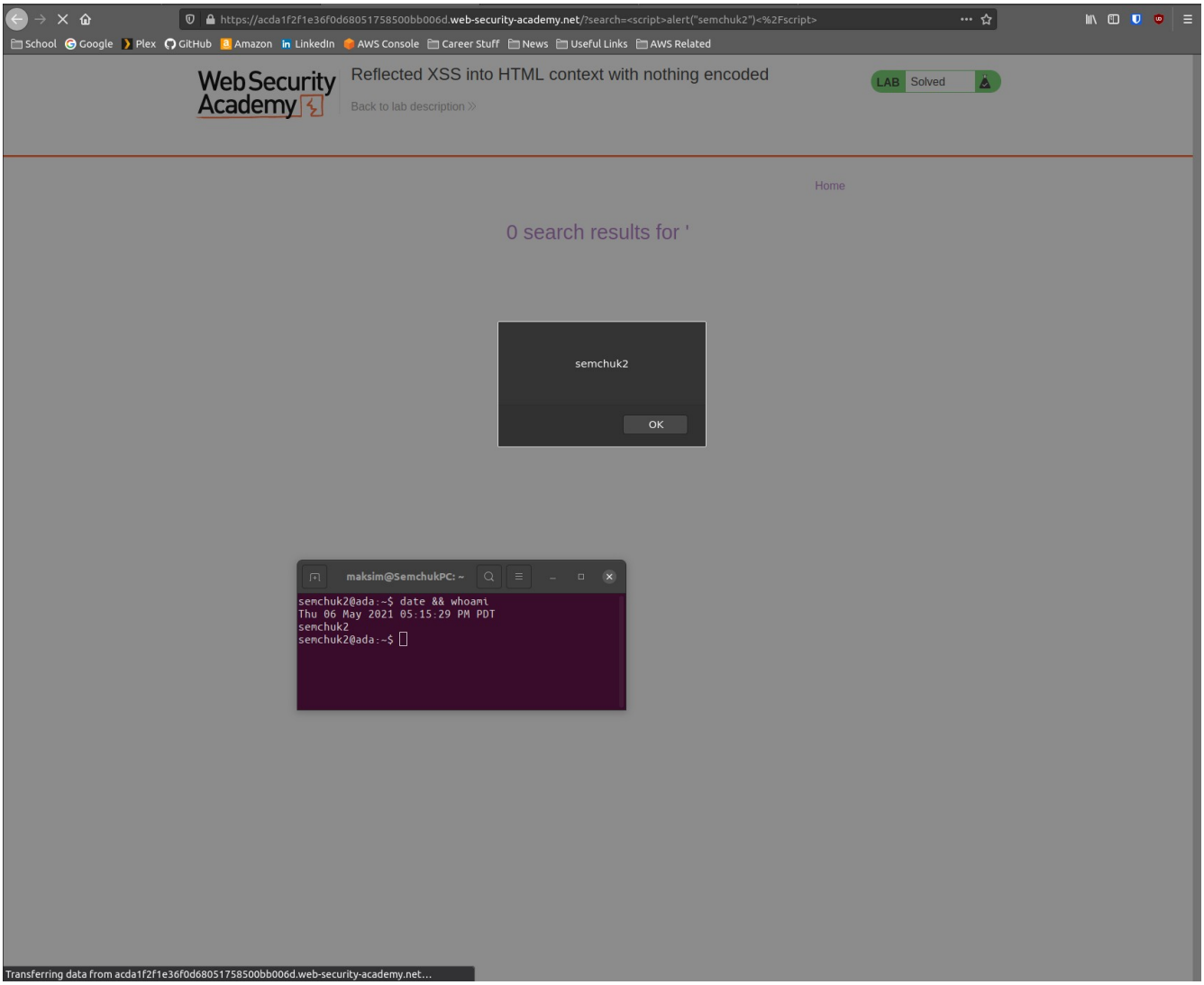
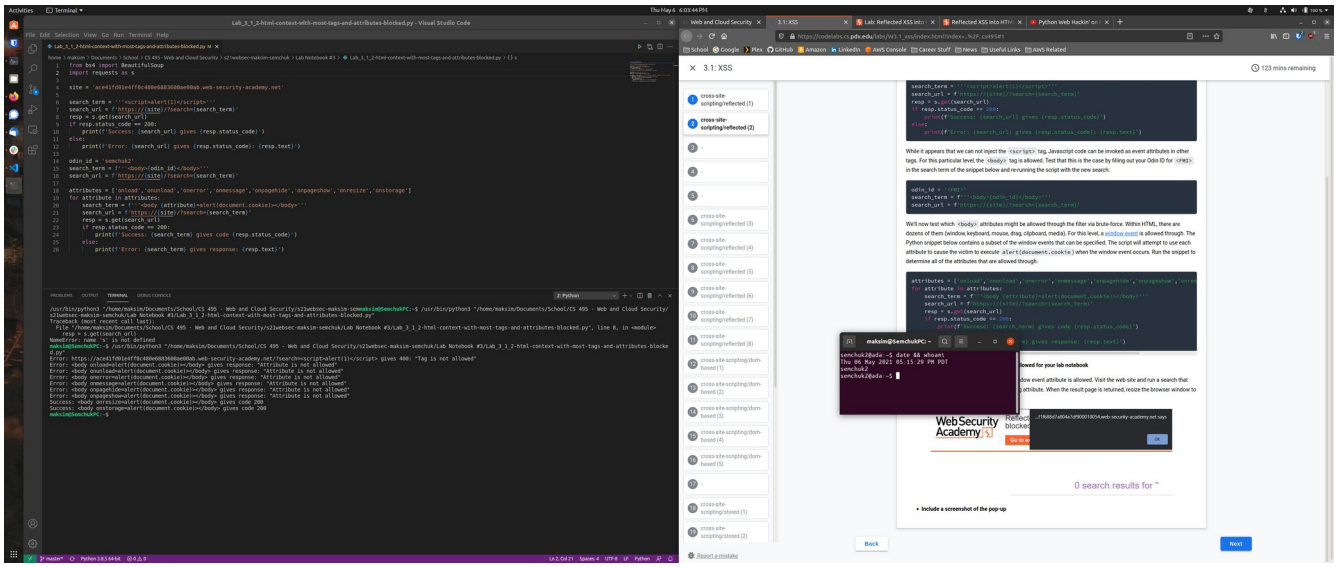
Table of Contents

Lab 3.1 html-context-nothing-encoded:

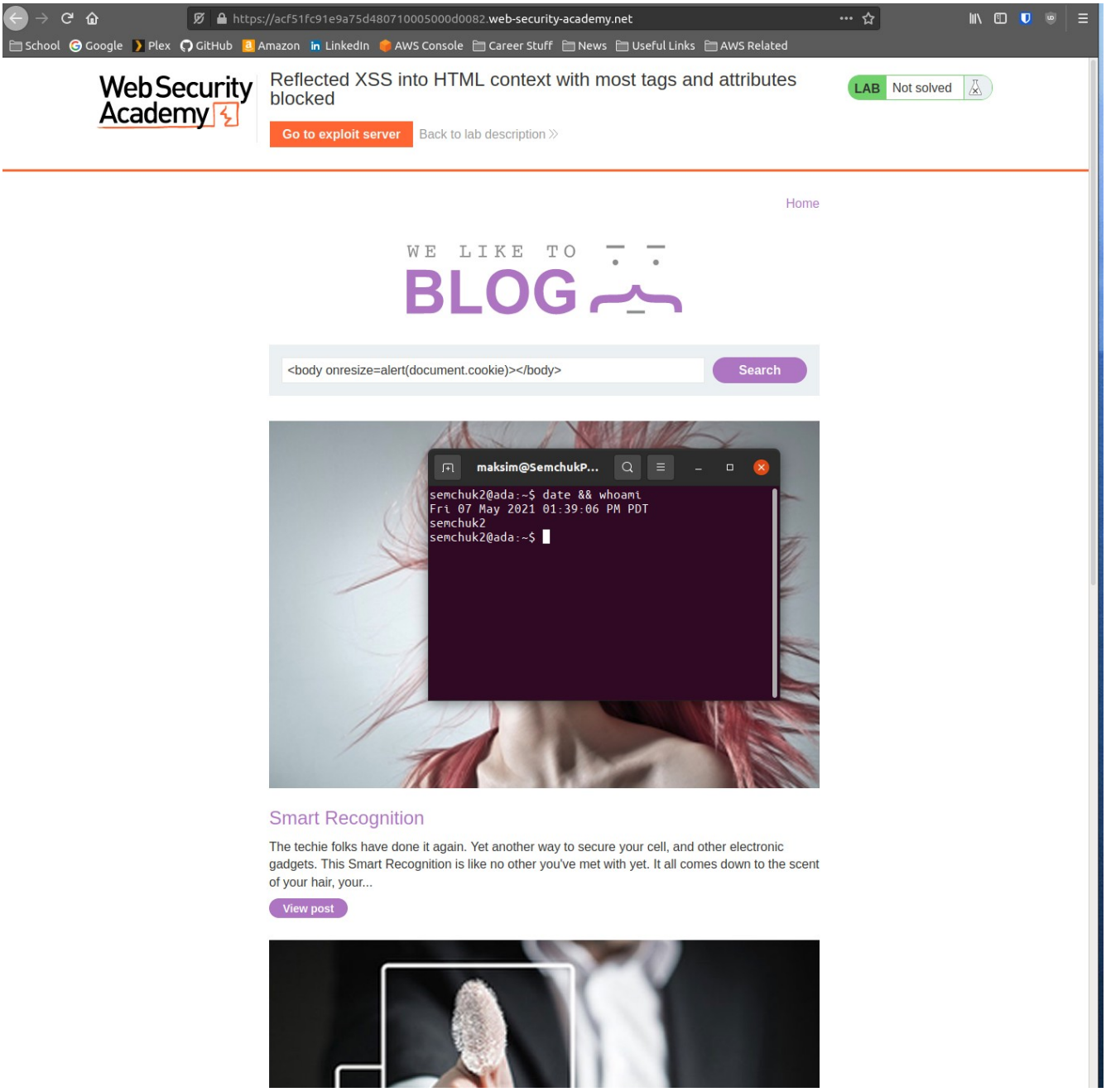


This vulnerablity comes from the fact that the user is able to pass in simple JS commands and have the remote server actually run them. This is shown by the fact that the web server displays an alert with my username.

Lab 3.1 html-context-with-most-tags-and-attributes-blocked

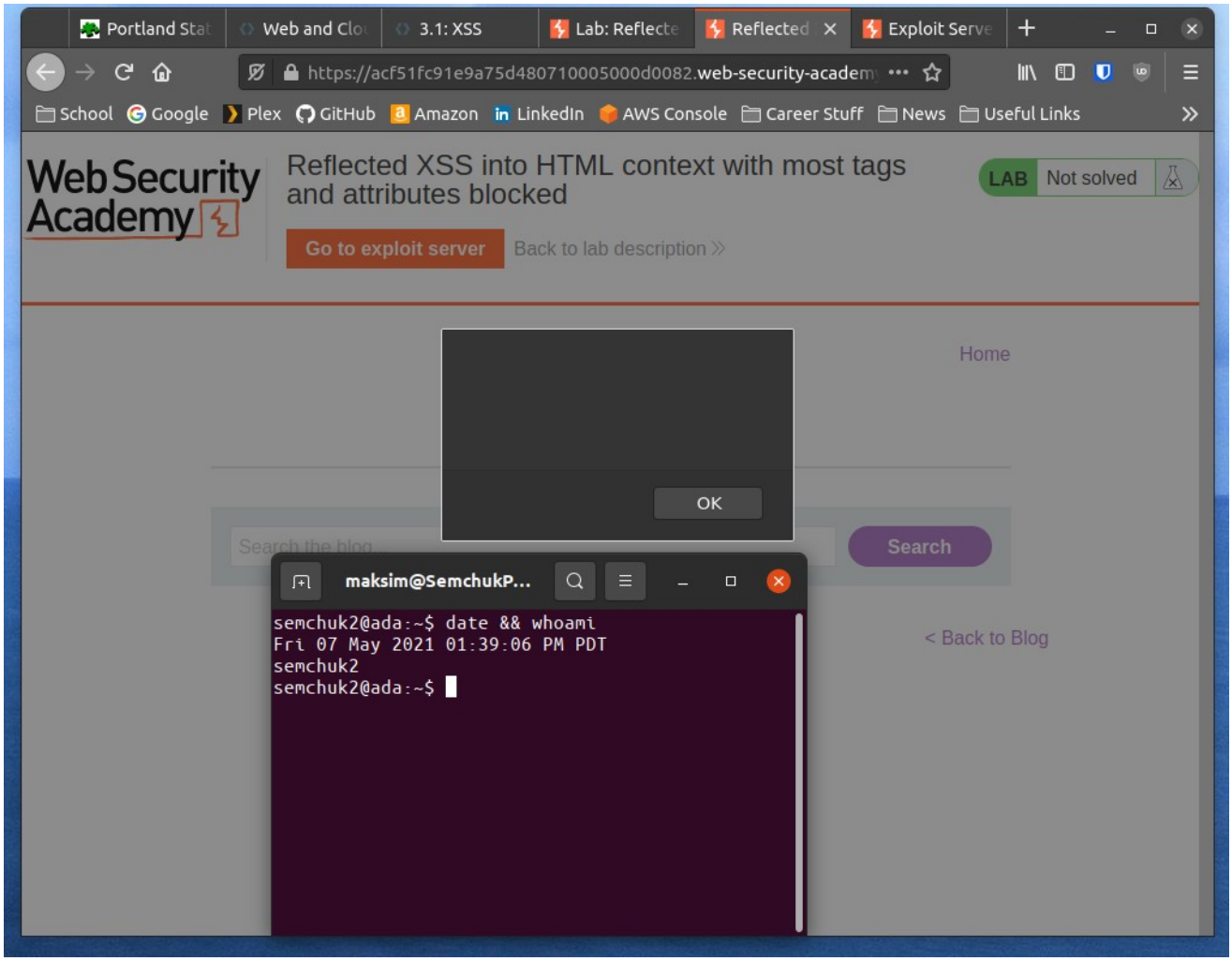


The one ones allowed are on resize and onstorage.



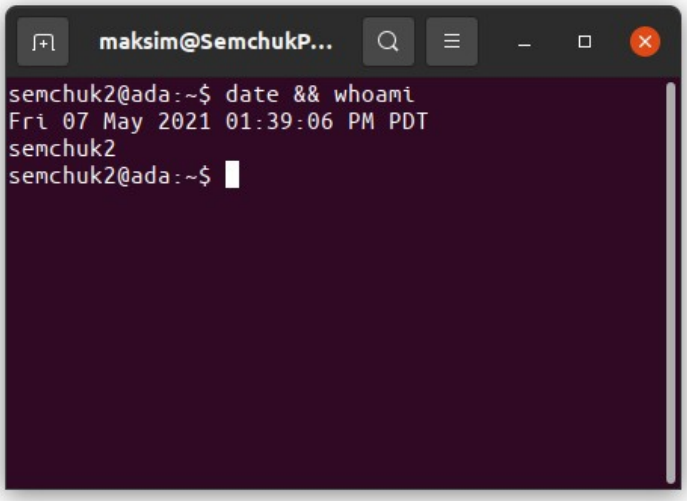
This is before resize, I searched for: <body onresize=alert(document.cookie)></body>

Therefore it should pop up a alert after I resize the window.

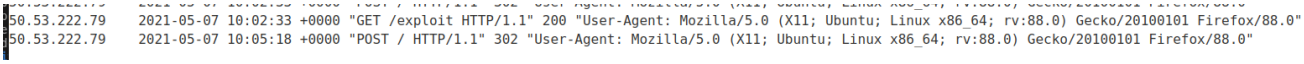


And it does resize after I searched for the exploit and resized the window.

Hello semchuk2



This is the exploit being sent to the server.

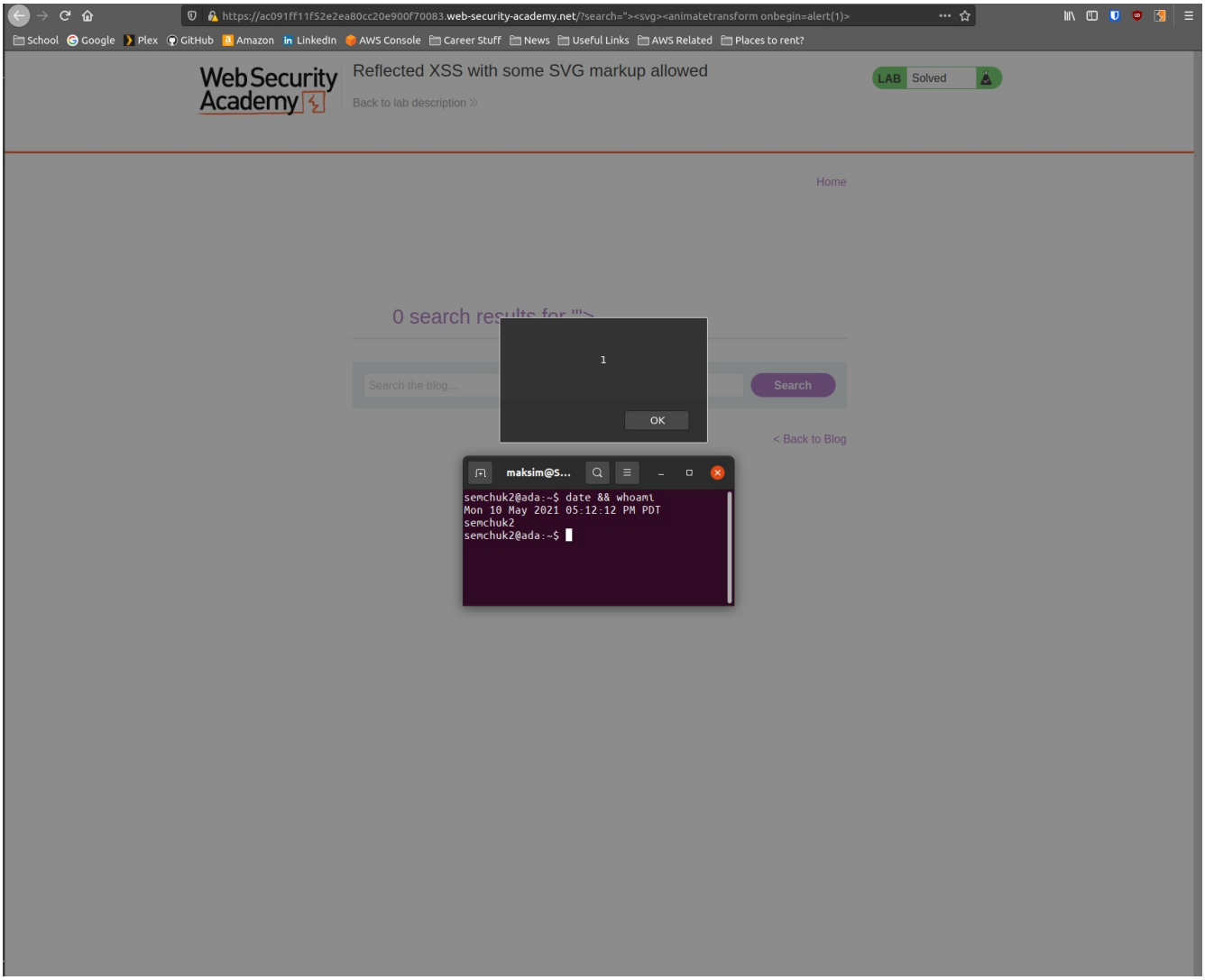


Here my my IP Address.

Here it is completed.

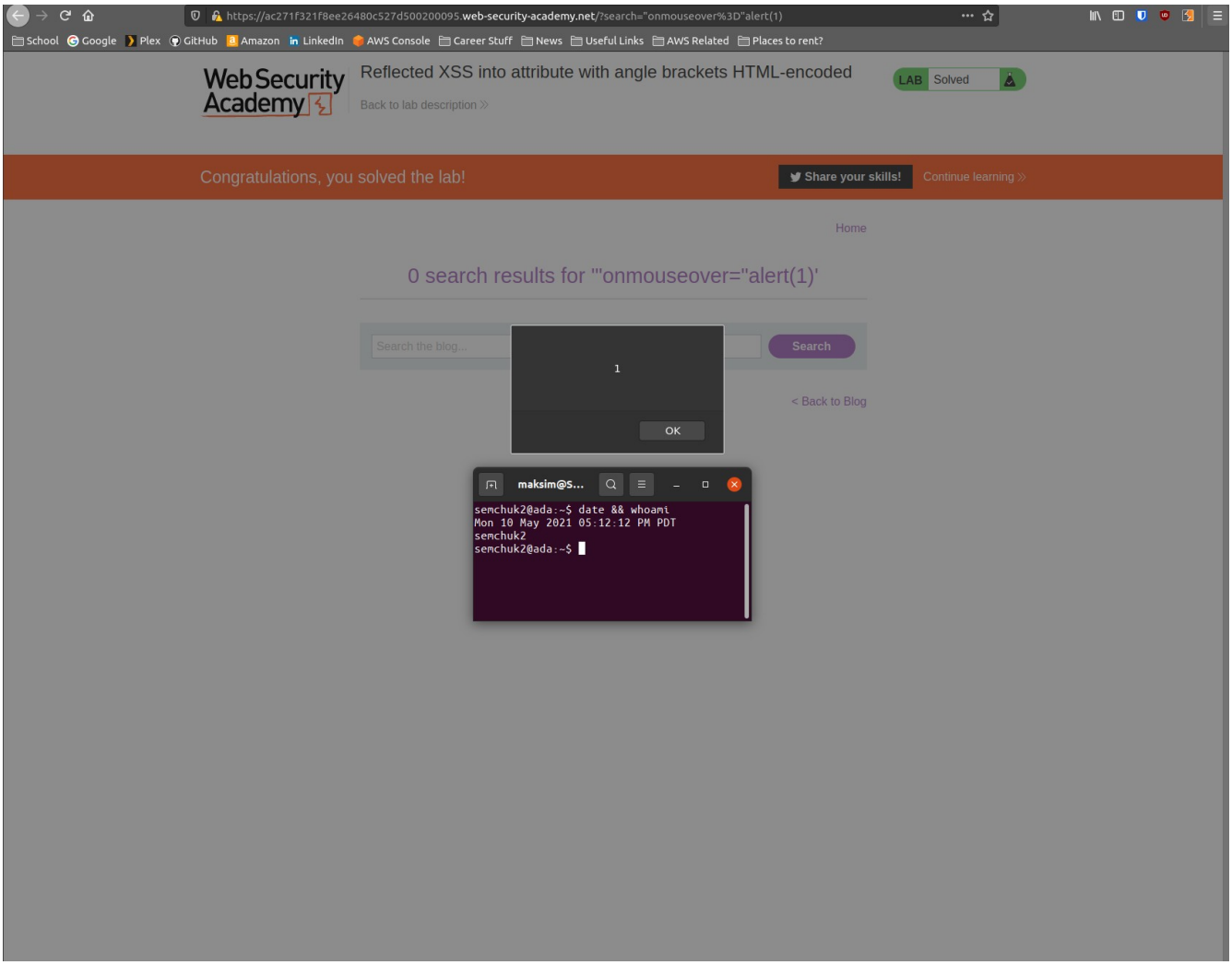
The reason why this worked, is because it does not protect body tags, and the places are execute the commands. The way to prevent this is to parse out any rags and take everything as a string, and not as a command.

3.6 some-svg-markup-allowed



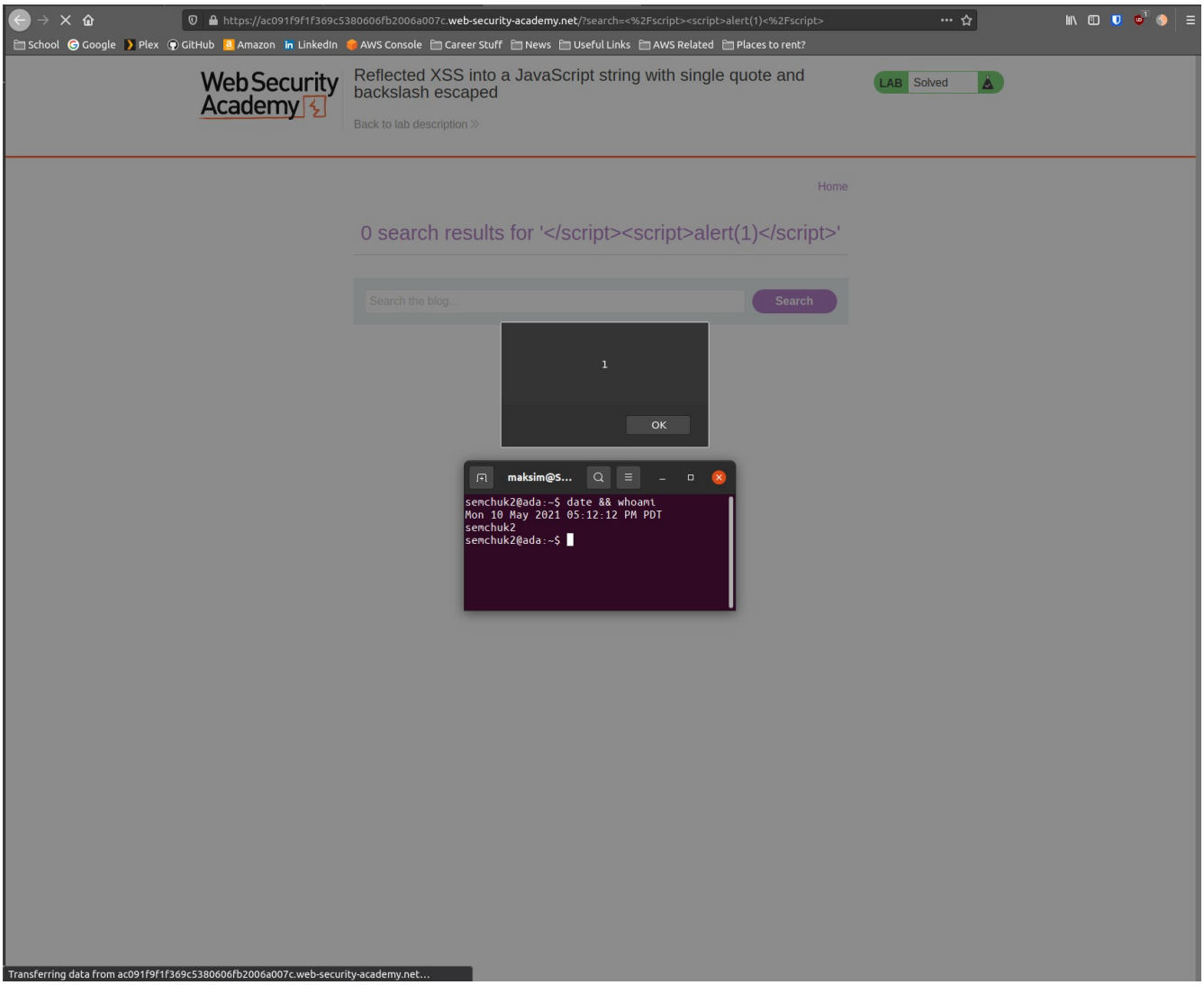
Here we are able to see that certain tags are allowed. By doing a brute force attack which tests which of the tags are allowed (meaning get a 200 response back) we are able to target those weak tags to force through an alert.

Lab 3.1.7 attribute-angle-brackets-html-encoded



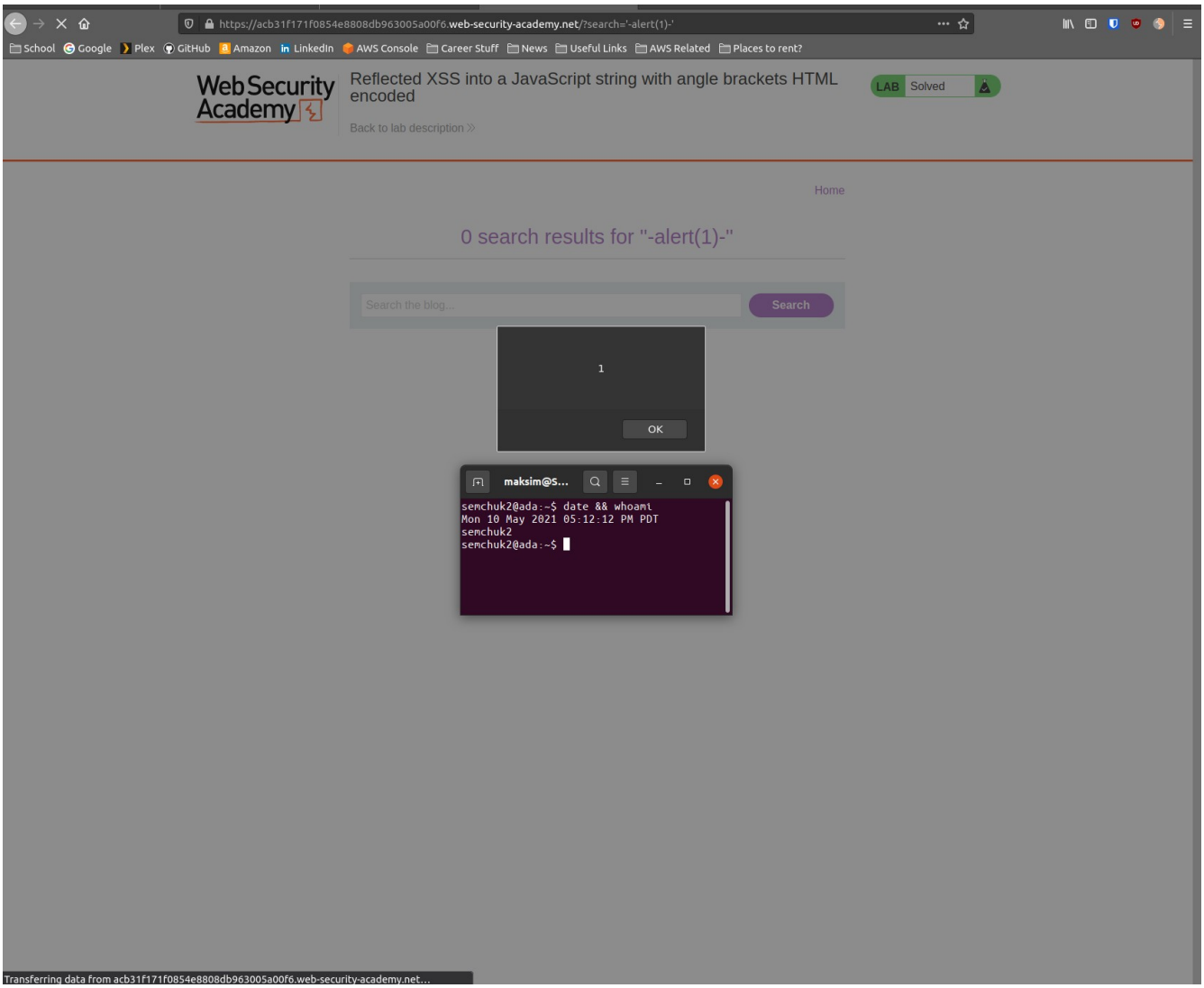
What happens here, is that since the search request is reflected, then it allows us to put the event in double quotes and have a action occur since when it get send to the reflected server it is formatted differently.

3.1.8 javascript-string-single-quote-backslash-escaped



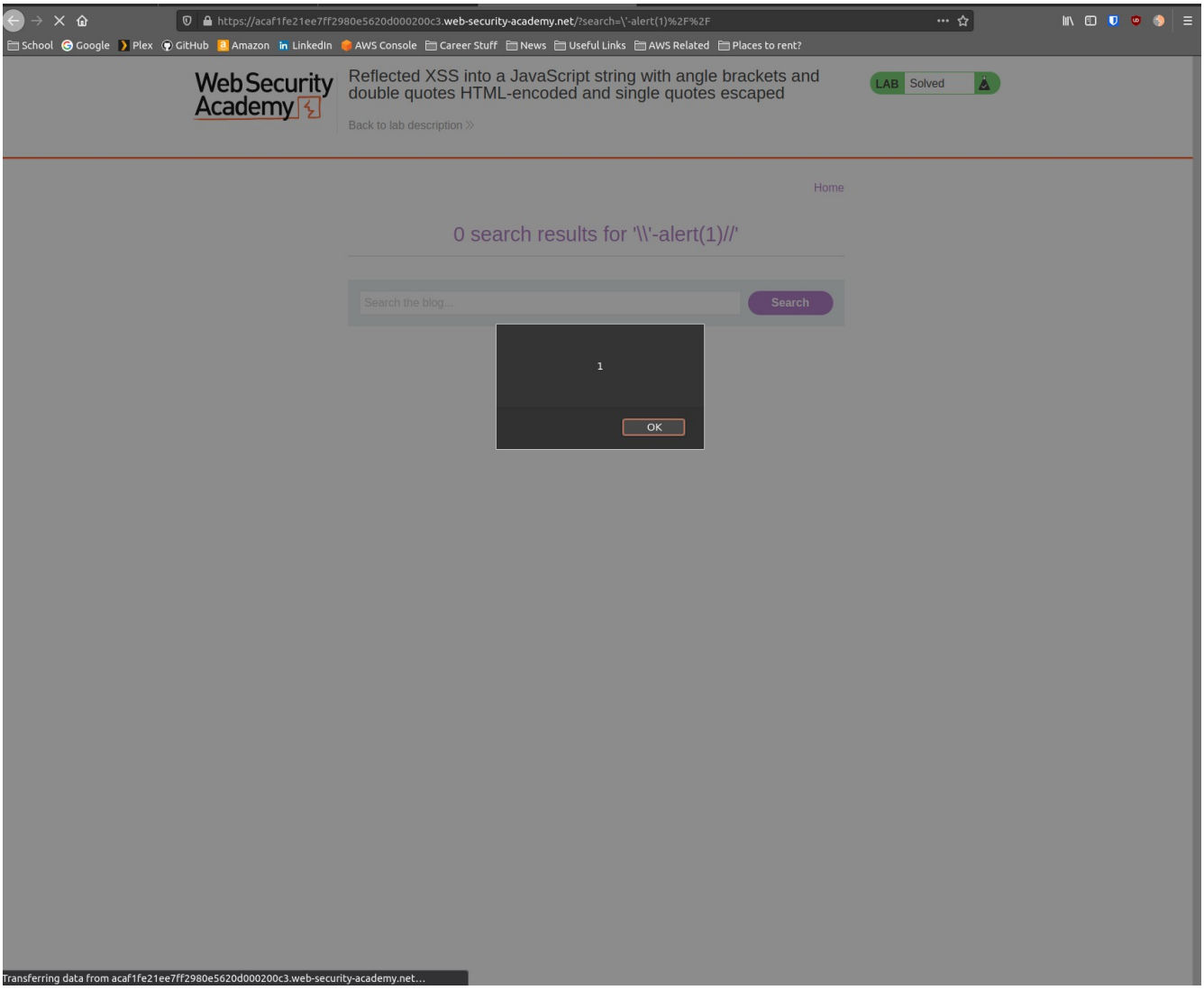
This is a vulnerablity since it allows a user to pass in a script escape followed by a script tags to then inject a command into the search bar. The function is then called by javascript.

3.1.9 javascript-string-angle-brackets-html-encoded



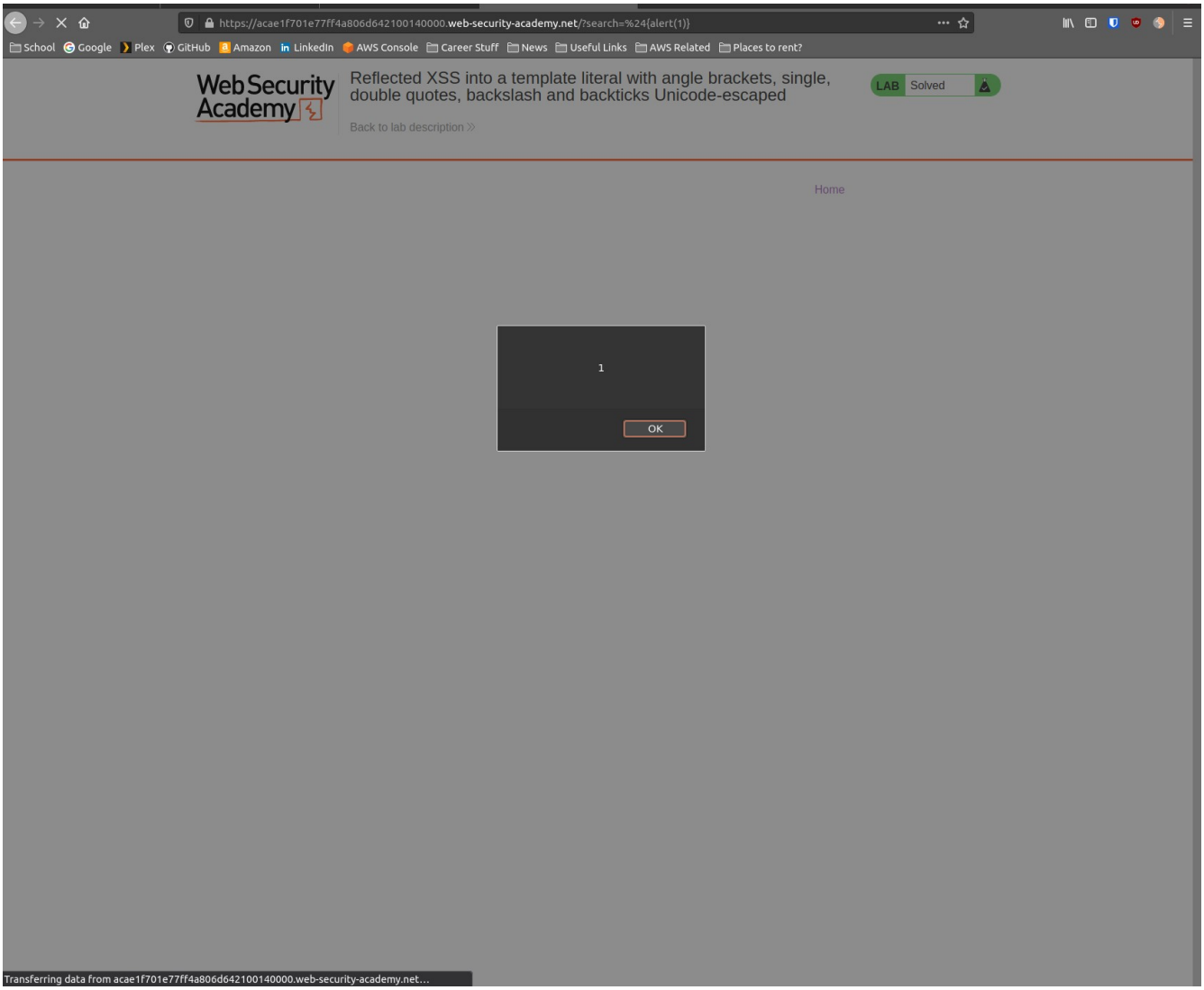
This allows a person to inject javascript into the searched bar, and run code. This allows the user to escape out of SQL, and then into java where they can call commands to do things, like an alert. This is done by having escape characters not be checked and assumed safe by the developer.

3.1.10 javascript-string-angle-brackets-double-quotes-encoded-single-quotes-escaped



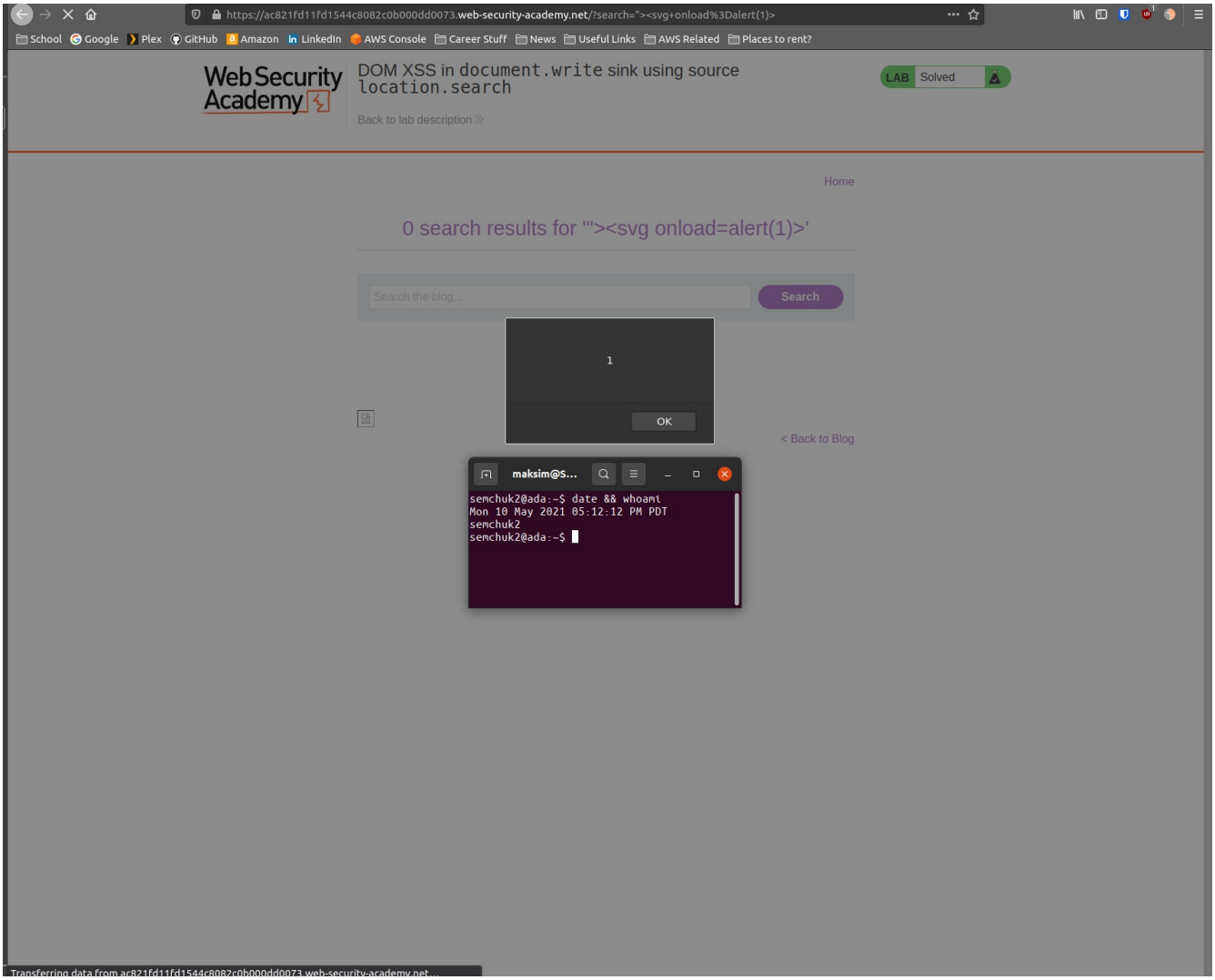
This vulnerablity is due to the fact that the HTML escape characters can be escaped and then are once again injected into the Javascript and run as a command.

3.1.11 javascript-template-literal-angle-brackets-single-double-quotes-backslash-backticks-escaped



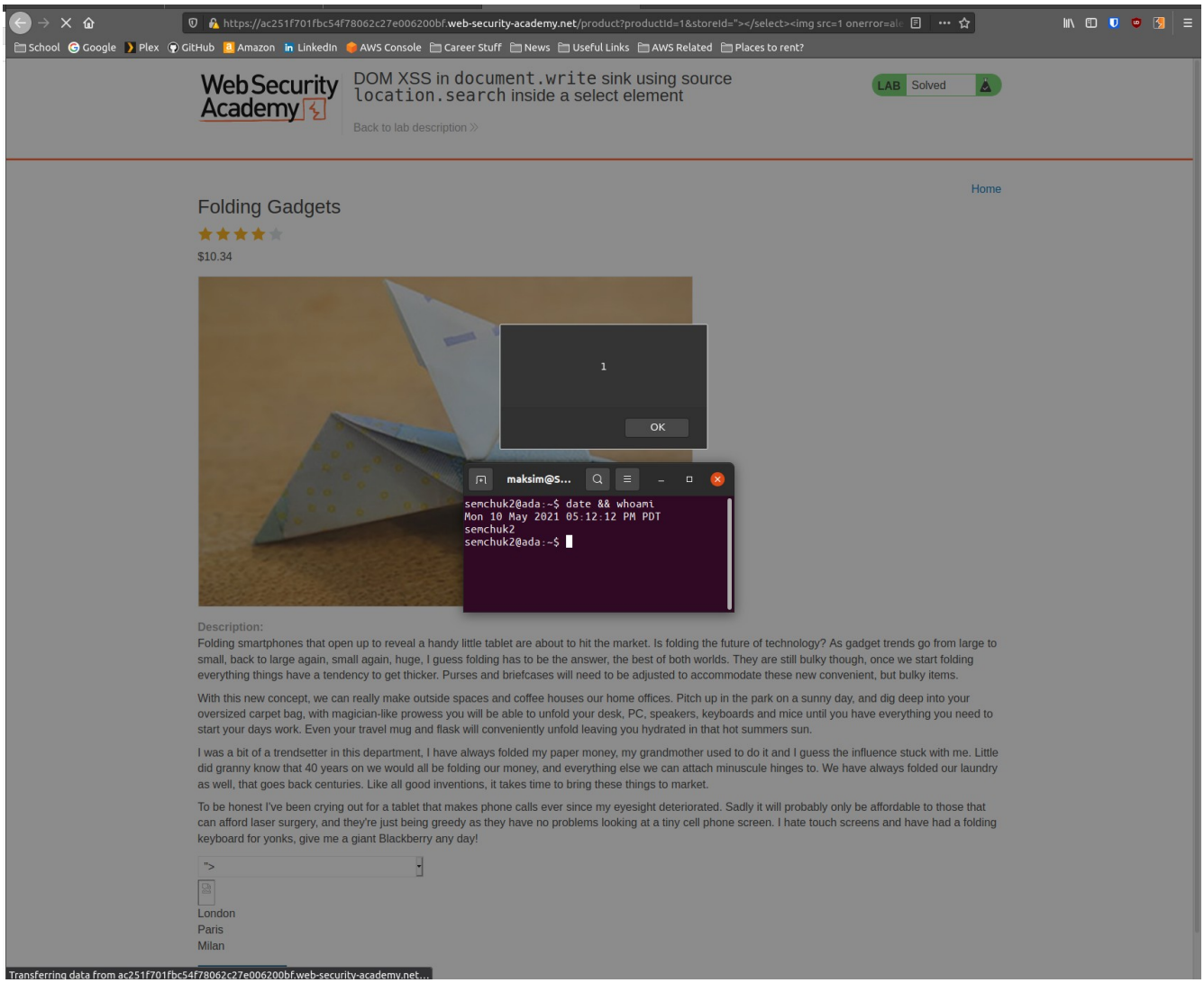
These weakenesses come from the fact that the html characters are encoded, and sent to the refrection server. The server decodes them and then runs the code that was sent, resulting in a attack being successful.

3.1.12 document-write-sink



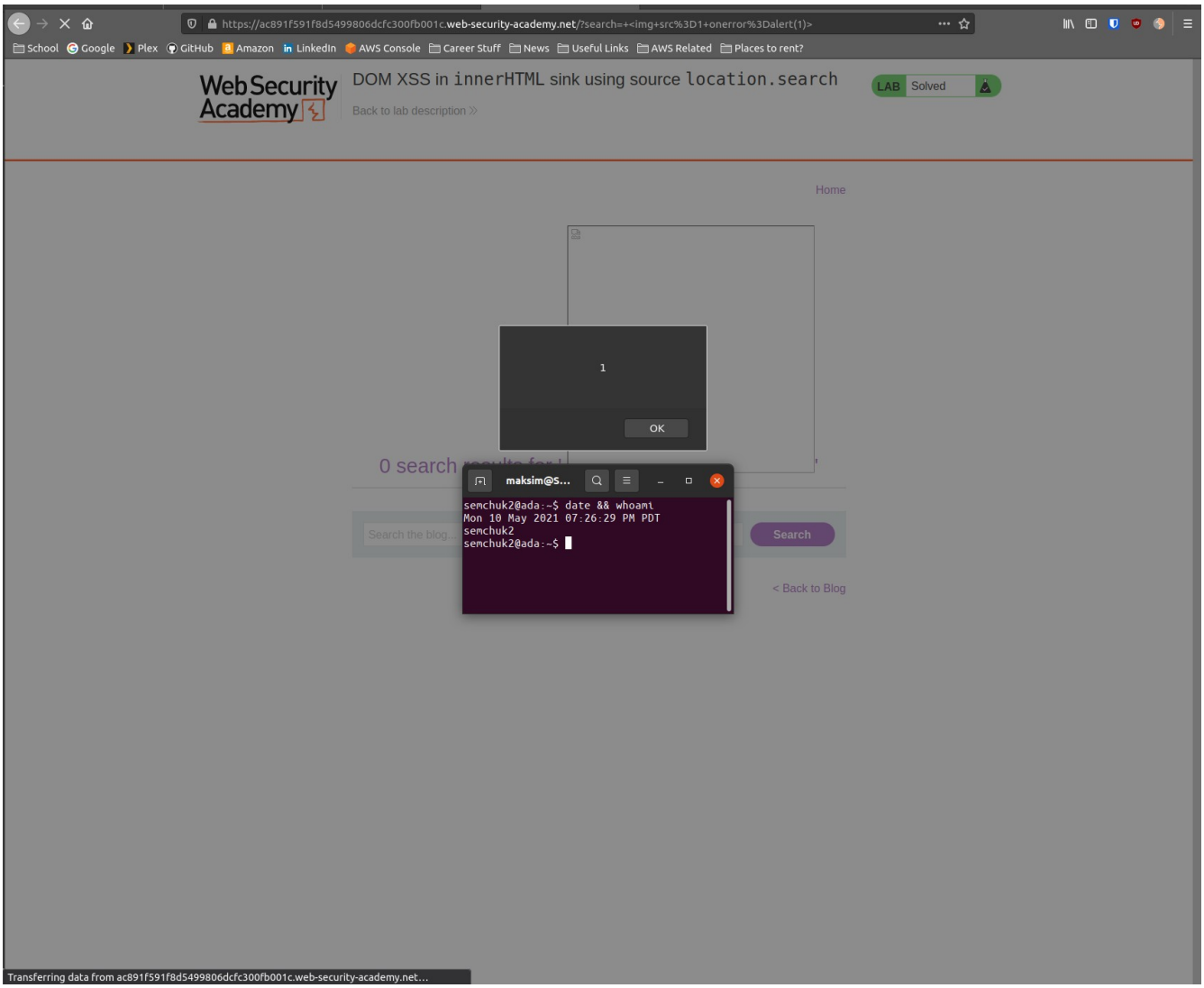
The vulnerablity here, gets called from the document where the document in the webpage and gets executed on the server by doing escapes.

3.1.13 document-write-sink-inside-select-element



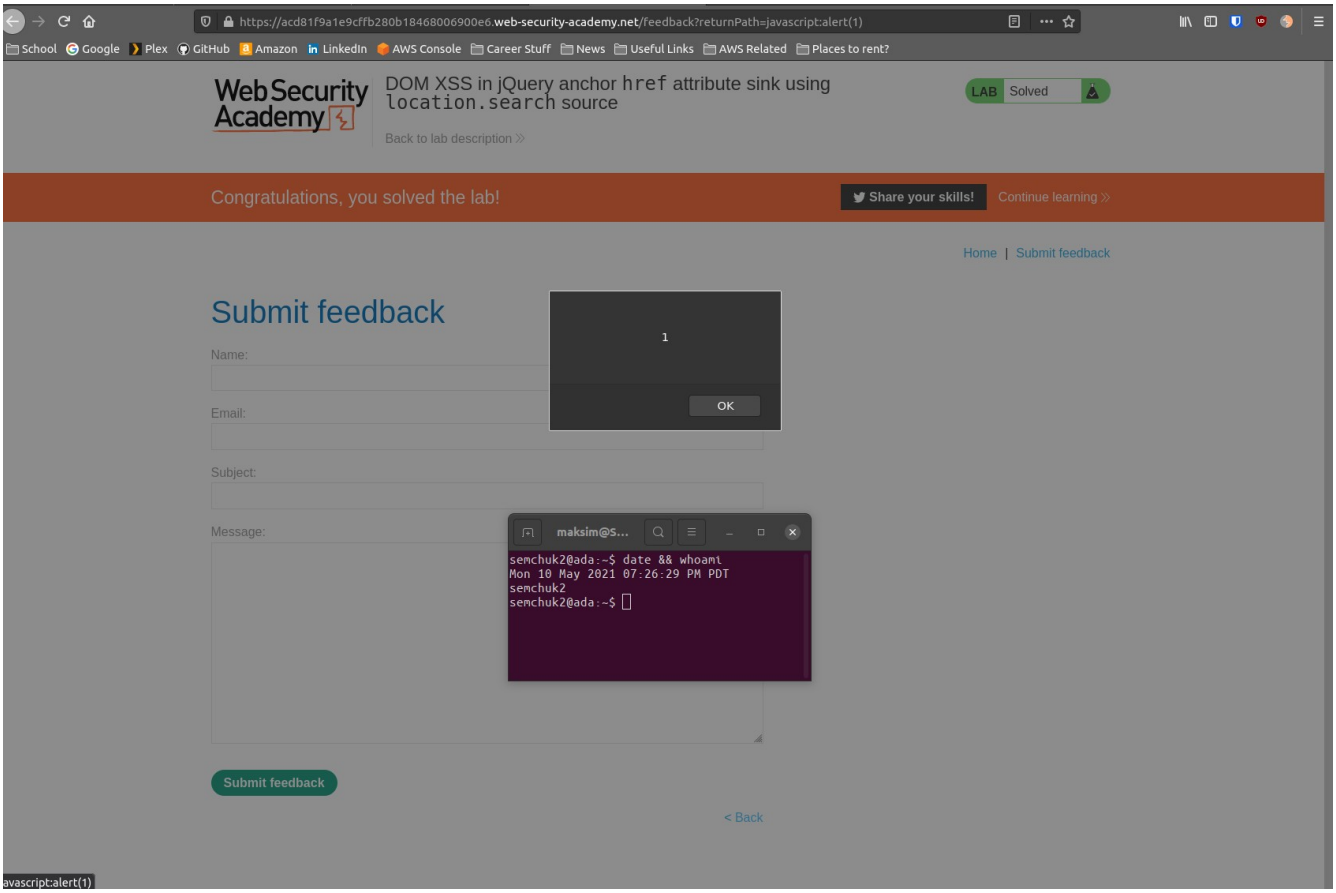
The javascrpt is exposed and is able to be injected into the search bar. The code is then executed

3.1.14 innerhtml-sink



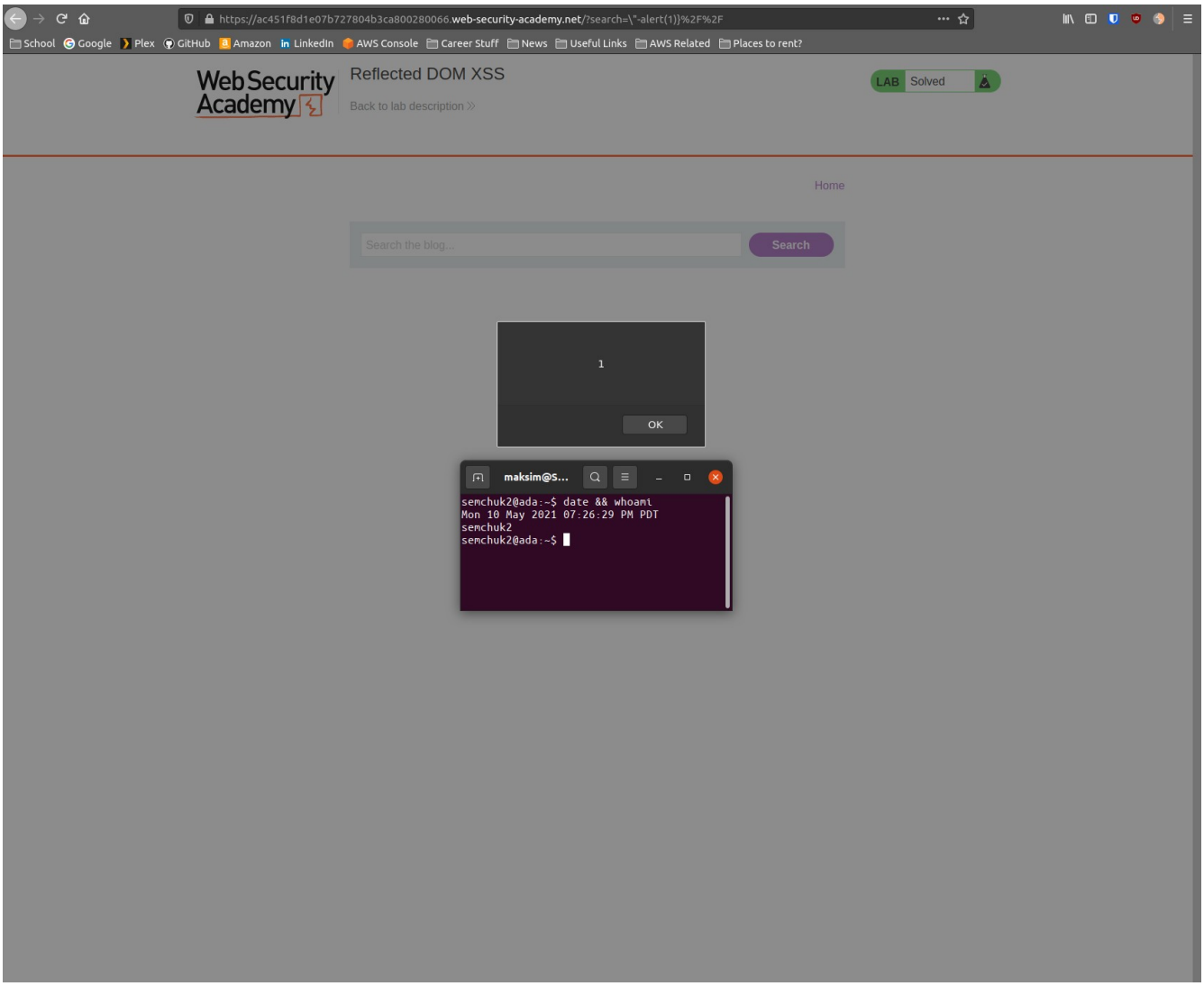
Here, we are able to show that the HTML is taken into a source as a javascript file. The second site is taken into the source file as a image, but contain the function to alert.

3.1.15 jquery-href-attribute-sink



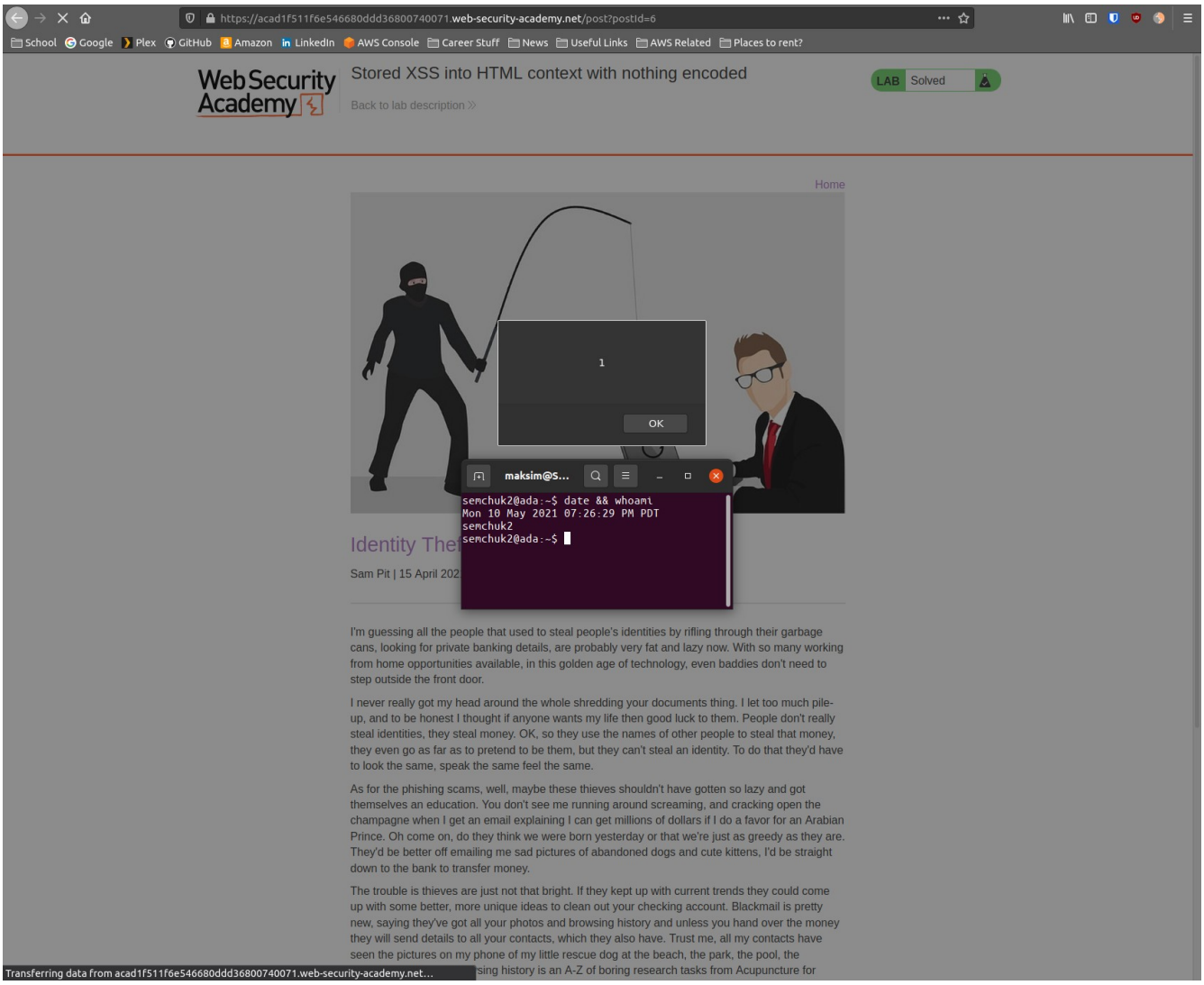
This allows us to insert a javascript as a argument into the return link, so when a user clicks back it shows an alert function with a 1.

3.1.16 dom-xss-reflected



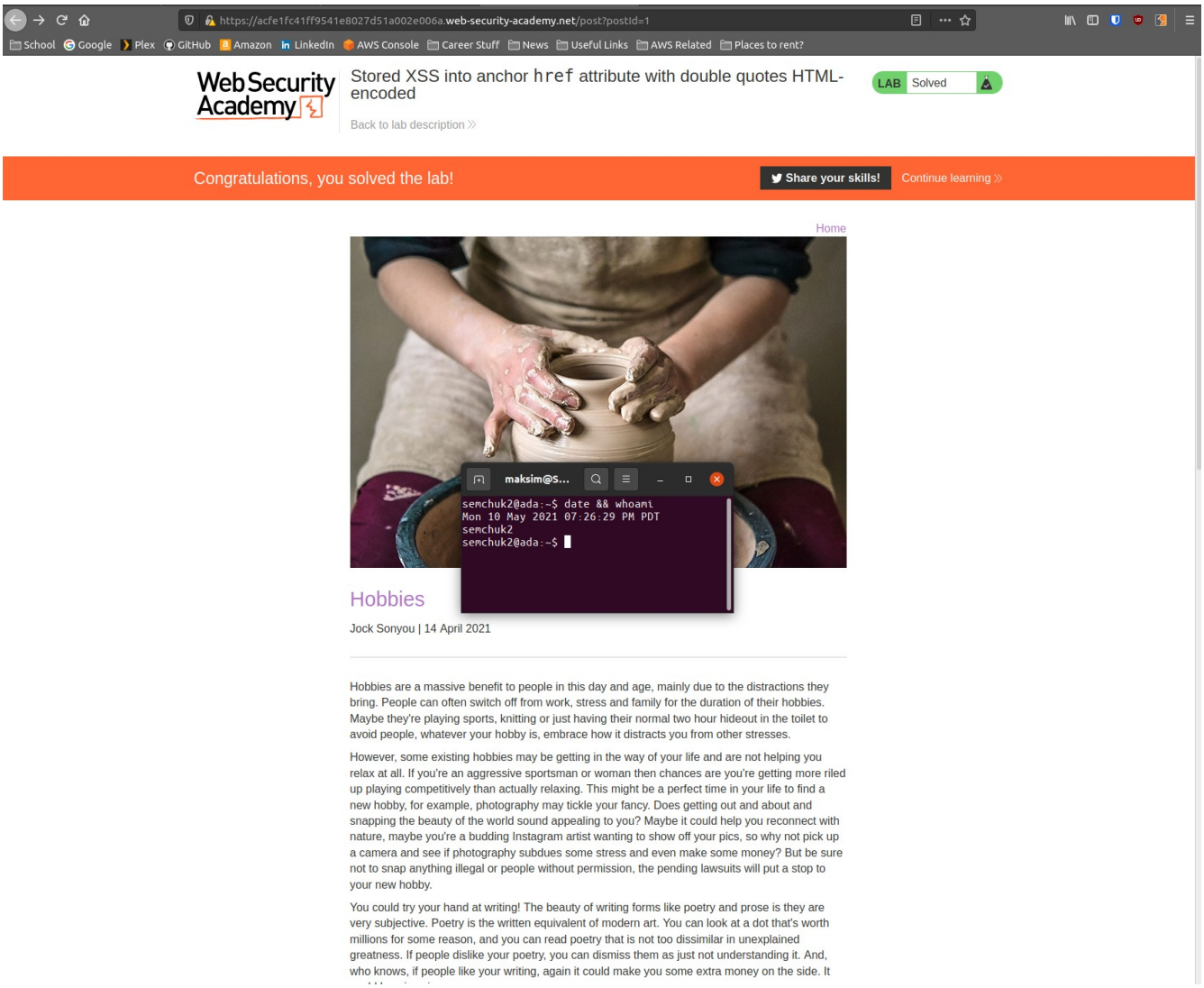
This is a cross site scripting attack that gets reflected by a server. Then the server runs the script.

3.1.18 html-context-nothing-encoded



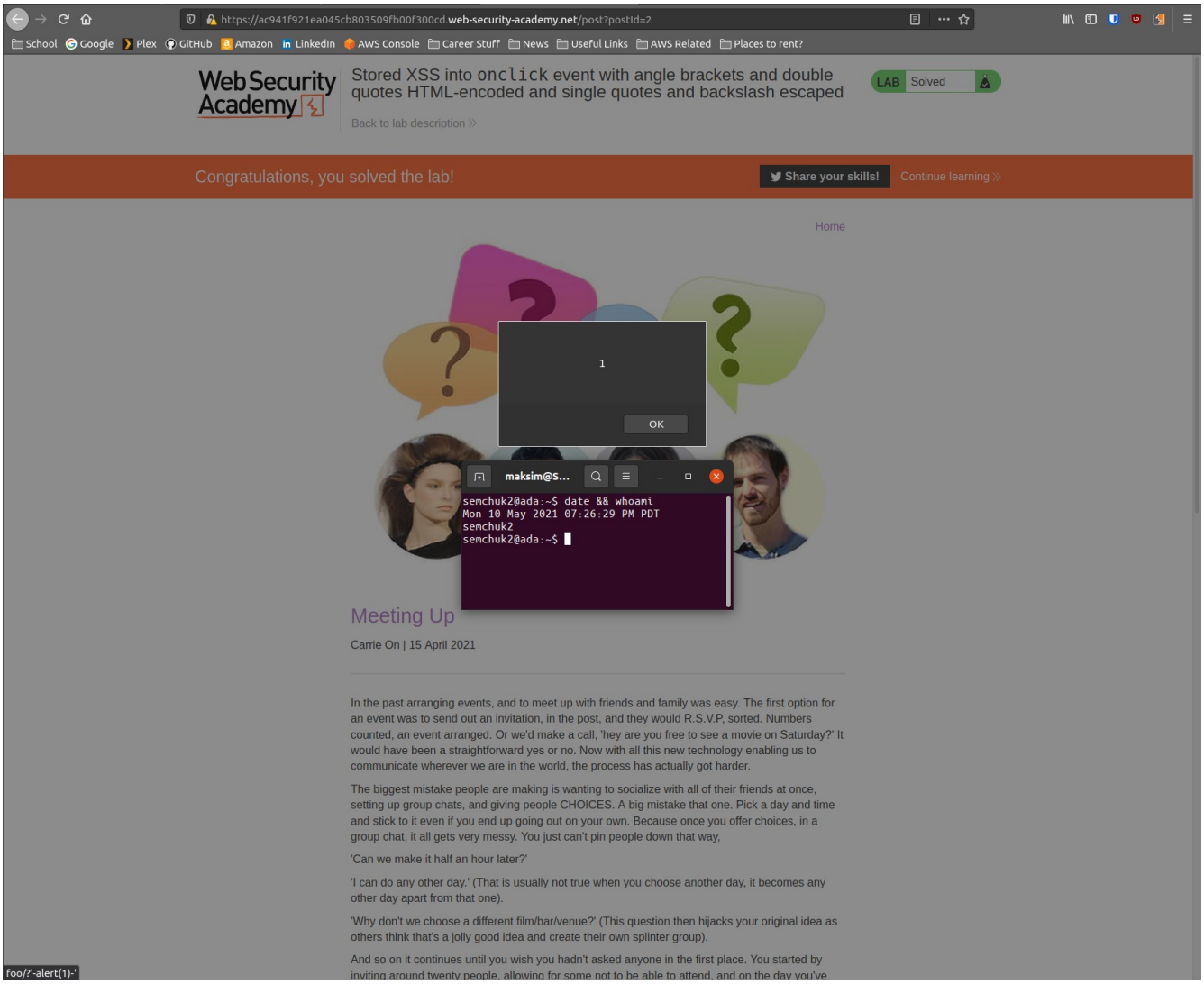
The cross site script attack allows us to put html to the server, and when the comment part gets parsed and since the comment is not sanitized then it gets executed.

3.1.19 href-attribute-double-quotes-html-encoded



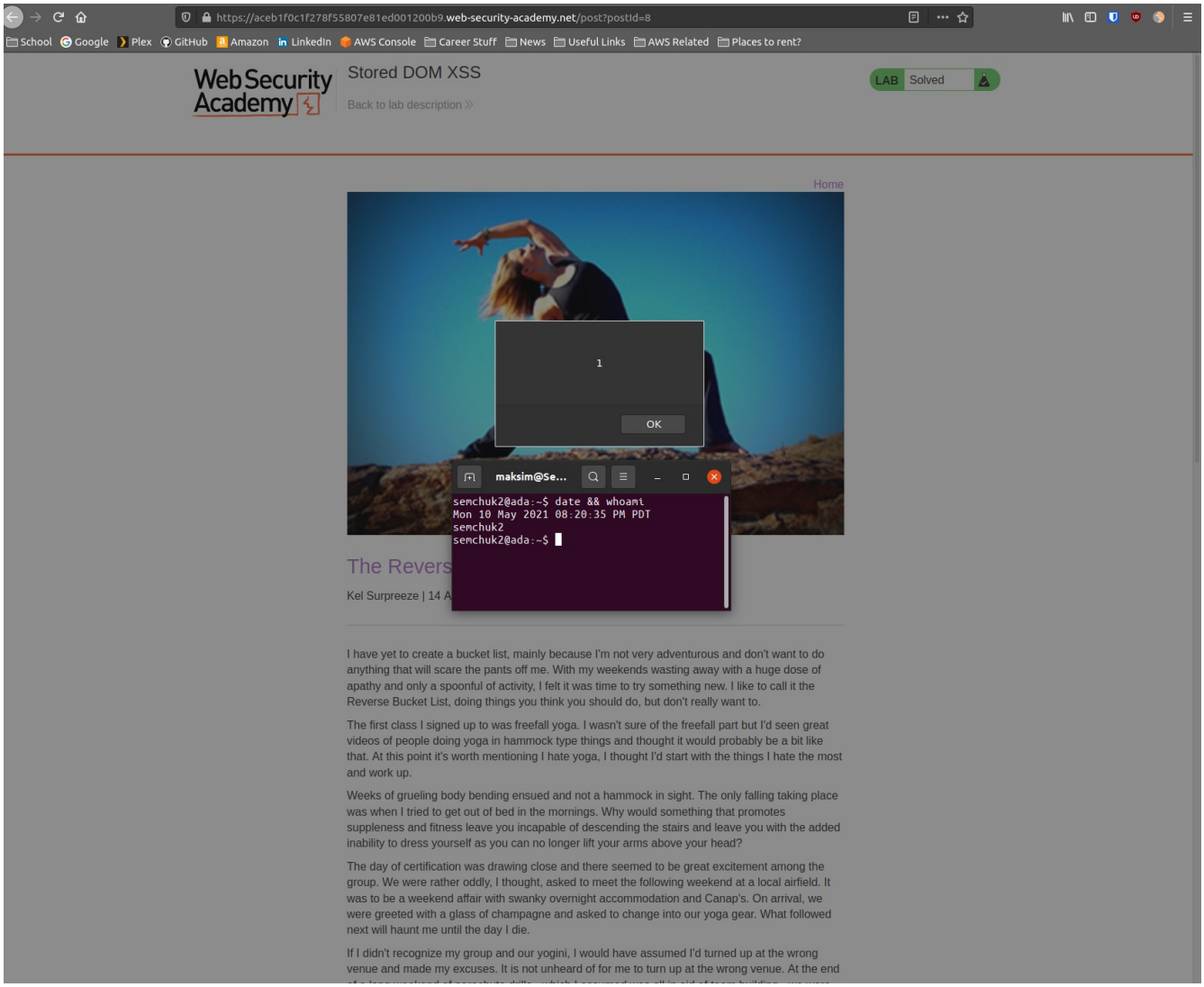
So here what happens is, if you do some filtering of the data, then you are able to intercept a packet and then edit it through burp. In this case, normally you cannot do a non formated website name but if you intercept the packet you can repeat the request with a custom website payload, sending javascript functions instead. This is dangerous since you bypass the check and gain access to the remote server.

3.1.20 onclick-event-angle-brackets-double-quotes-html-encoded-single-quotes-backslash-escaped



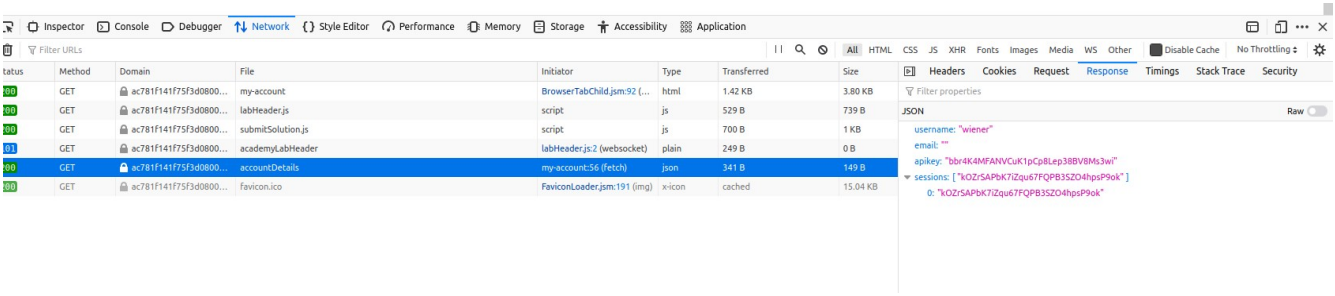
Here, a comment gets forwarded through a one click redirect, and this is is really dangerous since you are able to then inject a javascript function into it and have it run on redirect.

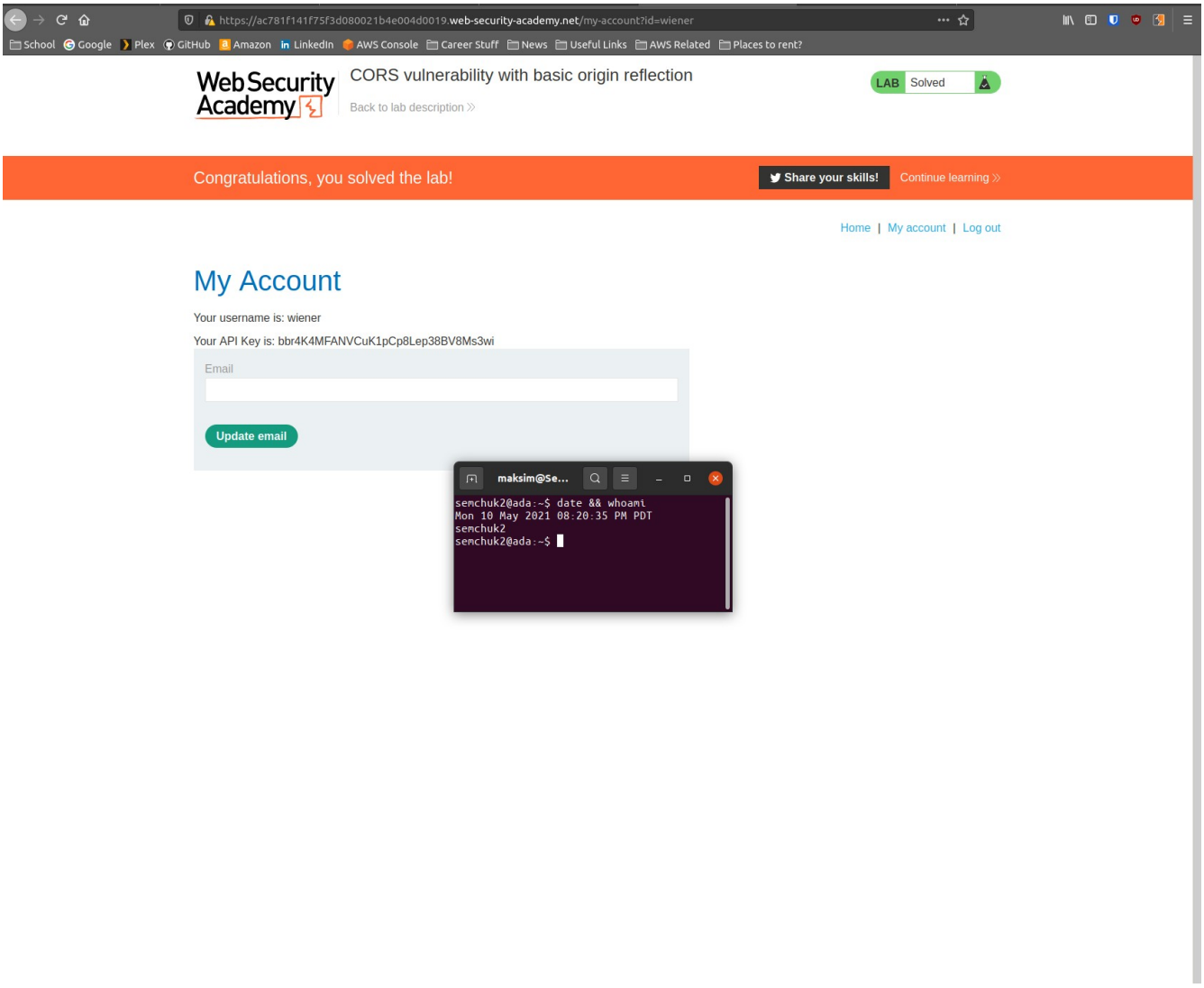
3.1.23 dom-xss-stored



This is a simple cross site attack where the website attempted to prevent this by doing a replace but by allowing more then one set of brackets to come through, the attack succeeds since it does not do this to all brackets, just the first one.

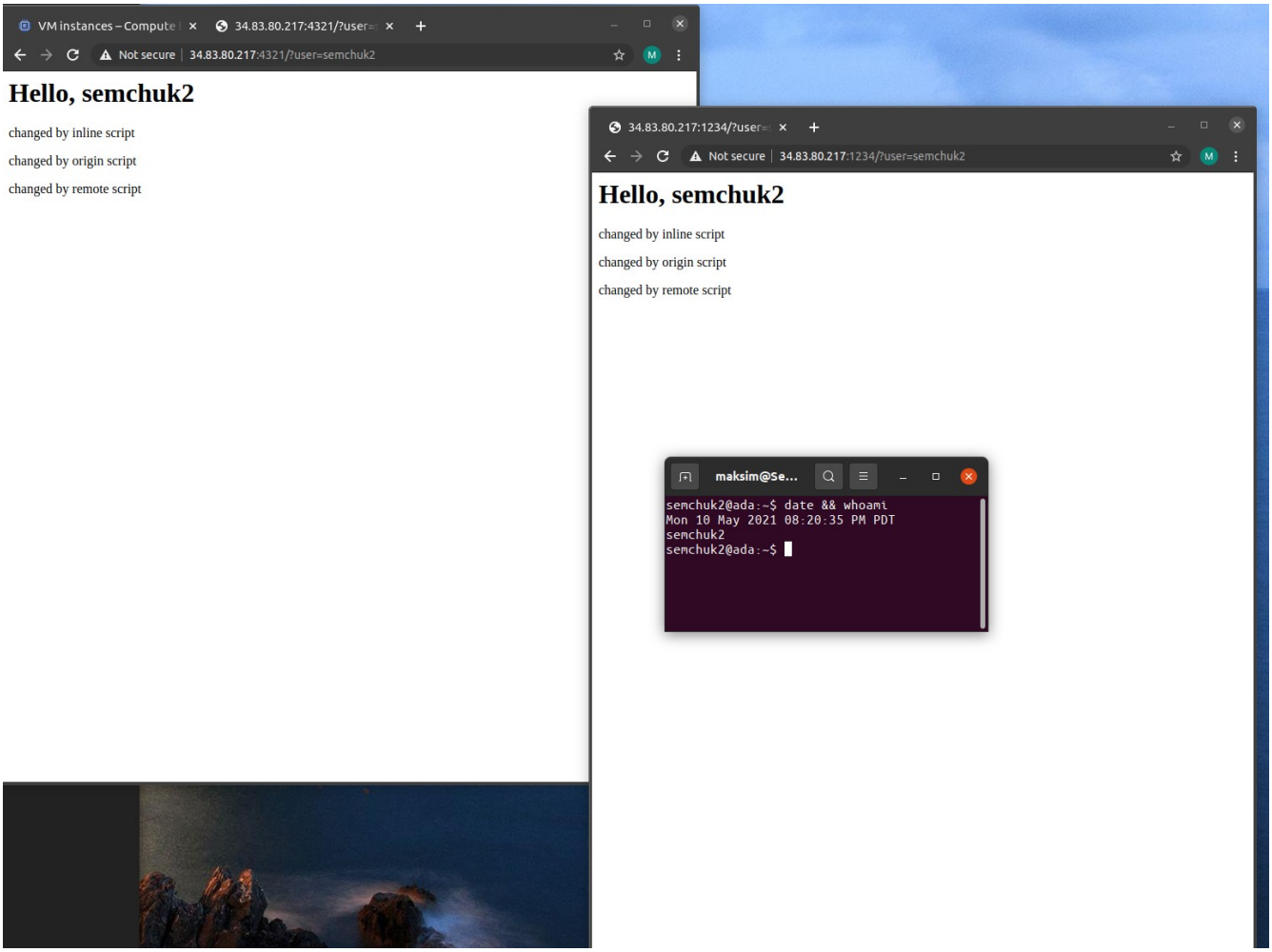
3.1.25 stealing-cookies



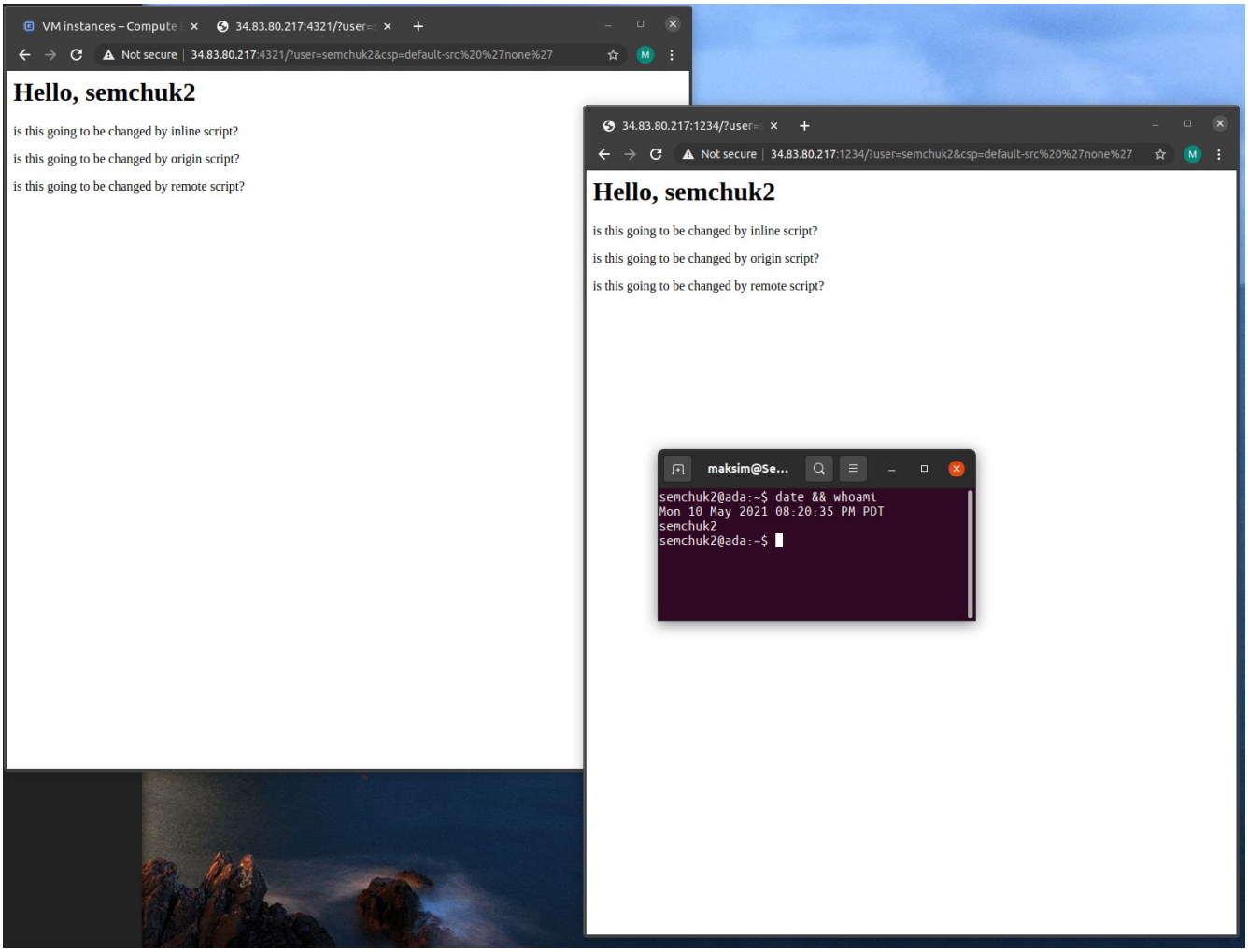


This is a vulnerablity because the server is to allowing and allows unknown domains who are redirected to access API keys.

3.2.4



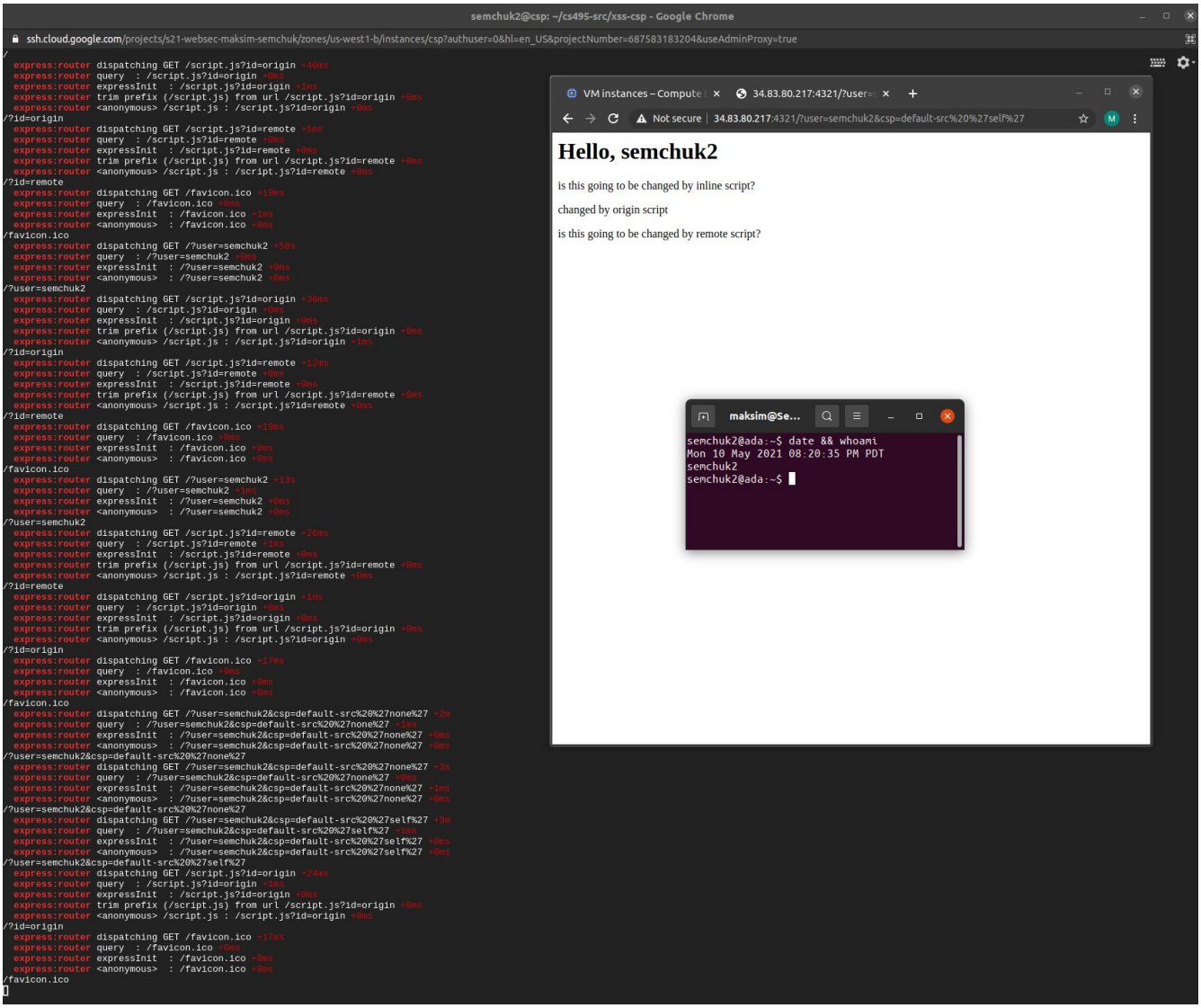
Submission of my username



here is the restriction being changed

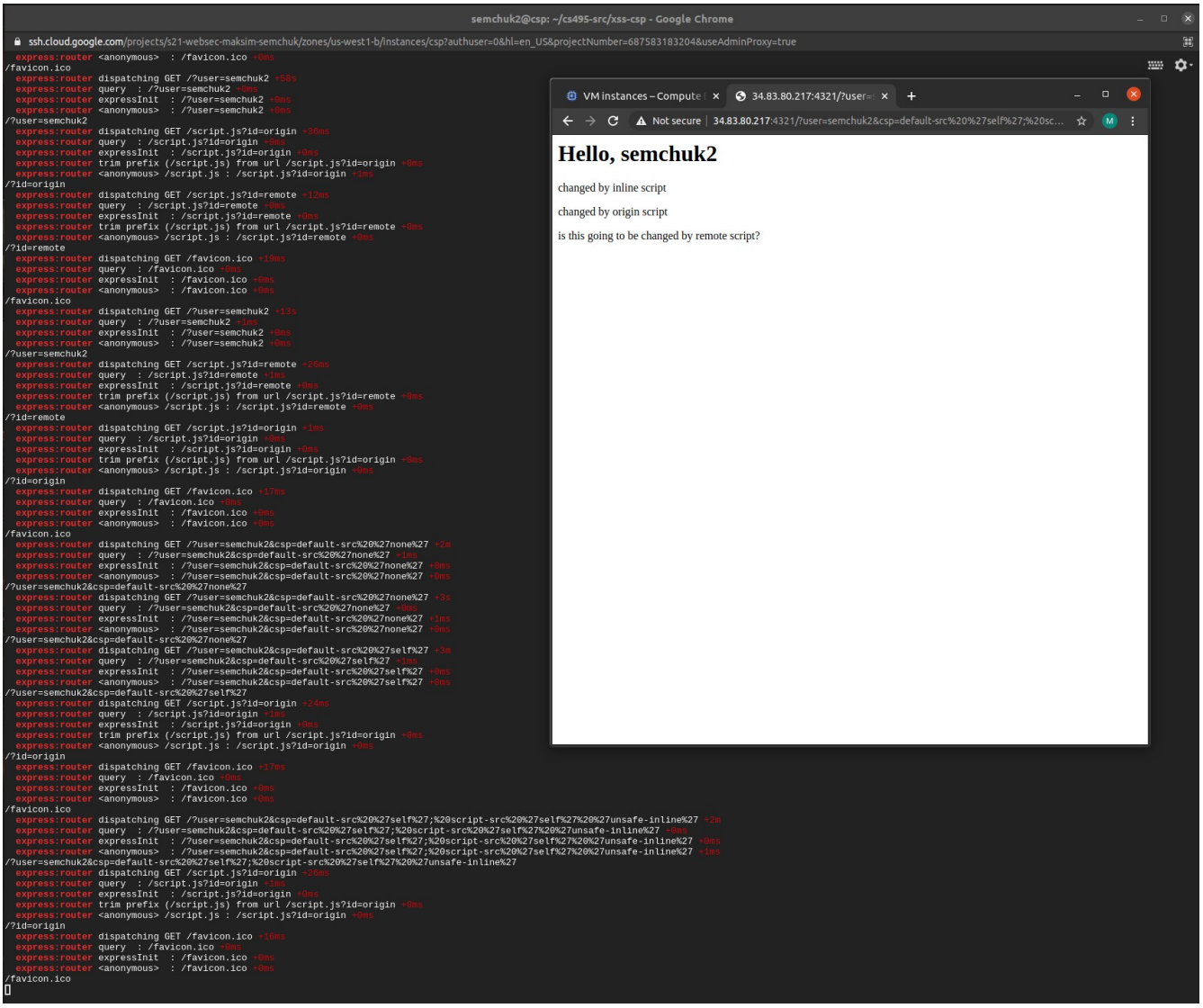
The console output for the last two steps:

Here is example #3:



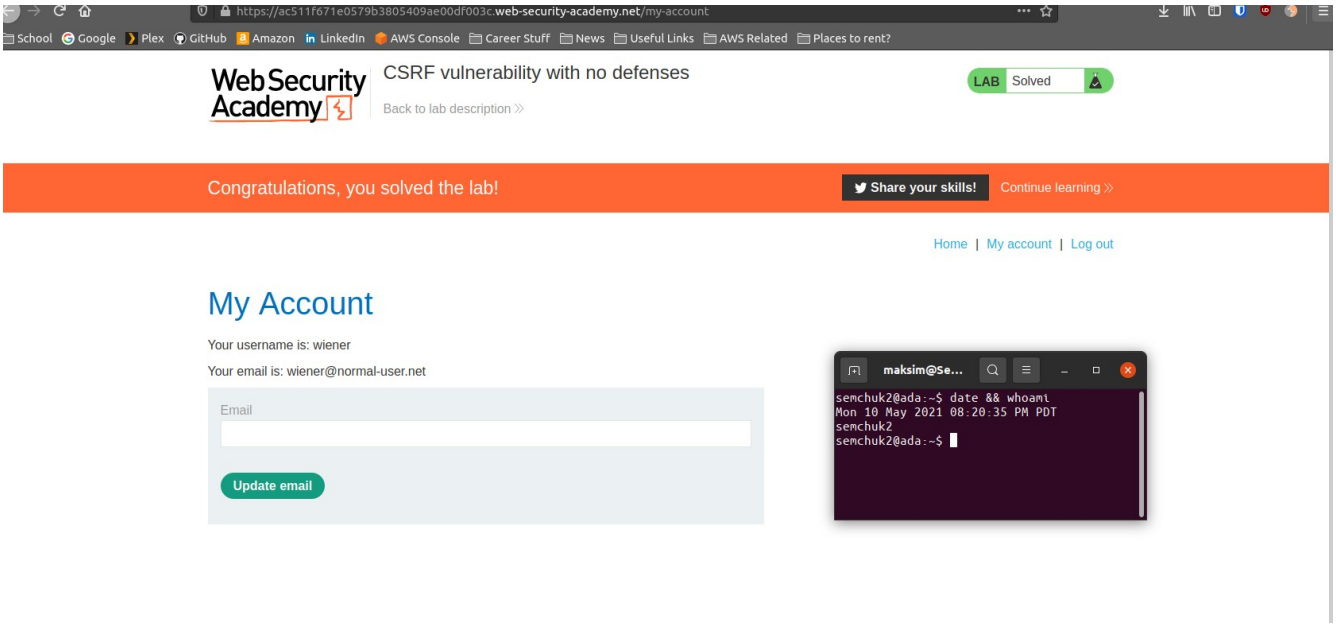
These differ because they only allow the user who owns the project to be able to connect.

Example #4:



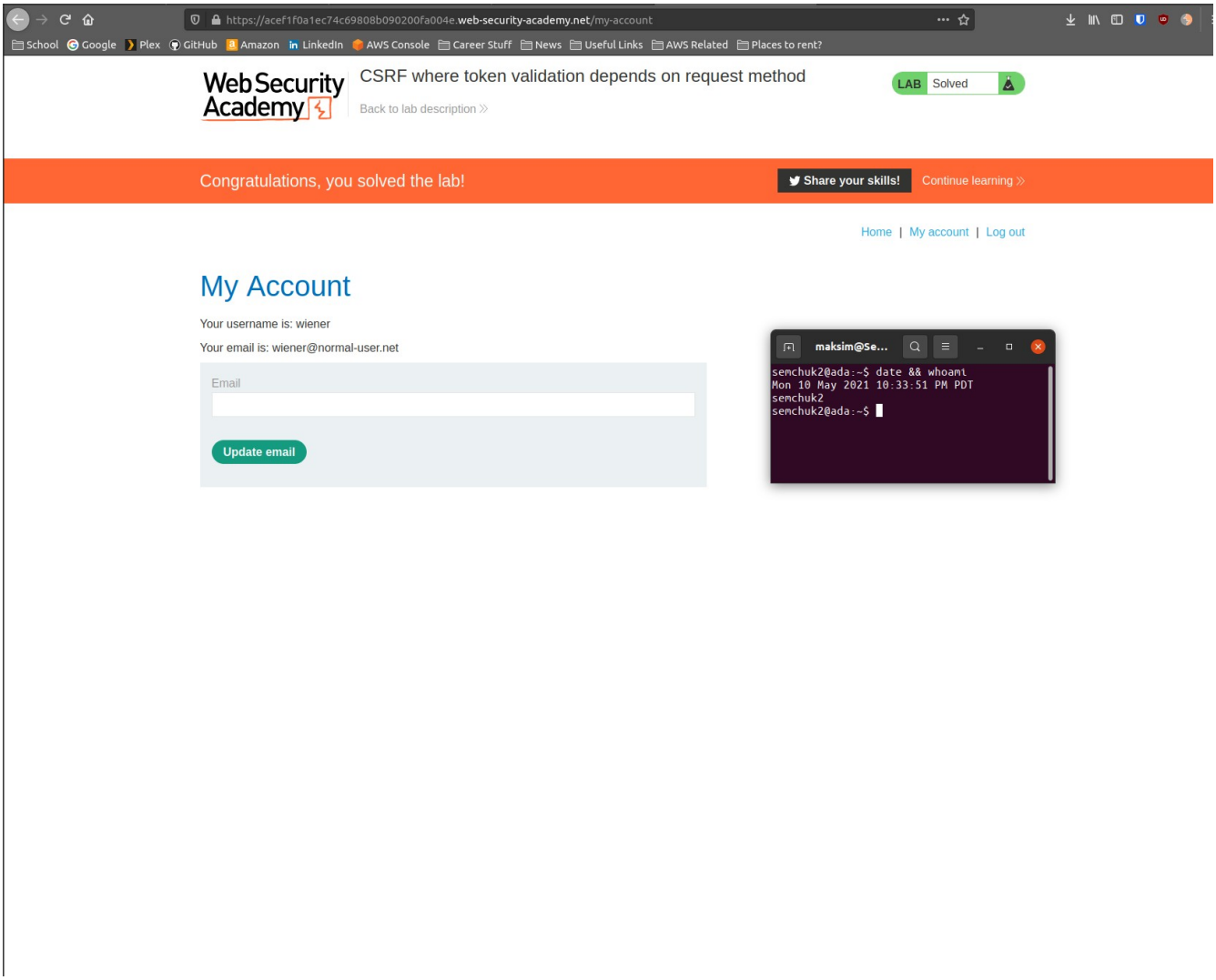
Here, it allows the unsafe inline arguments to be passed in. This means that this is less secure then the previous example.

3.3.1-2 no-defenses



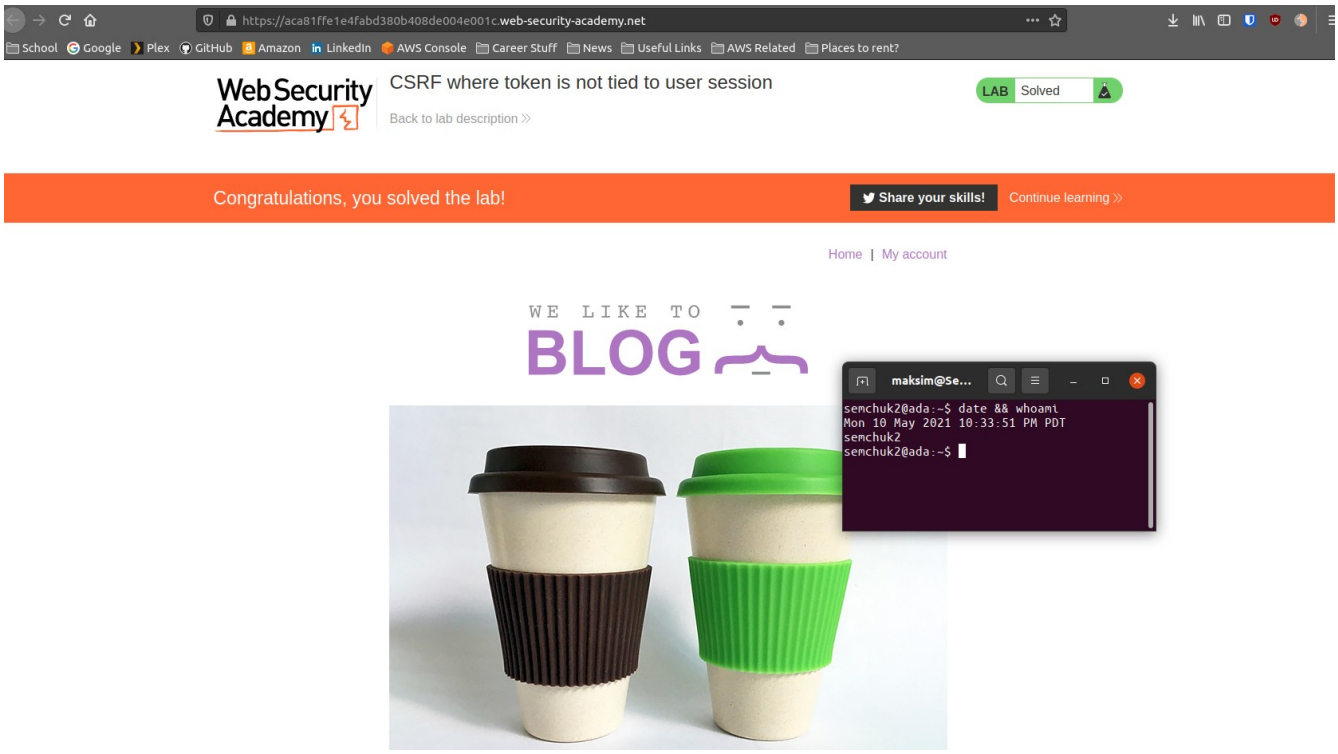
This is a simple thing that allows the users to change email address, but since this does not protect from redirects then it allows a server to do this automatically allowing an attacker to gain access to a users account.

3.3.3 token-validation-depends-on-request-method



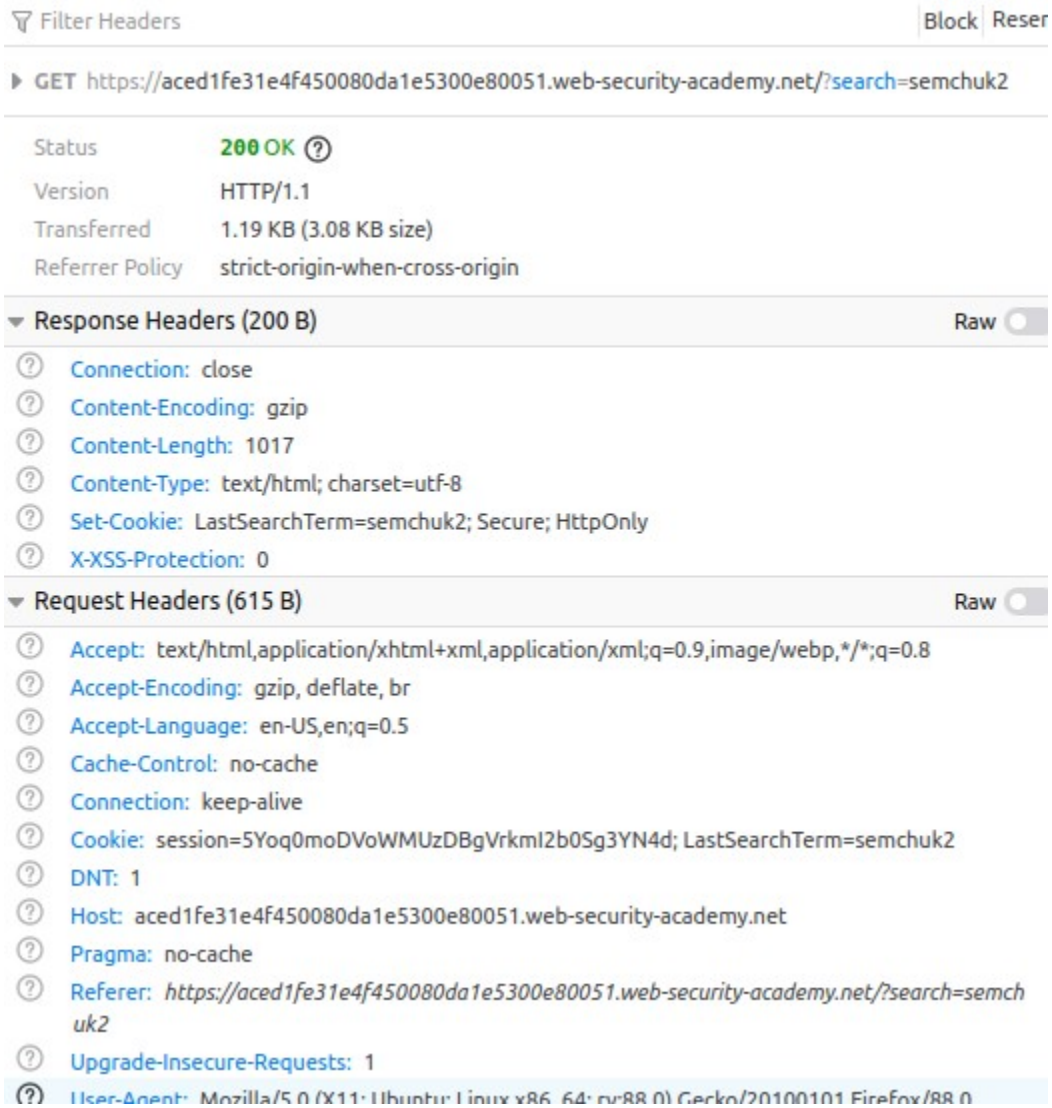
This is a simple thing that allows the users to change email address, but since this does not protect from redirects then it allows a server to do this automatically allowing an attacker to gain access to a users account. This is to show that you MUST secure both gets and posts.

3.3.4 token-not-tied-to-user-session



The weakness here, is that due to the fact that crsf tokens are not tied to the user, and the session leading to a vulnerability in allowing another user to claim the crsf token and present it as their own.

3.3.5 token-duplicated-in-cookie



Picture of the request headers

How many cookies are returned? What are their names?

According to the content legnth, 1017.

How have `foo` and `bar` been interpreted?

They have been interpreted as a pair for usernames. And sets up for a new username.

How many cookies have been returned?

There was only 2 cookies returned.

Explain the results. Give one reason why a developer would choose to implement CSRF protection in this manner

This would be because they are trying to tie the user to the cookie, to prevent sharing of cookies.

What status code is returned? What is the value of the `csrf` field in the HTML form that is given back as a response?

A 200 , the cookie of semchuk2

How might a developer use a keyed hash function on the server to prevent this request from succeeding without being forced to store each token?

They can create a hash to the key, and the users can provide the hash to ensure that it is correct.

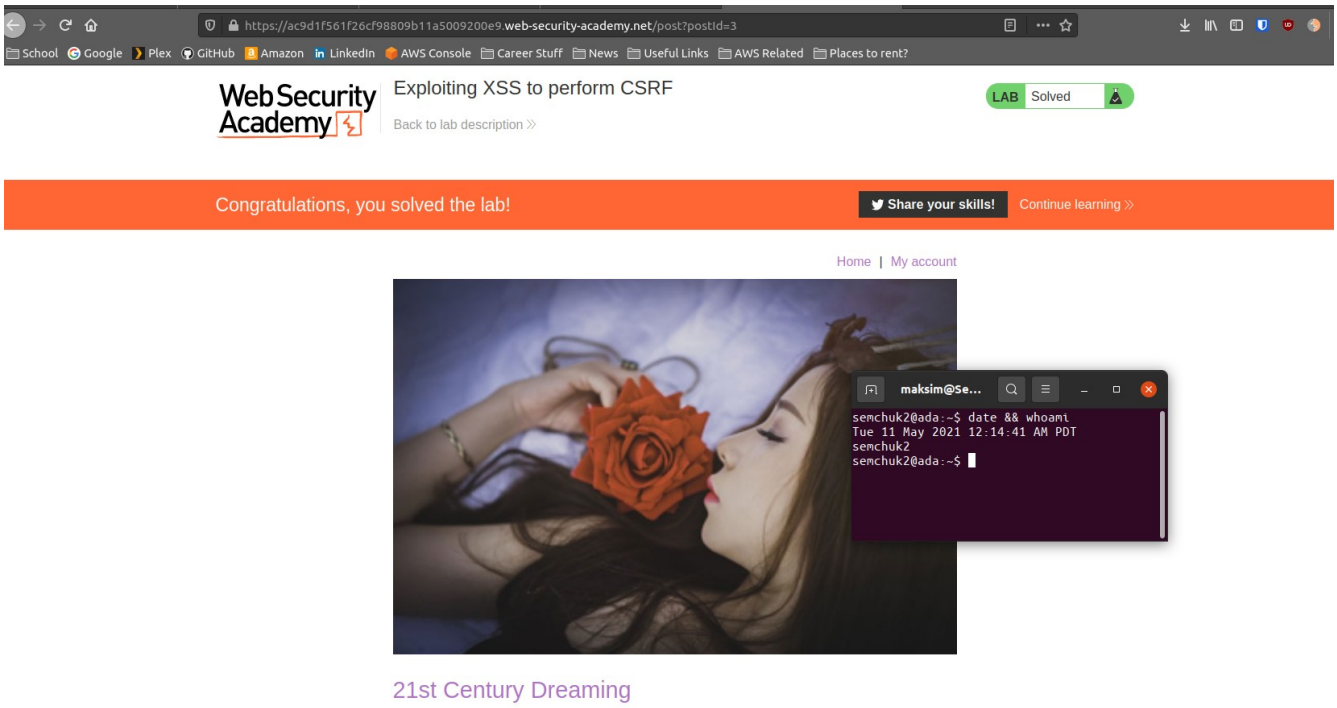
3.3.9 referer-validation-broken

What status code and response text is returned?

HTTP status code: 400 with response text "Invalid referer header"

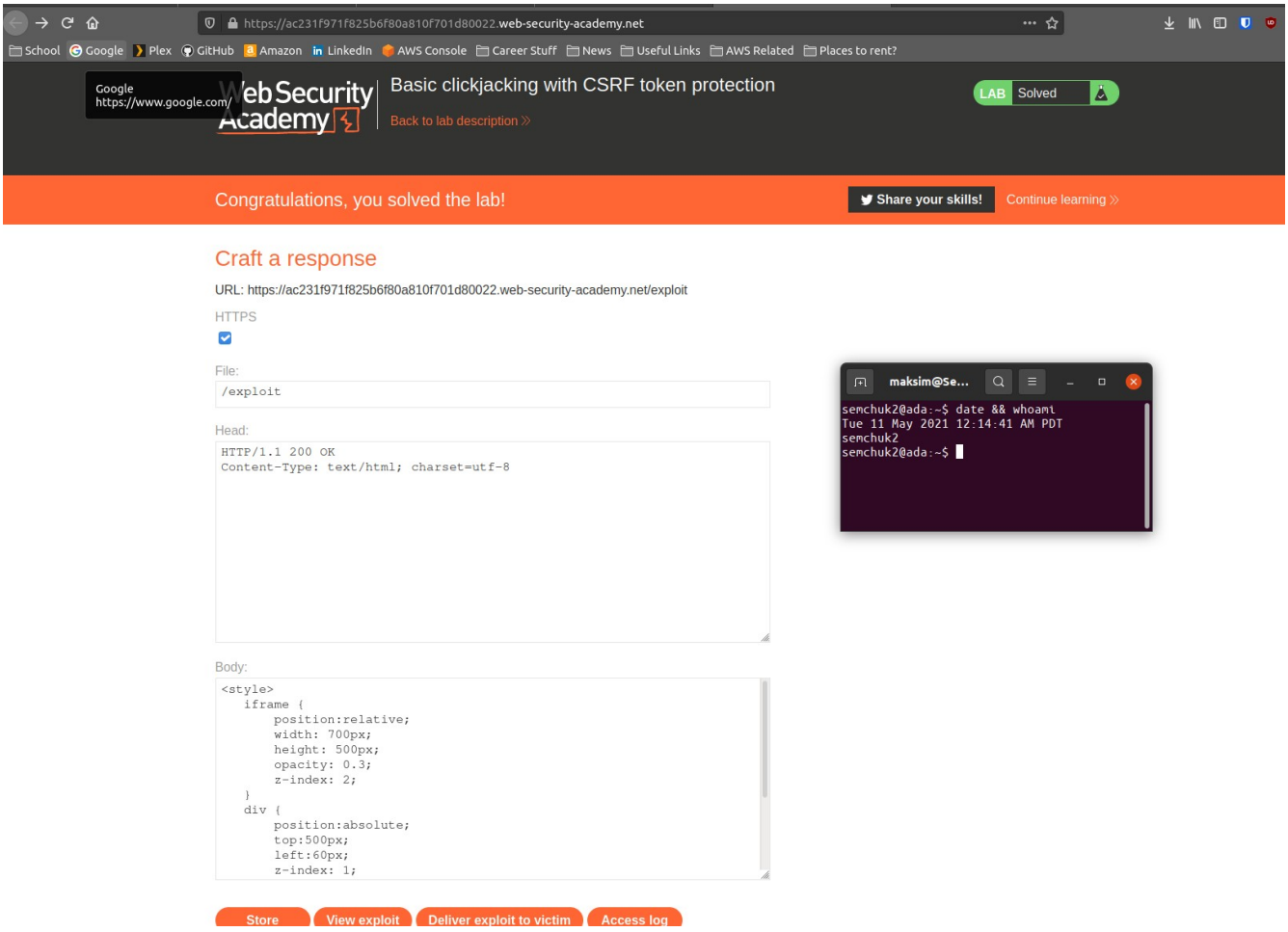
The weakness here is that the referer can be set and overridden allowing a remote or 3rd party user to put their own and use it. This will allow a person to forge the identity to forge the csrf token and gain access since it never gets validated.

3.3.11 perform-csrf



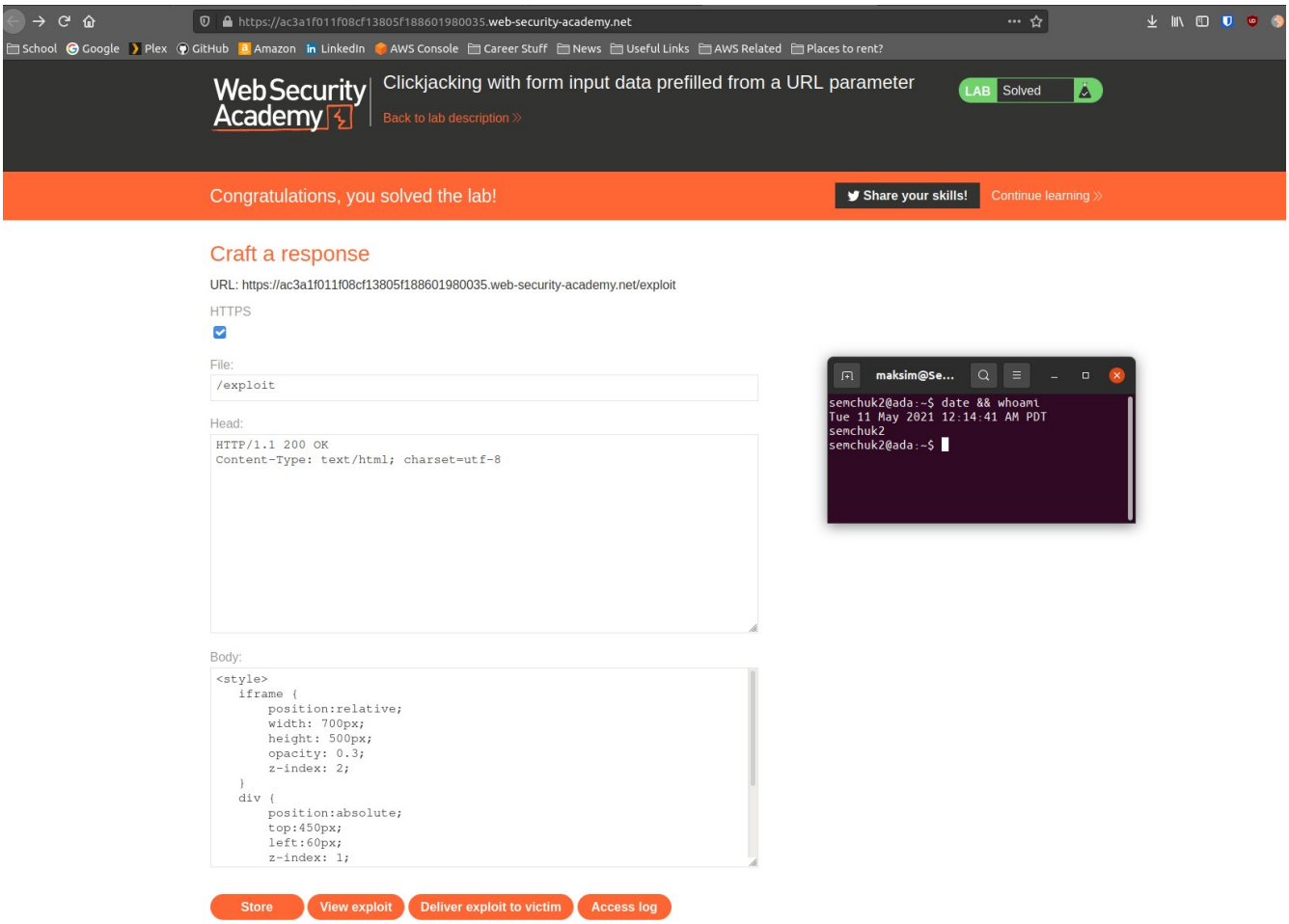
This is due to the fact that you are able to use javascript to invoke functions and in fact use the javascript to parse the csrf token out and post it.

3.4.1 basic-csrf-protected



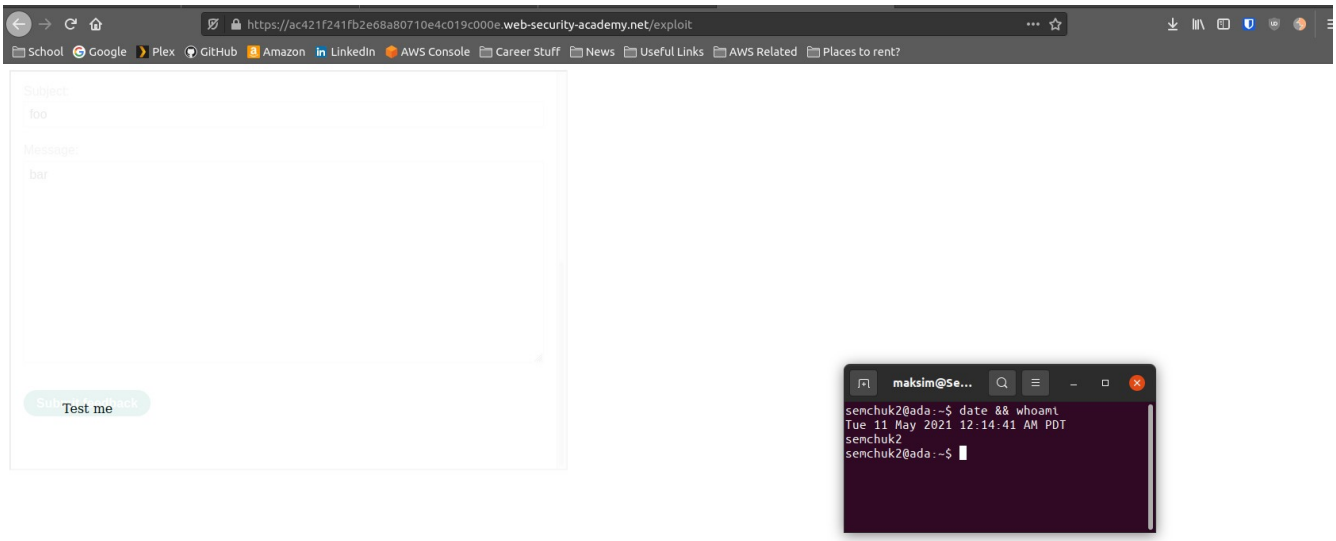
This works because we place a translucent button over the real one and then the user gets tricked into clicking it since they do not even know it is there,

3.4.4 prefilled-form-input



This works because we place a translucent button over the real one and then the user gets tricked into clicking it since they do not even know it is there, except here the victims email address becomes pre filled by us.

3.4.5 trigger-dom-based-xss



This did not work, despite it being right over the button. I am not sure why.s

3.5

Highlight the coordinates you entered previously via the UI in the decoded output for your lab notebook

maksim@SemchukPC:~$ base64 -d decode.txt

a:11:{i:0;a:4:{s:2:"x1";s:0:"";s:2:"y1";s:0:"";s:2:"x2";s:3:"100";s:2:"y2";s:3:"100";}i:1;a:4:
{s:2:"x1";s:1:"1";s:2:"y1";s:1:"1";s:2:"x2";s:2:"99";s:2:"y2";s:2:"99";}i:2;a:4:
{s:2:"x1";s:1:"1";s:2:"y1";s:1:"1";s:2:"x2";s:2:"99";s:2:"y2";s:2:"99";}i:3;a:4:
{s:2:"x1";s:0:"";s:2:"y1";s:0:"";s:2:"x2";s:3:"100";s:2:"y2";s:3:"100";}i:4;a:0:{}i:5;a:4:
{s:2:"x1";s:1:"0";s:2:"y1";s:1:"0"

3.6