

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.

Application Name	{IDOR}, {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
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 value of id parameter in the URL.	
Vulnerable URLs	
https://labs.hackify.in/HTML/idor_lab/lab_1/profile.php?id=1869	
Consequences of not Fixing the Issue	
This vulnerability violates the privacy of a User.	
Suggested Countermeasures	
<ol style="list-style-type: none">1. Developers should avoid displaying private object references such as keys or file names.2. Validation of parameters should be properly implemented.3. Verification of all the referenced objects should be checked.4. Tokens should be generated in such a way that they can only be mapped to the user and is not public.	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/access-control/idor2. https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html3. https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/	
Payload Used	
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 window with the URL labs.hackify.in/HTML/idor_lab/lab_1/profile.php?id=1869. The page is titled "User Profile". It features several input fields: "Email" (containing "mail@mail.com"), "Credit Card" (containing "0"), "Transaction 1", "Transaction 2", and "Transaction 3". At the bottom are two buttons: "Update" and "Log out". The page is branded with the "HACKIFY cybersecurity" logo and an ISO 27001 certification seal. A "Happy Hacking" message is visible in the top right corner. The footer contains the copyright notice: "© 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	
https://labs.hacktify.in/HTML/idor_lab/lab_2/profile.php?id=1327	
Consequences of not Fixing the Issue	
This vulnerability violates the privacy of a User.	
Suggested Countermeasures	
<ol style="list-style-type: none">1. Developers should avoid displaying private object references such as keys or file names.2. Validation of parameters should be properly implemented.3. Verification of all the referenced objects should be checked.4. Tokens should be generated in such a way that they can only be mapped to the user and is not public.	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/access-control/idor2. https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html3. https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/	
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

The screenshot shows a web browser displaying a user profile page from the URL https://labs.hackify.in/HTML/idor_lab/lab_2/profile.php?id=1327. The page is titled "User Profile". It contains three text input fields:

- Username: hacker05@gmail.com
- First Name: Hacker06
- Last Name: Hacker06

Below the inputs are two orange buttons: "Update" and "Log out". The browser's address bar shows the full URL. The top right corner of the browser window has the text "Happy Hacking". The overall layout is clean, with a white background and a light gray header.

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	
https://labs.hacktify.in/HTML/idor_lab/lab_3/changepassword.php?username=hacker	
Consequences of not Fixing the Issue	
This vulnerability violates the privacy of a User.	
Suggested Countermeasures	
<ol style="list-style-type: none">1. Developers should avoid displaying private object references such as keys or file names.2. Validation of parameters should be properly implemented.3. Verification of all the referenced objects should be checked.4. Tokens should be generated in such a way that they can only be mapped to the user and is not public.	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/access-control/idor2. https://cheatsheetseries.owasp.org/cheatsheets/Insecure Direct Object Reference Prevention Cheat Sheet.html3. https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/	
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

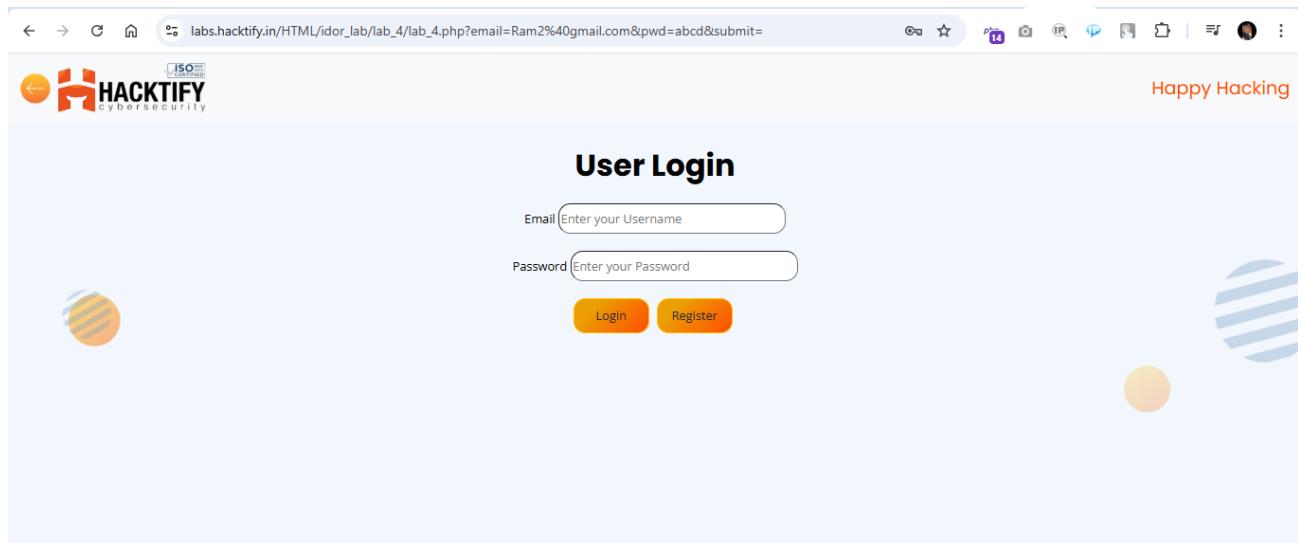
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.4. Change your methods!

Reference	Risk Rating
Change your methods!	Medium
Tools Used	
Google chrome	
Vulnerability Description	
<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	
<p>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.</p>	
Vulnerable URLs	
<p>https://labs.hacktify.in/HTML/idor_lab/lab_4/lab_4.php?email=Ram2%40gmail.com&pwd=abcd&submit=</p>	
Consequences of not Fixing the Issue	
<p>This vulnerability violates the privacy of a User.</p>	
Suggested Countermeasures	
<ol style="list-style-type: none">1. Developers should avoid displaying private object references such as keys or file names.2. Validation of parameters should be properly implemented.3. Verification of all the referenced objects should be checked.4. Tokens should be generated in such a way that they can only be mapped to the user and is not public.	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/access-control/idor2. https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html3. https://www.bugcrowd.com/blog/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/	
Payload Used	
<p>Appending "?id=2157" at last in the URL followed by updating random details in the profile section of different user.</p>	

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 login form. The address bar shows the URL: `labs.hacktify.in/HTML/idor_lab/lab_4.php?email=Ram2%40gmail.com&pwd=abcd&submit=`. The page has a light blue header with the Hacktify Cybersecurity logo and the text "Happy Hacking". The main content area is titled "User Login" and contains two input fields: "Email" and "Password", both with placeholder text "Enter your Username" and "Enter your Password" respectively. Below the inputs are two orange buttons labeled "Login" and "Register". The background of the page features abstract circular shapes in orange and blue.

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

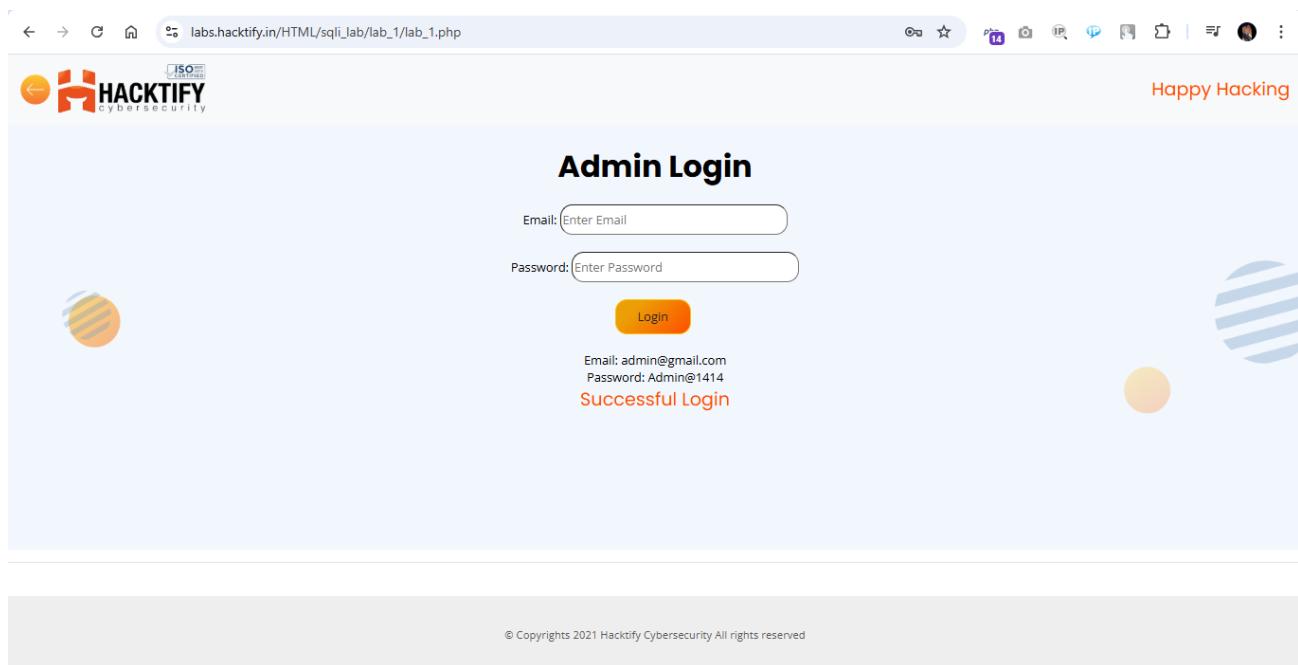
2. SQL Injection

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	
https://labs.hackify.in/HTML/sql_injection/lab_1/lab_1.php	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
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

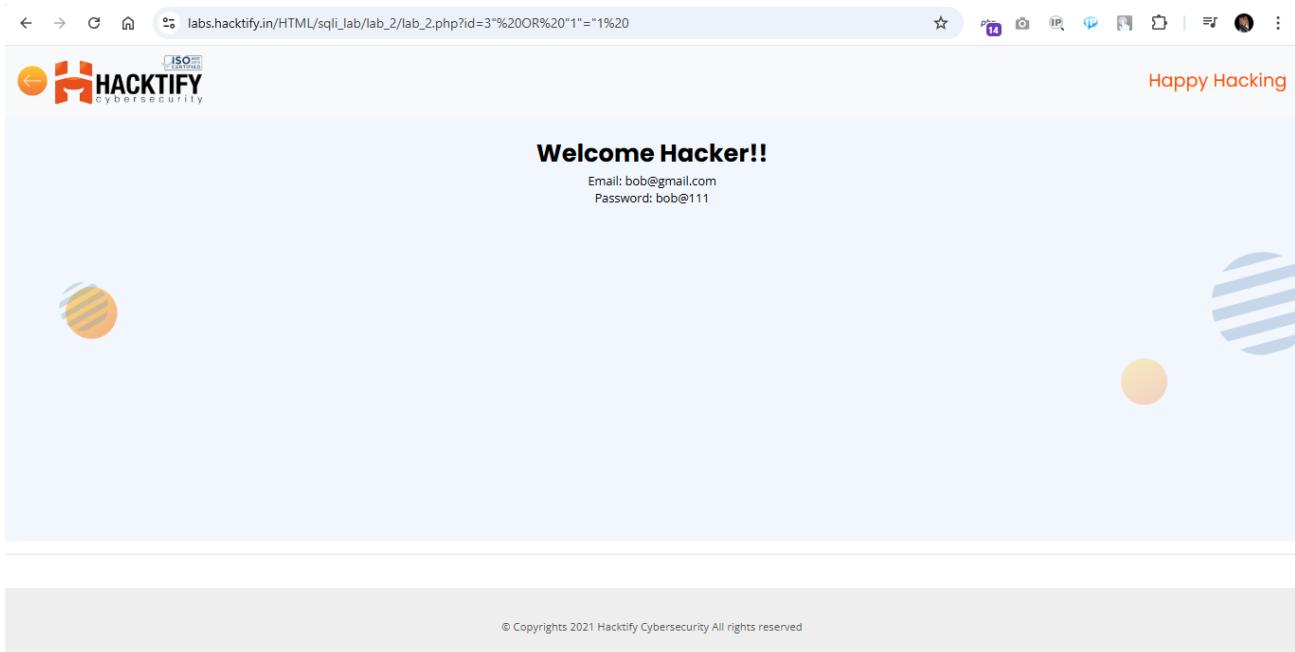


2.2. Strings and Errors Part2!

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	
https://labs.hackify.in/HTML/sql_injection/lab_2/lab_2.php?id=3%22%20OR%20%221%22=%221%20	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
?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	
https://labs.hacktify.in/HTML/sql_injection/lab_3/lab_3.php?id=1%22%20OR%20%221%22=%221%20	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
?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	
https://labs.hackify.in/HTML/sql_injection/lab_4/lab_4.php	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
1' '1='1	

Proof of Concept

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

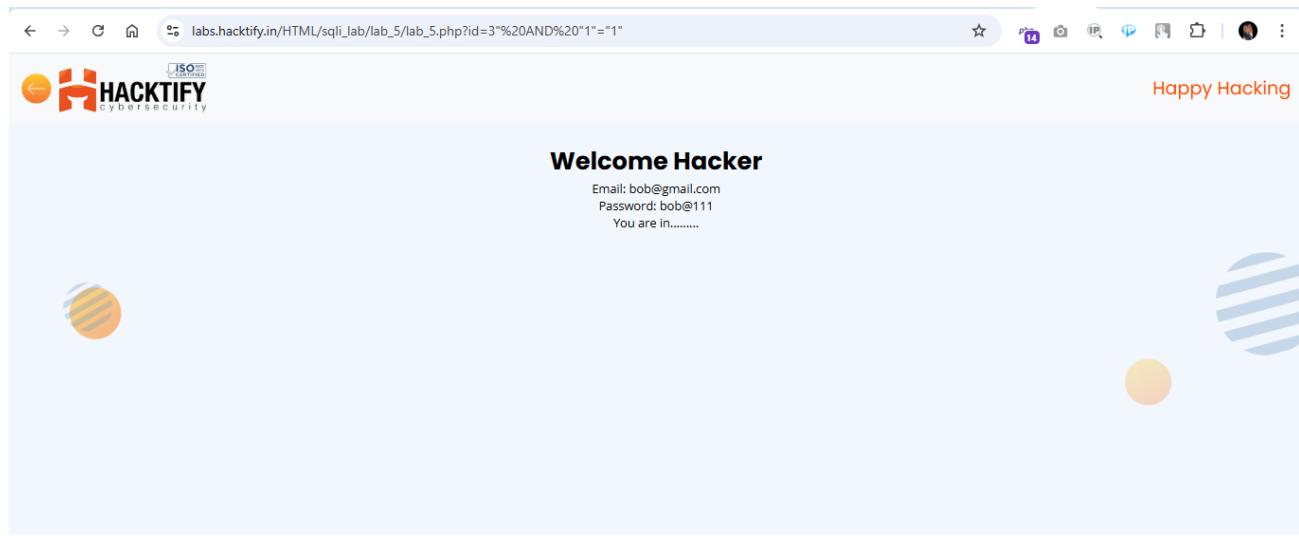
The screenshot shows a web browser window with the URL `labs.hackify.in/HTML/sql_injection/lab_4/lab_4.php` in the address bar. The page title is "Admin Login". On the left, there is a logo for "HACKIFY cybersecurity" with an ISO 27001 certification badge. On the right, the text "Happy Hacking" is displayed. The main content area contains two input fields: "Email: Enter Email" and "Password: Enter Password", both currently empty. Below these fields is a large orange "Login" button. To the right of the password field, the text "Email: admin@gmail.com" and "Password: admin123" is visible. Below the "Login" button, the message "Successful Login" is displayed in orange. The browser's toolbar at the top includes icons for back, forward, search, and other standard functions.

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	
https://labs.hackify.in/HTML/sql_injection/lab_5/lab_5.php?id=3%22%20AND%20%221%22=%221%22	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
?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	
https://labs.hackify.in/HTML/sql_injection/lab_6/lab_6.php	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
") 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. At the top of the browser window, there are standard navigation icons and a toolbar.

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	
https://labs.hackify.in/HTML/sql_injection/lab_7/lab_7.php	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
' or '1'='1	

Proof of Concept

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

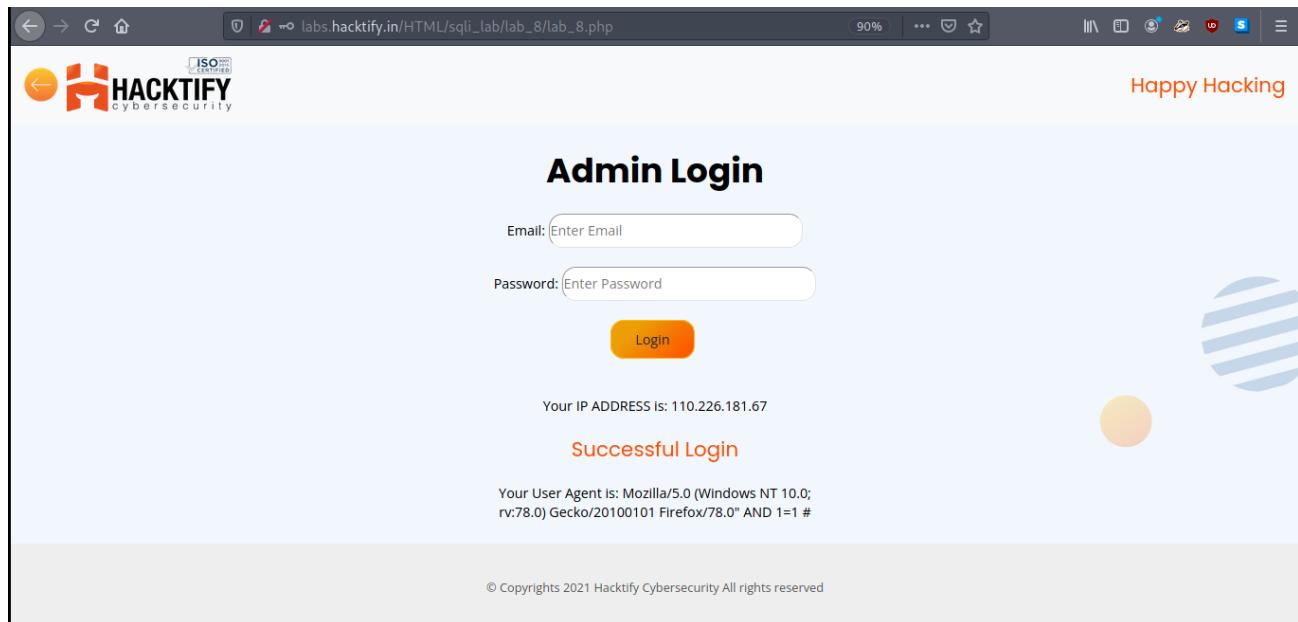
The screenshot shows a web browser window with the URL `labs.hackify.in/HTML/sqli_lab/lab_7/lab_7.php`. The page is titled "Admin Login". It features two input fields: "Email: Enter Email" and "Password: Enter Password", both currently empty. Below these fields is an orange "Login" button. To the right 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 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	
https://labs.hackify.in/HTML/sql_injection/lab_8/lab_8.php	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
" 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	
https://labs.hackify.in/HTML/sql_injection/lab_9/lab_9.php	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
' 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	
https://labs.hackify.in/HTML/sql_injection/lab_10/lab_10.php	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
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 the URL `labs.hackify.in/HTML/sqli_lab/lab_10/lab_10.php`. The page content includes the Hackify logo, user agent information, IP address, and a cookie message. On the right, the Chrome DevTools Application tab is open, displaying a table of cookies. The table has columns for Name, Value, and several checkboxes labeled D, P, E, S, H, S, P, C, and M. Two rows are present:

Name	Value	D	P	E	S	H	S	P	C	M
PHPSESS...	75ad3201f53b5e7...	L...	/	S...	41	✓				M...
username	1' OR '1='1	L...	/...	2...	20	✓				M...

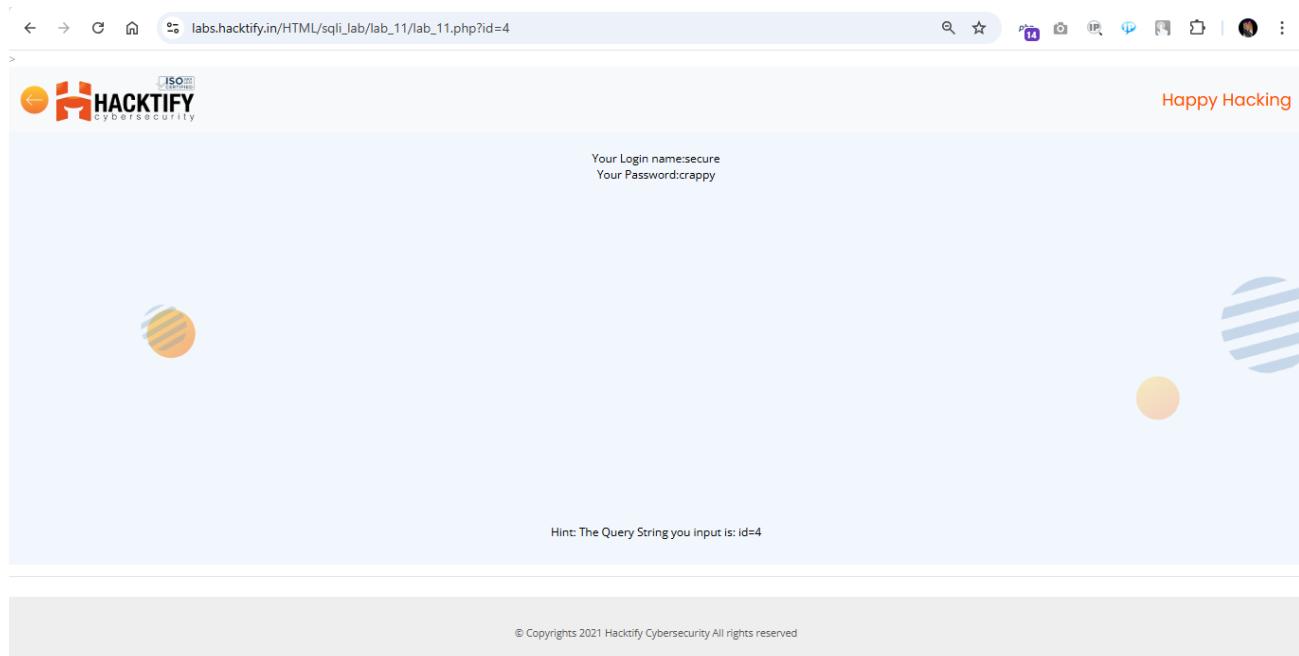
A tooltip points to the first cookie row, identifying it as `https://labs.hackify.in`. Below the table, a message says "Select a cookie to preview its value". At the bottom of the DevTools panel, there's a "What's new" section.

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	
https://labs.hackify.in/HTML/sql_injection/lab_11/lab_11.php?id=4	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
?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	
https://labs.hackify.in/HTML/sql_injection/lab_12/lab_12.php?id=8	
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">1. Implement Input Validation and Sanitization2. Using Escaping for User Input3. Utilize Parameterized Statements (Prepared Statements)4. Incorporate Stored Procedures5. Conduct Continuous Scanning and Penetration Testing6. Adopt the Least Privilege Principle7. Deploy Web Application Firewalls (WAF)	
References	
<ol style="list-style-type: none">1. https://portswigger.net/web-security/sql-injection/cheat-sheet2. https://portswigger.net/web-security/sql-injection3. https://owasp.org/www-community/attacks/SQL_Injection	
Payload Used	
?id=8	

Proof of Concept

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

