Web Penetration Testing Report

Full Name: Geshom Lukoshi
Program: HCPT
Date: 24th February, 2025

Introduction

This report document hereby describes the proceedings and results of the web site vulnerabilities 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 strategies and recommendation guidelines to help mitigate the identified vulnerabilities and risks during the activity.

2. Scope

The scope of this penetration testing will cover **(02) labs** by **Hactify Cyber Security**. The tested attacks are the web application flaws **SQL injection** and **Insecure Direct Object Reference (IDOR)**. The testing will include identifying vulnerabilities, ways in which these flaws can be taken advantage off to exploit the vulnerable web applications. The testing will focus on detecting the **SQLi** and **IDOR** vulnerabilities that could lead to unauthorized access and actions on the web applications by unauthorized users **(attackers)**. The injection point entries such as URL parameter, login forms and manipulation of the source code will be assessed to identify potential avenues for malicious code insertion. The boundaries of this penetration testing labs exclude the testing network infrastructure of the web application. The results will be provided with recommendations for mitigation to enhance the web application security posture.

Application Name	**Lab 1 – SQL Injection** **Lab 2 – Insecure Direct Object Reference (IDOR)**

3. Summary

Outlined is the Hackitify web site Security assessment for the **Week 2 Labs**.

Total number of Sub-labs: (17) 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. Lab 1 SQL Injection

1.1. Sub-lab-1 Strings and errors part 1

Reference	**Risk Rating**
Sub-lab-1 Strings and errors part 1	**Low**
Tools Used	
Browser	
Vulnerability Description	
SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. SQL injections typically fall under three categories**: In-band SQLi** (Classic), **Inferential SQLi** (Blind) and **Out-of-band SQLi**. You can classify SQL injections types based on the methods they use to access backend data and their damage potential. In this case an **Error-based SQLi**— was used, in this regard the attacker performs actions that cause the database to produce error messages. The attacker can potentially use the data provided by these error messages to gather information about the structure of the database.	
How It Was Discovered	
Automated Tools – Browser View Source Page	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
Not patching SQL Injection (SQLi) vulnerabilities can have severe impacts on the web application, database, and organization: 1. Unauthorized data access 2. Data tampering 3. Data exfiltration	

4. Malware injection
5. Privilege escalation
6. Lateral movement

It's essential to prioritize patching SQLi vulnerabilities to prevent these severe impacts and protect the organization's sensitive data and systems.

Suggested Countermeasures

To prevent SQL injection (SQLi) attacks, the following countermeasures should be considered:

1. **Input Validation and Sanitization**
   - Validate user input: Ensure that user input conforms to expected formats and patterns.
   - Sanitize user input: Remove or escape any special characters that could be used to inject malicious SQL code.

2. **Secure Coding Practices**
   - Use prepared statements: Prepared statements separate code from user input, preventing injection attacks.
   - Use parameterized queries: Parameterized queries pass user input as parameters, rather than concatenating it into the query.
   - Avoid dynamic SQL: Dynamic SQL can be vulnerable to injection attacks; instead, use static SQL or prepared statements.

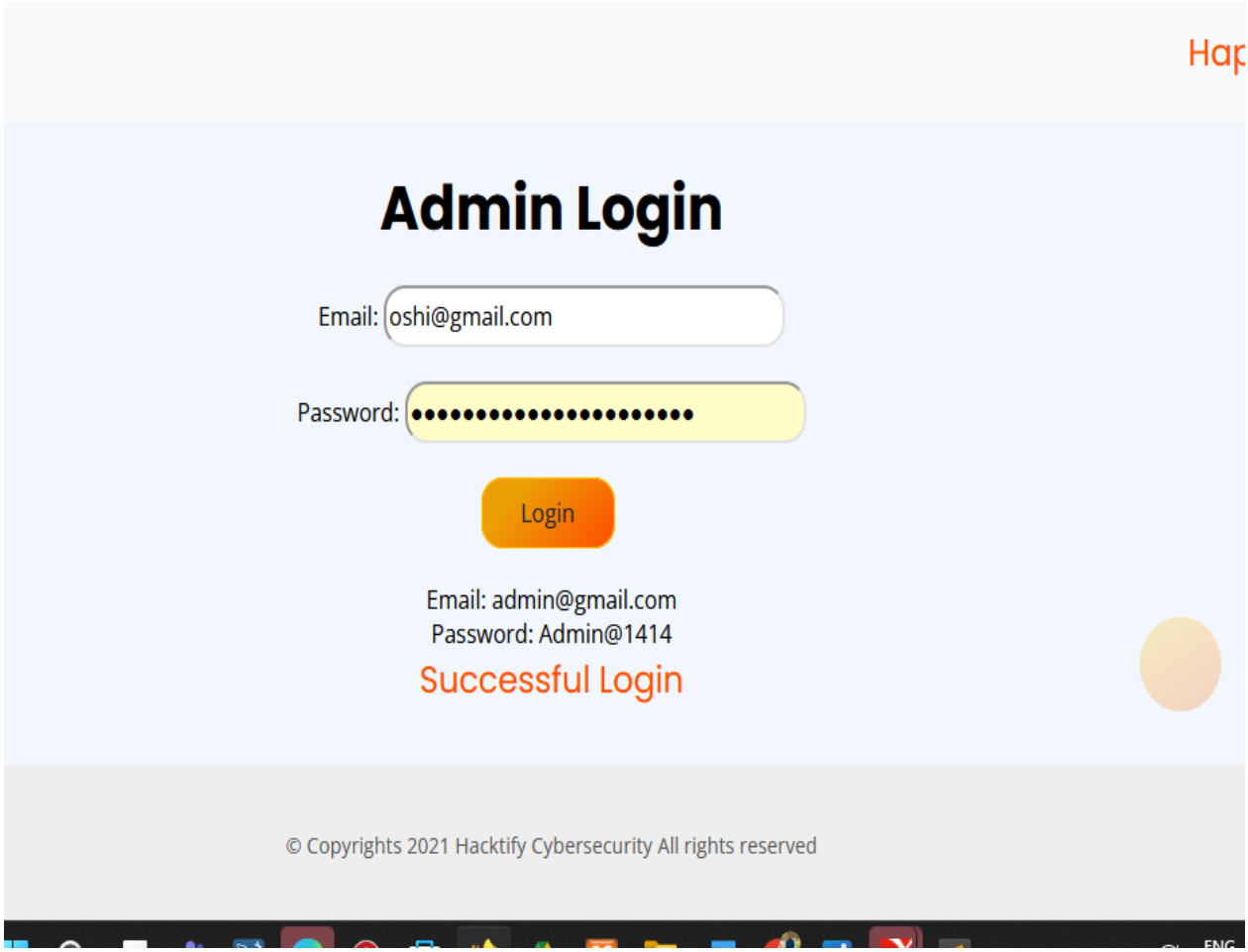
3. **Database Configuration and Patching**
   - Keep databases and frameworks up-to-date: Regularly update databases and frameworks to ensure you have the latest security patches.
   - Configure databases securely: Implement secure database configurations, such as limiting privileges and enabling logging.
   - Use a web application firewall (WAF): A WAF can help detect and prevent SQLi attacks.

References

https://owasp.org
https://www.acunetix.com
https://github.com
https://www.imperva.com/learn/application-security/sql-injection-sqli/

Proof of Concept

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



1.2. Sub-lab-2 Strings and errors part 2

Reference	**Risk Rating**
Sub-lab-2: Strings and errors	**Low**
Tools Used	
Browser	
Vulnerability Description	
SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. SQL injections typically fall under three categories**: In-band SQLi** (Classic), **Inferential SQLi** (Blind) and **Out-of-band SQLi**. You can classify SQL injections types based on the methods they use to access backend data and their	

damage potential. In this case an **Error-based SQLi**— was used, in this regard the attacker performs actions that cause the database to produce error messages. The attacker can potentially use the data provided by these error messages to gather information about the structure of the database.

How It Was Discovered

Automated Tools “By trying the entry points of injecting looking for vulnerabilities”.

Vulnerable URLs

https://labs.hacktify.in/HTML/sqli_lab/lab_2/lab_2.php?id=1%27%20UNION%20SELECT%20username,%20password%20FROM%20users%20--

Consequences of not Fixing the Issue

Not Fixing SQL Injection (SQLi) vulnerabilities can have severe impacts on your web application, database, and organization:

1. Denial of Service (DoD)
2. Data Breaches
3. Increased Risk of Future Attacks
4. Regulatory compliance
5. System downtime
6. Competitive Disadvantage

It's essential patching SQLi vulnerabilities to prevent these severe impacts and protect your organization's sensitive data and systems.

Suggested Countermeasures

SQL injection attack prevention involves a positively engaged approach to maintaining the security of the web applications. The likelihood of an attack can significantly be brought to light with the implementation of the following techniques:

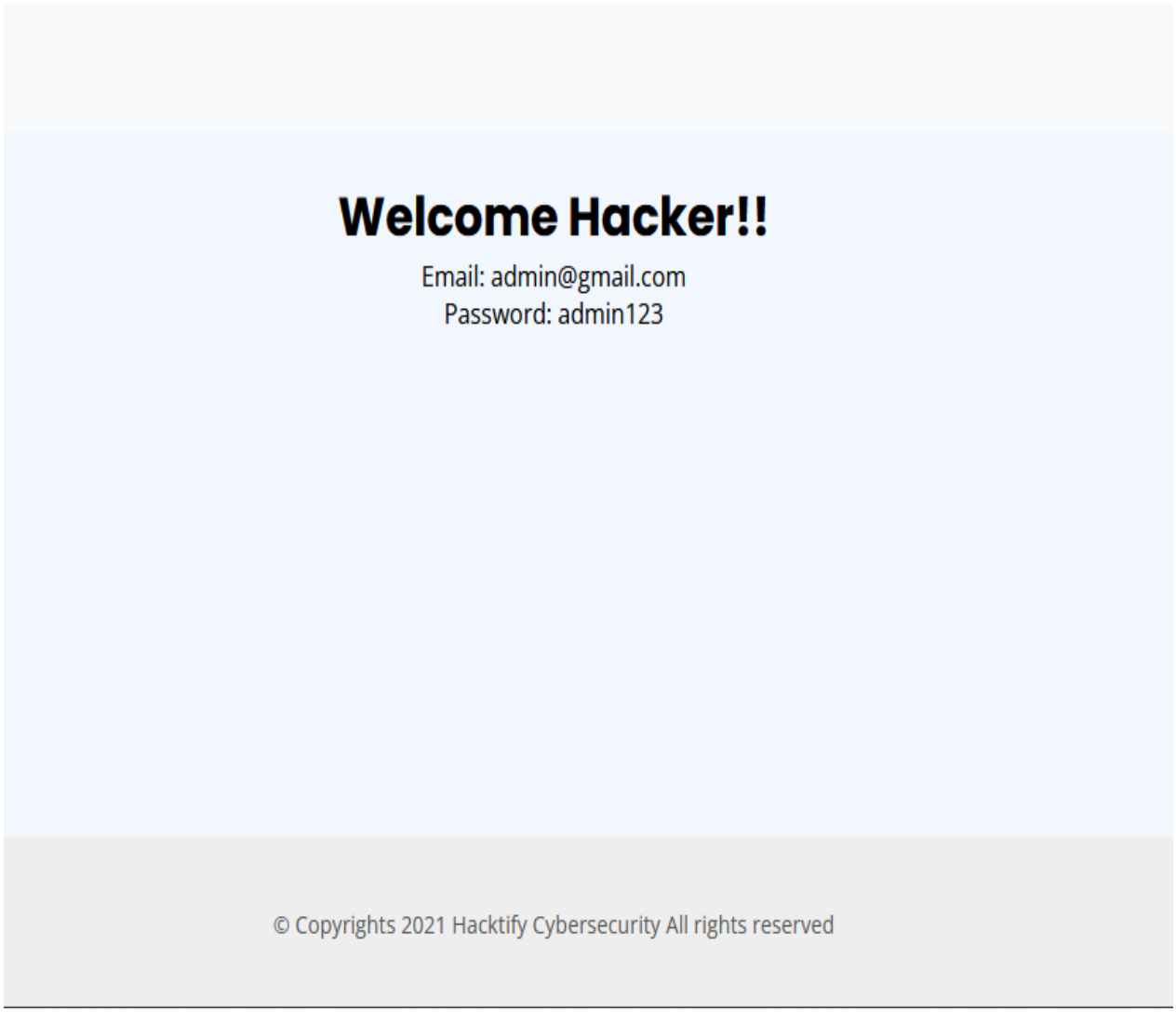
1. Prepared Statements and Parameterized Queries
2. Input Validation.
3. Web Application Firewall or WAF
4. Data Sanitization
5. Least Privilege Access Principle

References

https://www.sentinelone.com
https://www.imperva.com/learn/application-security/sql-injection-sqli/
https://www.vaadata.com/blog/sql-injections-principles-impacts-exploitations-security-best-practices/
https://portswigger.net

Proof of Concept

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



1.3. Sub-lab-3 Strings and errors part 3

Reference	Risk Rating
Sub-lab-3: Strings and errors part 3	**Low**
Tools Used	
Browser	
Vulnerability Description	

SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. SQL injections typically fall under three categories**: In-band SQLi** (Classic), **Inferential SQLi** (Blind) and **Out-of-band SQLi**. You can classify SQL injections types based on the methods they use to access backend data and their damage potential. In this case a **Union-based SQLi**— was used, this technique takes advantage of the UNION SQL operator, which fuses multiple select statements generated by the database to get a single HTTP response. This response may contain data that can be leveraged by the attacker. An attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

How It Was Discovered

Manual Analysis

Vulnerable URLs

https://labs.hacktify.in/HTML/sqli_lab/lab_3/lab_3.php?id=3%27%20UNION%20SELECT%20username,%20password%20FROM%20users%20--

Consequences of not Fixing the Issue

Not Fixing SQL Injection (SQLi) vulnerabilities can have severe impacts on your web application, database, and organization:

1. Compliance Issues
2. Business Disruption
3. Physical Security Risks
4. Regulatory compliance
5. System downtime
6. Competitive Disadvantage

It's essential patching SQLi vulnerabilities to prevent these severe impacts and protect your organization's sensitive data and systems.

Suggested Countermeasures

To prevent SQL injection (SQLi) attacks, the following countermeasures should be considered:

1) Monitoring and Logging

- Monitor database logs: Regularly review database logs to detect suspicious activity.
- Implement intrusion detection and prevention systems (IDPS): IDPS can help detect and prevent SQLi attacks.

2) Additional Measures

- Limit database privileges: Restrict database privileges to the minimum required for each user or application.
- Use encryption: Encrypt sensitive data to protect it from unauthorized access.
- Regularly perform security audits: Conduct regular security audits to identify and address vulnerabilities.

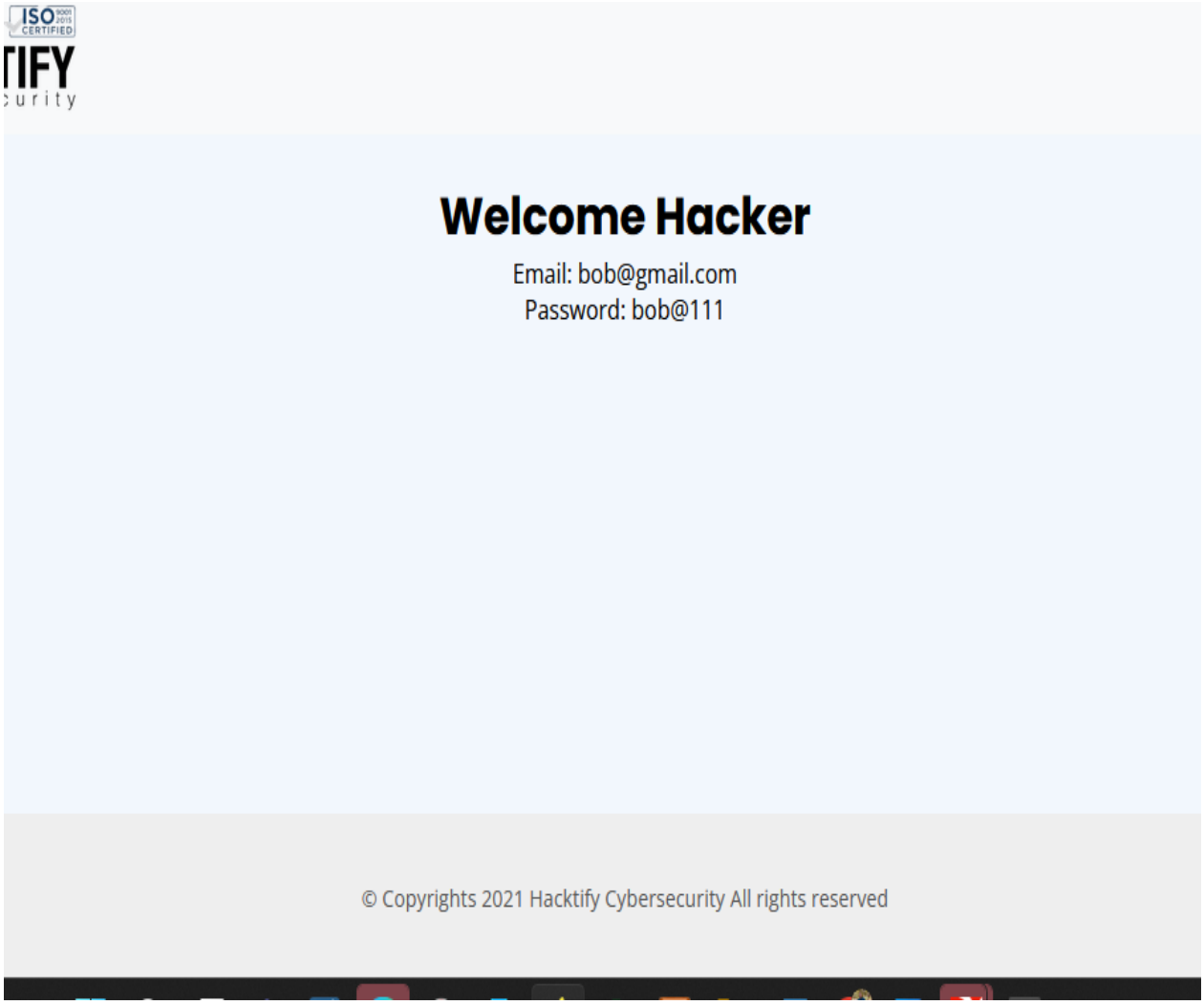
3) Database Configuration and Patching

- Keep databases and frameworks up-to-date: Regularly update databases and frameworks to ensure you have the latest security patches.
- Configure databases securely: Implement secure database configurations, such as limiting privileges and enabling logging.
- Use a web application firewall (WAF): A WAF can help detect and prevent SQLi attacks.

References
https://www.vaadata.com/blog/sql-injections-principles-impacts-exploitations-security-best-practices/ https://www.w3schools.com https://portswigger.net

Proof of Concept

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



1.4. Sub-lab-4 Let's trick 'em!

Reference	Risk Rating
Sub-lab-4: Let's trick 'em!	**Low**
Tools Used	
Sublime text editor "used to craft a payload"	
Vulnerability Description	
SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. SQL injections typically fall under three categories**: In-band SQLi** (Classic), **Inferential SQLi** (Blind) and **Out-of-band SQLi**. You can classify SQL injections types based on the methods they use to access backend data and their damage potential. In this case a **Union-based SQLi**— was used, this technique takes advantage of the UNION SQL operator, which fuses multiple select statements generated by the database	

to get a single HTTP response. This response may contain data that can be leveraged by the attacker. An attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

How It Was Discovered

Manual Analysis “By uploading the file with malicious content”

Vulnerable URLs

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

Consequences of not Fixing the Issue

Not patching SQL Injection (SQLi) vulnerabilities can have severe impacts on the web application, database, and organization:

1. Unauthorized data access
2. Data tampering
3. Malware injection
4. Unauthorized data access
5. Data exfiltration
6. Privilege escalation

It's essential to prioritize patching SQLi vulnerabilities to prevent these severe impacts and protect the organization's sensitive data and systems.

Suggested Countermeasures

To prevent SQL injection (SQLi) attacks, the following countermeasures should be considered:

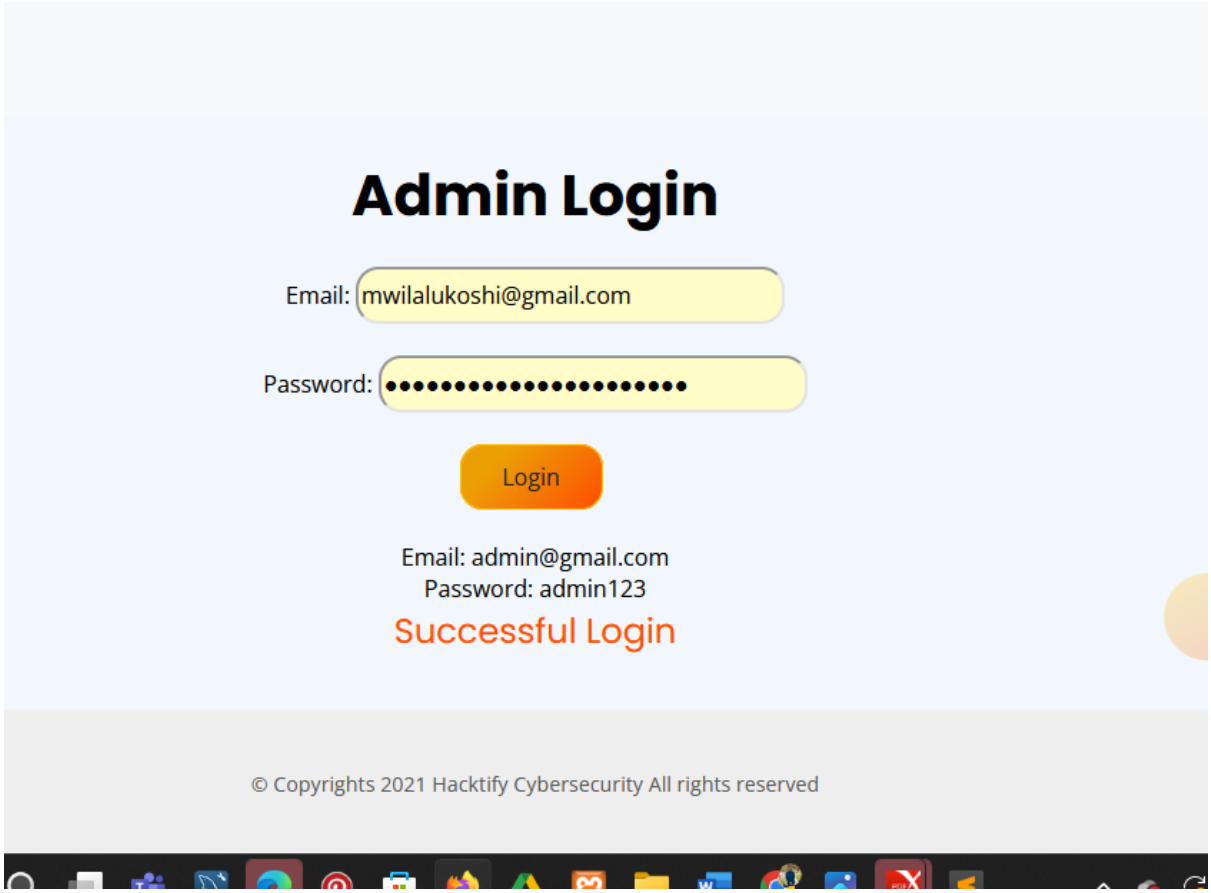
1. **Monitor database logs:** Regularly review database logs to detect suspicious activity.
2. **Limit database privileges:** Restrict database privileges to the minimum required for each user or application.
3. **Regularly perform security audits:** Conduct regular security audits to identify and address vulnerabilities.
4. **Keep databases and frameworks up-to-date**: Regularly update databases and frameworks to ensure you have the latest security patches.
5. **Configure databases securely:** Implement secure database configurations, such as limiting privileges and enabling logging.
6. **Use a web application firewall (WAF):** A WAF can help detect and prevent SQLi attacks.

References

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/SQL%20Injection
https://portswigger.net/web-security/file-upload
https://www.vaadata.com/blog/sql-injections-principles-impacts-exploitations-security-best-practices/
https://www.w3schools.com
https://portswigger.net

Proof of Concept

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



1.5. Sub-lab-5 Boolean and Blind!

Reference	**Risk Rating**
Sub-lab-5: Booleans and Blind!	**Low**
Tools Used	
Manual analysis “Browser”	
Vulnerability Description	
SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. There are quite a number of **SQLi** types and classifications**,** this is a **Blind SQLi**, **Boolean** one to be specific—in this type an attacker sends a SQL query to the database prompting the application to return a result. The result will vary depending on whether the query is true or false. Based on the result, the information within the HTTP response will modify or stay unchanged. The attacker can then work out if the message generated a true or false result.	

How It Was Discovered
Manual Analysis
Vulnerable URLs
https://labs.hacktify.in/HTML/sqli_lab/lab_5/lab_5.php?%20id=3%20AND%201=1%20--
Consequences of not Fixing the Issue
The impact of not Patching SQLi vulnerabilities can have a severe consequence like: 1. Business Disruption 2. Physical Security Risks 3. Regulatory compliance 4. System downtime 5. Competitive Disadvantage
Suggested Countermeasures
To prevent SQL injection (SQLi) attacks, the following countermeasures should be considered: 1. Monitor database logs: 2. Limit database privileges: 3. Regularly perform security audits: 4. Keep databases and frameworks up-to-date: 5. Configure databases securely: 6. Use a web application firewall (WAF):
References
https://www.imperva.com/learn/application-security/sql-injection-sqli/ https://portswigger.net/web-security/file-upload https://www.vaadata.com/blog/sql-injections-principles-impacts-exploitations-security-best-practices/

Proof of Concept

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

1.6. Sub-lab-6 Error Based: Tricked!

Reference	Risk Rating
Sub-lab-6: Error Based: Tricked!	**High**
Tools Used	
Browser	
Vulnerability Description	
SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior. In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk	
How It Was Discovered	
Manual Analysis “The vulnerability was discovered after trying the entry points with scripts and payloads, then the script was encoded into the URL format and input it into the form, and it got executed.”	

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 have a bad impact on the application, the following are the issues to face:

1. Compliance Issues
2. Business Disruption
3. Physical Security Risks
4. Regulatory compliance
5. Competitive Disadvantage

It's essential patching SQLi vulnerabilities to prevent these severe impacts and protect sensitive data.

Suggested Countermeasures

To prevent SQL injection (SQLi) attacks, the following countermeasures should be considered:

4. **Input Validation and Sanitization**
   - Validate user input: Ensure that user input conforms to expected formats and patterns.
   - Sanitize user input: Remove or escape any special characters that could be used to inject malicious SQL code.

5. **Secure Coding Practices**
   - Use prepared statements: Prepared statements separate code from user input, preventing injection attacks.
   - Use parameterized queries: Parameterized queries pass user input as parameters, rather than concatenating it into the query.
   - Avoid dynamic SQL: Dynamic SQL can be vulnerable to injection attacks; instead, use static SQL or prepared statements.

6. **Database Configuration and Patching**
   - Keep databases and frameworks up-to-date: Regularly update databases and frameworks to ensure you have the latest security patches.
   - Configure databases securely: Implement secure database configurations, such as limiting privileges and enabling logging.
   - Use a web application firewall (WAF): A WAF can help detect and prevent SQLi attacks.

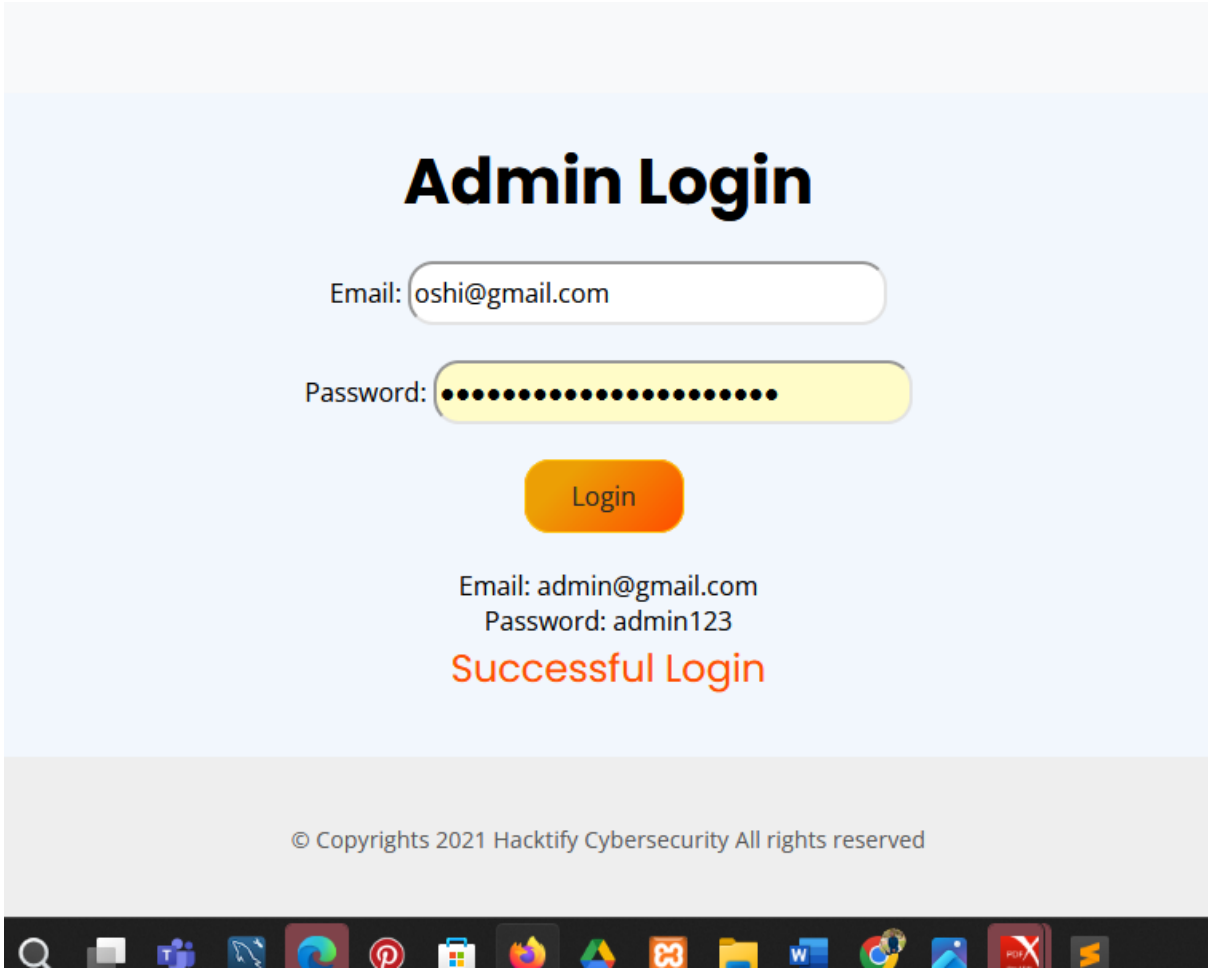
References

] https://www.imperva.com/learn/application-security/sql-injection-sqli/
https://portswigger.net/web-security/file-upload

https://www.vaadata.com/blog/sql-injections-principles-impacts-exploitations-security-best-practices/

Proof of Concept

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



1.7. Sub-lab-7 Error and Post!

Reference	Risk Rating
Sub-lab-7 Error and Post!	**Low**
Tools Used	

Manual analysis “Browser tools”

Vulnerability Description

SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk

How It Was Discovered

Manual Analysis “By trying all the entry points with scripts”

Vulnerable URLs

https://labs.hacktify.in/HTML/sqli_lab/lab_7/lab_7.php

Consequences of not Fixing the Issue

The impact of not Patching SQLi vulnerabilities can have a severe consequence like:

1. Business Disruption
2. Physical Security Risks
3. Regulatory compliance
4. System downtime
5. Competitive Disadvantage

Suggested Countermeasures

Failure to Fix SQL Injection (SQLi) vulnerabilities can have a very bad impact on the application, namely:

1. Compliance Issues
2. Business Disruption
3. Physical Security Risks
4. Regulatory compliance
5. Competitive Disadvantage

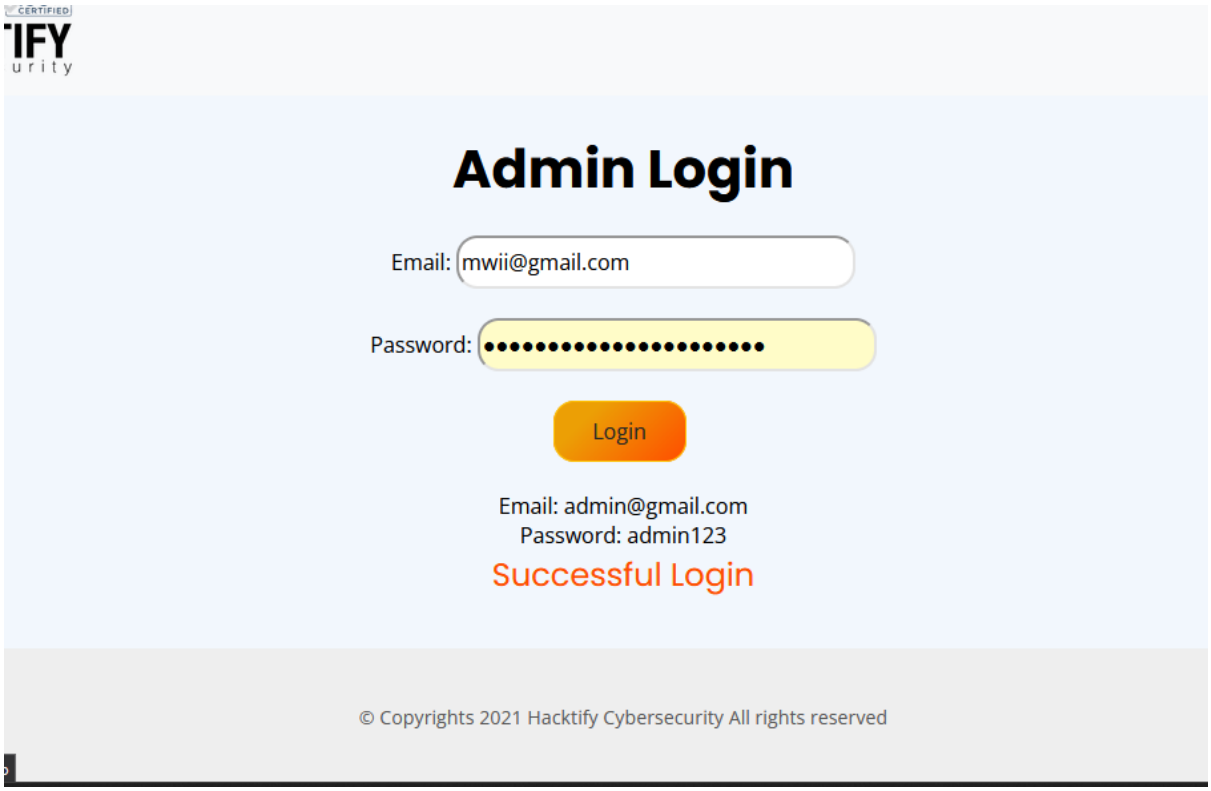
It's essential patching SQLi vulnerabilities to prevent these severe impacts and protect sensitive data.

References

https://www.imperva.com/learn/application-security/sql-injection-sqli/
https://portswigger.net/web-security/file-upload
https://www.vaadata.com/blog/sql-injections-principles-impacts-exploitations-security-best-practices/

Proof of Concept

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



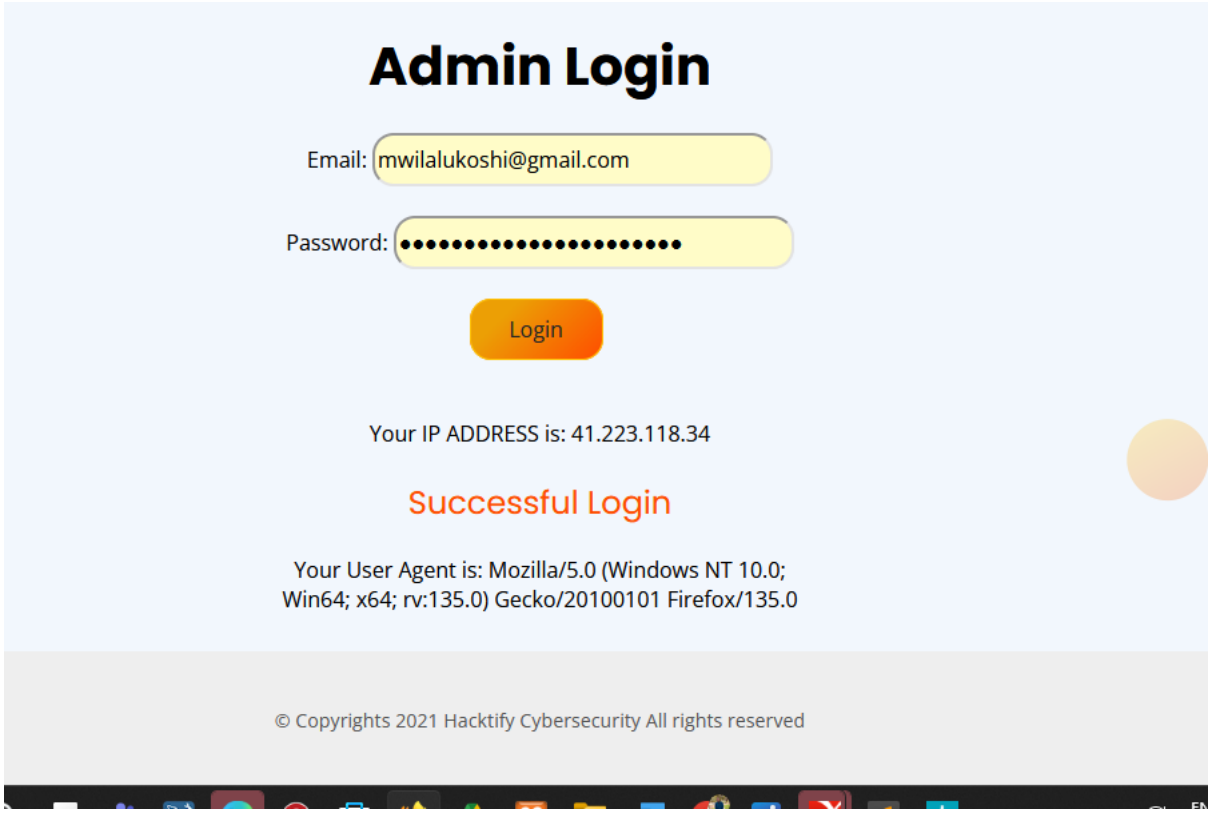
1.8. Sub-lab-2: User agents lead us!

Reference	Risk Rating
Sub-lab-2: User agents lead us!	**Low**
Tools Used	
Vulnerability Description	
SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.	

In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk
How It Was Discovered
Automated Tools “Browser tool, I inspected the source code and balanced that script”
Vulnerable URLs
https://labs.hacktify.in/HTML/sqli_lab/lab_8/lab_8.php
Consequences of not Fixing the Issue
Failure to Fix SQL Injection (SQLi) vulnerabilities can have a bad impact on the application, the following are the issues to face: 1. Compliance Issues 2. Business Disruption 3. Physical Security Risks 4. Regulatory compliance 5. Competitive Disadvantage It's essential patching SQLi vulnerabilities to prevent these severe impacts and protect sensitive data.
Suggested Countermeasures
1. Patch Management 2. Secure Coding Practices 3. Code Refactoring 4. Security Information and Event Management (SIEM) 5. Penetration Testing
References
https://www.imperva.com/learn/application-security/sql-injection-sqli/ https://portswigger.net/web-security/file-upload https://www.vaadata.com/blog/sql-injections-principles-impacts-exploitations-security-best-practices/ https://owasp.org https://portswigger.net

Proof of Concept

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



1.9. Sub-lab-9: Referrer lead us!

Reference	**Risk Rating**
Sub-lab-3: Referrer lead us!	**Low**
Tools Used	
Vulnerability Description	
SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior. In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk	
How It Was Discovered	
Automated Tools “I tried different inputs (scripts), the code passed unvalidated input from the request	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	

Not Fixing SQL Injection (SQLi) vulnerabilities can have severe impacts on your web application, database, and organization:

1. Denial of Service (DoD)
2. Data Breaches
3. Increased Risk of Future Attacks
4. Regulatory compliance
5. System downtime
6. Competitive Disadvantage

It's essential patching SQLi vulnerabilities to prevent these severe impacts and protect your organization's sensitive data and systems.

Suggested Countermeasures

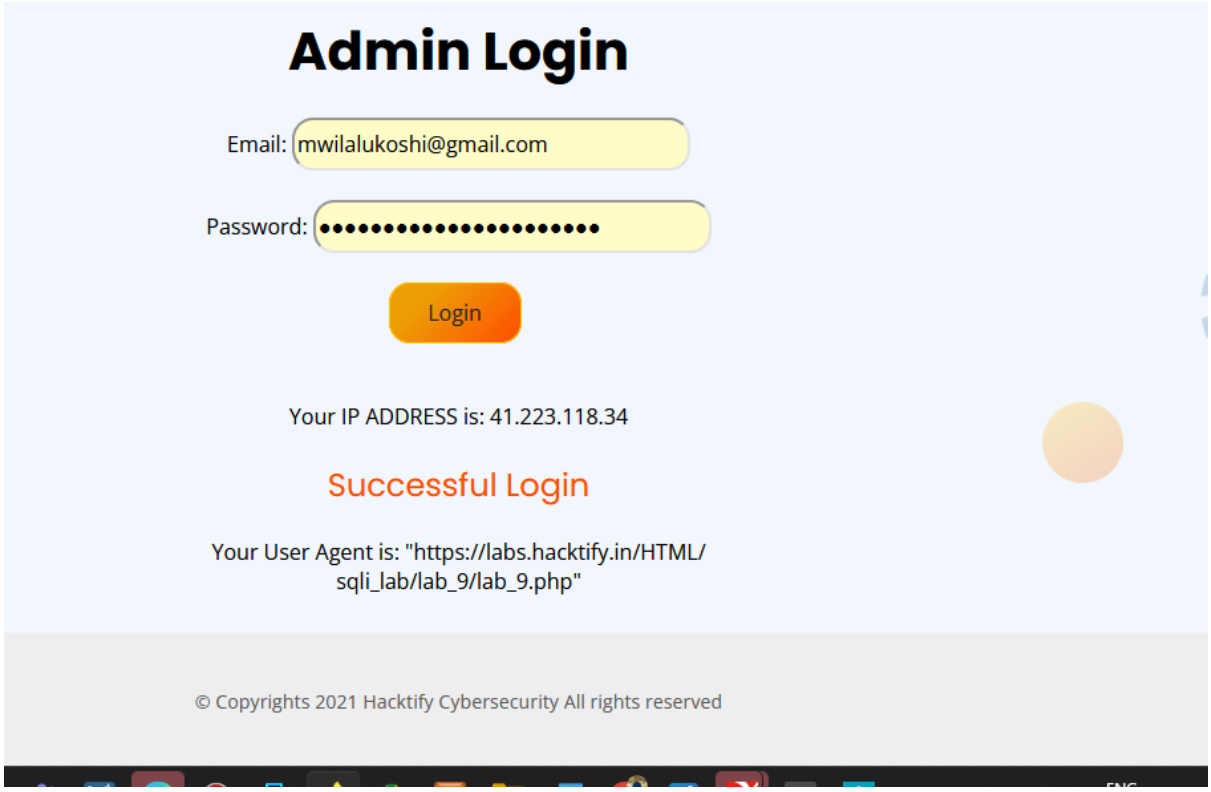
1. Patch Management
2. Secure Coding Practices
3. Code Refactoring
4. Security Information and Event Management (SIEM)
5. Penetration Testing

References

https://www.imperva.com/learn/application-security/sql-injection-sqli/
https://portswigger.net/web-security/file-upload
https://www.vaadata.com/blog/sql-injections-principles-impacts-exploitations-security-best-practices/
https://portswigger.net

Proof of Concept

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



1.10. Sub-lab-10: Oh cookies!

Reference	**Risk Rating**
Sub-lab-10: Oh cookies!	**Medium**
Tools Used	
Browser	
Vulnerability Description	
SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior. In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk	
How It Was Discovered	
Automated Tools “By trying different entry points and different scripts payload”	
Vulnerable URLs	
https://labs.hacktify.in/HTML/sqli_lab/lab_10/lab_10.php	
Consequences of not Fixing the Issue	

To prevent SQL Injection (SQLi) vulnerabilities, consider the following countermeasures:

1. Business Disruption
2. Physical Security Risks
3. Regulatory compliance
4. Competitive Disadvantage
5. Secure Coding Practices
6. Regular Security Audits and Testing

It's essential patching SQLi vulnerabilities to prevent these severe impacts and protect sensitive data.

Suggested Countermeasures

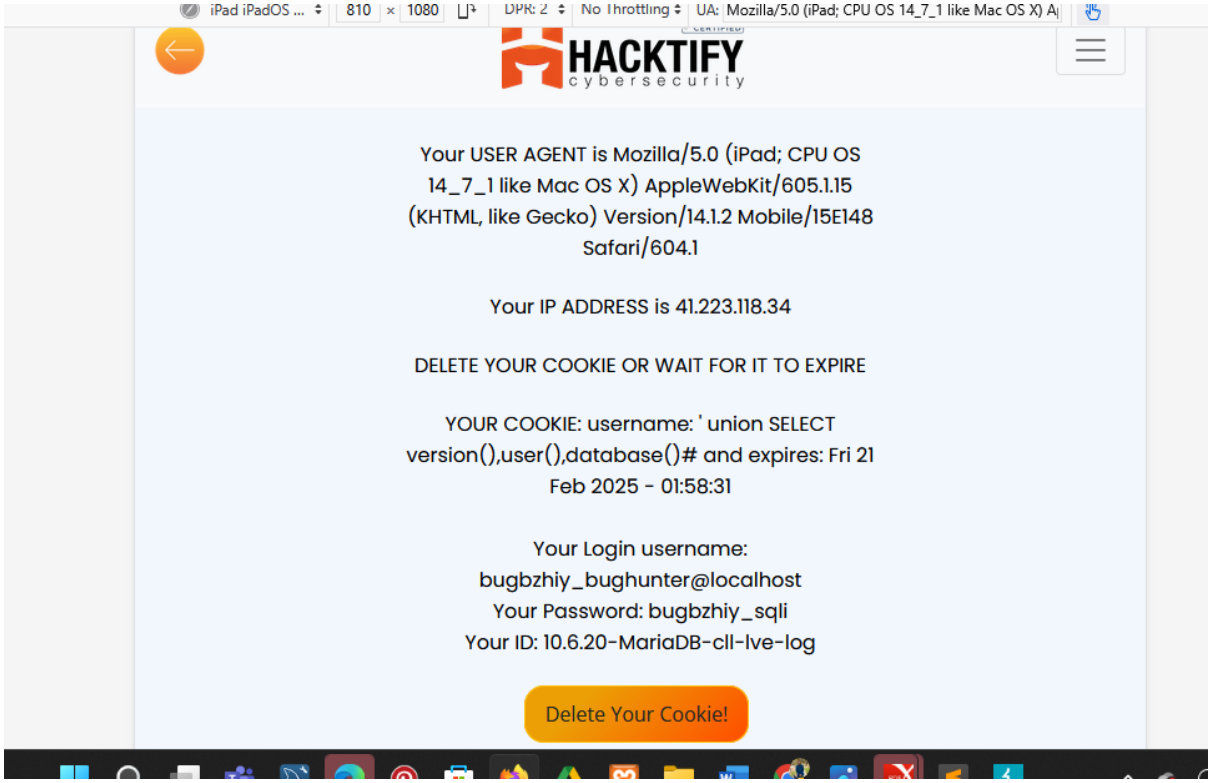
6. Patch Management
7. Secure Coding Practices
8. Code Refactoring
9. Security Information and Event Management (SIEM)
10. Penetration Testing

References

https://www.invcti.com
https://portswigger.net
https://www.acunetix.com

Proof of Concept

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



1.11. Sub-lab-11: WAF are Injectables payload!

Reference	**Risk Rating**
Sub-lab-5: Developers hate Script!	**High**
Tools Used	
Tools that you have used to find the vulnerability.	
Vulnerability Description	
SQL injection **(SQLi)** is a type of web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior. In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php?email=%22%2F%3E%3Cimg+src%3Dq+onerror%3Dprompt%28document.cookie%29%3E	
Consequences of not Fixing the Issue	

1.
Suggested Countermeasures
Failure to Fix SQL Injection (SQLi) vulnerabilities can have a bad impact on the application, the following are the issues to face: 1. Business Disruption 2. Physical Security Risks 3. Regulatory compliance 4. Competitive Disadvantage 5. Secure Coding Practices 6. Regular Security Audits and Testing It's essential patching SQLi vulnerabilities to prevent these severe impacts and protect sensitive data.
References
https://owasp.org https://www.invcti.com https://portswigger.net https://www.acunetix.com

Proof of Concept

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

2. Lab 2 (Indirect Document Object References (IDOR)

2.1. Sub-lab-Give me my amount!!

Reference	**Risk Rating**
Sub-lab-1: Give me my amount!!	**low**
Tools Used	
Browser	
Vulnerability Description	

Insecure direct object reference (IDOR), is the type of web vulnerability that allows 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 and **ID**. And if these IDs are easy enough to guess or an object can be used by an attacker to bypass access check somehow, then an (**IDOR**) attack can be considered possible at this point. Referring to this lab, I was able to exploit the URL parameters and change the user account to any user, as long as that ID number is in the database. After trying a couple of **ID numbers**, the database was getting exploited and showed user accounts of those **ID numbers**, I was able to see the user accounts information, the latest 3 transactions and a credit number successfully.

How It Was Discovered

"The vulnerability was discovered after trying the URL parameter point"

Vulnerable URLs

https://labs.hacktify.in/HTML/idor_lab/lab_1/profile.php

Consequences of not Fixing the Issue

1. Privacy violations
2. Data manipulation
3. Data breaches
4. Reputation damage
5. Account Hijacking

Suggested Countermeasures

To prevent Insecure Direct Object Reference **(IDOR)** vulnerabilities the following measurements should be considered:

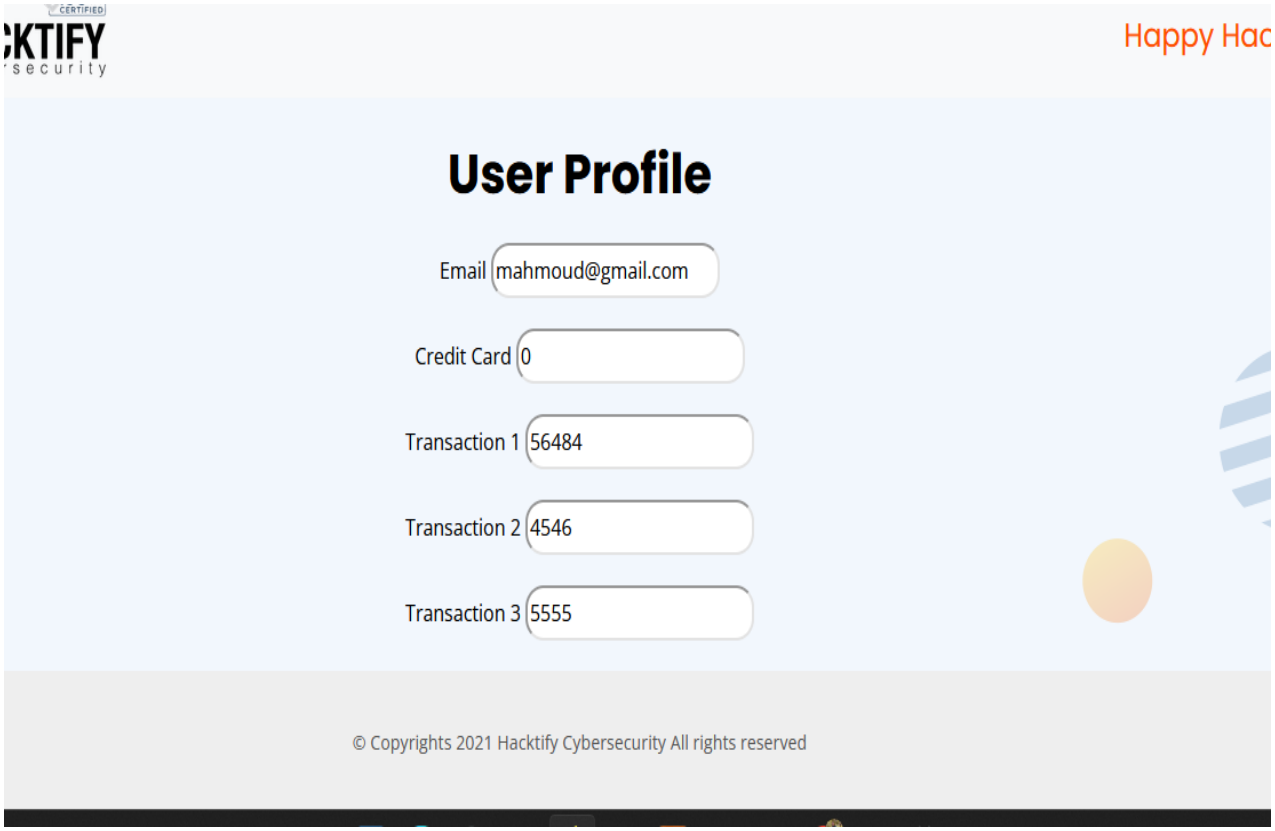
1. Provision of Regular security testing
2. Secure session management
3. Conduction of Penetration Testing
4. Utilizing of Security Information and Event Management (SIEM)
5. Providing of Security Trainings

References

https://avatao.com/blog-best-practices-to-prevent-idor-vulnerabilities/
https://www.imperva.com
https://avatao.com/blog-best-practices-to-prevent-idor-vulnerabilities/
https://portswigger.net

Proof of Concept

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



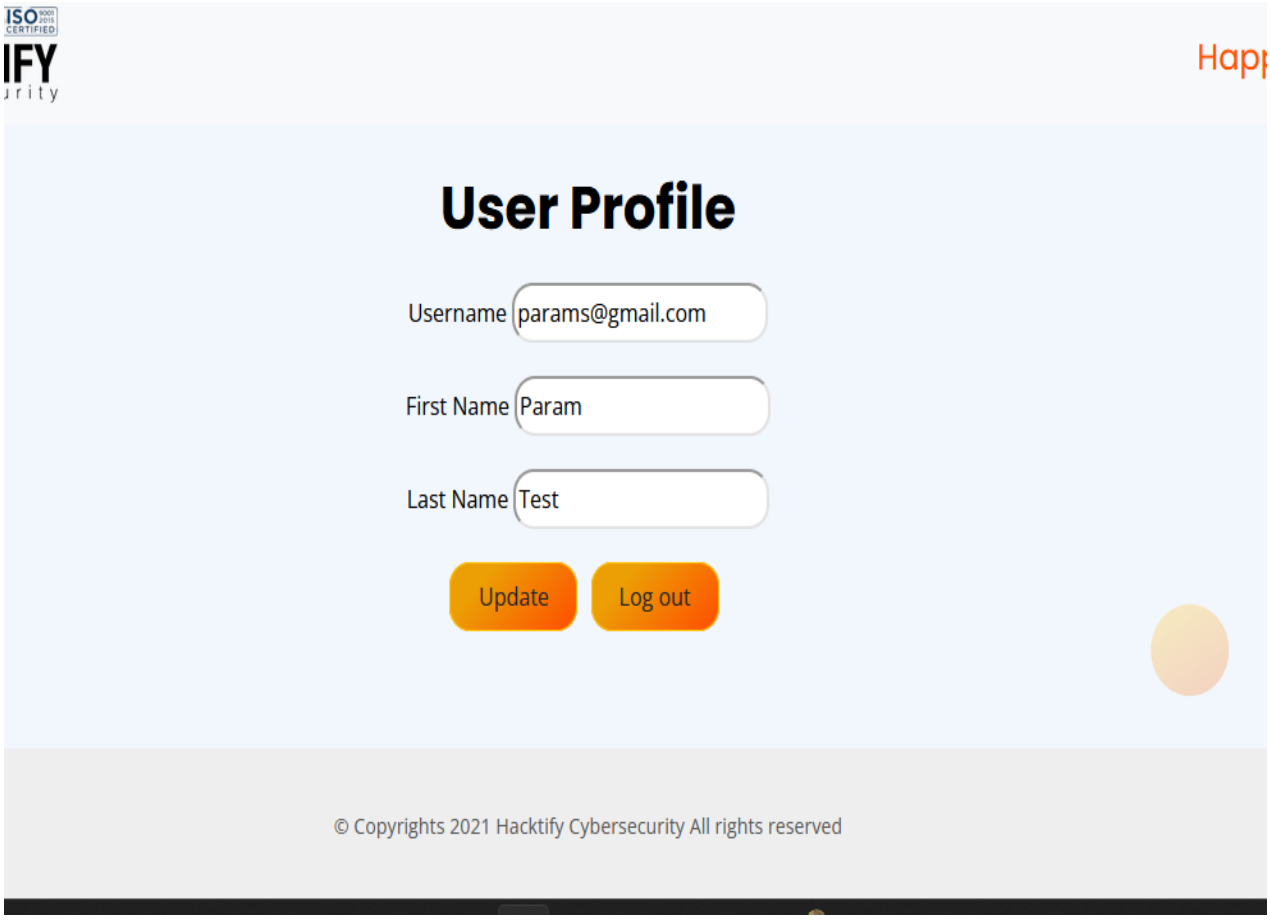
2.2. Sub-lab-2 Stop polluting my params!

Reference	**Risk Rating**
{Sub-lab-2: Stop polluting my params!	**medium**
Tools Used	
Browser	
Vulnerability Description	
Insecure direct object reference (IDOR), is the type of web vulnerability that allows 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 and **ID**. And if these IDs are easy enough to guess or an object can be used by an attacker to bypass access check somehow, then an (**IDOR**) attack can be considered possible at this point. Referring to this lab, I was able to exploit the URL parameters and change the user profile to any user, as long as that ID number is in the database. After trying a couple of **ID numbers**, the database was getting exploited and showed user profiles of those **ID numbers**, I was able to see the user profiles and information of many **ID numbers** successfully.	

How It Was Discovered
Automated Tools
Vulnerable URLs
https://labs.hacktify.in/HTML/idor_lab/lab_2/profile.php
Consequences of not Fixing the Issue
The consequences of not fixing **IDOR** vulnerabilities are that, they can lead to: 1. Data breaches 2. Privacy violations 3. Data manipulation 4. Account takeover 5. Reputation damage 6. Regulatory non-compliance
Suggested Countermeasures
To prevent Insecure Direct Object Reference **(IDOR)** vulnerabilities the following measurements should be considered: 1. Conduction of Penetration Testing 2. Utilizing of Security Information and Event Management (SIEM) 3. Providing of Security Trainings 4. Secure session management 5. Provision of Regular security testing
References
https://laburity.com/idor-case-study-manipulating-billing-information/ https://www.nodejs-security.com https://avatao.com/blog-best-practices-to-prevent-idor-vulnerabilities/ https://www.portswigger.net

Proof of Concept

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



2.3. Sub-lab-3 Someone Changed my password!

Reference	**Risk Rating**
Sub-lab-3: Someone changed my password!	**Medium**
Tools Used	
Browser	
Vulnerability Description	
Insecure direct object reference (IDOR), is the type of web vulnerability that allows 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 and ID. And if these IDs are easy enough to guess or an object can be used by an attacker to bypass access check somehow, then an IDOR attack can be considered possible at this point. Regarding this lab, I was able to create an account. After I registered my account, I logged in back with the credentials I had registered with. I then took advantage of the **URL parameters** and changed	

the user account to **id=abc**, I was able to change the password for account abc, and I was able to login with the changed credentials successfully.

How It Was Discovered

Manual Analysis “by changing the id account in to abc”

Vulnerable URLs

https://labs.hacktify.in/HTML/idor_lab/lab_3/profile.php

Consequences of not Fixing the Issue

The consequences of not fixing **IDOR** vulnerabilities are that, they can lead to:

1. Data breaches
2. Privacy violations
3. Data manipulation
4. Account takeover
5. Reputation damage
6. Regulatory non-compliance

Suggested Countermeasures

To prevent Insecure Direct Object Reference **(IDOR)** vulnerabilities the following measurements should be considered:

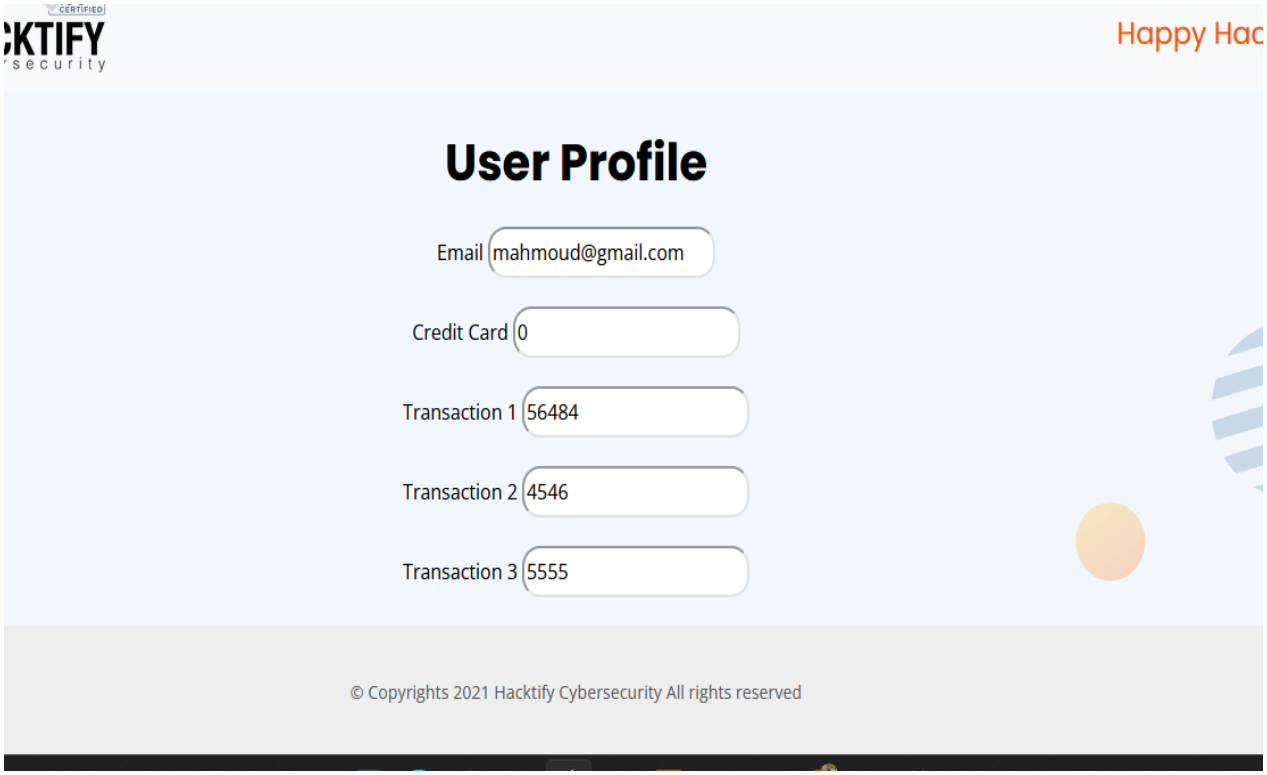
1. **Proper input validation and Sanitization:** Always validate user input to ensure it is in the expected format and does not contain malicious modifications.
2. **Server-side authorization:** Implement robust access controls on the server-side to verify user permissions before granting access to any resource.
3. **Secure session management:** Using of secure session mechanisms to prevent unauthorized access to sensitive data.
4. **Regular security testing:** Conduct thorough application security testing to identify and address potential IDOR vulnerabilities.

References

https://avatao.com/blog-best-practices-to-prevent-idor-vulnerabilities/
https://www.imperva.com
https://hadrian.io/blog/insecure-direct-object-reference-idor-a-deep-dive

Proof of Concept

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



2.4. Sub-lab-4 Change your methods!

Reference	Risk Rating
Sub-lab-2: Change your methods!	**Medium**
Tools Used	
Browser “Tried all the Entry points”	
Vulnerability Description	
Insecure direct object reference (IDOR), is the type of web vulnerability that allows 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 and ID. And if these IDs are easy enough to guess or an object can be used by an attacker to bypass access check somehow, then an **IDOR** attack can is possible at this point. Referring to this lab, an attacker by ethical means can only get the document numbered 101 which is legally his. But what if the web application does not validate the number and an attacker puts the number of its victim. This can cause the attacker get hold of the sensitive document which he should not have access to.	
How It Was Discovered	

This was discovered by testing the fields which stores values in a server, in this case a malicious script was supplied in a f_name and l_name fields.

Vulnerable URLs

https://labs.hacktify.in/HTML/idor_lab/lab_4/profile.php

Consequences of not Fixing the Issue

The consequences of not fixing **IDOR** vulnerabilities are that, they can lead to:

1. Data breaches
2. Privacy violations
3. Data manipulation
4. Account takeover
5. Reputation damage
6. Regulatory non-compliance

Suggested Countermeasures

To prevent Insecure Direct Object Reference **(IDOR)** vulnerabilities, consider the following countermeasures:

1. Access Control
2. Input Validation and Sanitization
3. Secure Coding Practices
4. Error Handling and Logging
5. Regular Security Audits and Testing

References

https://www.nodejs-security.com
https://escape.tech
https://avatao.com/blog-best-practices-to-prevent-idor-vulnerabilities/
https://portswigger.net

Proof of Concept

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