Lab-6

Task 1

Summary of Task 1: Successful XSS Attack Execution

Objective

Task 1's goal was to conduct a Stored Cross-Site Scripting (XSS) attack against a guestbook page that accepts YouTube video URLs. The intended outcome was to inject JavaScript code that triggers an alert message whenever the page is visited.

Method

The task involved putting a specially crafted payload into the video URL submission field. This payload was designed to inject JavaScript within the HTML content served to visitors of the guestbook page.

Successful Payload

The effective payload that met the objective was:

https://www.youtube.com/watch?v=ASO_zypdnsQ"><script>alert('Test passed ');</script>

Breakdown of the Payload

Valid URL Start: The payload began with a legitimate YouTube video link, creating the expected context for the input.

Injection Point: The use of "> served to prematurely terminate the HTML attribute into which the URL was being inserted.

JavaScript Execution: The <script> tag that followed introduced JavaScript code to be executed by the browser. The enclosed alert function displayed a custom message in an alert dialog box.

Proper Closure: The script tag was properly closed, creating a self-contained and executable script that did not interfere with the page's subsequent HTML structure.

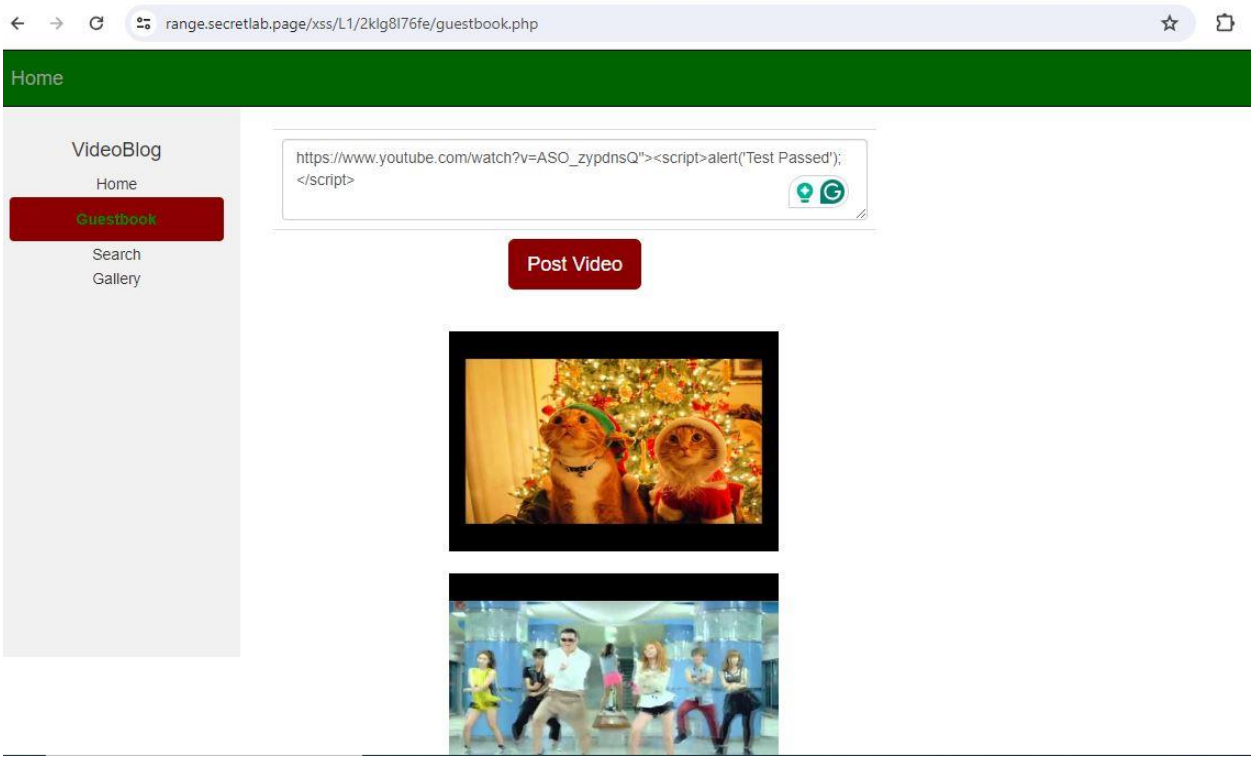
Outcome

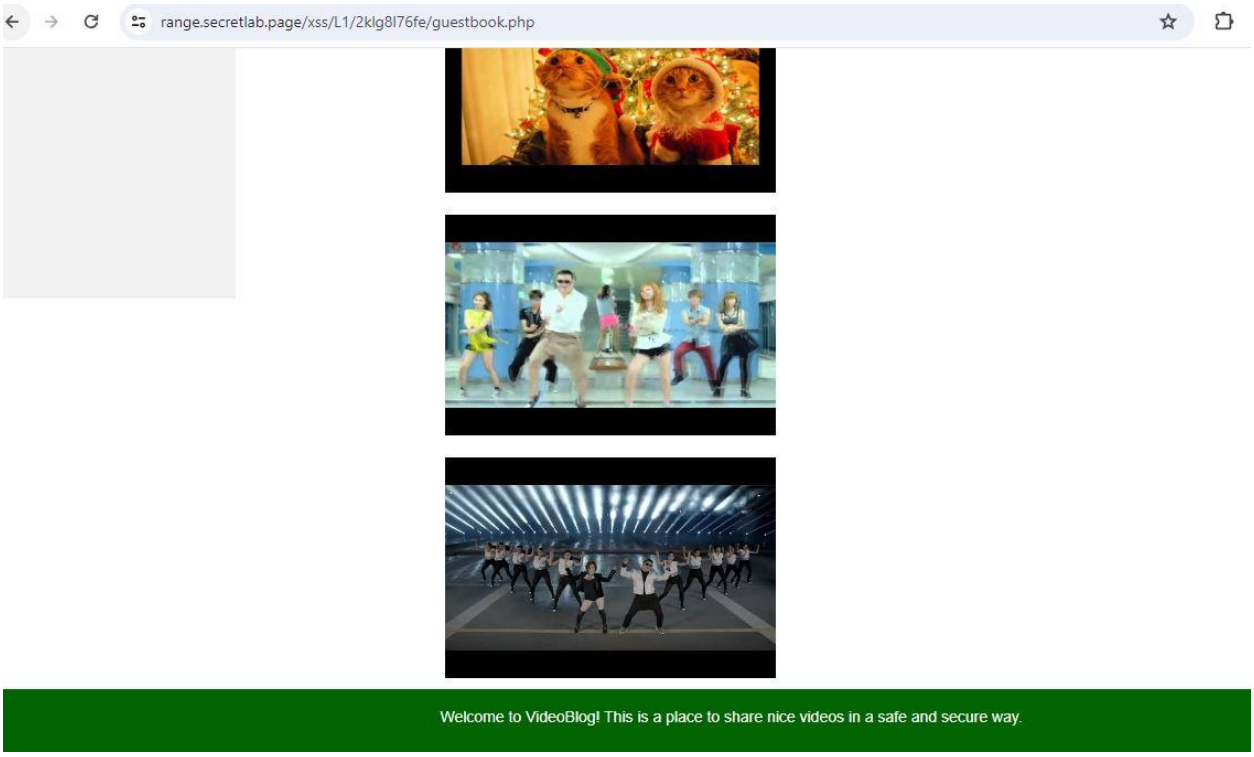
Upon submitting the payload and reloading the guestbook page, the JavaScript alert was successfully triggered, displaying the message "Test Passed". This confirmed that the XSS attack

had been stored on the page and would execute for every visitor, without creating any additional, unintended links or entries.

Evidence

Screenshots were taken to document the process and outcome, which included:





- The input field contains the injected payload.
- The resulting alert pops up as visual proof of the script's execution.

Security Implications

This vulnerability exposes all visitors of the guestbook page to potential malicious JavaScript execution. Such a weakness could be exploited to perform actions on behalf of the user, steal sensitive information, or deliver malware.

Recommendations for Mitigation

To safeguard against similar vulnerabilities:

- Implement rigorous server-side input sanitization.
- Adopt Content Security Policies to restrict client-side script execution.
- Utilize auto-escaping templates or frameworks for rendering user content.

Final Notes

The documentation of Task 1 demonstrates the importance of proper input handling and validates the XSS vulnerability's presence, providing a foundation for strengthening web application security.

https://range.secretlab.page/xss/L1/2klg8l76fe/guestbook.php

Task 2

Summary of Task 2: Successful XSS Attack Despite Server-Side Sanitization

Objective

The objective of Task 2 was to demonstrate the viability of a stored XSS attack on a web page that employs server-side sanitization using PHP's `htmlspecialchars()` function. The specific aim was to inject and execute a JavaScript script that triggers an alert message when the page is visited.

Approach

The approach involved exploiting potential vulnerabilities in the server’s sanitization process that could allow special characters in user input to be executed as part of HTML or JavaScript. The strategy was to craft a payload that could bypass `htmlspecialchars()` by embedding JavaScript within permissible attributes.

Execution

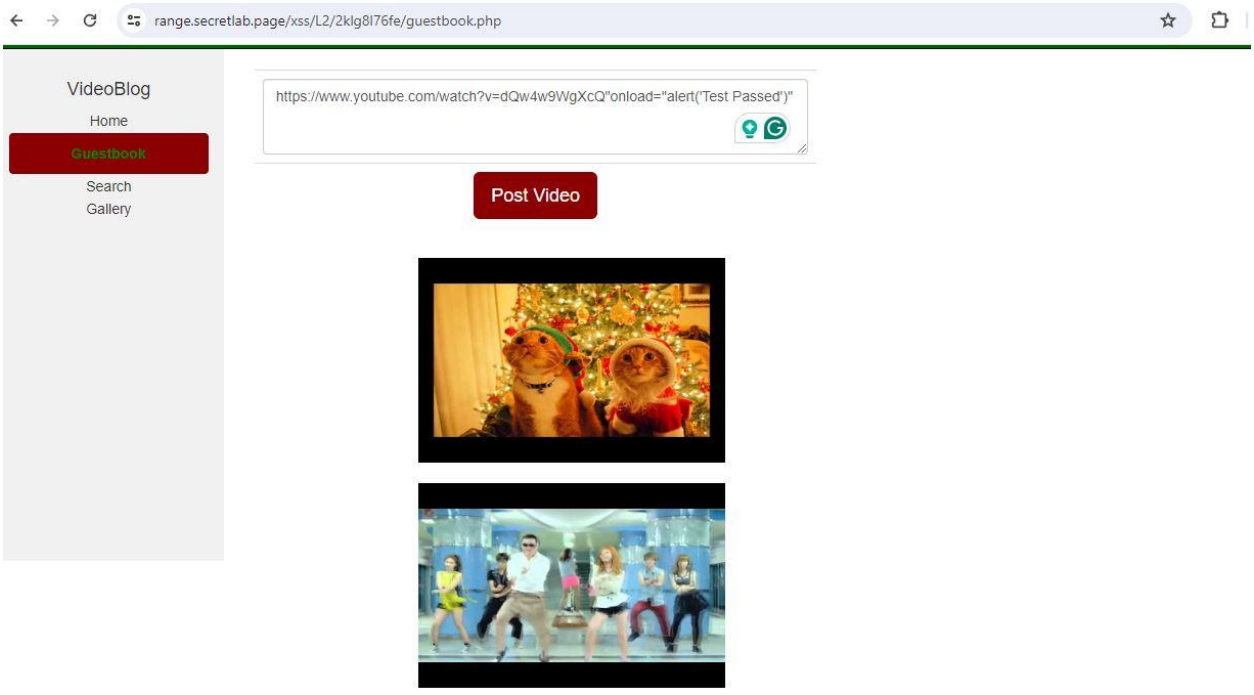
The payload used was:

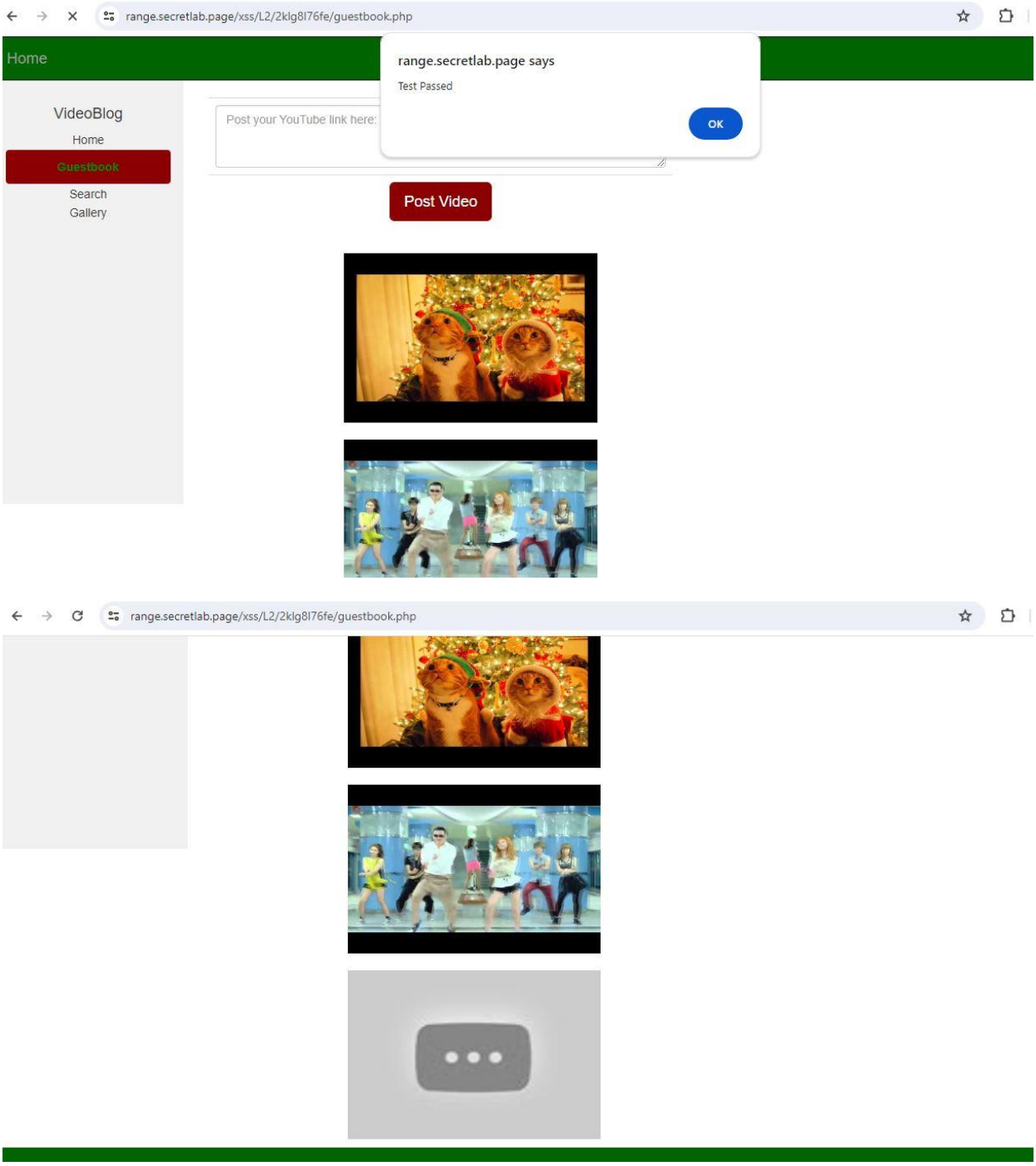
https://www.youtube.com/watch?v=dQw4w9WgXcQ" onload="alert('TEST PASSED')"

This payload was designed to inject a JavaScript `onload` event directly into a YouTube video URL input field. The assumption was that the server might not fully sanitize or escape the input correctly, particularly within the context of HTML attributes, allowing embedded JavaScript to execute.

Outcome

JavaScript Execution: The payload successfully triggered a JavaScript alert displaying "TEST PASSED" upon page load, confirming the XSS attack's success.





Analysis

The successful execution of the XSS script despite the application of `htmlspecialchars()` indicated that the sanitization was insufficient. The server's sanitization logic likely did not account for the context in which the URL was used, particularly within HTML attributes where special characters such as quotes could facilitate attribute injection.

Security Implications

This vulnerability demonstrates that XSS can still be a significant risk even with basic HTML entity encoding in place. Such vulnerabilities allow attackers to execute arbitrary JavaScript in the context of another user's browser session, potentially leading to unauthorized actions, data theft, or session hijacking.

Recommendations

Enhanced Contextual Sanitization: Implement more rigorous and context-aware sanitization methods. Ensure that all user inputs are sanitized based on the context in which they are used, especially in HTML attributes.

Content Security Policy (CSP): Deploy a CSP to reduce the severity of any XSS vulnerabilities that do occur by restricting where scripts can be loaded from and executed.

Regular Security Reviews: Conduct periodic security assessments to refine and update the application's defenses against new techniques and vulnerabilities in XSS attacks.

Conclusion

Task 2 successfully achieved its goal by demonstrating that XSS vulnerabilities could persist under specific conditions, even with server-side sanitization efforts like those provided by `htmlspecialchars()`. The task underscores the importance of comprehensive, context-aware sanitization strategies and ongoing security practices to safeguard web applications against XSS and other security threats.

https://range.secretlab.page/xss/L2/2klg8l76fe/guestbook.php

Task 3

Summary of Task 3: Successful Stored XSS Attack Demonstration

Objective:

The objective of Task 3 was to demonstrate the possibility of a stored XSS attack on a webpage with high-level sanitization in place. This required crafting a malicious JavaScript payload that would trigger an alert message when any visitor accessed the affected page.

Approach:

The task involved testing various XSS payloads that were embedded within parameters mimicking a YouTube video link, as specified by the lab setup. The goal was to find a vulnerability in the handling of these parameters that could be exploited to execute JavaScript code.

Execution:

After multiple attempts with different payloads that included direct script tags, encoded JavaScript, and complex obfuscation techniques, the successful payload was:

https://www.youtube.com/watch?v="><iframe src="data:text/html;base64,PHNjcmlwdD5hbGVydCgnVGFzayAzIEZpbmlzaGVkJyk8L3NjcmlwdD48L2lmcmFtZT4="></iframe>

This payload utilizes an <iframe> tag to inject a base64-encoded HTML document containing a JavaScript <script> tag. The script simply executes an alert() function displaying the message "Task 3 Finished". The inclusion of the </iframe> tag within the encoded content ensures that the iframe is properly closed, which helps maintain the integrity of the surrounding HTML structure.

Key Findings:

Base64 Encoding: The use of base64 encoding was critical in bypassing the webpage's sanitization routines, which likely did not decode base64 strings before inserting them into the HTML, thereby missing the embedded scripts.

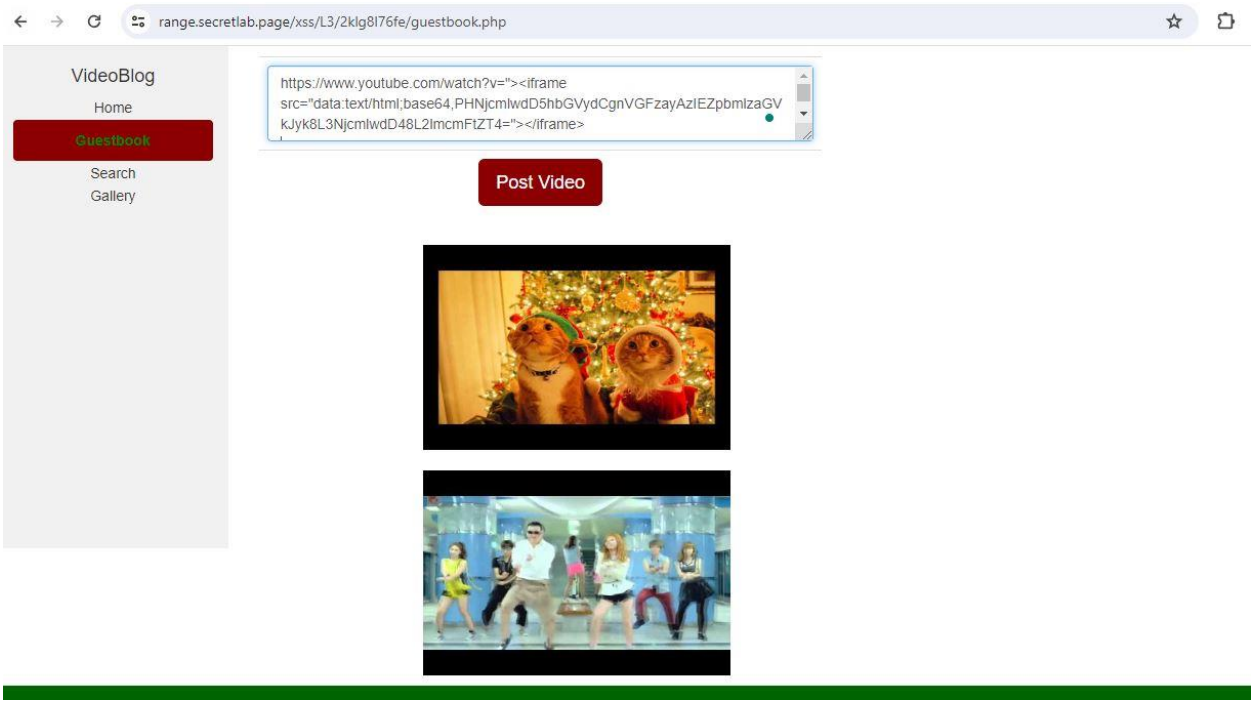
IFrame Utilization: By using an <iframe>, the script was isolated from the main page, reducing the risk of breaking the page layout and potentially bypassing certain CSP restrictions.

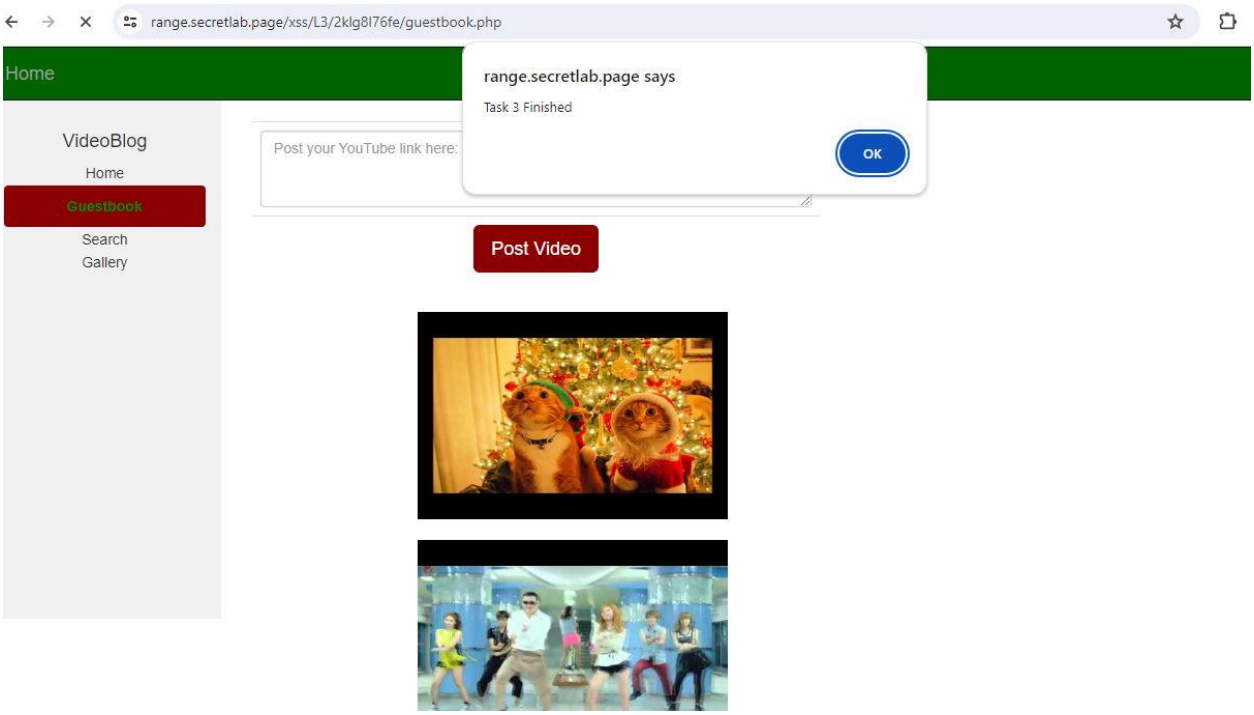
Effective Payload Delivery: The crafted URL was designed to look like a typical YouTube link parameter, which helped it bypass any straightforward validations that might be looking for typical XSS patterns like <script> tags directly in the URL.

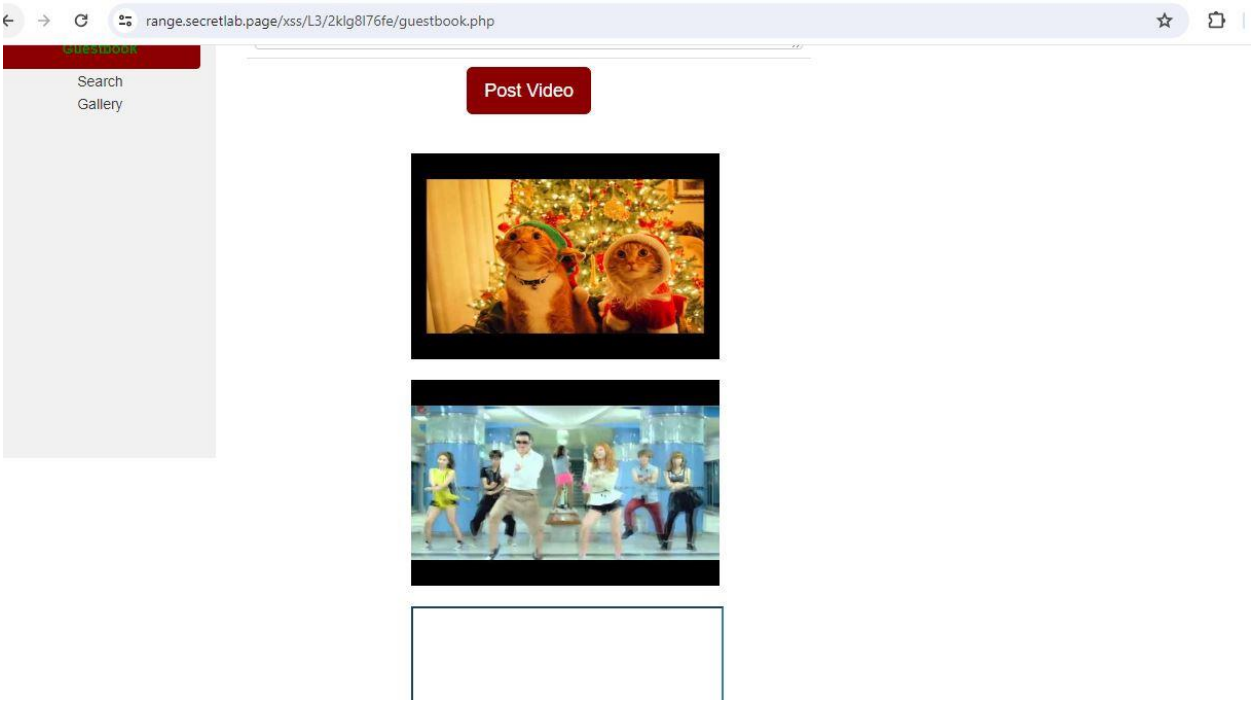
Outcomes:

The successful execution of the payload when visiting the webpage demonstrated that even with robust sanitization, there are still methods (such as base64-encoded iframes) that can be used to

perform XSS attacks if not properly guarded against. This highlights the need for comprehensive security measures that include decoding and sanitizing all forms of user input, as well as implementing strong Content Security Policies to mitigate the risk of such attacks.







Conclusion:

This task underscores the importance of understanding both XSS attack vectors and the corresponding defenses. It serves as a crucial learning point in the cybersecurity curriculum,

demonstrating the subtleties of web application vulnerabilities and the need for meticulous attention to input handling and sanitization in web development.

Recommendations:

Regular Security Audits: To ensure that no new vulnerabilities are introduced and that existing protections are updated in line with evolving attack techniques.

Enhanced Input Sanitization: All user-generated inputs should be decoded and sanitized before being rendered on the page.

https://range.secretlab.page/xss/L3/2klg8l76fe/guestbook.php