**Penetration Testing Report**

**Full Name: Karthikaa. R & Dharanidharan. V**

**Program: HCS - Penetration Testing Internship Week-2**

**Date:02/03/2024**

**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** | **Cross-Site Scripting, Insecure Direct Object References** |

**3. Summary**

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

**Total number of Sub-labs: 15 Sub-labs**

|  |  |  |
| --- | --- | --- |
| **High** | **Medium** | **Low** |
| **4** | **5** | **6** |

**High - Number of Sub-labs with hard difficulty level**

**Medium - Number of Sub-labs with Medium difficulty level**

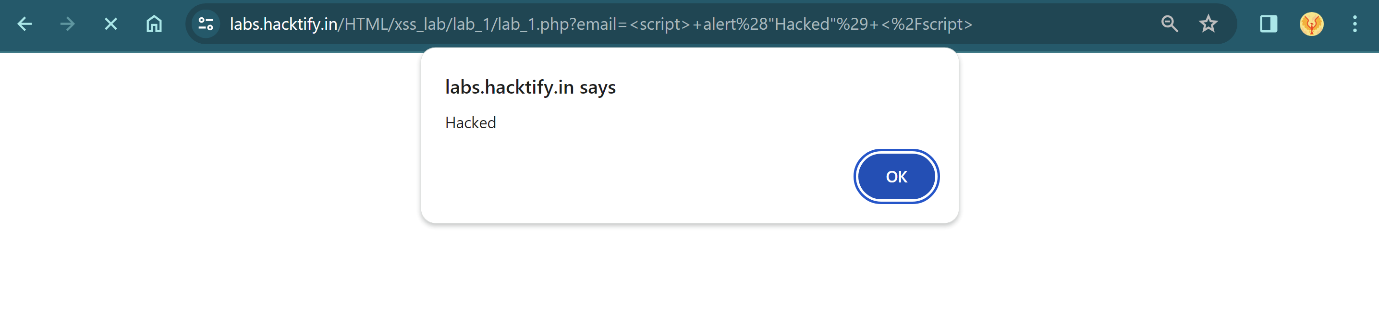
**Low - Number of Sub-labs with Easy difficulty level**

# 1. Cross-Site Scripting

# 1.1. Let’s Do IT!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Let’s Do IT! | **Low** |
| **Tools Used** | |
| Exploiting a straightforward XSS method via the email input functionality (No tool used). | |
| **Vulnerability Description** | |
| The XSS (Cross-Site Scripting) vulnerability in the email subscription input box of a website occurs when the application fails to properly sanitize user-supplied input. Specifically, when an attacker inserts a malicious script into the input box, the website's server reflects this input back to the user's browser without proper validation or encoding. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_1/index.php | |
| **Consequences of not Fixing the Issue** | |
| Failure to patch this vulnerability can lead the attackers to steal sensitive user information (such as cookies or credentials), execution of arbitrary scripts in the context of other users' browsers, leading to unauthorized actions, compromise of user sessions, allowing attackers to impersonate legitimate users and perform malicious activities. | |
| **Suggested Countermeasures** | |
| Implement strict input validation and output encoding to sanitize user-supplied data and prevent script injection. | |
| **References** | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> | |

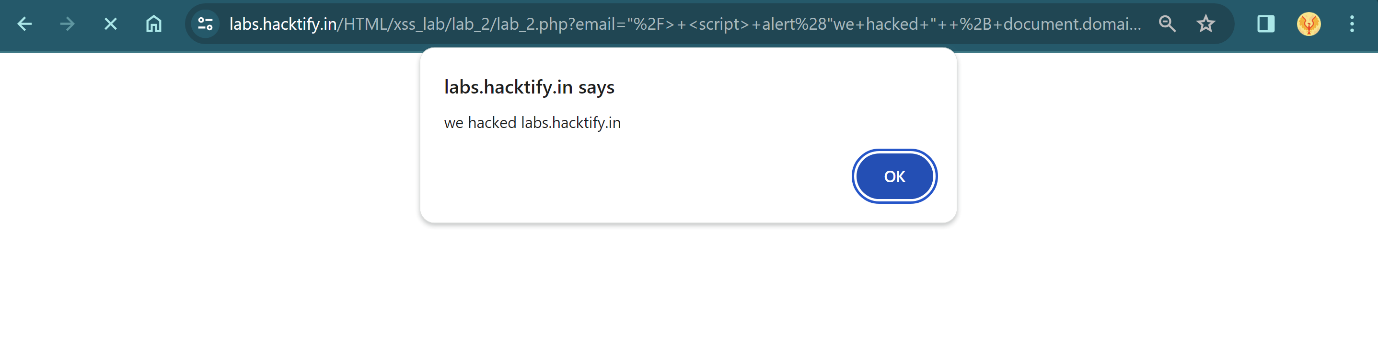
# Proof of Concept



# 1.2. Balancing Is Important in Life!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Balancing Is Important in Life! | **Low** |
| **Tools Used** | |
| Manual analysis without specific tools. | |
| **Vulnerability Description** | |
| The XSS (Cross-Site Scripting) vulnerability in the email subscription input box of a website occurs when the application fails to properly sanitize user-supplied input. Specifically, when an attacker inserts a malicious script into the input box, the website's server reflects this input back to the user's browser without proper validation or encoding. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_2/index.php | |
| **Consequences of not Fixing the Issue** | |
| Failure to patch this vulnerability can lead the attackers to steal sensitive user information (such as cookies or credentials), execution of arbitrary scripts in the context of other users' browsers, leading to unauthorized actions, compromise of user sessions, allowing attackers to impersonate legitimate users and perform malicious activities. | |
| **Suggested Countermeasures** | |
| Implement strict input validation and output encoding to sanitize user-supplied data and prevent script injection. | |
| **References** | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/> | |

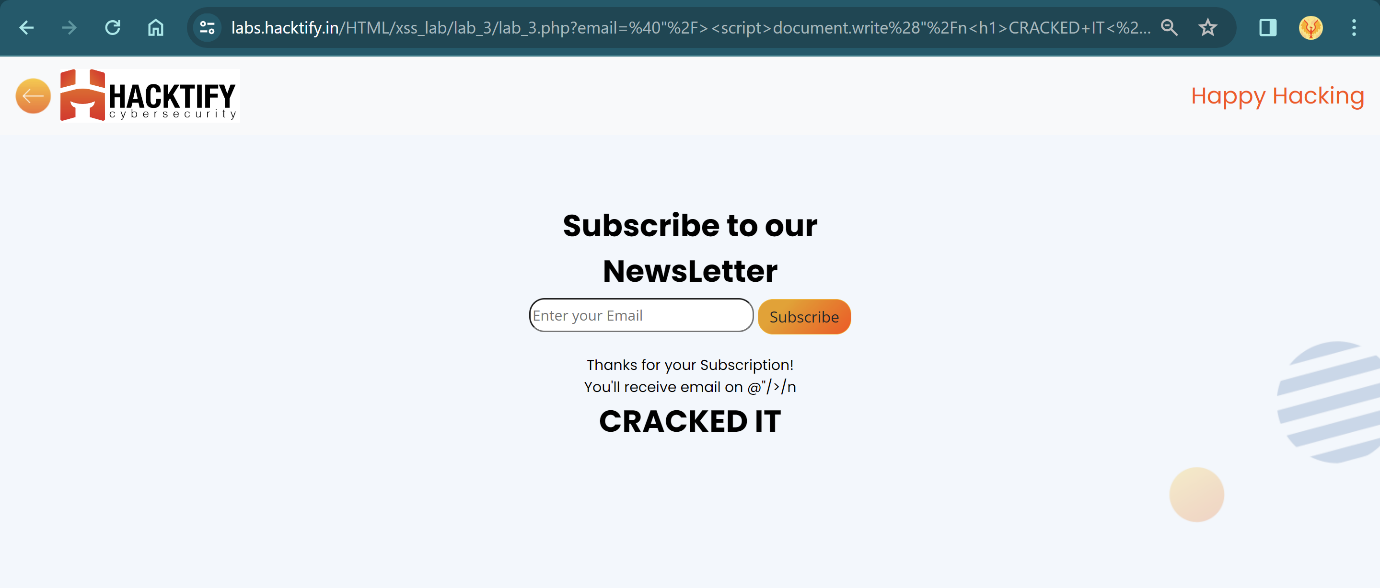
# Proof of Concept



# 1.3. XSS is Everywhere!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| XSS is Everywhere! | **Low** |
| **Tools Used** | |
| Manual analysis without specific tools. | |
| **Vulnerability Description** | |
| The vulnerability in the email subscription input box of a website persists despite the addition of email validation. In this scenario, the website employs email validation to ensure that the input conforms to an email address format. However, the validation mechanism fails to adequately sanitize the input, allowing for XSS (Cross-Site Scripting) attacks. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_3/index.php | |
| **Consequences of not Fixing the Issue** | |
| Failure to patch this vulnerability can lead the attackers to steal sensitive user information (such as cookies or credentials), execution of arbitrary scripts in the context of other users' browsers, leading to unauthorized actions, compromise of user sessions, allowing attackers to impersonate legitimate users and perform malicious activities. | |
| **Suggested Countermeasures** | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| **References** | |
| <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> | |

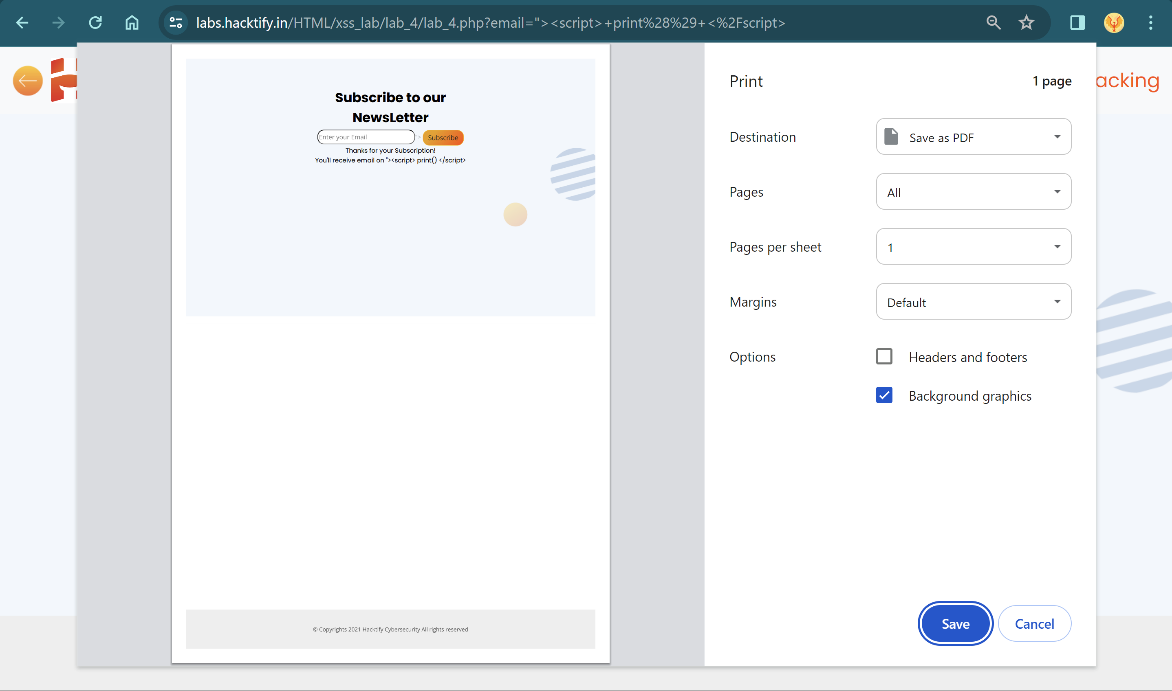
# Proof of Concept



# 1.4. Alternatives are Must!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Alternatives are Must! | **Medium** |
| **Tools Used** | |
| No tools are used. | |
| **Vulnerability Description** | |
| Despite the addition of alert obfuscation techniques, the vulnerability in the email subscription input box of a website persists. In this scenario, the website attempts to mitigate XSS (Cross-Site Scripting) attacks by employing alert obfuscation to obscure the malicious payload. However, the obfuscation mechanism fails to adequately sanitize the input, allowing for the execution of alternative payloads. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_4/index.php | |
| **Consequences of not Fixing the Issue** | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions. | |
| **Suggested Countermeasures** | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| **References** | |
| <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

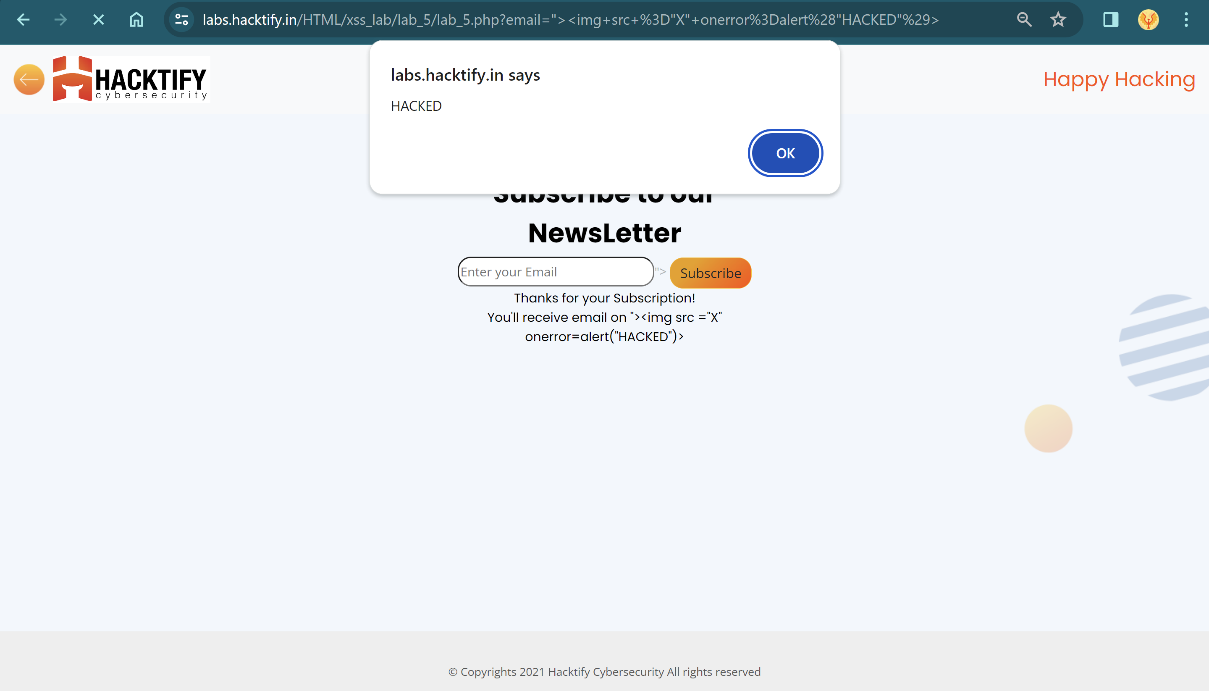
# Proof of Concept



# 1.5. Developer Hates Scripts!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Developer Hates Scripts! | **High** |
| **Tools Used** | |
| No tools are used. | |
| **Vulnerability Description** | |
| Despite the addition of script obfuscation techniques, the vulnerability in the email subscription input box of a website persists. In this scenario, the website attempts to mitigate XSS (Cross-Site Scripting) attacks by employing script obfuscation to obscure the malicious payload. However, the obfuscation mechanism fails to adequately sanitize the input, allowing for the execution of alternative payloads | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_5/index.php | |
| **Consequences of not Fixing the Issue** | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions. | |
| **Suggested Countermeasures** | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| **References** | |
| <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> | |

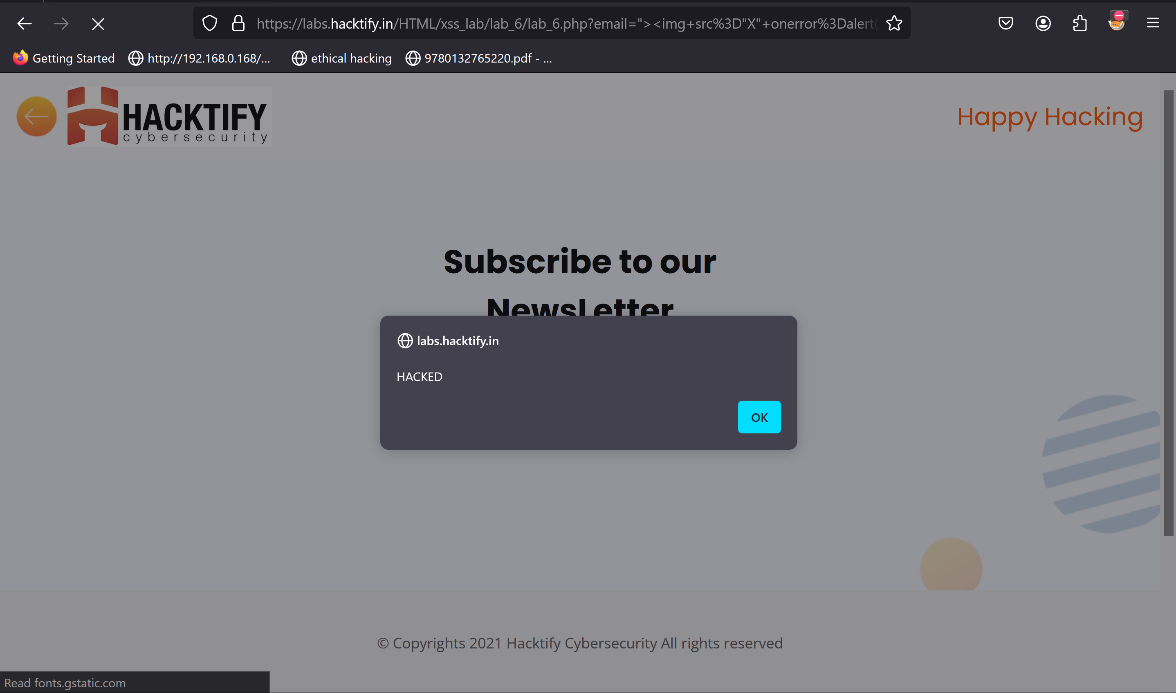
# Proof of Concept



# 1.6. Change the Variation!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Change the Variation! | **High** |
| **Tools Used** | |
| Manual analysis without any specific tools. | |
| **Vulnerability Description** | |
| Despite the addition of script sanitization techniques, the vulnerability in the email subscription input box of a website persists. In this scenario, the website attempts to mitigate XSS (Cross-Site Scripting) attacks by employing script sanitization to remove or neutralize potentially malicious payloads. However, the sanitization mechanism fails to adequately handle all input variations, allowing for the execution of alternative payloads. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_6/index.php | |
| **Consequences of not Fixing the Issue** | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions. | |
| **Suggested Countermeasures** | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| **References** | |
| <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

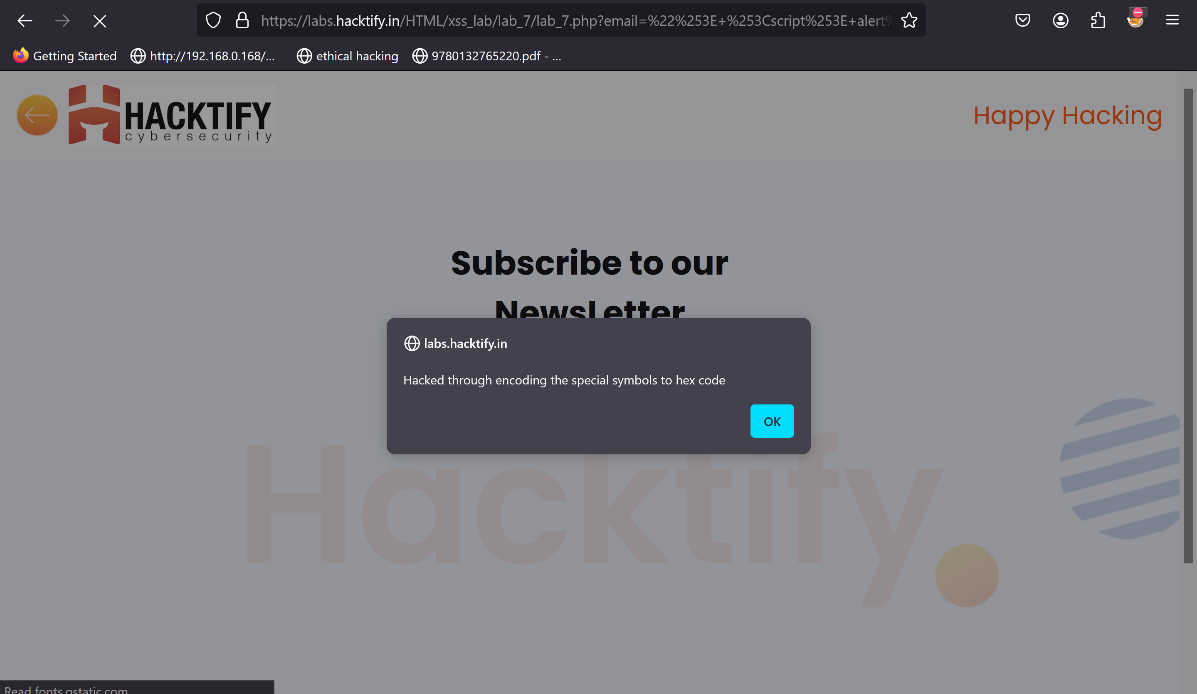
# Proof of Concept



# 1.7. Encoding Is the Key?

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Encoding Is the Key? | **Medium** |
| **Tools Used** | |
| No tools are used. | |
| **Vulnerability Description** | |
| Despite the addition of encoding for symbols in tags, the vulnerability in the email subscription input box of a website persists. In this scenario, the website attempts to mitigate XSS (Cross-Site Scripting) attacks by encoding special characters within HTML tags. However, the encoding mechanism fails to adequately handle all input variations, allowing for the execution of alternative payloads. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_7/index.php | |
| **Consequences of not Fixing the Issue** | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions | |
| **Suggested Countermeasures** | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| **References** | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

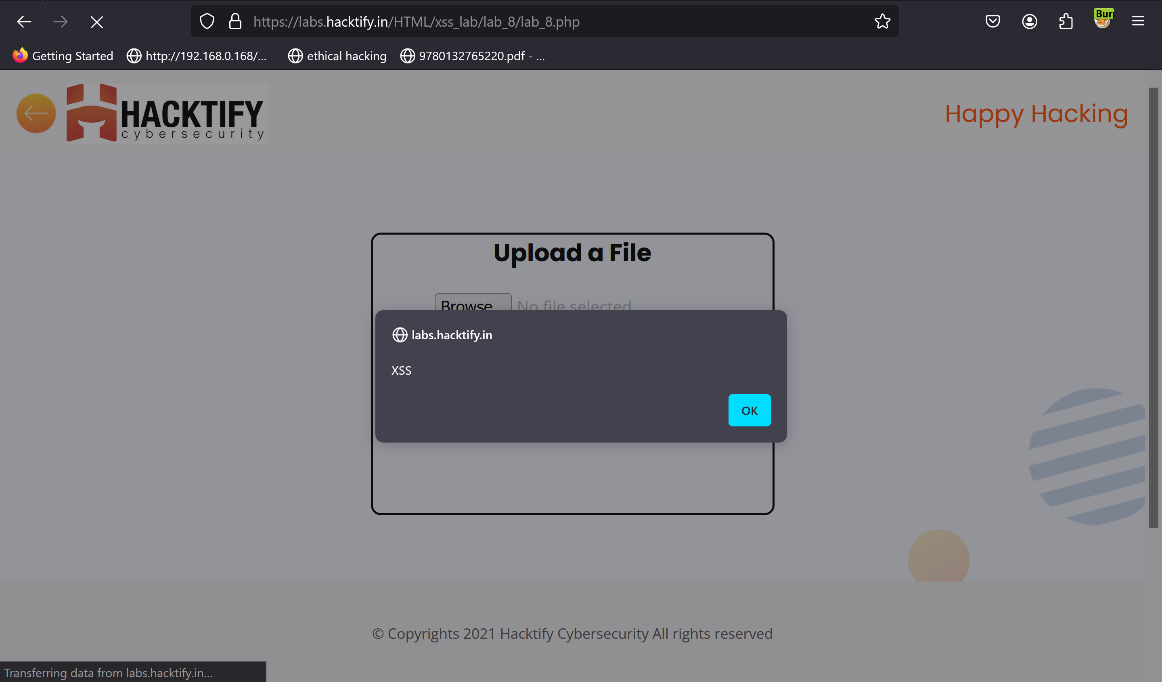
# Proof of Concept



# 1.8. XSS with File Upload (File Name)

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| XSS with File Upload (File Name) | **Low** |
| **Tools Used** | |
| Burp Suite for intercepting and modifying requests. | |
| **Vulnerability Description** | |
| Despite the attempt to mitigate XSS (Cross-Site Scripting) attacks through file upload, the vulnerability in the email subscription input box of a website persists. In this scenario, the website allows users to upload files, but the filename input is not properly sanitized or validated. Attackers exploit this vulnerability by intercepting the file upload request using tools like Burp Suite and modifying the filename to include a malicious payload | |
| **How It Was Discovered** | |
| Discovered through manual analysis by intercepting and modifying requests using Burp Suite. | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_8/index.php | |
| **Consequences of not Fixing the Issue** | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions | |
| **Suggested Countermeasures** | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| **References** | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

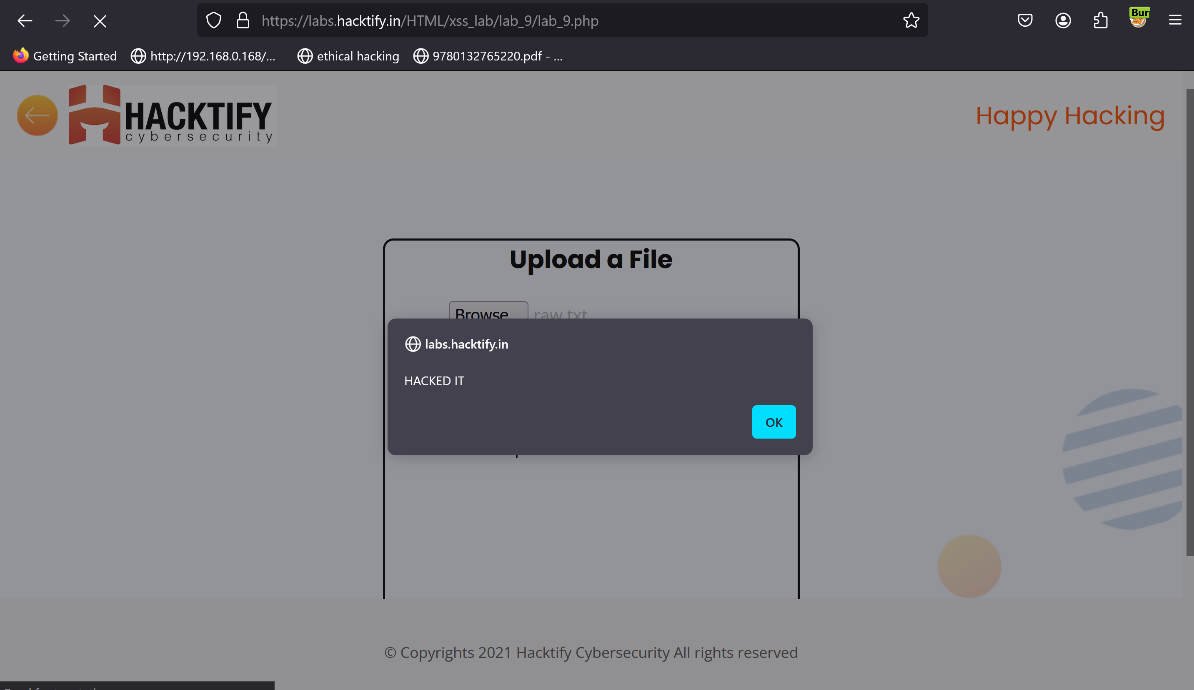
# Proof of Concept



# 1.9. XSS with File Upload (File Content)

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| XSS with File with File Upload (File Content) | **Medium** |
| **Tools Used** | |
| No specialized tools were used; manual testing was conducted. | |
| **Vulnerability Description** | |
| Despite the attempt to mitigate XSS (Cross-Site Scripting) attacks through file upload, the vulnerability in the email subscription input box of a website persists. In this scenario, the website allows users to upload files, but the content of the uploaded files is not properly sanitized or validated. Attackers exploit this vulnerability by uploading a file containing malicious script content | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_9/index.php | |
| **Consequences of not Fixing the Issue** | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions | |
| **Suggested Countermeasures** | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| **References** | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

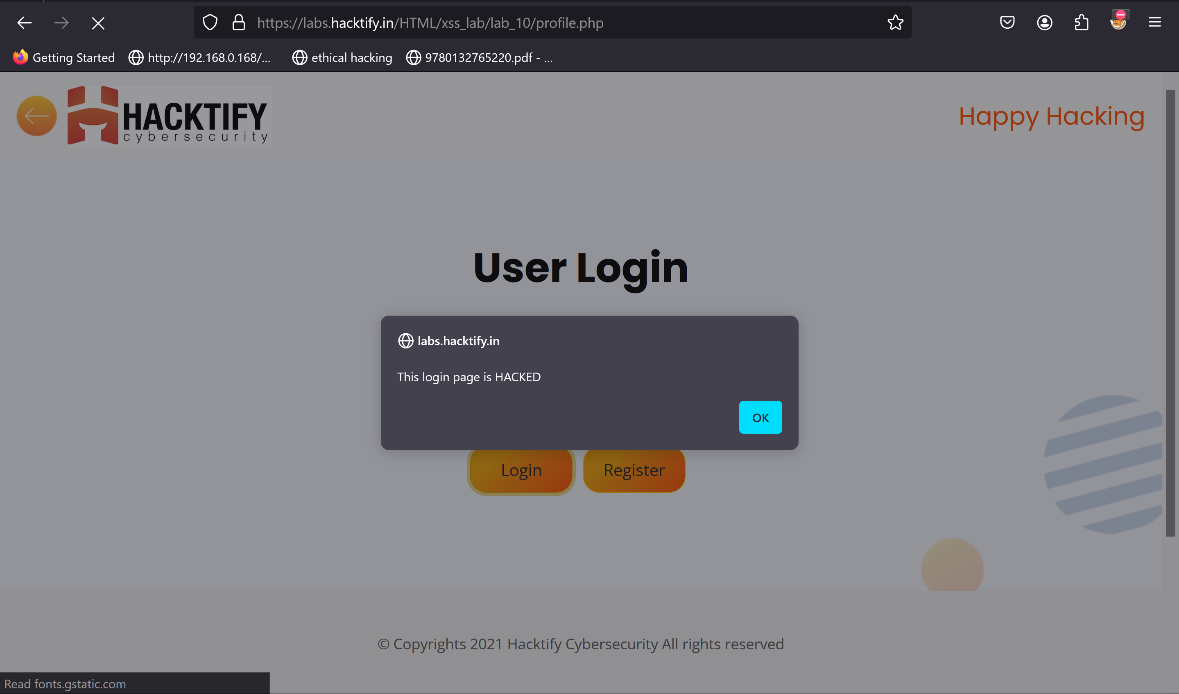
# Proof of Concept



# 1.10. Stored Everywhere!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Stored Everywhere! | **Low** |
| **Tools Used** | |
| No specialized tools were used; manual testing was conducted. | |
| **Vulnerability Description** | |
| In the user login page of a website, an XSS (Cross-Site Scripting) vulnerability exists due to inadequate input sanitization in the profile upload page's first name section. We exploit this vulnerability by inserting a malicious script payload. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_10/index.php | |
| **Consequences of not Fixing the Issue** | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions | |
| **Suggested Countermeasures** | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| **References** | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

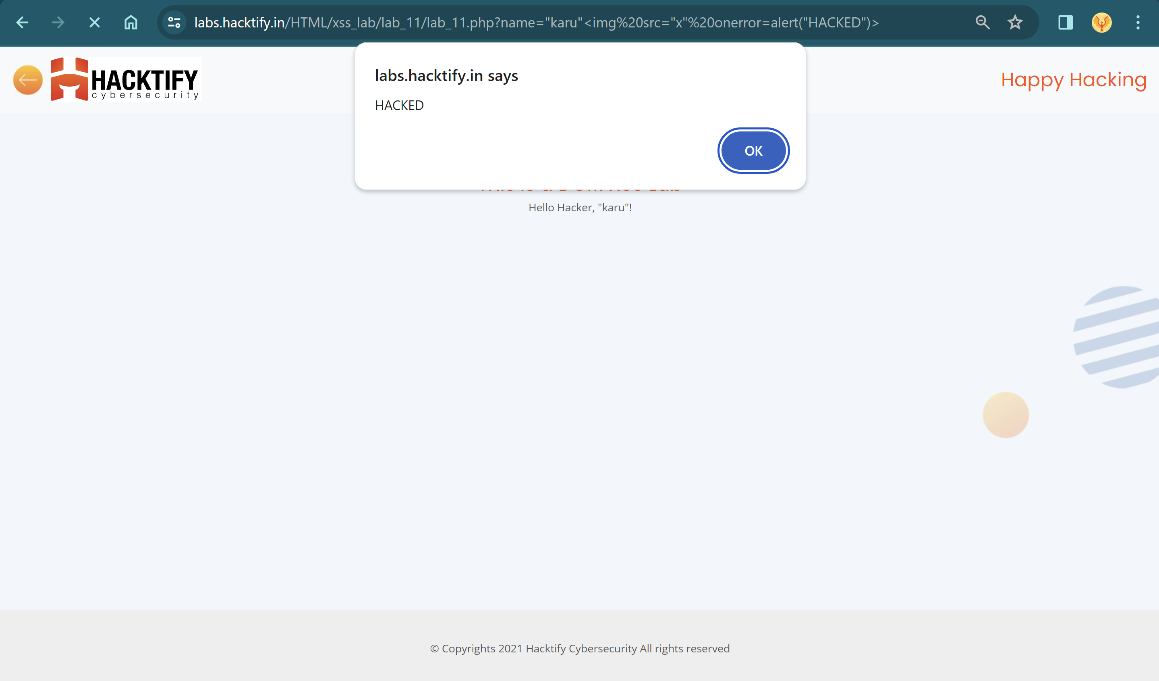
# Proof of Concept



# 1.11. DOM’S Are Love!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| DOM’S Are Love! | **High** |
| **Tools Used** | |
| No specialized tools were used; manual testing was conducted. | |
| **Vulnerability Description** | |
| In the website, a DOM-based XSS (Cross-Site Scripting) vulnerability exists, allowing attackers to execute malicious scripts within the context of other users' browsers. This vulnerability arises due to improper handling of user-controlled data in the URL parameters, specifically the "name" parameter. We exploit this vulnerability by injecting a malicious payload. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_11/index.php | |
| **Consequences of not Fixing the Issue** | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions | |
| **Suggested Countermeasures** | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| **References** | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

# Proof of Concept

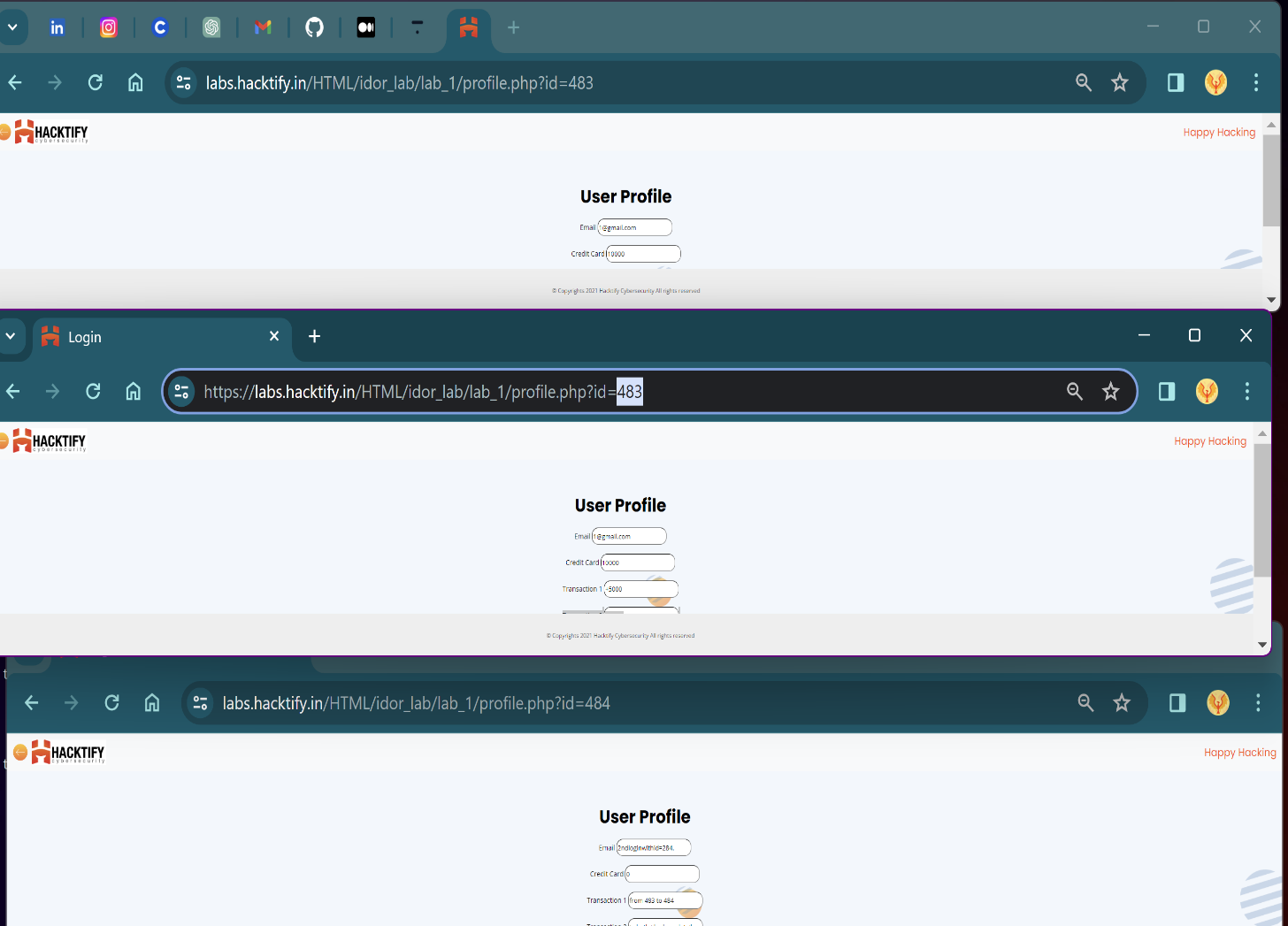


# 2. Insecure Direct Object References

# 2.1. Give Me My Amount!!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Give Me My Amount!! | **Low** |
| **Tools Used** | |
| No tool is used. | |
| **Vulnerability Description** | |
| In the bank user login page of a website, an IDOR (Insecure Direct Object Reference) vulnerability exists, allowing unauthorized access to other users' accounts. This vulnerability occurs due to inadequate access controls and improper validation of user IDs. We exploit this vulnerability by intercepting and modifying the user ID parameter in the login request using tools like Burp Suite. | |
| **How It Was Discovered** | |
| Discovered through manual analysis by intercepting and modifying requests using Burp Suite. | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/idor\_lab/lab\_1/index.php | |
| **Consequences of not Fixing the Issue** | |
| Unauthorized access to sensitive financial information of users, leading to potential identity theft or fraud. Unauthorized transactions or changes to user account settings, causing financial loss or reputational damage to the bank. | |
| **Suggested Countermeasures** | |
| Implement proper access controls and authentication mechanisms to ensure that users can only access their own accounts.  Use session tokens or other secure identifiers instead of directly exposing user IDs in URLs or requests | |
| **References** | |
| <https://owasp.org/www-community/attacks/Insecure_Direct_Object_Reference>  <https://portswigger.net/burp/documentation> | |

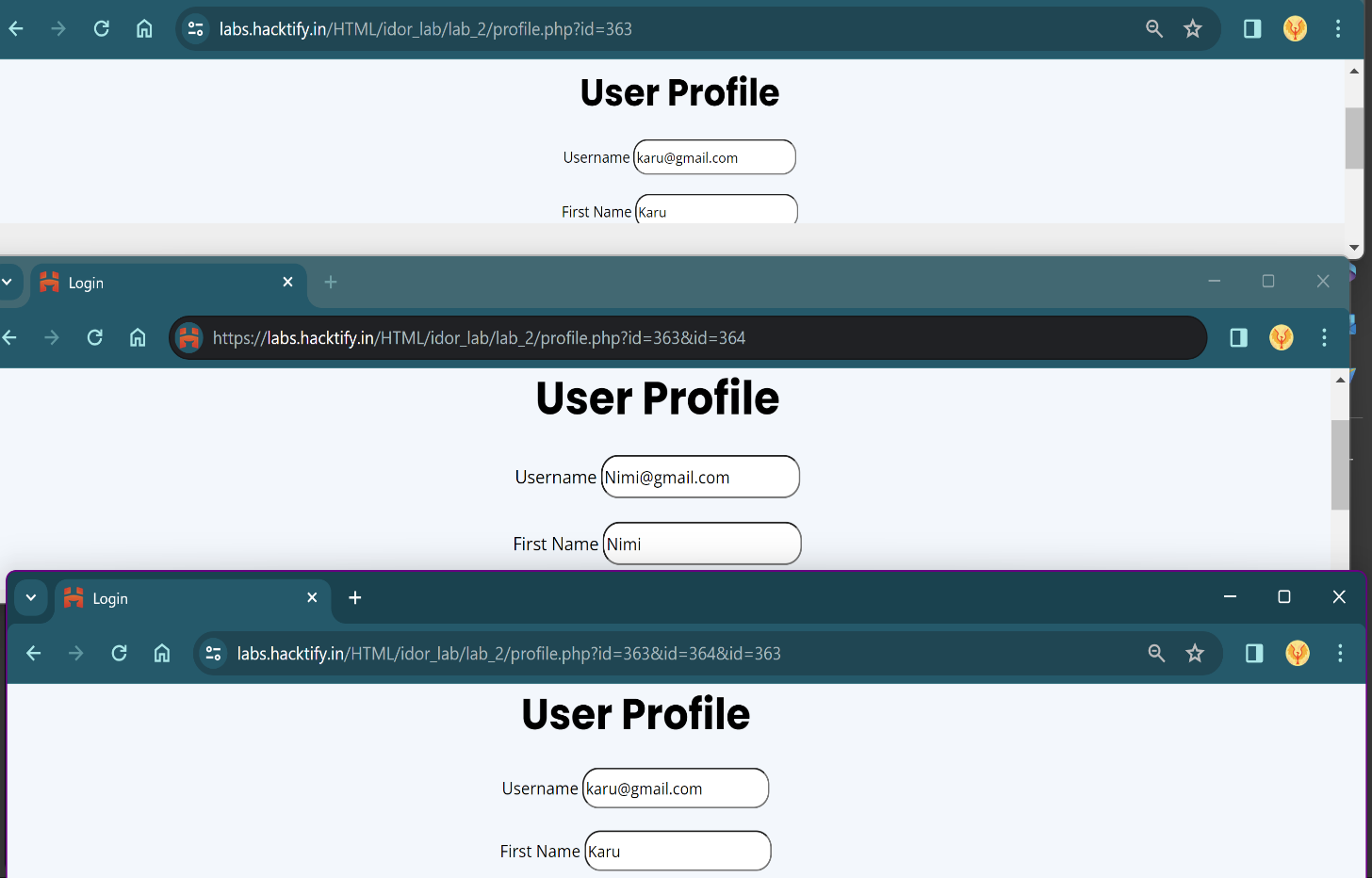
# Proof of Concept



# 2.2. Stop Polluting My Params!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Stop Polluting My Params! | **Medium** |
| **Tools Used** | |
| Manual analysis and no specific tools is used. | |
| **Vulnerability Description** | |
| In the user login page of a website, an IDOR (Insecure Direct Object Reference) vulnerability exists, enabling unauthorized access to other users' accounts by manually changing the user ID in the URL. This vulnerability arises due to inadequate access controls and improper validation of user IDs. We exploit this vulnerability by directly manipulating the user ID parameter in the URL to access other user’s accounts without authentication. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/idor\_lab/lab\_2/index.php | |
| **Consequences of not Fixing the Issue** | |
| Unauthorized access to sensitive financial information of users, leading to potential identity theft or fraud. Unauthorized transactions or changes to user account settings, causing financial loss or reputational damage to the bank | |
| **Suggested Countermeasures** | |
| Implement proper access controls and authentication mechanisms to ensure that users can only access their own accounts.  Use session tokens or other secure identifiers instead of directly exposing user IDs in URLs or requests | |
| **References** | |
| <https://www.varonis.com/blog/what-is-idor-insecure-direct-object-reference> | |

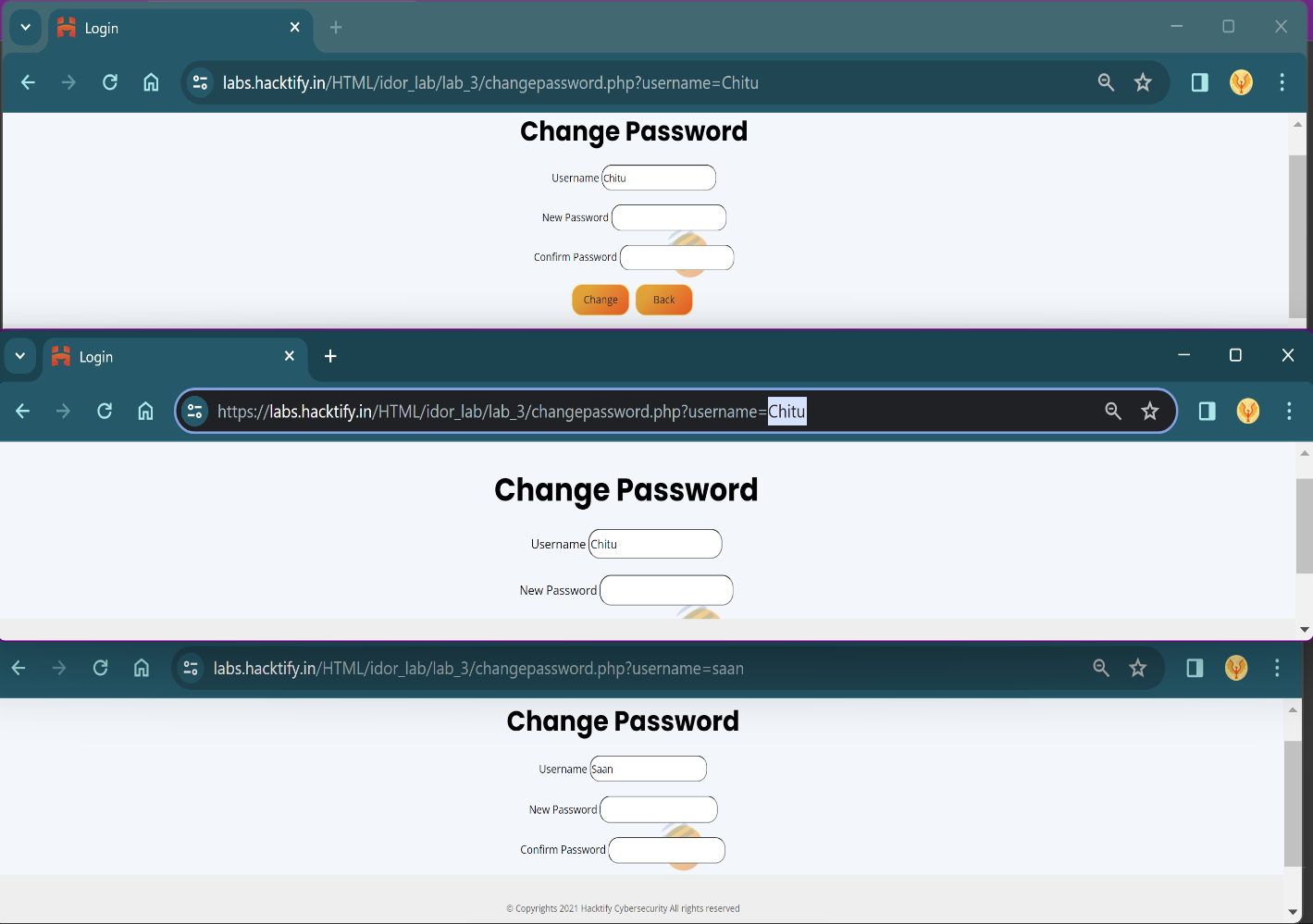
# Proof of Concept



# 2.3. Someone Changed My Password!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Someone Changed My Password! | **High** |
| **Tools Used** | |
| None, manual analysis was conducted. | |
| **Vulnerability Description** | |
| Insecure Direct Object Reference (IDOR) allows unauthorized access to resources by manipulating object identifiers in URLs. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/idor\_lab/lab\_3/index.php | |
| **Consequences of not Fixing the Issue** | |
| Potential unauthorized access to sensitive data, compromise of user accounts, and legal liabilities. | |
| **Suggested Countermeasures** | |
| Implement proper access controls, such as session-based access instead of direct object references, and perform input validation on user-supplied data. | |
| **References** | |
| <https://www.scaler.com/topics/cyber-security/idor/> | |

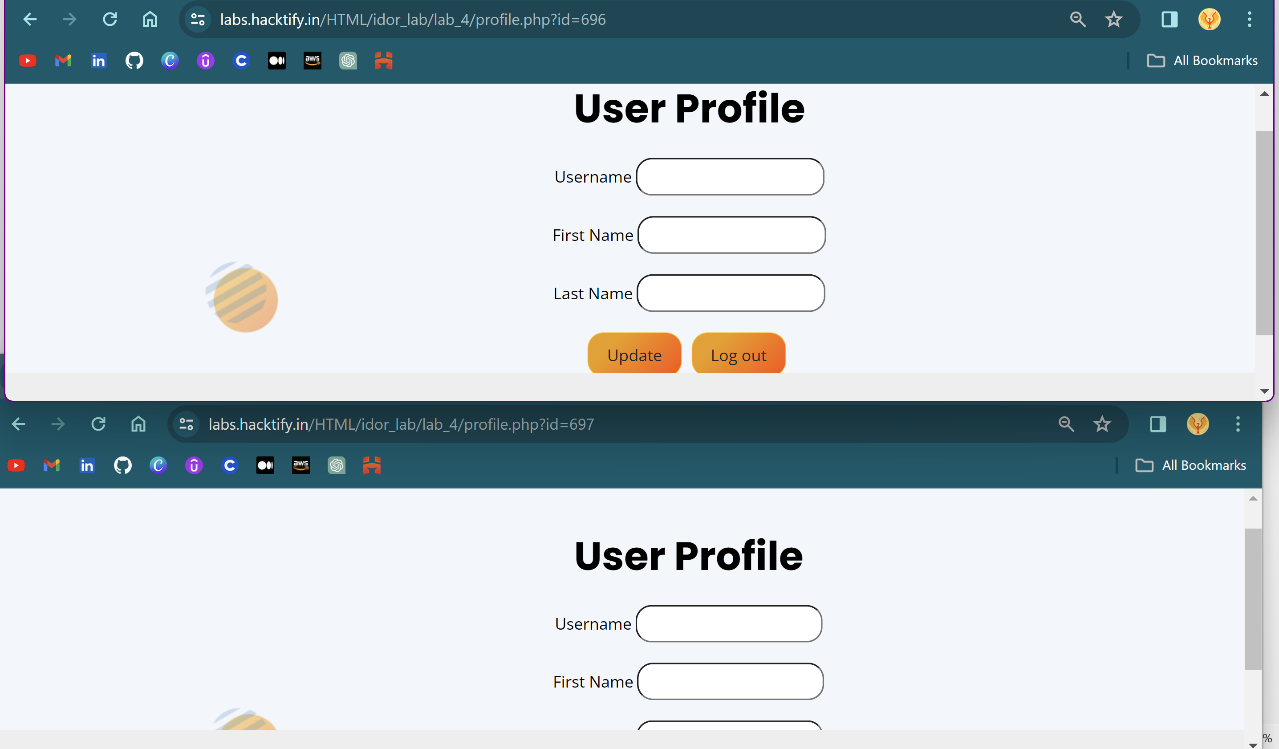
# Proof of Concept

****

# 2.4. Change your Methods!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Change your Methods! | **Medium** |
| **Tools Used** | |
| None, manual analysis was conducted. | |
| **Vulnerability Description** | |
| Insecure Direct Object Reference (IDOR) allows unauthorized access to resources by manipulating object identifiers in URLs. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/idor\_lab/lab\_4/index.php | |
| **Consequences of not Fixing the Issue** | |
| Potential unauthorized access to sensitive data, compromise of user accounts, and legal liabilities. | |
| **Suggested Countermeasures** | |
| Implement proper access controls, such as session-based access instead of direct object references, and perform input validation on user-supplied data. | |
| **References** | |
| -NIL- | |

# Proof of Concept:

****