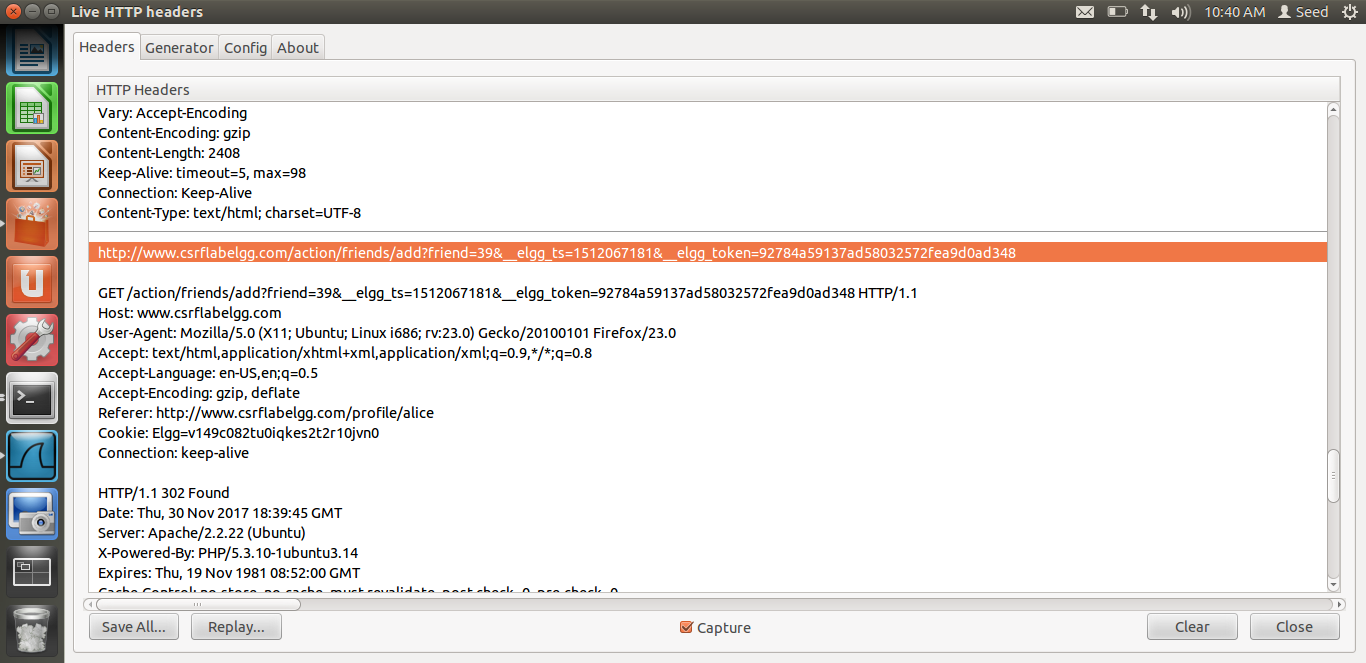
**Cross Site Request Forgery Lab Report (Elgg)**

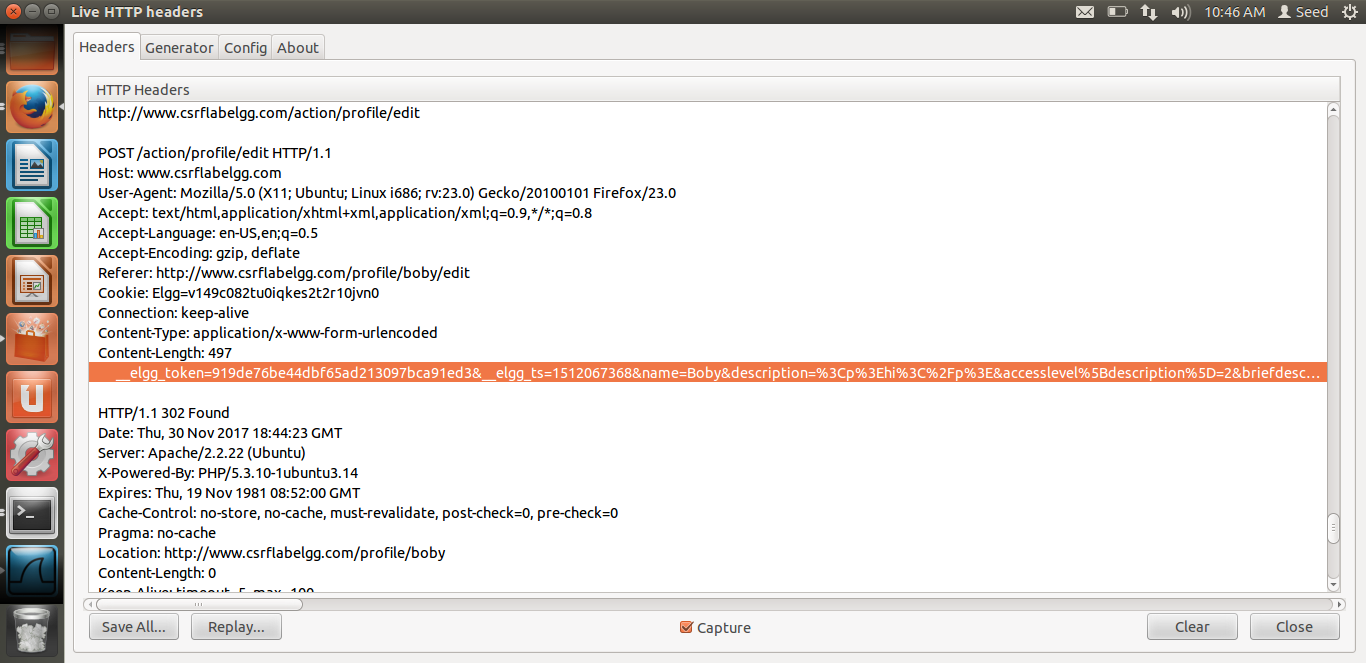
**Task 1 Procedure:**

1. **When Boby adds Alice as a friend from his account, we can see the Live HTTP Header and identify the Add friend HTTP request, which is a GET request as shown below.**

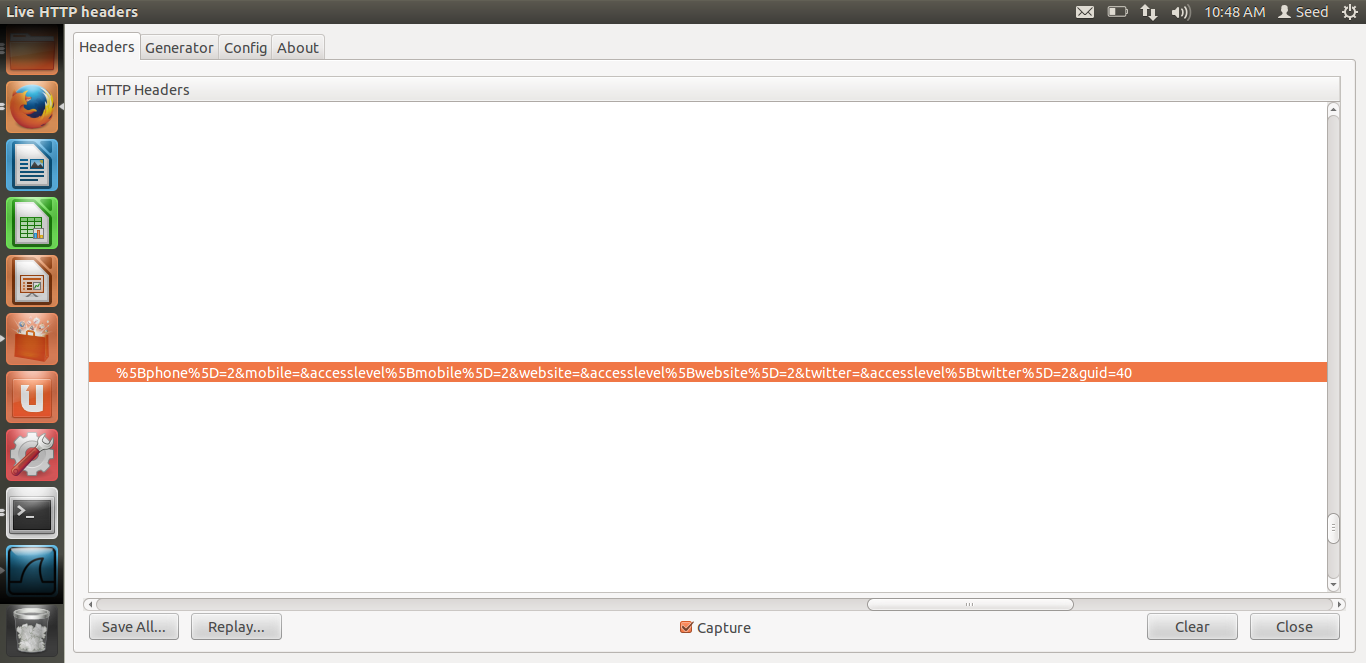
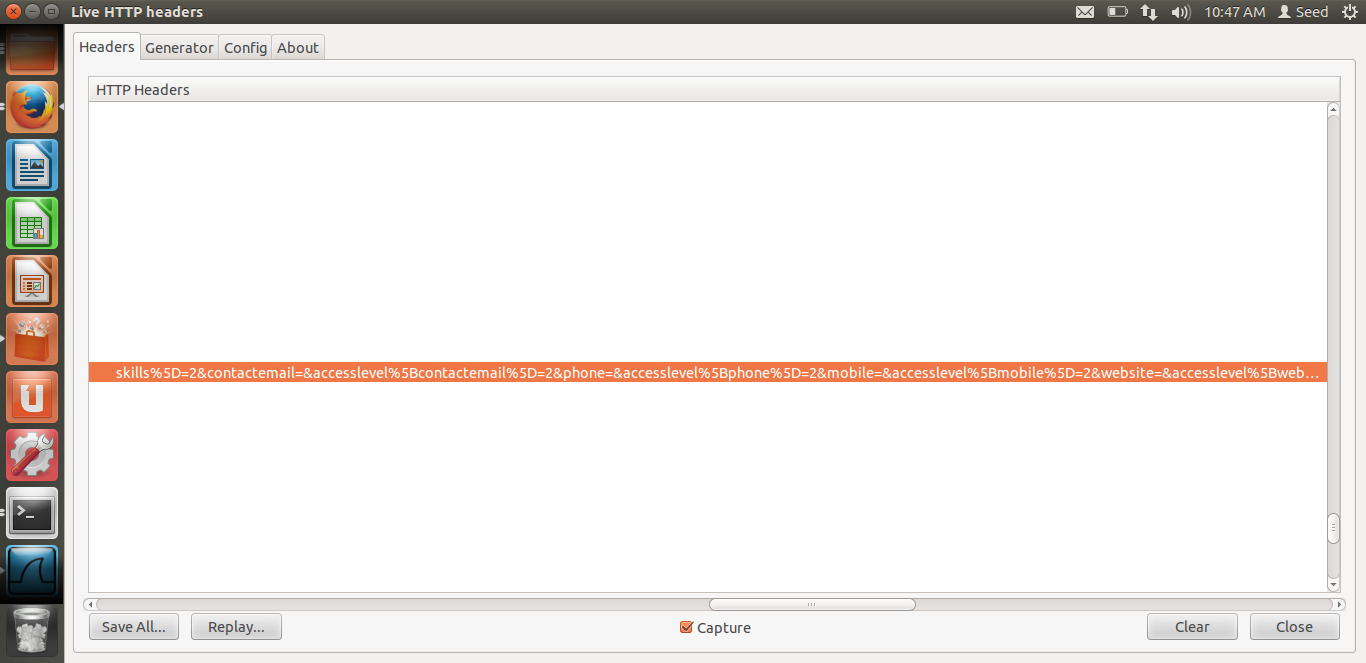
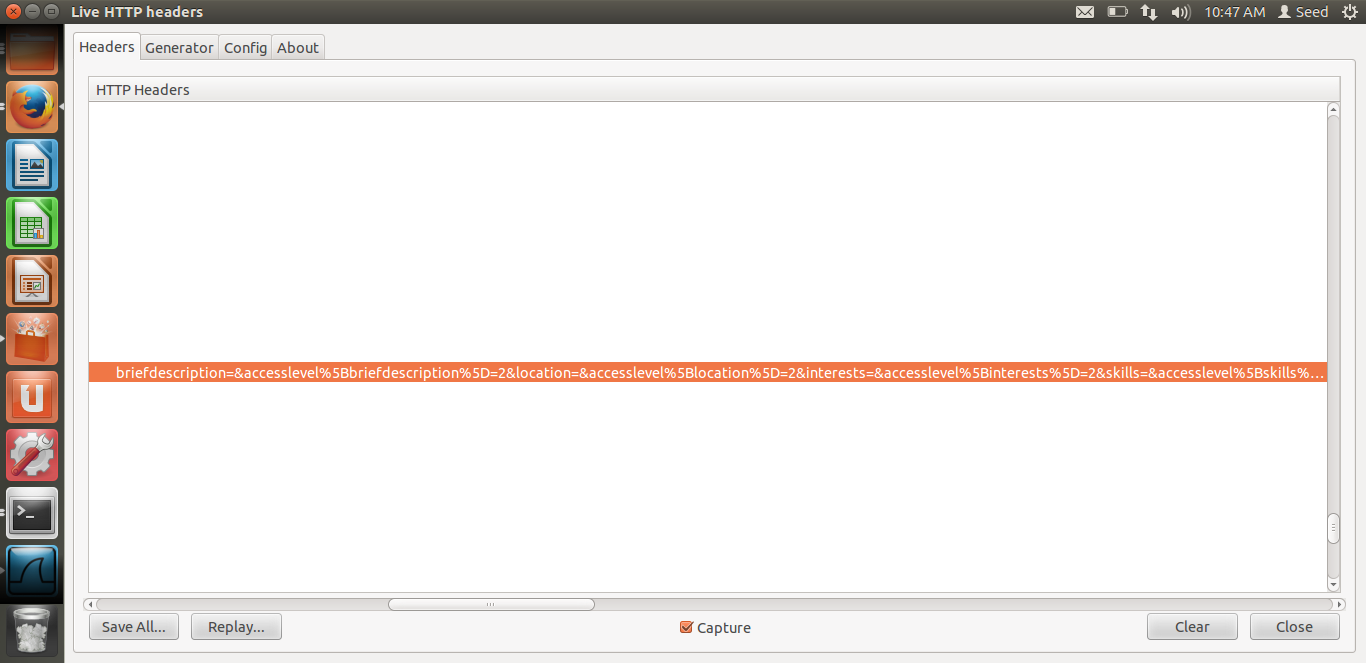
****

**Figure 1.1**

1. **When logged into Boby’s account, Boby can check his guid by looking at his HTTP Request Header, when he clicks edit account on his homepage. By looking at his edit Profile request HTTP Header, we are able to determine that his guid is 40, we will use this knowledge to write the code for the malicious website.**

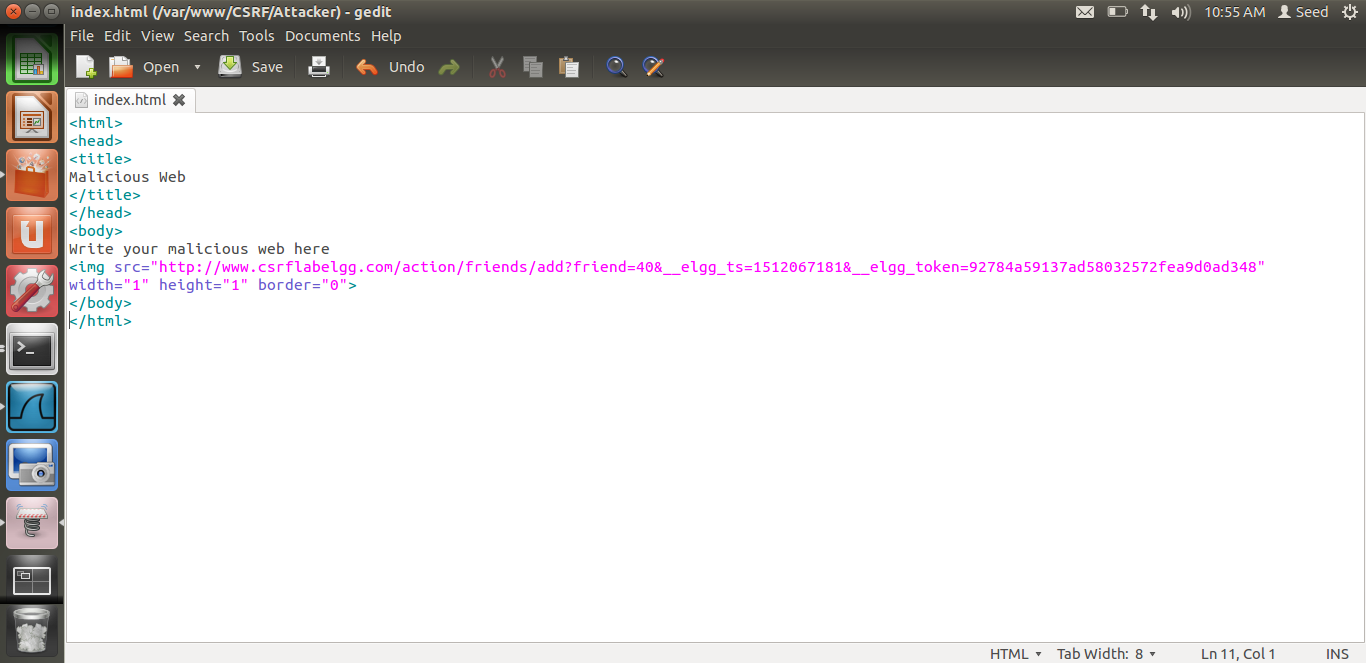
****

Guid is 40.

****

**Figure 1.2**

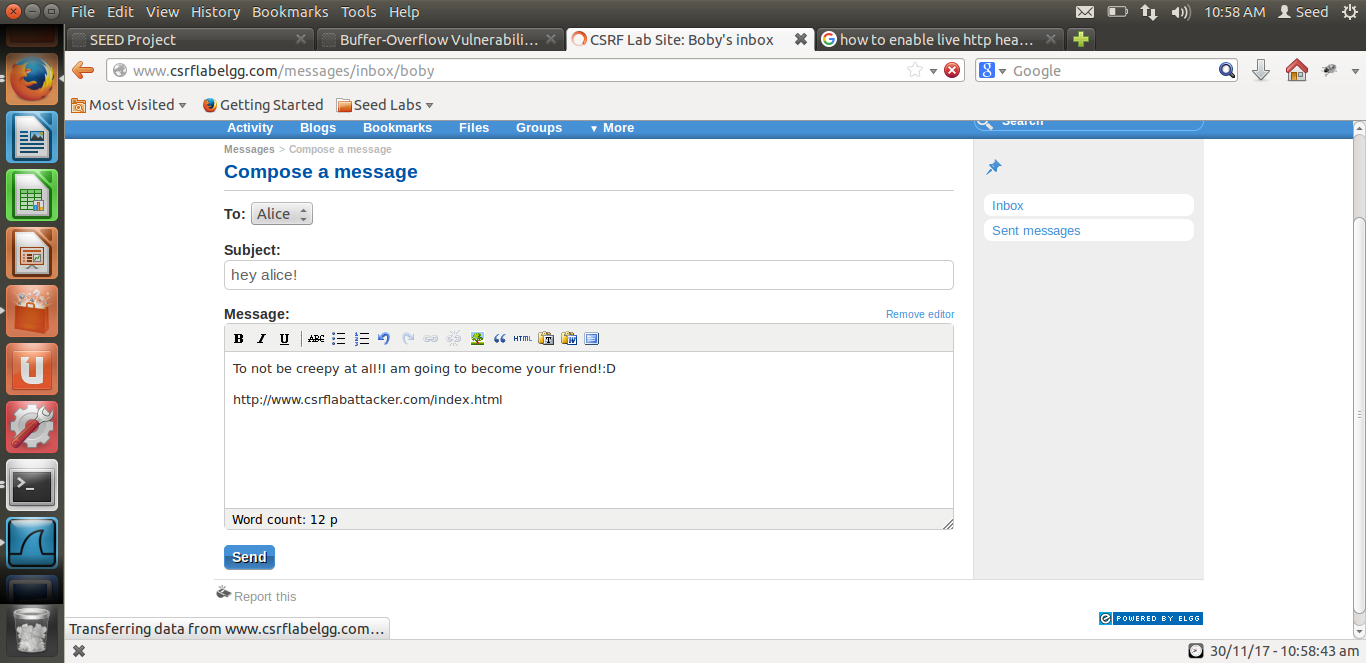
1. **We then edit the Malicious code with this img tag.This should have the link we get from the GET request we get for adding Alice as a friend of Boby. Now to make Alice add Boby as a friend, we submit the request in the form of an image of 1px by 1px size. So when, Alice clicks this link, the request is immediately sent to Elgg website and Boby is added as a friend.**

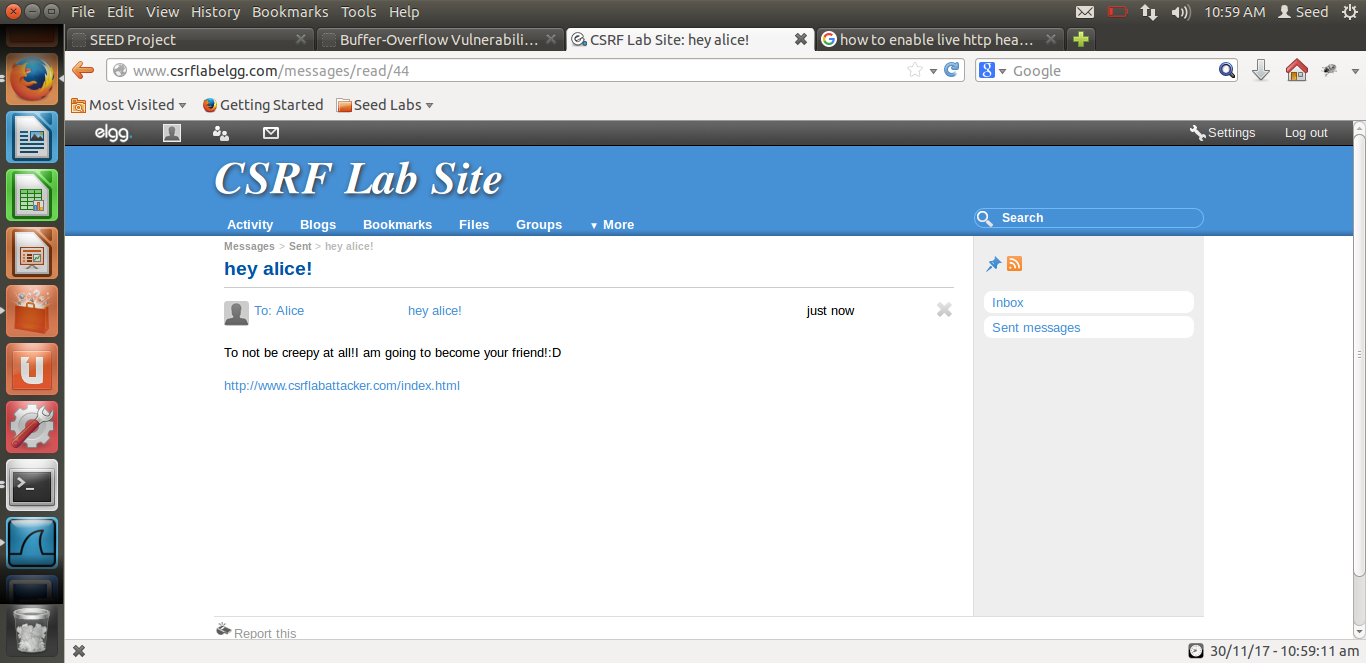
****

Malicious code edited.

**Figure 1.3**

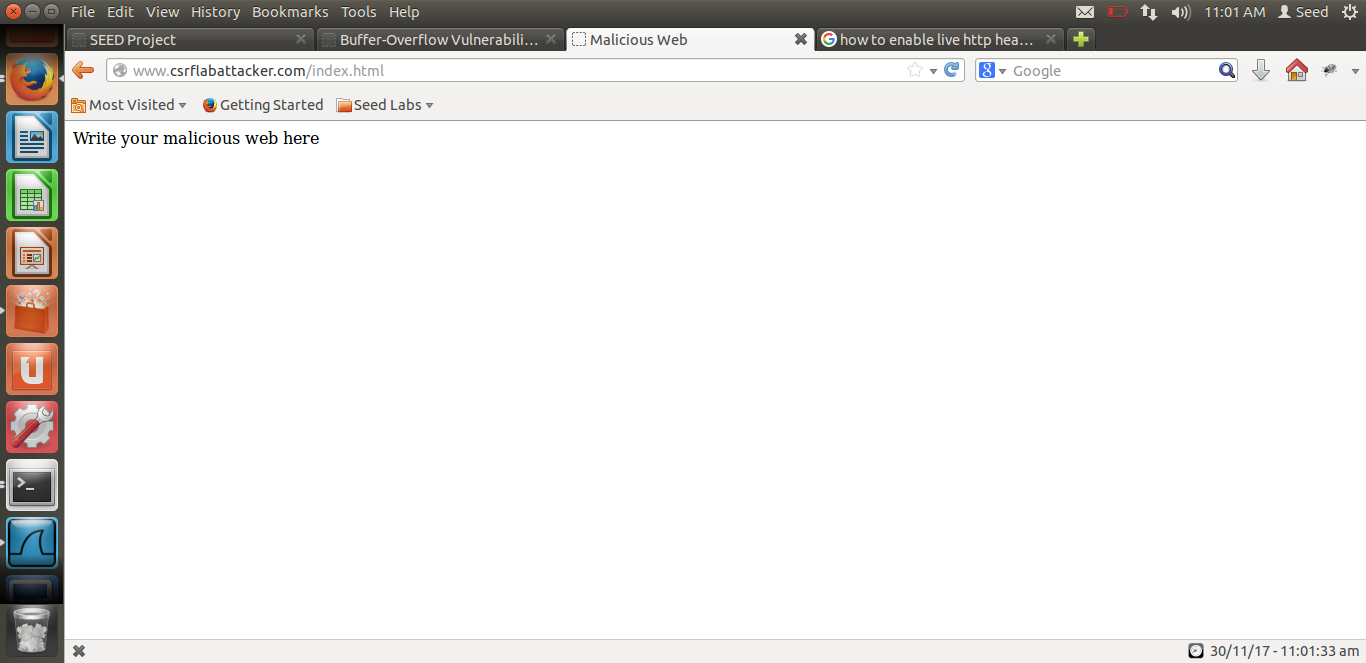
1. **Boby sends a mail to Alice along with the link. This link contains the malicious code as shown below.**

****

****

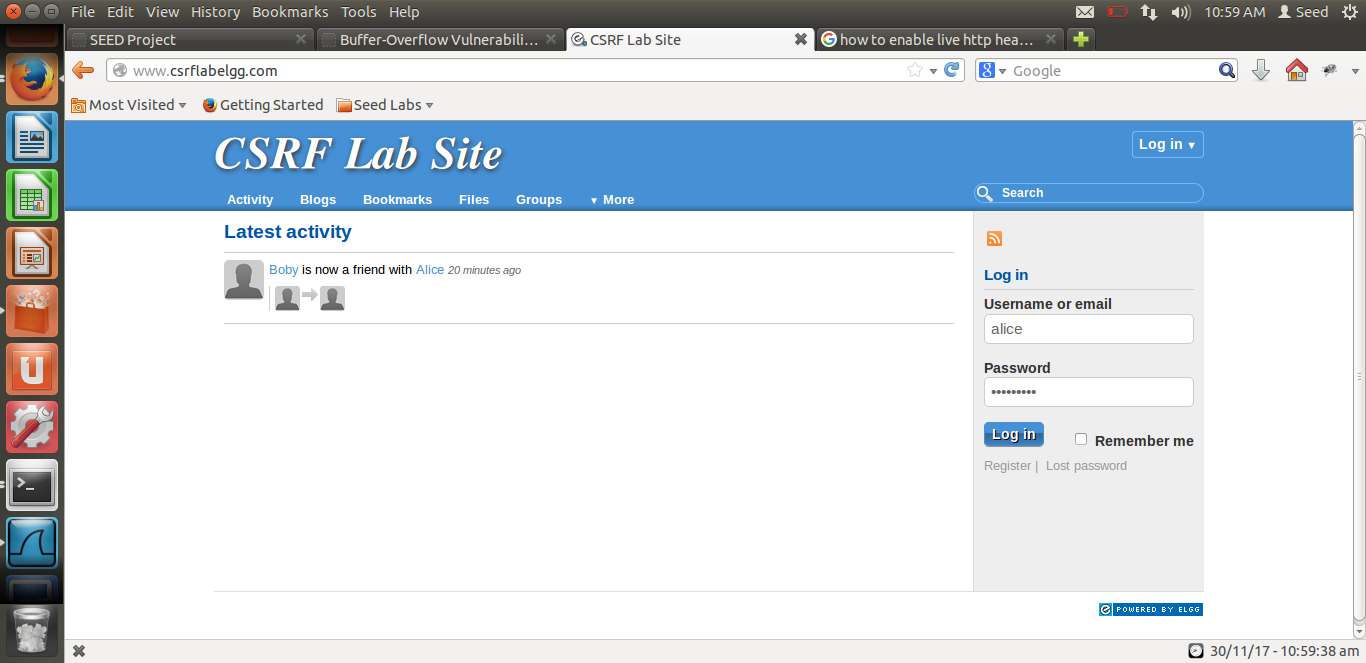
**Figure 1.4**

1. **Alice sees a message from Boby with a link in it and opens the link, where she gets redirected to the malicious website.**

****

**Figure 1.5**

1. **When Alice clicks on the link in the message, she is redirected to the** [**www.csrflabattacker.com/index.html**](http://www.csrflabattacker.com/index.html) **page which contains the img tag that has the malicious line of code. So as soon as the page is opened, without Alice clicking anything, the img tag is read a request for an image from the add friend link is made and Alice adds Boby as a friend, where Boby’s guid is 40.**
2. **As we see in the below figure , the attack is successful. Boby is added as a friend of Alice.**

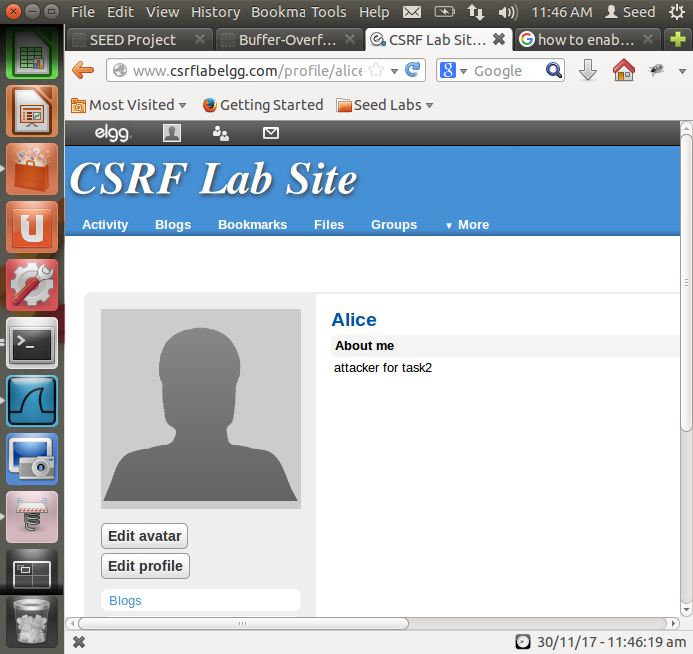
****

Attack successful

**Figure 1.7**

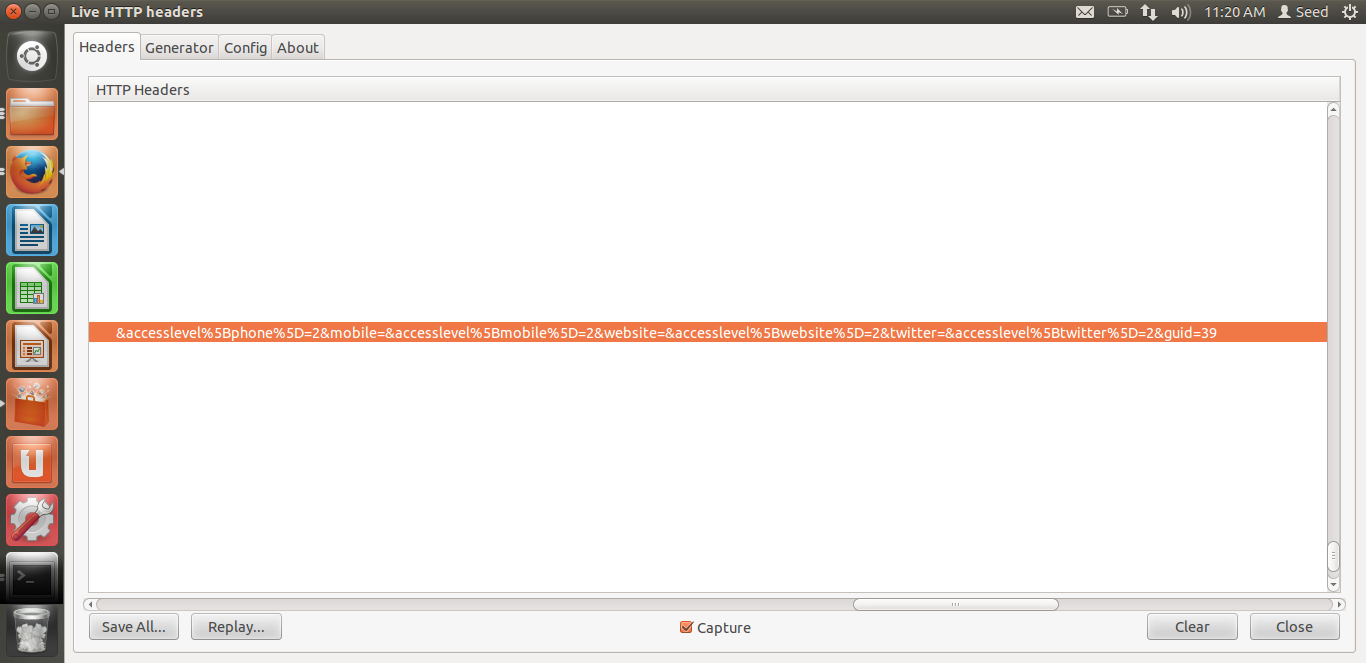
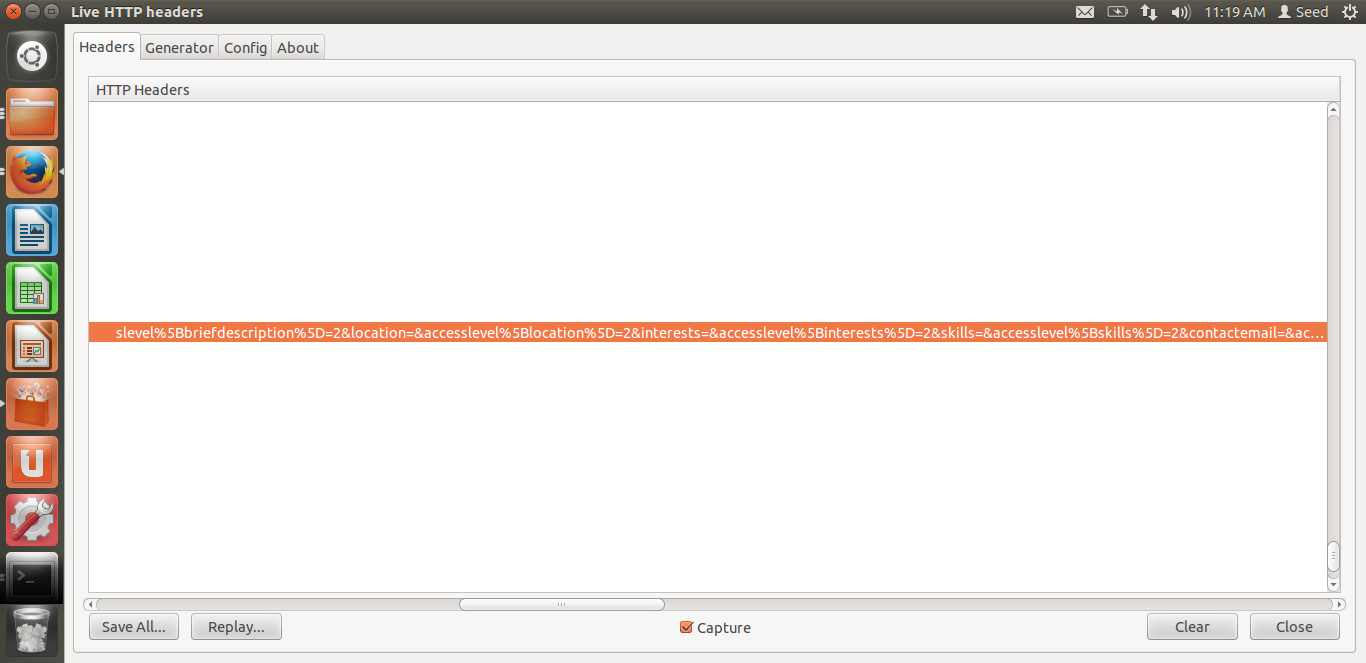
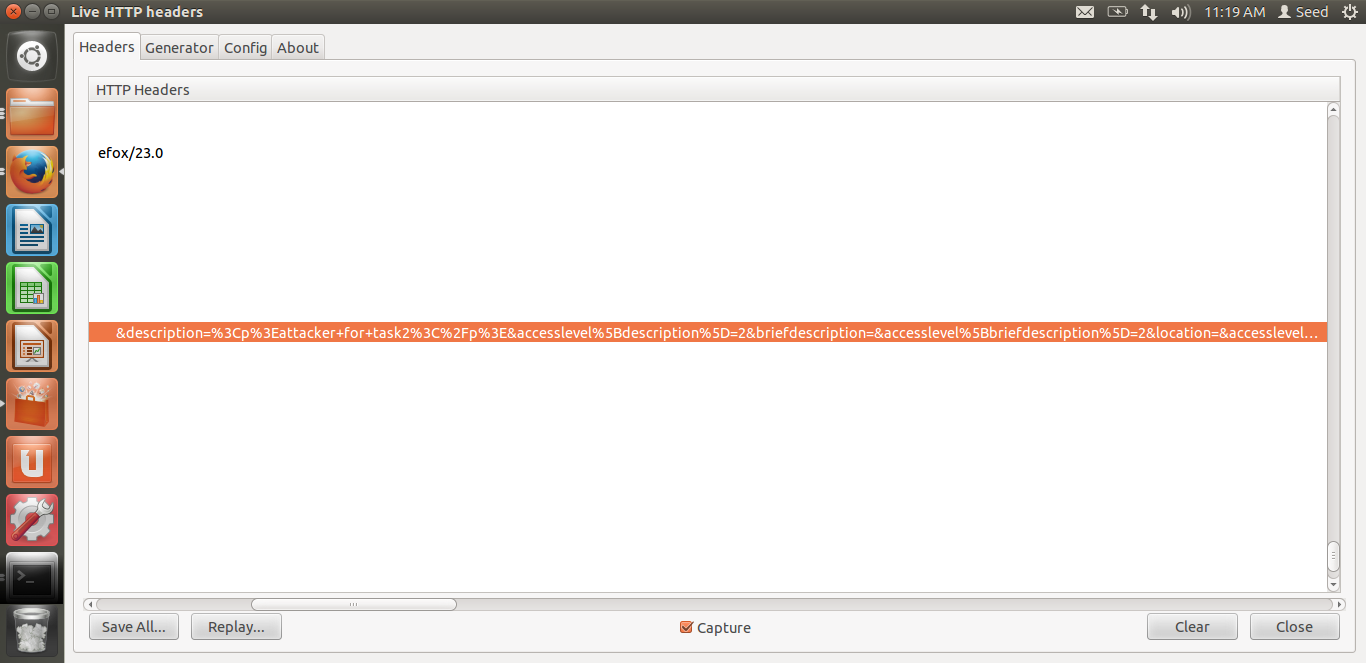
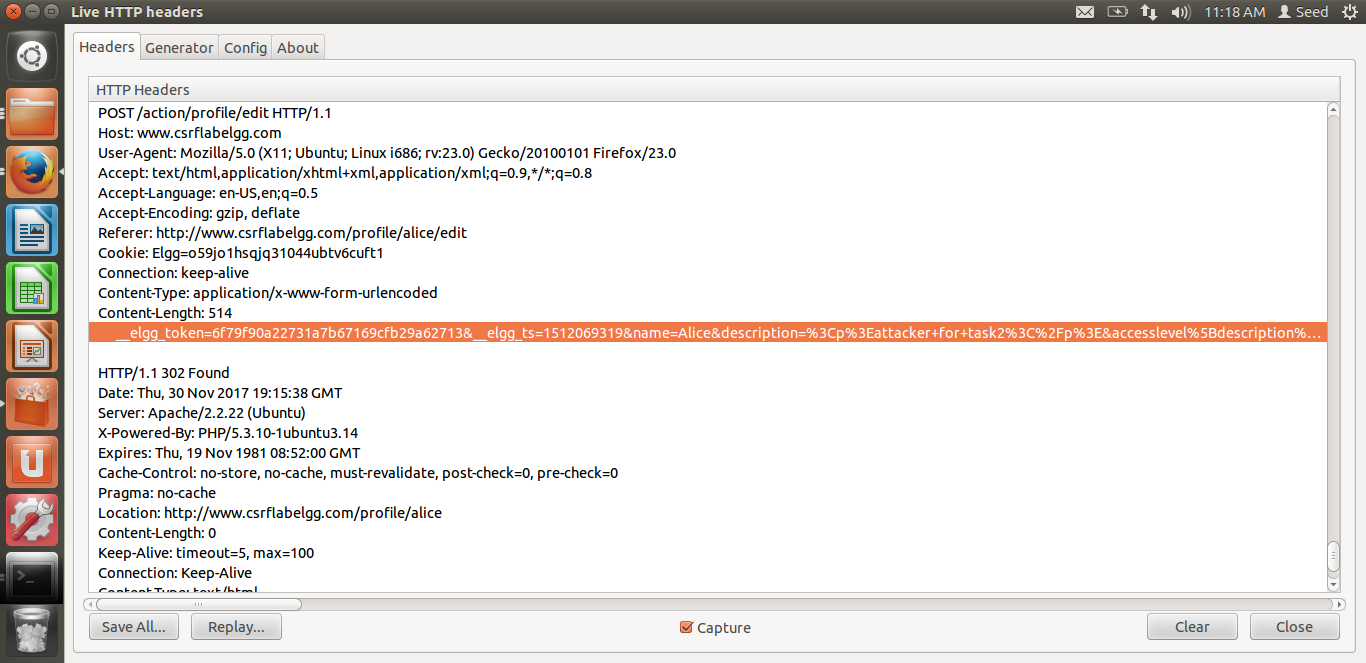
**Task 2 Procedure:**

1. **Alice becomes the attacker in this Task, she is going to add something to Boby’s “About me” section. As shown below in her description she is the attacker.**

****

**Figure 2.1**

**We make an edit to Alices page and check the HTTP header. From there we get the URL to be used in the javascript function(shown below). This URL must be added to a javascript code to perform this action without Boby’s permission.**

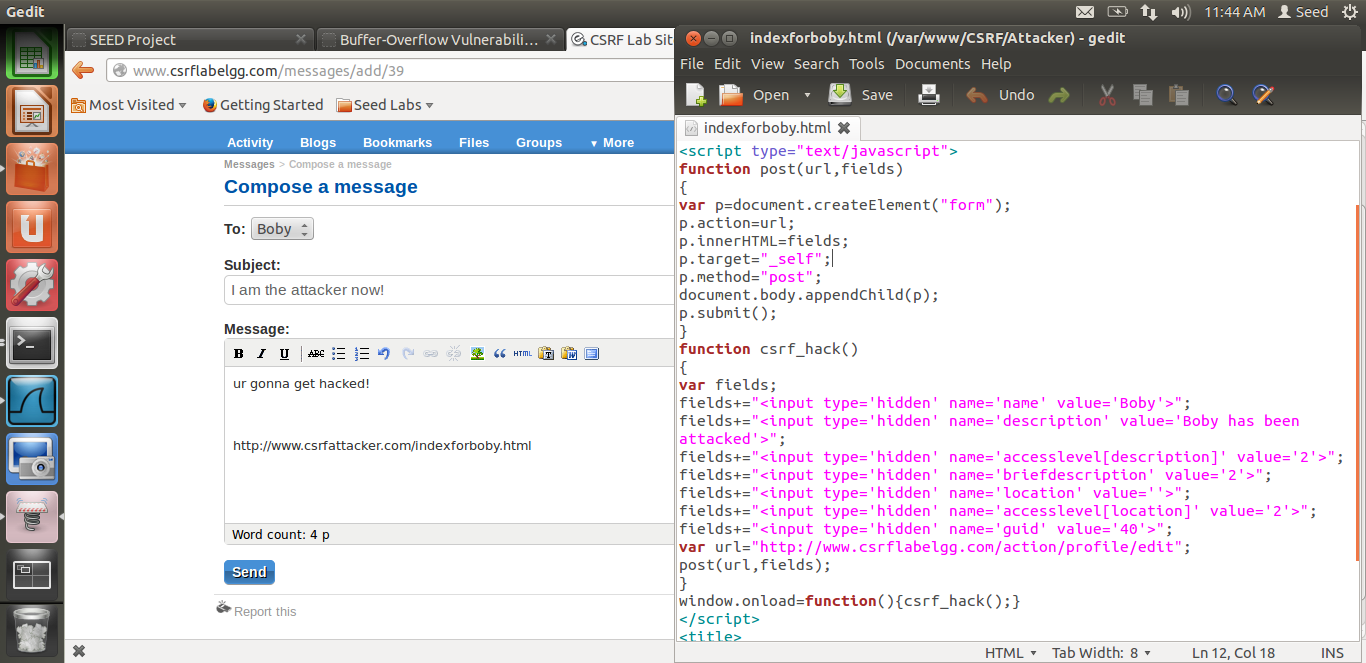
****

**Figure 2.2**

1. **The javascript code to attack Boby is shown below. Alice sends a message to Boby, with this link attached to the message.**

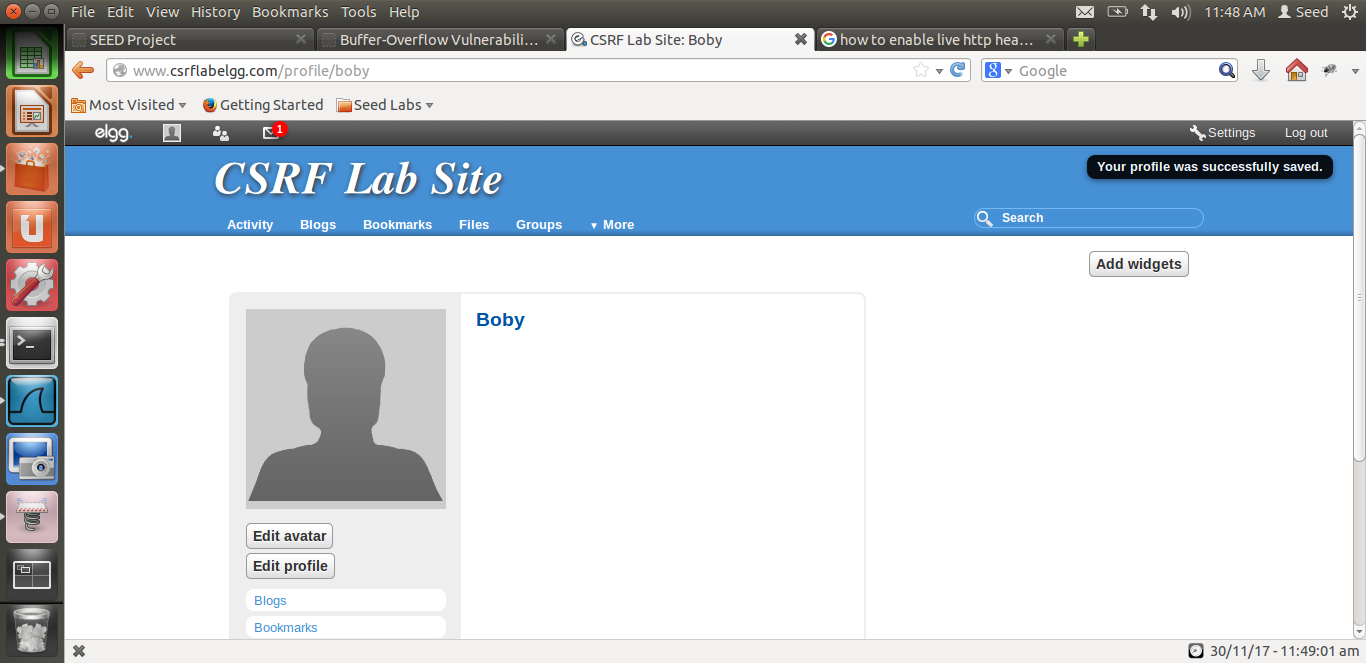
**When Boby opens this mail the javascript code will run and add the entry that Alice wants him to have in his “about me” section.**

**The code is shown below as “indexforboby.html”.**

****

**Figure 2.3**

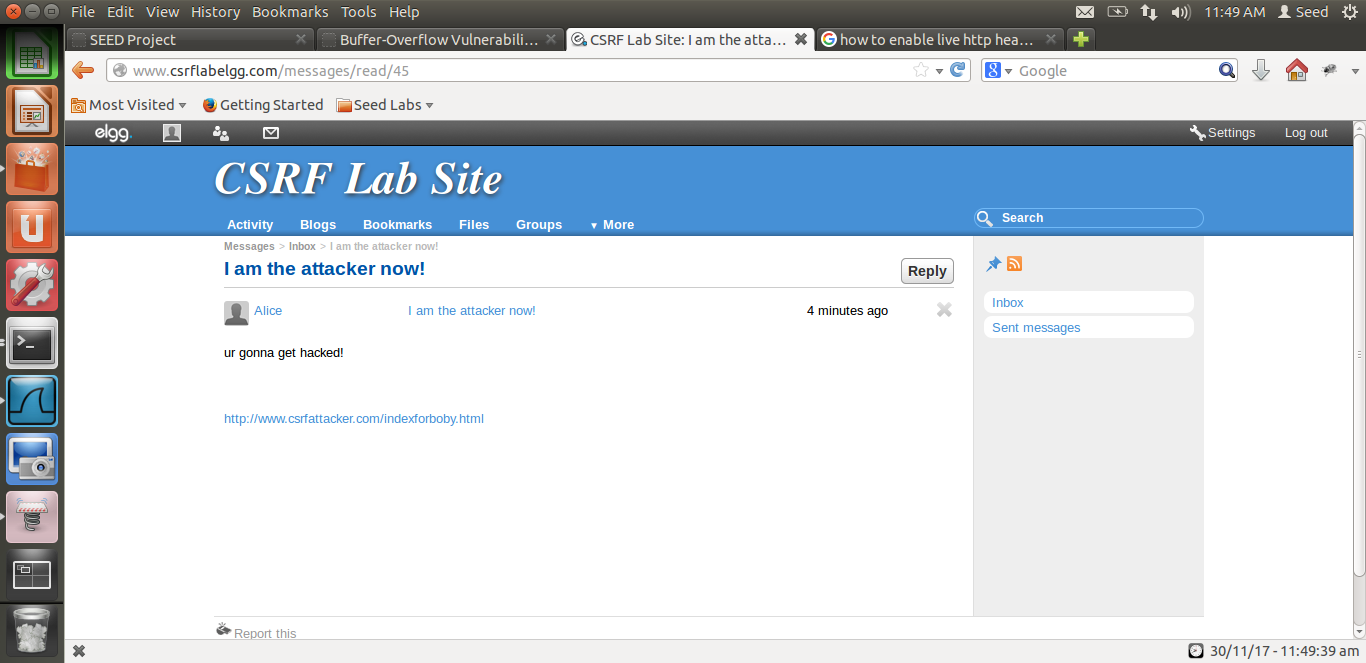
1. **Boby’s webpage before he opens Alices message and clicks on the link, along with the HTTP Header on opening his webpage. His about me section is blank.**

****

Empty desc.

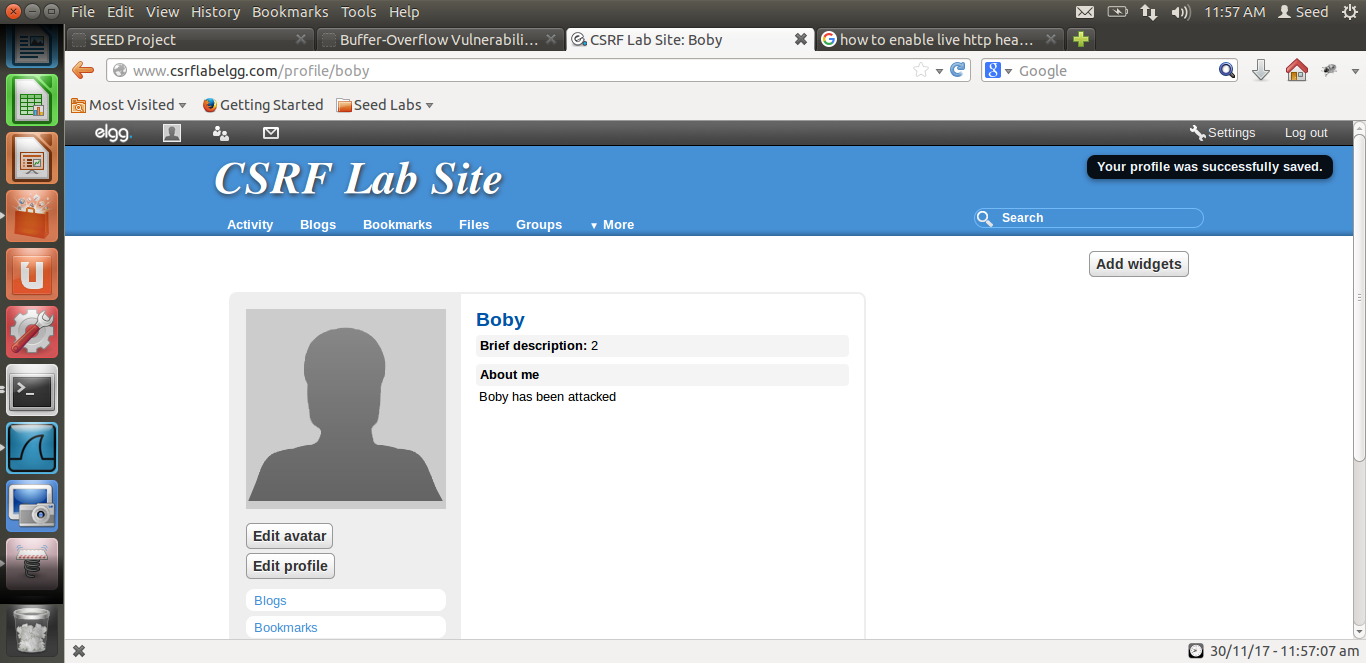
**Figure 2.4**

1. **He receives mail from Alice.**

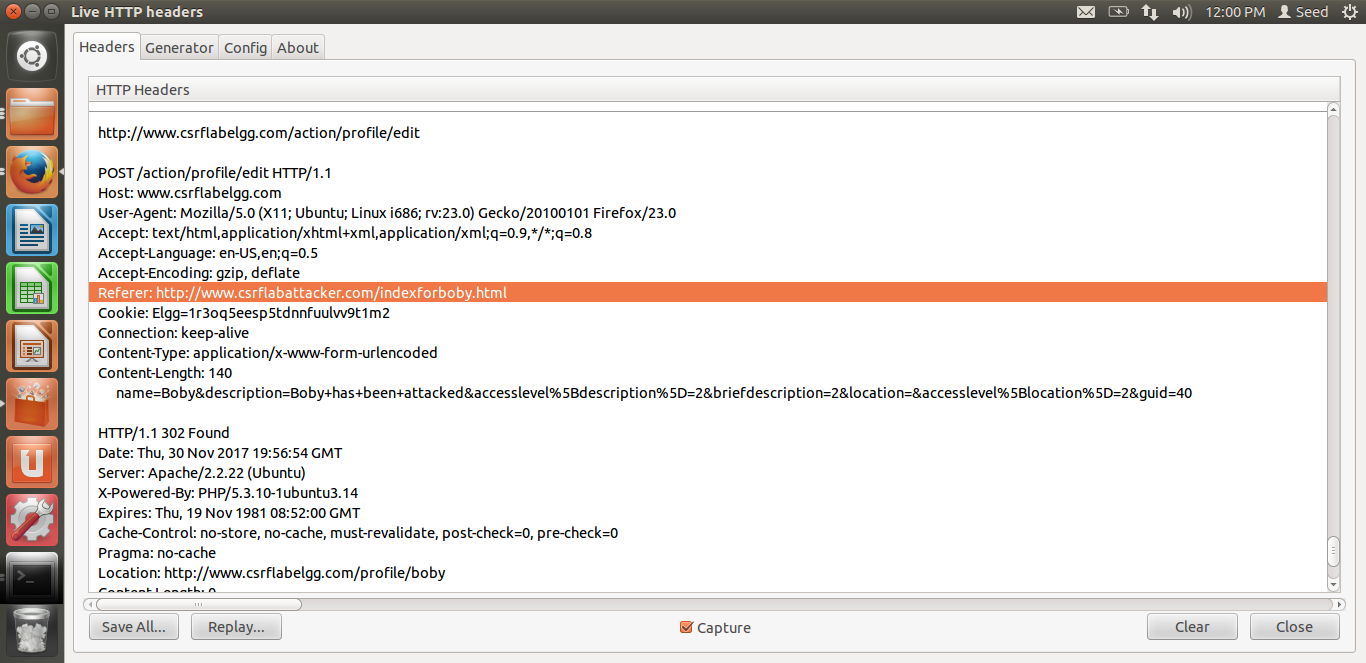
****

**Figure 2.5**

1. **On clicking the malicious link, Boby’s description in his profile page is changed due to the javascript code.**

****

**Figure 2.6**

****

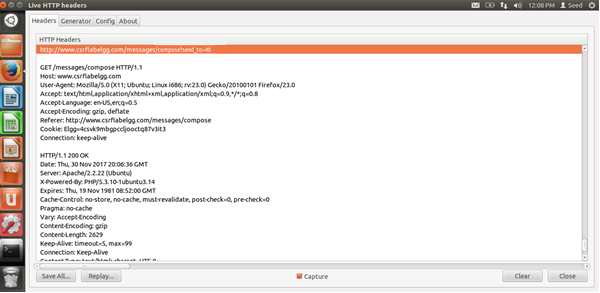
See attack without Boby’s knowledge as referrer is indexforboby.html in attaker

**Figure 2.7**

1. **The request header shown above tells that the modification that Alice wanted to make to Boby’s page is appended to the page and executed by the action request. The POST request requires javascript to attack the Boby’s profile.**

**Question 1:**

**The forged HTTP request needs Boby’s user id (guid) to work properly. If Alice targets Boby specifically, before the attack, she can find ways to get Boby’s user id. Alice does not know Boby’s Elgg password, so she cannot log into Boby’s account to get the information. Please describe how Alice can find out Boby’s user id.**

****

**Figure 2.6**

**Answer:**

**Alice needs to look at the HTTP Header to find Boby’s unique id when she tries to send him a message as shown above. She can then use this obtained unique id in the forged HTTP request to make the modification to Boby’s “About me” section.This is done when Boby opens the link that she sends him.**

**Question 2:**

**If Alice would like to launch the attack to anybody who visits her malicious web page. In this case, she does not know who is visiting the web page before hand. Can she still launch the CSRF attack to modify the victim’s Elgg profile? Please explain.**

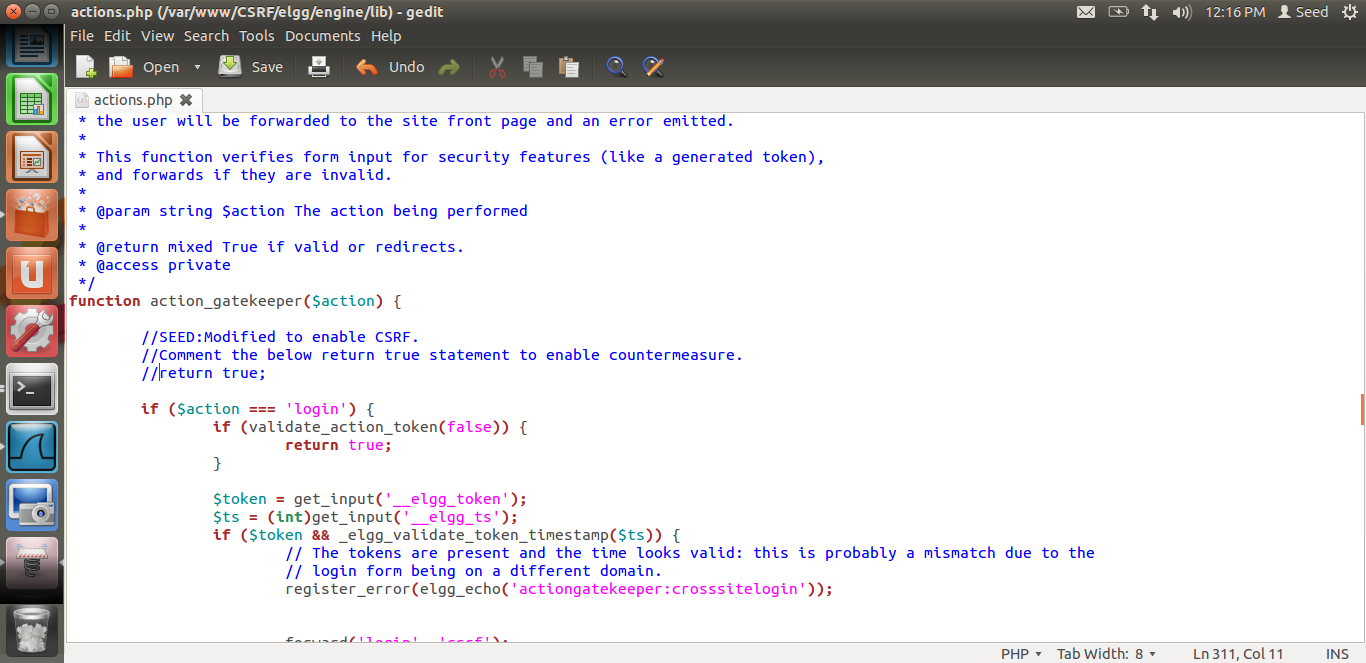
**Answer:**

* **If Alice wants to launch the attack to anybody who is visiting her page, without knowing who is visiting her page, it is not possible.**
* **The reason it is not possible is that to attack any given user, Alice requires the unique id of that user.**
* **Without knowing who is visiting that page, she cannot attain the user id of a user.**
* **In case of Boby, , by capturing the live HTTP request header , she manually figures out the unique id for Boby. As seen in the above diagrams , the id of the user being attacked needs to be grabbed by initiating requests on the profile of the victim.**
* **By using the technique in Task 1, since edit.php accepts both GET and POST requests, Alice can add an img tag to her website that launches this attack,.**
* **Although she has done this, dynamically getting the unique id of the victim visiting her page is not possible.**
* **Therefore she cannot launch the attack on anybody without knowing who is visiting her page.**

**Task 3:**

1. **In the action\_gatekeeper($action) function in the actions.php file the countermeasure for CSRF attacks is present.**

**We ensure the execution of the function whenever a user action is required on any given page by commenting the return true line.**

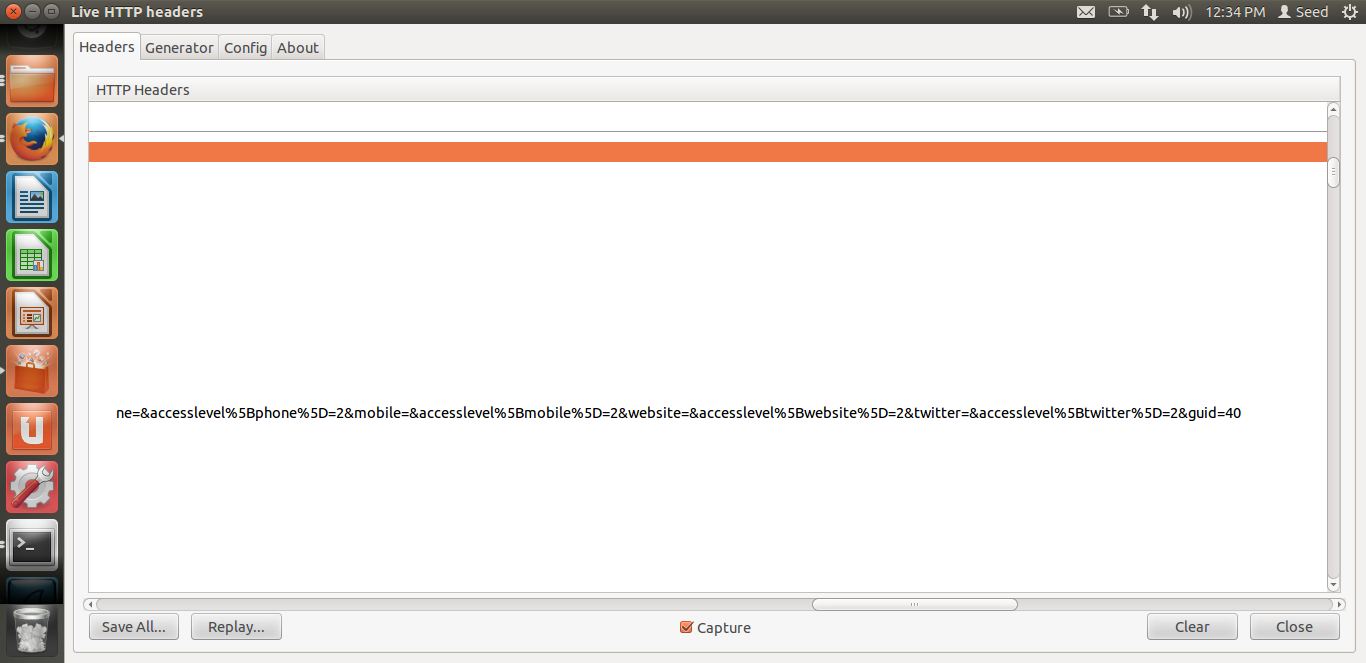
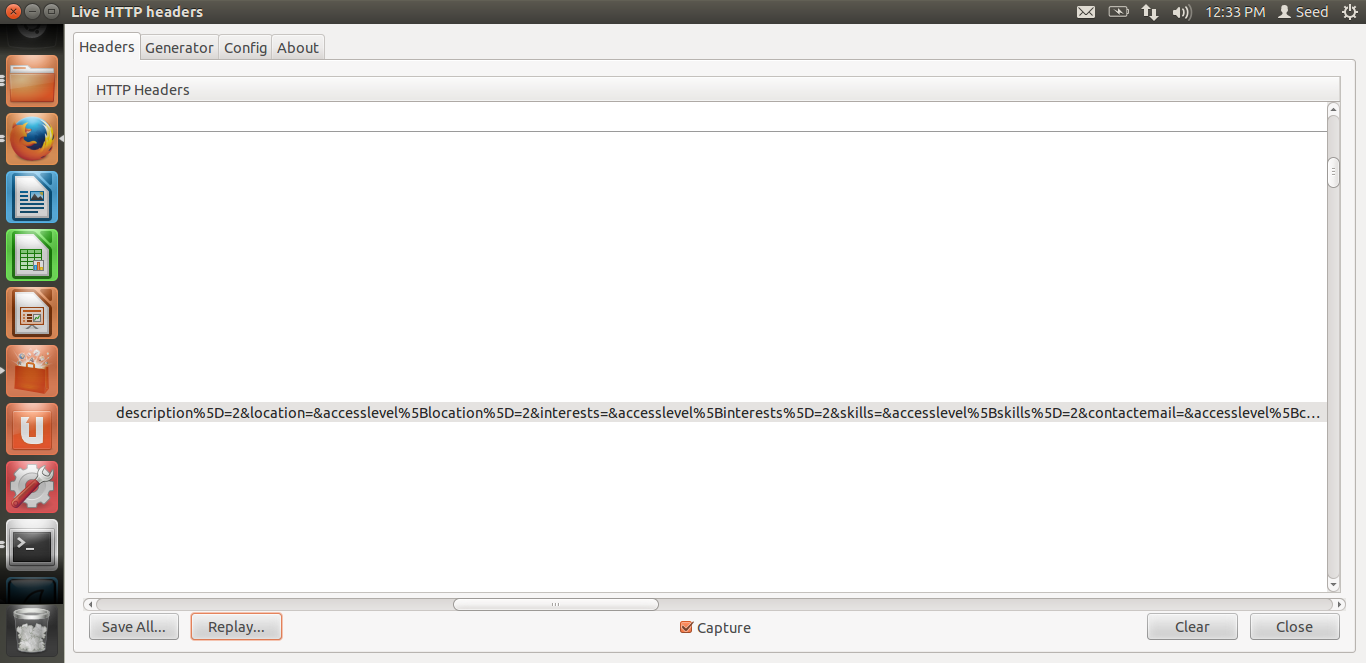
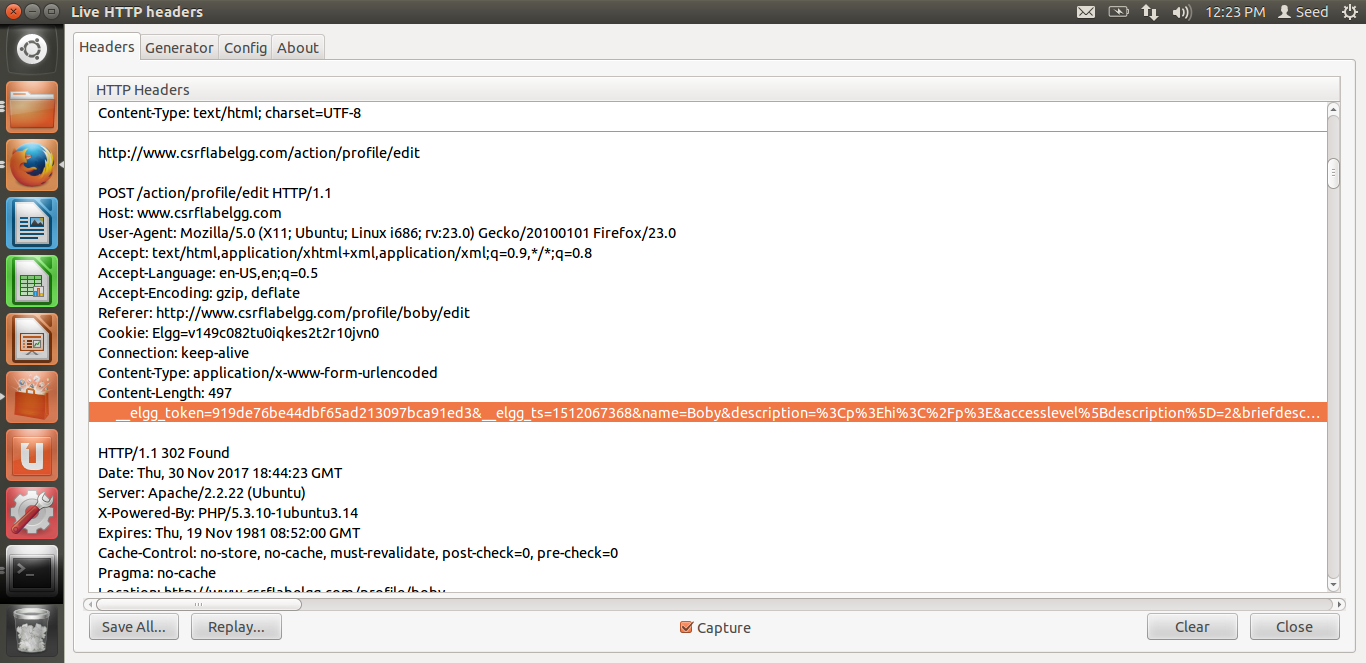
****

Since return true is commented=countermeasure activated!

**Figure 3.1**

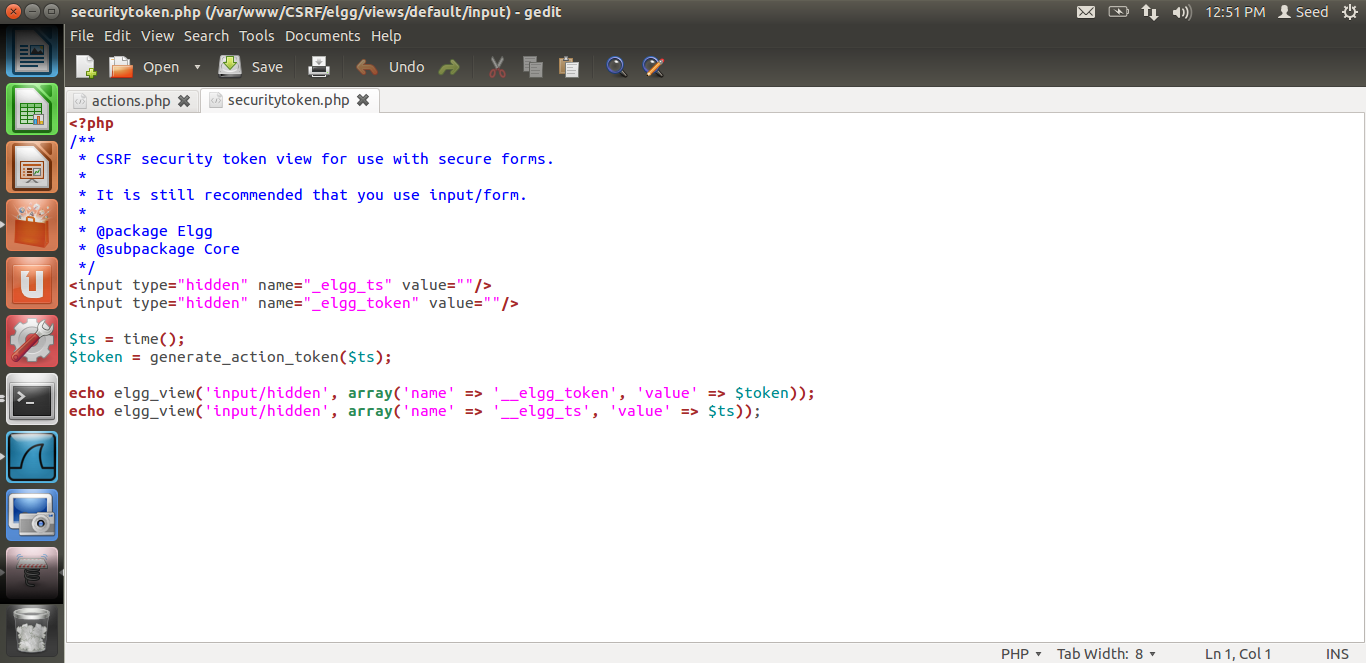
1. **In general, Elgg adds security token and timestamp to all the user actions to be performed.**

**This is performed by the views/default/input/securitytoken.php module.**

****

Token and time stamp is only generated by the real website. In forged HTTP request this won’t be present.

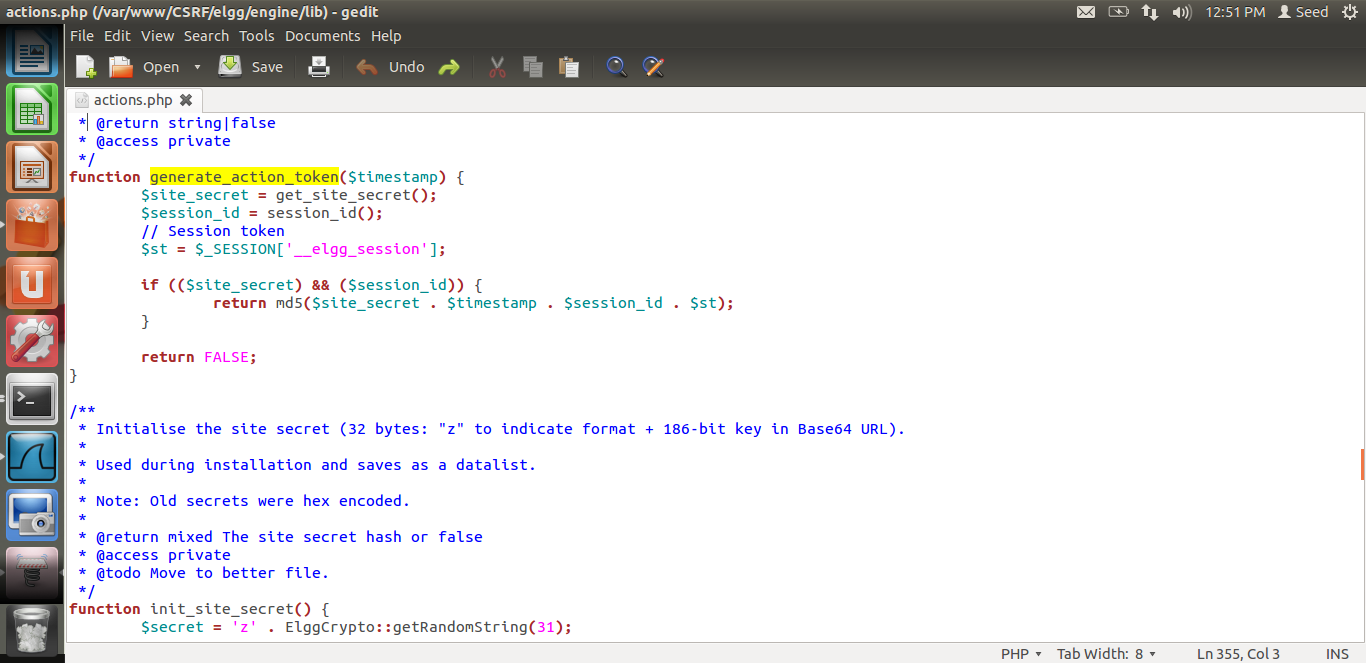
**Figure 3.2**

****

**Figure 3.3**

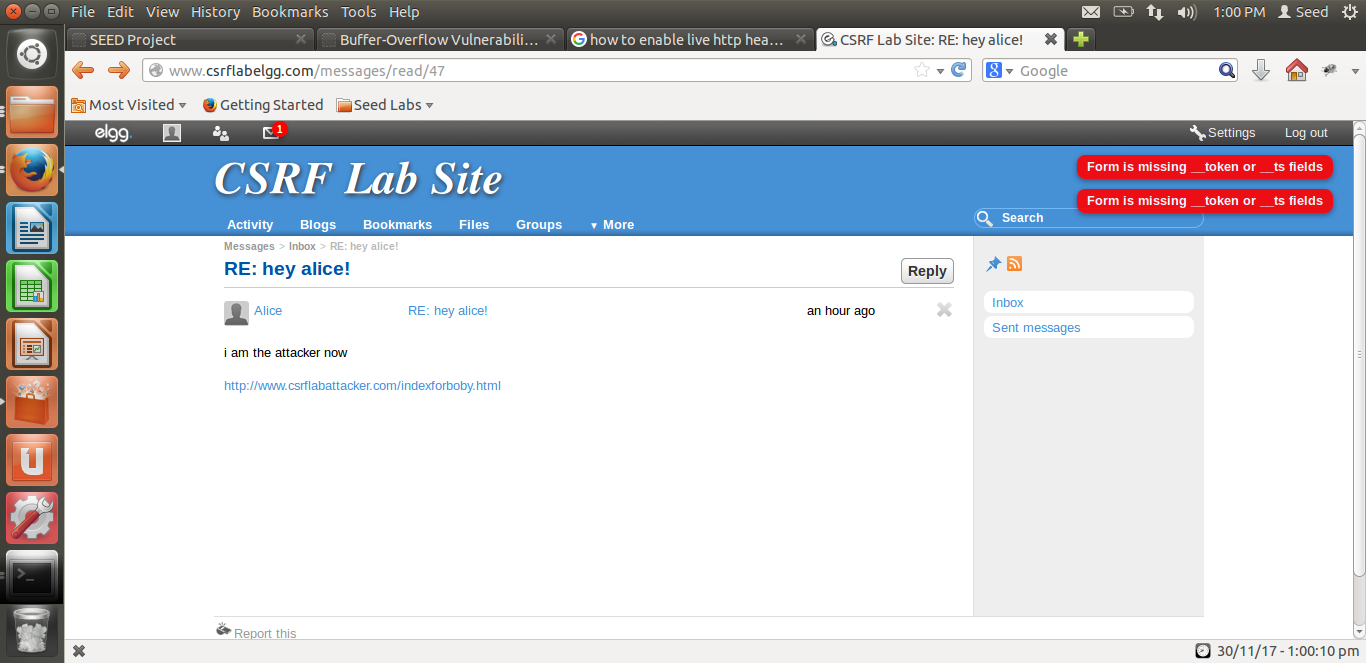
1. **A hash value (md5) of the site secret value(retrieved from the database), timestamp, user SessionID and the randomly generated sessions string is the Elgg security token .**

**The generate\_Action\_token($timesamp) function generates the secret token as shown below.**

****

**Figure 3.4**

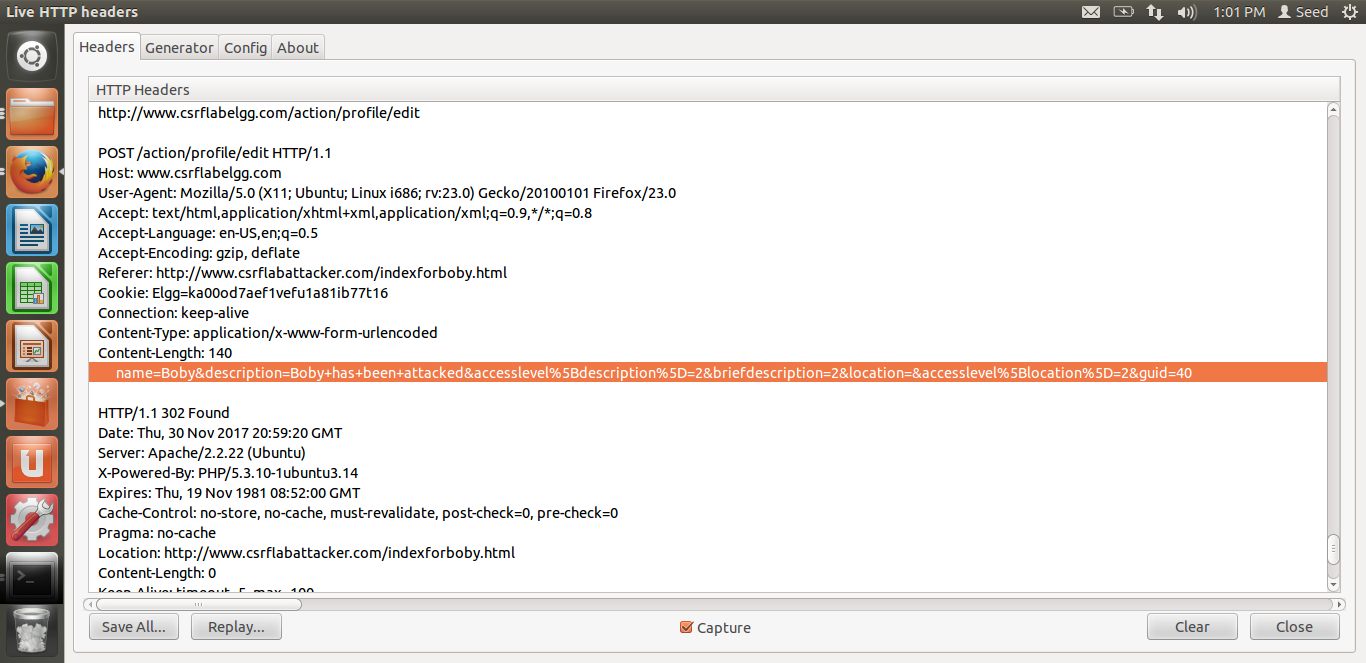
1. **The function validate\_action\_taken does secret-token validation.If the tokens are not present or invalid then the action will be denied and the user will be redirected.**

****

Notice absent token and timestamp-noticed by countermeasure on the forged POST action.

**Figure 3.6**

1. **The secret token is generated by the website that actually hosts the session, in our case this website is elgg.**
2. **Because of this the attacker website cannot successfully generate the right token, as it doesn’t have all the details to generate this token and get it validated by the website being attacked.**
3. **This attack will fail since the attacker is unable to place the correct tokens in his request.**
4. **The tokens are generated as a result of php code, source of which is unavailable for the attacker to grab, this acts as added security to the website.**
5. **The countermeasure will detect that it is a forged HTTP request**.

****

Token and timestamp missing from HTTP request Header.

**Figure 3.7**