

ỦY BAN NHÂN DÂN TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN



MÔN HỌC: AN TOÀN VÀ BẢO MẬT DỮ LIỆU TRONG HTTT
ĐỀ TÀI: KỸ THUẬT TẤN CÔNG CSRF VÀ CÁCH PHÒNG CHỐNG

THÀNH VIÊN NHÓM 1:

3119560061	Trần Thị Thu Thanh
3119560038	Nguyễn Phú Hồng Loan
3119560005	Tô Phương Dũng

GIÁO VIÊN HƯỚNG DẪN: Thầy Trương Tấn Khoa
Tp. Hồ Chí Minh, tháng 04 năm 2023

MỤC LỤC

I. GIỚI THIỆU ĐỀ TÀI.....	2
II. MÔ HÌNH CLIENT-SERVER.....	2
1. Giới thiệu.....	2
2. Nguyên tắc hoạt động.....	3
III. KỸ THUẬT TẤN CÔNG CSRF.....	4
1. CSRF là gì.....	4
2. Kịch bản tấn công CSRF.....	4
IV. CÁC PHƯƠNG PHÁP PHÒNG CHỐNG.....	6
1. Phía user.....	6
2. Phía server.....	6
2.1. Lựa chọn việc sử dụng GET VÀ POST.....	6
2.2 Sử dụng captcha, các thông báo xác nhận.....	7
2.3 Sử dụng csrf token.....	7
2.4 Sử dụng cookie riêng biệt cho trang quản trị.....	8
2.5 Kiểm tra REFERRER.....	8
2.6 Kiểm tra IP.....	8
2.7 Sử dụng JWT để xác thực.....	9
V. DEMO.....	9
1. Cách thức hoạt động của mô hình Client-Server khi sử dụng cookie để xác thực người dùng.....	9
2. Cách thức tấn công CSRF.....	13
3. Phương pháp phòng chống.....	15
3.1. Sử dụng JWT (JSON Web Token).....	15
3.2. Sử dụng CSRF token.....	18
3.2. Sử dụng Samesite cookie.....	24
VII. TÀI LIỆU THAM KHẢO.....	25

I. GIỚI THIỆU ĐỀ TÀI

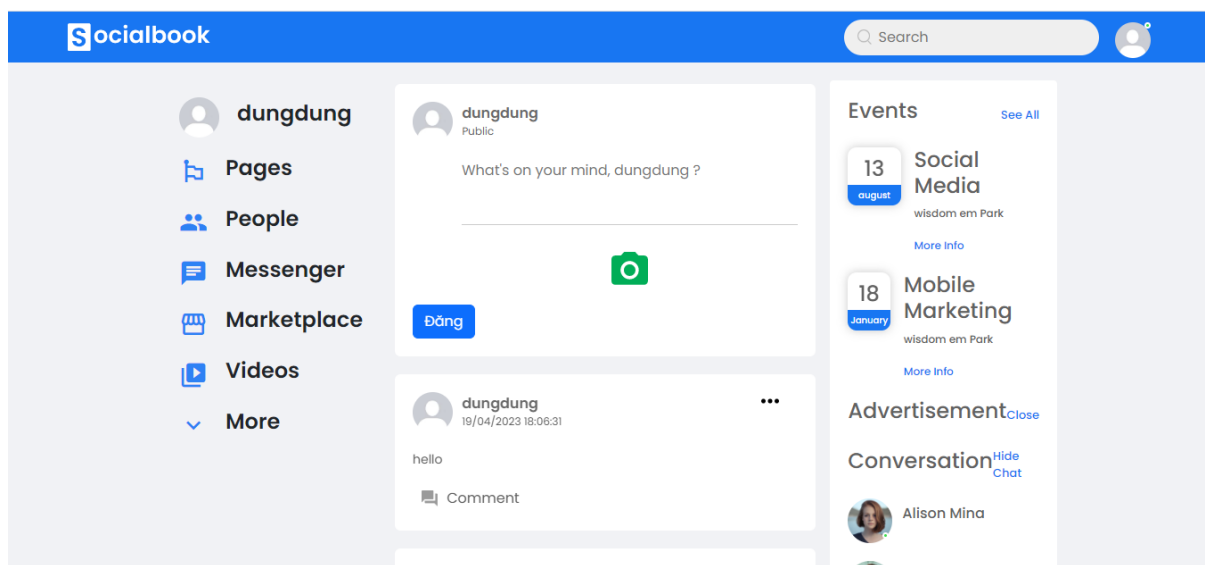
Trong thời đại công nghệ với sự phát triển mạnh mẽ của các ứng dụng web, thì việc bảo mật cho các ứng dụng này là cực kỳ quan trọng. Tấn công Cross-Site Request

Forgery (CSRF) là một trong những kỹ thuật tấn công hiệu quả nhằm chiếm được quyền điều khiển tài khoản của người dùng, tiết lộ thông tin cá nhân và tạo ra các hành động bất hợp pháp. Trong báo cáo này, chúng tôi sẽ tìm hiểu về kỹ thuật tấn công CSRF, cách thức thực hiện và các phương pháp để phòng chống các cuộc tấn công này.

Để mô tả rõ hơn về kỹ thuật trên chúng tôi đã xây dựng một website tương tự như Twitter, facebook có những chức năng như đăng bài, xem bài đăng, bình luận,...

Trong đó:

- Backend được viết bằng nodejs, typescript.
- Frontend được viết bằng reactjs, javascript.



Giao diện website

II. MÔ HÌNH CLIENT-SERVER

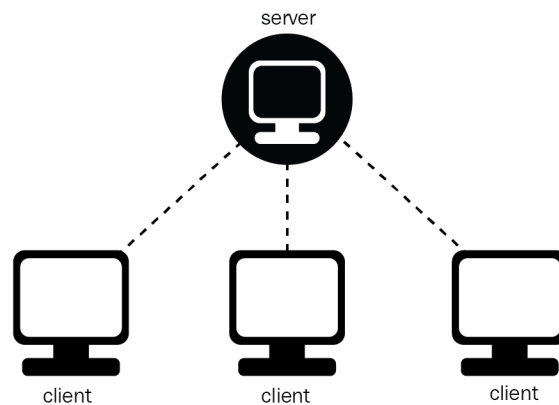
1. Giới thiệu

Mô hình client-server là mô hình phổ biến trong phát triển ứng dụng web. Đối với mô hình này, client và server là hai phần riêng biệt của hệ thống web. Trong trường hợp này, client đóng vai trò là một nơi mà người dùng tương tác với ứng dụng, còn server có nhiệm vụ xử lý các yêu cầu từ client và trả về các phản hồi tương ứng.

Client và Server về bản chất là 2 máy tính giao tiếp và truyền tải dữ liệu cho nhau.

Client: Với vai trò là máy khách, chúng sẽ không cung cấp tài nguyên đến các máy tính khác mà chỉ sử dụng tài nguyên được cung cấp từ máy chủ. Một client trong mô hình này có thể là một server cho mô hình khác, tùy thuộc vào nhu cầu sử dụng của người dùng.

Server: Là máy chủ có khả năng cung cấp tài nguyên và các dịch vụ đến các máy khách khác trong hệ thống mạng. Server đóng vai trò hỗ trợ cho các hoạt động trên máy khách client diễn ra hiệu quả hơn.

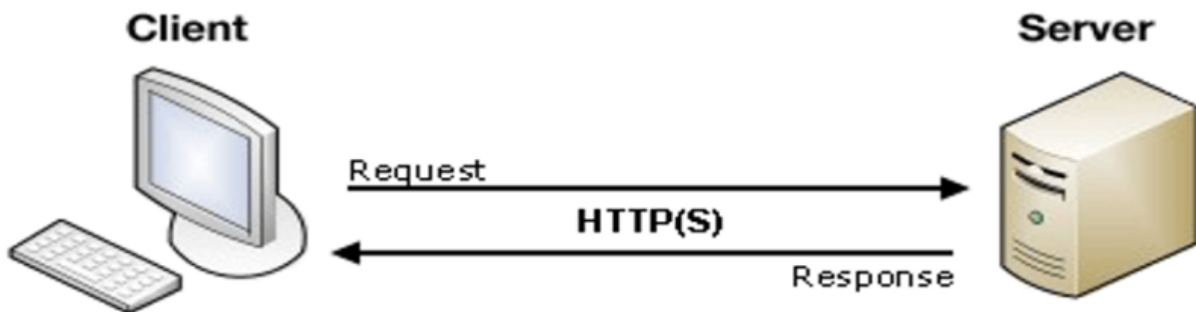


Hình minh họa về mô hình Client-Server

2. Nguyên tắc hoạt động

Trong mô hình Client-Server, Server chấp nhận tất cả các yêu cầu hợp lệ từ mọi nơi khác nhau trên Internet, sau đó trả kết quả về máy tính đã gửi yêu cầu đó.

Máy tính được coi là máy khách khi chúng làm nhiệm vụ gửi yêu cầu đến các máy chủ và đợi câu trả lời được gửi về.



Để máy khách và máy chủ có thể giao tiếp được với nhau thì giữa chúng phải có một chuẩn nhất định, và chuẩn đó được gọi là giao thức. Một số giao thức được sử dụng phổ biến hiện nay như: HTTPS, TCP/IP, FTP,...

Nếu máy khách muốn lấy được thông tin từ máy chủ, chúng phải tuân theo một giao thức mà máy chủ đó đưa ra. Nếu yêu cầu đó được chấp nhận thì máy chủ sẽ thu thập thông tin và trả về kết quả cho máy khách yêu cầu. Bởi vì Server – máy chủ luôn luôn trong trạng thái sẵn sàng để nhận request từ Client nên chỉ cần client gửi yêu cầu tín hiệu

và chấp nhận yêu cầu đó thì Server sẽ trả kết quả về phía Client trong thời gian ngắn nhất.

III. KỸ THUẬT TẤN CÔNG CSRF

1. CSRF là gì

CSRF được biết đến dưới nhiều cái tên khác nhau – Cross site request forgery (CSRF – XSRF), Sea Surf hay là Session Riding. Đây là một vector tấn công, có khả năng đánh lừa trình duyệt web thực hiện các hoạt động không mong muốn trong ứng dụng người dùng đã đăng nhập.

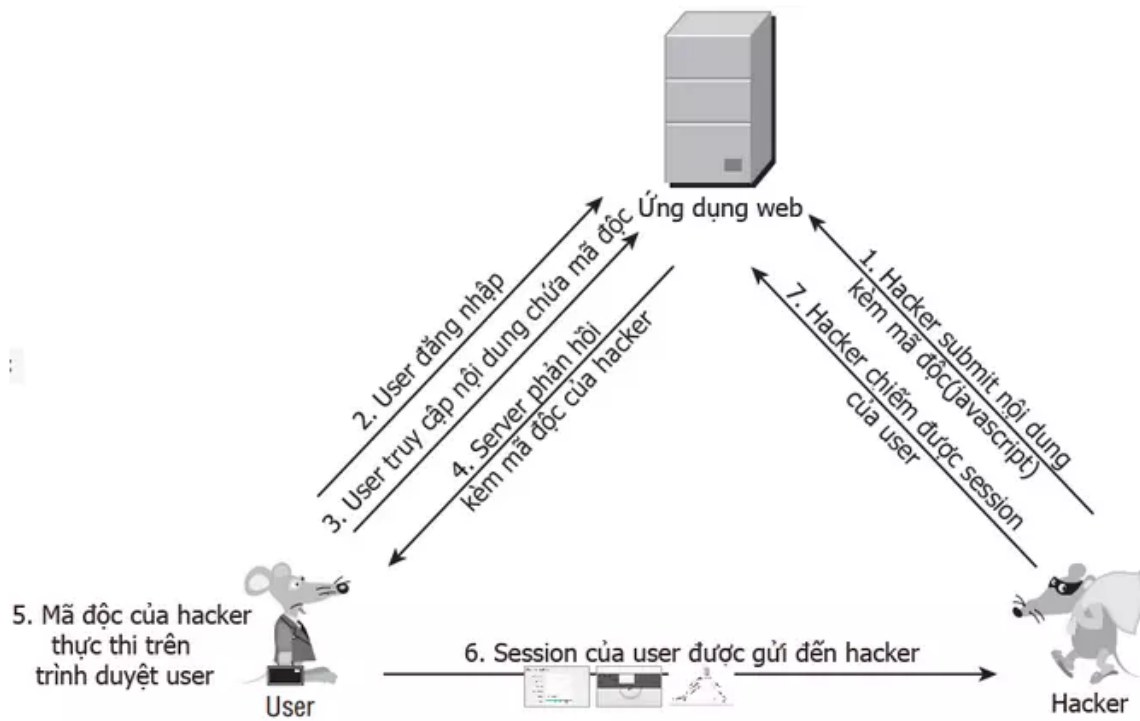
Một cuộc tấn công giả mạo chính chủ thể sẽ gây hậu quả nghiêm trọng cho các doanh nghiệp lẫn người dùng. Cụ thể là dẫn đến các hoạt động chuyển tiền trái phép, thay đổi mật khẩu, đánh cắp dữ liệu – gồm cả các session cookies.

Các cuộc tấn công mạng này thường lợi dụng các email hoặc liên kết giả mạo, nhằm đánh lừa nạn nhân gửi các request đến server. Nhiều trường hợp ứng dụng người dùng sử dụng đã được xác thực trước đó, nên rất khó phân biệt giữa các request hợp pháp và giả mạo.

2. Kịch bản tấn công CSRF

Kẻ tấn công sẽ tạo ra một trang web giả mạo, giả lập lại yêu cầu từ trang web đích và bắt buộc người dùng truy cập vào trang web giả mạo này. Khi đó, thông qua phiên làm việc của người dùng, kẻ tấn công có thể thực hiện các yêu cầu đối với trang web đích mà không được sự cho phép của người dùng.

Ví dụ cụ thể, khi người dùng đăng nhập vào trang web A, trang web này sẽ cấp cho người dùng một phiên làm việc (session), được lưu trữ trên trình duyệt. Một kẻ tấn công có thể tạo ra một trang web giả mạo, giả lập lại yêu cầu từ trang web A và buộc người dùng truy cập vào trang web giả mạo này. Khi đó, thông qua phiên làm việc của người dùng, kẻ tấn công có thể thực hiện các yêu cầu đối với trang web A mà không được sự cho phép của người dùng.



Hình minh họa cách thức tấn công CSRF

IV. CÁC PHƯƠNG PHÁP PHÒNG CHỐNG

Dựa trên nguyên tắc của CSRF "lừa trình duyệt của người dùng (hoặc người dùng) gửi các câu lệnh HTTP", các kĩ thuật phòng tránh sẽ tập trung vào việc tìm cách phân biệt và hạn chế các câu lệnh giả mạo.

1. Phía user

Để phòng tránh trở thành nạn nhân của các cuộc tấn công CSRF, người dùng internet nên thực hiện một số lưu ý sau:

- Nên thoát khỏi các website quan trọng: Tài khoản ngân hàng, thanh toán trực tuyến, các mạng xã hội, gmail, yahoo... khi đã thực hiện xong giao dịch hay các công việc cần làm. (Check - email, checkin...)
- Không nên click vào các đường dẫn mà bạn nhận được qua email, qua facebook ... Khi bạn đưa chuột qua 1 đường dẫn, phía dưới bên trái của trình duyệt thường có địa chỉ website đích, bạn nên lưu ý để đến đúng trang mình muốn.
- Không lưu các thông tin về mật khẩu tại trình duyệt của mình (không nên chọn các phương thức "đăng nhập lần sau", "lưu mật khẩu" ...)
- Trong quá trình thực hiện giao dịch hay vào các website quan trọng không nên vào các website khác, có thể chứa các mã khai thác của kẻ tấn công.

2. Phía server

Có nhiều lời khuyên cáo được đưa ra, tuy nhiên cho đến nay vẫn chưa có biện pháp nào có thể phòng chống triệt để CSRF. Sau đây là một vài kĩ thuật sử dụng.

2.1. Lựa chọn việc sử dụng GET VÀ POST

Sử dụng GET và POST đúng cách. Dùng GET nếu thao tác là truy vấn dữ liệu. Dùng POST nếu các thao tác tạo ra sự thay đổi hệ thống (theo khuyến cáo của W3C tổ chức tạo ra chuẩn http) Nếu ứng dụng của bạn theo chuẩn RESTful, bạn có thể dùng thêm các HTTP verbs, như PATCH, PUT hay DELETE

2.2 Sử dụng captcha, các thông báo xác nhận

Captcha được sử dụng để nhận biết đối tượng đang thao tác với hệ thống là con người hay không? Các thao tác quan trọng như "đăng nhập" hay là "chuyển khoản", "thanh toán" thường là hay sử dụng captcha. Tuy nhiên, việc sử dụng captcha có thể gây khó khăn cho một vài đối tượng người dùng và làm họ khó chịu. Các thông báo xác nhận cũng thường được sử dụng, ví dụ như việc hiển thị một thông báo xác nhận "bạn có muốn xóa hay k" cũng làm hạn chế các kỹ thuật Cả hai cách trên vẫn có thể bị vượt qua nếu kẻ tấn công có một kịch bản hoàn hảo và kết hợp với lỗi XSS.

2.3 Sử dụng csrf token

CSRF Token là một kỹ thuật bảo mật được sử dụng để ngăn chặn tấn công CSRF (Cross-Site Request Forgery), một loại tấn công mạng đáng ngại. Thuật toán của CSRF Token thường được thực hiện như sau:

- Khi người dùng truy cập vào trang web, máy chủ sẽ tạo một CSRF Token ngẫu nhiên và gán giá trị này cho một cookie hoặc một thẻ ẩn trong biểu mẫu (form) được gửi đến trình duyệt của người dùng.
- Khi người dùng gửi yêu cầu tới máy chủ, CSRF Token sẽ được gửi kèm theo yêu cầu này.
- Máy chủ sẽ kiểm tra CSRF Token trong yêu cầu và so sánh với giá trị được lưu trữ trong cookie hoặc thẻ ẩn. Nếu giá trị này trùng khớp, yêu cầu sẽ được chấp nhận và thực hiện, ngược lại, yêu cầu sẽ bị từ chối.
- Như vậy, kỹ thuật CSRF Token sẽ đảm bảo rằng chỉ những yêu cầu được gửi từ trang web đó mới được chấp nhận, và ngăn chặn được những tấn công CSRF từ các trang web khác.

Thuật toán được sử dụng để sinh ra CSRF Token:

Trong Node.js, csrf sử dụng thuật toán ngẫu nhiên để tạo ra CSRF token. Thuật toán này được gọi bởi csrf middleware mỗi khi yêu cầu được nhận từ trình duyệt của người dùng.

Cụ thể, csrf sử dụng hàm `crypto.randomBytes()` trong module `crypto` của `Node.js` để tạo ra một chuỗi ngẫu nhiên có độ dài 32 byte, sau đó chuyển đổi chuỗi này thành một chuỗi hexa và trả về kết quả.

Mã nguồn sau đây là ví dụ về cách csrf tạo CSRF token trong `Node.js`:

```
const crypto = require('crypto');

function generateCSRFToken() {
  return crypto.randomBytes(32).toString('hex');
}
```

Hàm `generateCSRFToken()` sử dụng `crypto.randomBytes(32)` để tạo ra một chuỗi ngẫu nhiên có độ dài 32 byte, và sau đó sử dụng phương thức `toString('hex')` để chuyển đổi chuỗi này thành một chuỗi hexa.

Bạn có thể sử dụng hàm `generateCSRFToken()` để tạo CSRF token trong ứng dụng `Node.js` của bạn. Tuy nhiên, khi sử dụng csrf, bạn không cần phải tạo CSRF token thủ công, vì csrf middleware sẽ tự động tạo và thêm CSRF token vào các yêu cầu POST, PUT và DELETE.

2.4 Sử dụng cookie riêng biệt cho trang quản trị

Một cookie không thể dùng chung cho các domain khác nhau, chính vì vậy việc sử dụng `"admin.site.com"` thay vì sử dụng `"site.com/admin"` là an toàn hơn.

2.5 Kiểm tra REFERER

Kiểm tra xem các câu lệnh http gửi đến hệ thống xuất phát từ đâu. Một ứng dụng web có thể hạn chế chỉ thực hiện các lệnh http gửi đến từ các trang đã được chứng thực. Tuy nhiên cách làm này có nhiều hạn chế và không thật sự hiệu quả.

2.6 Kiểm tra IP

Một số hệ thống quan trọng chỉ cho truy cập từ những IP được thiết lập sẵn

2.7 Sử dụng JWT để xác thực

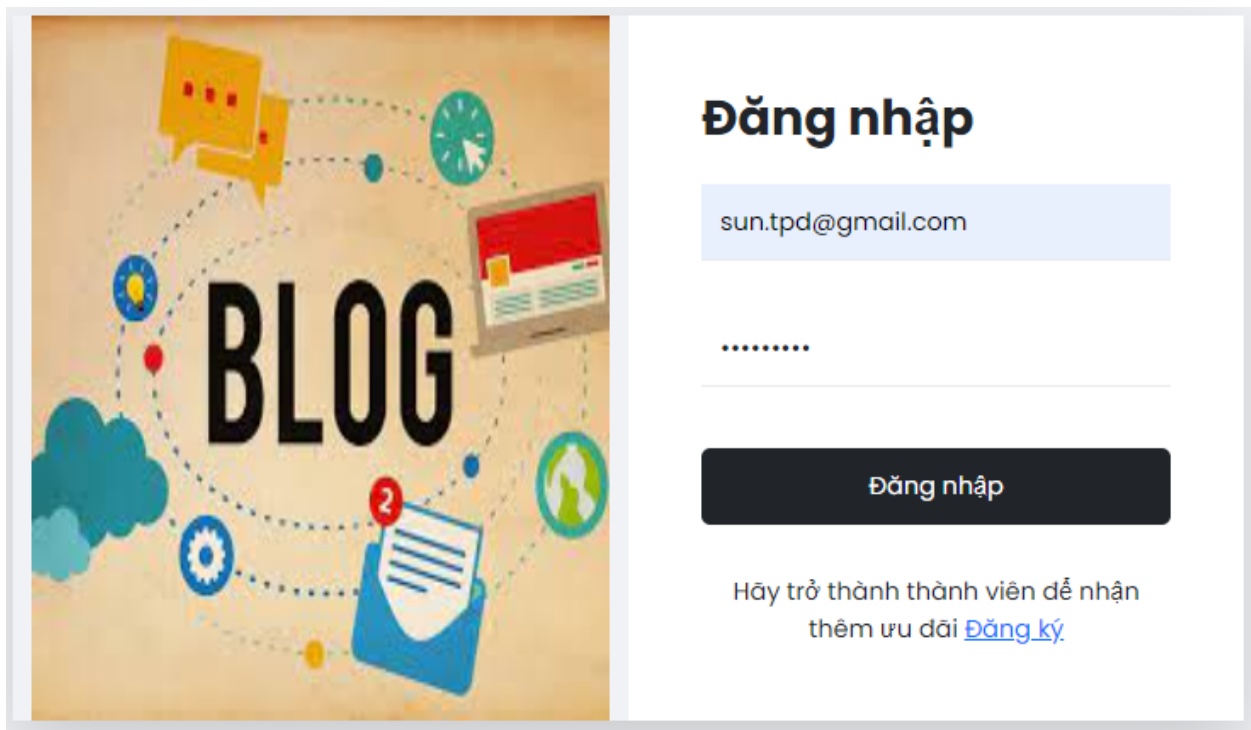
JWT là một phương pháp sử dụng mã thông báo để xác thực người dùng trong các ứng dụng web. JWT có thể lưu trữ thông tin về người dùng và các phiên làm việc của họ trong mã thông báo, giúp giảm thiểu việc lưu trữ session trên máy chủ và giúp giảm nguy cơ tấn công CSRF.

Các phần của JWT bao gồm Header, Payload và Signature. Khi người dùng đăng nhập vào hệ thống, một mã JWT sẽ được tạo ra và gửi đến trình duyệt, được lưu trữ trong một cookie hoặc tại local của user.

Mọi request cần xác thực user sẽ phải gửi kèm theo mã JWT, giúp máy chủ xác thực và xác định liệu yêu cầu có hợp lệ hay không.

V. DEMO

1. Cách thức hoạt động của mô hình Client-Server khi sử dụng cookie để xác thực người dùng



Giao diện đăng nhập

Request URL: http://localhost:3004/users/signin
Request Method: POST
Status Code: 200 OK
Remote Address: [::1]:3004
Referrer Policy: strict-origin-when-cross-origin

Set-Cookie: user=%7B%22id%22%3A%221%22%2C%22fullname%22%3A%22dungdung
vatar.jpg%22%2C%22createdAt%22%3A%222023-04-09T06%3A26%3A17.112Z%22%

Khi thực hiện đăng nhập, Client sẽ gửi 1 request lên phía Server để xử lý. Server sau khi thực hiện xong sẽ lưu thông tin người dùng vào cookie

The screenshot shows the Chrome DevTools 'Storage' tab with 'Cookies' selected for the URL 'http://localhost:3000'. A table lists cookies, with the 'user' cookie highlighted. Below the table, the 'Cookie Value' is shown as a decoded JSON string.

Name	Value	Domain	Path	Expires	Size	HttpOnly	Secure	SameSite	Partitioned
user	%7B%22id%22%3A%221%22%2C%22fullname%22%3A%22dungdung	localhost	/	Session	306		✓	Strict	
_ga_FR8K74HL5H	GS1.1.1680355962.5.0.168...	localhost	/	20...	51				
_ga	GA1.1.2003527704.167956...	localhost	/	20...	30				

Cookie Value ☒ Show URL decoded
{"id":"1","fullname":"dungdung","email":"sun.tpd@gmail.com","role":"user","image":"default-avatar.jpg","createdAt":"2023-04-09T06:26:17.112Z","updatedAt":"2023-04-09T06:26:17.112Z","deletedAt":null}

Sau khi đăng nhập thành công, thông tin tài khoản người dùng sẽ được lưu trong cookie. Server sẽ sử dụng phần dữ liệu này để xác thực xem là tài khoản nào đang sử dụng và thực hiện những chức năng của hệ thống.

Ví dụ: thực hiện tạo 1 bài post



dungdung

Public

Hôm nay trời đẹp quá



Đăng

▼ General

Request URL: `http://localhost:3004/posts`

Request Method: `POST`

Status Code: ● 201 Created

Remote Address: `[::1]:3004`

Referrer Policy: `strict-origin-when-cross-origin`

▼ Response Headers

[View source](#)

`Access-Control-Allow-Credentials: true`

`Access-Control-Allow-Origin: http://localhost:3000`

`Connection: keep-alive`

`Content-Length: 0`

`Date: Wed, 12 Apr 2023 10:49:53 GMT`

`Keep-Alive: timeout=5`

▼ Request Headers

View source

Accept: application/json, text/plain, */*

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US,en;q=0.9,vi;q=0.8

Connection: keep-alive

Content-Length: 165

Content-Type: multipart/form-data; boundary=----WebKitFormBoundarytPMVUanx4a1XomZA

Cookie: _ga=GA1.1.2003527704.1679565375; _ga_FRBK74HL5H=GS1.1.1680355962.5.0.1680355962.0.0.0; user=%7B%22id%22%3A%221%22%2C%22fullname%22%3A%22dungdung%22%2C%22email%22%3A%22sun.tpd%40gmail.com%22%2C%22role%22%3A%22user%22%2C%22image%22%3A%22default-avatar.jpg%22%2C%22createdAt%22%3A%222023-04-09T06%3A26%3A17.112Z%22%2C%22updatedAt%22%3A%222023-04-09T06%3A26%3A17.112Z%22%2C%22deletedAt%22%3Anull%7D

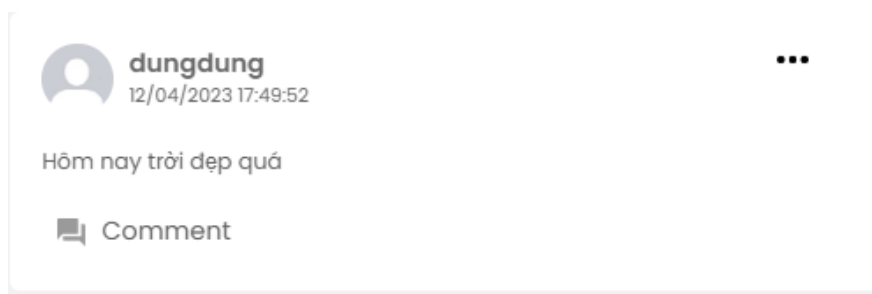
Host: localhost:3004

Origin: http://localhost:3000

Referer: http://localhost:3000/

Client sẽ gửi 1 request lên phía Server để xử lý kèm theo thông tin của người thực hiện được lưu trong cookie

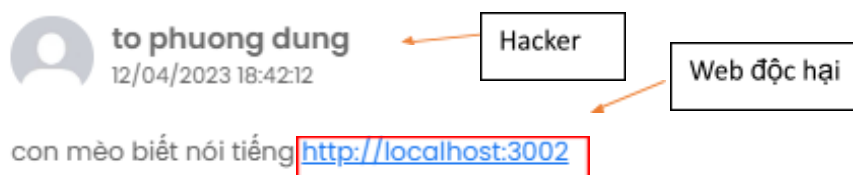
```
try {
  const user = JSON.parse(req.cookies.user);
  if (!user) {
    throw new Unauthorized('Cookie unauthorized');
  }
  if (roles.length && !roles.some((role) => role === user.role)) {
    throw new Forbidden('Forbidden accessible');
  }
  req.user = user;
  return next();
} catch (error) {
  const err = new Unauthorized(error.message)
  next(err)
}
```



Phía Server sẽ dựa vào cookie được lưu khi đăng nhập để biết ai là người thực hiện chức năng đó, sau đó xử lý và trả về kết quả

2. Cách thức tấn công CSRF

Hacker tạo 1 bài post kèm theo đường link dẫn đến trang web độc hại



Lợi dụng sự tò mò để dụ dỗ người dùng truy cập vào trang web độc hại

▼ General

Request URL: http://localhost:3004/posts
Request Method: POST
Status Code: 201 Created
Referrer Policy: strict-origin-when-cross-origin

▼ Response Headers

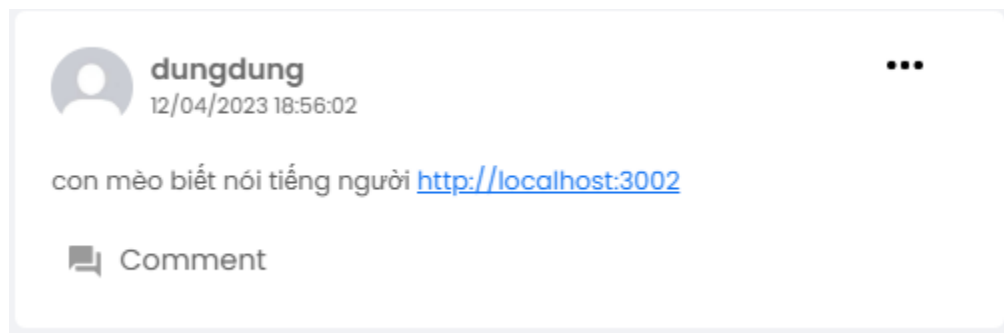
View source

Access-Control-Allow-Credentials: true
Connection: keep-alive
Content-Length: 0
Date: Wed, 12 Apr 2023 11:56:33 GMT
Keep-Alive: timeout=5
Vary: Origin
X-Powered-By: Express

Cookie: _ga=GA1.1.2003527704.1679565375; _ga_FRBK74HL5H=GS1.1.1680355962.5.0.1680355962.0.0.0; user=%7B%22id%22%3A%221%22%2C%22fullname%22%3A%22dungdung%22%2C%22email%22%3A%22sun.tpd%40gmail.com%22%2C%22role%22%3A%22user%22%2C%22image%22%3A%22default-avatar.jpg%22%2C%22createdAt%22%3A%222023-04-09T06%3A26%3A17.112Z%22%2C%22updatedAt%22%3A%222023-04-09T06%3A26%3A17.112Z%22%2C%22deletedAt%22%3Anull%7D

Host: localhost:3004
Origin: http://localhost:3002
Referer: http://localhost:3002/
sec-ch-ua: "Google Chrome";v="111", "Not(A:Brand";v="8", "Chromium";v="111"
sec-ch-ua-mobile: ?0

Khi người dùng truy cập vào trang web độc hại, nó sẽ tự động gửi một request kèm theo cookie của người dùng lên phía Server của trang web đích



Người dùng thực hiện tạo bài post theo mong muốn của hacker mà không hề hay biết

```
function App() {
  useEffect(() => {
    let bodyFormData = new FormData();
    const config = {
      headers: { "Content-type": "application/json" }
    }
    bodyFormData.append('content', 'con mèo biết nói tiếng người http://localhost:3002');
    axios.post(
      'http://localhost:3004/posts',
      bodyFormData,
      { withCredentials: true },
      config
    )
  }, [])
}
```

Mã nguồn trang web độc hại

Trang web độc hại sẽ tự động gửi 1 request đến Server của web đích khi người dùng truy cập vào.

3. Phương pháp phòng chống

3.1. Sử dụng JWT (JSON Web Token)

Cài đặt thư viện (Với NodeJS):

```
npm install jsonwebtoken
```

```
npm install @types/jsonwebtoken --save-dev
```

```
export async function signin(CreateUserDTO: LoginDTO) {
  const user = await userRepo.findOneBy({
    email: CreateUserDTO.email,
  })
  if (!user) {
    throw new BadRequest('email or password is incorrect');
  }
  const iPwd = bcrypt.compare(CreateUserDTO.password, user.password);
  if (!iPwd) {
    throw new BadRequest('email or password is incorrect');
  }
  const payload = { id: user.id, email: user.email, role: user.role };
  const accessToken = jwt.sign(payload, process.env.JWT_SECRET_KEY, { expiresIn: process.env.JWT_EXPIRATION });
  delete user.password;

  return { information: user, accessToken: accessToken };
}
```

Mã nguồn chức năng đăng nhập của server

- Ở phương pháp này chúng ta sẽ sử dụng 1 secret key để tạo ra các token gửi về cho user. Token này chúng ta sẽ đọc được nội dung của phần Header và Payload

do 2 phần này sử dụng hàm mã hóa. Riêng phần Signature sẽ không đọc được nội dung do phần này sử dụng hàm Hash để tạo ra.

[illegible]

```
try {  
  
  let response = await loginService(login);  
  const data = response && response.data ? response.data : '';  
  if (data) {  
    dispatch(userLogin(data.information))  
    cookies.save('user', data.information)  
    window.localStorage.setItem('Token', data.accessToken);  
  
    navigate('/');  
  }  
}
```

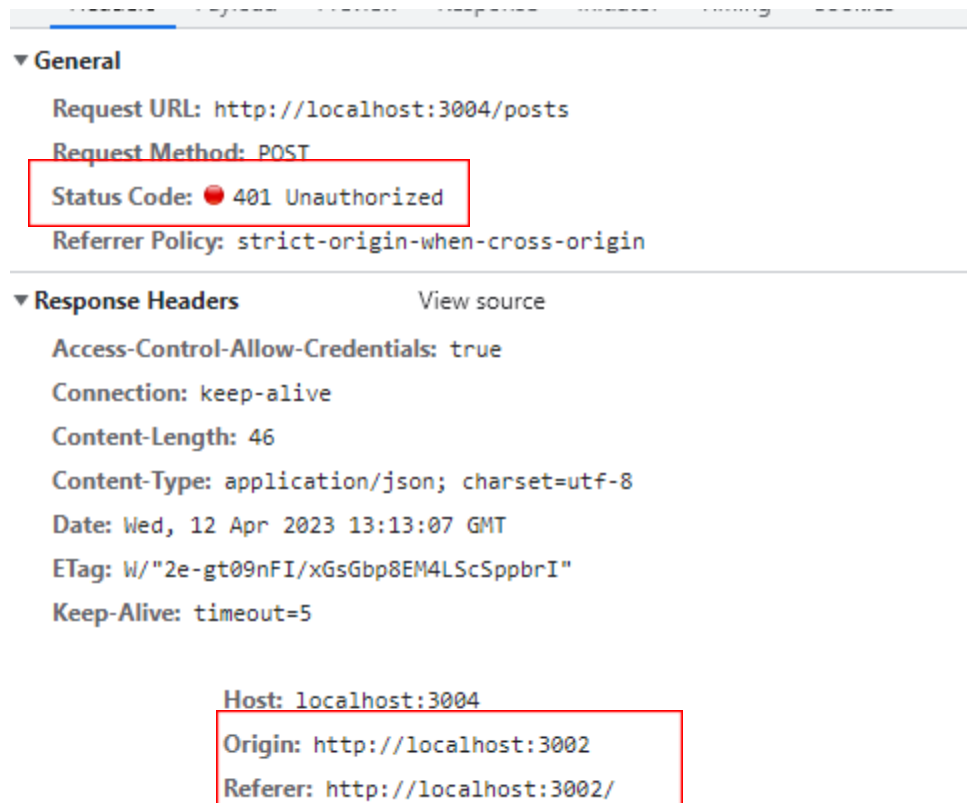
Khi đăng nhập vào hệ thống, Server sẽ tạo ra 1 Token chứa thông tin của người dùng và được mã hóa. Phía Client sẽ lưu Token vào local storage.

```
const createPost = async (post, token) => {
  const config = {
    headers: { Authorization: `Bearer ${token}` }
  };
  return await request.post(
    'posts',
    post,
    config
  ).then(data => console.log(data))
  .catch(error => console.log(error))
}
```

Phía Client khi request đến Server, phải gửi kèm theo Token trên headers

```
export function authorization(...roles: string[]) {
  return (req: Request, res: Response, next: NextFunction) => {
    try {
      const token = req.headers.authorization;
      if (!token || !token.startsWith('Bearer ')) {
        throw new Unauthorized('Token schema is invalid or missing');
      }
      const accessToken = token.replace('Bearer ', '');
      const user = jwt.verify(accessToken, process.env.JWT_SECRET_KEY, { ignoreExpiration: false });
      console.log(user);
      if (roles.length && !roles.some((role) => role === user.role)) {
        throw new Forbidden('Forbidden accessible');
      }
      req.user = user;
      return next();
    } catch (error) {
      const err = new Unauthorized(error.message);
      next(err);
    }
  }
}
```

Phía Server sẽ nhận Token và kiểm tra xác thực người dùng.



Khi người dùng truy cập vào trang web độc hại sẽ bị Server chặn ngay lập tức

Như vậy phía hacker muốn thực hiện tấn công bắt buộc phải có Token của người dùng, mà việc này rất khó để thực hiện. Nên phương pháp phòng chống này hiện tại là an toàn.

3.2. Sử dụng CSRF token

Phía Server:

Cài đặt thư viện :

```
> npm install csrf
```

Tạo một middleware mã csrf được cấu hình lưu trong cookie:

```
import csrf from 'csrf';
export const csrfProtection = csrf({
  cookie: true
});
```

Thực hiện tạo mã csrf:

```
app.use(cookieParser());
app.use(csrfProtection);
```

[illegible]

Cookie Value ☒ Show URL decoded
sEgvN3D3YiF-uFI7Os5ky-sE

Khi truy cập vào trang web, mã csrf sẽ tự động được tạo và lưu trong cookie

```
app.get('/api/csrf-token', (req: Request, res: Response) => {
  res.cookie('csrftoken', req.csrfToken());
  res.status(200).send('');
});
```

Tạo một API lưu mã csrf Token vào cookie hoặc bất cứ nơi nào Client có thể khai thác được.

Phía Client:

Thực hiện request lên Server để lấy mã csrf token:

```
useEffect(() => {
  request.get('/api/csrf-token')
}, [])
```

▼ General

Request URL: http://localhost:3004/api/csrf-token

Request Method: GET

Status Code: 200 OK

Remote Address: [::1]:3004

Referrer Policy: strict-origin-when-cross-origin

▼ [Response Headers](#) [View source](#)

[View source](#)

Access-Control-Allow-Credentials: true

Access-Control-Allow-Origin: http://localhost:3000

Content-Length: 0

Content-Type: text/html; charset=utf-8

Date: Wed, 19 Apr 2023 10:49:00 GMT

Etag: W/"0-2jmj715rSw0yVb/v1WAYkK/YBwk"

Set-Cookie: csrftoken=1aa53DV1-nJbQyKInBBGkxpUN9KXShH_AXis; Path=/; SameSite=Strict

Vary: Origin

X-Powered-By: Express

[View source](#)[View source](#)[illegible]

Csrf token đã được lưu trong cookie.

```
const token = cookies.load('csrftoken');
```

```

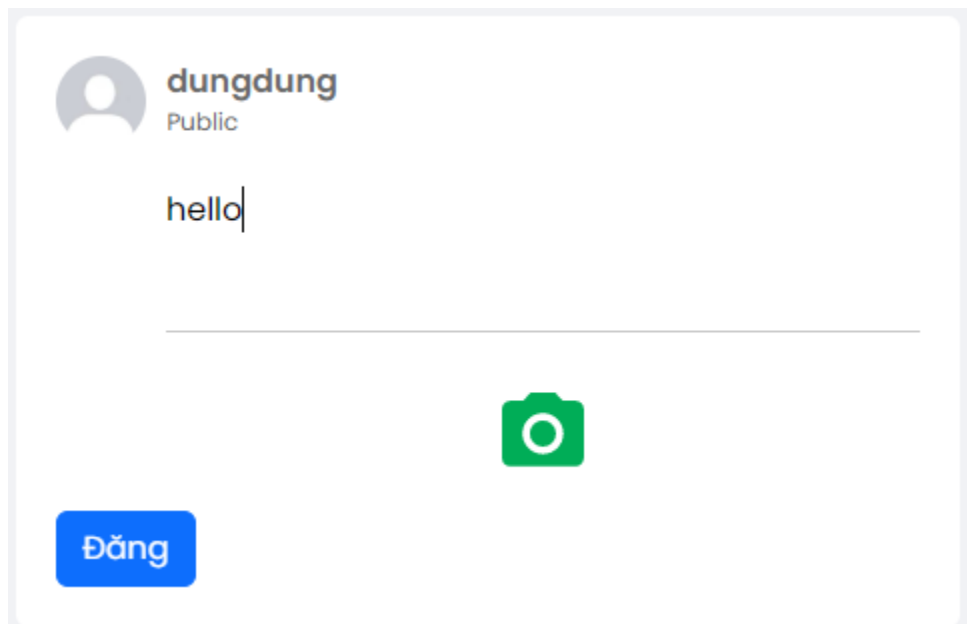
const createPost = async (post) => {
  const config = {
    headers: { "Content-Type": "multipart/form-data", 'X-CSRF-Token': token }
  };
  return await request.post(
    'posts',
    post,
    config
  )
}

const deletePost = async (post) => {
  const config = {
    headers: { 'X-CSRF-Token': token }
  };
  return await request.delete(
    'posts/' + post,
    config
  ).then(data => console.log(data))
  .catch(error => console.log(error))
}

```

Khi thực hiện một chức năng có yêu cầu xác thực từ phía Server. Phía Client cần gửi kèm csrf token trên headers

Ví dụ thực hiện tạo 1 bài post:



The screenshot shows a user profile for 'dungdung' with a 'Public' status. Below the profile, there is a text input field containing the word 'hello'. Underneath the text field is a horizontal line, and below that is a green camera icon. At the bottom left, there is a blue button with the text 'Đăng' (Post).

×	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
▼ General							
Request URL: http://localhost:3004/posts							
Request Method: POST							
Status Code: 201 Created							
Remote Address: [::1]:3004							
Referrer Policy: strict-origin-when-cross-origin							

Host: localhost:3004

Origin: http://localhost:3000

Referer: http://localhost:3000/

sec-ch-ua: "Chromium";v="112", "Google Chrome";v="112", "Not:A-Brand";v="99"

sec-ch-ua-mobile: ?0

sec-ch-ua-platform: "Windows"

Sec-Fetch-Dest: empty

Sec-Fetch-Mode: cors

Sec-Fetch-Site: same-site

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36

X-CSRF-Token: 1aa53DV1-nJbQyKInBBGkxpUN9KXShH_AXis

CSRF Token kèm theo

Truy cập vào trang web độc hại do hacker tạo ra:

Name	Headers	Payload	Preview	Response	Initiator	Ti
react_devtools_backend.js	▼ General					
localhost	Request URL: http://localhost:3004/posts					
installHook.js	Request Method: POST					
bundle.js	Status Code: 500 Internal Server Error					
react_devtools_backend.js	Referrer Policy: strict-origin-when-cross-origin					
logo.6ce24c58023cc2f8fd88fe9d...	▼ Response Headers View source					
posts	Connection: keep-alive					
posts	Content-Length: 33					
ws	Content-Type: application/json; charset=utf-8					
favicon.ico	Date: Wed, 19 Apr 2023 11:12:51 GMT					
manifest.json	ETag: W/"21-Fau8GdrOCoyGNNH/IiTx2DuMu0"					
logo192.png	Keep-Alive: timeout=5					
nr-ext-select-icon.html	X-Powered-By: Express					
icon@2x.png	▼ Request Headers View source					
	Accept: application/json, text/plain, */*					
	Accept-Encoding: gzip, deflate, br					

14 requests | 1.8 MB transferred | ...

```
ForbiddenError: invalid csrf token
    at csrf (D:\HOCKY2NAM4\ATBM\project\soucre-code\atbm-backend\node_modules\csrf\index.js:112:19)
    at Layer.handle [as handle_request] (D:\HOCKY2NAM4\ATBM\project\soucre-code\atbm-backend\node_modules\express\lib\router\layer.js:95:5)
    at trim_prefix (D:\HOCKY2NAM4\ATBM\project\soucre-code\atbm-backend\node_modules\express\lib\router\index.js:328:13)
    at D:\HOCKY2NAM4\ATBM\project\soucre-code\atbm-backend\node_modules\express\lib\router\index.js:286:9
    at Function.process_params (D:\HOCKY2NAM4\ATBM\project\soucre-code\atbm-backend\node_modules\express\lib\router\index.js:346:12)
    at next (D:\HOCKY2NAM4\ATBM\project\soucre-code\atbm-backend\node_modules\express\lib\router\index.js:280:10)
    at cookieParser (D:\HOCKY2NAM4\ATBM\project\soucre-code\atbm-backend\node_modules\cookie-parser\index.js:71:5)
    at Layer.handle [as handle_request] (D:\HOCKY2NAM4\ATBM\project\soucre-code\atbm-backend\node_modules\express\lib\router\layer.js:95:5)
    at trim_prefix (D:\HOCKY2NAM4\ATBM\project\soucre-code\atbm-backend\node_modules\express\lib\router\index.js:328:13)
    at D:\HOCKY2NAM4\ATBM\project\soucre-code\atbm-backend\node_modules\express\lib\router\index.js:286:9 {
  code: 'EBADCSRFTOKEN'
}
```

Khi người dùng truy cập vào trang web độc hại thì sẽ bị Server báo lỗi csrf token không hợp lệ. Điều này sẽ giúp ngăn chặn tấn công CSRF rất hiệu quả.

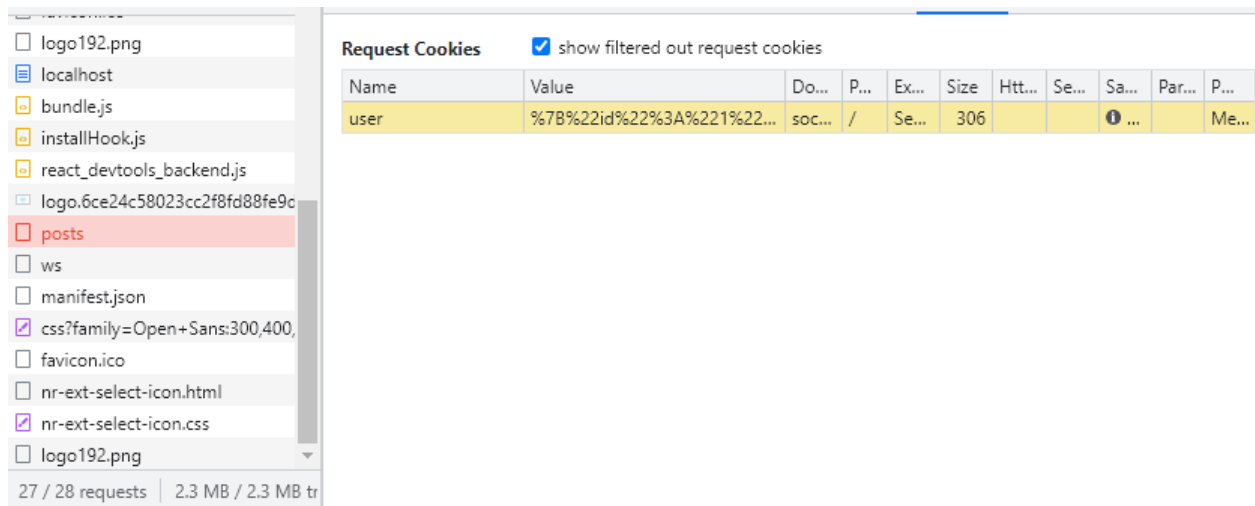
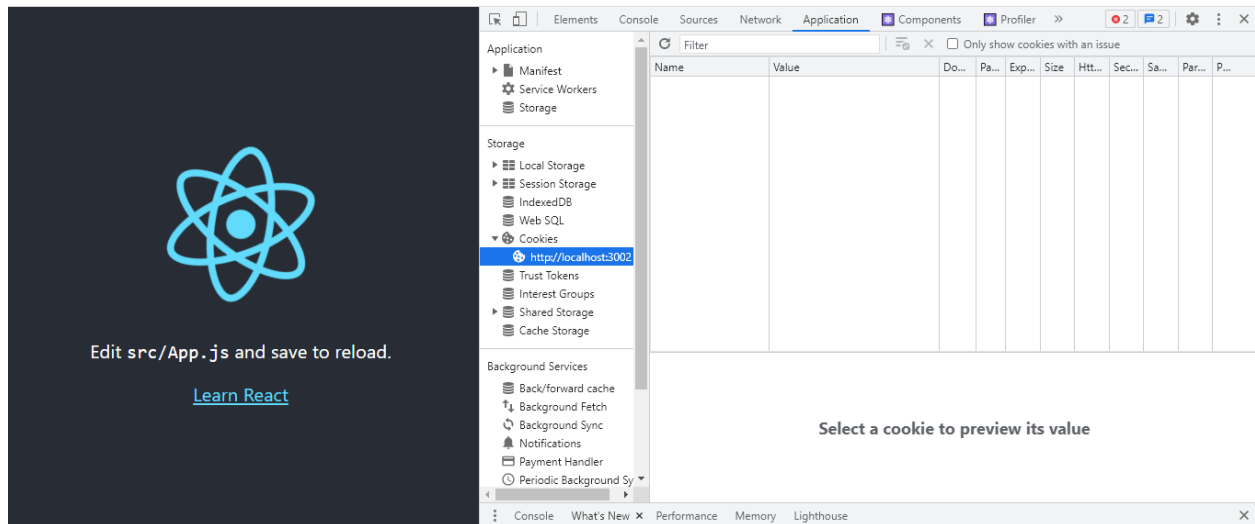
3.2. Sử dụng Samesite cookie

```
export async function signin(req: Request, res: Response, next: NextFunction) {
  try {
    const schema = Joi.object({
      email: Joi.string().email().required(),
      password: Joi.string().min(8).required()
    })
    const value = validate<LoginDTO>(req.body, schema);
    const result = await userService.signin(value);
    //res.cookie('user', JSON.stringify(result.information), );
    res.cookie('user', JSON.stringify(result.information), { sameSite: 'strict' });
    return res.status(200).send(result);
  } catch (error) {
    return next(error);
  }
}
```

[illegible]☒ Show URL decoded

```
{
  "id": "1",
  "fullName": "dungdung",
  "email": "sun.tpd@gmail.com",
  "role": "user",
  "image": "default-avatar.jpg",
  "createdAt": "2023-04-09T06:26:17.112Z",
  "updatedAt": "2023-04-09T06:26:17.112Z",
  "deletedAt": null
}
```

Khi đăng nhập phía Server sẽ lưu thông tin user vào cookie với thuộc tính sameSite là strict.



Khi người dùng truy cập vào trang web độc hại, sẽ không thể lấy được cookie ở trang web đích. Bằng cách này chúng ta có thể chặn được các cuộc tấn công CSRF.

VI. KẾT LUẬN

Tấn công CSRF là một trong những kỹ thuật tấn công nguy hiểm đối với các ứng dụng web. Khi phát triển các ứng dụng web, cần phải đảm bảo tính bảo mật cao để ngăn chặn việc kẻ tấn công được áp dụng kỹ thuật tấn công này.

Việc sử dụng các phương pháp như CSRF token, JWT và SameSite cookie đều đóng vai trò rất quan trọng trong việc chống lại tấn công CSRF. Một cách tốt nhất để bảo vệ ứng dụng của bạn là sử dụng đồng thời các phương pháp này và áp dụng các chính sách bảo mật khác như input validation, escape output, rate limiting,...

VII. TÀI LIỆU THAM KHẢO

"Cross-Site Request Forgery (CSRF)". OWASP Foundation.

"JSON Web Tokens (JWT)". jwt.io.

"Cookies". Mozilla Developer Network.

Chat GPT: <https://voicegpt.us/chatgpt>.

CSRF token: <https://www.youtube.com/watch?v=gMre7FBbchQ&t=1194s>