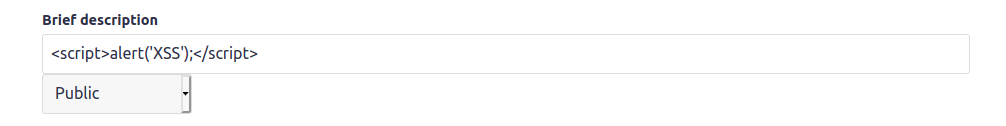
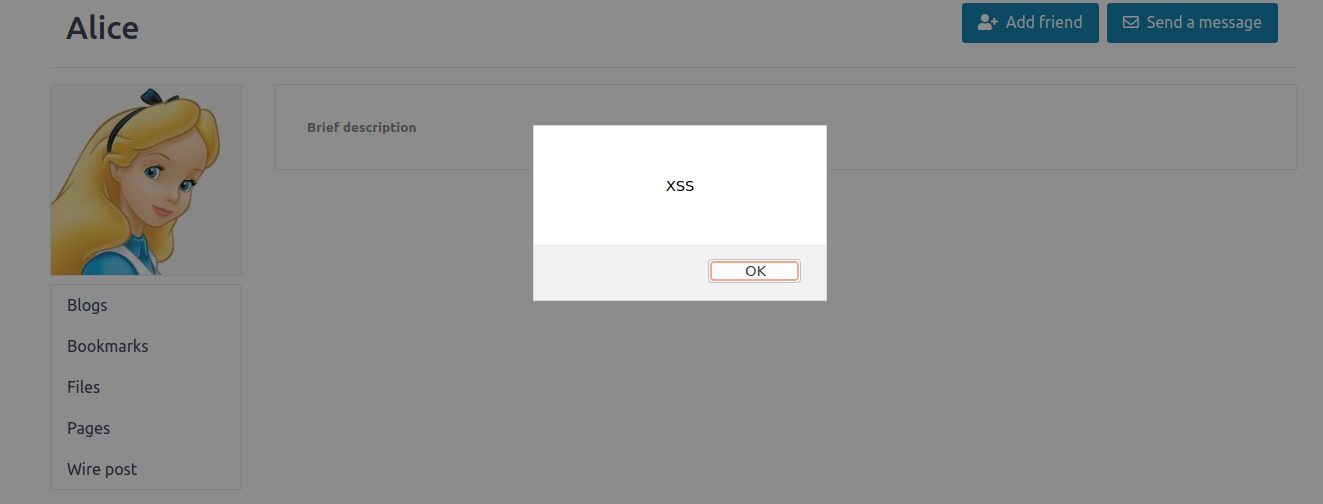
Cross-Site Scripting (XSS) Attack Lab on Elgg WebApp - Fatjon Freskina

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

After modifying the brief description field in Alice’s profile, I was able to display an alert every time a user visited that profile. This happens because the brief description field in the WebApp does not implement any input sanitization and a malicious user is able to write javascript code that is treated by the victim’s browser as “legitimate” code to be run.



When a user sends an http get request to the server asking for alice’s profile resource, the code is sent and executed, generating this alert:



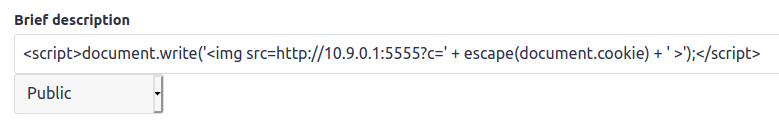
# Task 2: Posting Malicious Message to Display Cookies

The cookie code works in the same way, we are able to access the cookie attribute of the document object with no big effort (‘document.cookie’):



# Task 3: Stealing Cookies from the Victim’s Machine

In order to receive the cookies, the attacker can open a tcp connection between the victim’s and his machine by sending an http get request to his machine attaching the cookies. This is done using html and Javascript: in fact when the browser encounters the img tag parses the url and open a get request for that image, appending the escaped version of the cookies.

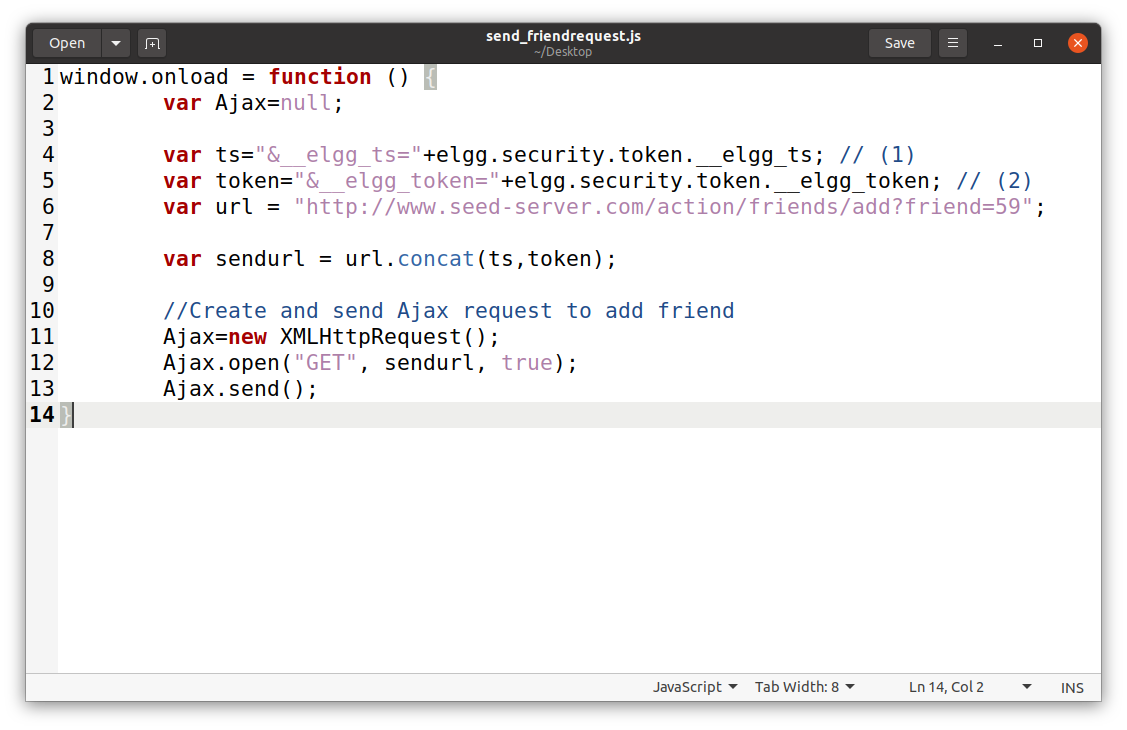




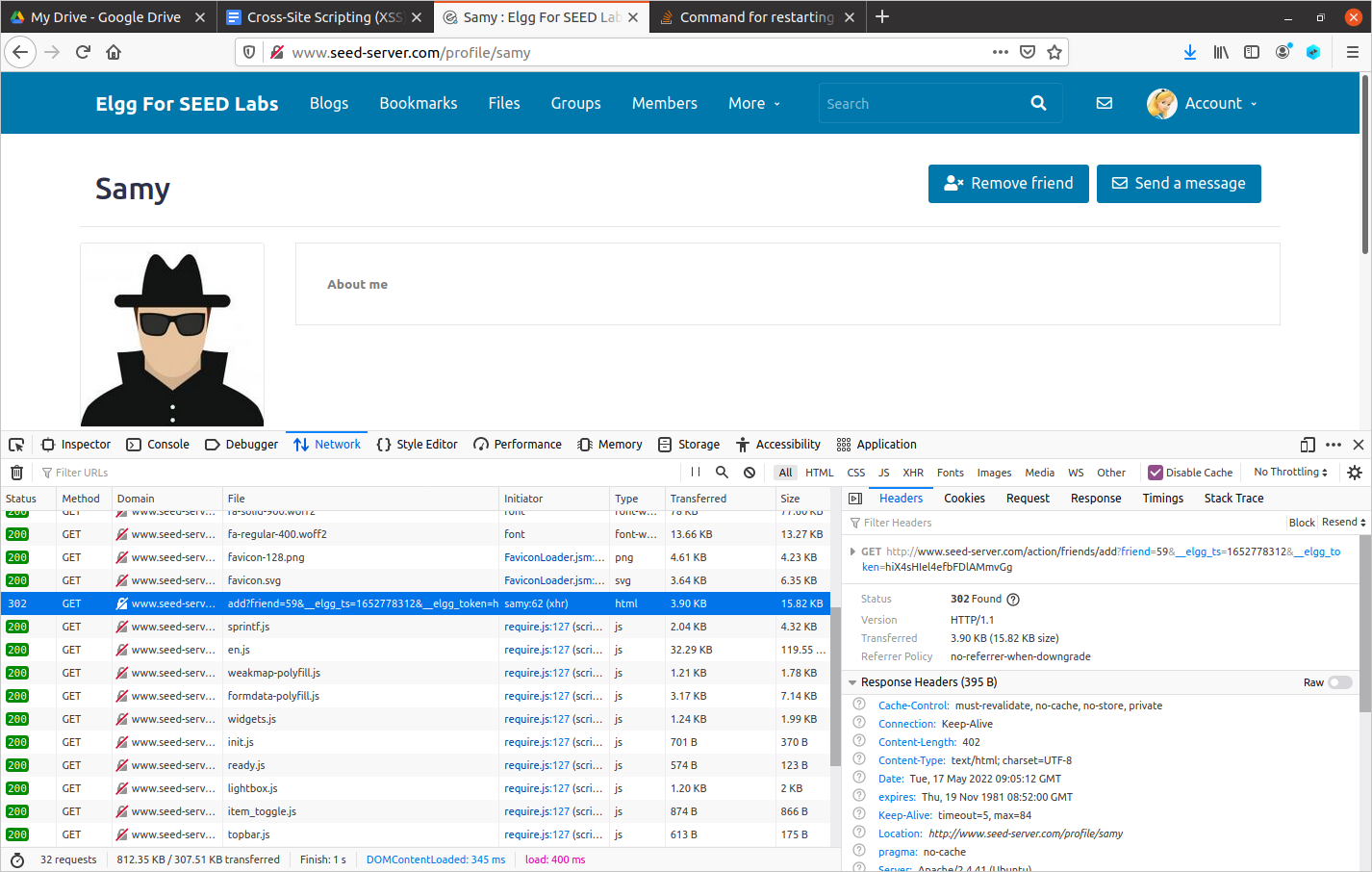
By listening on port 5555 (using netcat: *nc -lknv 5555*) from the attacker’s machine, we can see the incoming http request with it’s headers: method, host, user-agent, accept, language etc…

# Task 4: Becoming the Victim’s Friend

By inspecting the Get http request generated by a legitimate “add friend request” to samy’s profile, I could see the parameters of the request.



By posting this code inside the script tag in the about me section of samy’s profile, every user that visits this page will execute an http get request to the server basically adding samy as a friend.

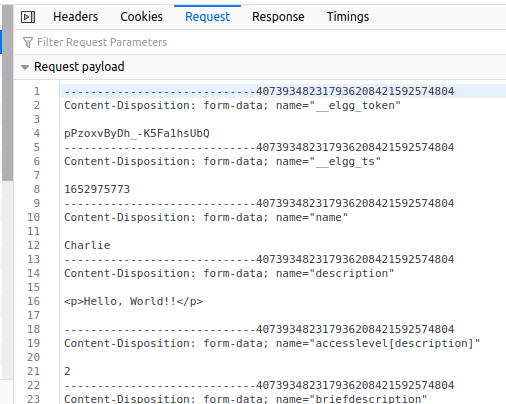


This request was indeed sent automatically when visiting samy’s profile from Alice’s one.

Questions.

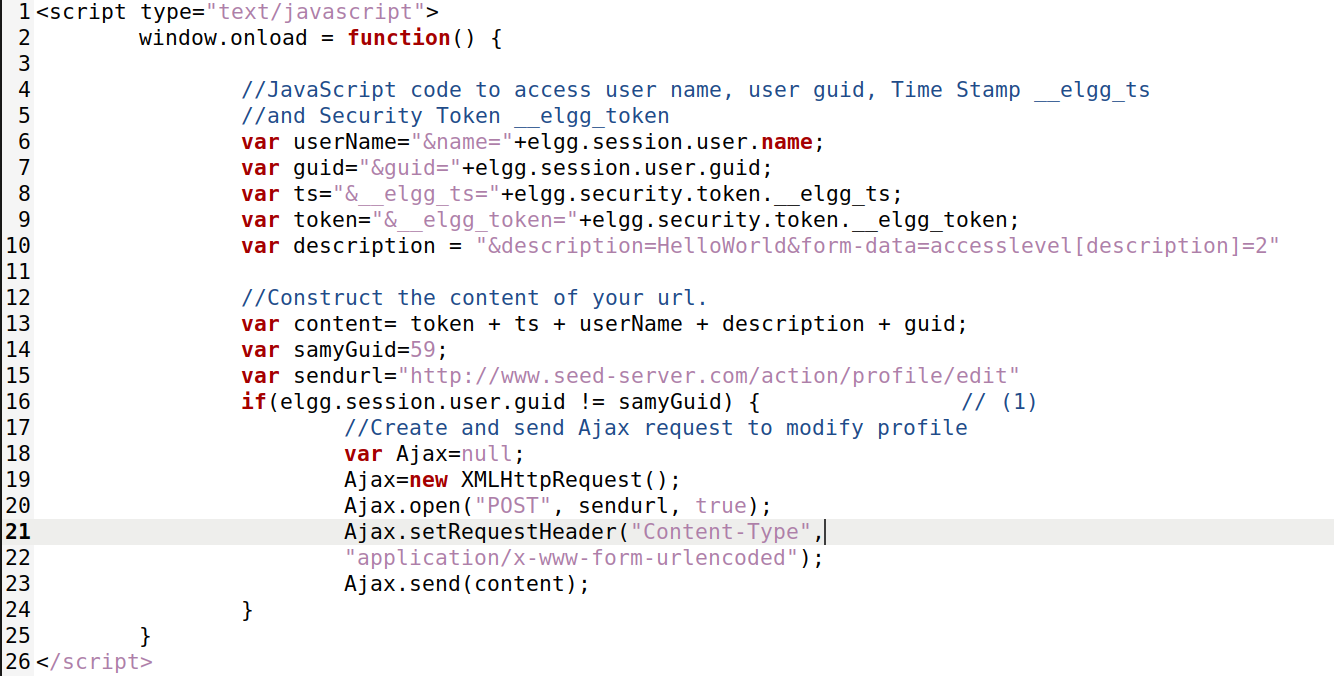
* Line 4 and 5 of the code are security measures to avoid cross site request forgery, they represent the timestamp and a security token. They need to be included in the http get request, but since they are saved inside a variable, they are easy to access.
* If the Elgg application only provided the Editor mode for the About me section, we could try to escape the controls in someway, or try to put our code in different fields. Since the description field as we saw before was vulnerable, we could try to inject there a js program which actually refers to another program (we can’t post directly the main one since it is too long)o

# Task 5: Modifying the Victim’s Profile



By modifying charlie’s profile, I could see what a post request looks like.

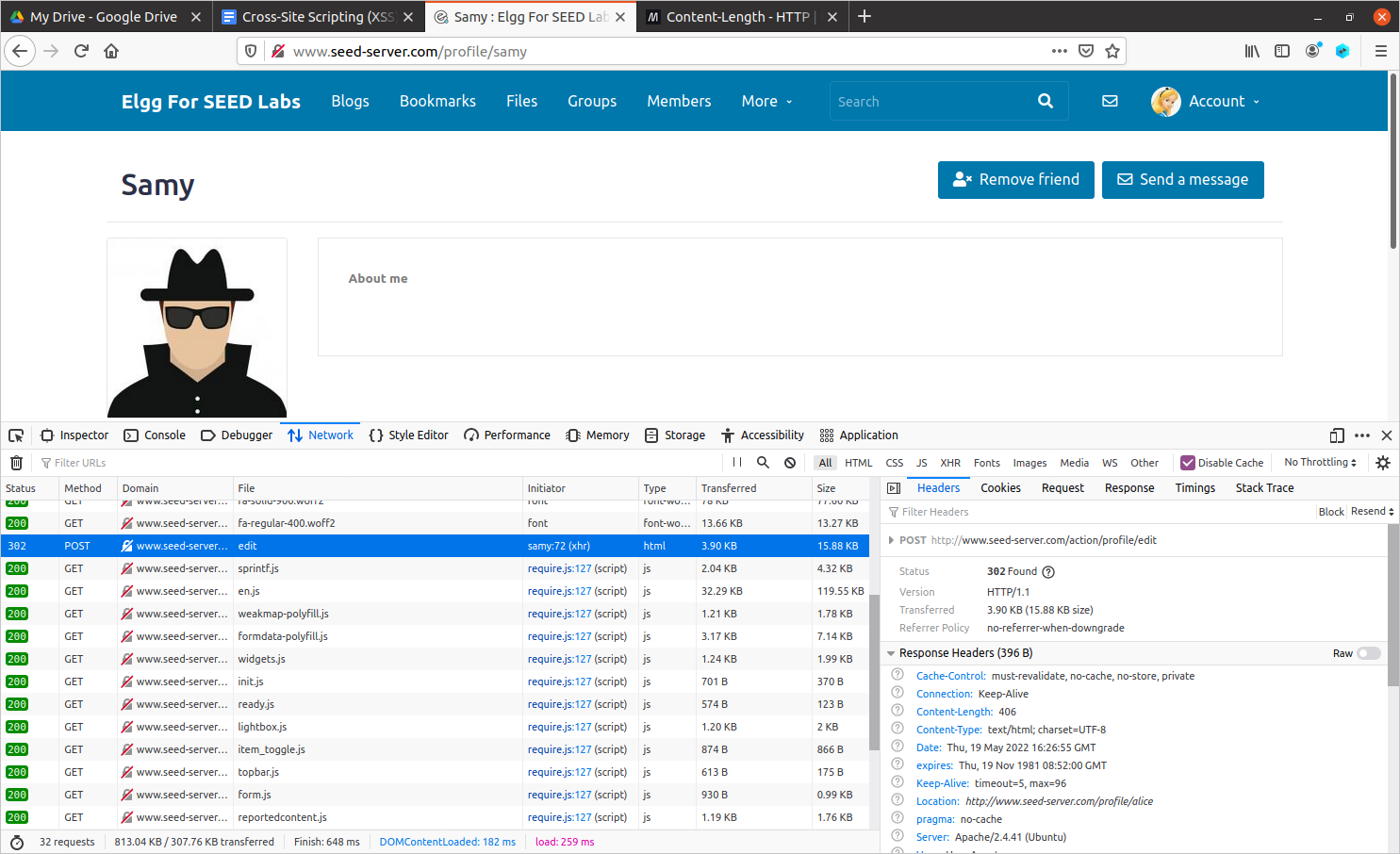
We now need to concatenate these fields in order to send a legitimate post request to modify our own profile.

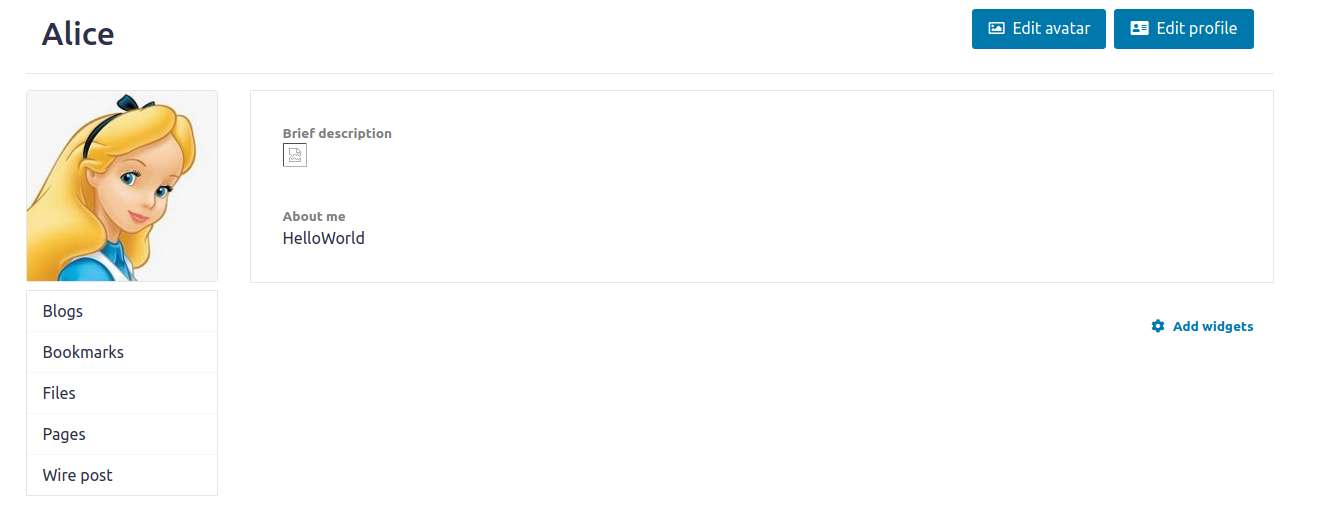


The description variable adds the information needed to modify the about me section.

The samyGuid is used to not check samy’s own profile, since it would delete the code used to infect other profiles.

By visiting samy’s profile with alice’s one, we can see that a post request is sent and the profile is successfully modified.

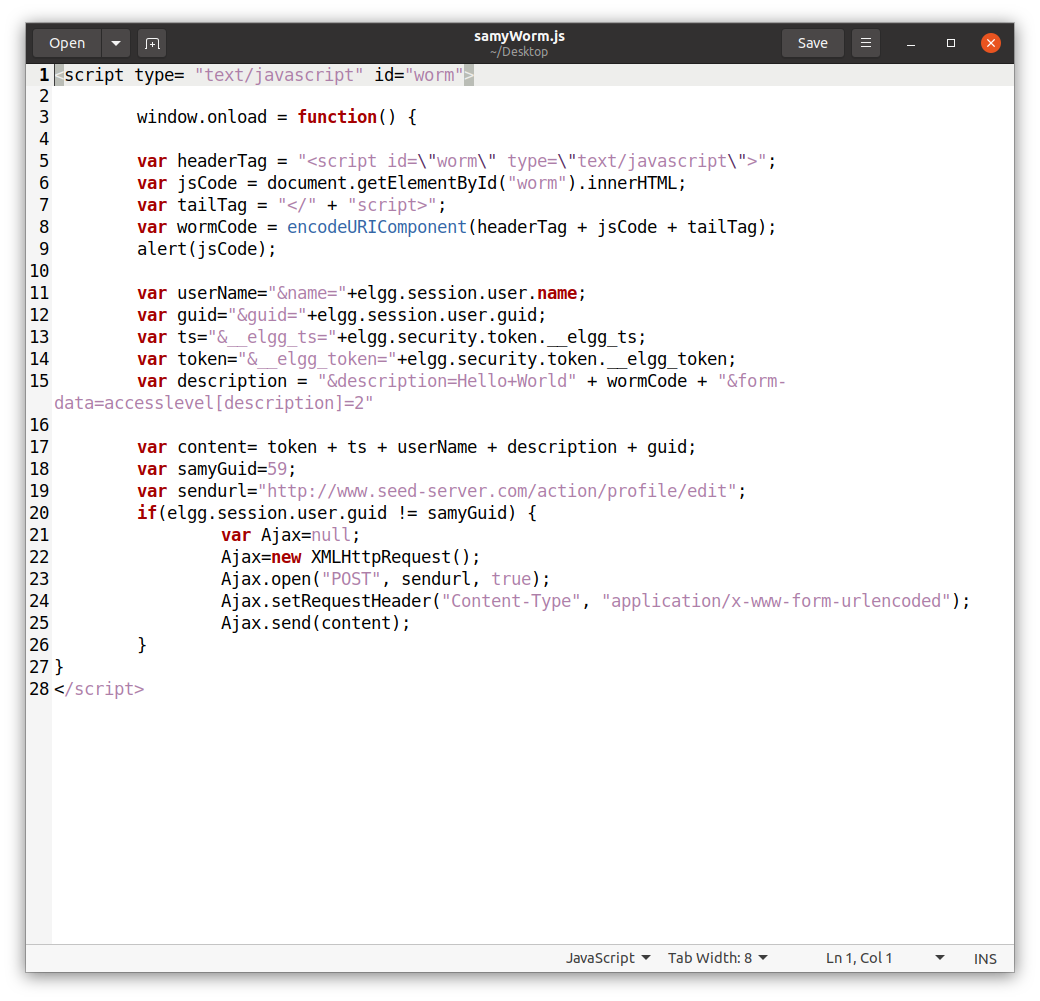




# Task 6: Writing a Self-Propagating XSS Worm

## DOM Approach

Code:



We basically did the same thing as in the previous task, but now we added in the description field the script itself (not only the HelloWorld string but also the encoded version of the script). To access the ‘script itself’ we used the DOM API and we get the html element having the id equal to ‘worm’ (our script).

This is shown when loading Alice’s profile after visiting Samy’s one:



The alert is triggered and therefore we propagated the worm successfully.

## Link Approach

To succeed with the link approach we just need to copy-paste the script that links to the source hosting the malicious code. This will open a http get request to our server which will send a response with almost the same script we wrote before.