

Penetration Testing Report

Full Name: **BELKEBIR KAOUTHAR**
Program: HCS - Penetration Testing Internship Week-2
Date:23/02/2025

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.

Application Name	Sql injection , IDOR
------------------	----------------------

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. SQL INJECTION

1.1.Strings and Errors Part 1!

Reference	Risk Rating
Strings and Errors Part 1!	Low
Tools Used	
BURBSUITE	
Vulnerability Description	
The SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. Attackers insert malicious SQL code into input fields (like login forms) that are not properly sanitized.	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sql_i_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
Data Breaches: Unauthorized access to sensitive data.	
Suggested Countermeasures	
<ul style="list-style-type: none">Input Validation: Ensure all user inputs are properly validated and sanitized.Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.	
References	
https://portswigger.net/web-security/sql-injection	

Proof of Concept

Admin Login

Email:

Password:

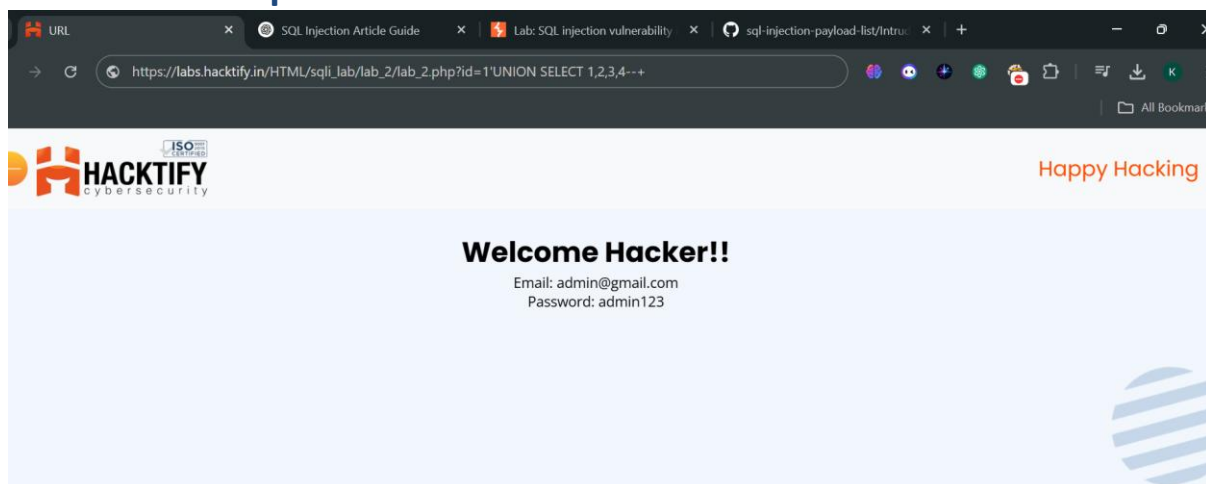
Login

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

1.2. Strings and Errors Part 2!

Reference	Risk Rating
Strings and Errors Part 2!	Low
Tools Used	
manual	
Vulnerability Description	
The SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. Attackers insert malicious SQL code into input fields (like login forms) that are not properly sanitized	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sql_i_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Data Breaches: Unauthorized access to sensitive data.• Data Manipulation: Alteration or deletion of data.• System Compromise: In some cases, attackers can gain control over the entire database server.	
Suggested Countermeasures	
<ul style="list-style-type: none">• Input Validation: Ensure all user inputs are properly validated and sanitized.• Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.• Stored Procedures: Utilize stored procedures to handle database operations.	
References	
https://portswigger.net/web-security/sql-injection	

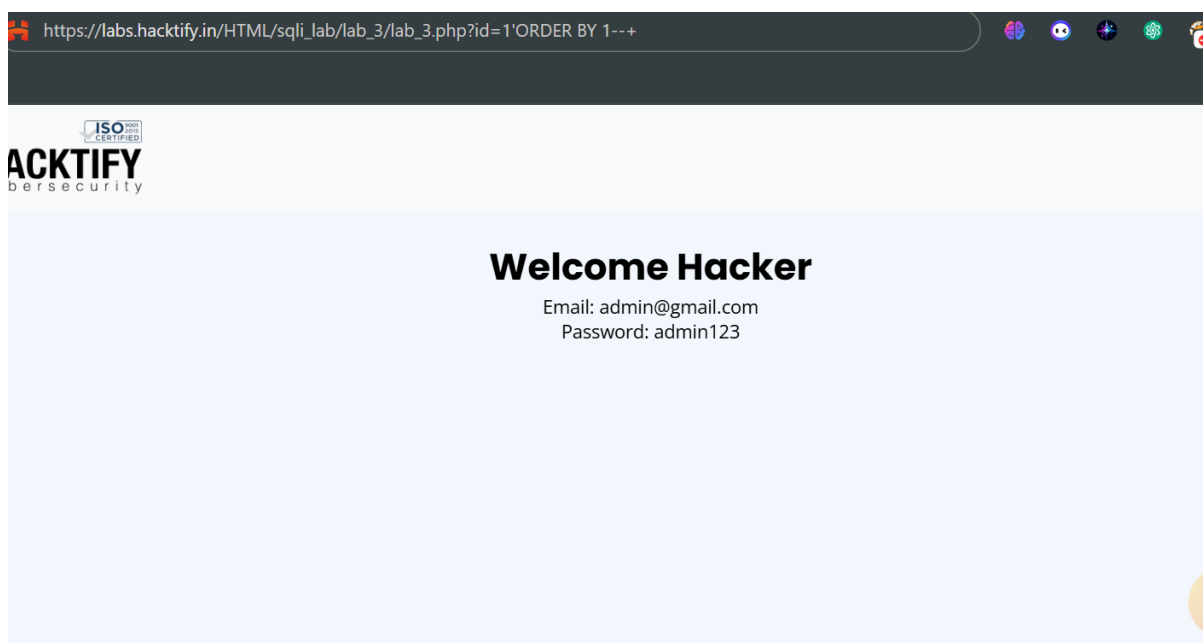
Proof of Concept



1.3. Strings and Errors Part 3!

Reference	Risk Rating
Strings and Errors Part 3!	Low
Tools Used	
manual	
Vulnerability Description	
The SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. Attackers insert malicious SQL code into input fields (like login forms) that are not properly sanitized	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sql_i_lab/lab_3/lab_3.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Data Breaches: Unauthorized access to sensitive data.• Data Manipulation: Alteration or deletion of data.• System Compromise: In some cases, attackers can gain control over the entire database server.	
Suggested Countermeasures	
<ul style="list-style-type: none">• Input Validation: Ensure all user inputs are properly validated and sanitized.• Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.• Stored Procedures: Utilize stored procedures to handle database operations.	
References	
https://portswigger.net/web-security/sql-injection	

Proof of Concept



1.4. Let's Trick 'em !

Reference	Risk Rating
Let's Trick 'em!	medium
Tools Used	
manual	
Vulnerability Description	
<p>The SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.</p> <p>Attackers insert malicious SQL code into input fields (like login forms) that are not properly sanitized. Unauthorized Data Access: This can lead to unauthorized access to sensitive data, such as user credentials, personal information, and financial records.</p>	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sql_i_lab/lab_4/lab_4.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Data Breaches: Unauthorized access to sensitive data.• Data Manipulation: Alteration or deletion of data.• System Compromise: In some cases, attackers can gain control over the entire database server.	
Suggested Countermeasures	
<ul style="list-style-type: none">• Input Validation: Ensure all user inputs are properly validated and sanitized.• Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.• Stored Procedures: Utilize stored procedures to handle database operations.	
References	
https://portswigger.net/web-security/sql-injection	

Proof of Concept

Admin Login

Email:

Password:

Login

Email: admin@gmail.com
Password: admin123

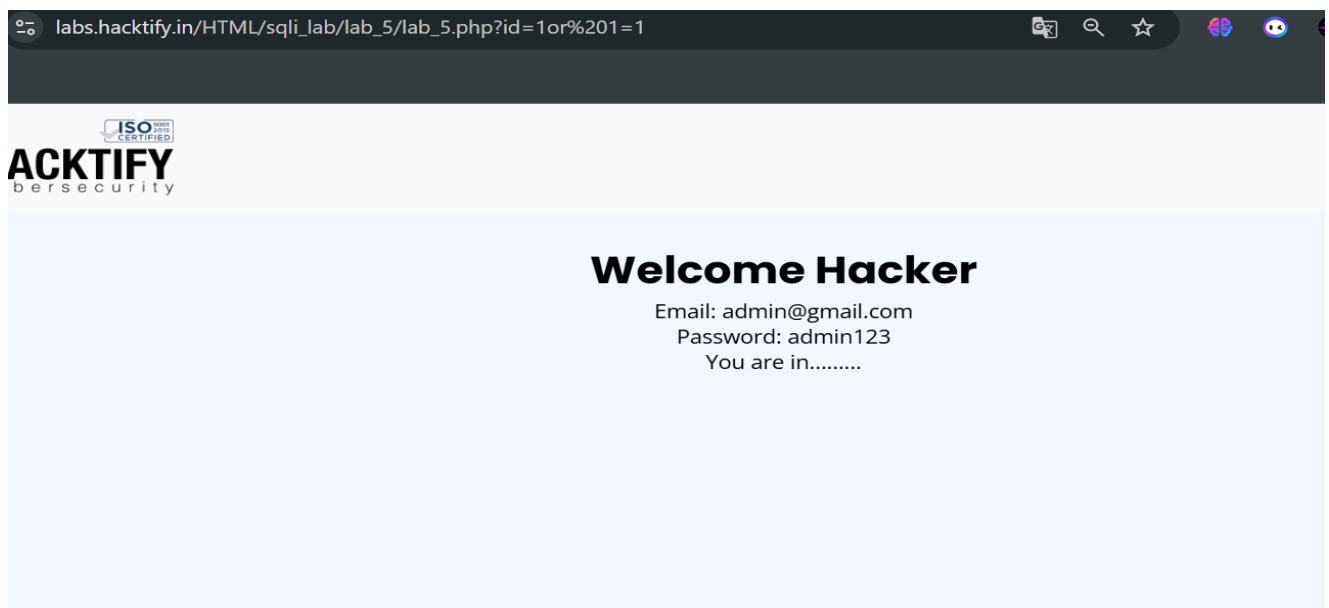
Successful Login

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.5. Booleans and Blind!

Reference	Risk Rating
Booleans and Blind!	HIGH
Tools Used	
manual	
Vulnerability Description	
<p>The SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.</p> <p>Attackers insert malicious SQL code into input fields (like login forms) that are not properly sanitized. Unauthorized Data Access: This can lead to unauthorized access to sensitive data, such as user credentials, personal information, and financial records.</p>	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sql_i_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Data Breaches: Unauthorized access to sensitive data.• Data Manipulation: Alteration or deletion of data.• System Compromise: In some cases, attackers can gain control over the entire database server.	
Suggested Countermeasures	
<ul style="list-style-type: none">• Input Validation: Ensure all user inputs are properly validated and sanitized.• Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.• Stored Procedures: Utilize stored procedures to handle database operations.	
References	
https://portswigger.net/web-security/sql-injection	

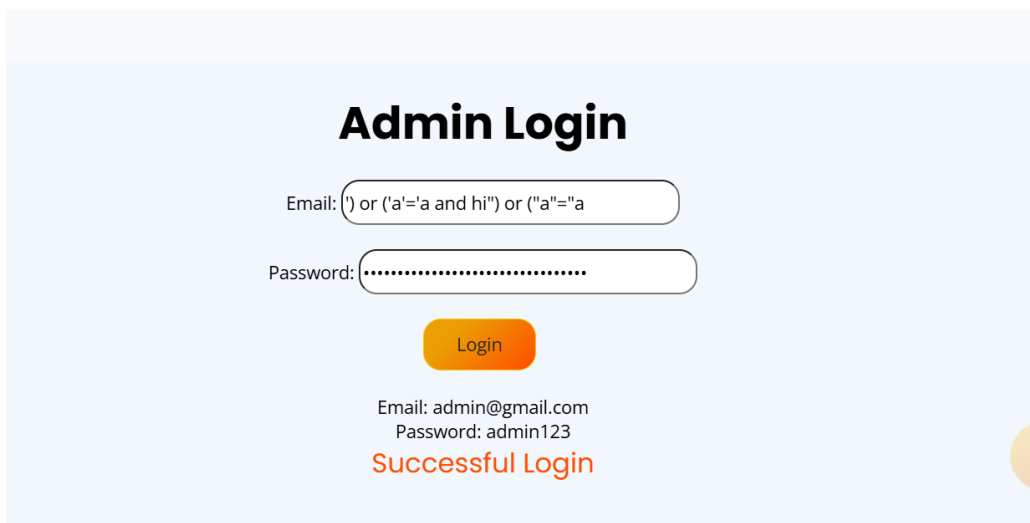
Proof of Concept



1.6. ERROR BASED:TRICKED!

Reference	Risk Rating
ERROR BASED:TRICKED!	MEDIUM
Tools Used	
manual	
Vulnerability Description	
<p>The SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.</p> <p>Attackers insert malicious SQL code into input fields (like login forms) that are not properly sanitized. Unauthorized Data Access: This can lead to unauthorized access to sensitive data, such as user credentials, personal information, and financial records.</p>	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sql_i_lab/lab_6/lab_6.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Data Breaches: Unauthorized access to sensitive data.• Data Manipulation: Alteration or deletion of data.• System Compromise: In some cases, attackers can gain control over the entire database server.	
Suggested Countermeasures	
<ul style="list-style-type: none">• Input Validation: Ensure all user inputs are properly validated and sanitized.• Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.• Stored Procedures: Utilize stored procedures to handle database operations.	
References	
https://portswigger.net/web-security/sql-injection	

Proof of Concept



Admin Login

Email:

Password:

Email: admin@gmail.com
Password: admin123

Successful Login

1.7. ERROR and Post

Reference	Risk Rating
ERROR and Post!	low
Tools Used	
manual	
Vulnerability Description	
<p>The SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.</p> <p>Attackers insert malicious SQL code into input fields (like login forms) that are not properly sanitized. Unauthorized Data Access: This can lead to unauthorized access to sensitive data, such as user credentials, personal information, and financial records.</p>	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sql_i_lab/lab_7/lab_7.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Data Breaches: Unauthorized access to sensitive data.• Data Manipulation: Alteration or deletion of data.• System Compromise: In some cases, attackers can gain control over the entire database server.	
Suggested Countermeasures	
<ul style="list-style-type: none">• Input Validation: Ensure all user inputs are properly validated and sanitized.• Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.• Stored Procedures: Utilize stored procedures to handle database operations.	
References	
https://portswigger.net/web-security/sql-injection	

Proof of Concept

Admin Login

Email:

Password:

Login

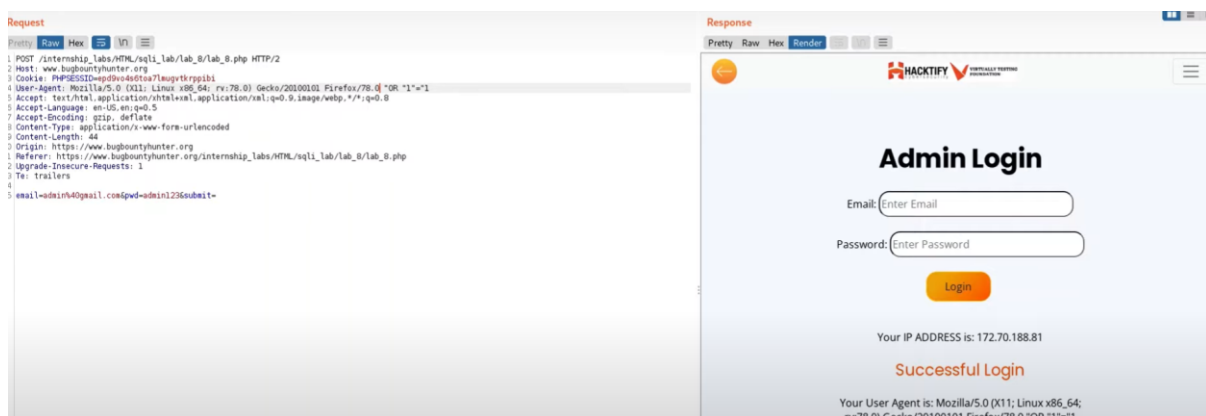
Email: admin@gmail.com
Password: admin123

Successful Login

1.8. User Agents lead us!

Reference	Risk Rating
User Agents lead us	high
Tools Used	
burpsuite	
Vulnerability Description	
<p>The SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.</p> <p>Attackers insert malicious SQL code into input fields (like login forms) that are not properly sanitized. Unauthorized Data Access: This can lead to unauthorized access to sensitive data, such as user credentials, personal information, and financial records.</p>	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sql_i_lab/lab_8/lab_8.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Data Breaches: Unauthorized access to sensitive data.• Data Manipulation: Alteration or deletion of data.• System Compromise: In some cases, attackers can gain control over the entire database server.	
Suggested Countermeasures	
<ul style="list-style-type: none">• Input Validation: Ensure all user inputs are properly validated and sanitized.• Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.• Stored Procedures: Utilize stored procedures to handle database operations.	
References	
https://portswigger.net/web-security/sql-injection	

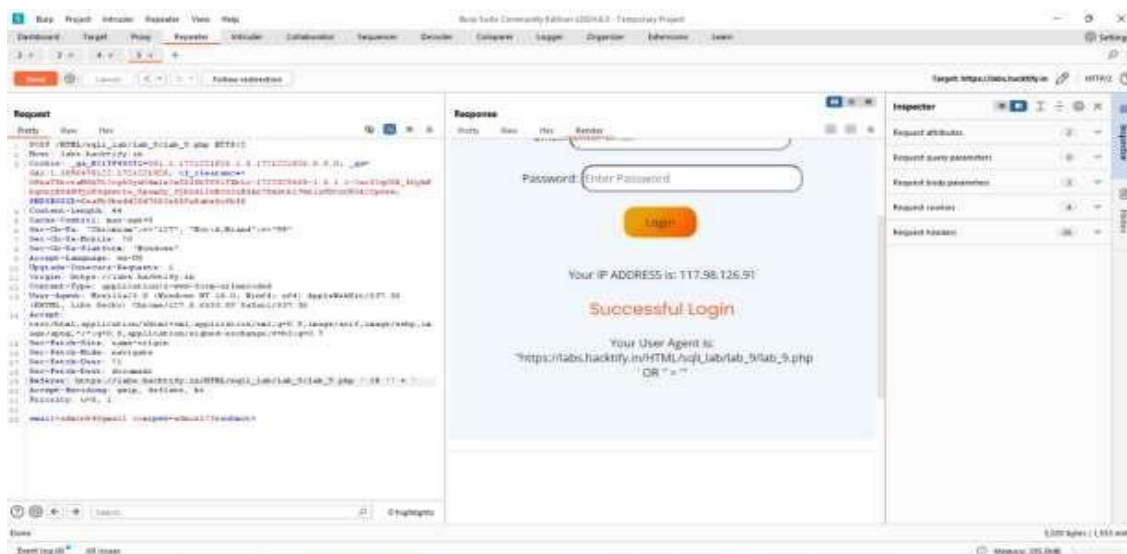
Proof of Concept



1.9. Referer lead us!

Reference	Risk Rating
Referer lead us	MEDIUM
Tools Used	
burpsuite	
Vulnerability Description	
<p>Attackers modify the “User-Agent” header in their HTTP requests to include malicious SQL code. If the application logs or processes the “User-Agent” header without proper sanitization, the malicious code can be executed as part of an SQL query.</p> <p>This can lead to unauthorized access, data manipulation, or even full control over the database.</p>	
How It Was Discovered	
Automated Tools -burp	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Data Breaches: Unauthorized access to sensitive data.• Data Manipulation: Alteration or deletion of data.• System Compromise: In some cases, attackers can gain control over the entire database server.	
Suggested Countermeasures	
<ul style="list-style-type: none">• Input Validation: Ensure all user inputs are properly validated and sanitized.• Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.• Stored Procedures: Utilize stored procedures to handle database operations.	
References	
https://portswigger.net/web-security/sql-injection	

Proof of Concept



1.10. Oh Cookies!!

Reference	Risk Rating
Oh Cookies!	high
Tools Used	
burpsuite	
Vulnerability Description	
<p>Attackers modify the “User-Agent” header in their HTTP requests to include malicious SQL code. If the application logs or processes the “User-Agent” header without proper sanitization, the malicious code can be executed as part of an SQL query.</p> <p>This can lead to unauthorized access, data manipulation, or even full control over the database.</p>	
How It Was Discovered	
Automated Tools -burp	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sql_i_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Data Breaches: Unauthorized access to sensitive data.• Data Manipulation: Alteration or deletion of data.• System Compromise: In some cases, attackers can gain control over the entire database server.	
Suggested Countermeasures	
<ul style="list-style-type: none">• Input Validation: Ensure all user inputs are properly validated and sanitized.• Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.• Stored Procedures: Utilize stored procedures to handle database operations.	
References	
https://portswigger.net/web-security/sql-injection	

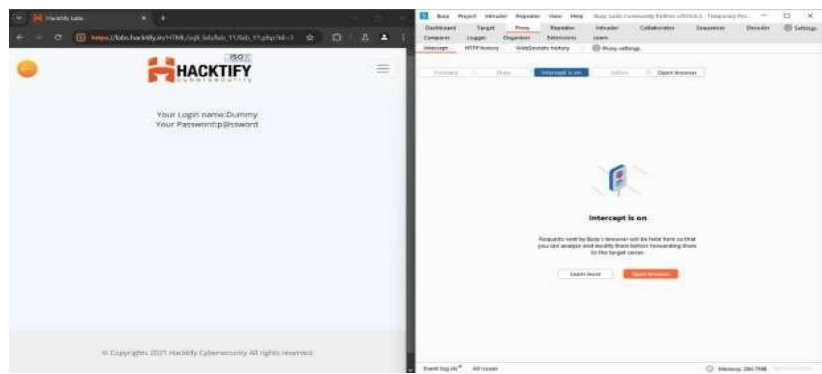
Proof of Concept

The image displays a proof of concept for a SQL injection vulnerability. On the left, the Burp Suite HTTP history shows a POST request to `/internship_labs/HTML/sql_i_lab/lab_10/lab_10.php`. The request includes a malicious `User-Agent` header: `Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100103 Firefox/78.0 union SELECT version(),user(),database()#`. On the right, the web application's login page is shown. The page title is "User Login". The login form has fields for "Username" and "Password". Below the form, a "Login" button is visible. The page displays a "Successful Login" message and the text "I LOVE YOU COOKIES".

1.11 WAF's are injected!

Reference	Risk Rating
Sub-lab-11: WAF's are injected	High
Tools Used	
Burp Suite	
Vulnerability Description	
Attackers use sophisticated evasion techniques to manipulate SQL queries in a way that bypasses WAF filters. This can include encoding, obfuscation, and altering the structure of the SQL payload.	
How It Was Discovered	
Automated Tool--- Burp-suite	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_12/hacked.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none"> • Data Breaches: Unauthorized access to sensitive data. • Data Manipulation: Alteration or deletion of data. • System Compromise: Potential control over the entire database server 	
Suggested Countermeasures	
<ol style="list-style-type: none"> 1. Advanced WAF Configuration: Regularly update and configure WAF rules to handle new evasion techniques. 2. Input Validation: Implement robust input validation and sanitization on the server side. 3. Parameterized Queries: Use parameterized queries or prepared statements to handle inputs safely. 4. Regular Security Audits: Conduct regular security audits and penetration testing to identify and fix potential vulnerabilities. 	
References	
https://portswigger.net/web-security/sql-injection https://www.varonis.com/blog/what-is-sql-injection https://owasp.org/www-community/attacks/SQL_Injection	

Proof of Concept



1.12. WAF's are injected Part 2!

Reference	Risk Rating
WAF's are injected Part 2!	medium
Tools Used	
burpsuite	
Vulnerability Description	
<p>Attackers modify the “User-Agent” header in their HTTP requests to include malicious SQL code. If the application logs or processes the “User-Agent” header without proper sanitization, the malicious code can be executed as part of an SQL query.</p> <p>This can lead to unauthorized access, data manipulation, or even full control over the database.</p>	
How It Was Discovered	
Automated Tools -burp	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Data Breaches: Unauthorized access to sensitive data.• Data Manipulation: Alteration or deletion of data.• System Compromise: In some cases, attackers can gain control over the entire database server.	
Suggested Countermeasures	
<ul style="list-style-type: none">• Input Validation: Ensure all user inputs are properly validated and sanitized.• Parameterized Queries: Use parameterized queries or prepared statements to prevent SQL code injection.• Stored Procedures: Utilize stored procedures to handle database operations.	
References	
https://portswigger.net/web-security/sql-injection	

Proof of Concept

The screenshot displays a Burp Suite interface with two panels. The left panel, titled 'Raw', shows an HTTP request from a Firefox browser. The 'User-Agent' header is modified to include a malicious SQL payload: `Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0 t: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 t-Language: en-US,en;q=0.5 t-Encoding: gzip, deflate ar: https://www.bugbountyhunter.org/internship_labs/HTML/sqli_lab/lab_12/hacked.php de-Insecure-Requests: 1 -Control: max-age=0 railers`. The right panel, titled 'Response', shows the application's response, which is a 200 OK status from `labs.hacktify.in`. The response body contains the text: `Your Login name:Dumb` and `Your Password:Dumb`, indicating that the malicious payload successfully executed a SQL query to retrieve user credentials.

2.IDOR

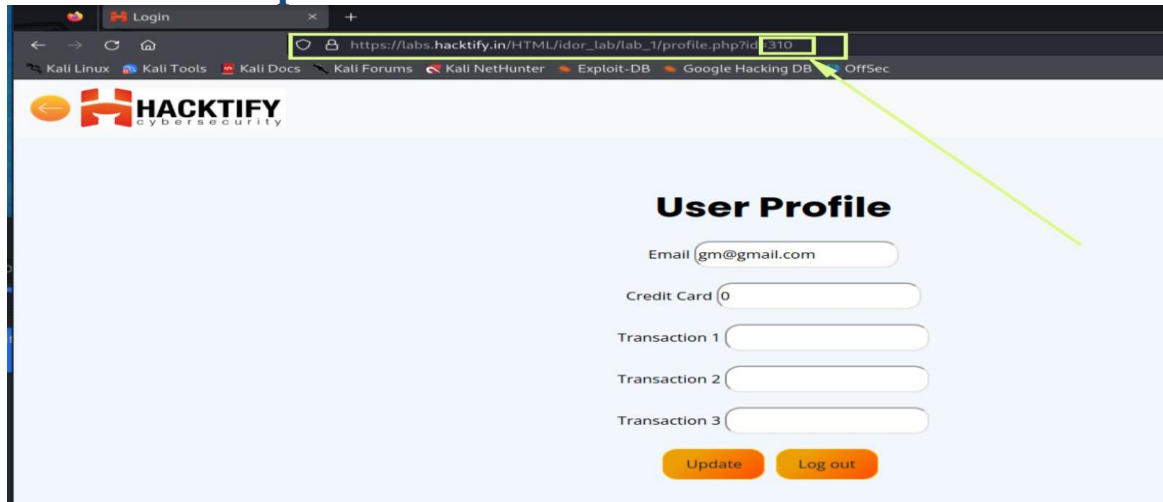
2.2. Give Me My Amount!

Reference	Risk Rating
Give Me My Amount!	Low
Tools Used	
Firefox Browser	
Vulnerability Description	
<p>A Cross-Site Scripting (XSS) vulnerability can occur when a web application allows user input to be included in a URL without properly validating or sanitizing the input. This can potentially allow an attacker to inject malicious JavaScript code into the URL, which can then be executed by a victim user's browser when they visit the URL.</p> <p>For example, consider a web application that includes a user-supplied id parameter in a URL, such as <code>https://example.com/user?id=123</code>. If the web application does not properly validate or sanitize the id parameter, an attacker could potentially inject malicious JavaScript code into the parameter value, such as <code>https://example.com/user?id=<script>alert(1)</script></code>.</p> <p>When the victim user's browser loads the URL with the malicious JavaScript code, it will execute the JavaScript code in the context of the vulnerable web page.</p>	
How It Was Discovered	
Manual Analysis	

Vulnerable URLs
<code>https://labs.hacktify.in/HTML/idor_lab/lab_1/profile.php?id=310</code>
Consequences of not Fixing the Issue
<p>The consequences of not fixing the id parameter in a URL XSS vulnerability can be severe, as it can allow attackers to inject malicious JavaScript code into a web application. This could allow attackers to:</p> <p>Steal sensitive information: The malicious JavaScript code could steal sensitive information, such as login credentials, credit card numbers, or other personal data, by sending it to the attacker's server.</p> <p>Redirect users to malicious websites: The malicious JavaScript code could redirect users to malicious websites that may contain malware, phishing scams, or other threats.</p>
Suggested Countermeasures
<p>The following countermeasures can be implemented to address the id parameter in URL XSS vulnerability:</p> <p>Input validation: Validate the id parameter to ensure that it does not contain malicious characters or code. This can be done using a variety of techniques, such as regular expressions or blacklists.</p> <p>Output encoding: Encode the id parameter before it is displayed in the web application. This can be done using a variety of techniques, such as HTML entity encoding or URL encoding.</p> <p>Use a web application firewall (WAF): A WAF can help to block malicious requests that attempt to exploit XSS vulnerabilities.</p> <p>Educate users about XSS attacks: Users should be aware of the risks of XSS attacks and should avoid clicking on suspicious links or opening untrusted files.</p>
References

<https://medium.com/bugbountywriteup/xss-through-parameter-pollution-9a55da150ab2>

Proof of Concept

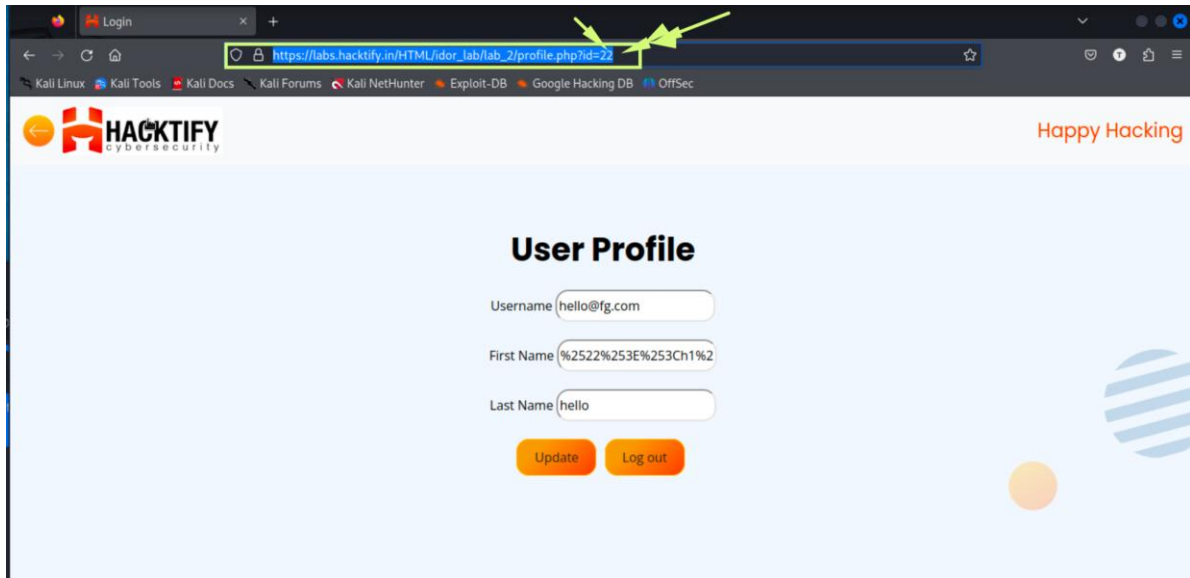


2.2. Stop polluting my params!

Reference	Risk Rating
Stop polluting my params!	Medium
Tools Used	
Firefox Browser	
Vulnerability Description	
<p>A URL ID parameter XSS vulnerability occurs when an attacker is able to inject malicious JavaScript code into a web application via the id parameter in a URL. This can allow the attacker to execute arbitrary code on the victim's computer, which could lead to a variety of security risks, such as:</p> <p>Stealing sensitive information Redirecting users to malicious websites Installing malware</p>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_2/profile.php?id=22	

Consequences of not Fixing the Issue
Not fixing the id parameter in a URL XSS vulnerability can also have specific consequences for the affected web application. For example, if the web application is used for e-commerce, an attacker could use the vulnerability to steal credit card numbers or other sensitive financial information.
Suggested Countermeasures
<p>The best way to prevent URL ID parameter XSS vulnerabilities is to validate and encode all user input. This can be done using a variety of techniques, such as:</p> <p>Input validation: Validate the id parameter to ensure that it does not contain malicious characters or code.</p> <p>Output encoding: Encode the id parameter before it is displayed in the web application.</p> <p>By implementing these measures, web developers can help to protect their applications from XSS attacks.</p>
References
https://medium.com/bugbountywriteup/xss-through-parameter-pollution-9a55da150ab2

Proof of Concept



2.3. Someone changed my Password!

Reference	Risk Rating
Someone changed my Password!	High
Tools Used	
Firefox Browser	
Vulnerability Description	
An IDOR vulnerability can occur when a web application uses predictable or user-supplied input to directly access objects or resources without proper authorization checks. In the context of a username	
in a URL, this vulnerability can arise when the application allows users to access resources based solely on the username included in the URL, without validating whether the user has the necessary permissions to access those resources.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_3/lab_3.php	
Consequences of not Fixing the Issue	
By exploiting this IDOR vulnerability, an attacker could gain unauthorized access to sensitive user data, such as personal information, financial records, or private messages. They could also potentially modify or delete data, or perform other actions that they should not be authorized to do.	
Suggested Countermeasures	

To mitigate IDOR vulnerabilities related to usernames in URLs, it is important to implement the following measures:

Validate the username parameter to ensure that the user has the necessary permissions to access the requested resource.

Use unique and unpredictable identifiers for objects and resources, rather than relying on user-supplied values.

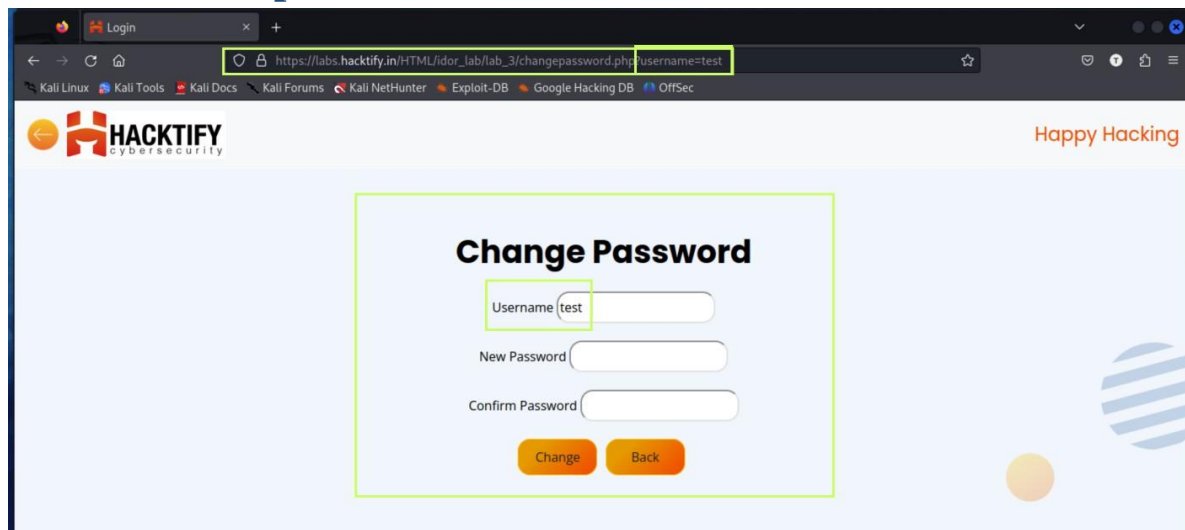
Implement role-based access controls to restrict access to specific resources based on the user's role or permissions.

Regularly review and test your application for potential IDOR vulnerabilities.

References

<https://portswigger.net/web-security/access-control/idor>

Proof of Concept



2.4. Change your methods!

Reference	Risk Rating
Change your methods!	Medium
Tools Used	
Firefox Browser	
Vulnerability Description	
<p>An IDOR vulnerability can occur when a web application uses predictable or user-supplied input to directly access objects or resources without proper authorization checks. In the context of a POST request, this vulnerability can arise when the application allows users to modify or delete resources based on user-supplied input, without validating whether the user has the necessary permissions to perform those actions.</p>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_4/profile.php?id=625	
Consequences of not Fixing the Issue	
<p>By exploiting this IDOR vulnerability, an attacker could delete or modify any post on the website, regardless of their permissions. This could have a significant impact on the integrity and availability of the application's data.</p>	
Suggested Countermeasures	
<p>To mitigate IDOR vulnerabilities related to POST requests, it is important to implement the following measures:</p> <p>Validate the post_id parameter in the POST request body to ensure that the user has the necessary permissions to delete or modify the specified post.</p> <p>Use unique and unpredictable identifiers for objects and resources, rather than relying on user-supplied values.</p> <p>Implement role-based access controls to restrict access to specific resources based on the user's role or permissions.</p> <p>Regularly review and test your application for potential IDOR vulnerabilities..</p>	
References	
https://portswigger.net/web-security/idor	

Proof of Concept

