

# Week 2 Penetration Testing Report

**Full Name:** Atharva Jagdale

**Program:** HCPT

**Date:** 24-02-2025

## Introduction

This report hereby describes the proceedings and results of a Black Box security assessment conducted against **Week 2 Labs**. The report hereby lists the findings and corresponding best practice mitigation actions and recommendations.

### 1. Objective

The objective of the assessment was to uncover vulnerabilities in the **Week 2 Labs** and provide a final security assessment report comprising vulnerabilities, remediation strategy, and recommendation guidelines to help mitigate the identified vulnerabilities and risks during the activity.

### 2. Scope

This section defines the scope and boundaries of the project.

|                         |  |
|-------------------------|--|
| <b>Application Name</b> | Insecure Direct Object References, SQL Injection |
|-------------------------|--|

### 3. Summary

Outlined is a Black Box Application Security assessment for **Week 2 Labs**.

**Total number of Sub-labs: 16 Sub-labs**

| High | Medium | Low |
|------|--------|-----|
| 5    | 6      | 5   |

**High** - Number of Sub-labs with hard difficulty level

**Medium** - Number of Sub-labs with medium difficulty level,

**Low** - Number of Sub-labs with Easy difficulty level

## 1. Insecure Direct Object References (IDOR)

### **1.1. Give me my amount!!**

| Reference  | Risk Rating |
|--|-------------|
| Give me my amount!!  | Low         |
| <b>Tools Used</b>  |             |
| Google chrome  |             |
| <b>Vulnerability Description</b>   |             |
| An insecure direct object reference (IDOR) is an access control vulnerability where invalidated user input can be used for unauthorized access to resources or operations. It occurs when an attacker gains direct access by using user supplied input to an object that has no authorization to access. Attackers can bypass the authorization mechanism to access resources in the system directly by exploiting this vulnerability. Every resource instance can be called as an object and often represented with an ID. And if these IDs are easy enough to guess or an object can be used by an attacker to bypass access check somehow, we can talk about an IDOR at this point.                           |             |
| <b>How It Was Discovered</b>   |             |
| I found this vulnerability by manipulating value of id parameter in the URL.   |             |
| <b>Vulnerable URLs</b>   |             |
| <a href="https://labs.hackify.in/HTML/idor_lab/lab_1/profile.php?id=1869">https://labs.hackify.in/HTML/idor_lab/lab_1/profile.php?id=1869</a>  |             |
| <b>Consequences of not Fixing the Issue</b>  |             |
| This vulnerability violates the privacy of a User.   |             |
| <b>Suggested Countermeasures</b>   |             |
| <ol style="list-style-type: none"><li>1. Developers should avoid displaying private object references such as keys or file names.</li><li>2. Validation of parameters should be properly implemented.</li><li>3. Verification of all the referenced objects should be checked.</li><li>4. Tokens should be generated in such a way that they can only be mapped to the user and is not public.</li></ol>   |             |
| <b>References</b>  |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/access-control/idor">https://portswigger.net/web-security/access-control/idor</a></li><li>2. <a href="https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html">https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html</a></li><li>3. <a href="https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/">https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/</a></li></ol> |             |
| <b>Payload Used</b>  |             |
| Appending "?id=1869" at last in the URL  |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

The screenshot shows a web browser displaying a user profile page. The URL in the address bar is `labs.hackify.in/HTML/idor_lab/lab_1/profile.php?id=1869`. The page title is **User Profile**. It contains the following form fields:

- Email: `mail@mail.com`
- Credit Card: `0`
- Transaction 1
- Transaction 2
- Transaction 3

At the bottom of the form are two orange buttons: **Update** and **Log out**. The browser's toolbar is visible at the top, and the status bar at the bottom indicates `© Copyrights 2021 Hackify Cybersecurity All rights reserved`.

## 1.2. Stop polluting my params!

| Reference  | Risk Rating |
|--|-------------|
| Stop polluting my params!  | Medium      |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| <p>An insecure direct object reference (IDOR) is an access control vulnerability where invalidated user input can be used for unauthorized access to resources or operations. It occurs when an attacker gains direct access by using user supplied input to an object that has no authorization to access. Attackers can bypass the authorization mechanism to access resources in the system directly by exploiting this vulnerability. Every resource instance can be called as an object and often represented with an ID. And if these IDs are easy enough to guess or an object can be used by an attacker to bypass access check somehow, we can talk about an IDOR at this point.</p>                    |             |
| How It Was Discovered  |             |
| I found this vulnerability by manipulating value of id parameter in the URL.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hacktify.in/HTML/idor_lab/lab_2/profile.php?id=1327">https://labs.hacktify.in/HTML/idor_lab/lab_2/profile.php?id=1327</a>  |             |
| Consequences of not Fixing the Issue   |             |
| This vulnerability violates the privacy of a User.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Developers should avoid displaying private object references such as keys or file names.</li><li>2. Validation of parameters should be properly implemented.</li><li>3. Verification of all the referenced objects should be checked.</li><li>4. Tokens should be generated in such a way that they can only be mapped to the user and is not public.</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/access-control/idor">https://portswigger.net/web-security/access-control/idor</a></li><li>2. <a href="https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html">https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html</a></li><li>3. <a href="https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/">https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/</a></li></ol> |             |
| Payload Used   |             |
| Appending "?id=1327" at last in the URL  |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

labs.hackify.in/HTML/idor\_lab/lab\_2/profile.php?id=1327

Happy Hacking

### User Profile

Username

First Name

Last Name

© Copyrights 2021 Hackify Cybersecurity All rights reserved

## 1.3. Someone changed my Password!

| Reference  | Risk Rating |
|--|-------------|
| Someone changed my Password!   | High        |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| An insecure direct object reference (IDOR) is an access control vulnerability where invalidated user input can be used for unauthorized access to resources or operations. It occurs when an attacker gains direct access by using user supplied input to an object that has no authorization to access. Attackers can bypass the authorization mechanism to access resources in the system directly by exploiting this vulnerability. Every resource instance can be called as an object and often represented with an ID. And if these IDs are easy enough to guess or an object can be used by an attacker to bypass access check somehow, we can talk about an IDOR at this point.                           |             |
| How It Was Discovered  |             |
| I found this vulnerability by manipulating the value of username parameter in the URL.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hacktify.in/HTML/idor_lab/lab_3/changepassword.php?username=hacker">https://labs.hacktify.in/HTML/idor_lab/lab_3/changepassword.php?username=hacker</a>  |             |
| Consequences of not Fixing the Issue   |             |
| This vulnerability violates the privacy of a User.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Developers should avoid displaying private object references such as keys or file names.</li><li>2. Validation of parameters should be properly implemented.</li><li>3. Verification of all the referenced objects should be checked.</li><li>4. Tokens should be generated in such a way that they can only be mapped to the user and is not public.</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/access-control/idor">https://portswigger.net/web-security/access-control/idor</a></li><li>2. <a href="https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html">https://cheatsheetseries.owasp.org/cheatsheets/Insecure Direct Object Reference Prevention Cheat Sheet.html</a></li><li>3. <a href="https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/">https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/</a></li></ol> |             |
| Payload Used   |             |
| Appending "?username=hacker" at last in the URL  |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

labs.hacktify.in/HTML/idor\_lab/lab\_3/changepassword.php?username=hacker

HACKTIFY  
cybersecurity ISO 27001

Happy Hacking

### Change Password

Username

New Password

Confirm Password

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

## 1.4. Change your methods!

| Reference  | Risk Rating |
|--|-------------|
| Change your methods!   | Medium      |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| An insecure direct object reference (IDOR) is an access control vulnerability where invalidated user input can be used for unauthorized access to resources or operations. It occurs when an attacker gains direct access by using user supplied input to an object that has no authorization to access. Attackers can bypass the authorization mechanism to access resources in the system directly by exploiting this vulnerability. Every resource instance can be called as an object and often represented with an ID. And if these IDs are easy enough to guess or an object can be used by an attacker to bypass access check somehow, we can talk about an IDOR at this point.                           |             |
| How It Was Discovered  |             |
| I found this vulnerability by manipulating the value of id parameter in the URL followed by updating random details in the profile section of different user.  |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hacktify.in/HTML/idor_lab/lab_4/lab_4.php?email=Ram2%40gmail.com&amp;pwd=abcd&amp;submit=">https://labs.hacktify.in/HTML/idor_lab/lab_4/lab_4.php?email=Ram2%40gmail.com&amp;pwd=abcd&amp;submit=</a>  |             |
| Consequences of not Fixing the Issue   |             |
| This vulnerability violates the privacy of a User.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Developers should avoid displaying private object references such as keys or file names.</li><li>2. Validation of parameters should be properly implemented.</li><li>3. Verification of all the referenced objects should be checked.</li><li>4. Tokens should be generated in such a way that they can only be mapped to the user and is not public.</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/access-control/idor">https://portswigger.net/web-security/access-control/idor</a></li><li>2. <a href="https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html">https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html</a></li><li>3. <a href="https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/">https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/</a></li></ol> |             |
| Payload Used   |             |
| Appending "?id=2157" at last in the URL followed by updating random details in the profile section of different user.  |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

labs.hacktify.in/HTML/idor\_lab/lab\_4/lab\_4.php?email=Ram2%40gmail.com&pwd=abcd&submit=

**HACKTIFY**  
cybersecurity

**User Login**

Email

Password

**Login** **Register**

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

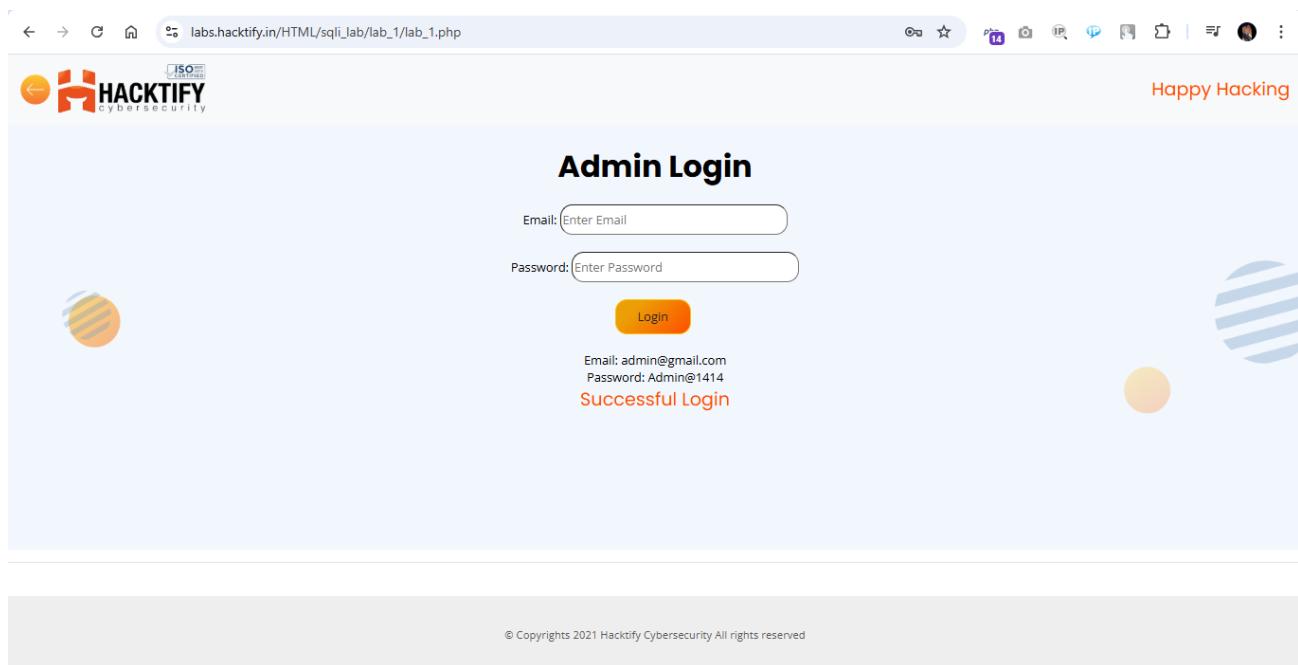
## 2. SQL Injection

### 2.1. Strings and Errors Part1!

| Reference  | Risk Rating |
|--|-------------|
| Strings and Errors Part1!  | Low         |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into user input fields, i.e., Email and Password.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_1/lab_1.php">https://labs.hackify.in/HTML/sql_injection/lab_1/lab_1.php</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| In both the user input fields insert below payload and click login<br>1" OR "1"="1   |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

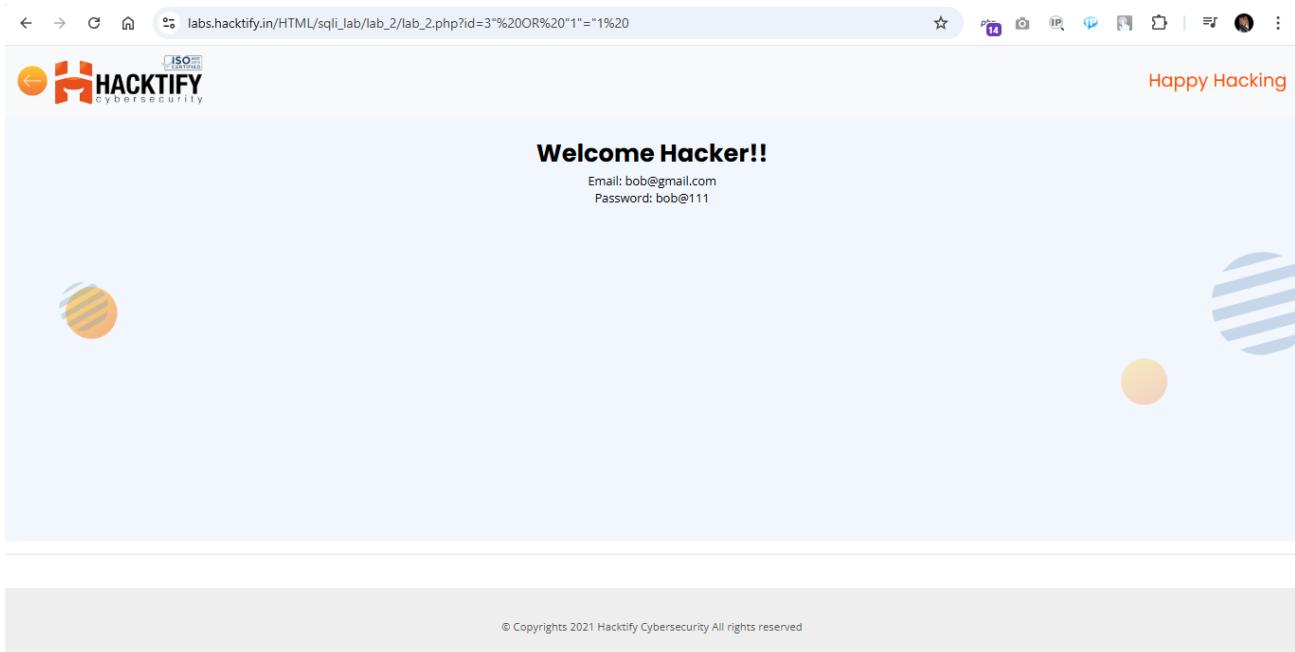


## 2.2. Strings and Errors Part2!

| Reference  | Risk Rating |
|--|-------------|
| Strings and Errors Part2!  | Low         |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into the URL.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_2/lab_2.php?id=3%22%20OR%20%221%22=%221%20">https://labs.hackify.in/HTML/sql_injection/lab_2/lab_2.php?id=3%22%20OR%20%221%22=%221%20</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| Appending below payload at last in the URL<br>?id=3"%20OR%20"1"="1%20  |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab



## 2.3. Strings and Errors Part3!

| Reference  | Risk Rating |
|--|-------------|
| Strings and Errors Part3!  | Low         |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into the URL.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_3/lab_3.php?id=1%22%20OR%20%221%22=%221%20">https://labs.hackify.in/HTML/sql_injection/lab_3/lab_3.php?id=1%22%20OR%20%221%22=%221%20</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| Appending below payload at last in the URL<br>?id=3" OR "1"="1   |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

The screenshot shows a web browser window with the URL `labs.hackify.in/HTML/sqli_lab/lab_3.php?id=1%20OR%20"1"="1%20`. The page title is "Welcome Hacker". On the right side, there is a "Happy Hacking" message. The main content area lists several user credentials:

| Email             | Password    |
|-------------------|-------------|
| admin@gmail.com   | admin123    |
| alice@gmail.com   | alice@121   |
| bob@gmail.com     | bob@111     |
| admin1@gmail.com  | admin#1234  |
| john@gmail.com    | john@789    |
| richard@gmail.com | richard#222 |

At the bottom of the page, a copyright notice reads: © Copyrights 2021 Hackify Cybersecurity All rights reserved.

## 2.4. Let's Trick 'em!

| Reference  | Risk Rating |
|--|-------------|
| Let's Trick 'em!   | Medium      |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into user input fields, i.e., Email and Password.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_4/lab_4.php">https://labs.hackify.in/HTML/sql_injection/lab_4/lab_4.php</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| In both the user input fields insert below payload and click login<br>1'    '1='1  |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

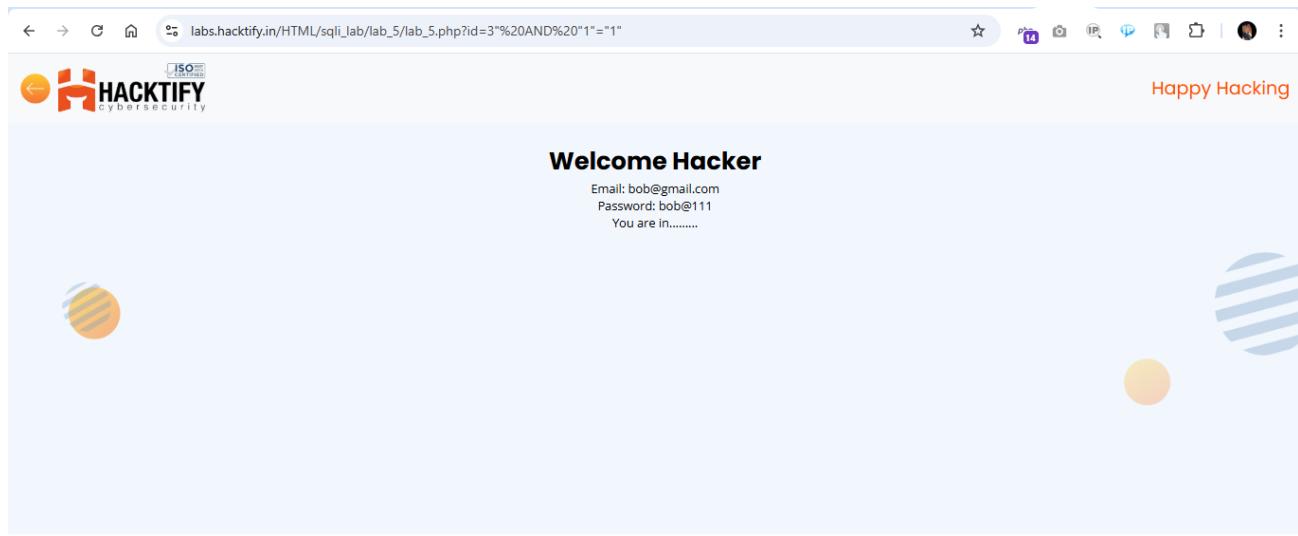
The screenshot shows a web browser window with the URL `labs.hackify.in/HTML/sql_injection/lab_4/lab_4.php`. The page is titled "Admin Login". It features two input fields: "Email: Enter Email" and "Password: Enter Password", both currently empty. Below these fields is an orange "Login" button. To the right of the password field, there is some small text: "Email: admin@gmail.com" and "Password: admin123". Below this text, the message "Successful Login" is displayed in orange. The background of the page is white with some abstract blue and yellow circular patterns. The browser's address bar and various icons are visible at the top.

## 2.5. Booleans and Blind!

| Reference  | Risk Rating |
|--|-------------|
| Booleans and Blind!  | High        |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into the URL.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_5/lab_5.php?id=3%22%20AND%20%221%22=%221%22">https://labs.hackify.in/HTML/sql_injection/lab_5/lab_5.php?id=3%22%20AND%20%221%22=%221%22</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| Appending below payload at last in the URL<br>?id=3" OR "1"="1   |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab



© Copyrights 2021 Hackify Cybersecurity All rights reserved

## 2.6. Error based: Tricked

| Reference  | Risk Rating |
|--|-------------|
| Error based: Tricked   | Medium      |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into user input fields, i.e., Email and Password.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_6/lab_6.php">https://labs.hackify.in/HTML/sql_injection/lab_6/lab_6.php</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| In both the user input fields insert below payload and click login<br>") or ("1")="1   |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

The screenshot shows a web browser window with the URL `labs.hackify.in/HTML/sqli_lab/lab_6/lab_6.php`. The page is titled "Admin Login". It features two input fields: "Email: Enter Email" and "Password: Enter Password", both currently empty. Below these fields is a yellow "Login" button. To the right of the password field, there is some small text: "Email: admin@gmail.com" and "Password: admin123". Below this text, the message "Successful Login" is displayed in orange. The background of the page is white with some abstract circular graphic elements. The top of the browser window shows standard navigation icons and a title bar.

## 2.7. Errors and Post!

| Reference  | Risk Rating |
|--|-------------|
| Errors and Post!   | Low         |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into user input fields, i.e., Email and Password.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_7/lab_7.php">https://labs.hackify.in/HTML/sql_injection/lab_7/lab_7.php</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| In both the user input fields insert below payload and click login<br>' or '1'='1  |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

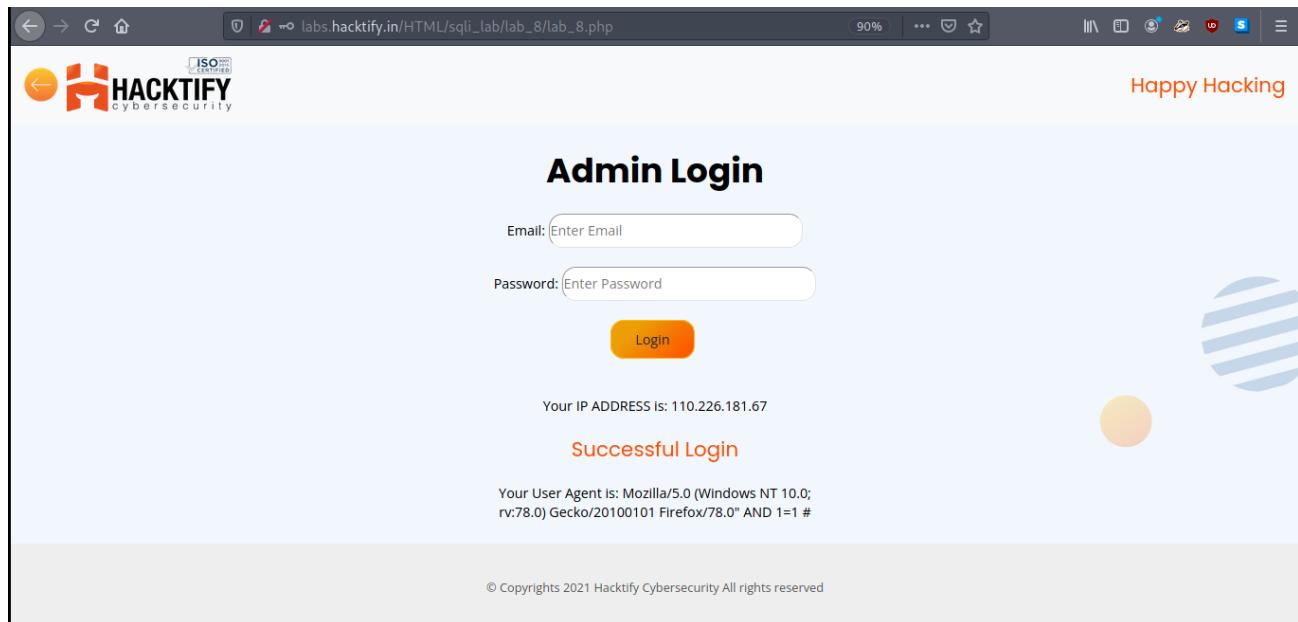
The screenshot shows a web browser window with the URL `labs.hackify.in/HTML/sqli_lab/lab_7/lab_7.php`. The page is titled "Admin Login". It features two input fields: "Email: Enter Email" and "Password: Enter Password", both currently empty. Below these fields is a yellow "Login" button. To the right of the password field, there is some small text: "Email: admin@gmail.com" and "Password: admin123". Below this text, the message "Successful Login" is displayed in orange. The background of the page has abstract circular patterns in orange and blue. The top of the browser window shows standard navigation icons and a toolbar.

## 2.8. User Agents lead us!

| Reference  | Risk Rating |
|--|-------------|
| User Agents lead us!   | High        |
| Tools Used   |             |
| Google chrome and Burp suite   |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into user agent field by intercepting login request into Burp Suite.  |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_8/lab_8.php">https://labs.hackify.in/HTML/sql_injection/lab_8/lab_8.php</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| Login with Admin credentials obtained in previous labs, Intercept the traffic with Burpsuite, In the User Agent field insert the below payload<br>" AND 1=1 #  |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab



## 2.9. Referer lead us!

| Reference  | Risk Rating |
|--|-------------|
| Referer lead us!   | Medium      |
| Tools Used   |             |
| Google chrome and Burp suite   |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into referrer field by intercepting login request into Burp Suite.  |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_9/lab_9.php">https://labs.hackify.in/HTML/sql_injection/lab_9/lab_9.php</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| Login with Admin credentials obtained in previous labs, Intercept the traffic with Burpsuite, In the Referer field insert the below payload<br>' OR '1'='1 #   |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

The screenshot shows a web browser window with the following details:

- Address Bar:** labs.hacktify.in/HTML/sql\_injection/lab\_9/lab\_9.php
- Page Title:** Admin Login
- Form Fields:** Email: Enter Email, Password: Enter Password, Login button.
- Information:** Your IP ADDRESS is: 110.226.181.67, Successful Login, Your User Agent is: "http://labs.hacktify.in/HTML/sql\_injection/lab\_9/lab\_9.php" OR '1'='1 #".
- Footer:** © Copyrights 2021 Hacktify Cybersecurity All rights reserved

## 2.10. Oh Cookies!

| Reference  | Risk Rating |
|--|-------------|
| Oh Cookies!  | High        |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into cookie value by inspecting the cookie settings after logging into the sign in page.  |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_10/lab_10.php">https://labs.hackify.in/HTML/sql_injection/lab_10/lab_10.php</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| Login with Admin credentials, click on Inspect and go to the Cookies section and insert the below payload into Username field<br>1" OR "1"="1  |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab

The screenshot shows a browser window with developer tools open. The main content area displays a web page from 'labs.hackify.in/HTML/sqli\_lab/lab\_10/lab\_10.php'. The page includes a logo for 'HACKIFY cybersecurity', user agent information ('Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36'), IP address ('110.226.181.67'), and a cookie table. The cookie table shows a row for 'username' with the value '1 OR "1"="1'. The developer tools application panel is visible on the right, showing the 'Application' tab selected. The 'Cookies' section is expanded, displaying the same cookie entry. A message at the bottom of the application panel says 'Select a cookie to preview its value'. Below the application panel, there is a 'What's new' section for DevTools 133, featuring a video thumbnail and a link to 'See all new features'.

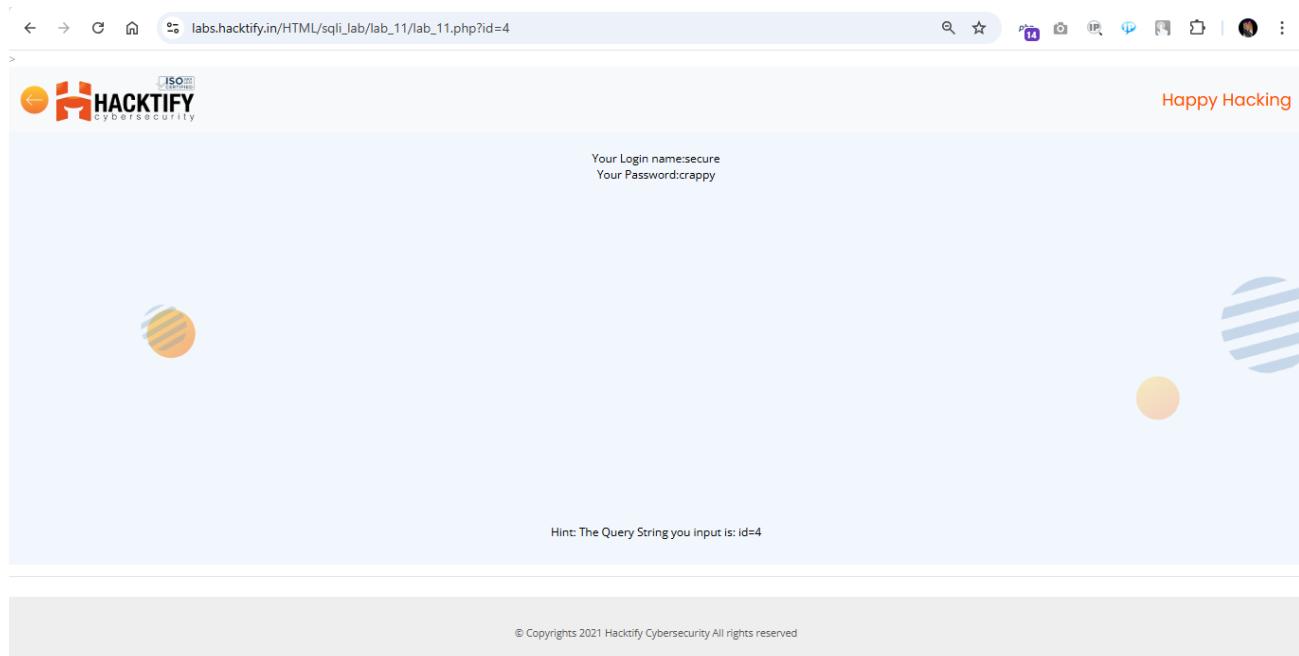
| Name       | Value              | D.   | P... | E... | S... | H... | S... | P... | C... | P... |
|------------|--------------------|------|------|------|------|------|------|------|------|------|
| PHPSESS... | 75ad3201f53b5e7... | l... | /    | S... | 41   | ✓    |      |      | M... |      |
| username   | 1" OR "1"="1       | l... | /... | 2... | 20   | ✓    |      |      | M... |      |

## 2.11. WAF's Are injected!

| Reference  | Risk Rating |
|--|-------------|
| WAF's Are injected!  | High        |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into the URL.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_11/lab_11.php?id=4">https://labs.hackify.in/HTML/sql_injection/lab_11/lab_11.php?id=4</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| Appending below payload at last in the URL<br>?id=4  |             |

## Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab



## 2.12. WAF's Are injected Part2!

| Reference  | Risk Rating |
|--|-------------|
| WAF's Are injected Part2!  | Medium      |
| Tools Used   |             |
| Google chrome  |             |
| Vulnerability Description  |             |
| SQL Injection allows an attacker to view data that attackers are normally not able to retrieve. Data can be information about users, their credentials, personal details etc. It is the process of inserting or injecting SQL queries through input fields to an application to give the hacker the data he wants! Attackers can modify or delete this data causing persistent changes to the application's content or behavior. SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack |             |
| How It Was Discovered  |             |
| I found this vulnerability by injecting malicious SQL injection code into the URL.   |             |
| Vulnerable URLs  |             |
| <a href="https://labs.hackify.in/HTML/sql_injection/lab_12/lab_12.php?id=8">https://labs.hackify.in/HTML/sql_injection/lab_12/lab_12.php?id=8</a>  |             |
| Consequences of not Fixing the Issue   |             |
| Stealing credentials, access to the database, altering or modifying data, access to the network.   |             |
| Suggested Countermeasures  |             |
| <ol style="list-style-type: none"><li>1. Implement Input Validation and Sanitization</li><li>2. Using Escaping for User Input</li><li>3. Utilize Parameterized Statements (Prepared Statements)</li><li>4. Incorporate Stored Procedures</li><li>5. Conduct Continuous Scanning and Penetration Testing</li><li>6. Adopt the Least Privilege Principle</li><li>7. Deploy Web Application Firewalls (WAF)</li></ol>   |             |
| References   |             |
| <ol style="list-style-type: none"><li>1. <a href="https://portswigger.net/web-security/sql-injection/cheat-sheet">https://portswigger.net/web-security/sql-injection/cheat-sheet</a></li><li>2. <a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a></li><li>3. <a href="https://owasp.org/www-community/attacks/SQL_Injection">https://owasp.org/www-community/attacks/SQL_Injection</a></li></ol>   |             |
| Payload Used   |             |
| Appending below payload at last in the URL<br>?id=8  |             |

## Proof of Concept

This section contains proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.

