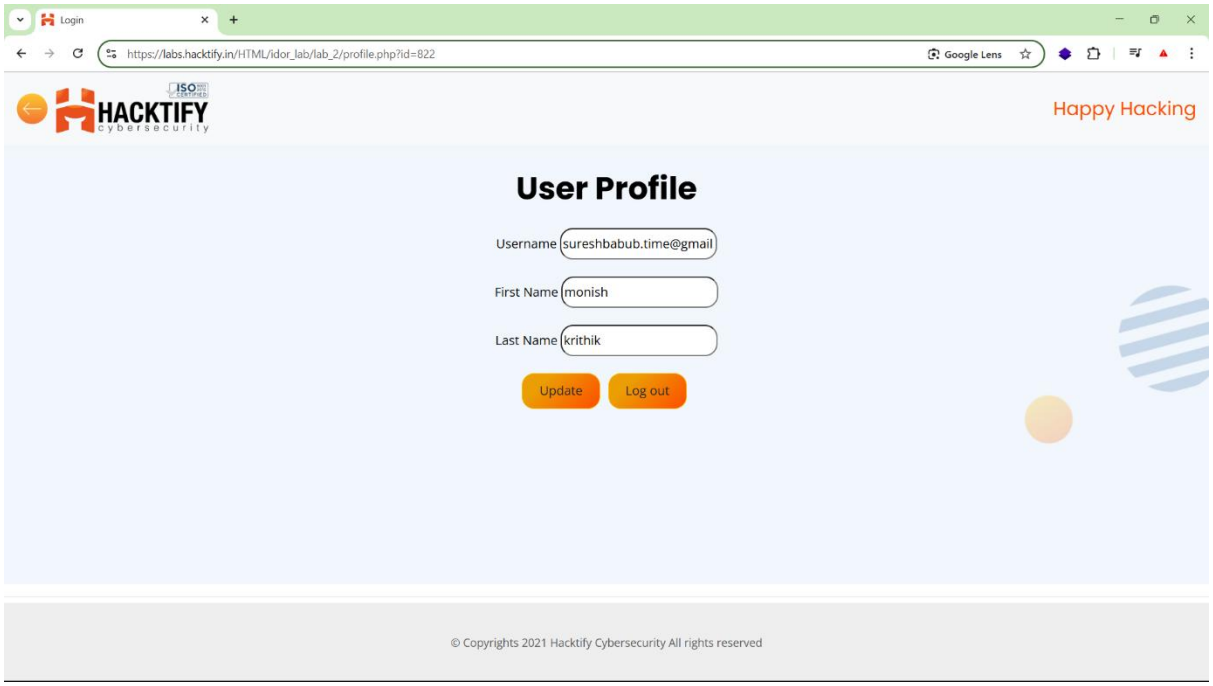
1.2. Stop Polluting My Params!

Reference	**Risk Rating**
Stop Polluting My Params!	**Medium**
Tools Used	
Burp Suite	
Vulnerability Description	
Insecure Direct Object Reference (IDOR) allows unauthorized access to other users' accounts and modification of their personal information.	
How It Was Discovered	
Manually changing the id parameter in the URL allowed access to another user's profile, enabling modification of username, first name, and last name.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_2/profile.php?id=24	
Consequences of not Fixing the Issue	
Attackers can modify user data, impersonate other users, and cause data integrity issues.	

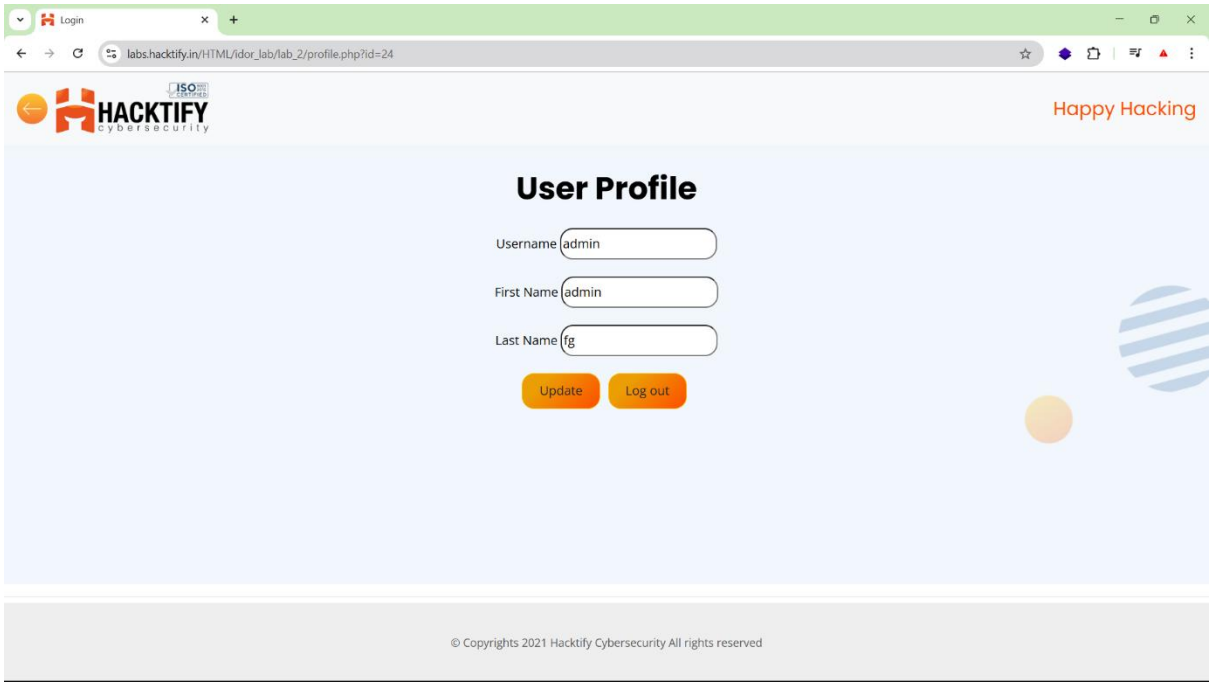
Suggested Countermeasures
Implement proper authorization checks at the server level. Use session-based authentication and avoid exposing direct object references.
References
OWASP IDOR Guide

Proof of Concept :

1.Create account and login.



2.After login I change the id parameter value into 24 then It allowed access to another user's profile, enabling modification of username, first name, and last name.



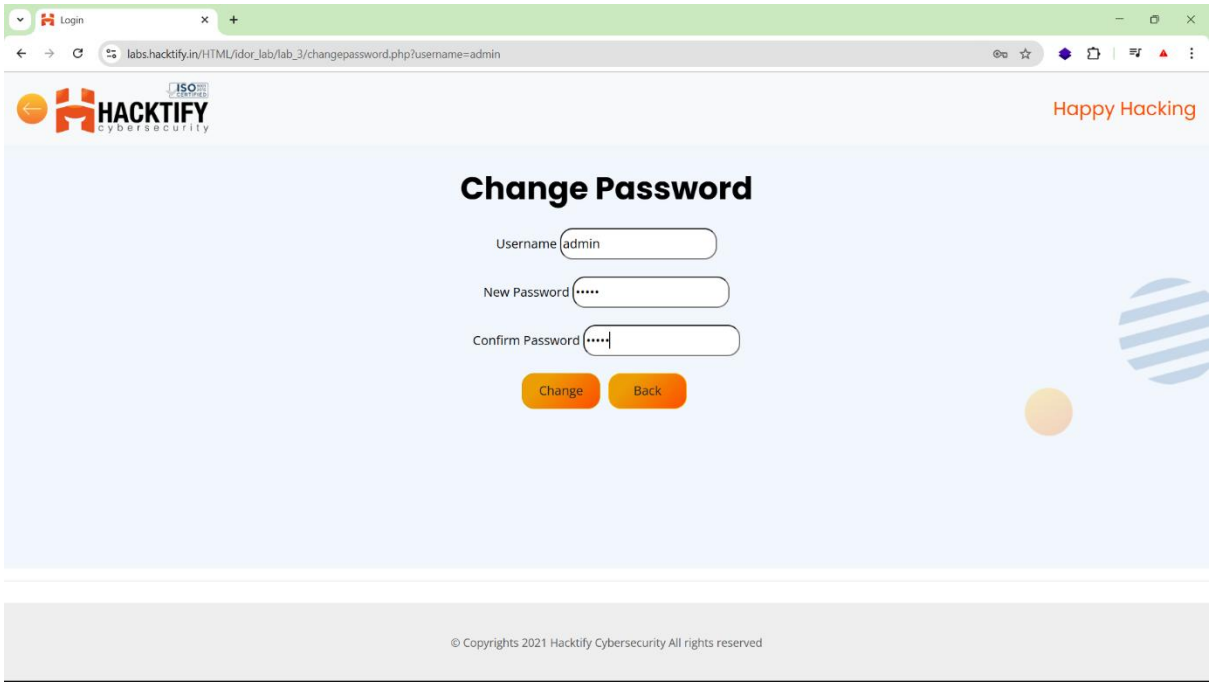
1.3. Someone Changed My Password!

Reference	**Risk Rating**
Someone Changed My Password!	**Medium**
Tools Used	
Burp Suite, Web Browser	
Vulnerability Description	
Insecure Direct Object Reference (IDOR) allows an attacker to change another user's password by modifying the username parameter in the password change request.	
How It Was Discovered	
After logging in, changing the username parameter in the password reset request granted access to another user's account. It is p3 vulnerable.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_3/changepassword.php?username=sureshbabub.time@gmail.com	

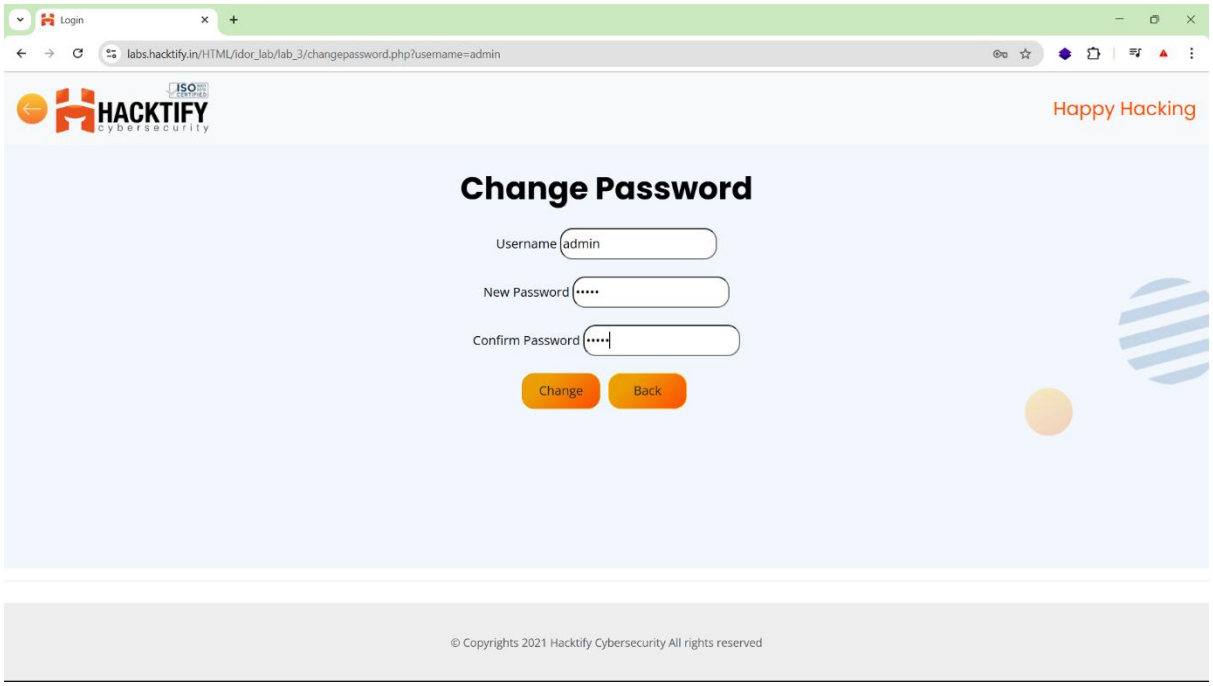
Consequences of not Fixing the Issue
Attackers can reset passwords, take over accounts, and lock out legitimate users.
Suggested Countermeasures
Enforce authentication checks on sensitive operations. Validate user identity before allowing password changes.
References
OWASP IDOR Guide.

Proof of Concept :

1.Create account and login.



2. After login to click change password ,then I found the username parameter to change admin name to go and change password.



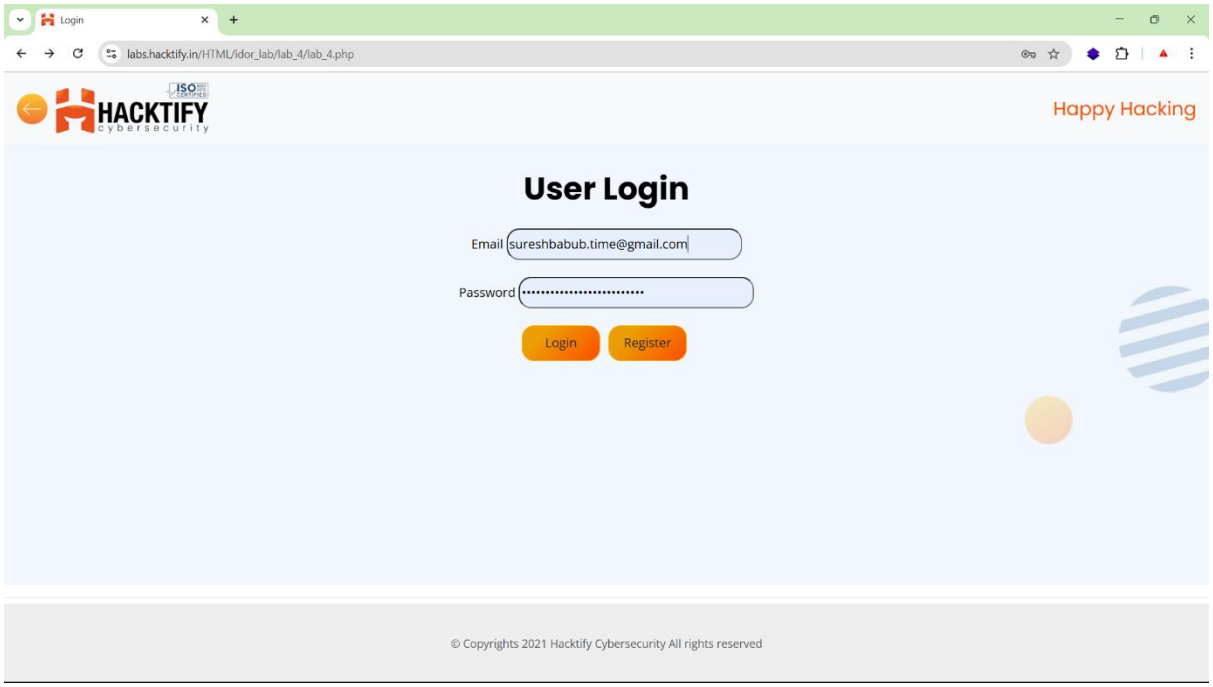
1.4. Change Your Methods!

Reference	**Risk Rating**
Change Your Methods!	**Hard**
Tools Used	
Burp Suite, Web Browser	
Vulnerability Description	
Insecure Direct Object Reference (IDOR) allows an attacker to modify another user's account details by changing the id parameter in the request. Although there is no page reflection, the changes are stored in the database.	
How It Was Discovered	
After registering and logging in, modifying the id parameter granted access to another user's profile, allowing changes to their username and password.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_4/profile.php?id=2223	
Consequences of not Fixing the Issue	

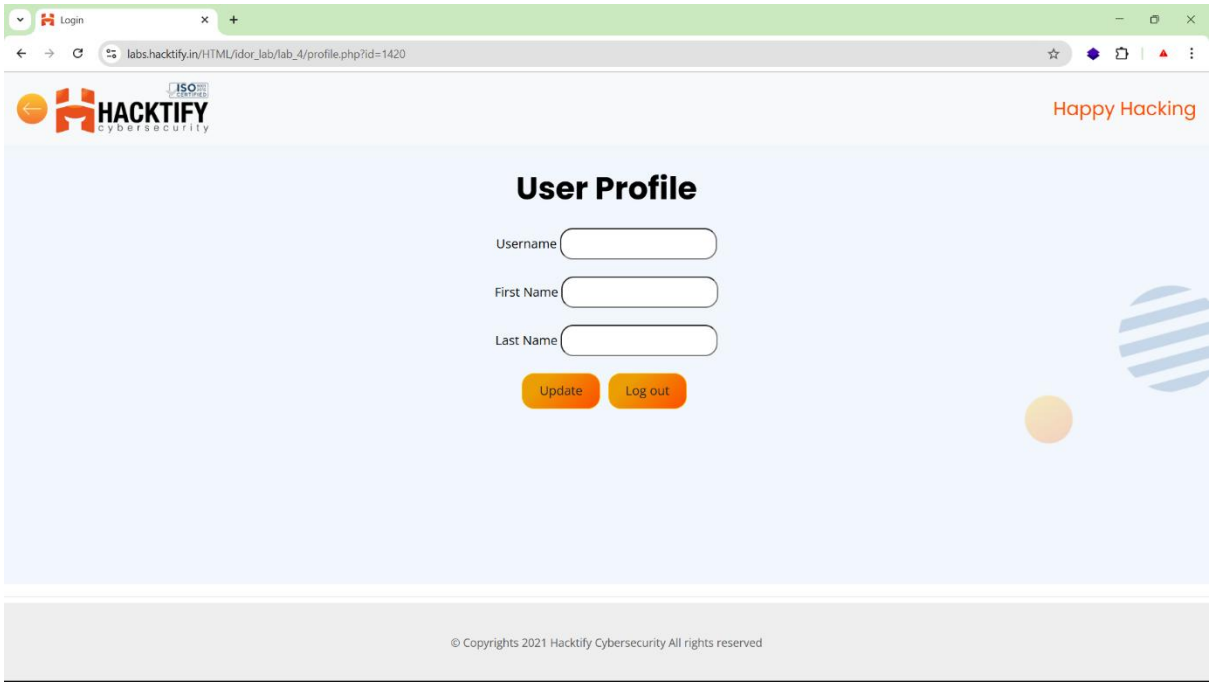
Attackers can hijack user accounts by modifying credentials, leading to unauthorized access.
Suggested Countermeasures
Implement proper authentication checks before updating user details. Use session-based verification instead of user-controlled parameters.
References
OWASP IDOR Guide

Proof of Concept :

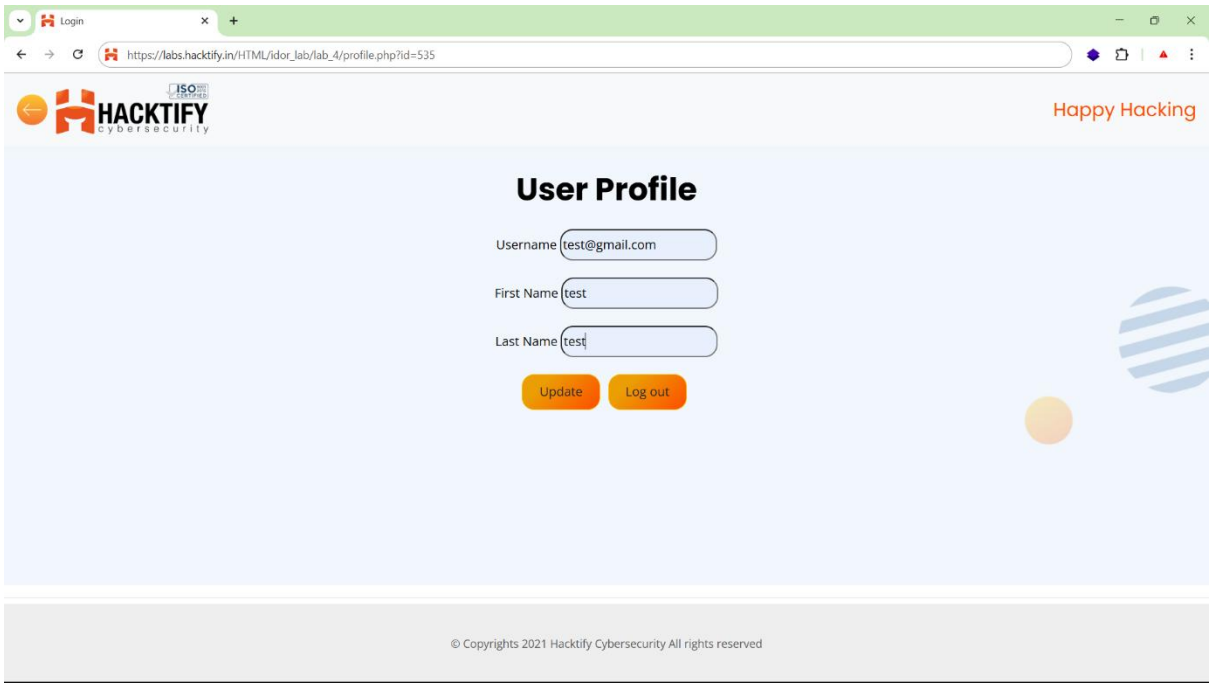
1.Create account and login.



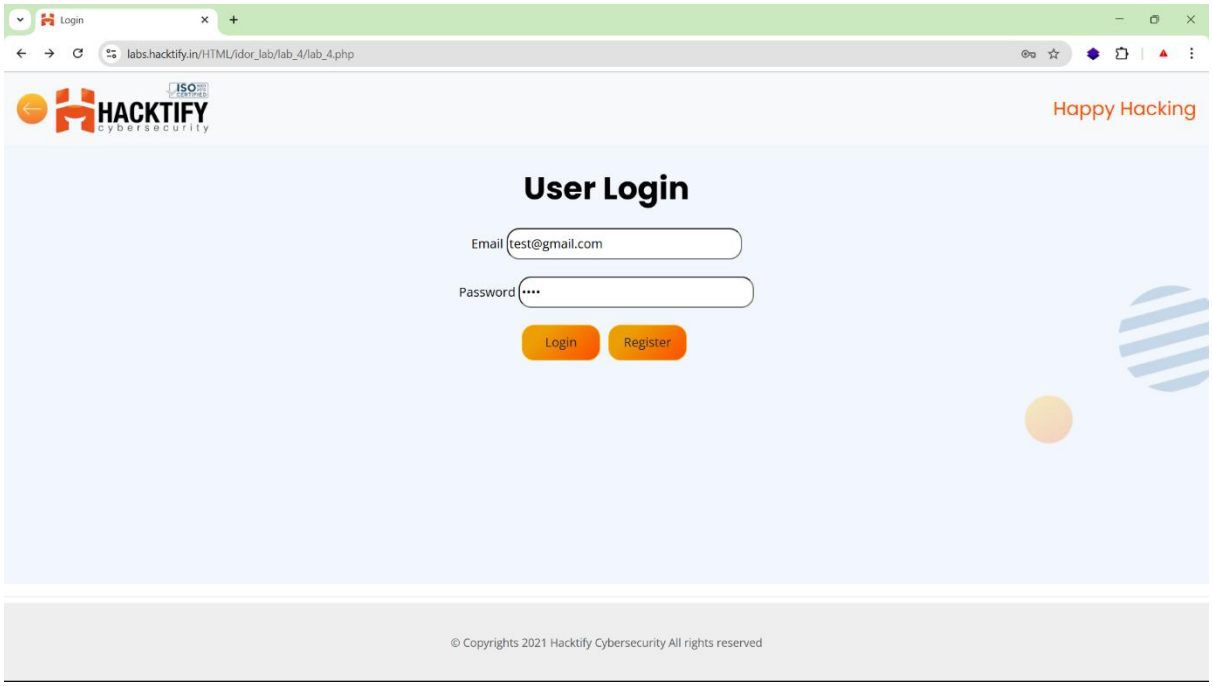
2.To see the id parameter on URL and try to change the id value ,accidently there is no data for my user profile login.



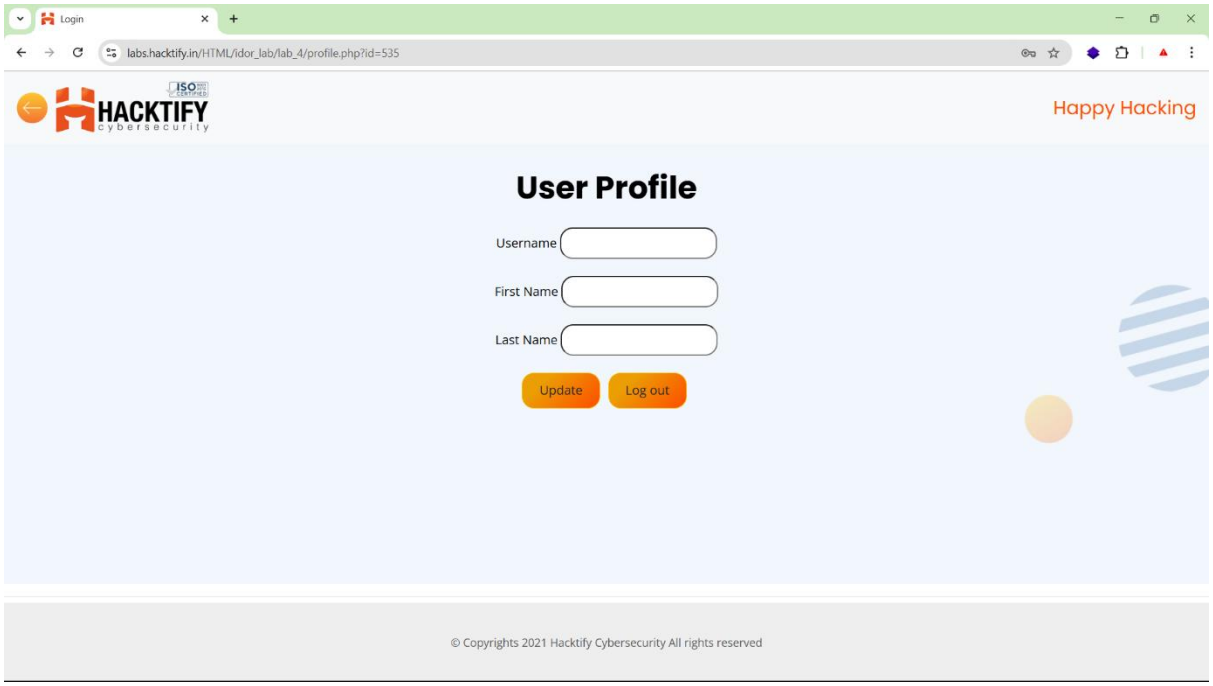
3.So I decide to change the id parameter value into 535.



4.Finally ,the data was changed successfully and I login into the other account.



5.It works there the IDOR vulnerability ,But it does’t show the data entry and all users data here this also be vulnerable.



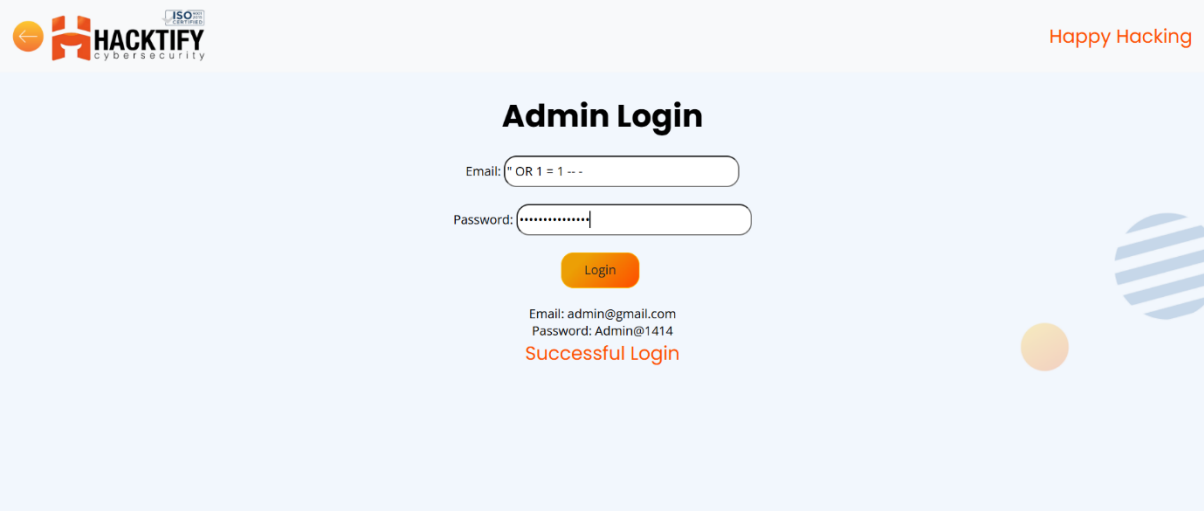
6. This last IDOR Lab namr is change your method but there is no another option like get method to post method.

2. SQL Injection

2.1. String & Error Part 1!

Reference	**Risk Rating**
String & Error Part 1!	**Low**
Tools Used	
Burp Suite	
Vulnerability Description	
SQL Injection (SQLi) allows an attacker to bypass authentication by injecting malicious SQL statements into input fields.	
How It Was Discovered	
During testing, using the payload 1' OR '1'='1 -- in the email field allowed successful login as an admin without valid credentials.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
Attackers can gain unauthorized access to accounts, extract sensitive database information, or modify records.	
Suggested Countermeasures	
Use prepared statements and parameterized queries to prevent SQL injection. Implement input validation and escape special characters.	
References	
OWASP SQL Injection Guide	

Proof of Concept :

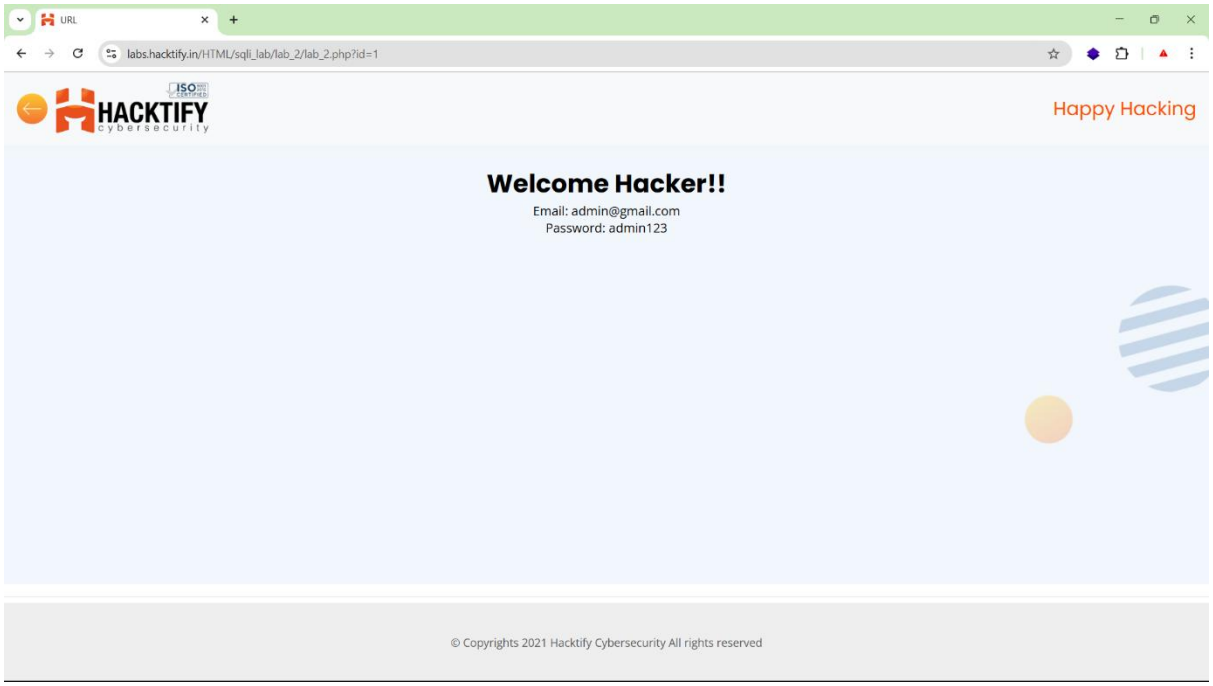


2.2. String & Error Part 2!

Reference	**Risk Rating**
String & Error Part 2!	**Low**
Tools Used	
Burp Suite	
Vulnerability Description	
SQL Injection (SQLi) allows an attacker to manipulate database queries by injecting malicious SQL code into the id parameter.	
How It Was Discovered	
By modifying the id parameter in the URL and injecting an SQL payload, the attacker was able to retrieve admin credentials	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_2/lab_2.php	

Consequences of not Fixing the Issue
Attackers can extract sensitive information from the database, modify or delete records, and compromise user accounts.
Suggested Countermeasures
Use prepared statements and parameterized queries. Validate and sanitize user input to prevent SQL injection attacks.
References
OWASP SQL Injection Guide

Proof of Concept :

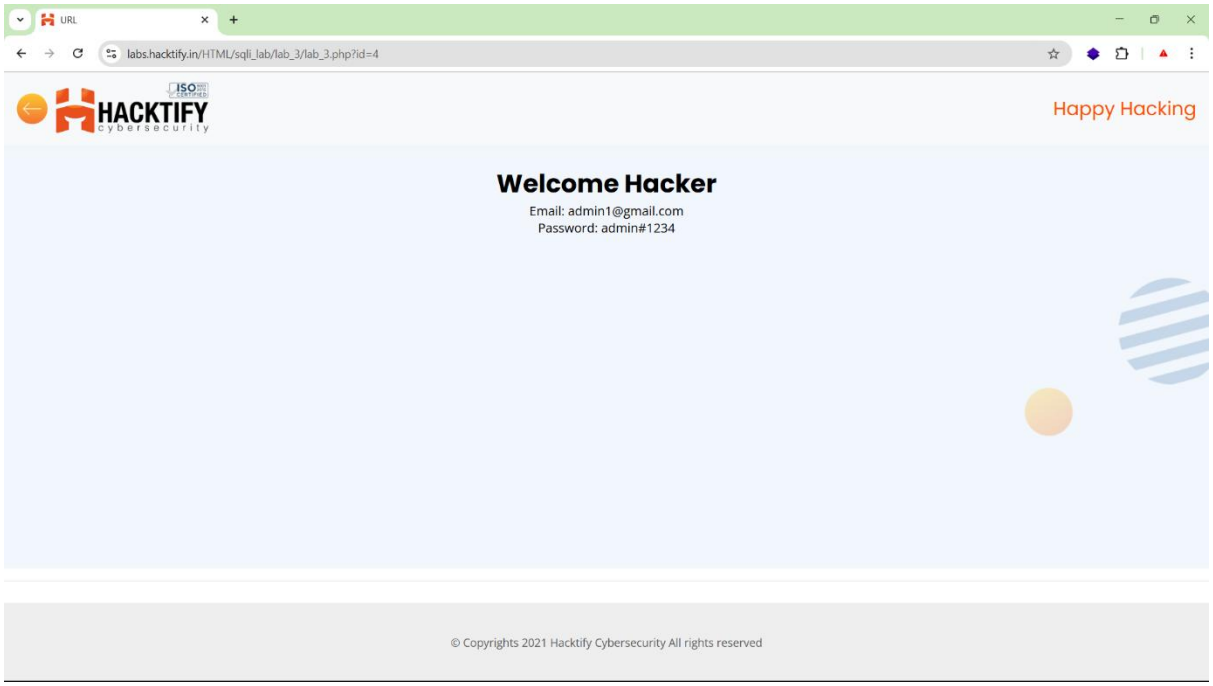


Note : If you the id parameter there also be listed the admin email and password ,it is looks like IDOR but here sql payload supported.

2.3. String & Error Part 3!

Reference	**Risk Rating**
String & Error Part 2!	**Low**
Tools Used	
Manual Injection, Burp Suite	
Vulnerability Description	
The application is vulnerable to SQL Injection via the id parameter. By injecting 1' OR '1'='1, the database returns all user records instead of just a specific one.	
How It Was Discovered	
By modifying the URL and injecting SQL payloads, the system exposed admin credentials.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_3/lab_3.php?id=4	
Consequences of not Fixing the Issue	
-Unauthorized access to admin accounts -Potential exposure of all database records -Privilege escalation and full database compromise	
Suggested Countermeasures	
-**Use Prepared Statements & Parameterized Queries:** Prevent user inputs from altering SQL queries. -**Sanitize and Validate Inputs:** Reject special characters like ' and --. -**Implement Web Application Firewall (WAF):** Block SQL injection attempts.	
References	
OWASP SQL Injection Guide	

Proof of Concept :

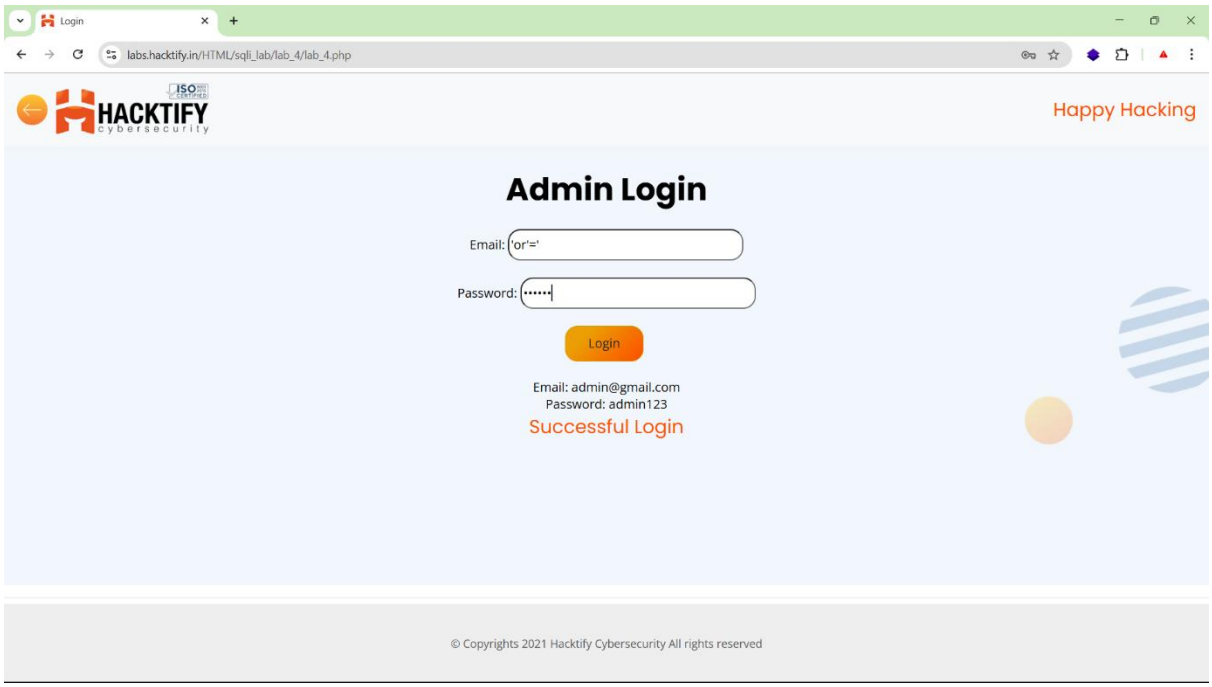


2.4. Let's Trick 'em!

Reference	**Risk Rating**
Let's Trick 'em!	**Medium**
Tools Used	
Burp Suite, Browser Developer Tools	
Vulnerability Description	
SQL Injection (SQLi) allows an attacker to manipulate database queries by injecting malicious SQL code into the email and password fields. This could bypass authentication and grant unauthorized access	
How It Was Discovered	
By entering 'or'=' into both the email and password fields, the application validated the condition as true ('or'=' always evaluates to true in SQL logic), leading to authentication bypass.	
Vulnerable URLs	

https://labs.hacktify.in/HTML/sqli_lab/lab_4/lab_4.php
Consequences of not Fixing the Issue
-Unauthorized access to user/admin accounts -Exposure of sensitive data (emails, passwords, financial details, etc.) -Potential database modification or deletion -Risk of full system compromise if chained with other vulnerabilities
Suggested Countermeasures
-Implement **prepared statements** and **parameterized queries** to prevent SQL injection. -Use **input validation** to allow only expected values (e.g., valid email formats). -**Escape special characters** to neutralize SQL commands injected by attackers. -Employ **web application firewalls (WAF)** to detect and block malicious input attempts.
References
-OWASP SQL Injection Guide -MITRE CWE-89: SQL Injection

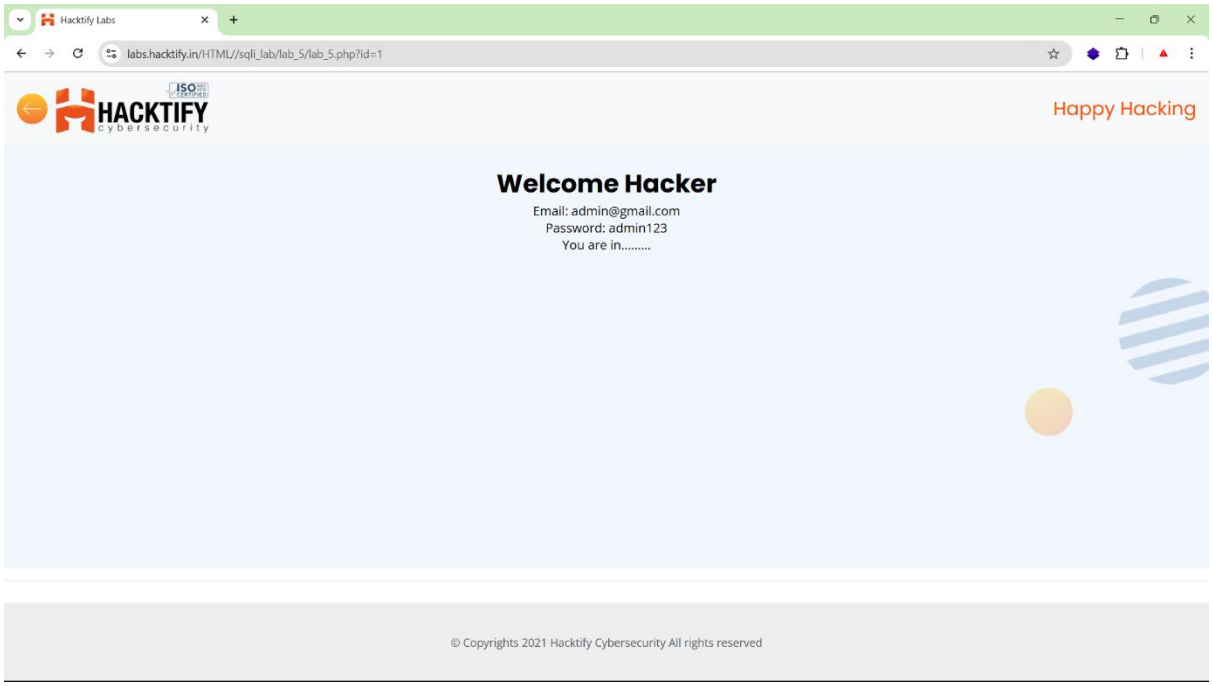
Proof of Concept :



2.5. Booleans Are Blind!

Reference	**Risk Rating**
Booleans Are Blind!	**Hard**
Tools Used	
Manual Testing	
Vulnerability Description	
The application exposes sensitive user information by allowing direct access to database records via the id parameter in the URL.	
How It Was Discovered	
-Changed id=1 in the URL without injecting any SQL payload. -The application displayed admin credentials (admin@gmail.com / admin123).	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
-Unauthorized users can access sensitive data of other users. -Attackers can enumerate id values to extract multiple user credentials. -It can lead to a full account takeover and privacy breaches.	
Suggested Countermeasures	
=Implement proper authentication and authorization checks. -Use session-based identifiers instead of numeric id values in URLs. -Apply access controls to prevent unauthorized access to user data.	
References	
OWASP Insecure Direct Object References (IDOR) Guide	

Proof of Concept :

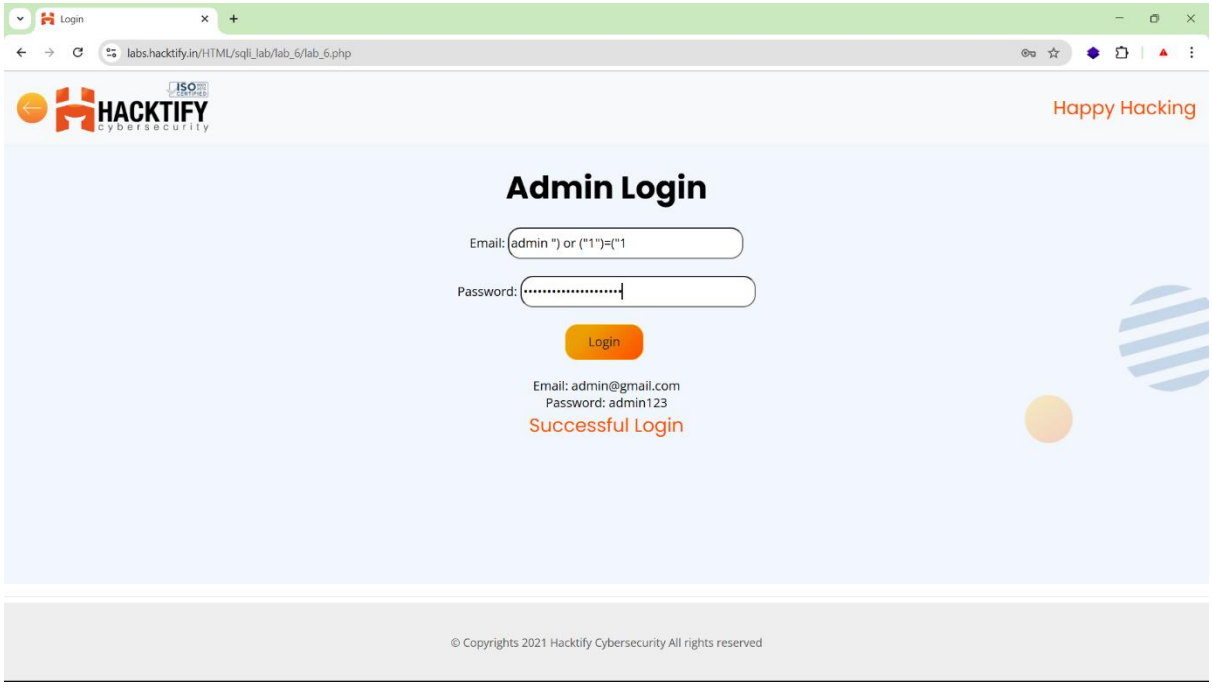


2.6. Error Based : Tricked.

Reference	**Risk Rating**
Error Based : Tricked	**Medium**
Tools Used	
Manual Testing	
Vulnerability Description	
The login form is vulnerable to SQL Injection, allowing an attacker to bypass authentication. By injecting SQL code into the email field, the attacker forces the query to always return true, granting unauthorized access.	
How It Was Discovered	
The payload admin") or ("1"="1 was used in the email field, bypassing authentication and revealing admin credentials.	

Vulnerable URLs
https://labs.hacktify.in/HTML/sqli_lab/lab_6/lab_6.php
Consequences of not Fixing the Issue
Attackers can gain unauthorized admin access, steal sensitive data, and manipulate the database.
Suggested Countermeasures
Implement prepared statements and parameterized queries. Validate and sanitize user inputs to prevent SQL Injection.
References
OWASP SQL Injection Guide

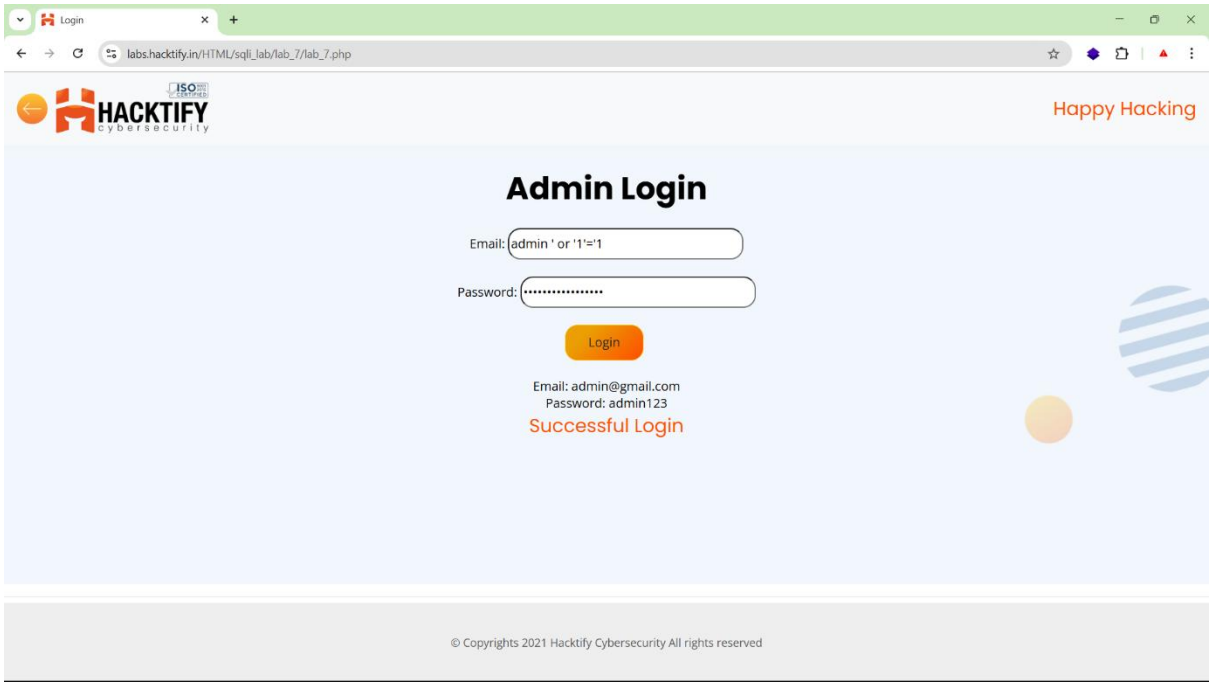
Proof of Concept :



2.7. Errors And Post!

Reference	**Risk Rating**
Errors And Post!	**Low**
Tools Used	
Manual Testing	
Vulnerability Description	
The login form is vulnerable to SQL Injection, allowing an attacker to bypass authentication. The payload used, admin ' or '1'='1, manipulates the SQL query logic, making the condition always true, resulting in a successful login.	
How It Was Discovered	
The SQL payload was inserted into the email field, and authentication was bypassed, revealing admin credentials.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_7/lab_7.php	
Consequences of not Fixing the Issue	
Attackers can log in as an admin without valid credentials, access sensitive data, and modify or delete database records.	
Suggested Countermeasures	
Use prepared statements and parameterized queries. Validate and sanitize all user inputs to prevent SQL Injection.	
References	
OWASP SQL Injection Guide	

Proof of Concept :

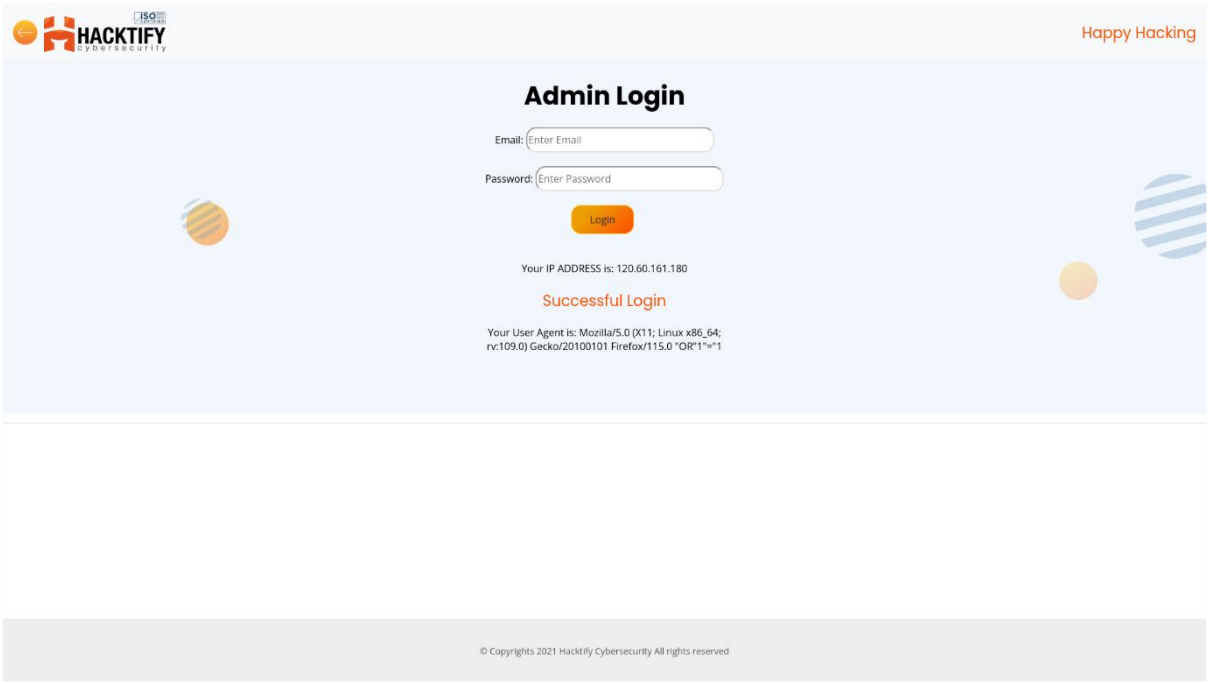


2.8. User Agents Lead us!

Reference	**Risk Rating**
User Agents Lead us!	**Hard**
Tools Used	
Burp Suite	
Vulnerability Description	
The application fails to properly sanitize input from the HTTP User-Agent header, making it vulnerable to SQL Injection. By injecting " OR "1"="1 in the User-Agent field, authentication was bypassed, leading to a successful login.	
How It Was Discovered	
The error message from the response revealed an injectable point. By modifying the User-Agent header in Burp Suite with a SQL payload, authentication was bypassed.	

Vulnerable URLs
https://labs.hacktify.in/HTML/sqli_lab/lab_8/lab_8.php
Consequences of not Fixing the Issue
Attackers can manipulate server-side SQL queries, gain unauthorized access, extract sensitive data, and potentially escalate privileges.
Suggested Countermeasures
Validate and sanitize all HTTP headers, especially User-Agent. Use parameterized queries and avoid direct concatenation of user input in SQL statements.
References
OWASP SQL Injection Guide

Proof of Concept :

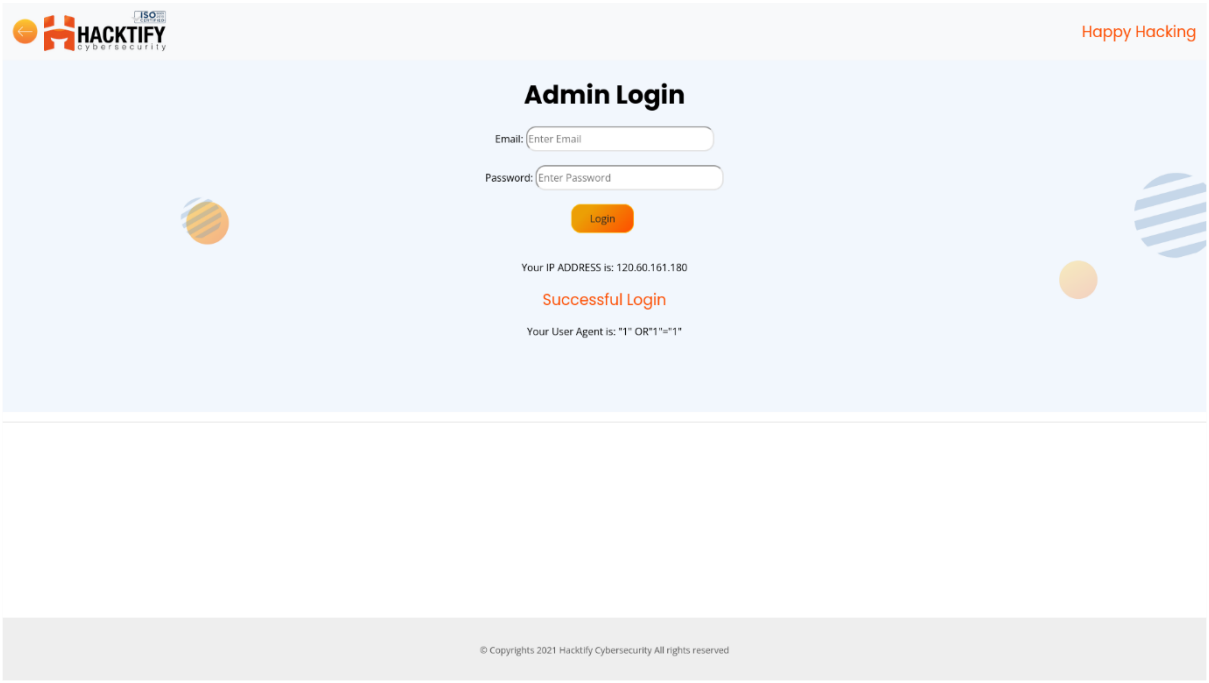


2.9. Referer Lead us!

Reference	**Risk Rating**
Referer Lead us!	**Medium**
Tools Used	
Burp Suite	
Vulnerability Description	
-The application is vulnerable to SQL injection through the HTTP User-Agent header. -By injecting "1" OR "1"="1 into the User-Agent field, authentication was bypassed, leading to a **Successful Login** message. -This indicates that the User-Agent input is directly included in an SQL query without proper sanitization.	
How It Was Discovered	
-Identified a reference to the User-Agent in the response. -Used Burp Suite to modify the User-Agent header and injected a SQL payload ("1" OR "1"="1"). -The system granted unauthorized access.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
-**Unauthorized access**: Attackers can bypass authentication and gain admin privileges. -**Data exposure**: Malicious users can extract sensitive information from the database. -**System compromise**: Attackers could manipulate SQL queries to execute arbitrary commands.	
Suggested Countermeasures	
✅ **Sanitize and validate input:** Ensure all user inputs, including HTTP headers, are sanitized. ✅ **Use prepared statements and parameterized queries** instead of concatenating input directly into SQL statements. ✅ **Implement web application firewalls (WAFs)** to detect and block malicious input.	

✅ **Enable least privilege access:** Ensure that the application uses a database user with minimal permissions.
References
-OWASP SQL Injection Guide

Proof of Concept :



2.10. Oh Cookies!

Reference	**Risk Rating**
Oh Cookies!	**Hard**
Tools Used	
Browser Developer Tools (Inspect Element - Storage)	
Vulnerability Description	
-The application stores the username in cookies for authentication.	

-By modifying the **username cookie** value in **Storage** (under Developer Tools), it is possible to inject a **UNION-based SQL injection payload** to extract database information.

-The payload used : union SELECT version(), user(), database()#

-This resulted in retrieving sensitive database details such as:

- **Database version**
- **Current user**
- **Current database name**

How It Was Discovered

-**Opened Developer Tools** (Inspect Element → Application → Storage → Cookies).

-Located the username cookie.

-Replaced the cookie value with union SELECT version(), user(), database()#

-Refreshed the page and observed the database details displayed.

Vulnerable URLs

https://labs.hacktify.in/HTML/sqli_lab/lab_10/lab_10.php

Consequences of not Fixing the Issue

Database Enumeration: Attackers can extract database version, usernames, and structure.

Privilege Escalation: If the database user has high privileges, attackers might gain **administrative access**.

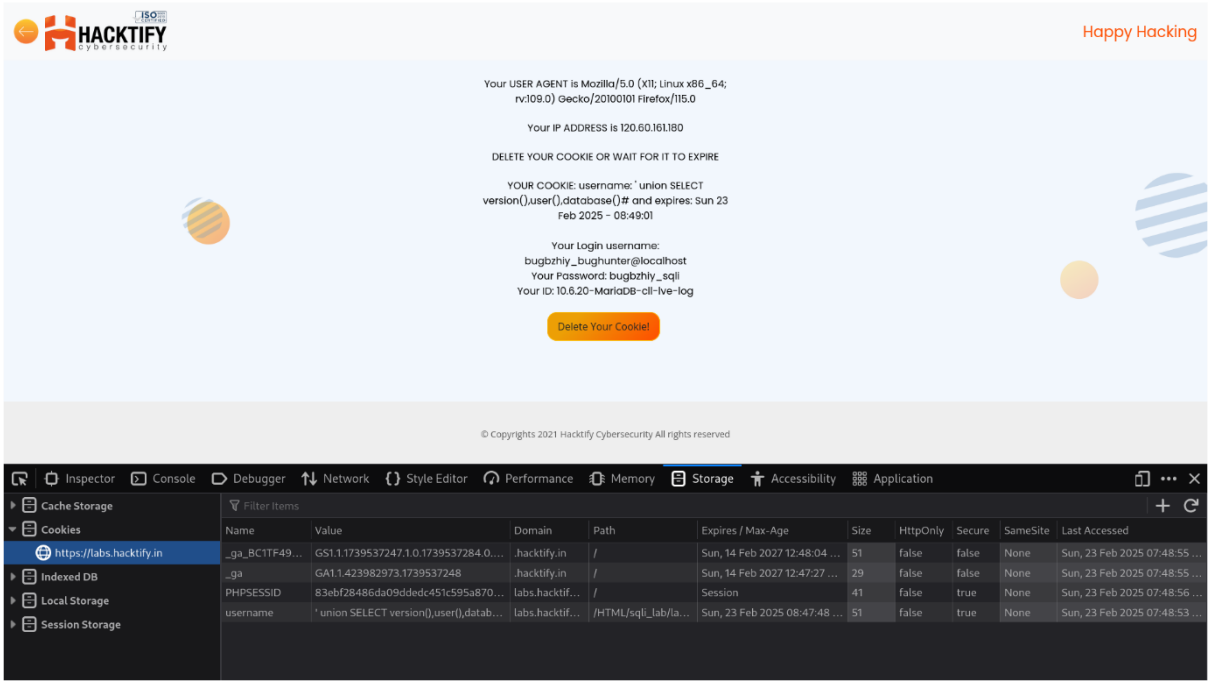
Potential Full Database Compromise: If further exploitation is possible, the entire database could be dumped.

Suggested Countermeasures

✅ **Use Secure Session Management:** Store authentication data on the server-side instead of client-side cookies.
✅ **Implement Input Validation:** Prevent direct SQL execution by **sanitizing cookie input**.
✅ **Use Prepared Statements:** Ensure **parameterized queries** are used to prevent SQL injection.
✅ **Enable HTTP-Only and Secure Flags for Cookies:** To prevent client-side manipulation.
✅ **Monitor and Log Cookie Changes:** Track suspicious cookie modifications in logs.

References
-OWASP SQL Injection Guide -SQL Injection Cheat Sheet -How to Secure Cookies

Proof of Concept



2.11. WAF's Are Injected!

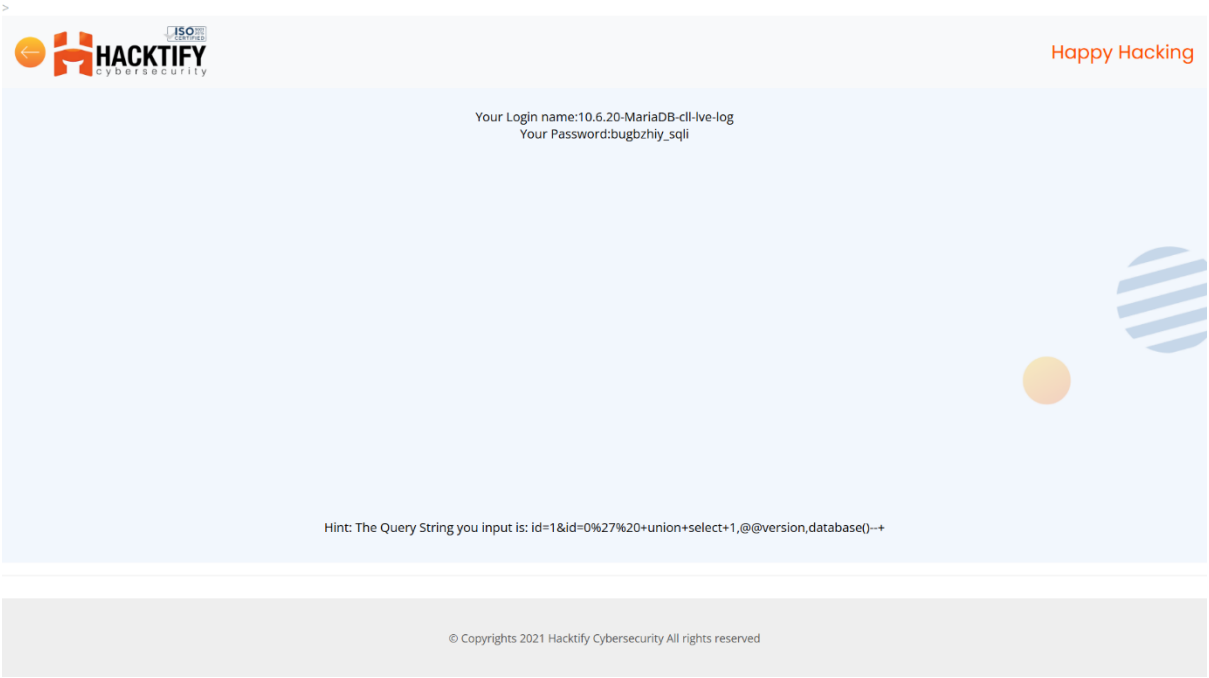
Reference	Risk Rating
WAF's Are Injected!	**Hard**
Tools Used	
Web Browser, Burp Suite	
Vulnerability Description	

The application is vulnerable to SQL Injection via the id parameter in the URL. By injecting a **UNION SELECT** statement, an attacker can retrieve sensitive database information. The following payload was used to extract the database version and name : id=1&id=0' union select 1,@@version,database()--+
How It Was Discovered
While testing input manipulation in the URL, an additional id parameter was added. The server response revealed critical database details, confirming a SQL Injection vulnerability.
Vulnerable URLs
https://labs.hacktify.in/HTML/sqli_lab/lab_11/lab_11.php
Consequences of not Fixing the Issue
-Attackers can retrieve database version and structure. -Sensitive user credentials and other stored data can be extracted. -Further exploitation may allow privilege escalation and full database compromise.
Suggested Countermeasures
-Use **prepared statements** (parameterized queries) instead of direct user input concatenation. -Implement **server-side input validation** to filter out malicious queries. -Enforce **least privilege principle** for database users to limit damage if exploited. -Monitor and log SQL errors for signs of exploitation attempts.
References
OWASP SQL Injection Guide

Proof of Concept :

1.Here the following payload was used to extract the database version and name : id=1&id=0' union select 1,@@version,database()--+

2.Now I get the user name and password.



2.12. WAF's Are Injected part 2!

Reference	**Risk Rating**
WAF's Are Injected part 2!	**Medium**
Tools Used	
Web Browser, Burp Suite	
Vulnerability Description	
The application is vulnerable to SQL Injection through the id parameter in the URL. By injecting a crafted SQL payload, an attacker was able to extract administrator credentials. The following SQL payload was used to retrieve the admin username and password : id=14 UNION SELECT 1,username,password FROM users--+	
How It Was Discovered	
By modifying the id parameter in the URL and injecting SQL queries, the server responded with sensitive information, revealing administrator credentials.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_12/lab_12.php	

Consequences of not Fixing the Issue
-Attackers can extract user credentials, including admin login details. -Full database compromise may be possible, leading to data leaks. -The attacker may escalate privileges and gain control over the application.
Suggested Countermeasures
-Implement **prepared statements (parameterized queries)** to prevent SQL Injection. -Validate and sanitize user input to reject malicious queries. -Apply **least privilege principle** to database users to limit data exposure. -Implement Web Application Firewalls (WAF) to detect and block SQL Injection attempts. -Regularly monitor and audit logs for suspicious SQL activity.
References
OWASP SQL Injection Guide

Proof of Concept :