

# Penetration Testing Report

**Full Name: Sahil Naik**

**Program: HCPT**

**Date: 24/02/2025**

## Introduction

This report document hereby describes the proceedings and results of a Black Box security assessment conducted against the **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>	<b>Lab 1 : Insecure Direct Object References,</b> <b>Lab 2 : SQL Injection</b>
-------------------------	---

## 3. Summary

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

Total number of Sub-labs: {count} Sub-labs

High	Medium	Low
4	7	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

### 1.1. Give me my amount!!

Reference	Risk Rating
Give me my amount!!	Low
Tools Used	
Manual analysis, tampering the url id parameter	
Vulnerability Description	
IDOR is a vulnerability where an attacker accesses unauthorized resources by modifying object identifiers in requests. It occurs due to missing or improper access controls.	
How It Was Discovered / Steps to Reproduce	
<ol style="list-style-type: none"><li>1. Register one account and log in into it</li><li>2. In the url there's an id parameter, for eg id=10</li></ol>	

- |   |
|---|
| 3. Change the id to 9 or 11 and gain access to user profiles of other accounts  |
| <b>Vulnerable URLs</b>  |
| <a href="https://labs.hackify.in/HTML/idor_lab/lab_1/lab_1.php">https://labs.hackify.in/HTML/idor_lab/lab_1/lab_1.php</a>   |
| <b>Consequences of not Fixing the Issue</b>   |
| If an IDOR vulnerability is not fixed, attackers can access sensitive user data, leading to privacy breaches and data leaks. This can result in regulatory penalties, reputational damage, and potential legal consequences for the organization.<br><br>Beyond data exposure, attackers may manipulate or delete user records, leading to account takeovers or service disruptions.  |
| <b>Suggested Countermeasures</b>  |
| <ol style="list-style-type: none"><li>1. Proper Authorization Checks – Implement server-side access controls to verify user permissions before serving any requested resource.</li><li>2. Use Indirect References – Replace direct object identifiers with secure alternatives like UUIDs or tokens to prevent enumeration.</li><li>3. Logging &amp; Monitoring – Detect and respond to suspicious access patterns by logging requests and setting up alerts for anomalies.</li></ol> |
| <b>References</b>   |
| <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>   |

## Proof of Concept

Changed id from 13 to 10

The screenshot shows a web browser window with the URL [labs.hackify.in/HTML/idor\\_lab/lab\\_1/profile.php?id=10](https://labs.hackify.in/HTML/idor_lab/lab_1/profile.php?id=10). The page title is "User Profile". The interface includes fields for "Email" (benep81280@whowow.com), "Credit Card" (100), and three "Transaction" entries: "Transaction 1" (200000), "Transaction 2" (Ronnie Maina), and "Transaction 3" (You have been pwned!). There are "Update" and "Log out" buttons at the bottom. The browser taskbar at the bottom shows various application icons, and the system tray indicates the date and time as 22-02-2025.

## 1.2. Stop polluting my params!

Reference	Risk Rating
Stop polluting my params!	Medium
Tools Used	
Manual analysis, tampering the url id parameter	
Vulnerability Description	
IDOR is a vulnerability where an attacker accesses unauthorized resources by modifying object identifiers in requests. It occurs due to missing or improper access controls.	

## **How It Was Discovered / Steps to Reproduce**

1. Registered 2 accounts, 1 attacker 1 victim
2. Logged in to attacker and victim account with right credentials and noted the user id's
3. Logged in as attacker and on the update profile page, changed the id parameter to victims id and changed his details.

## **Vulnerable URLs**

[https://labs.hackify.in/HTML/idor\\_lab/lab\\_2/lab\\_2.php](https://labs.hackify.in/HTML/idor_lab/lab_2/lab_2.php)

## **Consequences of not Fixing the Issue**

1. Unauthorized Profile Modification – The attacker can alter the victim's personal details, leading to misinformation, account misuse, or identity fraud.
2. Escalation Risks – If profile updates include email or password changes, the attacker could hijack the victim's account entirely.

## **Suggested Countermeasures**

1. Strict Access Controls – Ensure the server verifies if the logged-in user has permission to modify the requested profile.
2. Session-Based Identification – Use session tokens instead of user-controlled IDs for authentication and updates.
3. Audit Logs & Alerts – Track user activities and flag unusual ID modifications for investigation.

## **References**

[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)

# **Proof of Concept**

The screenshot shows a web browser window with three tabs: "HCS - Penetration Testing Week", "Login", and "Report - Google Docs". The active tab is "labs.hackify.in/HTML/idor\_lab/lab\_2/profile.php?id=12". The page itself has a header with the Hackify logo and ISO 9001 certification. The main content is titled "User Profile" and contains fields for "Username" (admin@hackify.org), "First Name" (admin), and "Last Name" (hackify). At the bottom are "Update" and "Log out" buttons.

## 1.3. Someone changed my Password 😱!

Reference	Risk Rating
Someone changed my Password 😱!	Medium
Tools Used	
Manual analysis, tampering the url id parameter	
Vulnerability Description	
IDOR is a vulnerability where an attacker accesses unauthorized resources by modifying object identifiers in requests. It occurs due to missing or improper access controls.	
How It Was Discovered / Steps to Reproduce	

1. Create an account and log in.
2. Navigate to the update password page.
3. Identify the `username` parameter in the URL.
4. Modify the `username` value to another user's ID.
5. Access another user's account details.
6. Change their password successfully.

#### Vulnerable URLs

[https://labs.hackify.in/HTML/idor\\_lab/lab\\_3/lab\\_3.php](https://labs.hackify.in/HTML/idor_lab/lab_3/lab_3.php)

#### Consequences of not Fixing the Issue

1. Account Takeover – The attacker can reset passwords and lock victims out of their accounts.
2. Data & Service Abuse – Compromised accounts can be misused for fraud, impersonation, or further attacks.

#### Suggested Countermeasures

1. Authorization Checks – Verify users can only update their own passwords.
2. Session-Based Identity – Use session tokens instead of user-controlled id parameters.
3. Rate Limiting & Alerts – Detect and block suspicious id modifications.

#### References

[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)

## Proof of Concept

HCS - Penetration Testing Week | Login | Report - Google Docs

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

**HACKTIFY** cybersecurity ISO 27001 CERTIFIED

## Change Password

Username

New Password

Confirm Password

**Change** **Back**

## 1.4. Change your methods!

Reference	Risk Rating
Change your methods!	Medium
Tools Used	
Manual analysis, tampering the url id parameter	
Vulnerability Description	
IDOR is a vulnerability where an attacker accesses unauthorized resources by modifying object identifiers in requests. It occurs due to missing or improper access controls.	
How It Was Discovered / Steps to Reproduce	

1. Registered first acc got id 2160 (attacker acc)
2. Registered second acc got id 1946 (victim acc)
3. Logged in from the acc with id 2160 and changed the id to 1946 and updated user details, then logged out and logged in using the updated details on the victim account.

#### Vulnerable URLs

[https://labs.hackify.in/HTML/idor\\_lab/lab\\_4/lab\\_4.php](https://labs.hackify.in/HTML/idor_lab/lab_4/lab_4.php)

#### Consequences of not Fixing the Issue

Full Account Takeover – The attacker gains control of the victim's account by modifying its details.

Identity & Data Theft – The attacker can misuse the victim's account, access sensitive data, or perform malicious actions.

#### Suggested Countermeasures

1. Enforce Access Controls – Ensure users can only modify their own account details.
2. Use Session-Based Authentication - Authenticate users via session tokens, not user-controlled IDs.
3. Enable Logging & Alerts – Monitor and flag unauthorized ID modifications.

#### References

[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)

## Proof of Concept

**User Profile**

Username

First Name

Last Name

**Update** **Log out**

## 2. SQL Injection

### 2.1. Strings & Errors Part 1!

Reference	Risk Rating
Strings & Errors Part 1!	Low
Tools Used	
Manual analysis, using sqli payloads in the input fields	
Vulnerability Description	
Boolean-Based SQLi occurs when an attacker injects conditions that always evaluate to TRUE or False	

and observes changes in application behavior.

#### How It Was Discovered / Steps to Reproduce

1. Add payload in email and password field : admin" or "1"="1
2. Log in, and it gets triggered

#### Vulnerable URLs

[https://labs.hacktify.in/HTML/sql\\_injection/lab\\_1/lab\\_1.php](https://labs.hacktify.in/HTML/sql_injection/lab_1/lab_1.php)

#### Consequences of not Fixing the Issue

Unauthorized Access – Attackers can bypass login and gain control of user accounts.

Data Theft & Manipulation – Sensitive information can be stolen, altered, or deleted.

#### Suggested Countermeasures

1. Use Prepared Statements – Prevent SQL injection by properly parameterizing queries.
2. Input Validation & Escaping – Sanitize user inputs to block malicious characters.
3. Least Privilege Principle – Limit database permissions to minimize damage from attacks.

#### References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

## Proof of Concept

HCS - Penetration Testing Week | Login | Report - Google Docs

labs.hacktify.in/HTML/sqli\_lab/lab\_1/lab\_1.php

**HACKTIFY** cybersecurity ISO 27001 CERTIFIED

## Admin Login

Email:

Password:

**Login**

Email: admin@gmail.com  
Password: Admin@1414

**Successful Login**

## 2.2. Strings & Errors Part 2!

Reference	Risk Rating
Strings & Errors Part 2!	Low
<b>Tools Used</b>	
Manual analysis, using sql payloads in the input fields	
<b>Vulnerability Description</b>	
Boolean-Based SQLi occurs when an attacker injects conditions that always evaluate to TRUE or False and observes changes in application behavior.	
<b>How It Was Discovered / Steps to Reproduce</b>	
1. Add payload in the id parameter of url field : id=1" or "1"="1	

- Triggering and showing email id of admin and other users

### Vulnerable URLs

[https://labs.hacktify.in/HTML/sql\\_injection/lab\\_2/lab\\_2.php](https://labs.hacktify.in/HTML/sql_injection/lab_2/lab_2.php)

### Consequences of not Fixing the Issue

- Unauthorized Data Access – Attackers can view sensitive records without authentication.
- Data Manipulation – Malicious queries can modify or delete database entries.
- Account Compromise – Login bypass can lead to full control of user accounts.

### Suggested Countermeasures

- Use Prepared Statements – Prevent SQL injection by properly parameterizing queries.
- Input Validation & Escaping – Sanitize user inputs to block malicious characters.
- Least Privilege Principle – Limit database permissions to minimize damage from attacks.

### References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

## Proof of Concept

The screenshot shows a browser window with three tabs: "HCS - Penetration Testing Week", "URL", and "Report - Google Docs". The active tab is the URL tab, which displays the URL [https://labs.hacktify.in/HTML/sql\\_injection/lab\\_2/lab\\_2.php?id=1%27+or+1=1--](https://labs.hacktify.in/HTML/sql_injection/lab_2/lab_2.php?id=1%27+or+1=1--). Below the tabs, the page content is visible, featuring the Hacktify logo and a "Welcome Hacker!!" message with login credentials.

**HACKTIFY**  
cybersecurity

Welcome Hacker!!

Email: admin@gmail.com  
Password: admin123

## 2.3. Strings & Errors Part 3!

Reference	Risk Rating
Strings & Errors Part 3!	Low
Tools Used	
Manual analysis, using sql payloads in the input fields	
Vulnerability Description	
Union-Based SQLi occurs when an attacker uses the UNION SQL clause to combine results from multiple tables, extracting sensitive data. By modifying parameters like id=1 ORDER BY 1--, attackers can determine column count and inject UNION SELECT queries.	
How It Was Discovered / Steps to Reproduce	
<ol style="list-style-type: none"><li>1. Add payload in the id parameter of url field : id=1' +ORDERBY 1--</li><li>2. Triggering and showing email id of admin and other users</li></ol>	
Vulnerable URLs	
<a href="https://labs.hackify.in/HTML/sql_injection/lab/lab_3/lab_3.php">https://labs.hackify.in/HTML/sql_injection/lab/lab_3/lab_3.php</a>	
Consequences of not Fixing the Issue	
<ol style="list-style-type: none"><li>1. Data Exposure – Attackers can retrieve sensitive information like admin credentials.</li><li>2. Account Takeover – Extracted credentials can be used to hijack accounts.</li></ol>	
Suggested Countermeasures	
<ol style="list-style-type: none"><li>1. Use Prepared Statements – Prevent UNION injection by parameterizing queries.</li><li>2. Limit Database Permissions – Restrict access to sensitive tables and data.</li></ol>	

3. Disable Error Messages – Prevent attackers from identifying table structures.

## References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

## Proof of Concept

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

- Tab bar: HCS - Penetration Testing Week, URL, Report - Google Docs.
- Address bar: labs.hackify.in/HTML/sqli\_lab/lab\_3/lab\_3.php?id=1%27+ORDERBY1--
- Header: ISO 27001 CERTIFIED
- Logo: HACKIFY cybersecurity
- Main Content:
  - Welcome Hacker
  - Email: admin@gmail.com
  - Password: admin123

## 2.4. Let's Trick 'em!

Reference	Risk Rating
_et's Trick 'em!	<b>Medium</b>
Tools Used	
Manual analysis, using sql payloads in the input fields	
Vulnerability Description	
Authentication Bypass SQL Injection. This attack occurs when login forms fail to properly validate user	

inputs, allowing attackers to inject SQL code to bypass authentication.

#### How It Was Discovered / Steps to Reproduce

1. Add payload in email and password field : ‘=”or’
2. Successful log in as admin.

#### Vulnerable URLs

[https://labs.hackify.in/HTML/sql\\_injection/lab\\_4/index.php](https://labs.hackify.in/HTML/sql_injection/lab_4/index.php)

#### Consequences of not Fixing the Issue

1. Admin Panel Access – Attackers can log in as an administrator without valid credentials.
2. Full System Compromise – Admin privileges may allow modification or deletion of data, user management, or further exploitation.

#### Suggested Countermeasures

1. Use Prepared Statements – Prevent input manipulation by parameterizing SQL queries.
2. Implement Strong Input Validation – Reject special characters and enforce strict input rules.
3. Use Multi-Factor Authentication (MFA) – Add an extra layer of security to prevent unauthorized logins.

#### References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

## Proof of Concept

HCS - Penetration Testing Week    Login    Report - Google Docs

labs.hackify.in/HTML/sql\_i\_lab/lab\_4/lab\_4.php

**HACKIFY**  
cybersecurity

## Admin Login

Email:

Password:

**Login**

Email: admin@gmail.com  
Password: admin123

**Successful Login**

## 2.5.Booleans and Blind!

Reference	Risk Rating
Booleans and Blind!	High
<b>Tools Used</b>	
Manual analysis, using sql payloads in the url fields	
<b>Vulnerability Description</b>	
Boolean-Based SQL Injection. This attack exploits improper input handling in SQL queries by injecting a condition that always evaluates to TRUE.	
<b>How It Was Discovered / Steps to Reproduce</b>	
<ol style="list-style-type: none"> <li>1. Add payload in the id parameter of url field : id=1 or "1" = "1"</li> <li>2. Triggering and showing email id of admin and other users</li> </ol>	

## Vulnerable URLs

[https://labs.hackify.in/HTML/sql\\_injection/lab\\_5/lab\\_5.php](https://labs.hackify.in/HTML/sql_injection/lab_5/lab_5.php)

## Consequences of not Fixing the Issue

1. Unauthorized Data Access – Attackers can retrieve sensitive records by bypassing query restrictions.
2. Data Enumeration – Attackers can iterate through database entries, extracting confidential information.

## Suggested Countermeasures

1. Use Prepared Statements – Prevent SQL injection by securely parameterizing queries.
2. Strict Input Validation – Block special characters and enforce expected input formats.
3. Limit Database Permissions – Restrict access to prevent unauthorized data extraction.

## References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

## Proof of Concept

The screenshot shows a browser window with three tabs: "HCS - Penetration Testing Week", "Hackify Labs", and "Report - Google Docs". The active tab is "Hackify Labs" which displays the URL "labs.hackify.in/HTML/sql\_injection/lab\_5/lab\_5.php?id=1%20or%221%22=%221%22". Below the tabs, the page content starts with the Hackify logo and an ISO 27001 certification badge. To the right, a large "Welcome Hacker" message is displayed, followed by email and password fields, and a message "You are in.....".

## 2.6. Error Based : Tricked

Reference	Risk Rating
Error Based : Tricked	Medium
<b>Tools Used</b>	
Manual analysis, using sql payloads in the input fields	
<b>Vulnerability Description</b>	
Authentication Bypass SQL Injection. This attack occurs when login forms fail to properly validate user inputs, allowing attackers to inject SQL code to bypass authentication.	
<b>How It Was Discovered / Steps to Reproduce</b>	
3. Add payload in email and password field : ')' or ('s' = 's and hack") or ("a"="a 4. Successful log in as admin.	
<b>Vulnerable URLs</b>	
<a href="https://labs.hackify.in/HTML/sqli_lab/lab_6/lab_6.php">https://labs.hackify.in/HTML/sqli_lab/lab_6/lab_6.php</a>	
<b>Consequences of not Fixing the Issue</b>	
1. Admin Panel Access – Attackers can log in as an administrator without valid credentials. 2. Full System Compromise – Admin privileges may allow modification or deletion of data, user management, or further exploitation.	
<b>Suggested Countermeasures</b>	
1. Use Prepared Statements – Prevent input manipulation by parameterizing SQL queries. 2. Implement Strong Input Validation – Reject special characters and enforce strict input rules.	

3. Use Multi-Factor Authentication (MFA) – Add an extra layer of security to prevent unauthorized logins.

## References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

## Proof of Concept

The screenshot shows a browser window with the following details:

- Tab titles: "HCS - Penetration Testing Week", "Login", and "Report - Google Docs".
- URL bar: "labs.hacktify.in/HTML/sql\_i\_lab/lab\_6/lab\_6.php".
- Page header: "HACKTIFY cybersecurity" with an ISO 27001 certification logo.
- Section title: "Admin Login".
- Form fields:
  - Email:
  - Password:
- Button: "Login" (yellow button).
- Success message: "Email: admin@gmail.com  
Password: admin123  
**Successful Login**" (in red text).

## 2.7. Errors and Post!

Reference	Risk Rating
Errors and Post!	Low
<b>Tools Used</b>	

Manual analysis, using sql payloads in the input fields

### Vulnerability Description

Authentication Bypass SQL Injection. This attack occurs when login forms fail to properly validate user inputs, allowing attackers to inject SQL code to bypass authentication.

### How It Was Discovered / Steps to Reproduce

5. Add payload in email and password field : ' or '1'='1
6. Successful log in as admin.

### Vulnerable URLs

[https://labs.hacktify.in/HTML/sql\\_i\\_lab/lab\\_7/lab\\_7.php](https://labs.hacktify.in/HTML/sql_i_lab/lab_7/lab_7.php)

### Consequences of not Fixing the Issue

1. Admin Panel Access – Attackers can log in as an administrator without valid credentials.
2. Full System Compromise – Admin privileges may allow modification or deletion of data, user management, or further exploitation.

### Suggested Countermeasures

1. Use Prepared Statements – Prevent input manipulation by parameterizing SQL queries.
2. Implement Strong Input Validation – Reject special characters and enforce strict input rules.
3. Use Multi-Factor Authentication (MFA) – Add an extra layer of security to prevent unauthorized logins.

### References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

# Proof of Concept

The screenshot shows a web browser window with three tabs: 'HCS - Penetration Testing Week', 'Login', and 'Report - Google Docs'. The active tab is 'labs.hackify.in/HTML/sql\_i\_lab/lab\_7/lab\_7.php'. The page itself is titled 'Admin Login' and features a logo for 'HACKIFY cybersecurity' with an ISO 27001 certification badge. There are two input fields: one for 'Email' and one for 'Password', both with placeholder text 'Enter Email' and 'Enter Password' respectively. A large orange 'Login' button is centered below the fields. At the bottom of the form, the email 'admin@gmail.com' and password 'admin123' are displayed, along with a red 'Successful Login' message.

## 2.8. User Agents lead us!

Reference	Risk Rating
User Agents lead us!	High
Tools Used	
Burpsuite and sql payloads	
Vulnerability Description	
SQL Injection via User-Agent Header. This vulnerability occurs when the application logs or displays the User-Agent header unsafely, allowing SQL injection. By intercepting the request with Burp Suite and	

appending a SQL payload to the User-Agent, the attacker can execute malicious queries.

### How It Was Discovered / Steps to Reproduce

1. Access Admin Login Page – Visit the admin panel sign-in page where the User-Agent is displayed.
2. Intercept Request – Use Burp Suite to capture the login request.
3. Modify User-Agent Header – Append an SQLi payload (e.g., User-Agent: Mozilla/5.0" OR "1"="1).
4. Forward the Request – Send the modified request and observe SQL execution.

### Vulnerable URLs

[https://labs.hackify.in/HTML/sql\\_i\\_lab/lab\\_8/lab\\_8.php](https://labs.hackify.in/HTML/sql_i_lab/lab_8/lab_8.php)

### Consequences of not Fixing the Issue

1. Data Exposure – Attackers can extract sensitive admin data.
2. Privilege Escalation – Malicious queries may grant admin-level access.

### Suggested Countermeasures

1. Sanitize Logged Inputs – Escape or filter user-supplied headers before processing.
2. Use Parameterized Queries – Prevent SQL execution from unsanitized input.
3. Restrict Debug Information – Avoid displaying untrusted data in the UI.

### References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

## Proof of Concept

# Admin Login

Email:

Password:

Your IP ADDRESS is: 117.222.93.24

Successful Login

Your User Agent is: Mozilla/5.0 (Windows NT 10.0;  
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/133.0.0.0 Safari/537.36 OR 1=1

## 2.9. Referer lead us!

Reference	Risk Rating
Referer lead us!	Medium
<b>Tools Used</b>	
Burpsuite and sql payloads	
<b>Vulnerability Description</b>	
SQL Injection via Referer Header. This vulnerability occurs when the application logs or displays the Referer header without sanitization, allowing SQL injection. By intercepting the request with Burp Suite and appending an SQL payload to the Referer, the attacker can execute arbitrary queries.	
<b>How It Was Discovered / Steps to Reproduce</b>	

1. Access Admin Login Page – Visit the admin panel where the Referer is processed or displayed.
2. Intercept Request – Use Burp Suite to capture the login request.
3. Modify Referer Header – Append an SQLi payload (e.g., Referer: http://site.com" OR "1"="1).
4. Forward the Request – Send the modified request and observe SQL execution.

#### Vulnerable URLs

[https://labs.hacktify.in/HTML/sql\\_i\\_lab/lab\\_9/lab\\_9.php](https://labs.hacktify.in/HTML/sql_i_lab/lab_9/lab_9.php)

#### Consequences of not Fixing the Issue

1. Data Breach – Attackers can extract sensitive database records.
2. Privilege Escalation – SQL queries may allow unauthorized admin access.

#### Suggested Countermeasures

1. Sanitize Logged Inputs – Properly escape or filter all user-supplied headers.
2. Use Parameterized Queries – Prevent SQL execution from injected inputs.
3. Limit Logging Exposure – Avoid displaying raw request headers in the UI.

#### References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

## Proof of Concept

/lab\_9/lab\_9.php

# Admin Login

Email:

Password:

Your IP ADDRESS is: 117.222.93.24

**Successful Login**

Your User Agent is: "" OR "1"="1"

The screenshot shows a web application titled 'Admin Login'. It has two input fields for 'Email' and 'Password', both labeled 'Enter [Field]'. Below the inputs is a yellow 'Login' button. After logging in, the page displays the user's IP address (117.222.93.24) and user agent ('"" OR "1"="1'). The word 'Successful Login' is displayed in orange at the bottom.

## 2.10. Oh Cookies!

Reference	Risk Rating
Oh Cookies!	<b>High</b>
Tools Used	
DevTools on Browser and sqli payloads	
Vulnerability Description	
SQL Injection via Cookie Header. This vulnerability occurs when the application displays cookie values in the UI without proper sanitization, allowing SQL injection. Using browser DevTools, an attacker can	

modify the cookie value with an SQLi payload, leading to database information disclosure.

### How It Was Discovered / Steps to Reproduce

1. Inspect Cookies in DevTools – Open browser DevTools and navigate to the storage tab.
2. Modify Cookie Value – Replace it with an SQLi payload like: ' UNION SELECT version(), user(), database()#
3. Refresh the Page – Observe the database version, admin credentials, or other sensitive data displayed in the UI.

### Vulnerable URLs

[https://labs.hacktify.in/HTML/sql\\_i\\_lab/lab\\_10/lab\\_10.php](https://labs.hacktify.in/HTML/sql_i_lab/lab_10/lab_10.php)

### Consequences of not Fixing the Issue

1. Database Information Disclosure – Attackers can view version, user, and database details.
2. Privilege Escalation – Sensitive data can be used for further exploitation

### Suggested Countermeasures

1. Sanitize and Validate Cookies – Never process user-controlled cookies without proper filtering.
2. Use Prepared Statements – Prevent direct execution of cookie values in SQL queries.
3. Restrict Debug Information – Avoid exposing raw cookie data in the UI.

### References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

## Proof of Concept

Your USER AGENT is Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36

Your IP ADDRESS is 117.222.93.24

DELETE YOUR COOKIE OR WAIT FOR IT TO EXPIRE

YOUR COOKIE: username: ' union select version(), user(), database()# and expires: Sun 23 Feb 2025 - 18:59:54

Your Login username: bugbzhiiy\_bughunter@localhost  
Your Password: bugbzhiiy\_sqli  
Your ID: 10.6.20-MariaDB-cll-lve-log

Delete Your Cookie!

Name	Value
PHPSESSID	6a431e90c08868e23...
_ga	GA1.1.647723510.17...
_ga_BC1TF...	GS1.1.1739355287.2...
cf_clearance	mJ1sR62751HBzw94...
username	' union select version...

## 2.11.WAF's are injected!

Reference	Risk Rating
WAF's are injected!	High
<b>Tools Used</b>	
Manual analysis, using sql payloads in the url fields	
<b>Vulnerability Description</b>	
WAF Bypassed SQL Injection via id Parameter. This attack targets a web application firewall protected site by obfuscating SQL payloads to bypass detection. By injecting SQL via the id using encoded or space-altered payloads, the attacker successfully extracts database information.	

## How It Was Discovered / Steps to Reproduce

1. Identify WAF Protection – Test for basic SQLi payloads being blocked.
2. Craft WAF-Bypassing Payload – Use encoded or obfuscated payload like:

```
id=1&id=0' +union+select+1,@@version,database()--+
```

3. Send the Request – Observe the extracted database version and name in the response.

## Vulnerable URLs

[https://labs.hackify.in/HTML/sql\\_injection/lab\\_11/lab\\_11.php](https://labs.hackify.in/HTML/sql_injection/lab_11/lab_11.php)

## Consequences of not Fixing the Issue

1. Database Fingerprinting – Attackers can identify the database type and version for further exploitation.
2. Security Bypass – Successfully bypassing WAF protections weakens overall security defenses.

## Suggested Countermeasures

Advanced WAF Rules – Implement strict SQL injection detection, including obfuscation techniques.

Use Parameterized Queries – Ensure backend queries do not directly process user-controlled input.

Monitor Logs & Anomalies – Detect and block suspicious payload patterns in real time.

## References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

# Proof of Concept

Your Login name:10.6.20-MariaDB-cll-lve-log  
Your Password:bugbzhiy\_sql

## 2.12.WAF's are injected Part 2!

Reference	Risk Rating
WAF's are injected Part 2!	Medium
Tools Used	
Manual analysis, using sql payloads in the url fields	
Vulnerability Description	
WAF Bypassed SQL Injection via id Parameter. This attack targets a web application firewall protected site by obfuscating SQL payloads to bypass detection. By injecting SQL via the id using encoded or space-altered payloads, the attacker successfully extracts database information.	
How It Was Discovered / Steps to Reproduce	
<ol style="list-style-type: none"><li>1. Identify WAF Protection – Test for basic SQLi payloads being blocked.</li><li>2. Craft WAF-Bypassing Payload – Use encoded or obfuscated payload like:  <code>id=0&amp;id=1' +union+select+1,@@version,database()--+</code></li><li>3. Send the Request – Observe the extracted database version and name in the response.</li></ol>	

## Vulnerable URLs

[https://labs.hackify.in/HTML/sql\\_injection/lab\\_12/lab\\_12.php](https://labs.hackify.in/HTML/sql_injection/lab_12/lab_12.php)

## Consequences of not Fixing the Issue

1. Database Fingerprinting – Attackers can identify the database type and version for further exploitation.
2. Security Bypass – Successfully bypassing WAF protections weakens overall security defenses.

## Suggested Countermeasures

Advanced WAF Rules – Implement strict SQL injection detection, including obfuscation techniques.

Use Parameterized Queries – Ensure backend queries do not directly process user-controlled input.

Monitor Logs & Anomalies – Detect and block suspicious payload patterns in real time.

## References

<https://github.com/payloadbox/sql-injection-payload-list>

[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)

# Proof of Concept

The screenshot shows a web browser window with three tabs open:

- HCS - Penetration Testing Week
- URL
- Report - Google Docs

The URL tab displays the following address:  
labs.hackify.in/HTML/sql\_injection/lab\_12/lab\_12.php?id=0&id=1%27%20+union+select+1,@@version,database()--+

The page content area shows the Hackify logo at the top left and a message at the bottom right:

Your Login name:Dumb  
Your Password:Dumb

Thank You!