This lab demonstrates a \*\*Reflected Cross-Site Scripting (XSS)\*\* vulnerability in which user input is reflected back into the page's HTML content without any encoding or sanitization. The objective is to inject a script into the page that will execute in the browser.

#### Steps to Solve the Lab

1. \*\*Understanding the Scenario\*\*:

- The web application reflects user input directly into an HTML page without encoding it. This makes it vulnerable to XSS attacks, as any input supplied by the user will be interpreted as HTML or JavaScript.

- The user input is likely included in an HTML element, allowing you to inject and execute malicious scripts.

2. \*\*Identifying the Vulnerable Input\*\*:

- Navigate to the lab's page where user input is reflected (usually a search box, query parameter, or form submission).

- Test how the input is reflected in the HTML by entering a simple test string like:

```

test

```

- Observe where the input appears in the HTML response and if it is vulnerable to XSS.

3. \*\*Injecting the XSS Payload\*\*:

- Since the input is reflected without encoding, you can inject a basic XSS payload to execute JavaScript. Use a standard script injection like:

```html

<script>alert('XSS');</script>

```

- Inject this payload into the vulnerable input field or URL. For example, if the input is in the URL like:

```

https://example.com/search?query=example

```

Modify it to:

```

https://example.com/search?query=<script>alert('XSS');</script>

```

4. \*\*Triggering the XSS\*\*:

- When you submit the payload, the application reflects the malicious script back into the HTML page.

- The browser interprets and executes the script, triggering a JavaScript `alert('XSS')` pop-up.

- Once the pop-up appears, it confirms that the vulnerability has been successfully exploited.

5. \*\*Completing the Lab\*\*:

- After confirming the alert box, the lab should automatically detect that the XSS payload has been executed and mark the task as complete.

#### How the Attack Works:

- \*\*<script>alert('XSS')</script>\*\*: The `<script>` tag is executed by the browser because the application does not properly sanitize or encode the user input before reflecting it into the page. This allows an attacker to inject and run arbitrary JavaScript in the victim's browser.

#### Preventing Reflected XSS:

To prevent XSS, ensure that all user input is properly \*\*encoded\*\* before being reflected into HTML content. Specifically, user-supplied data should be HTML-encoded to prevent script injection. Additionally, use security features such as \*\*Content Security Policy (CSP)\*\* to mitigate the impact of XSS attacks.

#### Conclusion:

This lab demonstrates the risk of reflecting user input directly into HTML without proper encoding. By injecting a script into the vulnerable input, an attacker can execute arbitrary code in the victim's browser, which could lead to data theft, session hijacking, or other malicious actions. Proper input sanitization and output encoding are critical to preventing such vulnerabilities.

