**Name: Bharath Karumudi**
**Lab: XSS Attack**

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

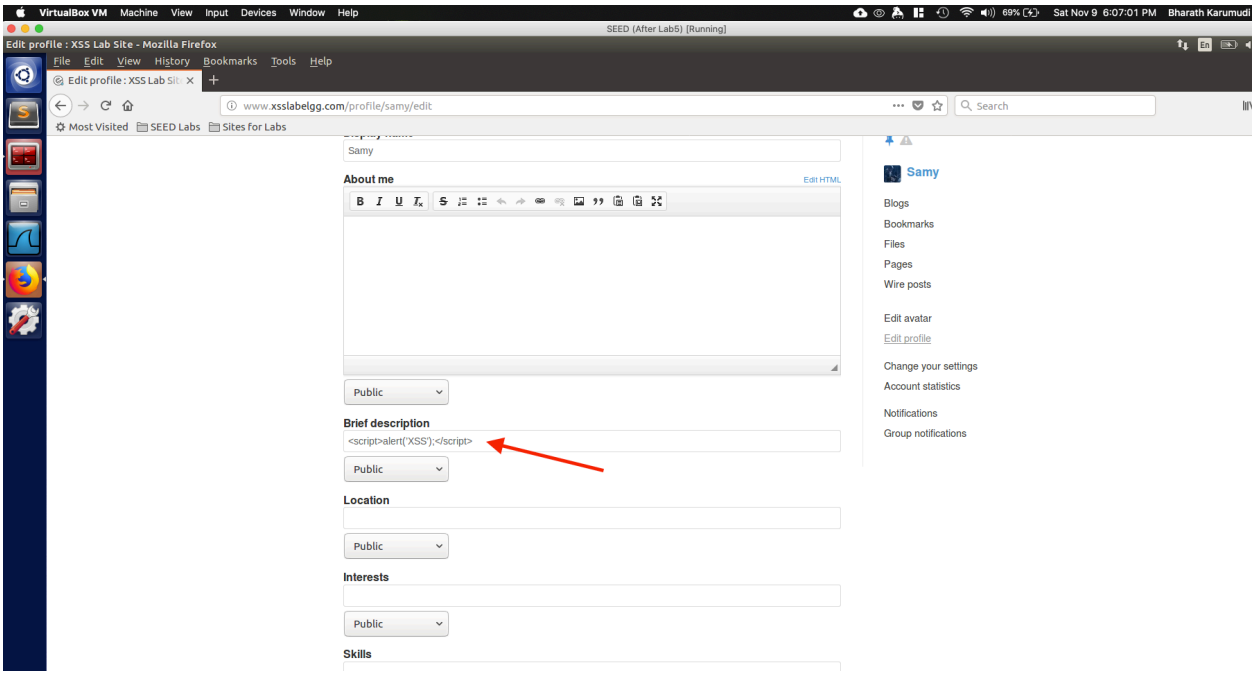



Fig1: Samy updating his profile with a test JavaScript.

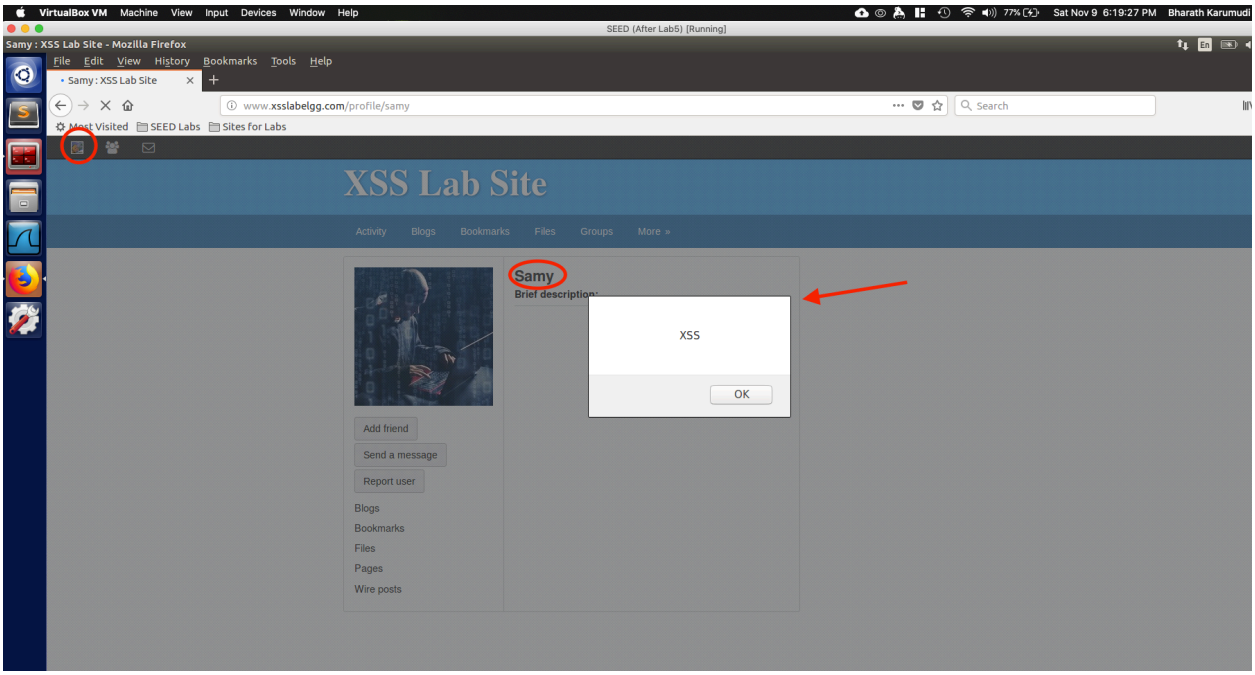



Fig2: Alice viewing the Samy profile and got the alert from JavaScript.

**Observation:** As Samy, I updated my profile “brief description” with a simple JavaScript which shows “XSS” in alert box. When Alice viewed the Samy profile, she received the alert box with “XSS” as message, as shown in Fig2.

**Explanation:** This is due to XSS attack, where the browser rendered the content of brief description as code instead of data and the code was executed by browser during page load and that is the reason, Alice got the alert box when she viewed the Samy’s profile page.

## Task 2: Posting a Malicious Message to Display Cookies

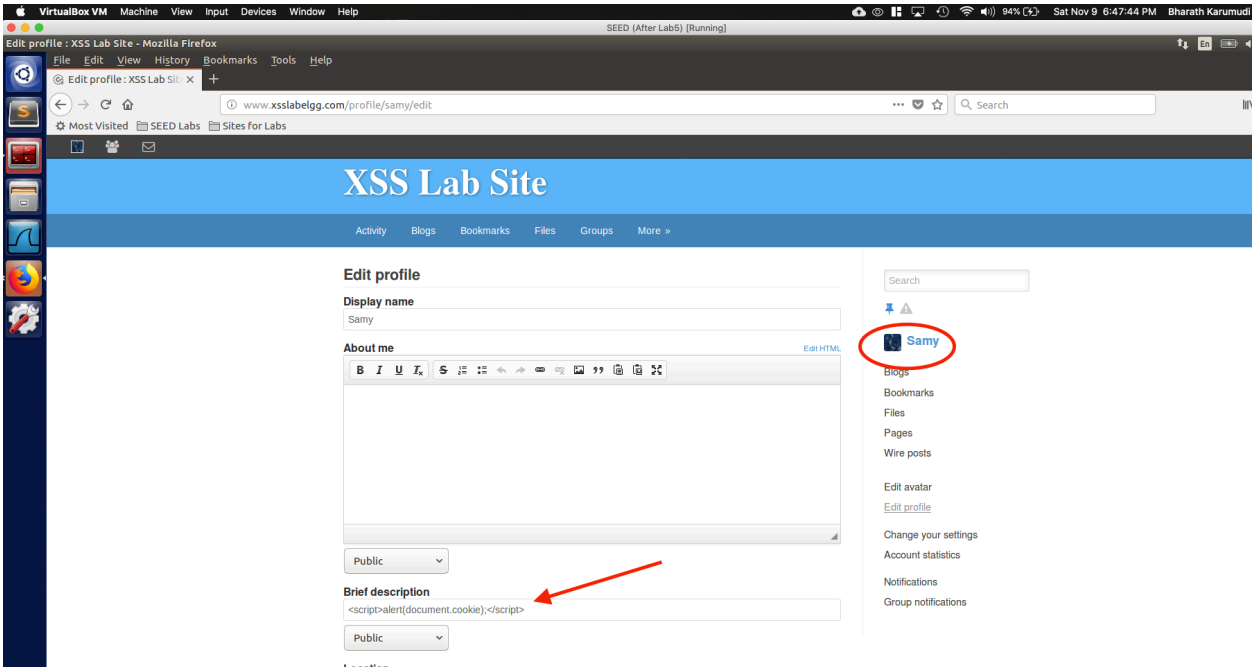



Fig: Samy updated his profile with malicious code to show cookie document

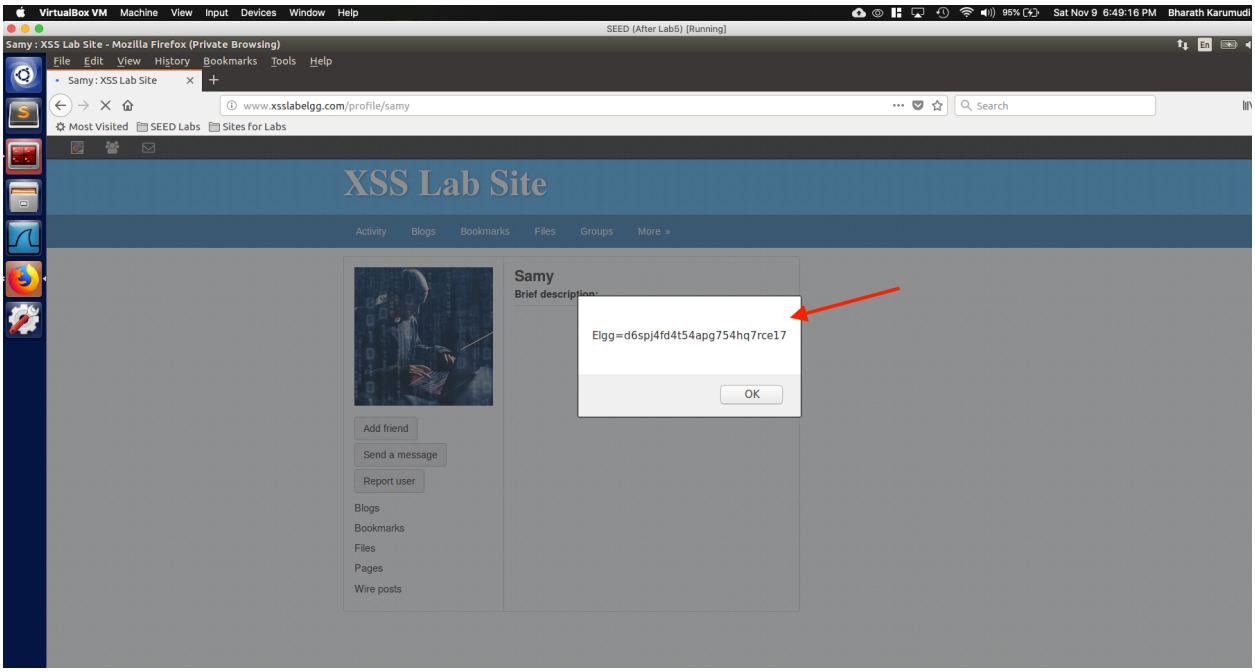



Fig: Alice seeing her cookie details when visited Samy profile

**Observation**: As Samy, I have updated the brief description in profile with a JavaScript that reads the current page cookie (document.cookie) and when Alice viewed the Samy profile, Alice got an Alert box with her own cookie information.

**Explanation**: This is due to XSS attack, where the browser rendered the content of brief description as code instead of data and the code was executed during page load. Unless CSRF, here the request was going from Elgg, so the malicious JavaScript helps us to show Alice Elgg page cookie details but only to her.

## Task 3: Stealing Cookies from the Victim's Machine

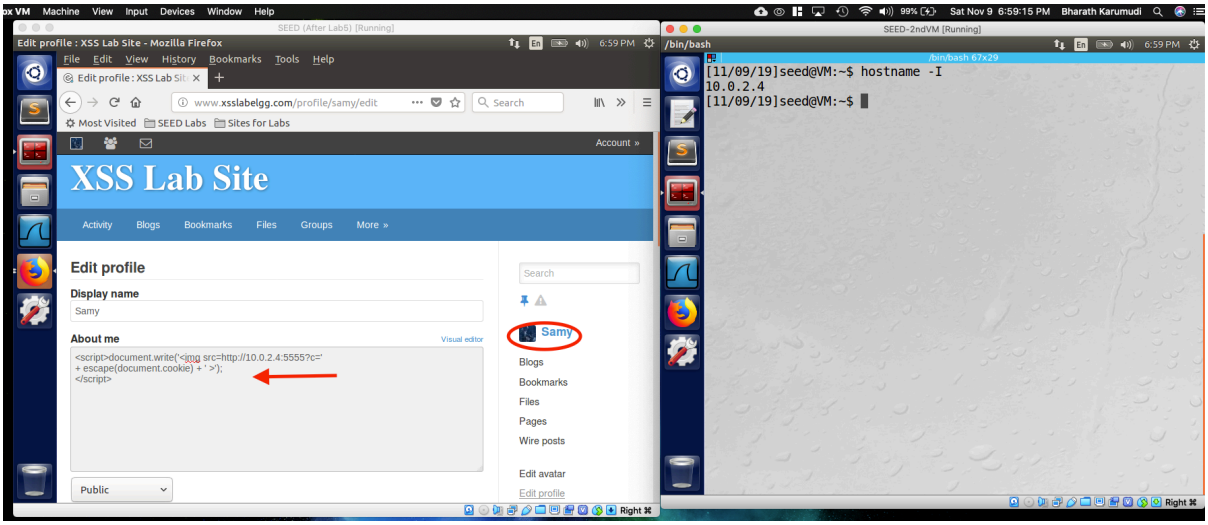



Fig: Samy updated JavaScript code that sends the cookie details to remote machine

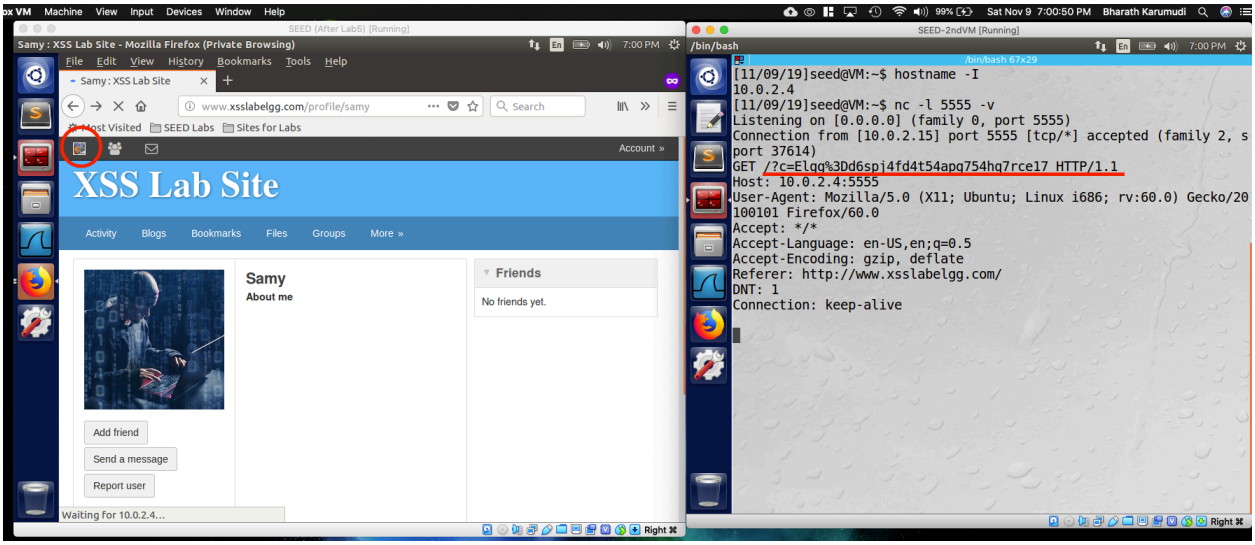


Fig: Alice cookie details are emitted on Samy terminal – remote machine

**Observation**: As Samy, I have updated my "About me" with a JavaScript that access the cookie and also sends a request to Samy machine. When Alice viewed the Samy profile, the details are sent to Samy's machine (remote) where he is listening using netcat for the traffic on port 5555.

**Explanation**: This is due to XSS attack, where the browser rendered the content of About me as code instead of data and the code was executed during page load. The JavaScript has fake image source as remote machine and so it sends a request to the remote machine who is listening using netcat.
The request also include the document.cookie, which is Alice Elgg's cookie details and when the request sent out for the fake img src (remote machine), Alice cookie details are also included in the request. Thus, the details are printed out on the Samy terminal as shown in second figure.

## Task 4: Becoming the Victim's Friend

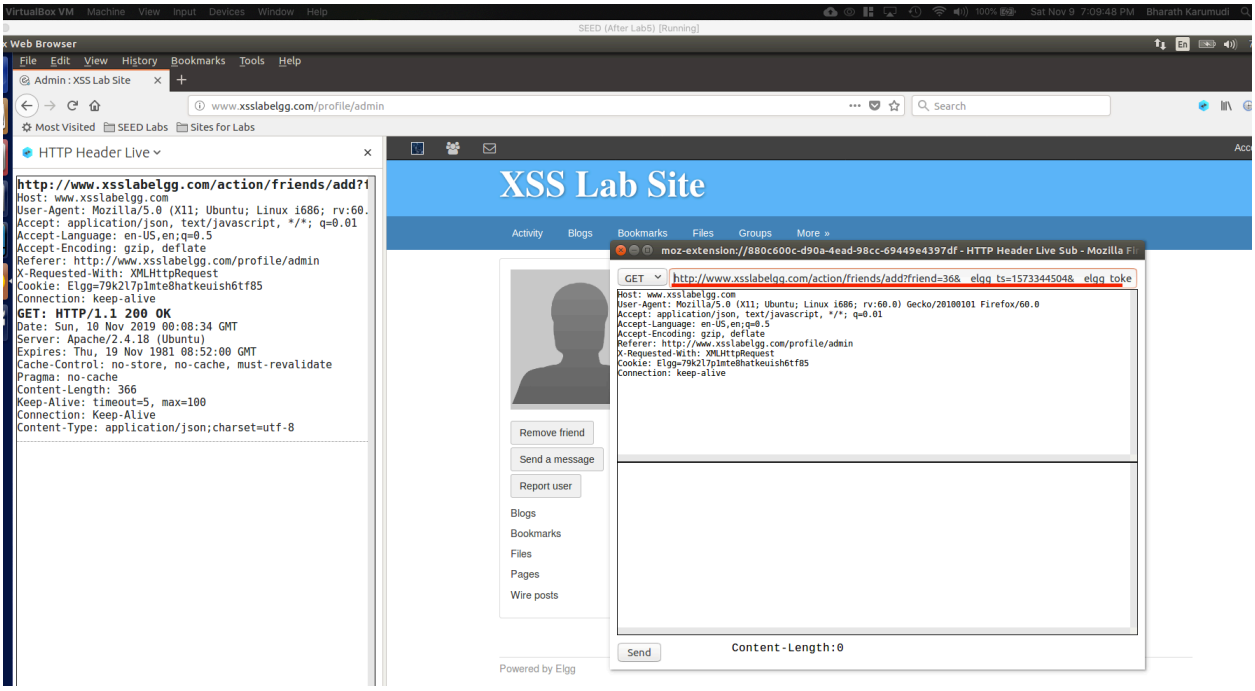



Fig: Examining the request headers for add friend

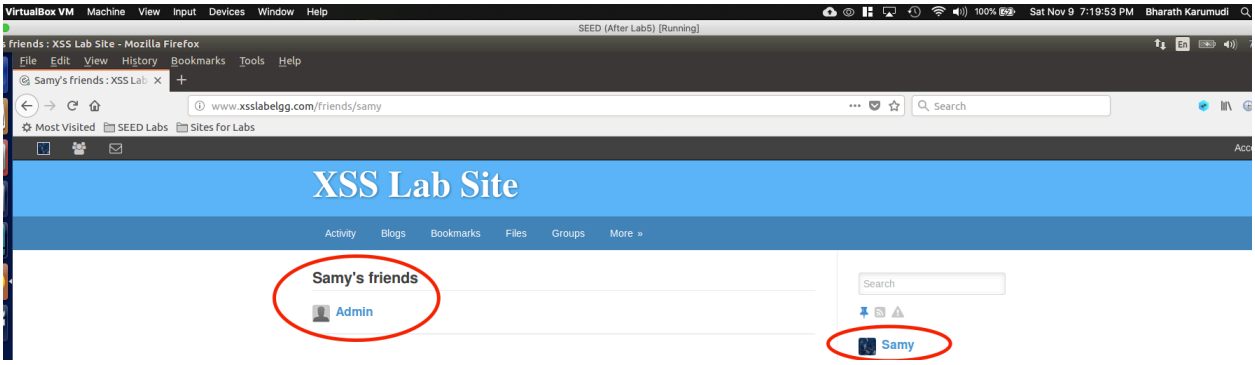



Fig: Showing Samy has only one friend at this time i.e. Admin

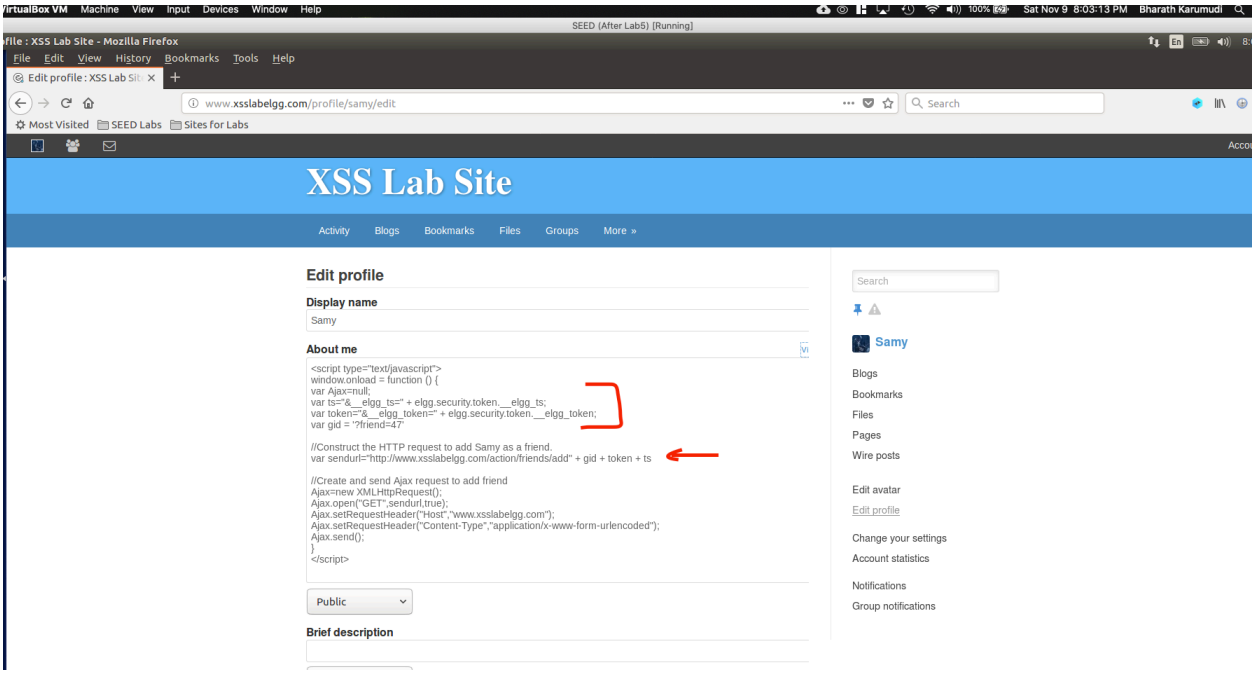



Fig: Adding the malicious code for XSS in Samy “About me”

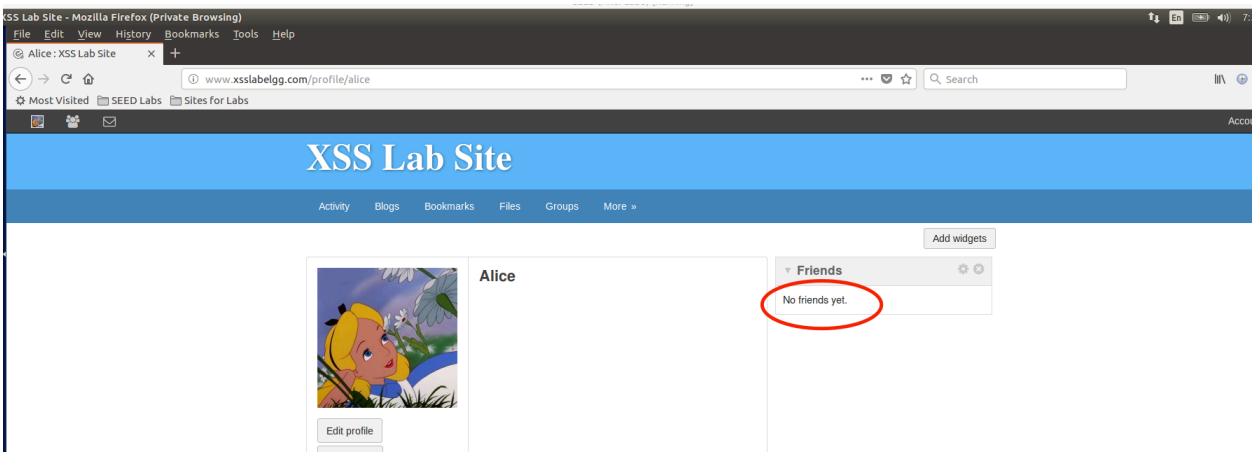



Fig: Alice has no friends at this time (before attack)

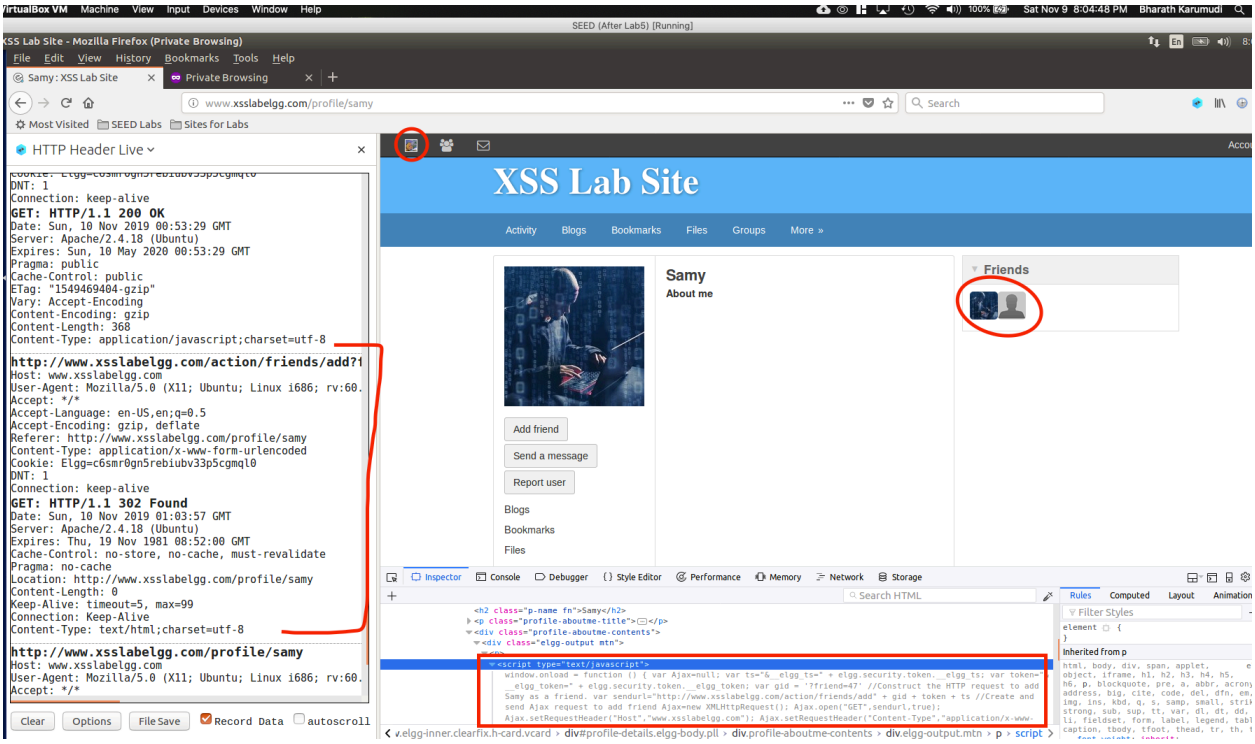



Fig: Alice viewed Samy profile and Samy was added to her list. Also, we can see HTTP requests and also script during that.

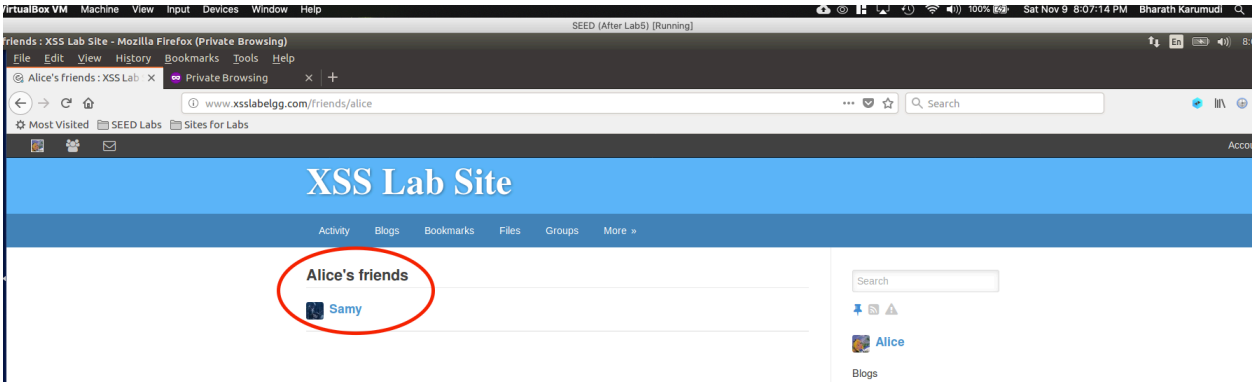



Fig: Showing Alice friends after XSS attack by Samy

**Observation**: As Samy, updated my profile with JavaScript that executes Add friend request with Samy, whoever view my profile page. When Alice, viewed the Samy profile, Samy was added to Alice friends without her knowledge.

**Explanation**: The JavaScript in the About me block was executed as code rather than data and the code has a Add friend request. So, when Alice viewed the Samy profile page, the code was executed and sent a request to Add Samy as her friend. Elgg server received the request as like a legit, because it also includes a valid cookie, timestamp and token – because the request is initiated from the Alice Elgg page directly, we can get all these attributes from Alice page. This is XSS attack.

**Question1:** The lines marked with 1 and 2 are needed to defend the CSRF countermeasure. Because, Elgg has CSRF defense enabled which validates the timestamp and session token. If the request comes with invalid tokens, the request will be rejected. So, we need to pass the correct timestamp and token to pass our requests.

**Question2**: No, because the Editor mode adds the additional tags to the content and the code will not be executed as expected. Thus, the attack fails.

## Task 5: Modifying the Victim's Profile

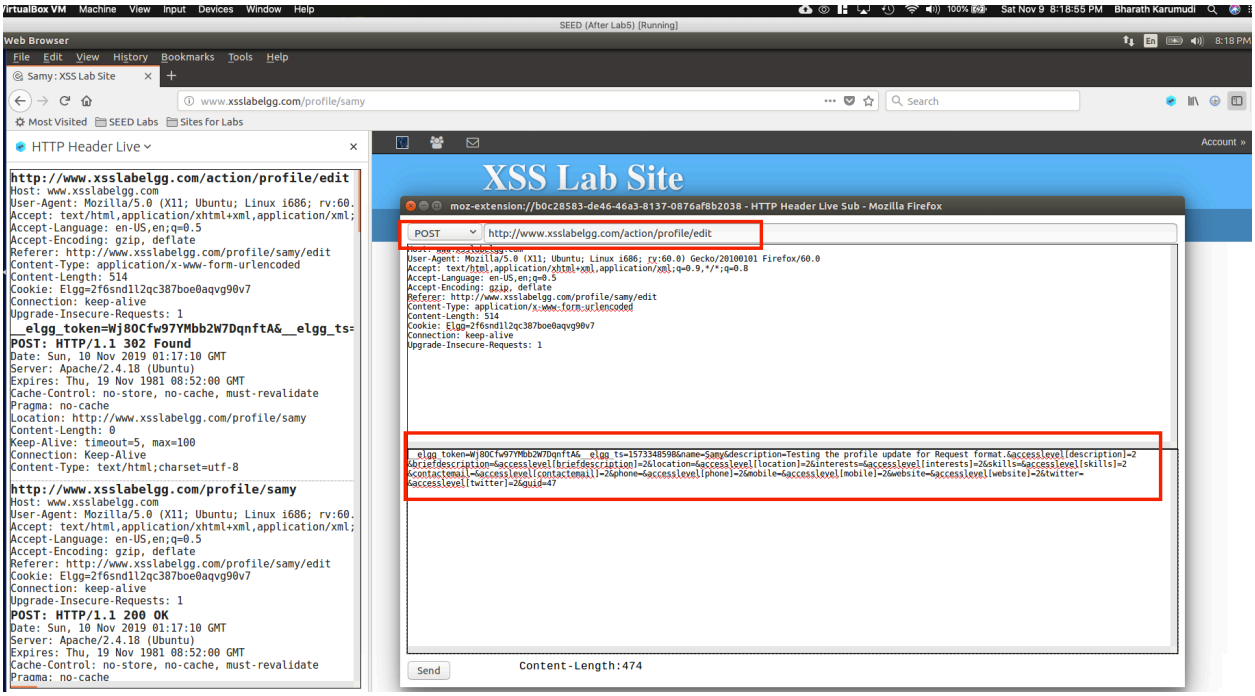



Fig: Examining the HTTP request for Profile edit

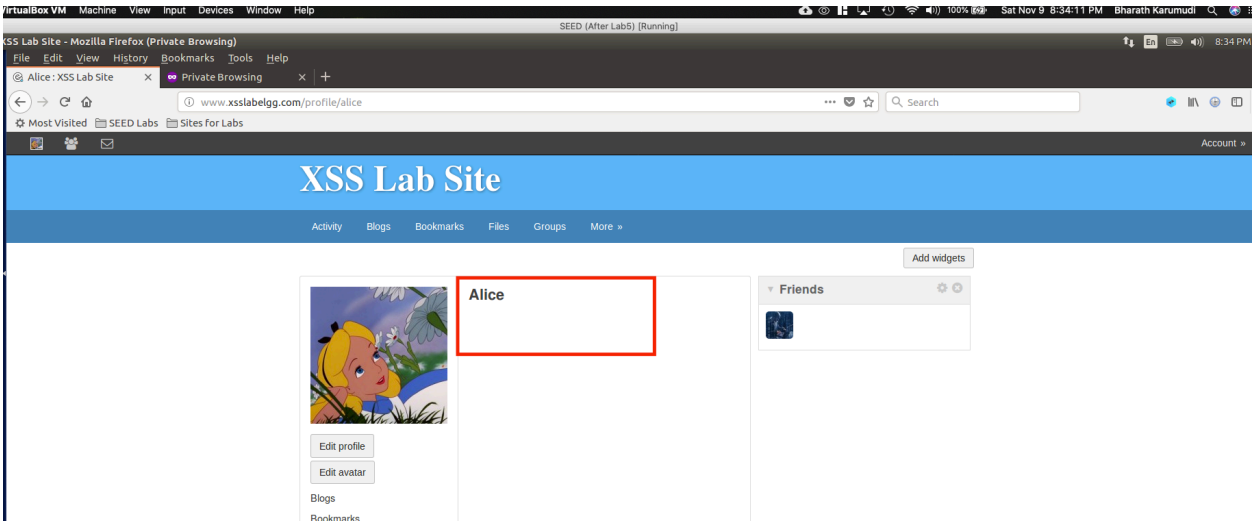



Fig: Showing Alice has no profile descriptions before XSS attack

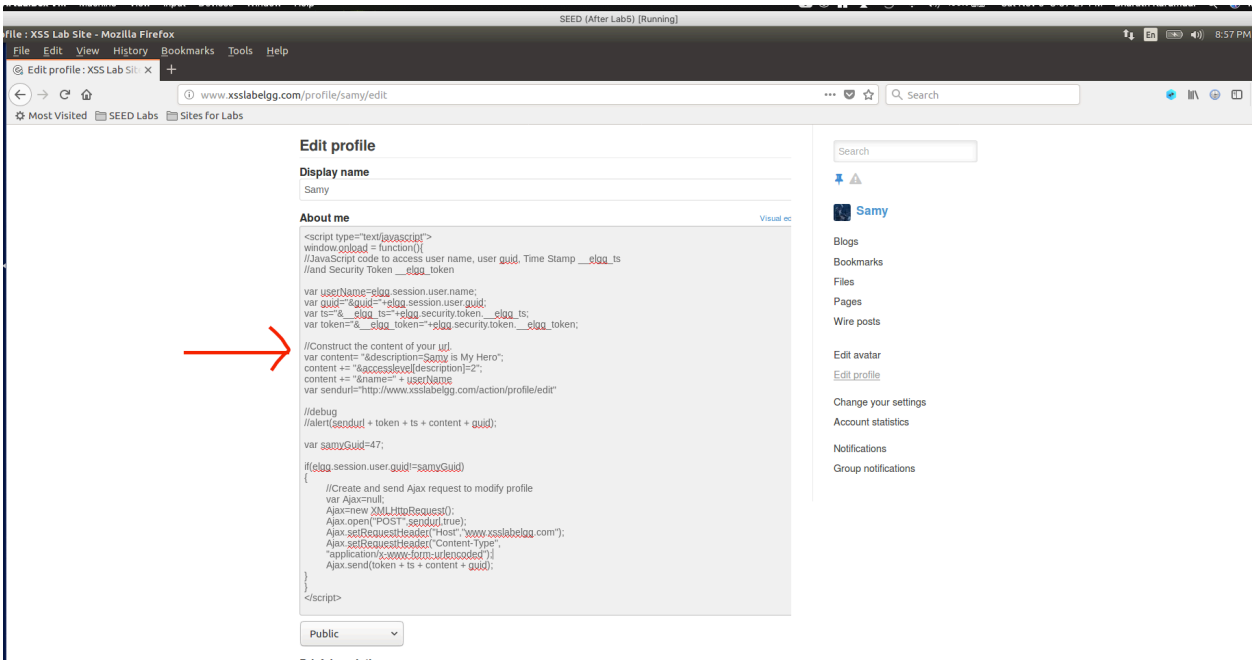



Fig: Samy Added the code to perform the XSS attack

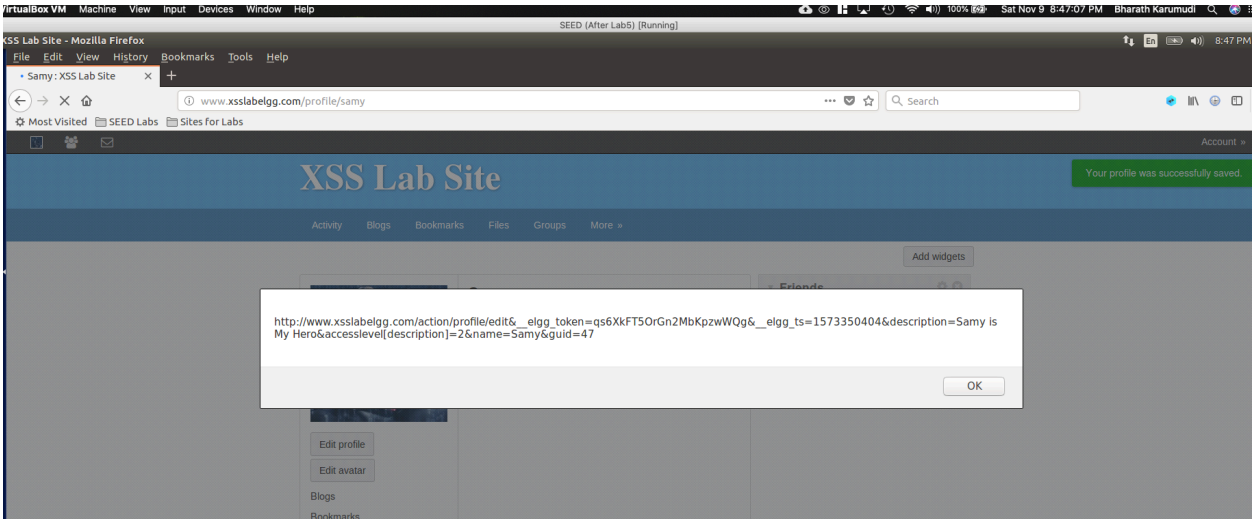



Fig: Samy debugging, to see the code is rendering the request correctly

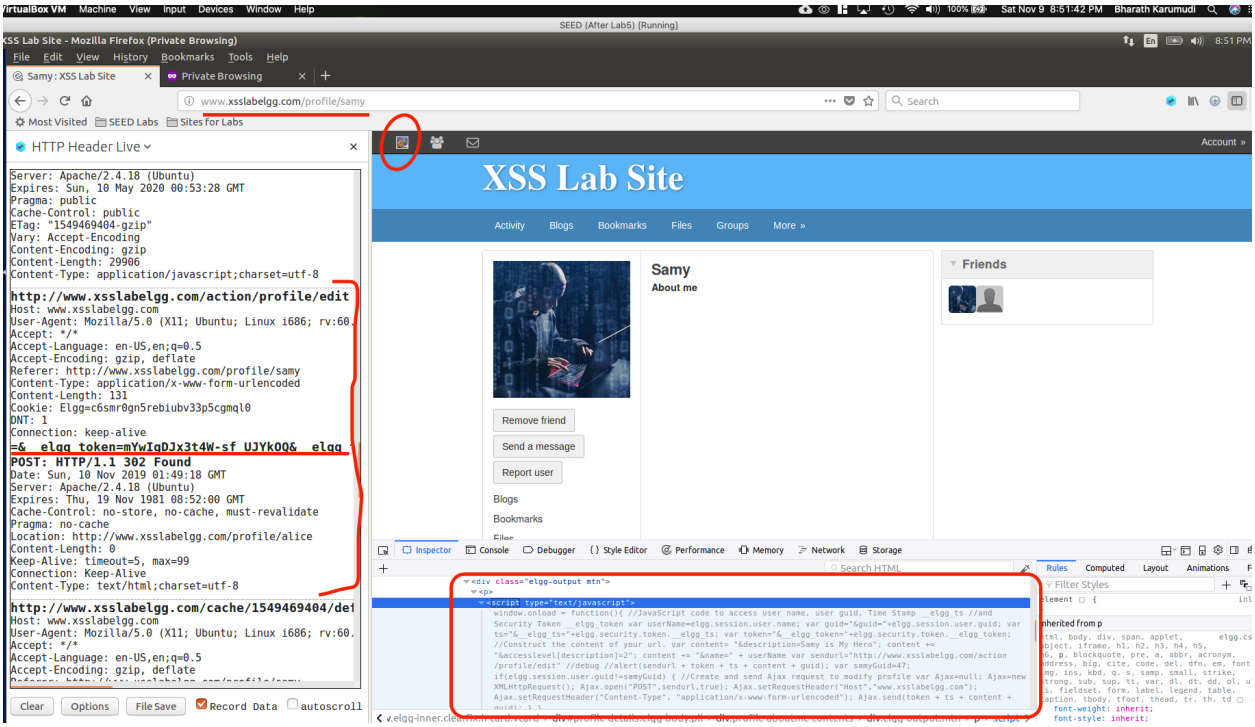



Fig: Alice viewed the Samy profile and XSS attack occurred – Also HTTP and page source code can be seen.

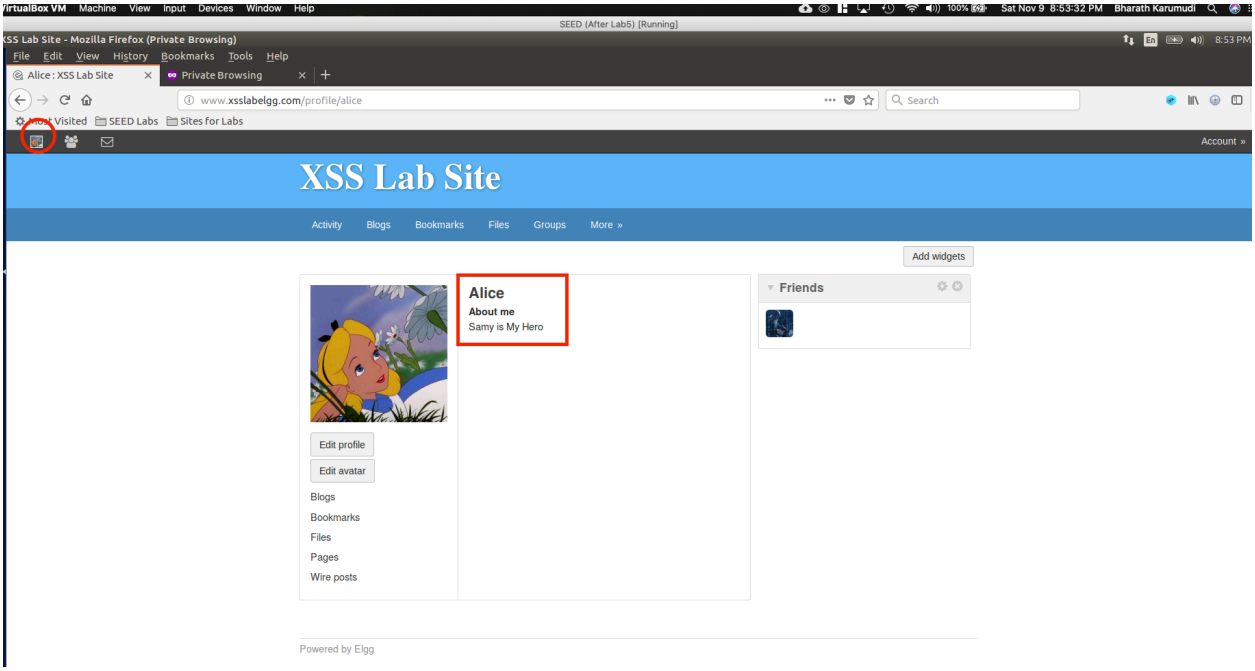


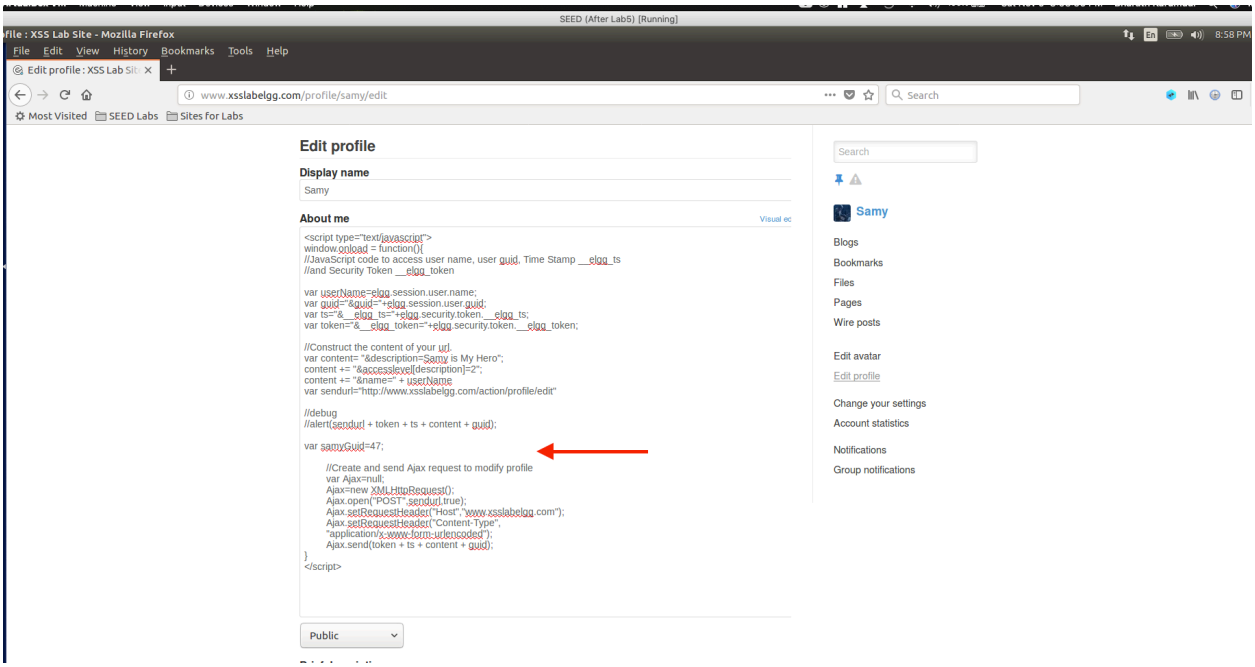

Fig: Alice profile description was updated – XSS Attack by Samy

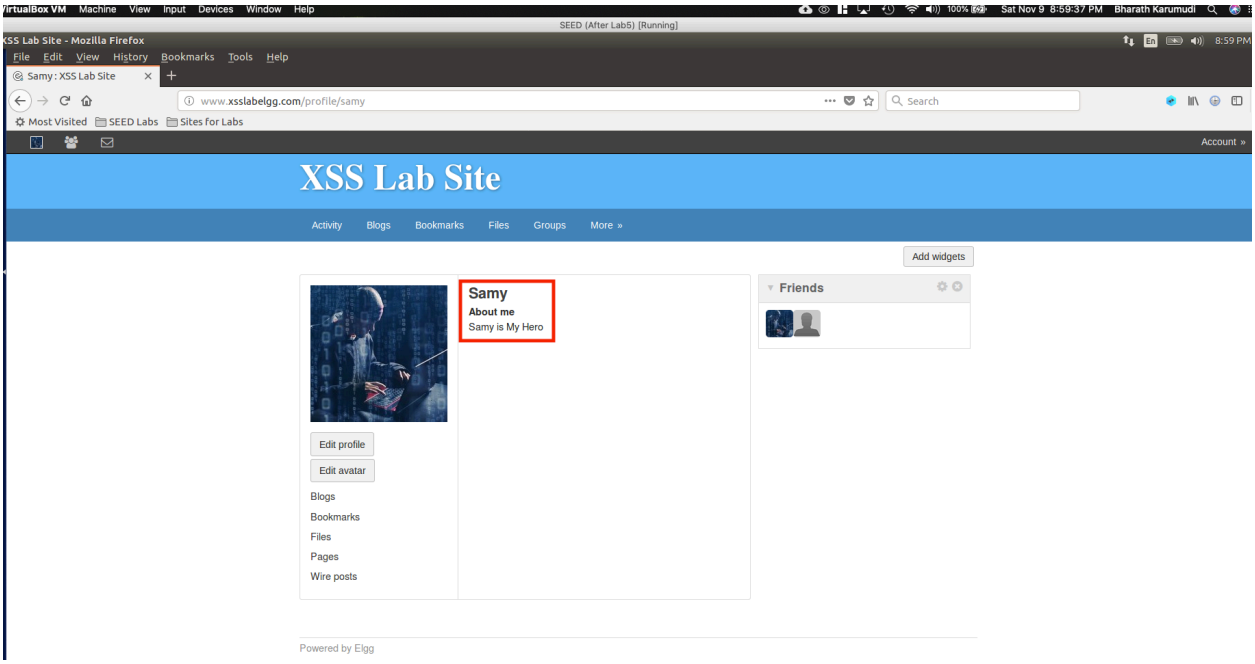
**Observation**: As Samy, I have updated the About me block with a Javascript code and when anyone view my profile page, it will get executed with an intention to edit the visitor profile

description as “Samy is My Hero”. When Alice viewed the Samy profile page, her profile description was updated to “Samy is My Hero”, without her knowledge.

**Explanation**: The JavaScript in the About me block was executed as code rather than data and the code has edit profile request. So, when Alice viewed the Samy profile page, the code was executed, and her profile description was also edited. Elgg server received the request as like a legit, because it includes a valid cookie and timestamp and token – because the request is initiated from the Alice Elgg page directly, we can get all these attributes from Alice page. This is due to XSS attack.

**Question3:**






**Q3 Observation:** If we remove the Line1, where the if condition, then the profile update will happen to Samy himself also. Because the same JavaScript will get loaded for him on profile page load and does the profile update for Samy to “Samy is my Hero” by replacing the malicious code. After this, if anyone visits the Samy profile the attacks will not happen as there is no code. This was happened in Task 4 also, where Samy made friend of himself, because there is no such validation/constraint defined in the JavaScript.

Task 6: Writing a Self-Propagating XSS Worm

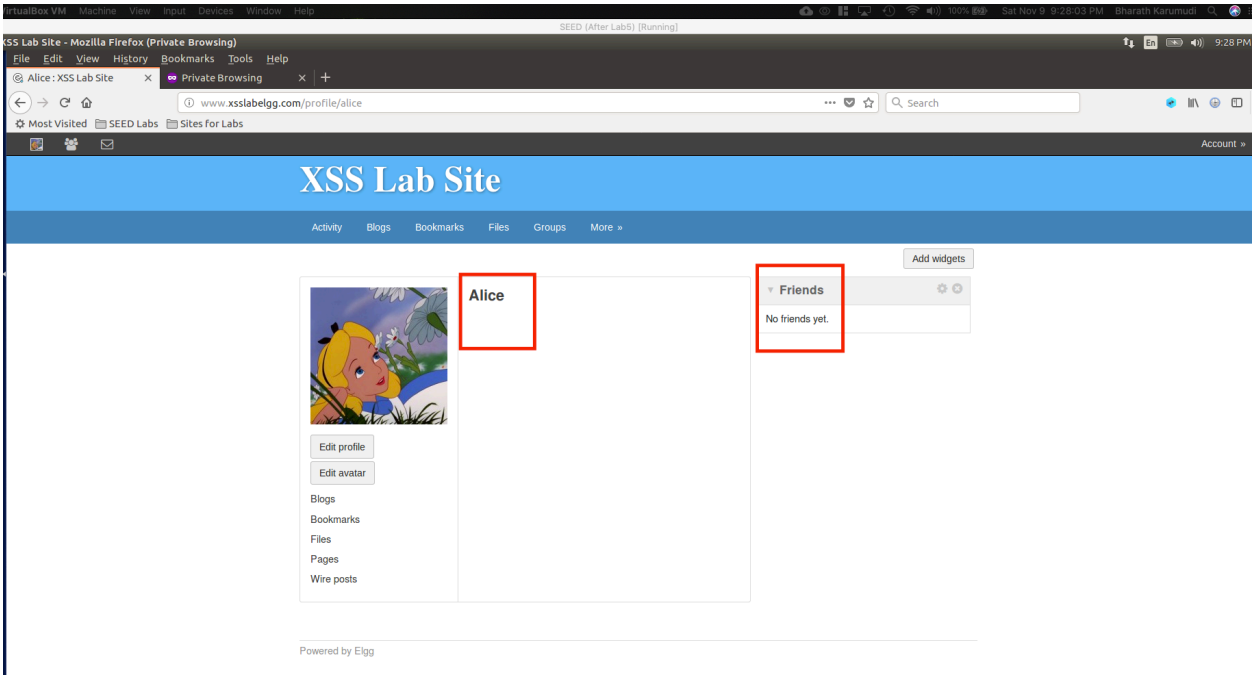



Fig: Showing Alice has no friends

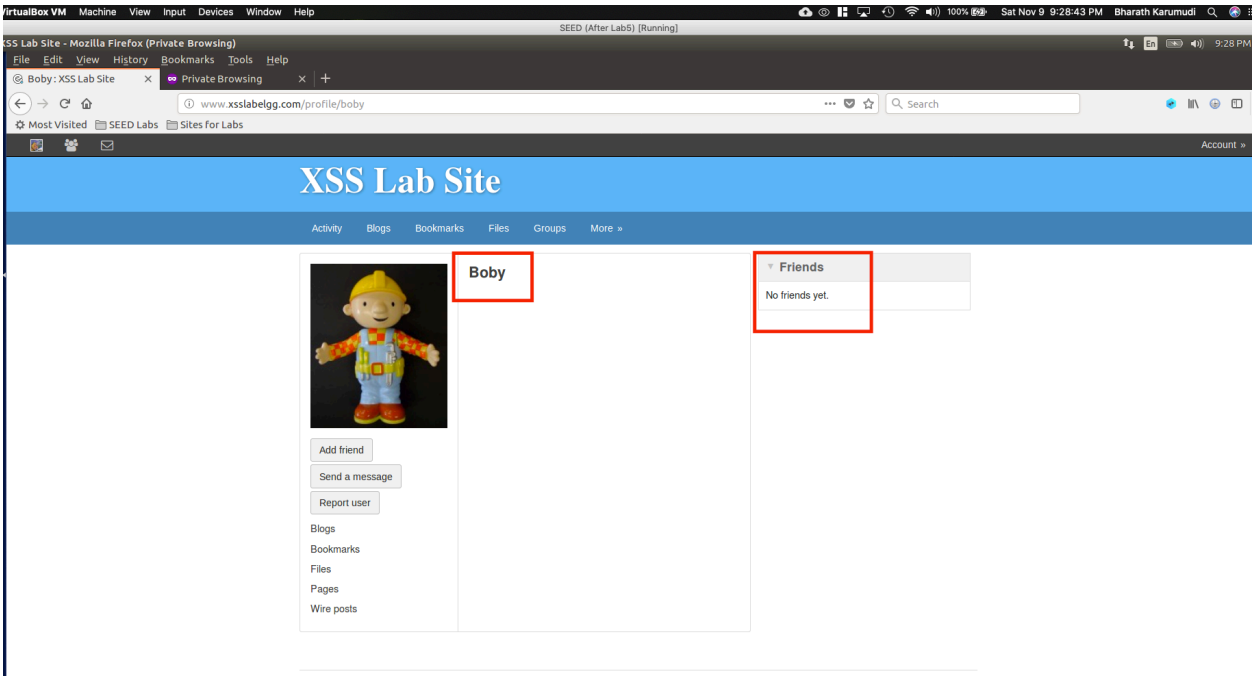



Fig: Showing Boby has no friends

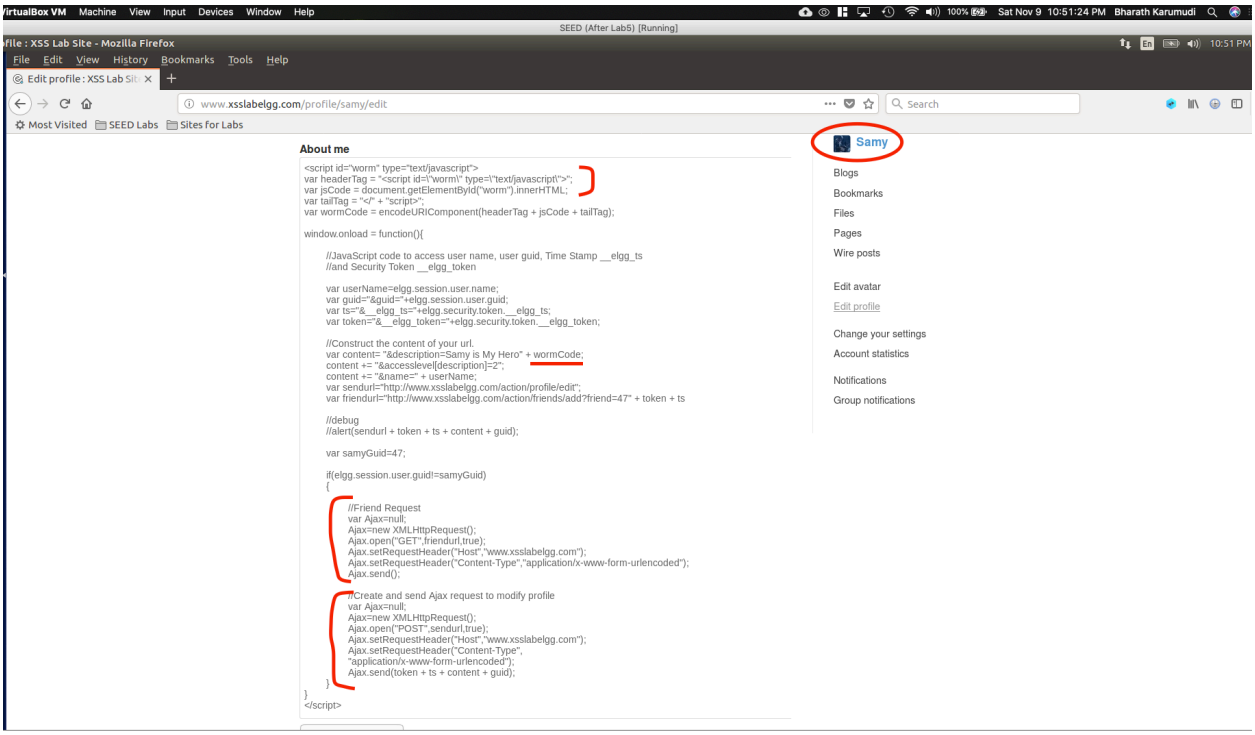



Fig: Samy updated his About me with malicious code – self propagating worm

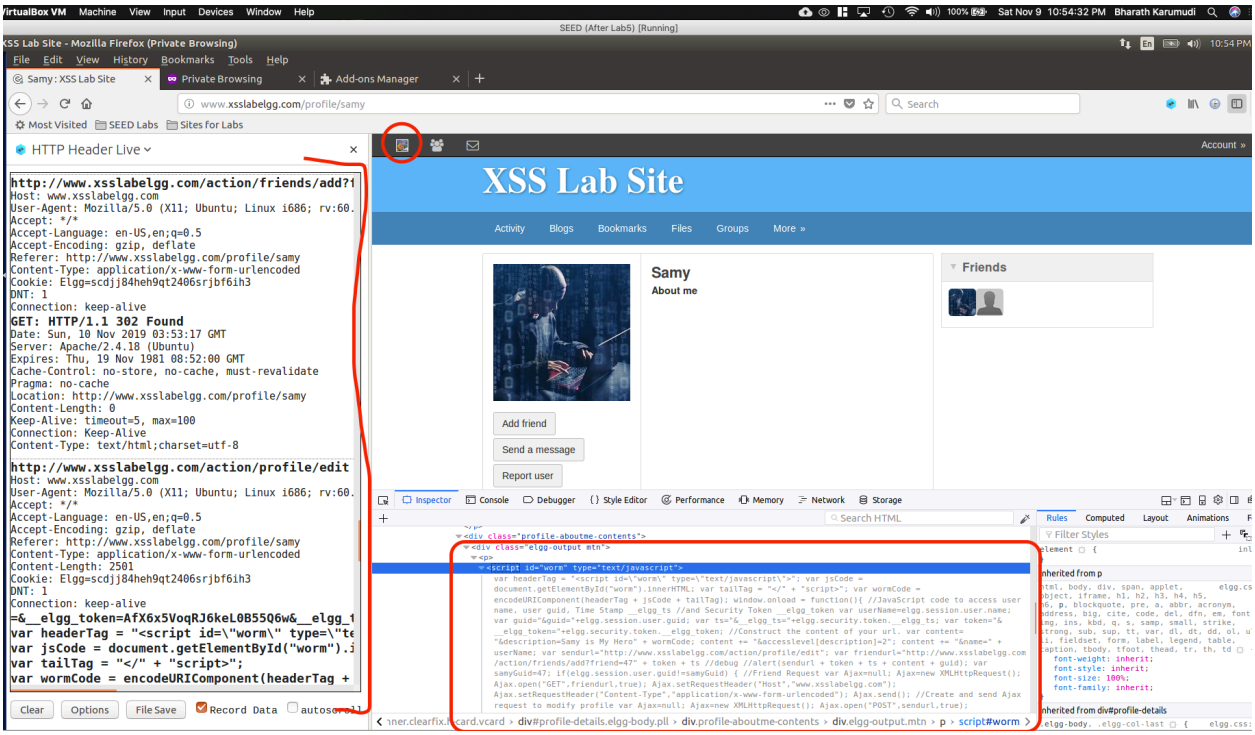



Fig: Alice viewed the Samy profile – showing HTTP requests and Code during that

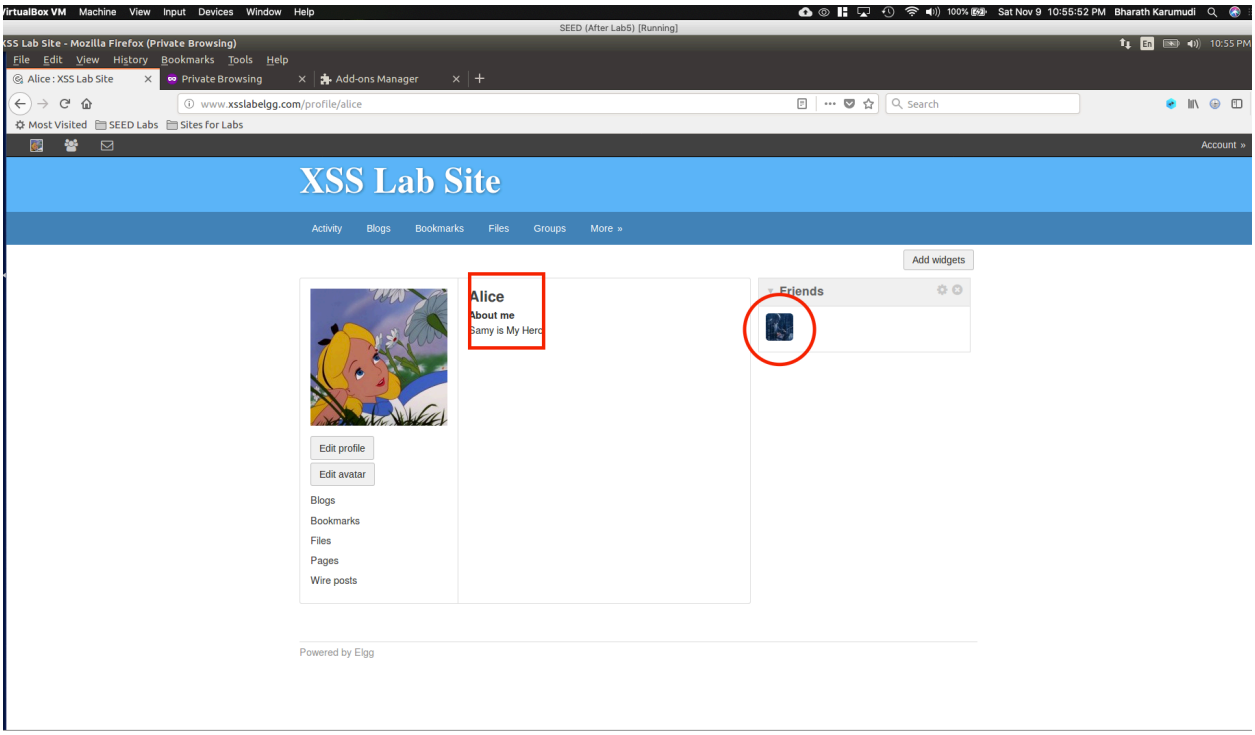



Fig: Alice now has Samy as friend and also Profile description was updated

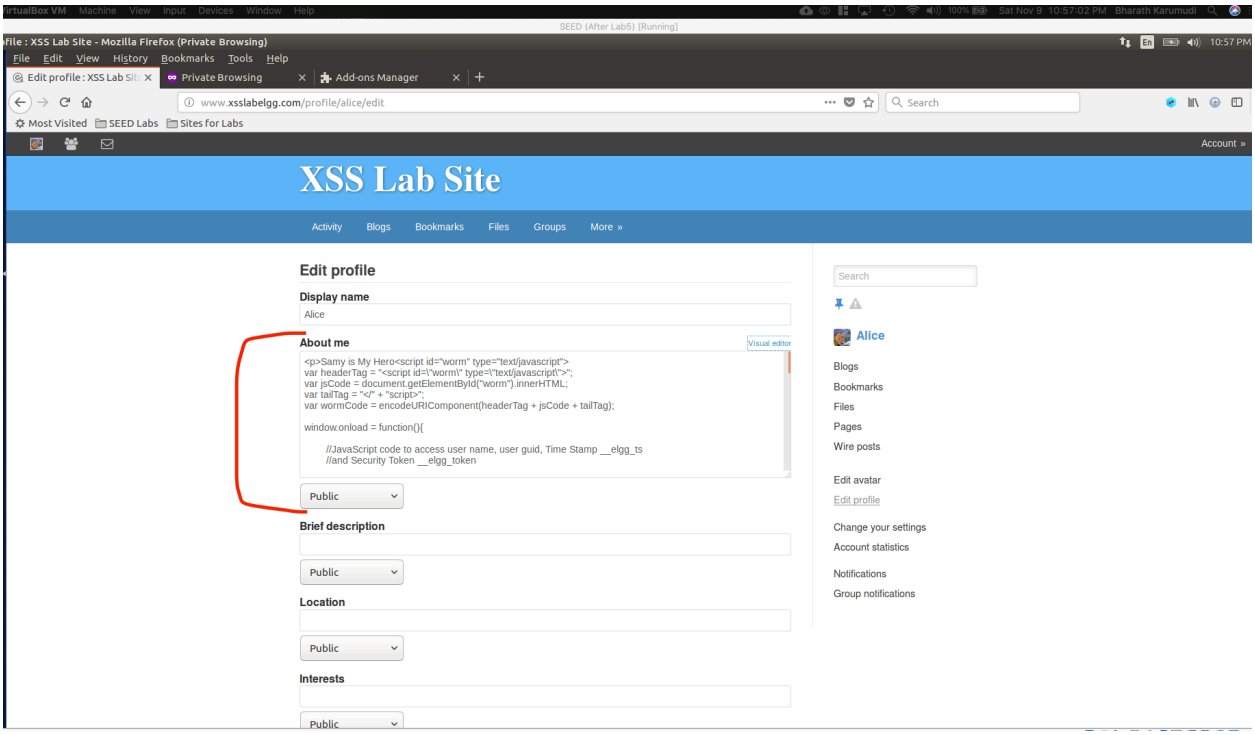



Fig: Alice profile has now malicious code

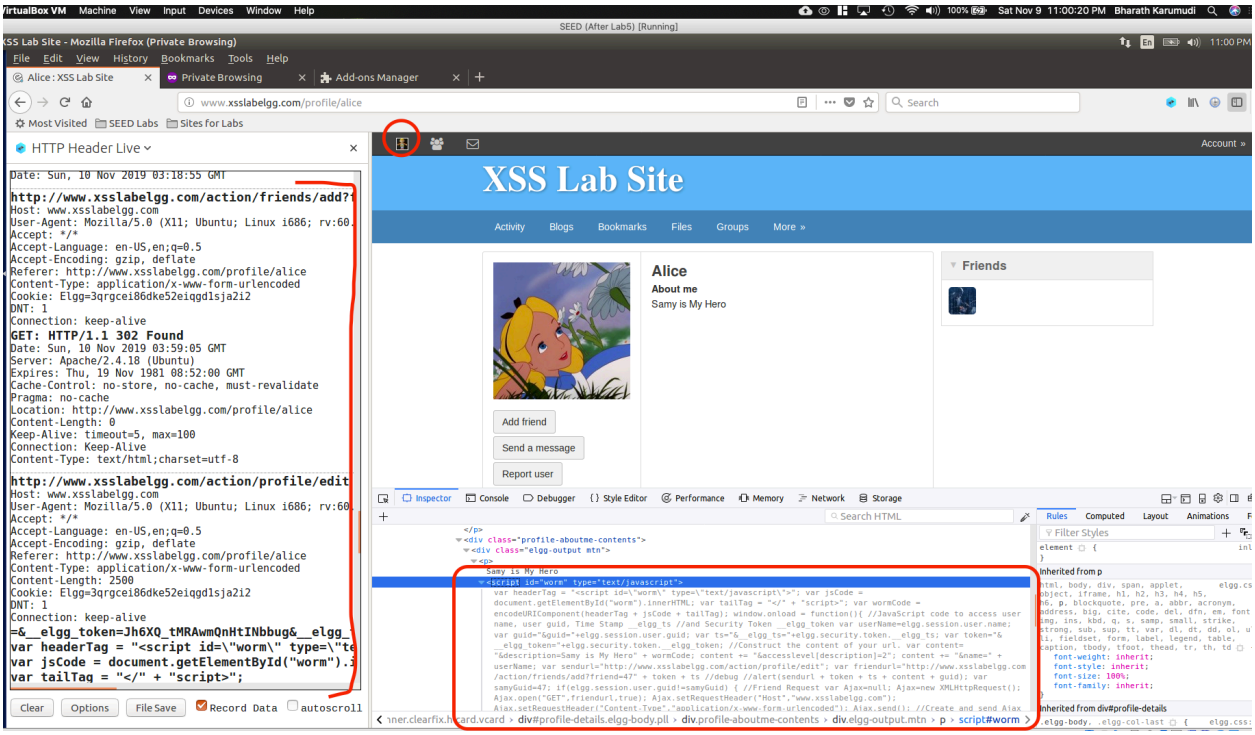



Fig: Boby viewing the Alice profile and showing HTTP Requests and Code during that

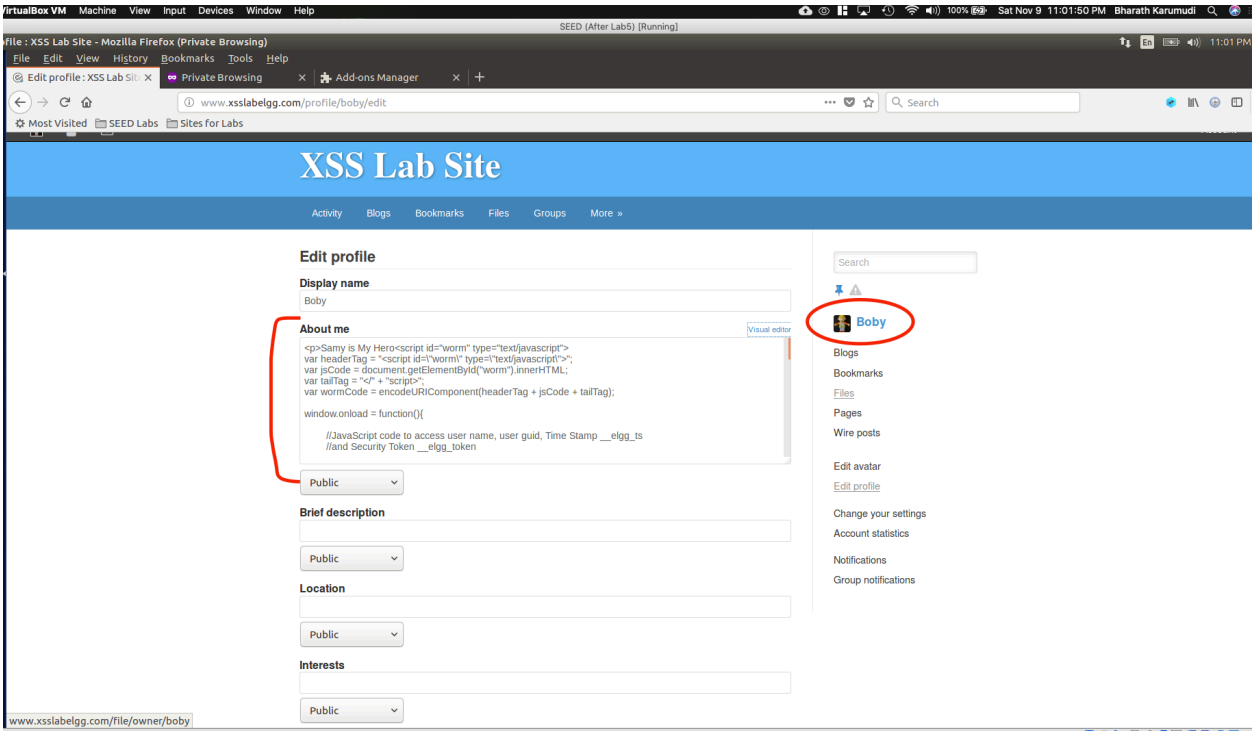



Fig: Boby profile description was also updated and code was copied

**Observation**: As Samy, I updated my “About me” with JavaScript that self-propagates. When Alice viewed my (Samy) profile, two actions are done - Added Samy as Alice friend and Alice

About me is also updated with “Samy is My Hero” along with the malicious code. When Boby visited the Alice profile, Samy was added to Boby friend and Boby About me is also updated with “Samy is My Hero” and the malicious code.

**Explanation**: This is XSS attack and the code this time Samy has a self-propagating worm. The JavaScript in the About me block was executed as code rather than data and the code has two Ajax calls (i)add friend and (ii)edit profile actions.

So, when Alice viewed the Samy profile page, the code was executed, and her profile description was also edited which has malicious code also. Now Alice is also acting as attacker (indirectly) and when Boby viewed Alice profile, the same action happened, and he also now turned in to attacker (indirectly). This way the worm was spreading from user to user, which was happened to Myspace.

During the profile visits Elgg received the requests as like a legit, because it includes a valid cookie and timestamp and token – because the requests are initiated from the user’s Elgg page directly. This is due XSS attack.

## Task 7: Countermeasures

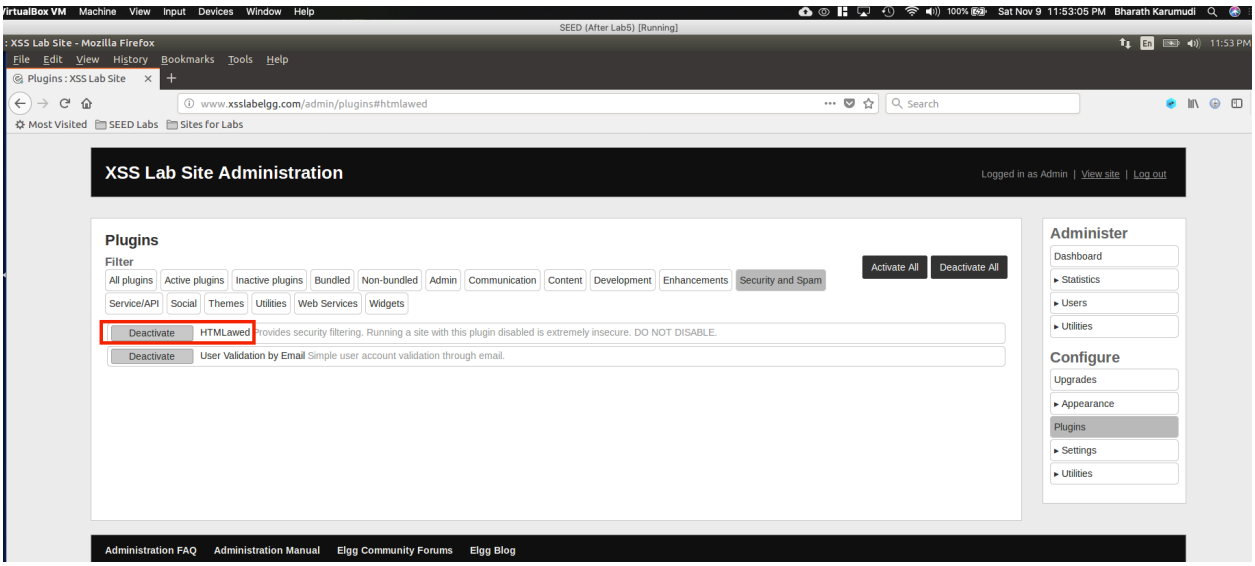



Fig: Showing HTMLawed plugin was activated as XSS countermeasure

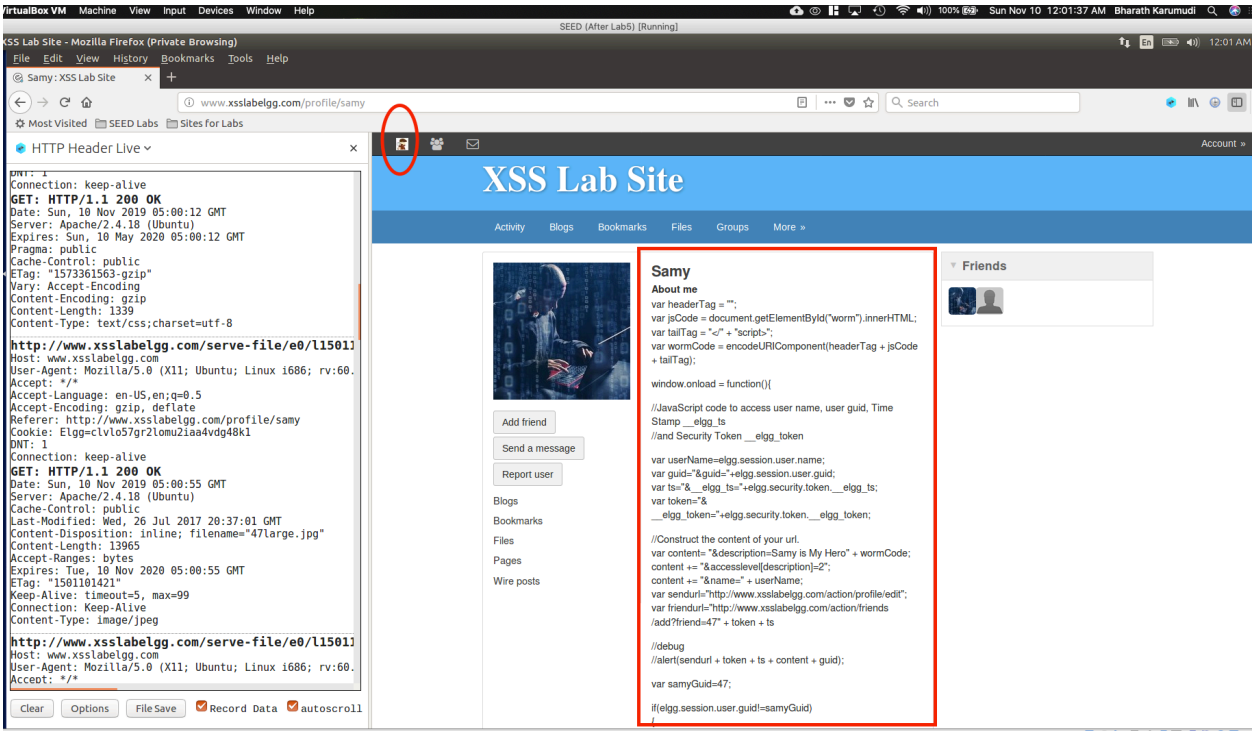



Fig: The code now turned into a text and all the tags were removed by countermeasure

**Observation**: After enabling the HTMLawed 1.9 security plugin, when Charlie visited Samy profile the About me was like normal text (broken code) and nothing effected on Charlie account. Where Samy intention was to do the Add friend and profile edit. Thus, XSS attack failed.

**Explanation**: HTMLawed plugin is an Elgg countermeasure for XSS attack. Which removes the code tags as part of security filtering and make them as normal text (like a broken code). So, nothing will be impacted during page loads (meaning no code executions). XSS attack was defended successfully.

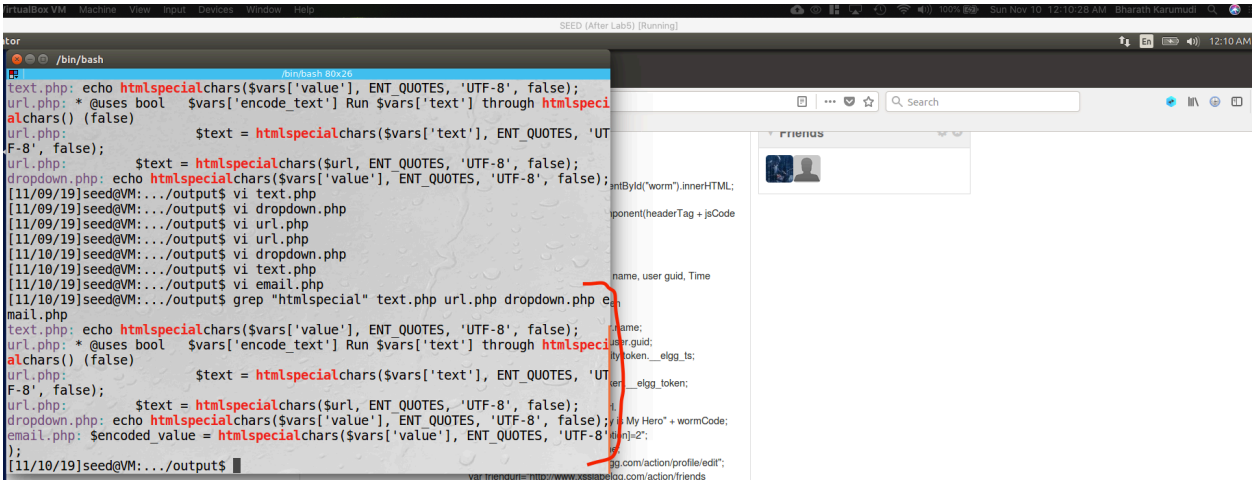



Fig: Showing enabling the htmlspecialchars() – XSS countermeasure

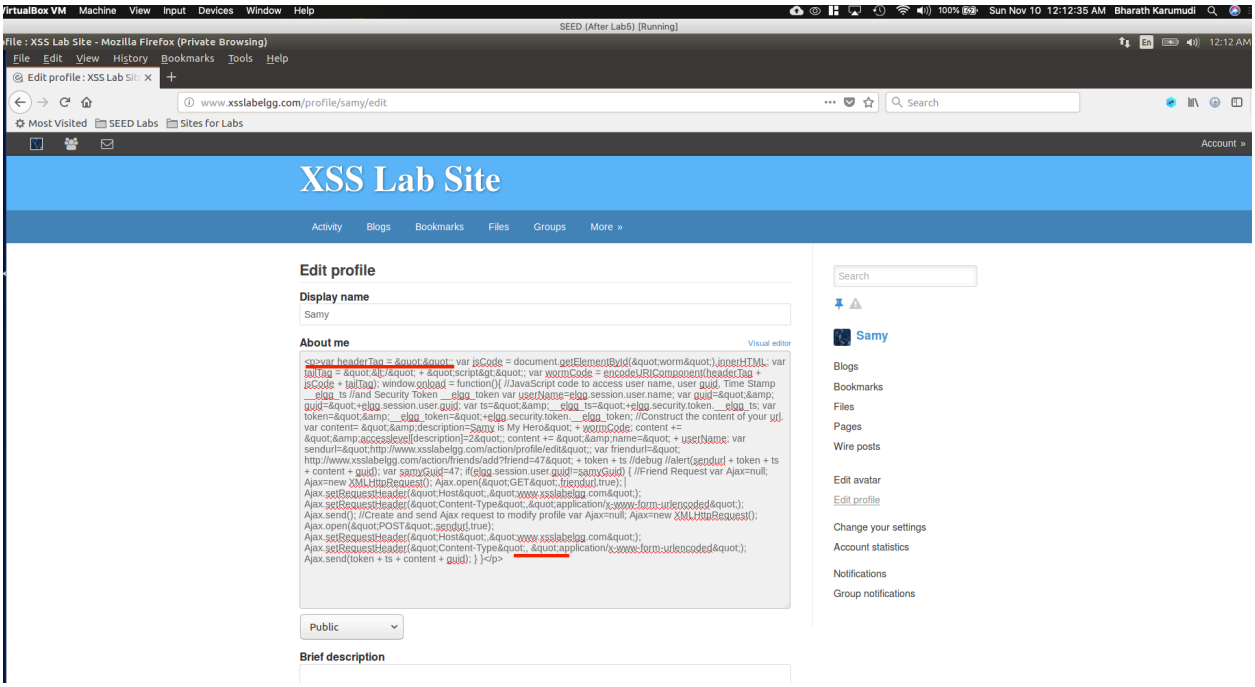



Fig: Showing HTML codes are transformed

**Observation**: After enabling the htmlspecialchars() in the text.php, url.php, dropdown.php and email.php files, the Samy profile's About me has changed with HTML special quotes like single quote to " So XSS attack was defended.

**Explanation**: This is Elgg XSS countermeasure to handle XSS attacks which translates the special characters to respective HTML based. This will make the code not executable. Even during the page load the JavaScript will be failed to execute and also the browser will not even identify it as a JavaScript code. Thus, it makes XSS attack fails.