Week 1 Penetration Testing Report

Full Name: Atharva Jagdale
Program: HCPT
Date: 15-02-2025

Introduction

This report document hereby describes the proceedings and results of a Black Box security assessment conducted against **Week 1 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 1 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	**{HTML Injection}, {Cross-Site Scripting}**

3. Summary

Outlined is a Black Box Application Security assessment for **Week 1 Labs**.

Total number of Sub-labs: 17 Sub-labs

High	Medium	Low
4	5	8

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**

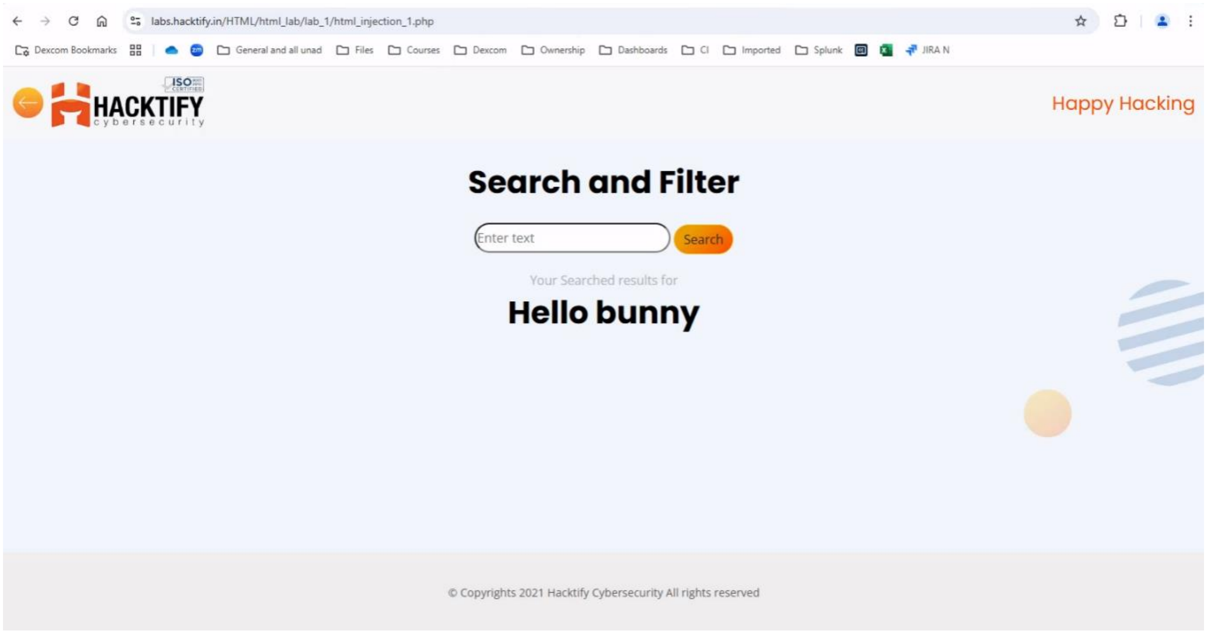
1. HTML Injection

1.1. HTML's are easy!

Reference	Risk Rating
Sub-lab-1: HTML's are easy!	Low
Tools Used	
Google chrome	
Vulnerability Description	
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. This can occur when user input is not properly validated or sanitized before being included in the webpage's output. Attackers can exploit HTML injection vulnerabilities to manipulate the appearance or functionality of the webpage, steal sensitive information such as login credentials or session cookies, or redirect users to malicious websites. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.	
How It Was Discovered	
I found this vulnerability by using basic HTML tags i.e., h1 tag.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
This can lead to managing and accessing other web pages and it can lead to phishing other users.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Educate developers on secure coding practices. 4. Regularly audit and test for vulnerabilities.	
References	
1. https://www.acunetix.com/vulnerabilities/web/html-injection/ 2. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection 3. https://www.imperva.com/learn/application-security/html-injection/	
Payload Used	
<h1>Hello bunny</h1>	

Proof of Concept

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



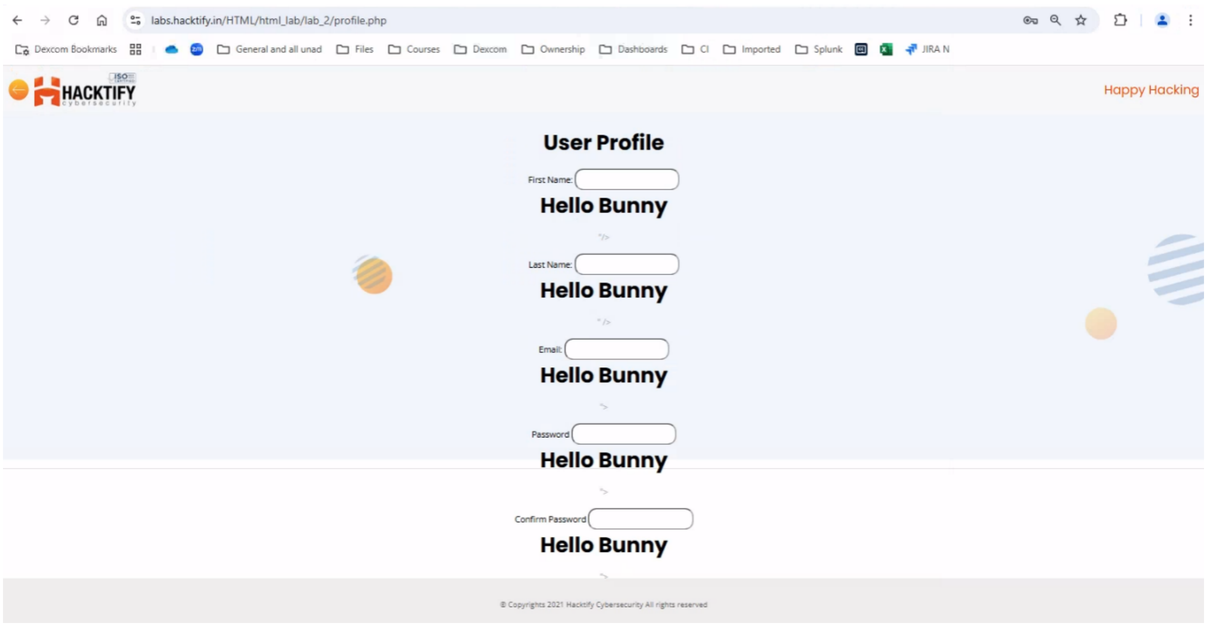
labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php
HACKTIFY cybersecurity
Happy Hacking
Search and Filter
Enter text
Search
Your Searched results for
Hello bunny
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.2. Let me store them

Reference	Risk Rating
Sub-lab-2: Let me store them	Low
Tools Used	
Google chrome	
Vulnerability Description	
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. This can occur when user input is not properly validated or sanitized before being included in the webpage's output. Attackers can exploit HTML injection vulnerabilities to manipulate the appearance or functionality of the webpage, steal sensitive information such as login credentials or session cookies, or redirect users to malicious websites. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.	
How It Was Discovered	
I found this vulnerability by trying multiple HTML tags on different input fields. Here, I can run every HTML valid code to inject malicious code.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ html_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
Can steal another person's identity.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Educate developers on secure coding practices. 4. Regularly audit and test for vulnerabilities.	
References	
1. https://www.acunetix.com/vulnerabilities/web/html-injection/ 2. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection 3. https://www.imperva.com/learn/application-security/html-injection/	
Payload Used	
"><h1>Hello Bunny</h1>	

Proof of Concept

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



labs.hacktify.in/HTML/html_lab/lab_2/profile.php

Happy Hacking

User Profile

First Name:

Hello Bunny

Last Name:

Hello Bunny

Email:

Hello Bunny

Password

Hello Bunny

Confirm Password

Hello Bunny

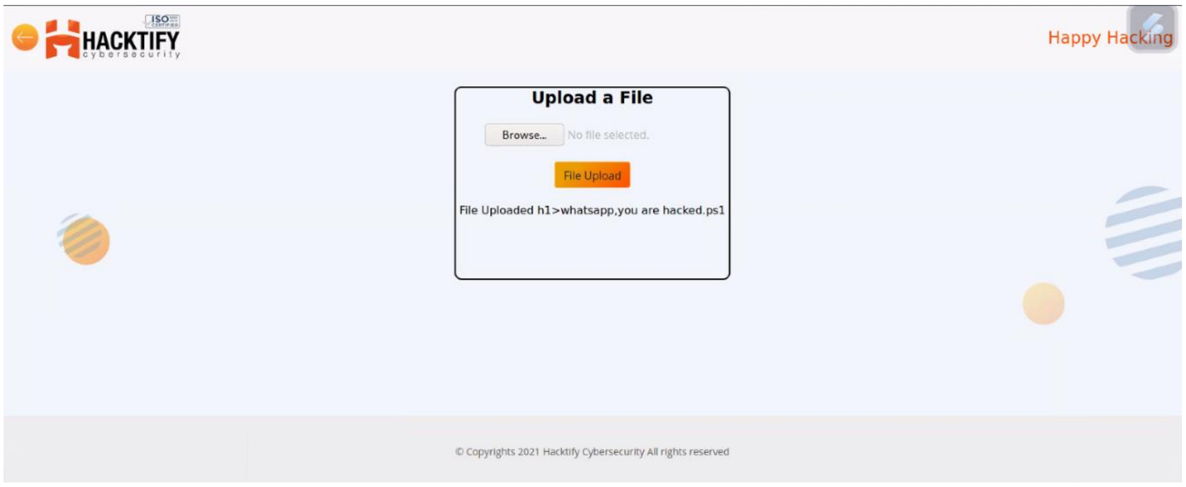
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.3. File names are also vulnerable

Reference	Risk Rating
Sub-lab-3: File names are vulnerable	Low
Tools Used	
Google chrome and Burp Suite	
Vulnerability Description	
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. This can occur when user input is not properly validated or sanitized before being included in the webpage's output. Attackers can exploit HTML injection vulnerabilities to manipulate the appearance or functionality of the webpage, steal sensitive information such as login credentials or session cookies, or redirect users to malicious websites. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.	
How It Was Discovered	
I found this vulnerability by intercepting a web app on BurpSuite and changing the filename on it send it back on the web to a successful run.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ html_lab/lab_3/lab_3.php	
Consequences of not Fixing the Issue	
Malicious file names may distract the user and it can cause the user to do something illegal or wrong on the web.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Educate developers on secure coding practices. 4. Regularly audit and test for vulnerabilities.	
References	
1. https://www.acunetix.com/vulnerabilities/web/html-injection/ 2. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection 3. https://www.imperva.com/learn/application-security/html-injection/	
Payload Used	
<h1>whatsapp,you are hacked.ps1</h1>	

Proof of Concept

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



File Uploaded h1>whatsapp,you are hacked.ps1

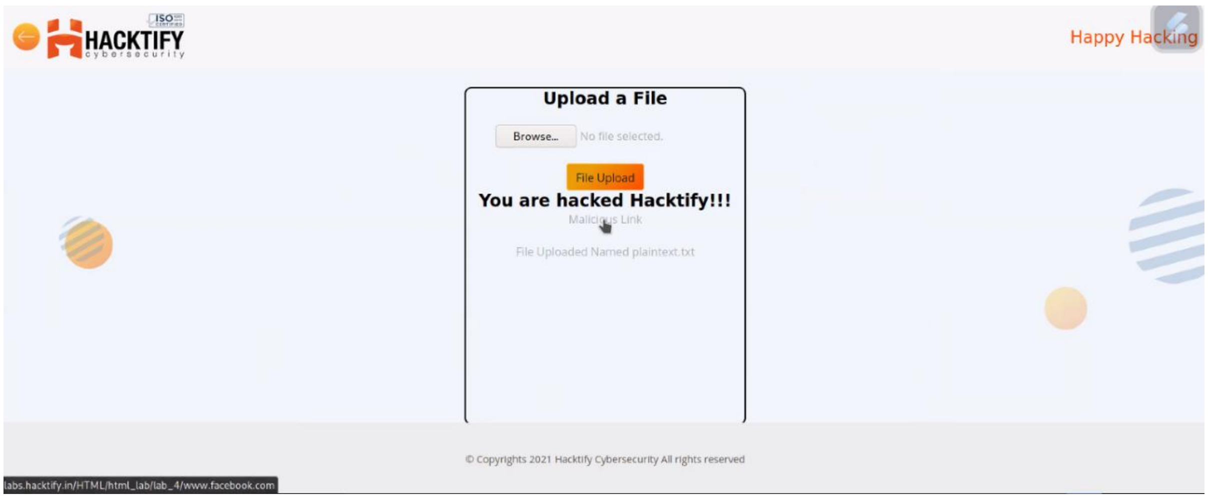
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.4. File content and HTML Injection a perfect pair

Reference	Risk Rating
Sub-lab-4: File content and HTML Injection are a perfect pair	Medium
Tools Used	
Google chrome	
Vulnerability Description	
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. This can occur when user input is not properly validated or sanitized before being included in the webpage's output. Attackers can exploit HTML injection vulnerabilities to manipulate the appearance or functionality of the webpage, steal sensitive information such as login credentials or session cookies, or redirect users to malicious websites. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.	
How It Was Discovered	
I found this vulnerability by just uploading malicious files in the file uploading section with different file extensions and containing malicious code in it.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ html_lab/lab_4/lab_4.php	
Consequences of not Fixing the Issue	
It can lead to taking down an entire website and the website will show wrong data content on the website.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Educate developers on secure coding practices. 4. Regularly audit and test for vulnerabilities.	
References	
1. https://www.acunetix.com/vulnerabilities/web/html-injection/ 2. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection 3. https://www.imperva.com/learn/application-security/html-injection/	
Payload Used	
<h1>You are Hacked</h1> in a text file and then uploaded over there	

Proof of Concept

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



You are hacked Hacktify!!!

Malicious Link

File Uploaded Named plaintext.txt

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

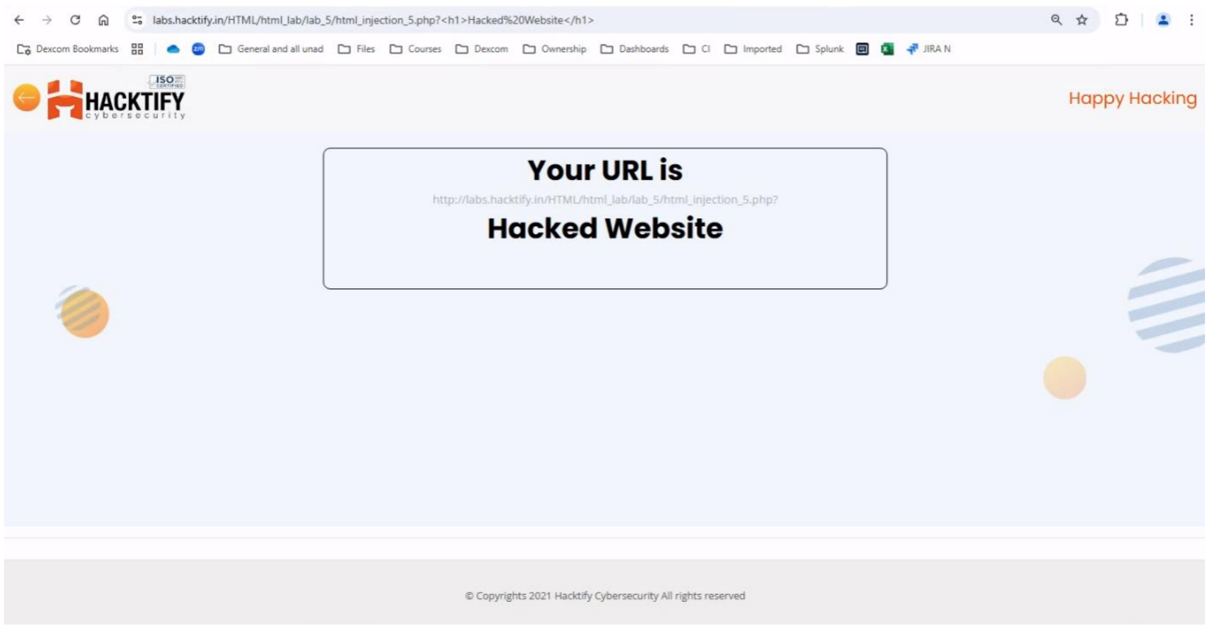
labs.hacktify.in/HTML/html_lab/lab_4/www.facebook.com

1.5. Injecting HTML using URL

Reference	**Risk Rating**
Sub-lab-5: Injecting HTML using URL	Medium
Tools Used	
Google chrome	
Vulnerability Description	
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. This can occur when user input is not properly validated or sanitized before being included in the webpage's output. Attackers can exploit HTML injection vulnerabilities to manipulate the appearance or functionality of the webpage, steal sensitive information such as login credentials or session cookies, or redirect users to malicious websites. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.	
How It Was Discovered	
I found this vulnerability by tampering the actual website of the code and injecting valid HTML code as malicious code or payload	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ html_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
It can allow attackers to modify the URL and modify the web pages	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Educate developers on secure coding practices. 4. Regularly audit and test for vulnerabilities.	
References	
1. https://www.acunetix.com/vulnerabilities/web/html-injection/ 2. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection 3. https://www.imperva.com/learn/application-security/html-injection/	
Payload Used	
?<h1>Hacked Website</h1> in the URL	

Proof of Concept

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



labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?<h1>Hacked%20Website</h1>

Your URL is

http://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?

Hacked Website

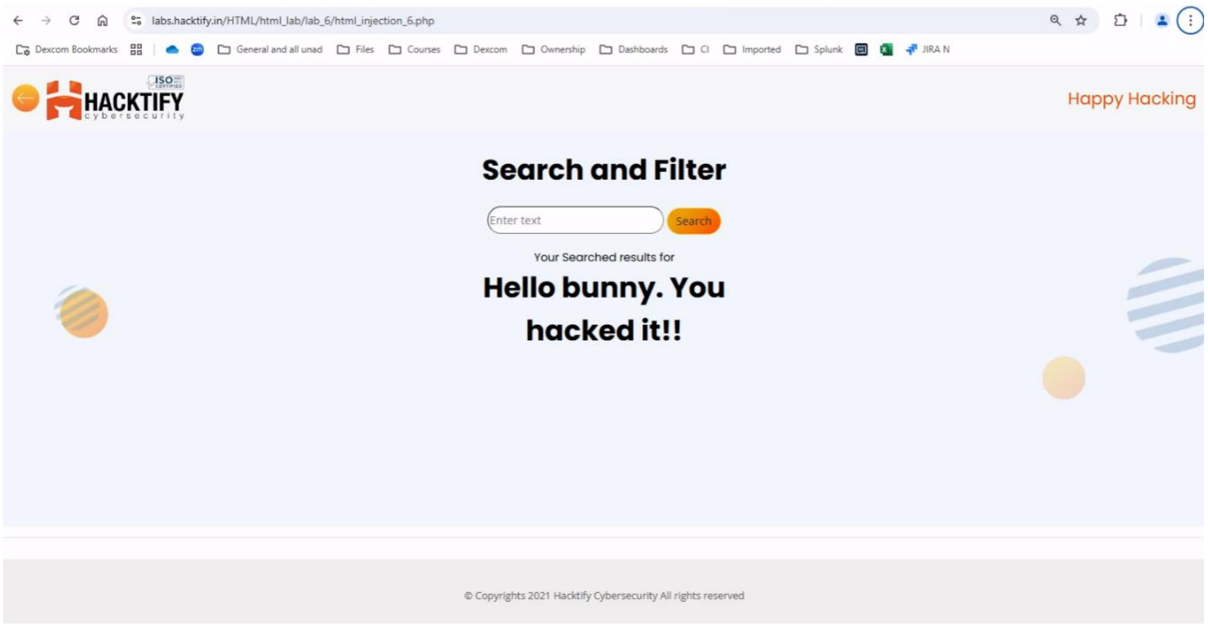
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.6. Encode it

Reference	**Risk Rating**
Sub-lab-6: Encode it	**High**
Tools Used	
Google chrome and URL encoder	
Vulnerability Description	
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. This can occur when user input is not properly validated or sanitized before being included in the webpage's output. Attackers can exploit HTML injection vulnerabilities to manipulate the appearance or functionality of the webpage, steal sensitive information such as login credentials or session cookies, or redirect users to malicious websites. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.	
How It Was Discovered	
I found this vulnerability by entering basic HTML valid code by encoding it into URL encoder	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ html_lab/lab_6/lab_6.php	
Consequences of not Fixing the Issue	
This can lead to managing and accessing other web pages and it can lead to phishing other users.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Educate developers on secure coding practices. 4. Regularly audit and test for vulnerabilities.	
References	
1. https://www.acunetix.com/vulnerabilities/web/html-injection/ 2. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection 3. https://www.imperva.com/learn/application-security/html-injection/ 4. https://www.urlencoder.org/	
Payload Used	
Plain Text - <h1?Hello bunny. You hacked it!!</h1> **Encoded Text** - %3Ch1%3FHello%20bunny.%20You%20hacked%20it%21%21%3C%2Fh1	

Proof of Concept

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



Search and Filter

Your Searched results for

Hello bunny. You hacked it!!

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

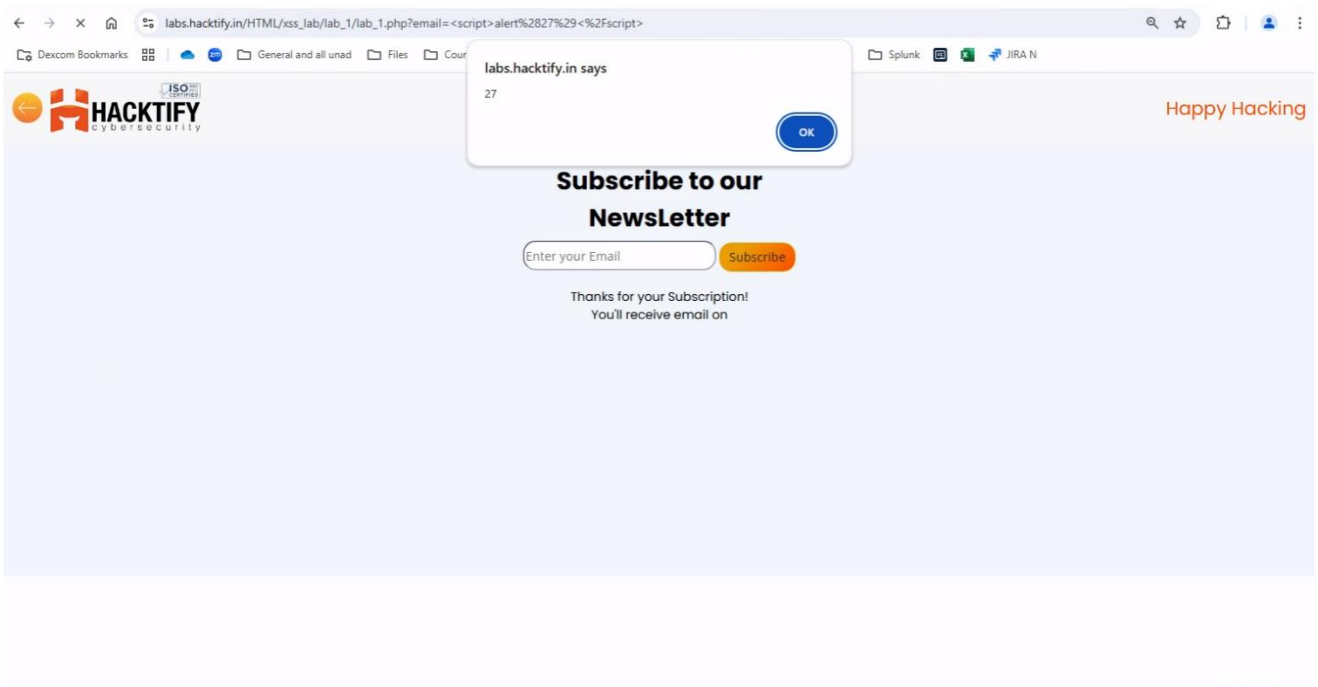
2. Cross-Site Scripting

2.1. Let's do it!

Reference	**Risk Rating**
Sub-lab-1: Let's Do IT!	Low
Tools Used	
Google chrome	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
I found this vulnerability by entering simple javascript code into the client side reflection field, i.e., I successfully performed reflected xss.	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
XSS enables an attacker to inject client-side code, so with this, an attacker will be able to capture client-side cookies which can capture account credentials of any user.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list	
Payload Used	
<script>alert(27)</script>	

Proof of Concept

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



labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php?email=<script>alert%2827%29<%2Fscript>

labs.hacktify.in says

27

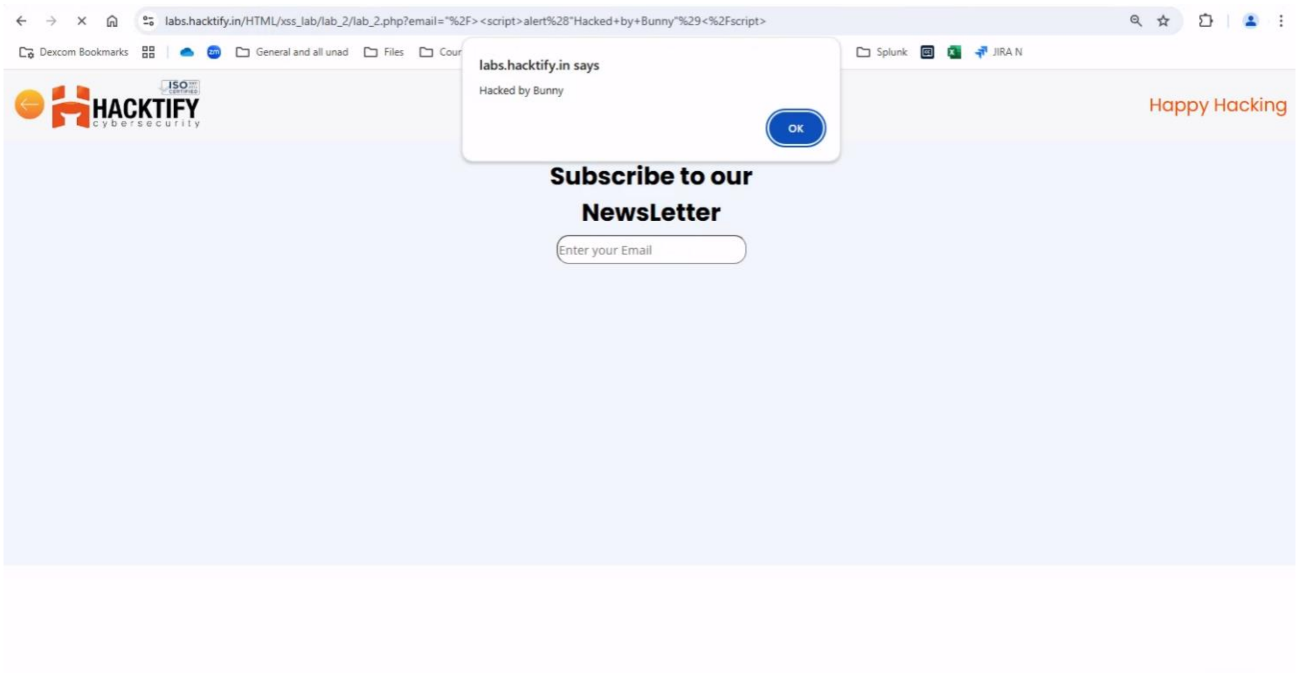
Thanks for your Subscription!
You'll receive email on

2.2. Balancing is important in life

Reference	**Risk Rating**
Sub-lab-2: Balancing is important in life	Low
Tools Used	
Google chrome	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
I found this vulnerability by modifying the URL after getting valid output as actually developed. Also, it will allow an attacker to carry out any action that the user is able to perform.	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list	
Payload Used	
"/><script>alert("Hacked by Bunny")</script>	

Proof of Concept

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


```
labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php?email="%2F><script>alert%28"Hacked+by+Bunny"%29<%2Fscript>
```

labs.hacktify.in says

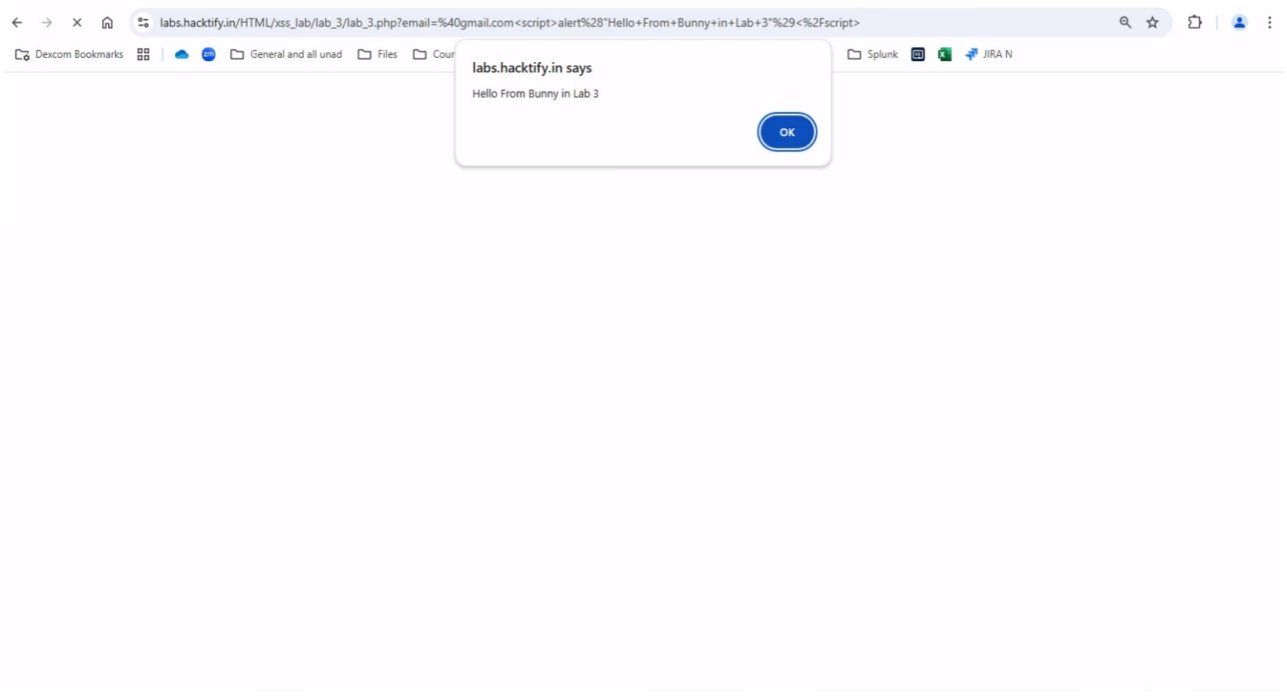
Hacked by Bunny

2.3. XSS is everywhere!

Reference	Risk Rating
Sub-lab-3: XSS is everywhere!	Low
Tools Used	
Google chrome	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
I found this vulnerability by modifying the URL after getting valid output as actually developed. Here, the developer has given validation while developing this web page as the text field must contain a valid email address which will end with "@gmail.com". If any other input is entered, then an error message will display called "Enter a valid email address". So, to bypass this I simple inserted my payload which will end with provided validation for input, i.e., "@gmail.com"	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list	
Payload Used	
@gmail.com<script>alert("Hello From Bunny in Lab 3")</script>	

Proof of Concept

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



```
labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=%40gmail.com<script>alert%28"Hello+From+Bunny+in+Lab+3"%29<%2Fscript>
```

labs.hacktify.in says

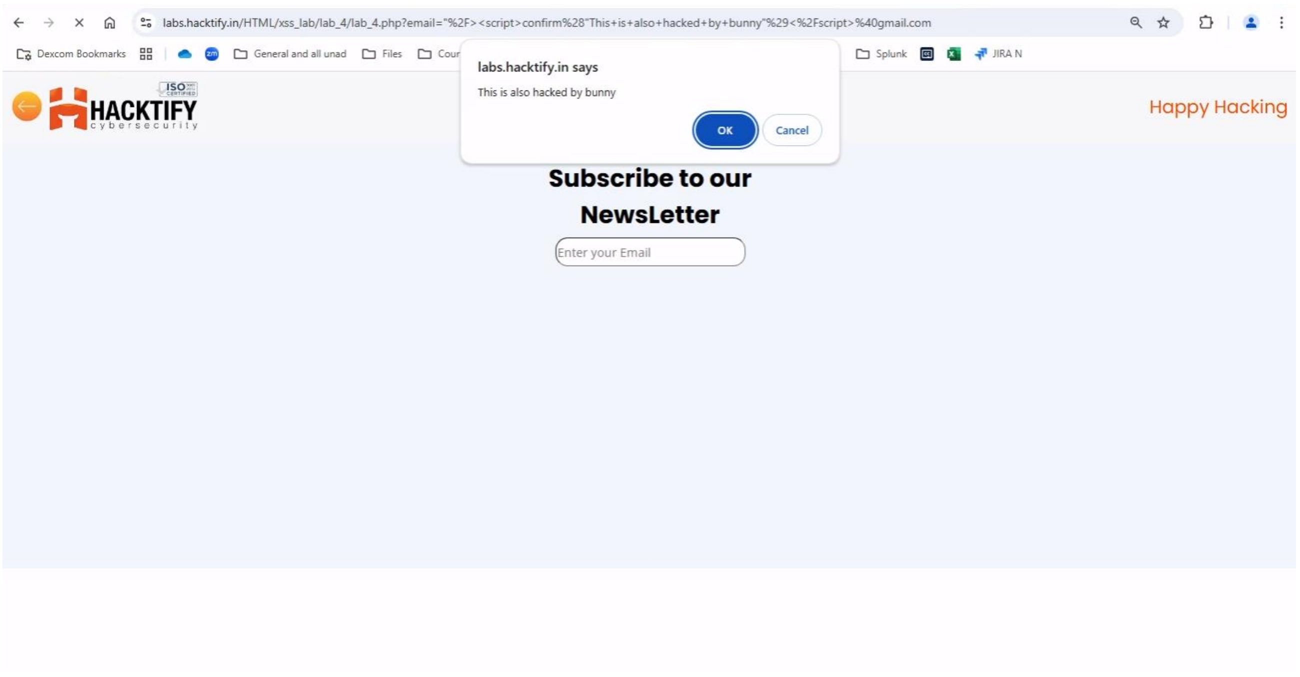
Hello From Bunny in Lab 3

2.4. Alternatives are a must!

Reference	**Risk Rating**
Sub-lab-4: Alternatives are a must	Medium
Tools Used	
Google chrome	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
I found this vulnerability by modifying the URL after getting valid output as actually developed. But, here the developer has sanitized the tag which is commonly used in JavaScript, i.e., alert(). So, to bypass this situation, I used the confirm() tag and I successfully got what I expected. Also, I am able to successfully run prompt() tag instead of alert() and alternative of confirm()	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list	
Payload Used	
"/><script>confirm("This is also hacked by bunny")</script>@gmail.com	

Proof of Concept

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



```
labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php?email="%2F><script>confirm%28"This+is+also+hacked+by+bunny"%29<%2Fscript>%40gmail.com
```

labs.hacktify.in says

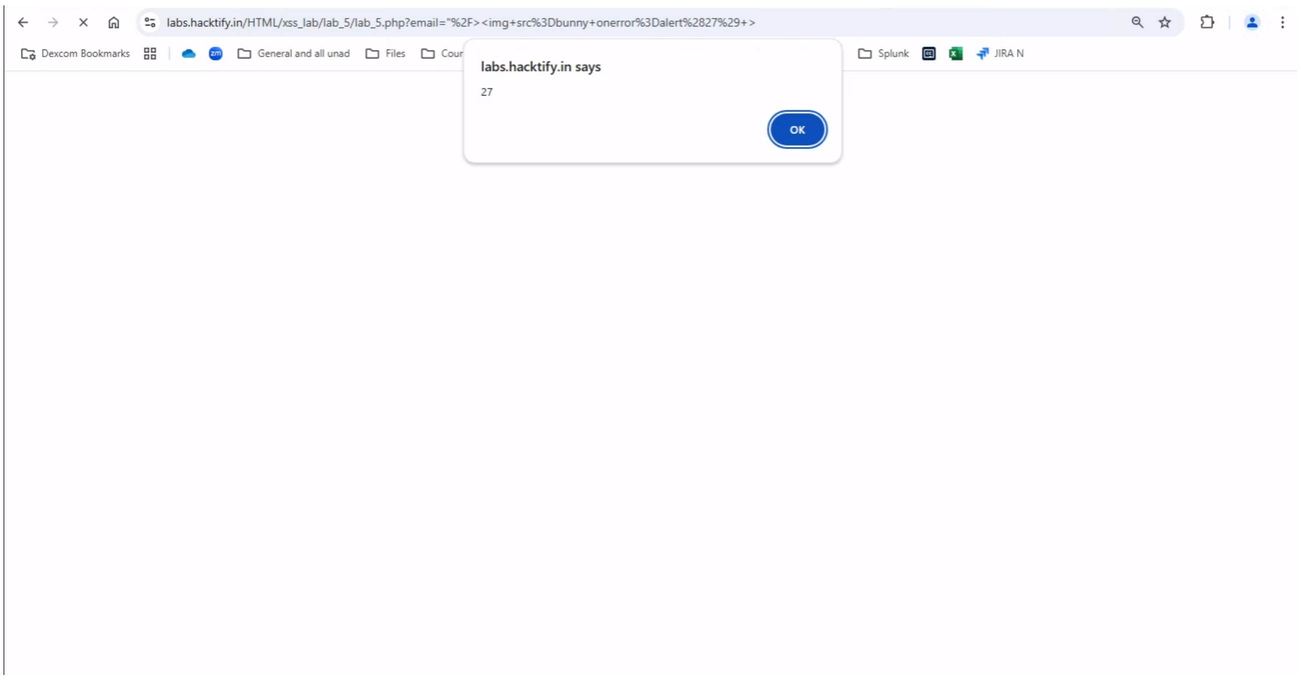
This is also hacked by bunny

2.5. Developer hates script!

Reference	**Risk Rating**
Sub-lab-5: Developers hate scripts!	**High**
Tools Used	
Google chrome	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
I found this vulnerability by using different javascript tags like <img>, <svg>, etc	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list	
Payload Used	
"/><img src=bunny onerror=alert(27) >	

Proof of Concept

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



```
labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php?email="%2F><img+src%3Dbunny+onerror%3Dalert%2827%29+>
```

labs.hacktify.in says

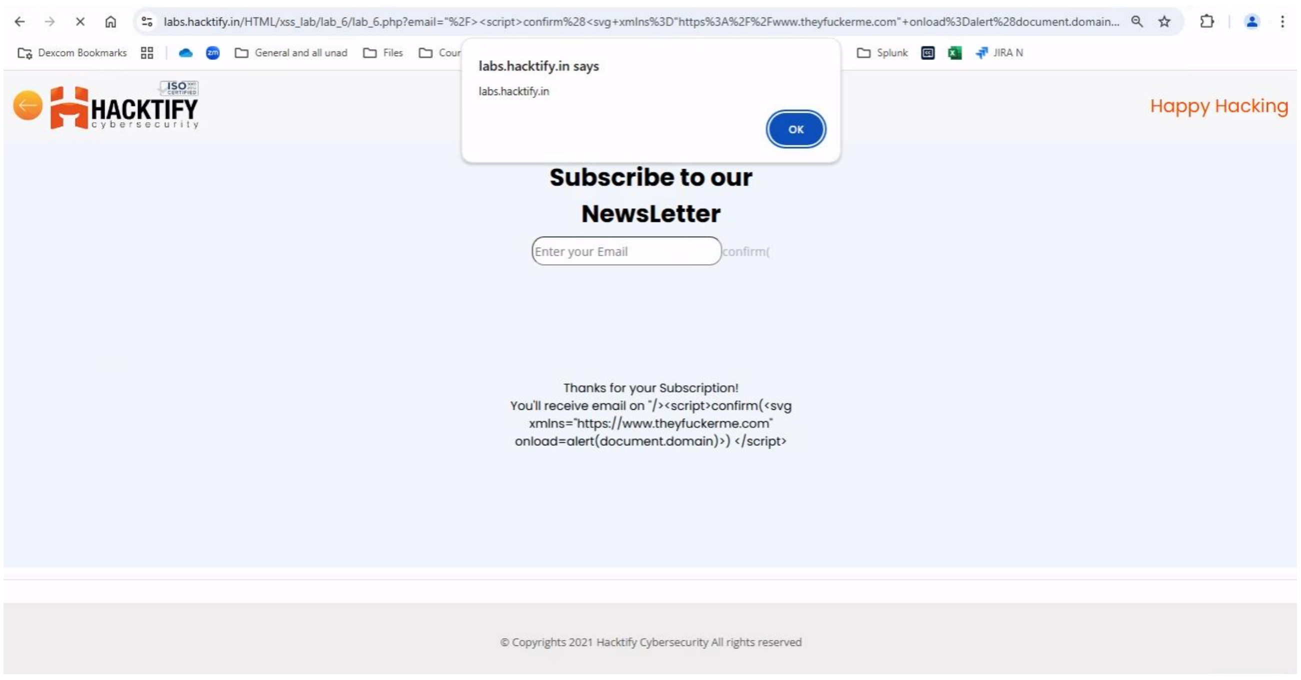
27

2.6. Change the variation!

Reference	Risk Rating
Sub-lab-6: Change the variation!	**High**
Tools Used	
Google chrome	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
I found this vulnerability by merging two different tags within a single command as payload to bypass the situation which the developer has created. I used, <script> tag in which I used <svg> tag to bypass it	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list	
Payload Used	
"/><script>confirm(<svg xmlns="https://www.theyfuckerme.com" onload=alert(document.domain)>) </script>	

Proof of Concept

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



```
labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php?email="%2F><script>confirm%28<svg+xmlns%3D"https%3A%2F%2Fwww.theyfuckerme.com"+onload%3Dalert%28document.domain...
```

labs.hacktify.in says

labs.hacktify.in

Subscribe to our NewsLetter

Thanks for your Subscription!
You'll receive email on "/><script>confirm(<svg xmlns="https://www.theyfuckerme.com" onload=alert(document.domain)>) </script>

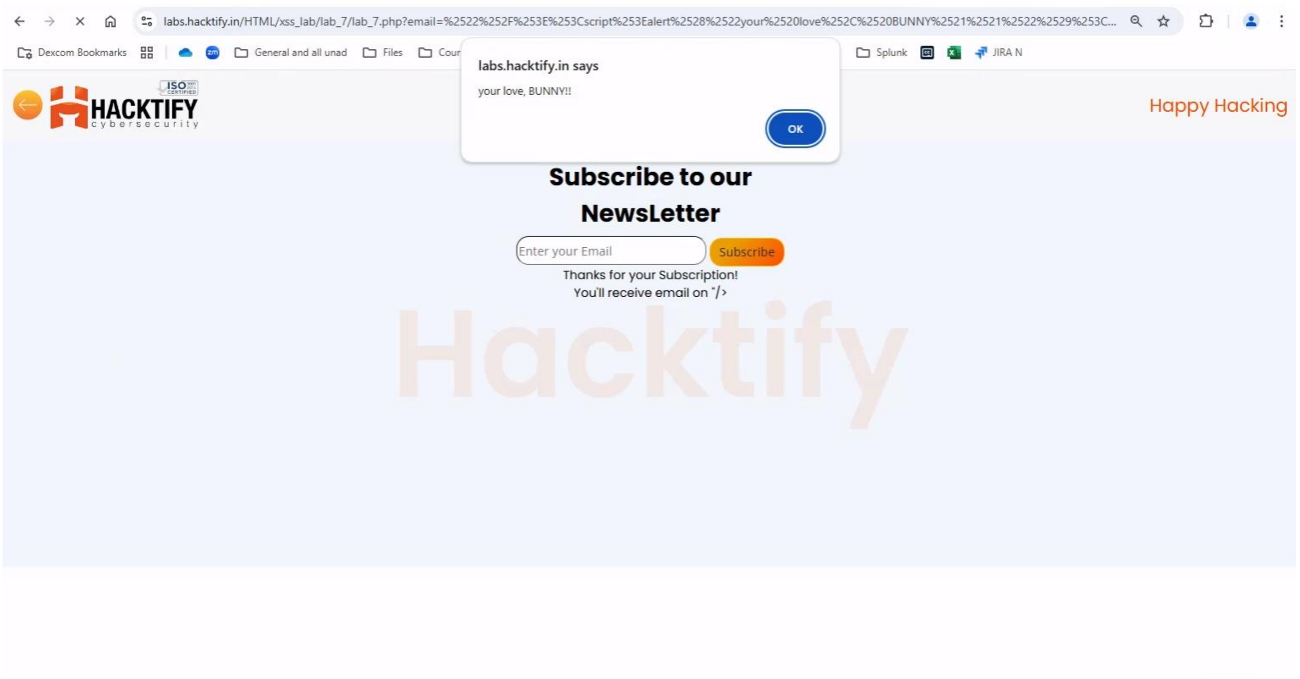
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

2.7. Encoding is the key?

Reference	**Risk Rating**
Sub-lab-7: Encoding is the key?	Medium
Tools Used	
Google chrome and URL encoder	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
This vulnerability was found by URL encoding a simple javascript command with the help of URL encoder	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list 5. https://www.urlencoder.org/	
Payload Used	
Plain Text - "/><script>alert("your love, BUNNY!!")</script> **Encoded Payload** - %22%2F%3E%3Cscript%3Ealert%28%22your%20love%2C%20BUNNY%21%21%22%29%3C%2Fscript%3E	

Proof of Concept

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



```
labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php?email=%2522%252F%253E%253Cscript%253Ealert%2528%2522your%2520love%252C%2520BUNNY%2521%2521%2522%2529%253C...
```

labs.hacktify.in says

your love, BUNNY!!

Subscribe to our NewsLetter

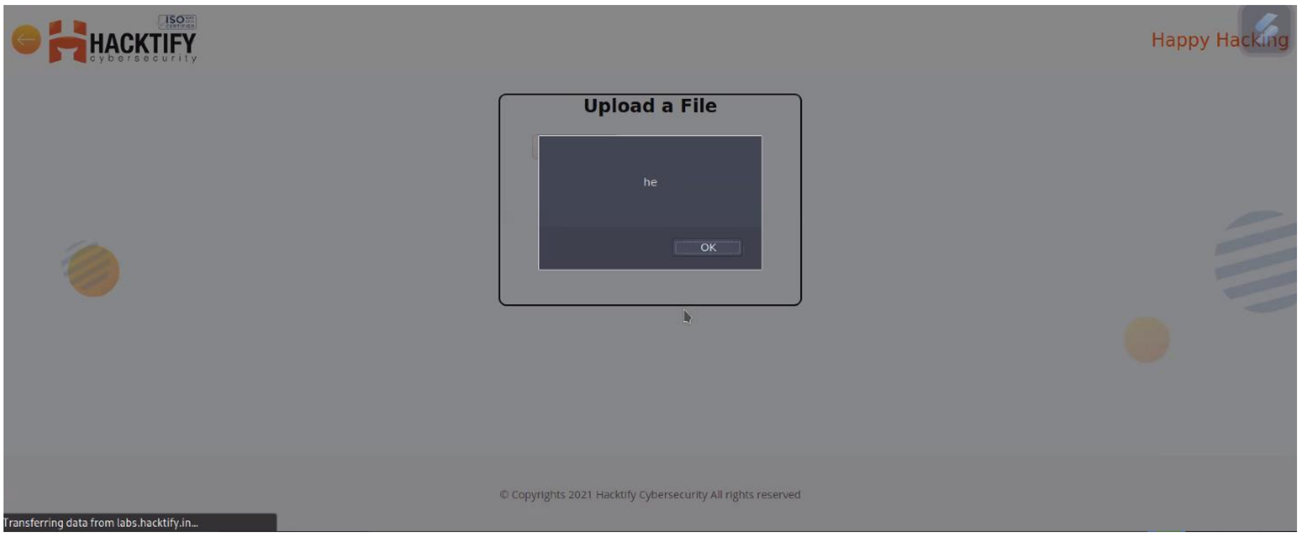
Thanks for your Subscription!
You'll receive email on "/>

2.8. XSS with file upload (file name)

Reference	**Risk Rating**
Sub-lab-8: XSS with file upload (file name)	**Low**
Tools Used	
Google chrome and Burp Suite	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
I found this vulnerability by intercepting a file upload request onto Burp Suite tool and renaming the file name to insert payload into filename to bypass all the validation to load payload into the web pages and database to store payload into it.	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list	
Payload Used	
"/><img src=bunny onerror=alert('he') >	

Proof of Concept

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

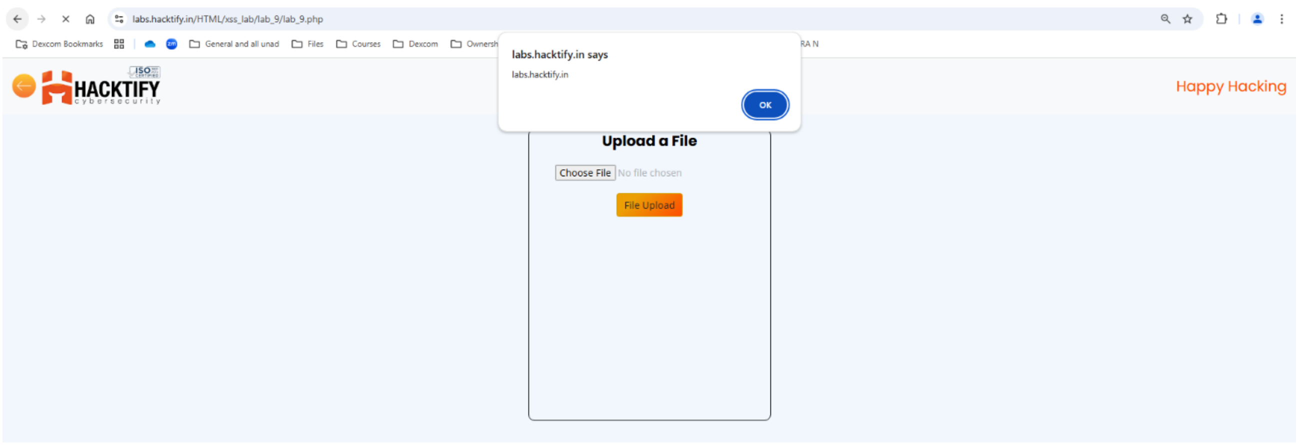

HACKTIFY
cybersecurity
Happy Hacking
Upload a File
he
OK
© Copyrights 2021 Hacktify Cybersecurity All rights reserved
Transferring data from labs.hacktify.in...

2.9. XSS with file upload (file content)

Reference	**Risk Rating**
Sub-lab-9: XSS with file upload (file content)	**Medium**
Tools Used	
Google chrome	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
I found this vulnerability by uploading a malicious code file into the file upload section. In malicious code, I used <script> tag to design a malicious code.	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list	
Payload Used	
<script>alert(document.domain)</script>	

Proof of Concept

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


labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php

labs.hacktify.in says

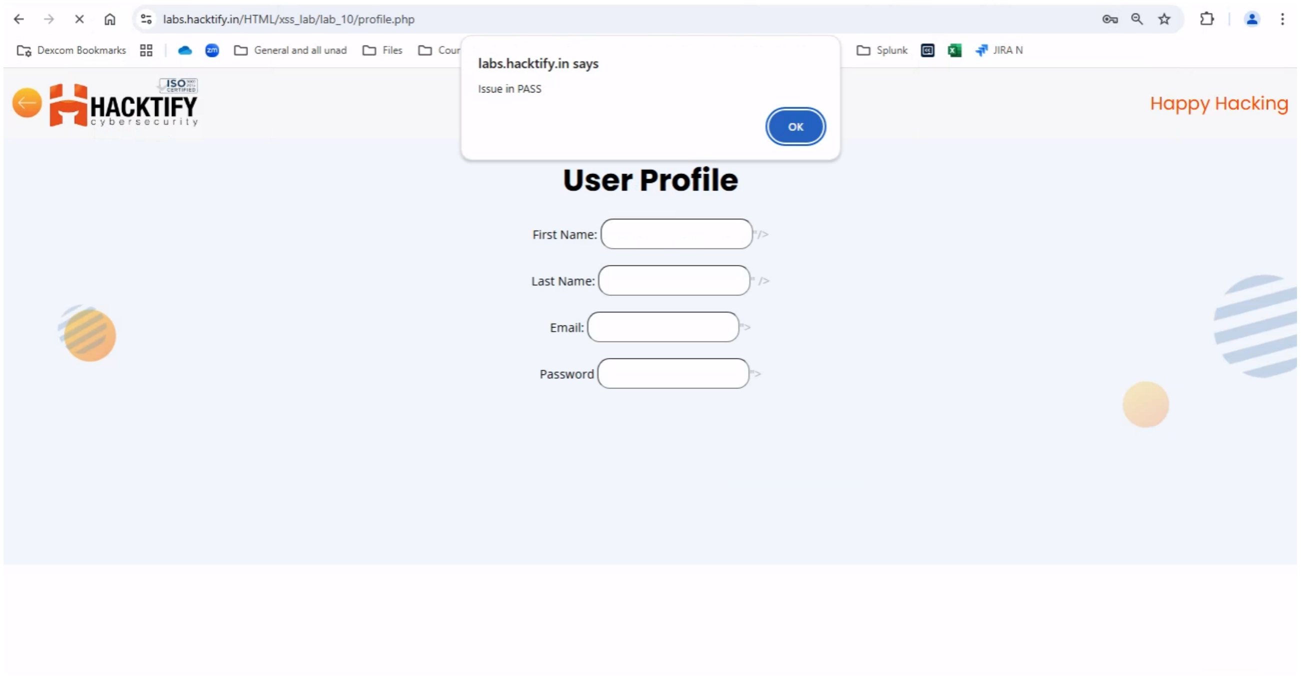
labs.hacktify.in

2.10. Stored Everywhere

Reference	**Risk Rating**
Sub-lab-10: Stored Everywhere	**Low**
Tools Used	
Google chrome	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
I found this vulnerability by registering myself into register web page. I inserted a simple <script> tag in all fields which are required to fill for registration. After registration, I tried logging in into the webpage and after the login request, I was able to verify that I successfully implemented a stored XSS attack. Here, these credentials will be stored into the database which can affect other used credentials as well as other database related features.	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_10/lab_10.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list	
Payload Used	
1. "/><script>alert("Issue in FNAME")</script> 2. "/><script>alert("Issue in LNAME")</script> 3. "/><script>alert("Issue in EMAIL")</script> 4. "/><script>alert("Issue in PASS")</script>	

Proof of Concept

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



labs.hacktify.in/HTML/xss_lab/lab_10/profile.php

labs.hacktify.in says

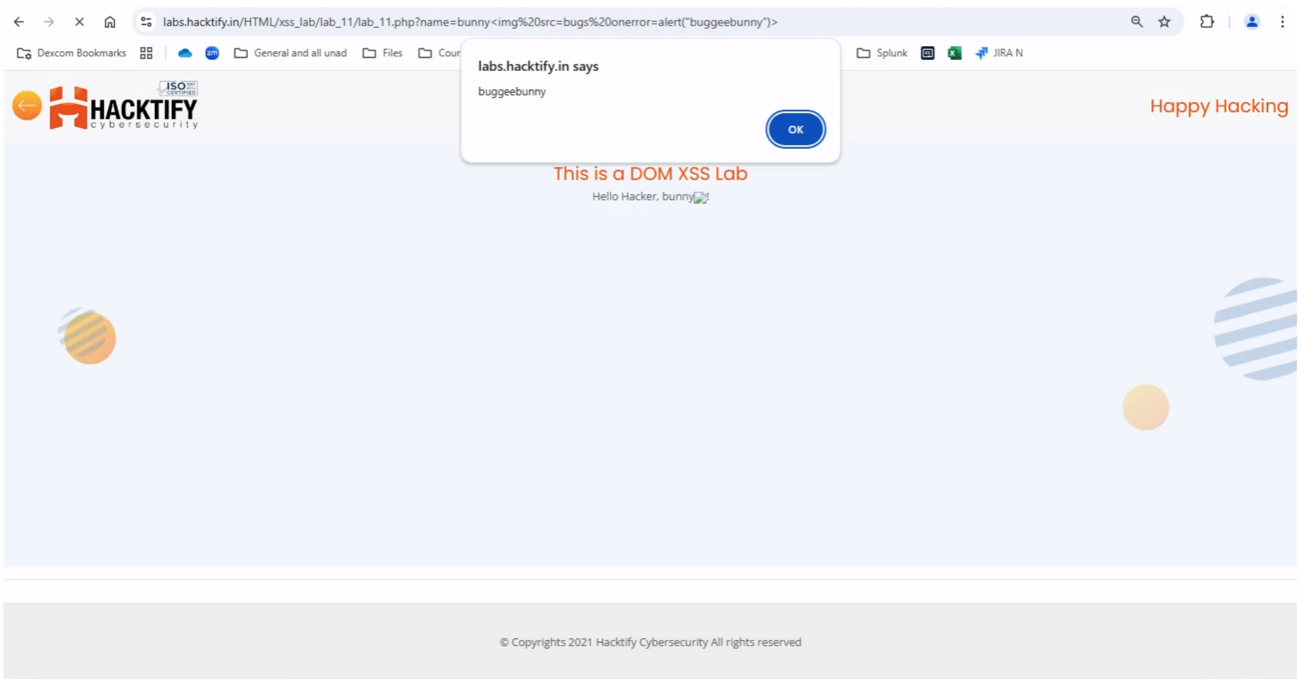
Issue in PASS

2.11. DOM’s are Love

Reference	Risk Rating
DOM’s are Love	**High**
Tools Used	
Google chrome	
Vulnerability Description	
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.	
How It Was Discovered	
I found this vulnerability by inspecting the source code of the web page and finding out what variables are used in this section. And later on, tried to load payload using <img> tag.	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
1. Comprehensive Input Validation and Sanitization 2. Proper Output Escaping to Prevent Code Execution 3. Configuring Content Security Policies (CSP) for Enhanced Protection 4. Configure Essential HTTP Security Headers for Enhanced Protection 5. Deploy a Web Application Firewall (WAF) to Block Malicious Payloads 6. Educate developers on secure coding practices. 7. Regularly audit and test for vulnerabilities.	
References	
1. https://github.com/s0md3v/AwesomeXSS 2. https://portswigger.net/web-security/cross-site-scripting 3. https://owasp.org/www-community/attacks/xss/ 4. https://github.com/payloadbox/xss-payload-list	
Payload Used	
Payload - name=bunny<img src=bugs onerror=alert("buggeebunny")> URL - https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=bunny%3Cimg%20src=bugs%20onerror=alert(%22buggeebunny%22)%3E	

Proof of Concept

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



labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=bunny<img%20src=bugs%20onerror=alert("buggeebunny")>

labs.hacktify.in says

buggeebunny

This is a DOM XSS Lab

Hello Hacker, bunny!

© Copyrights 2021 Hacktify Cybersecurity All rights reserved