

Penetration Testing Report

Full Name: G M Sohanur Rahman

Program: HCS - Penetration Testing Internship Week-2

Date: 2 March, 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: 14 Sub-labs

High	Medium	Low
4	5	5

High - Number of Sub-labs with hard difficulty level

Medium - Number of Sub-labs with Medium difficulty level

Low

-

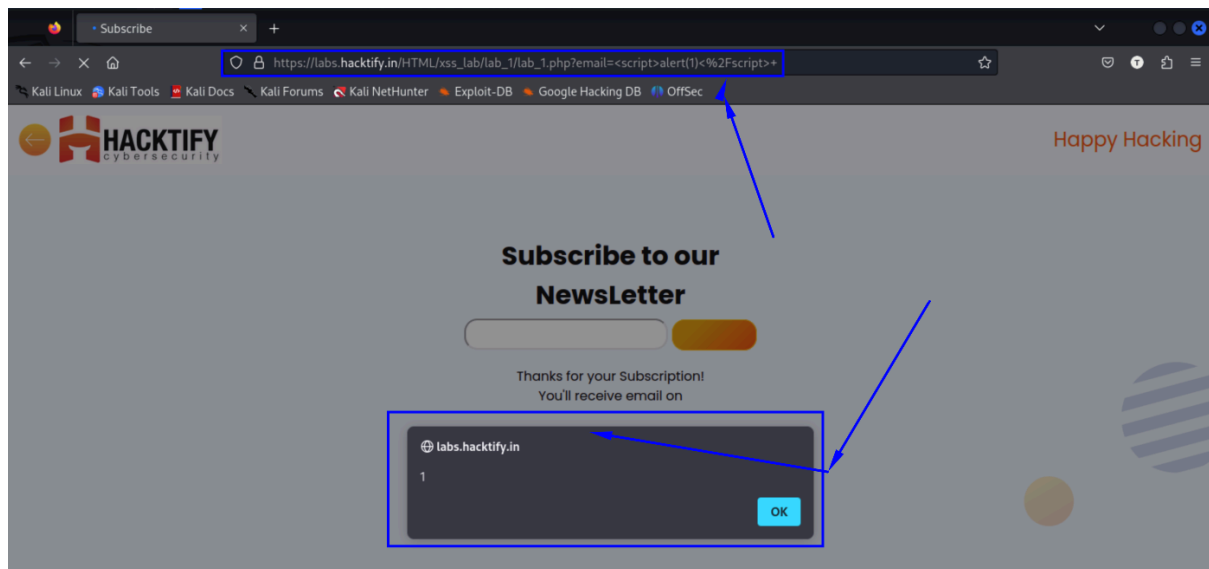
Number of Sub-labs with Easy difficulty level

1. Cross Site Scripting

1.1. Let's Do IT!

Reference	Risk Rating
Let's Do IT!	Low
Tools Used	
Firefox Browser	
Vulnerability Description	
<p>Reflected Cross-Site Scripting (XSS) is a type of security vulnerability that occurs in web applications. It allows an attacker to inject malicious scripts into the content of a trusted website, which are then executed by the victim's browser. Reflected XSS attacks are typically delivered via email or other web-based communication that includes a link with malicious script in the URL. When the victim clicks on the link, the malicious script is sent to the vulnerable website, which reflects the attack back to the victim's browser. The browser then executes the script because it appears to be a script from the trusted website.</p>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
<p>Not fixing a Reflected Cross-Site Scripting (XSS) vulnerability can lead to severe consequences, including theft of sensitive information, account takeovers, phishing attacks, and malware distribution. Such vulnerabilities compromise user trust and can damage an organization's reputation. Additionally, there may be legal and financial repercussions, especially if the breach involves personally identifiable information. Ensuring web applications are secure against XSS attacks is crucial to protect both users and organizations from these potential impacts.</p>	
Suggested Countermeasures	
<p>To counteract Reflected XSS vulnerabilities, it's essential to adopt secure coding practices that include validating and sanitizing all user inputs, encoding outputs to prevent malicious content from executing, and implementing Content Security Policies (CSP) to restrict the sources of executable scripts. Additionally, employing Web Application Firewalls (WAFs) can help detect and block XSS attacks. Regular security testing and keeping software up to date are also critical measures to identify vulnerabilities early and mitigate potential risks effectively.</p>	
References	
https://portswigger.net/web-security/cross-site-scripting/reflected#:~:text=What%20is%20reflected%20cross%2Dsite,response%20in%20an%20unsafe%20way.	

Proof of Concept



1.2. Balancing Is Important In Life!

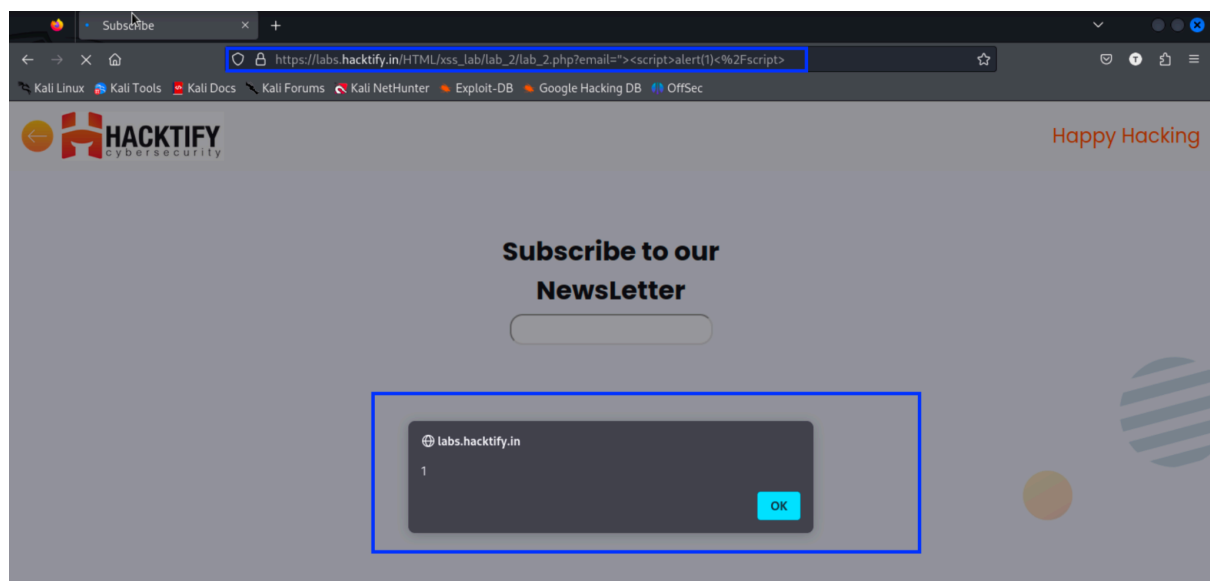
Reference	Risk Rating
Balancing Is Important In Life!	Low
Tools Used	
Firefox Browser	
Vulnerability Description	
The XSS payload "<script>alert(1)</script>" is a simple but effective demonstration of a Cross-Site Scripting attack, where "<script>" breaks out of HTML context to inject a JavaScript <script> tag, alert(1) executes a JavaScript alert function, showing the vulnerability to arbitrary script execution. This highlights the importance of sanitizing and validating user inputs in web applications to prevent attackers from injecting malicious scripts that can lead to data theft, session hijacking, and other security breaches.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
Not fixing an XSS (Cross-Site Scripting) vulnerability can have serious consequences for a website and its users. This security flaw allows attackers to inject malicious scripts into webpages viewed by other users. If not addressed, it can lead to a wide range of issues including theft of cookies, session tokens, or other sensitive information that the browser stores. Attackers can also manipulate or deface the website content displayed to users, redirect visitors to malicious sites, and perform actions on behalf of users without their consent. This not only compromises user security and privacy but can also damage the reputation of the website, erode user trust, and potentially lead to legal and financial repercussions for the entity responsible for the website.	
Suggested Countermeasures	
To counter XSS vulnerabilities, it's crucial to adopt a multifaceted approach. Primarily, input sanitization and validation should be enforced, ensuring that user inputs are checked, cleaned, and confirmed as safe before being used or displayed. Employing Content Security Policy (CSP) headers can also significantly reduce the risk by specifying which sources are trusted, thus preventing the browser from executing malicious scripts from unauthorized sources. Escaping user input is essential to ensure that any data received is treated as data, not executable code, especially in places where input is inserted	

into HTML, JavaScript, or database queries. Regular security audits and code reviews can help identify and rectify potential vulnerabilities. Additionally, using frameworks and libraries that automatically handle these security measures can further safeguard against XSS attacks by reducing the amount of security-critical code developers need to write themselves.

References

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting

Proof of Concept



1.3. XSS Is Everywhere!

Reference	Risk Rating
XSS Is Everywhere!	Low
Tools Used	
Firefox Browser	
Vulnerability Description	
Using regular expressions (regex) to detect XSS (Cross-Site Scripting) attacks involves crafting patterns that identify typical malicious inputs, such as <code><script></code> , <code>javascript:</code> , and other tags or attributes commonly used in these attacks. While regex can help flag potential XSS injections by searching for these patterns, it's not a foolproof method due to the sophistication and variety of attack vectors. False positives and negatives are common, and regex cannot account for the context of user input, making it a limited tool in XSS defense. Effective XSS prevention requires a comprehensive approach that includes input validation, output encoding, employing Content Security Policy (CSP), and adhering to secure coding practices. Relying solely on regex is insufficient; it should be part of a layered security strategy that is continuously updated to address new vulnerabilities.	
How It Was Discovered	
Manual Analysis	

Vulnerable URLs

https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php

Consequences of not Fixing the Issue

Not addressing XSS (Cross-Site Scripting) vulnerabilities can lead to severe consequences, including the theft of sensitive information such as cookies, session tokens, and personal data. Attackers can exploit these vulnerabilities to execute malicious scripts in the browsers of unsuspecting users, potentially leading to unauthorized actions on their behalf, defacement of web content, and redirection to malicious sites. This not only compromises the security and privacy of users but also damages the reputation of the affected website, eroding trust among its visitors. Furthermore, the website owners might face legal and financial repercussions for failing to safeguard user data. In summary, neglecting XSS vulnerabilities exposes both users and organizations to significant risks.

Suggested Countermeasures

Mitigating XSS vulnerabilities through regex involves carefully crafting expressions that identify and neutralize potential attack vectors in user input. However, solely relying on regex for XSS defense is not advisable due to its limitations in handling complex and obfuscated attacks. A more effective approach includes:

Input Sanitization: Cleanse user input by removing or escaping potentially harmful characters, especially in contexts where HTML or JavaScript could be executed.

Content Security Policy (CSP): Implement CSP to control the sources from which content can be loaded, effectively reducing the risk of executing unauthorized scripts.

Output Encoding: Encode output data to ensure that any input reflected back to the user is treated as data, not executable code.

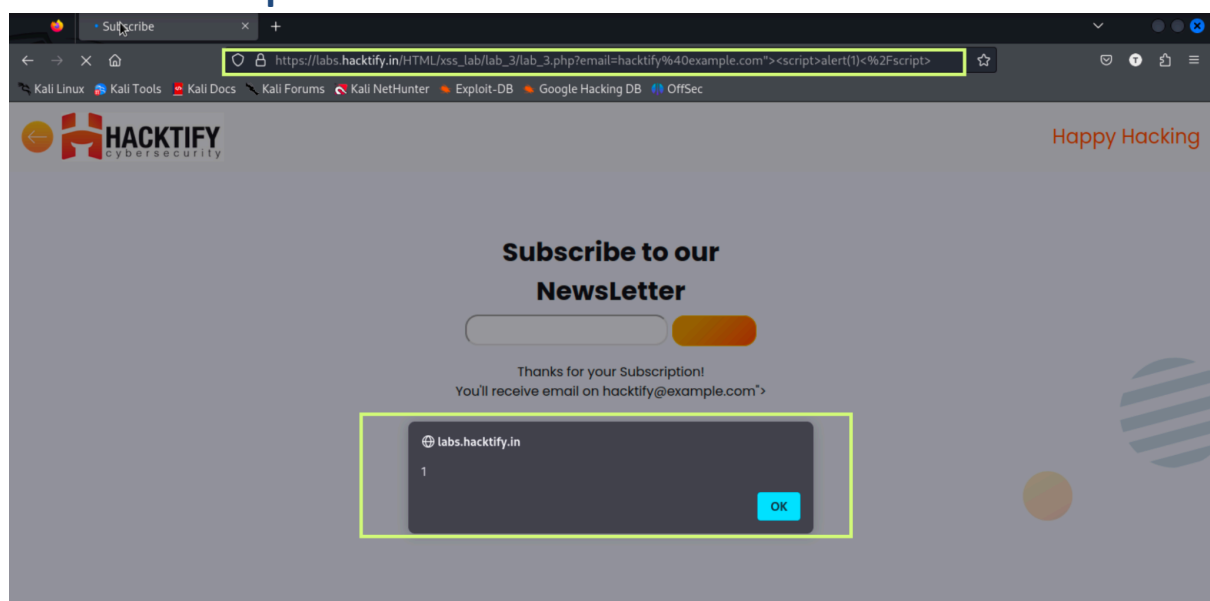
Use of Secure Frameworks: Employ frameworks and libraries that automatically apply these security measures to reduce the risk of developer oversight.

Regular Security Training: Educate developers on secure coding practices and the importance of regular updates to security measures.

References

<https://security.stackexchange.com/questions/215005/xss-mitigation-using-regex>

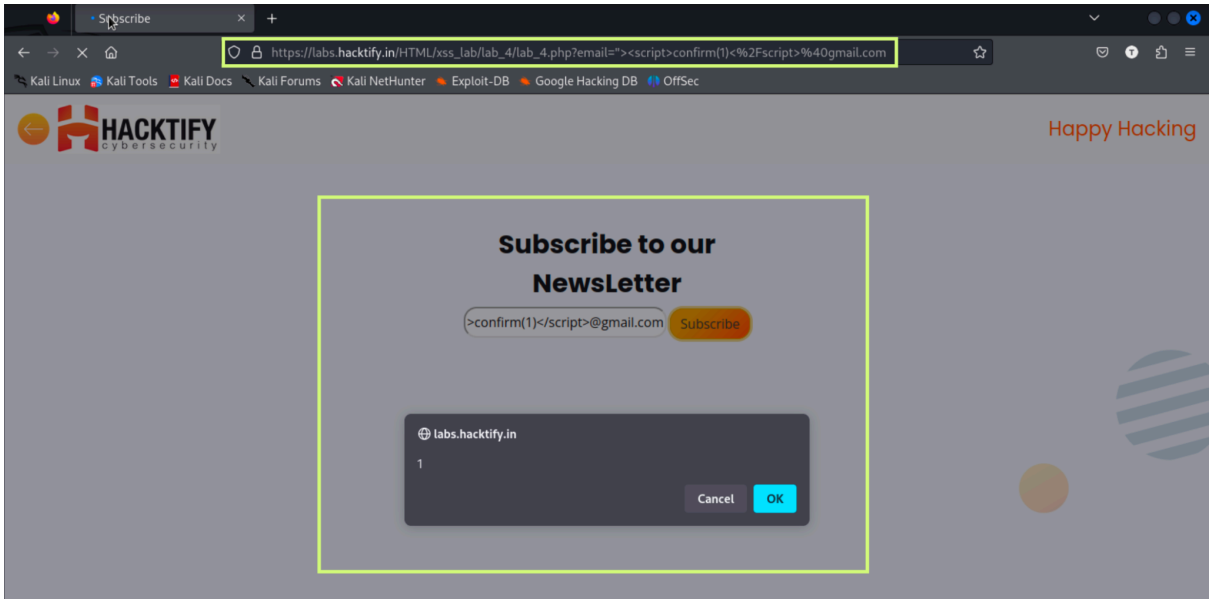
Proof of Concept



1.4. Alternatives Are Must!

Reference	Risk Rating
Alternatives Are Must!	Medium
Tools Used	
Firefox Browser	
Vulnerability Description	
Payload: "><script>confirm(1)</script>@gmail.com This payload appears to be an example of an XSS (Cross-Site Scripting) payload designed to demonstrate how malicious scripts can be injected into web applications. This specific payload attempts to execute JavaScript code (confirm(1)) when interpreted by a web browser, which could be part of an attack aimed at exploiting XSS vulnerabilities in a web application. In this case, if a web application improperly handles user input and directly includes this string in its output (for example, in an HTML page), the JavaScript code within the <script> tags could be executed in the context of the user's session. This execution could lead to various security issues, such as stealing cookies, session hijacking, redirecting the user to a malicious website, or performing actions on behalf of the user without their consent.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php	
Consequences of not Fixing the Issue	
Not addressing the XSS vulnerability demonstrated by the payload "><script>confirm(1)</script>@gmail.com" can have significant consequences. This specific attack vector can be used to execute arbitrary JavaScript code in the context of an unsuspecting user's browser. If exploited, it could lead to unauthorized actions such as session hijacking, where attackers gain control over a user's active session. It could also lead to the theft of sensitive information, including personal data and authentication credentials, as the script has the potential to access cookies and local storage. Additionally, this vulnerability can undermine the integrity and reputation of the affected website, eroding user trust and potentially resulting in legal and financial repercussions for failing to protect user data. It highlights the critical need for diligent web application security practices, including input validation and output encoding, to safeguard against XSS attacks.	
Suggested Countermeasures	
To effectively counter the XSS vulnerability exemplified by the payload "><script>confirm(1)</script>@gmail.com", a multi-layered security approach is essential. First, input validation should be rigorously applied to reject or sanitize inputs containing potentially dangerous characters or patterns. Second, output encoding is crucial; all user-supplied data displayed on web pages should be HTML-encoded to prevent the browser from interpreting it as executable code. Implementing Content Security Policy (CSP) can further restrict the execution of scripts to only trusted sources, significantly reducing the risk of XSS attacks. Additionally, adopting secure coding practices and frameworks that automatically handle these security concerns can help prevent such vulnerabilities from arising. Regular security audits and vulnerability assessments are also recommended to identify and remediate any potential XSS vulnerabilities proactively.	
References	
https://www.w3.org/TR/CSP3/	

Proof of Concept



1.5. Developer Hates Scripts!

Reference	Risk Rating
Developer Hates Scripts!	High
Tools Used	
Firefox Browser	
Vulnerability Description	
<p>Payload: "><sCript>alert(1)</sCript></p> <p>The XSS payload is designed to exploit Cross-Site Scripting vulnerabilities in web applications by injecting JavaScript code (alert(1)) into web pages. The payload uses a mix of uppercase and lowercase letters in the <script> tags to bypass simple input validation filters that might only look for case-sensitive matches of potentially dangerous tags. When this payload is improperly sanitized or encoded by a web application and embedded into an HTML page, it can be executed by the browser. This execution can result in various malicious outcomes, such as displaying an alert box (as demonstrated by alert(1)), but more harmful actions can include stealing cookies, session hijacking, and executing arbitrary JavaScript in the context of the victim's browser session. This highlights the importance of implementing robust input validation and sanitization measures that are case-insensitive and capable of detecting and neutralizing such attempts to exploit XSS vulnerabilities.</p>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
<p>Not addressing the XSS vulnerability demonstrated by the payload "><sCript>alert(1)</sCript> can lead to significant security breaches. This vulnerability allows attackers to execute arbitrary JavaScript within the context of a user's browser, potentially leading to session hijacking, where attackers gain unauthorized access to a user's session tokens or cookies. This can result in the theft of sensitive information, personal data, and authentication credentials. Furthermore, attackers can manipulate or</p>	

deface the web application, redirect users to malicious sites, or even use the compromised platform to distribute malware. The impact extends beyond individual users to the affected organization, risking its reputation, user trust, and possibly incurring legal and financial consequences for failing to safeguard user data. This underscores the critical need for thorough input sanitization and proactive security measures to prevent XSS attacks.

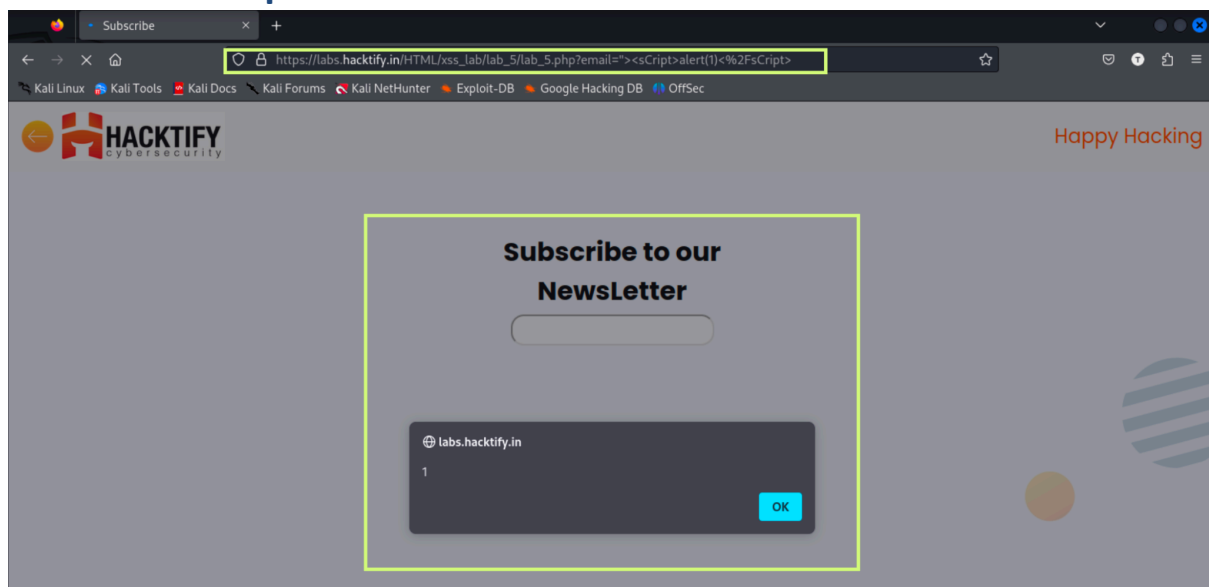
Suggested Countermeasures

To safeguard against XSS vulnerabilities like "<<sCript>alert(1)</sCript>", it's essential to adopt a layered security approach. Key countermeasures include sanitizing and validating user inputs to remove harmful scripts regardless of their case, encoding outputs to prevent browser execution of malicious code, and implementing Content Security Policy (CSP) to restrict script sources. Additionally, leveraging secure coding frameworks and libraries can automatically address these concerns. Regular security training for developers and conducting security audits can also help detect and mitigate potential vulnerabilities early. Together, these practices form a comprehensive defense strategy against XSS attacks, ensuring the security of web applications and the protection of user data.

References

<https://www.hackerone.com/knowledge-center/how-xss-payloads-work-code-examples-preventing-them>

Proof of Concept



1.6. Change The Variation!

Reference	Risk Rating
Change The Variation!	High
Tools Used	
Firefox Browser	
Vulnerability Description	

Payload: ">

The above code is a Cross-Site Scripting (XSS) payload. It works by injecting an HTML image tag () into a vulnerable web page.

Here's a breakdown of the code:

< and >: These are the start and end tags for an HTML element.

‘onerror’: This is an HTML event attribute. It specifies a JavaScript function to be executed when the browser is unable to load the specified image.

"alert(1)": This is a JavaScript function call. It will display a JavaScript alert dialog box with the number 1 when the ‘onerror’ event is triggered.

So, when this XSS payload is injected into a vulnerable web page and the browser encounters the tag with the ‘onerror’ attribute set to "alert(1)" and an invalid image source (src="invalid.jpg"), the ‘onerror’ event will be triggered, and the injected JavaScript code (alert(1) in this case) will be executed.

How It Was Discovered

Manual Analysis

Vulnerable URLs

https://labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php

Consequences of not Fixing the Issue

Failing to address the XSS vulnerability showcased by the payload "> can lead to severe consequences. This type of vulnerability allows attackers to execute arbitrary JavaScript in the context of the victim's browser, potentially leading to session hijacking, where attackers can gain unauthorized access to the user's session and steal sensitive information such as login credentials and personal data. It can also enable attackers to manipulate or deface the web page, redirect users to phishing or malware-infected sites, and perform actions on behalf of the user without their consent. The impact of such attacks extends beyond individual users to affect the integrity and reputation of the targeted website, potentially resulting in loss of trust, legal issues, and financial damages. This highlights the critical importance of promptly identifying and mitigating XSS vulnerabilities to protect both users and organizations from these risks.

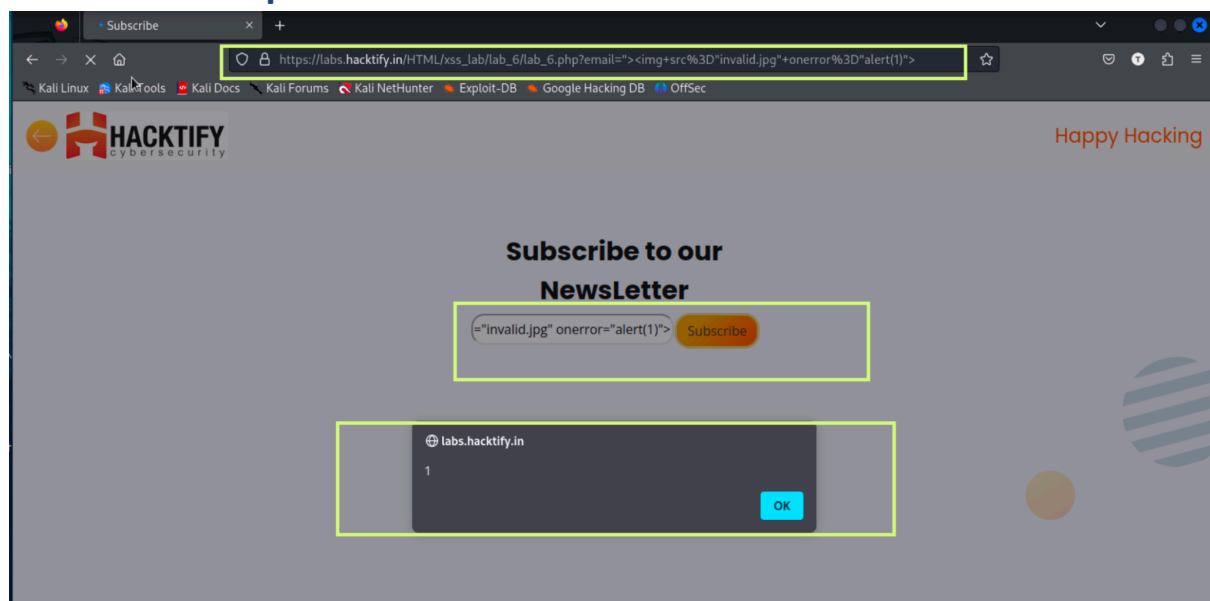
Suggested Countermeasures

To mitigate vulnerabilities exposed by payloads like ">, it's crucial to implement comprehensive XSS protection strategies. Start with input sanitization to remove or neutralize potentially malicious content from user inputs. Employ output encoding to ensure that any input echoed back to the browser is rendered harmless. Use Content Security Policy (CSP) to restrict the sources from which scripts can be executed and to prevent inline scripts from running. Additionally, adopting modern web development frameworks and libraries that automatically handle these security concerns can significantly reduce the risk of XSS attacks. Regular security training for developers and periodic code reviews can also help identify and remediate vulnerabilities early, making these practices essential components of a robust web application security posture.

References

https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept



1.7. Encoding Is The Key?

Reference	Risk Rating
Encoding Is The Key?	Medium
Tools Used	
Firefox Browser	
Vulnerability Description	
<p>Payload: %3Cscript%3Ealert%281%29%3C%2Fscript%3E%40gmail.com</p> <p>The encoded string %3Cscript%3Ealert%281%29%3C%2Fscript%3E%40gmail.com represents an XSS payload that has been URL-encoded to bypass basic security filters and input validation checks. When decoded, it translates to <script>alert(1)</script>@gmail.com, embedding a JavaScript <script> tag that executes the alert(1) function. This technique is often used to inject malicious scripts into web applications that do not properly decode or sanitize user inputs before rendering them on a page. The payload aims to exploit Cross-Site Scripting (XSS) vulnerabilities by executing arbitrary JavaScript code in the context of the victim's browser, potentially leading to unauthorized actions such as cookie theft, session hijacking, and personal data exposure. The presence of @gmail.com suggests an attempt to embed the payload within an email context, possibly to bypass filters that specifically look for malicious content in email addresses or inputs expecting email formats. This underscores the importance of thoroughly sanitizing and validating all user inputs, including decoding encoded strings, to protect against XSS attacks.</p>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php	
Consequences of not Fixing the Issue	
The encoded XSS payload %3Cscript%3Ealert%281%29%3C%2Fscript%3E%40gmail.com represents a URL-encoded form of a script injection attempt, where %3C, %3E, and other similar sequences are URL-encoded equivalents of <, >, and other characters crucial for HTML syntax. When decoded, it	

translates to `<script>alert(1)</script>@gmail.com`, aiming to execute a JavaScript `alert(1)` function within a web page. This type of payload can bypass simple input validation mechanisms that do not decode input before checking for malicious patterns. By embedding such encoded scripts in inputs that are reflected back into web pages, attackers can execute arbitrary JavaScript in the context of the user's browser session. This technique underscores the sophistication of XSS attacks and the necessity for web applications to implement robust decoding and sanitization routines in their input processing logic to prevent the execution of malicious scripts.

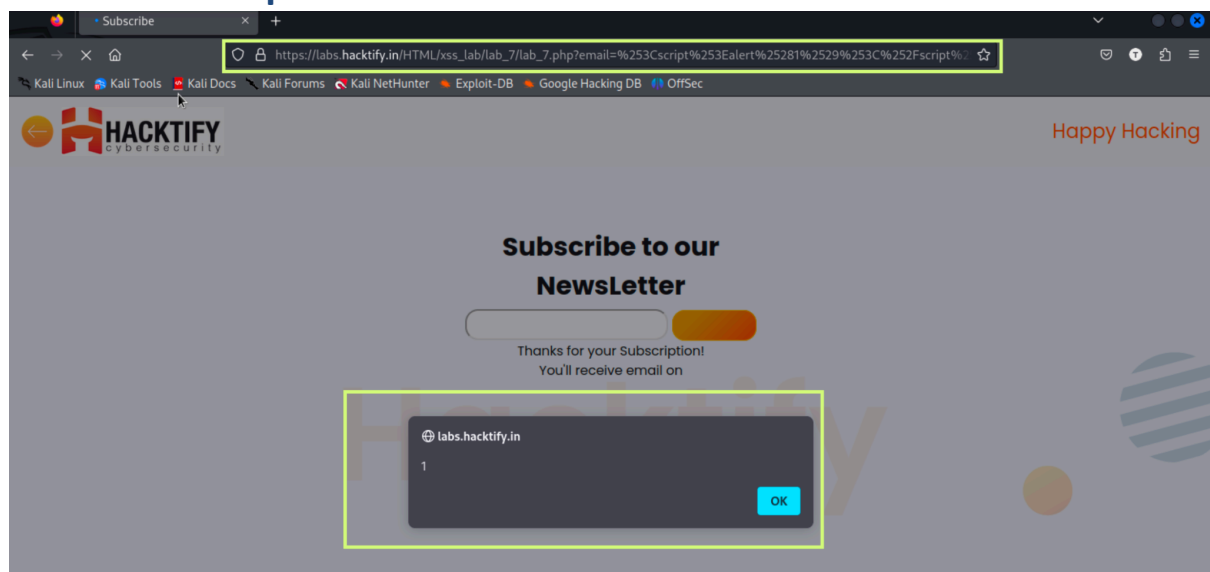
Suggested Countermeasures

The payload `%3Cscript%3Ealert%281%29%3C%2Fscript%3E%40gmail.com` uses URL encoding to disguise a script injection attempt as part of an email address. If not properly handled, it can enable attackers to execute arbitrary JavaScript in users' browsers, leading to security breaches such as data theft, session hijacking, and malicious redirection. To mitigate these risks, it's essential to sanitize and validate inputs, decode encoded inputs before processing, and implement Content Security Policy (CSP). Employing secure coding practices, regular security audits, and utilizing frameworks that automatically manage XSS protections are also key strategies in defending against such encoded XSS attacks, ensuring the safety of web applications and their users.

References

<https://medium.com/@zhyarr/double-url-encoded-xss-c3f72ff32ea8>

Proof of Concept



1.8. XSS with File Upload (File Name)

Reference	Risk Rating
XSS with File Upload (File Name)	Low
Tools Used	
Firefox Browser	
Vulnerability Description	
Payload: <code>filename=">"test.txt"</code>	

This payload could be used to exploit a system where filenames or file paths are dynamically inserted into the HTML output without proper sanitization or encoding. For instance, if a web application displays user-uploaded file names directly in the HTML and an attacker uploads a file with the name ``, the browser would attempt to execute the JavaScript within the context of the page, leading to an XSS attack.

How It Was Discovered

Manual Analysis

Vulnerable URLs

https://labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php

Consequences of not Fixing the Issue

The consequences of such an attack can vary from minor nuisances to significant security breaches, including stealing cookies, session tokens, or other sensitive information from users, redirecting users to malicious sites, or even performing actions on behalf of the users without their consent.

Suggested Countermeasures

To mitigate this type of vulnerability, it is crucial to:

Sanitize and validate all user inputs, filenames included, to ensure that potentially harmful characters or code snippets are neutralized or rejected.

Encode or escape output to prevent the browser from interpreting it as executable code rather than as plain text or data.

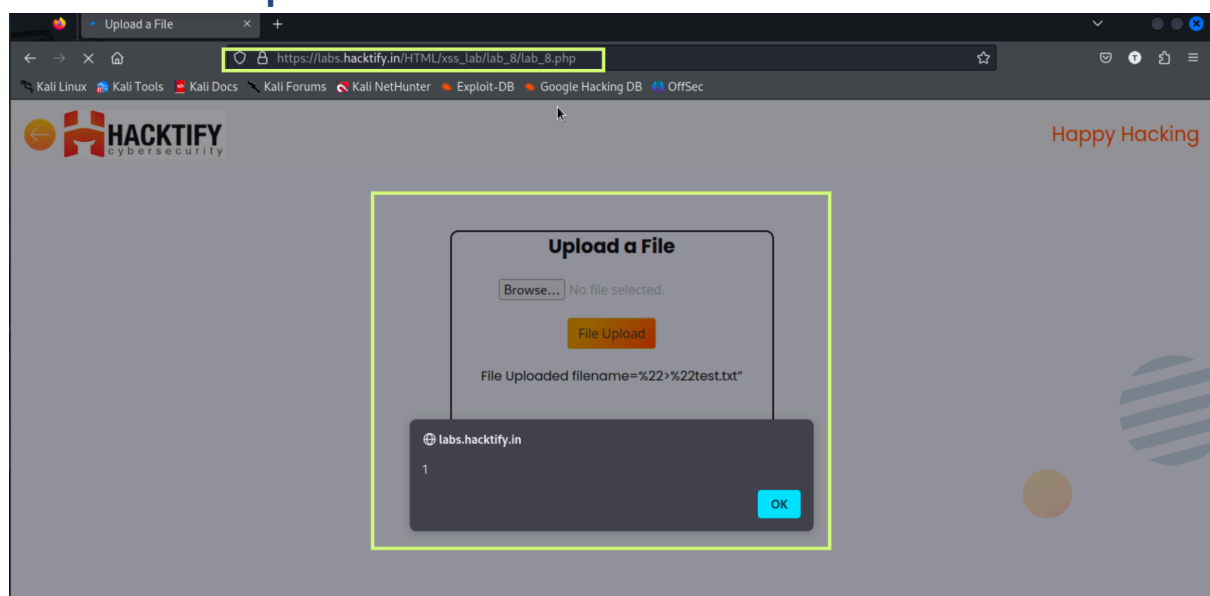
Adopt a Content Security Policy (CSP) to add an additional layer of protection by specifying which sources are valid for executing scripts, thus limiting the potential for XSS attacks.

Regularly conduct security reviews and vulnerability assessments to identify and rectify potential XSS vectors in existing applications.

References

<https://medium.com/@sarang6489/file-upload-xss-using-filename-f2f53e10033d>

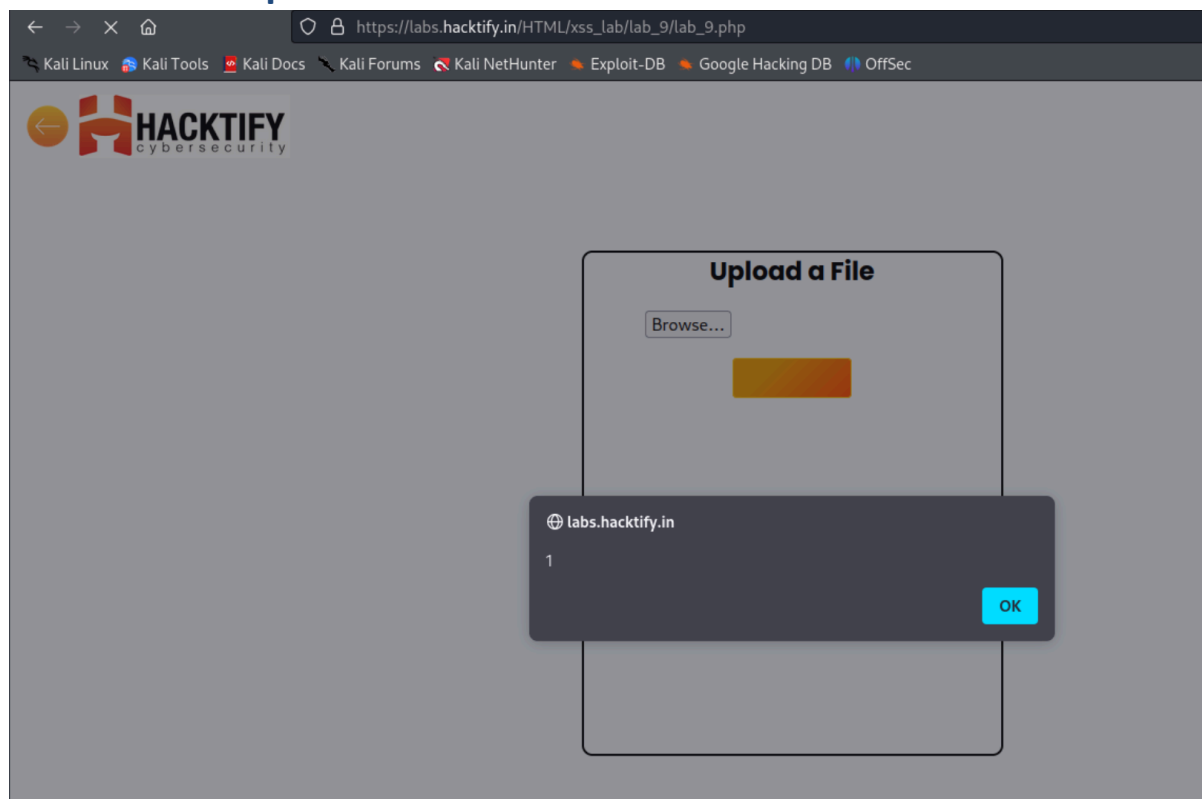
Proof of Concept



1.9. XSS with File Upload (File Content)

Reference	Risk Rating
XSS with File Upload (File Content)	Medium
Tools Used	
Firefox Browser	
Vulnerability Description	
<p>Payload:<script>alert(1)</script> <script>confirm(1)</script> "><script>alert(1)</script> "><script>confirm(1)</script> "></p> <p>In a typical file upload XSS attack scenario, an attacker would first upload a specially crafted file that contains malicious JavaScript code. They would then attempt to execute the JavaScript code by tricking a victim user into visiting a URL that references the uploaded file.</p> <p>When the victim user's browser loads the malicious file, it will execute the JavaScript code contained within the file. This can potentially allow the attacker to perform a variety of malicious actions, such as stealing the victim user's session cookies, redirecting the victim user to a malicious website, or even executing arbitrary system commands on the victim user's machine.</p>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
<p>Not fixing XSS vulnerabilities with file uploads can have serious consequences. Attackers can exploit these vulnerabilities to execute malicious scripts in the context of the victim's browser, leading to data breaches, theft of sensitive information, session hijacking, and potentially gaining unauthorized access to the system. This can damage the reputation of the organization, lead to legal consequences, and result in financial losses due to the compromise of user data and potential fines. It is essential to address these vulnerabilities promptly to maintain the security and integrity of the application and protect users' data.</p>	
Suggested Countermeasures	
<p>To mitigate XSS vulnerabilities with file uploads, consider the following countermeasures:</p> <p>Input Validation: Ensure that file uploads are strictly validated for type, size, and content. Only allow specific file types and reject any files that do not meet the criteria.</p> <p>Content Sanitization: Sanitize the content of uploaded files to remove any potentially malicious code. This can include stripping out JavaScript or other executable code from file contents.</p> <p>File Type Restrictions: Restrict the types of files that can be uploaded to reduce the risk of malicious files being accepted. For example, only allow image files (e.g., .jpg, .png) and reject executable files.</p> <p>Secure File Handling: Store uploaded files in a secure location, separate from the application's execution path. This prevents attackers from being able to execute uploaded files on the server.</p> <p>Content-Type Headers: Set appropriate Content-Type headers when serving uploaded files to ensure that browsers treat them as the intended file type, rather than executable code.</p>	
References	
https://medium.com/@ms-official5878/stored-xss-via-file-upload-svg-file-content-ba230c1448f6	

Proof of Concept



1.10. Stored Everywhere!

Reference	Risk Rating
Stored Everywhere!	Low
Tools Used	
Firefox Browser	
Vulnerability Description	
Payload: "><script>alert(1)</script> The XSS payload "><script>alert(1)</script>" is a simple example of a cross-site scripting attack. It works by breaking out of an HTML attribute context (e.g., an input value or a link) and injecting a <script> tag into the webpage. When the browser renders the page, it executes the JavaScript within the <script> tag, in this case, displaying an alert box with the number 1.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_10/lab_10.php	
Consequences of not Fixing the Issue	
If the Cross-Site Scripting (XSS) vulnerability that allows the injection of the <script>alert(1)</script> payload is not properly addressed and fixed, it can lead to a number of serious consequences.	

Unauthorized Access: An attacker could potentially use the XSS vulnerability to gain unauthorized access to sensitive information on the vulnerable web application.

Data Theft: An attacker could potentially steal sensitive data from the vulnerable web application, such as user credentials, session cookies, or other confidential information.

Session Hijacking: An attacker could potentially hijack a victim user's session by stealing their session cookies. This would allow the attacker to impersonate the victim user and perform actions on their behalf.

Redirection: An attacker could potentially redirect victim users to a malicious website, where they could be tricked into downloading malware, providing sensitive information, or performing other actions that could harm the victim users.

Suggested Countermeasures

To mitigate XSS vulnerabilities, such as the one demonstrated by the payload "><script>alert(1)</script>", consider the following countermeasures:

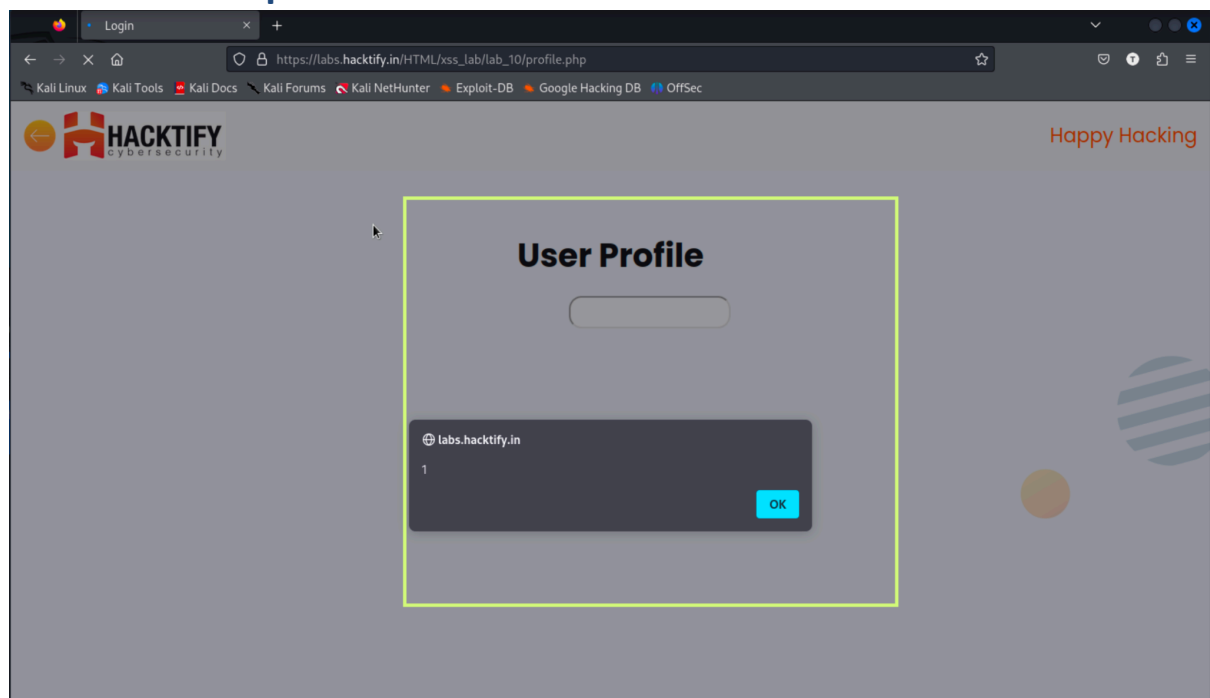
Input Validation: Validate user input on both the client and server sides to ensure that it conforms to expected formats and does not contain malicious content.

Output Encoding: Encode user-generated output before rendering it in the browser. This prevents potentially harmful characters from being interpreted as executable code.

References

<https://medium.com/@cyberw1ng/13-8-lab-stored-xss-into-anchor-href-attribute-with-double-quotes-html-encoded-2023-d9f698f43817>

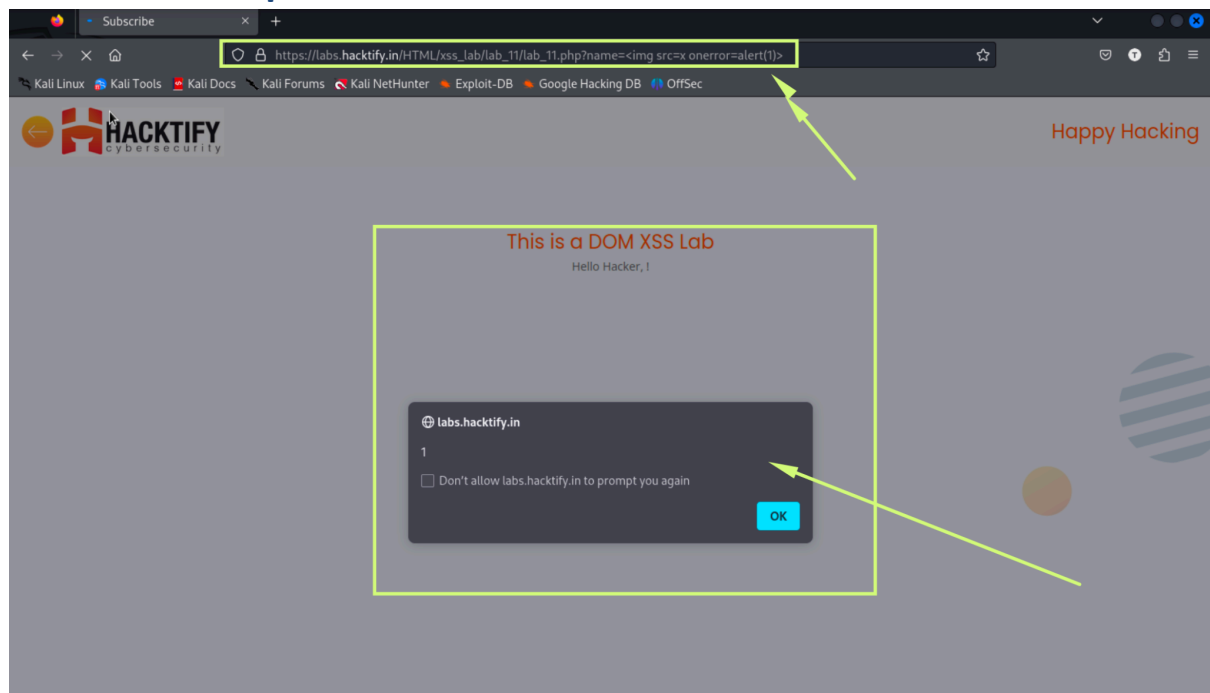
Proof of Concept



1.11. DOM's are love!

Reference	Risk Rating
DOM's are love!	High
Tools Used	
Firefox Browser	
Vulnerability Description	
<p>Payload: ?name=</p> <p>The XSS payload ?name= is an example of a cross-site scripting attack that injects an HTML image tag with a malicious JavaScript event handler. In this payload:</p> <p>?name= is a query parameter that could be part of a URL.</p> <p> is the malicious script injected into the application.</p> <p>The payload includes an tag with a non-existent image source (src=x). The 'onerror' attribute is used to execute JavaScript code (alert(1)) when the image fails to load, which is guaranteed since x is not a valid image source. When the browser renders this injected code, it triggers the 'onerror' event and executes the JavaScript, resulting in an alert box displaying the number 1.</p>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php	
Consequences of not Fixing the Issue	
<p>Not fixing the XSS vulnerability demonstrated by the payload ?name= can lead to several severe consequences:</p> <p>Data Theft: Attackers can use XSS attacks to steal sensitive information such as login credentials, personal data, or financial details from users.</p> <p>Session Hijacking: XSS can be exploited to steal session cookies, allowing attackers to impersonate users and gain unauthorized access to their accounts.</p> <p>Malware Distribution: Malicious scripts injected through XSS can be used to distribute malware to users' devices, leading to further security breaches.</p> <p>Phishing: XSS vulnerabilities can be leveraged to create convincing phishing attacks by injecting malicious content into trusted websites.</p>	
Suggested Countermeasures	
<p>For the specific payload ?name=, the same countermeasures apply. The payload attempts to inject an HTML image tag () with an 'onerror' attribute set to alert(1) JavaScript code. This can potentially allow the attacker to execute arbitrary JavaScript code in the context of the vulnerable web page. Therefore, it's important to implement the above mentioned countermeasures to help prevent such XSS attacks.</p>	
References	
https://medium.com/bugbountywriteup/write-up-dom-xss-in-innerhtml-sink-using-source-location-search-portswigger-academy-94c6691f89b0	

Proof of Concept



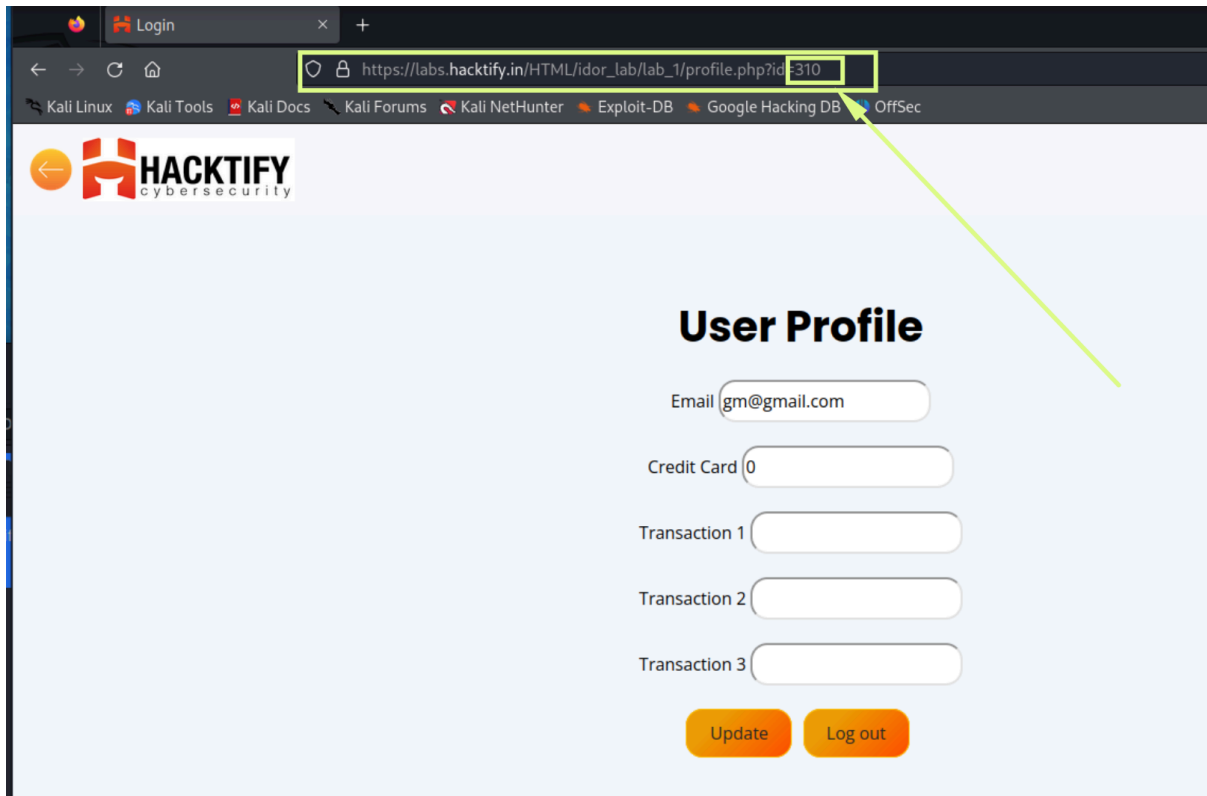
2. IDOR

2.1. Give Me My Amount!

Reference	Risk Rating
Give Me My Amount!	Low
Tools Used	
Firefox Browser	
Vulnerability Description	
<p>A Cross-Site Scripting (XSS) vulnerability can occur when a web application allows user input to be included in a URL without properly validating or sanitizing the input. This can potentially allow an attacker to inject malicious JavaScript code into the URL, which can then be executed by a victim user's browser when they visit the URL.</p> <p>For example, consider a web application that includes a user-supplied id parameter in a URL, such as https://example.com/user?id=123. If the web application does not properly validate or sanitize the id parameter, an attacker could potentially inject malicious JavaScript code into the parameter value, such as https://example.com/user?id=<script>alert(1)</script>.</p> <p>When the victim user's browser loads the URL with the malicious JavaScript code, it will execute the JavaScript code in the context of the vulnerable web page.</p>	
How It Was Discovered	
Manual Analysis	

Vulnerable URLs
https://labs.hacktify.in/HTML/idor_lab/lab_1/profile.php?id=310
Consequences of not Fixing the Issue
<p>The consequences of not fixing the id parameter in a URL XSS vulnerability can be severe, as it can allow attackers to inject malicious JavaScript code into a web application. This could allow attackers to:</p> <p>Steal sensitive information: The malicious JavaScript code could steal sensitive information, such as login credentials, credit card numbers, or other personal data, by sending it to the attacker's server.</p> <p>Redirect users to malicious websites: The malicious JavaScript code could redirect users to malicious websites that may contain malware, phishing scams, or other threats.</p>
Suggested Countermeasures
<p>The following countermeasures can be implemented to address the id parameter in URL XSS vulnerability:</p> <p>Input validation: Validate the id parameter to ensure that it does not contain malicious characters or code. This can be done using a variety of techniques, such as regular expressions or blacklists.</p> <p>Output encoding: Encode the id parameter before it is displayed in the web application. This can be done using a variety of techniques, such as HTML entity encoding or URL encoding.</p> <p>Use a web application firewall (WAF): A WAF can help to block malicious requests that attempt to exploit XSS vulnerabilities.</p> <p>Educate users about XSS attacks: Users should be aware of the risks of XSS attacks and should avoid clicking on suspicious links or opening untrusted files.</p>
References
https://medium.com/bugbountywriteup/xss-through-parameter-pollution-9a55da150ab2

Proof of Concept



2.2. Stop polluting my params!

Reference	Risk Rating
Stop polluting my params!	Medium
Tools Used	
Firefox Browser	
Vulnerability Description	
A URL ID parameter XSS vulnerability occurs when an attacker is able to inject malicious JavaScript code into a web application via the id parameter in a URL. This can allow the attacker to execute arbitrary code on the victim's computer, which could lead to a variety of security risks, such as:	
Stealing sensitive information	
Redirecting users to malicious websites	
Installing malware	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
<code>https://labs.hacktify.in/HTML/idor_lab/lab_2/profile.php?id=22</code>	

Consequences of not Fixing the Issue

Not fixing the id parameter in a URL XSS vulnerability can also have specific consequences for the affected web application. For example, if the web application is used for e-commerce, an attacker could use the vulnerability to steal credit card numbers or other sensitive financial information.

Suggested Countermeasures

The best way to prevent URL ID parameter XSS vulnerabilities is to validate and encode all user input. This can be done using a variety of techniques, such as:

Input validation: Validate the id parameter to ensure that it does not contain malicious characters or code.

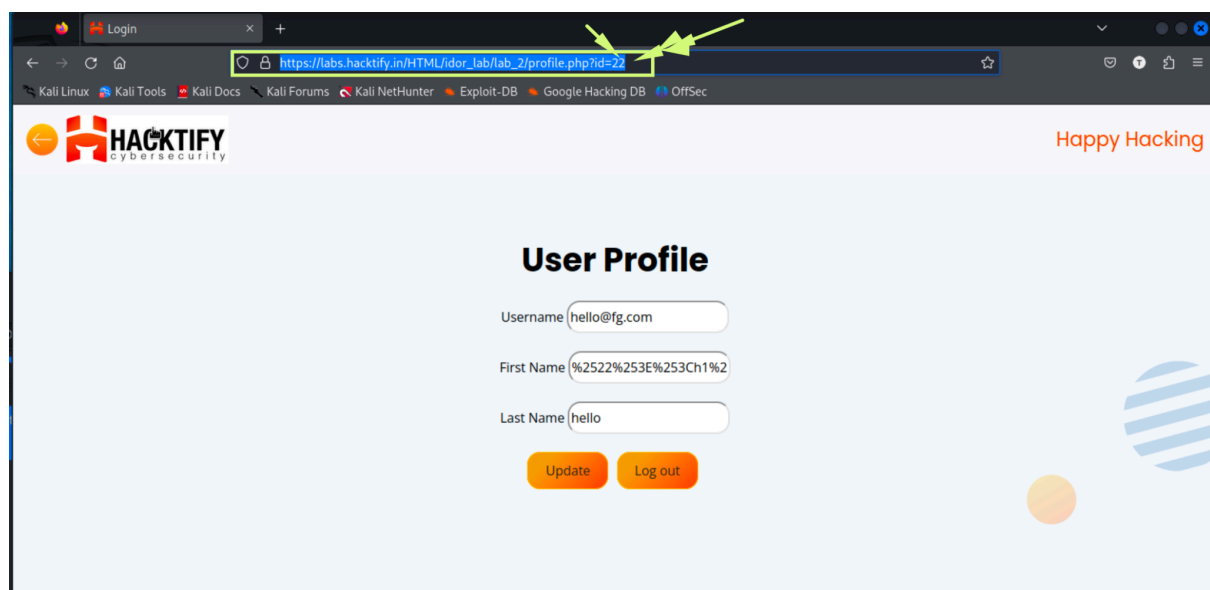
Output encoding: Encode the id parameter before it is displayed in the web application.

By implementing these measures, web developers can help to protect their applications from XSS attacks.

References

<https://medium.com/bugbountywriteup/xss-through-parameter-pollution-9a55da150ab2>

Proof of Concept



2.3. Someone changed my Password!

Reference	Risk Rating
Someone changed my Password!	High
Tools Used	
Firefox Browser	
Vulnerability Description	
An IDOR vulnerability can occur when a web application uses predictable or user-supplied input to directly access objects or resources without proper authorization checks. In the context of a username	

in a URL, this vulnerability can arise when the application allows users to access resources based solely on the username included in the URL, without validating whether the user has the necessary permissions to access those resources.

How It Was Discovered

Manual Analysis

Vulnerable URLs

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

Consequences of not Fixing the Issue

By exploiting this IDOR vulnerability, an attacker could gain unauthorized access to sensitive user data, such as personal information, financial records, or private messages. They could also potentially modify or delete data, or perform other actions that they should not be authorized to do.

Suggested Countermeasures

To mitigate IDOR vulnerabilities related to usernames in URLs, it is important to implement the following measures:

Validate the username parameter to ensure that the user has the necessary permissions to access the requested resource.

Use unique and unpredictable identifiers for objects and resources, rather than relying on user-supplied values.

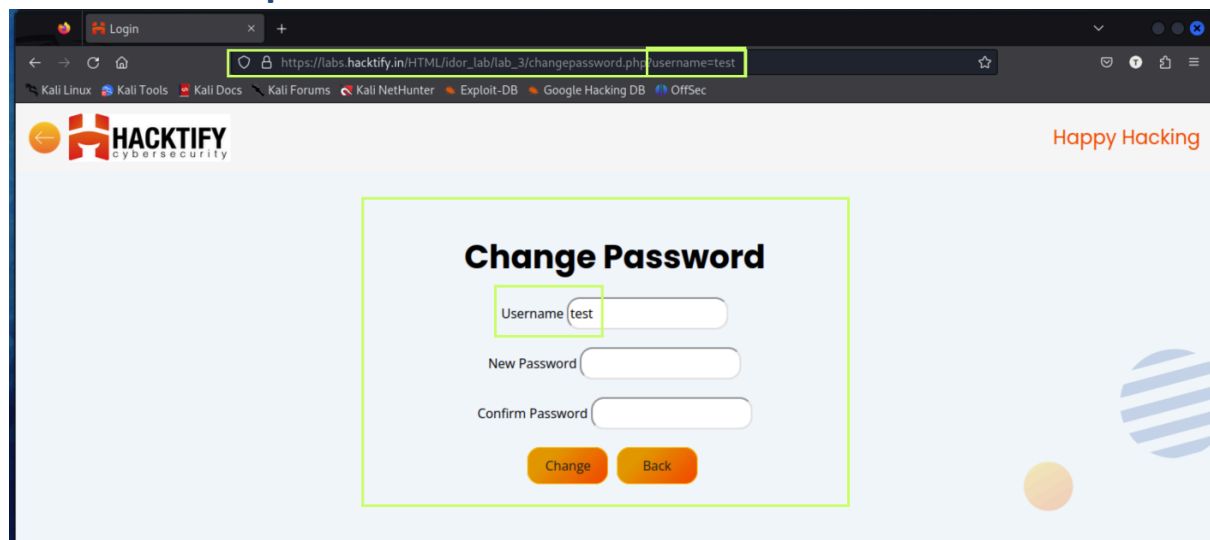
Implement role-based access controls to restrict access to specific resources based on the user's role or permissions.

Regularly review and test your application for potential IDOR vulnerabilities.

References

<https://portswigger.net/web-security/access-control/idor>

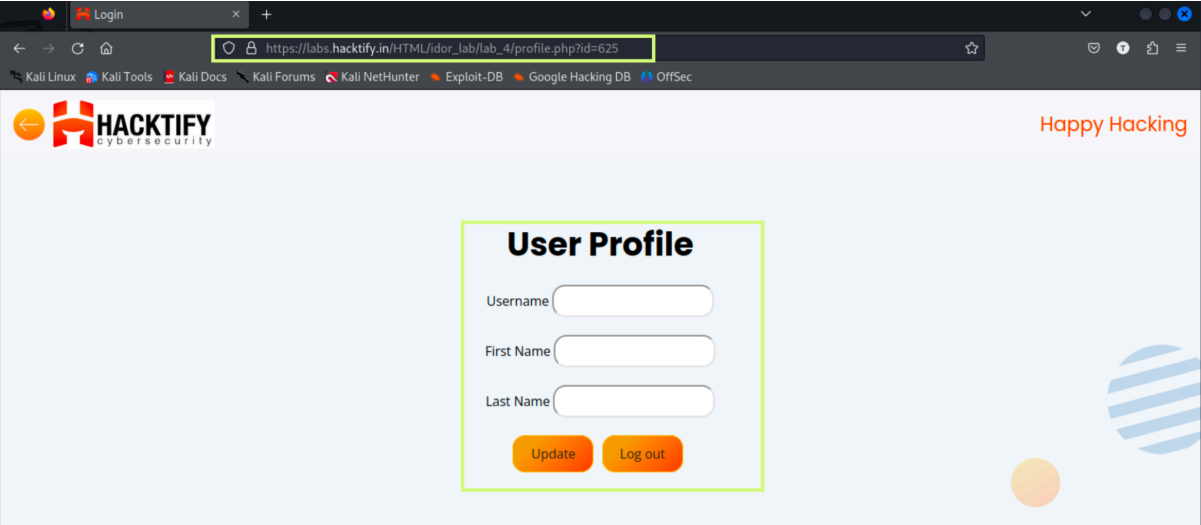
Proof of Concept



2.4. Change your methods!

Reference	Risk Rating
Change your methods!	Medium
Tools Used	
Firefox Browser	
Vulnerability Description	
<p>An IDOR vulnerability can occur when a web application uses predictable or user-supplied input to directly access objects or resources without proper authorization checks. In the context of a POST request, this vulnerability can arise when the application allows users to modify or delete resources based on user-supplied input, without validating whether the user has the necessary permissions to perform those actions.</p>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/idor_lab/lab_4/profile.php?id=625	
Consequences of not Fixing the Issue	
<p>By exploiting this IDOR vulnerability, an attacker could delete or modify any post on the website, regardless of their permissions. This could have a significant impact on the integrity and availability of the application's data.</p>	
Suggested Countermeasures	
<p>To mitigate IDOR vulnerabilities related to POST requests, it is important to implement the following measures:</p> <p>Validate the <code>post_id</code> parameter in the POST request body to ensure that the user has the necessary permissions to delete or modify the specified post.</p> <p>Use unique and unpredictable identifiers for objects and resources, rather than relying on user-supplied values.</p> <p>Implement role-based access controls to restrict access to specific resources based on the user's role or permissions.</p> <p>Regularly review and test your application for potential IDOR vulnerabilities..</p>	
References	
https://portswigger.net/web-security/idor	

Proof of Concept



The screenshot shows a web browser window with a single tab titled "Login". The address bar displays the URL `https://labs.hacktify.in/HTML/idor_lab/lab_4/profile.php?id=625`. The browser's bookmark bar includes links to "Kali Linux", "Kali Tools", "Kali Docs", "Kali Forums", "Kali NetHunter", "Exploit-DB", "Google Hacking DB", and "OffSec".

The website header features the "HACKTIFY cybersecurity" logo on the left and the text "Happy Hacking" on the right. The main content area is light blue and contains a "User Profile" form highlighted by a green border. The form includes three input fields for "Username", "First Name", and "Last Name", and two orange buttons at the bottom: "Update" and "Log out".

Decorative elements on the right side of the page include a blue and white striped circle and a solid orange circle.