# CSRF



## What is it

CSRF - Cross site request forgery

CSRF is an attack technique that attempts to circumvent a defensive technique that is marked by CSRF tokens.

Say you are a website builder and you are creating a new website. You create the profile section which allows you to update your address. Now along comes a bad actor. They analyse the request and are able to forge it. They create their own website and they put a button on there which will call the profile section of your website and which will update the address.

This means that the attacker can update my address from his website. This may seem pretty innocent but what if instead we replace the functionality with a bank? What if the attacker can send money from the current active account to his account? That would change the matter entirely.

To prevent this, you as a website builder have several options. One of them is implementing a CSRF token. This token is an extra parameter for your request and is generated on the server and visible to the browser but only via your website. As an attacker, there is no way to gain access to this token without using some illicit tactics (which we will dig deeper into later).  If the attacker wants to make the same request, he is missing the CSRF token parameter which will not complete the request and return an error. Hack successfully blocked... or is it?

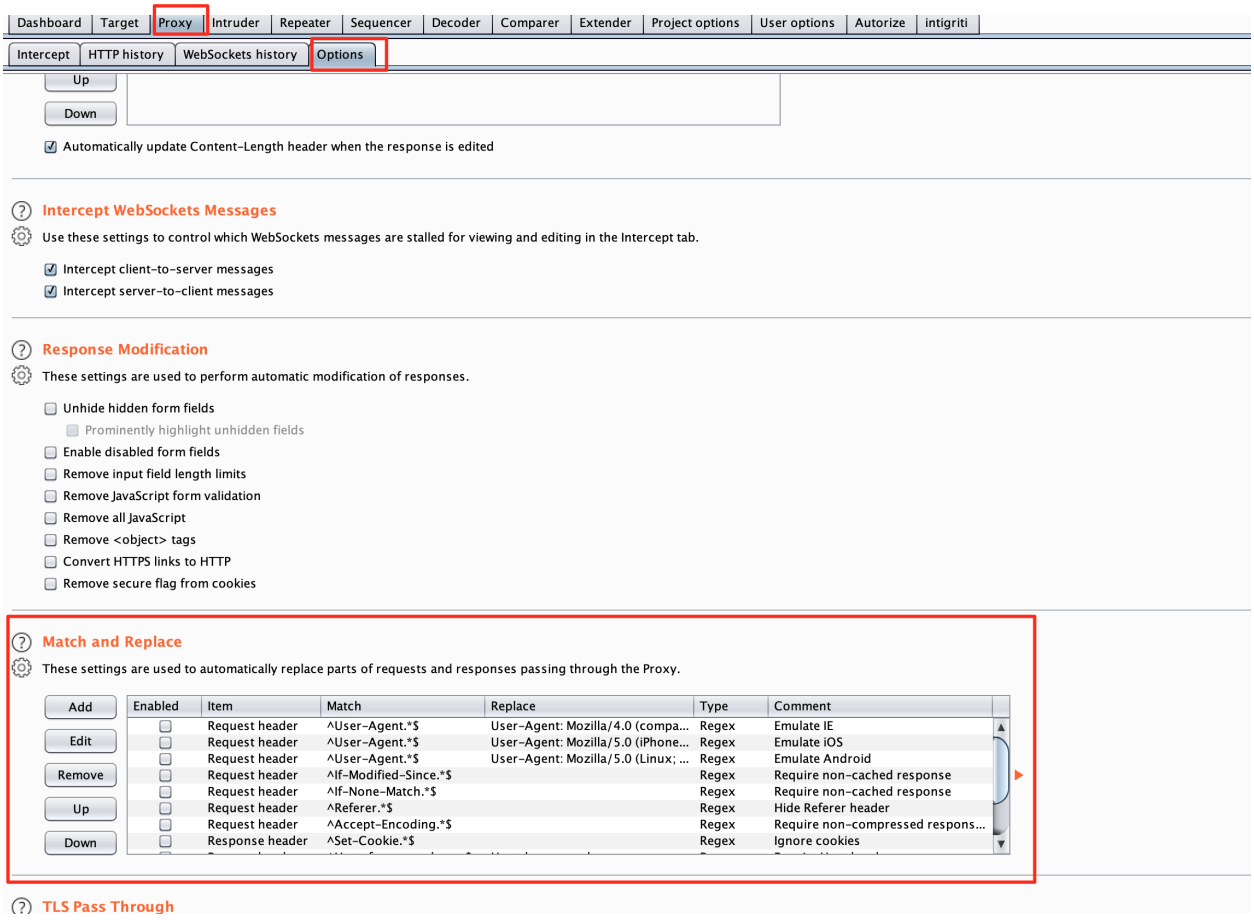
## Attack strategy

Though the idea of CSRF tokens is very solid, It's easy to mess up the implementation. We as pentesters have several options to test for:

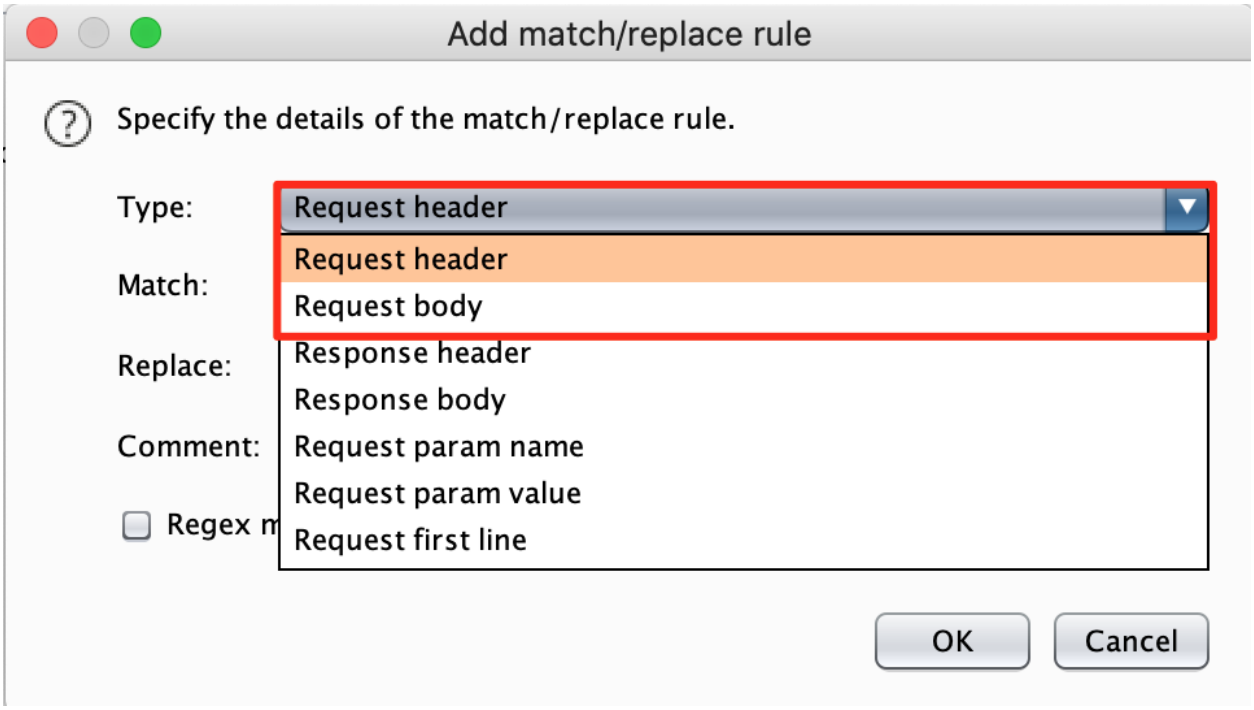
- Remove the CSRF token from requests
- Replace the CSRF token with a random value (for example 1)
- Replace the CSRF token with a random token of the same restraints
- Leave CSRF Parameter empty
- Use a CSRF token that has been used before
- See if you can request a CSRF by executing the call manually and use that token for the request

## Automation

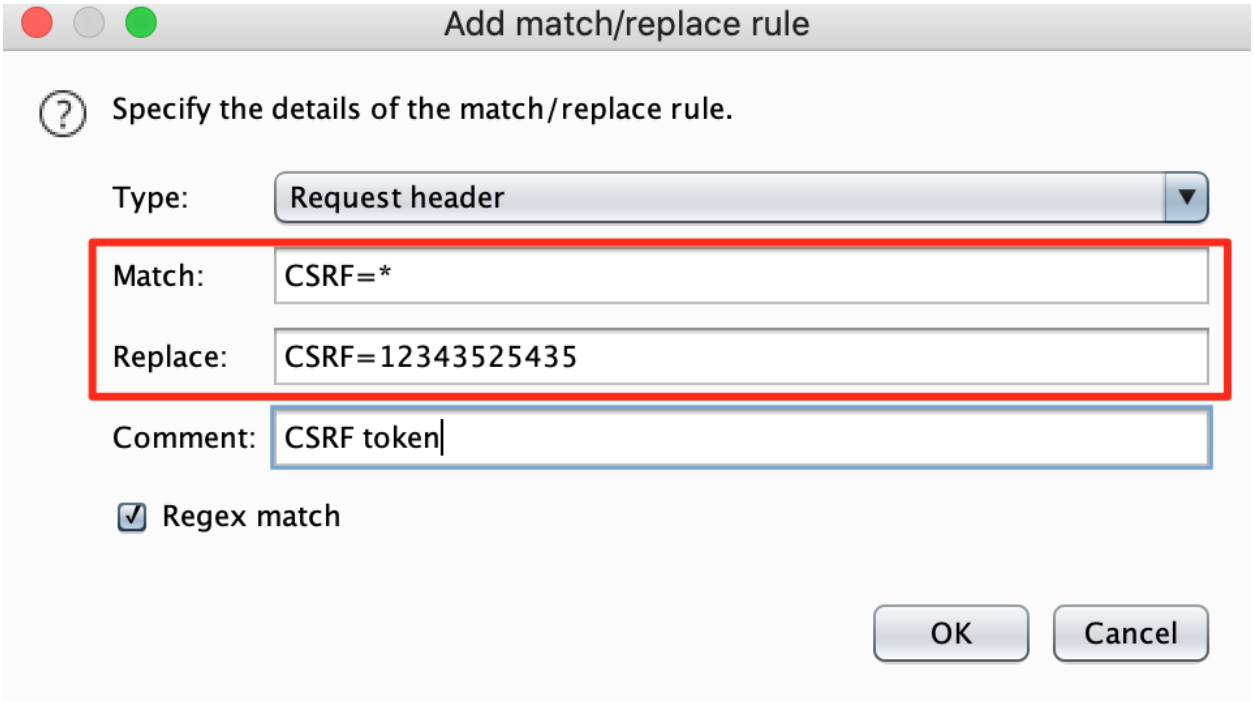
Yes, it is possible to automate this. We will be using the match and replace functions of burp.



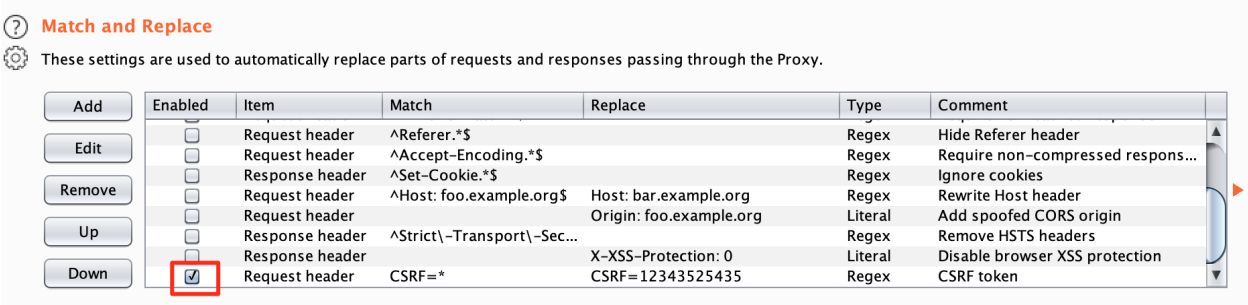

Depending on if the CSRF token is in the HEADER or the BODY section of the request, we will need to pick one.




Fill in the regex to indicate to burp how it can find the CSRF token in your request and replace it with a value of your own. Be careful, this is just an example, it may be different for your target.




When this rule is active, click through the application and try to make changes. If you are able to make changes where a CSRF token is normally expected, investigate this further. It may be a vulnerability. To restore normal functionality, simply disable this rule.

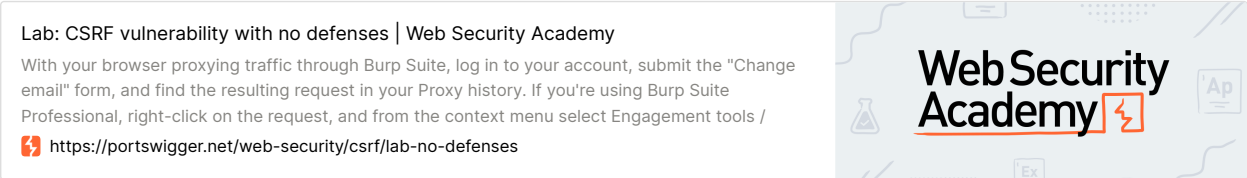



# Practical methodology

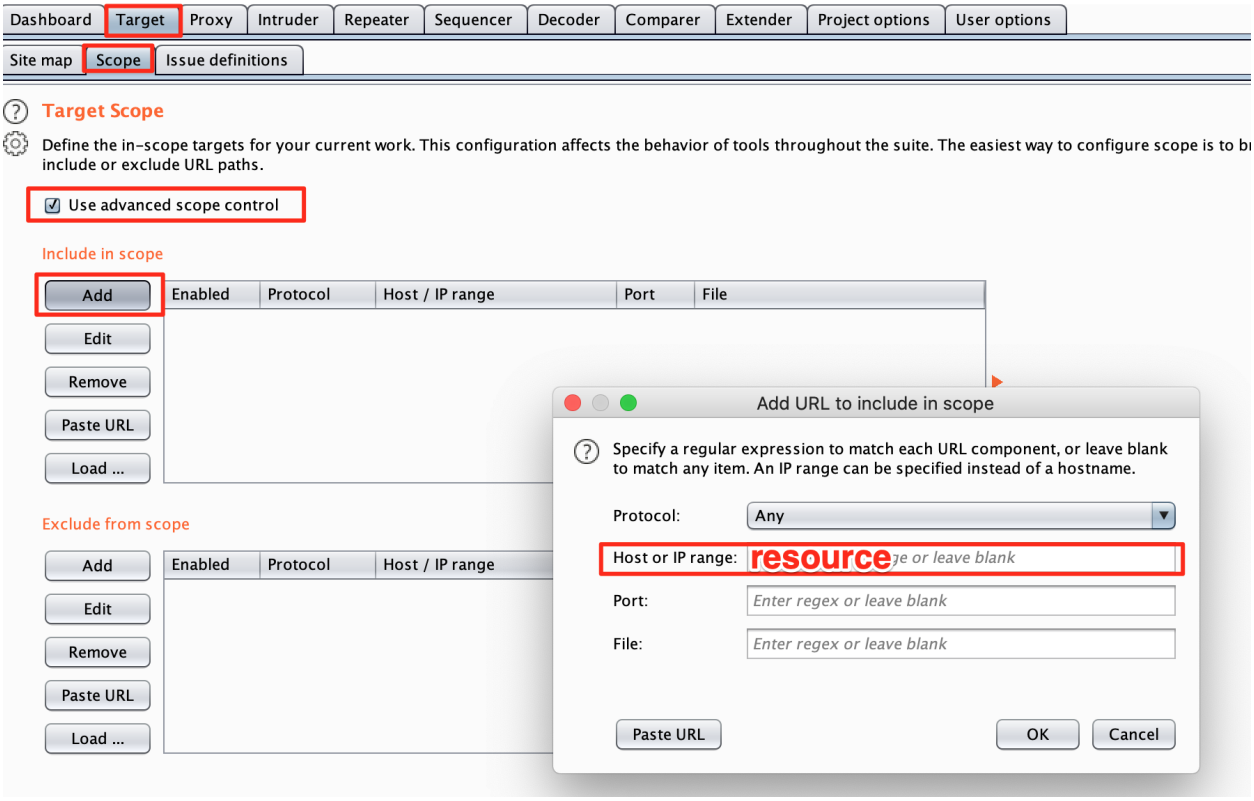
## Portswigger labs

We will be using the portswigger labs for this section and we will be creating our methodology in this section.

### CSRF vulnerability with no defenses




- Log into the academy, start the lab and open up your burp suite.
- Copy the URL of the lab, but only the resource name
  - ~~https://~~ac1d1fda1ed2a20f80ed507400ce0053.web-security-academy.net/~~post?postId=6~~
    - Only the first part of the URL, make sure you also remove the https in front of the URL:
  - ac1d1fda1ed2a20f80ed507400ce0053.web-security-academy.net
- Navigate to the target tab and open the scope tab.
- Make sure the "Use advanced scope control" checkmark is checked
- Click the "Add" button
- Paste the resource



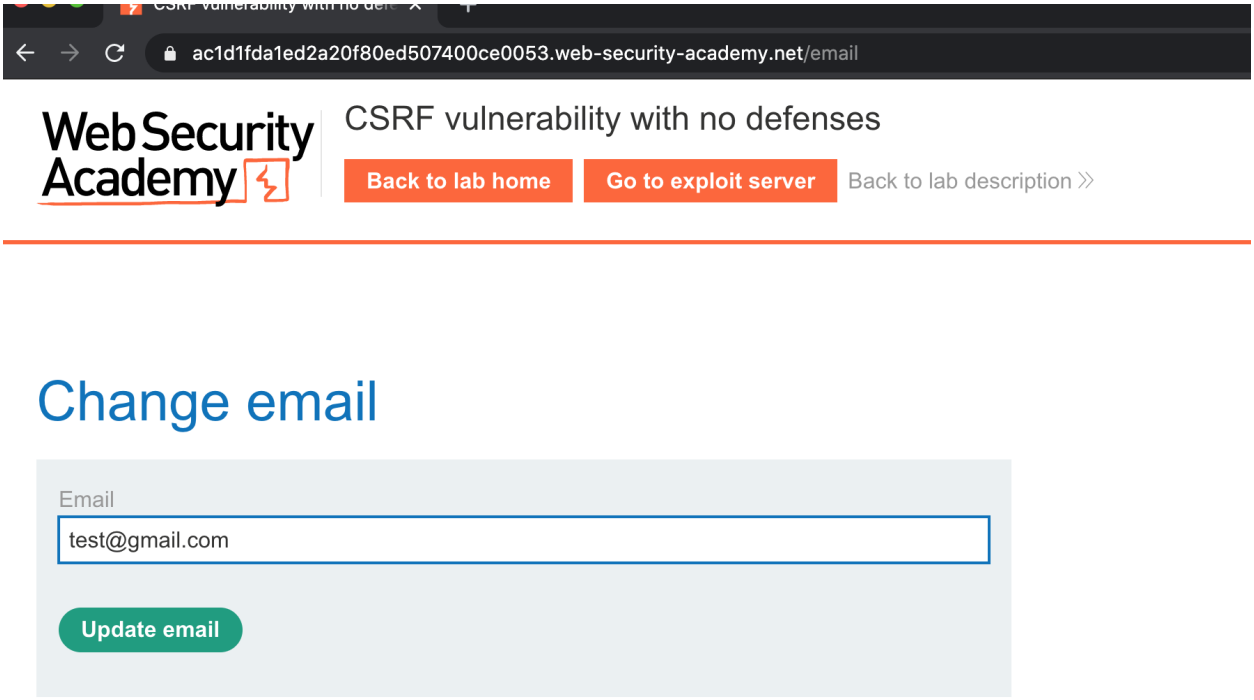

- Go to the proxy tab and open a browser
- Paste the full URL of the lab

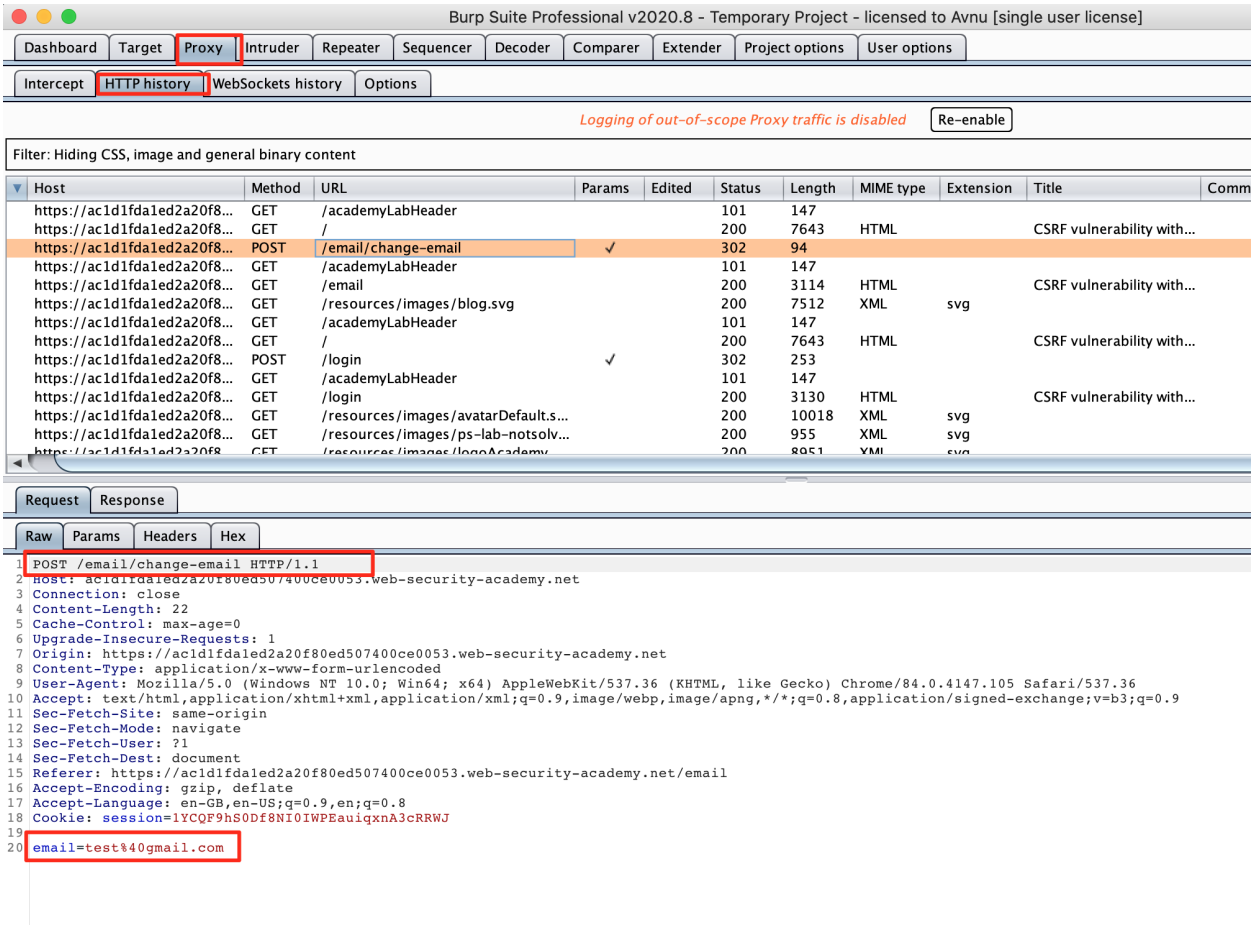
When we open up this lab, the first thing we have to do is look at what functionality is available.

We can see that we have a blogging website where we can:

- Login (carlos / montoya)
- Leave a comment
- Change our email

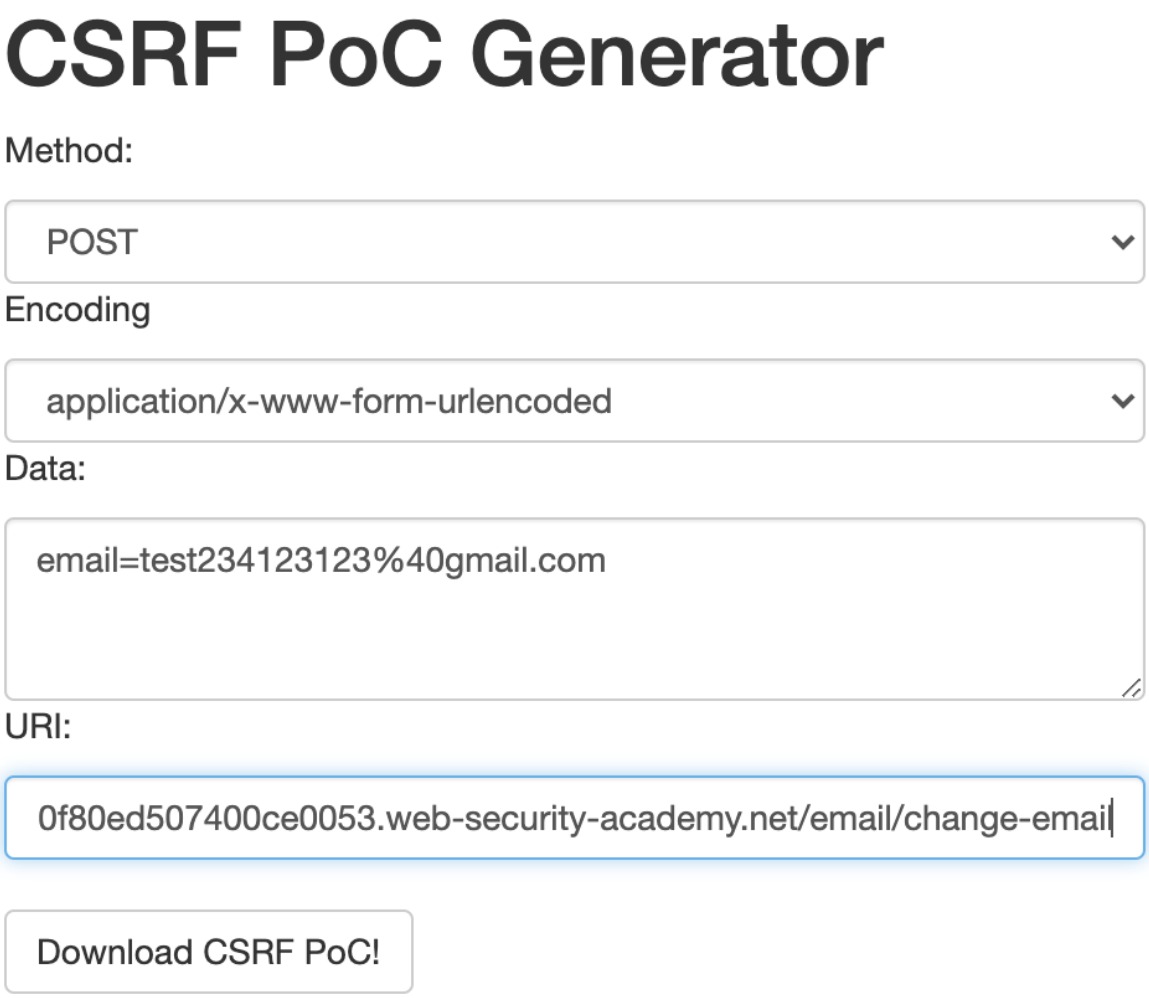
Let's try to change our email



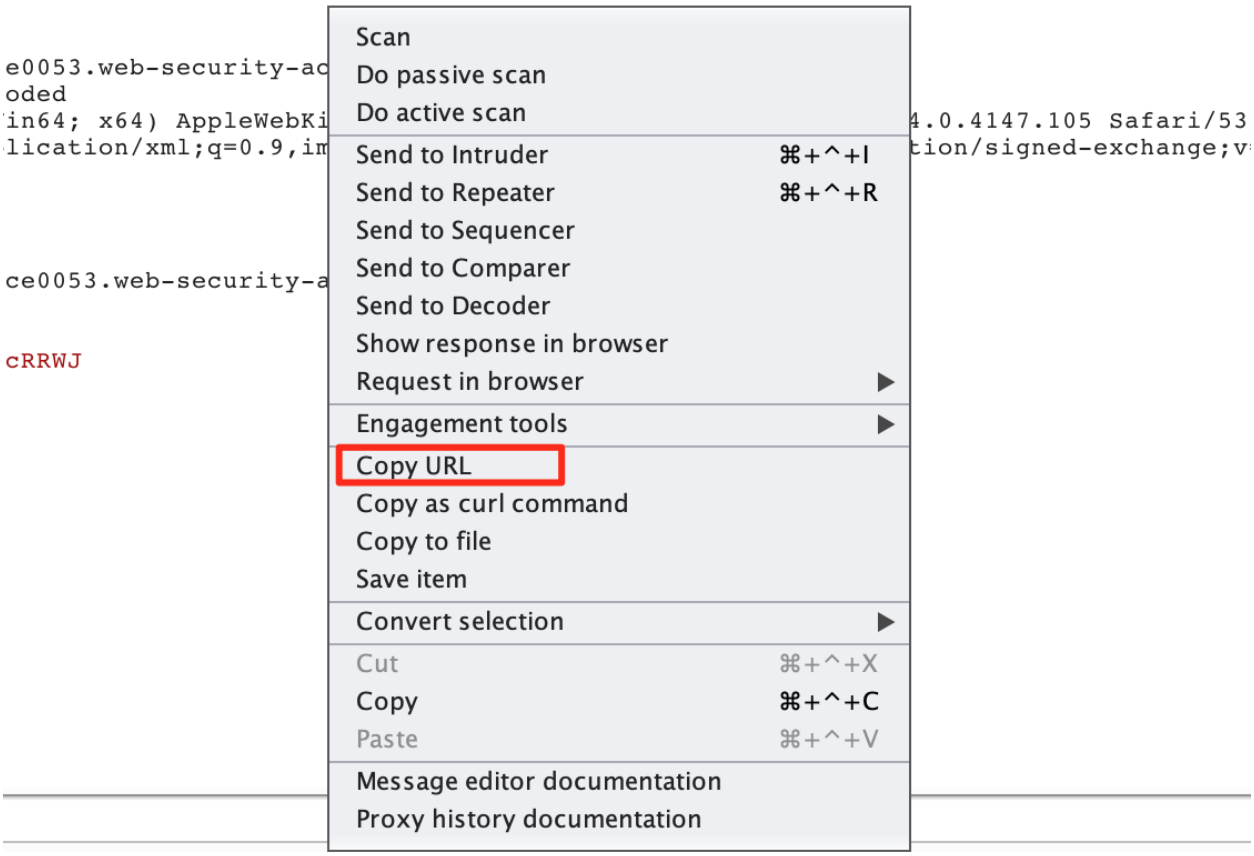



We can now do the see a request being made to POST /email/change-email with no CSRF token in place.

Now navigate to https://security.love/CSRF-PoC-Genorator/ , we are going to generate a PoC.




Copy the email parameter from the request in burp and paste it in the PoC generator, make sure you make some changes so you can verify this CSRF attack later on. To grab the URI, we can right click on our request in burp and click on "Copy URL" from the dropdown menu.




Now Download the PoC, it should download an HTML file. When you open this file, you should see a button. Make sure you are logged in in another tab and click the button. This should change the users email adress. Verify this if you are doing bug bounties.

For this lab, we can not complete it in this way and we can not verify the change. To complete the lab, open it again. There will be a button "Go to exploit server", click it and paste the following code into the "body" field:

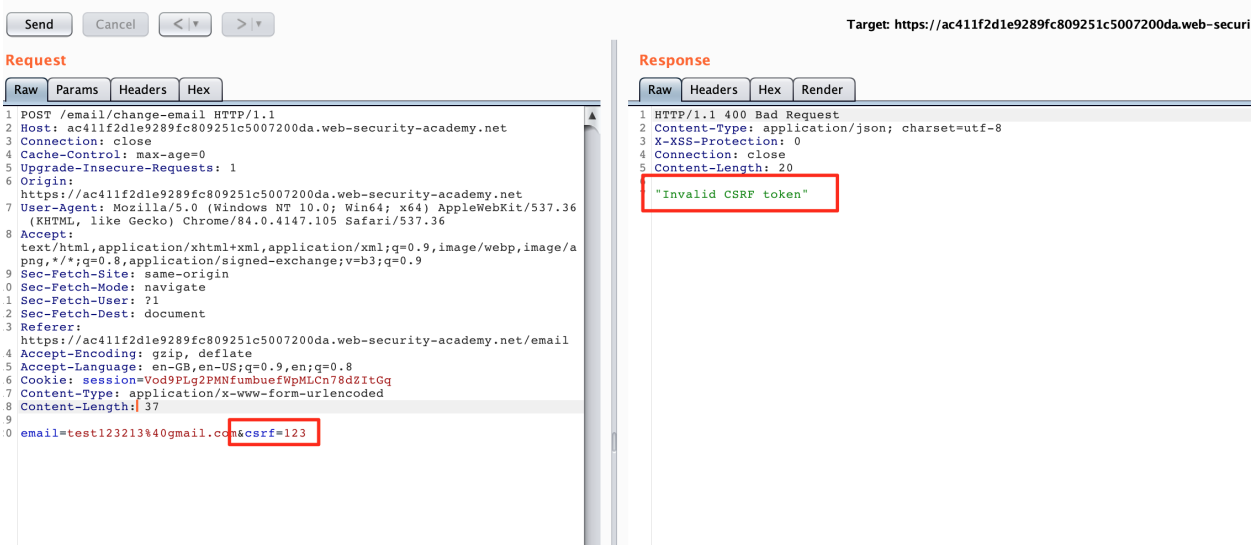
```
<form method="POST" action="https://ac1d1fda1ed2a20f80ed507400ce0053.web-security-academy.net/email/change-email">
     <input type="hidden" name="email" value="test231324@gmail.com">
</form>
<script>
      document.forms[0].submit();
</script>
```

Make sure you replace the action URL with the URL of your lab. Next click "store" and then click "exploit".

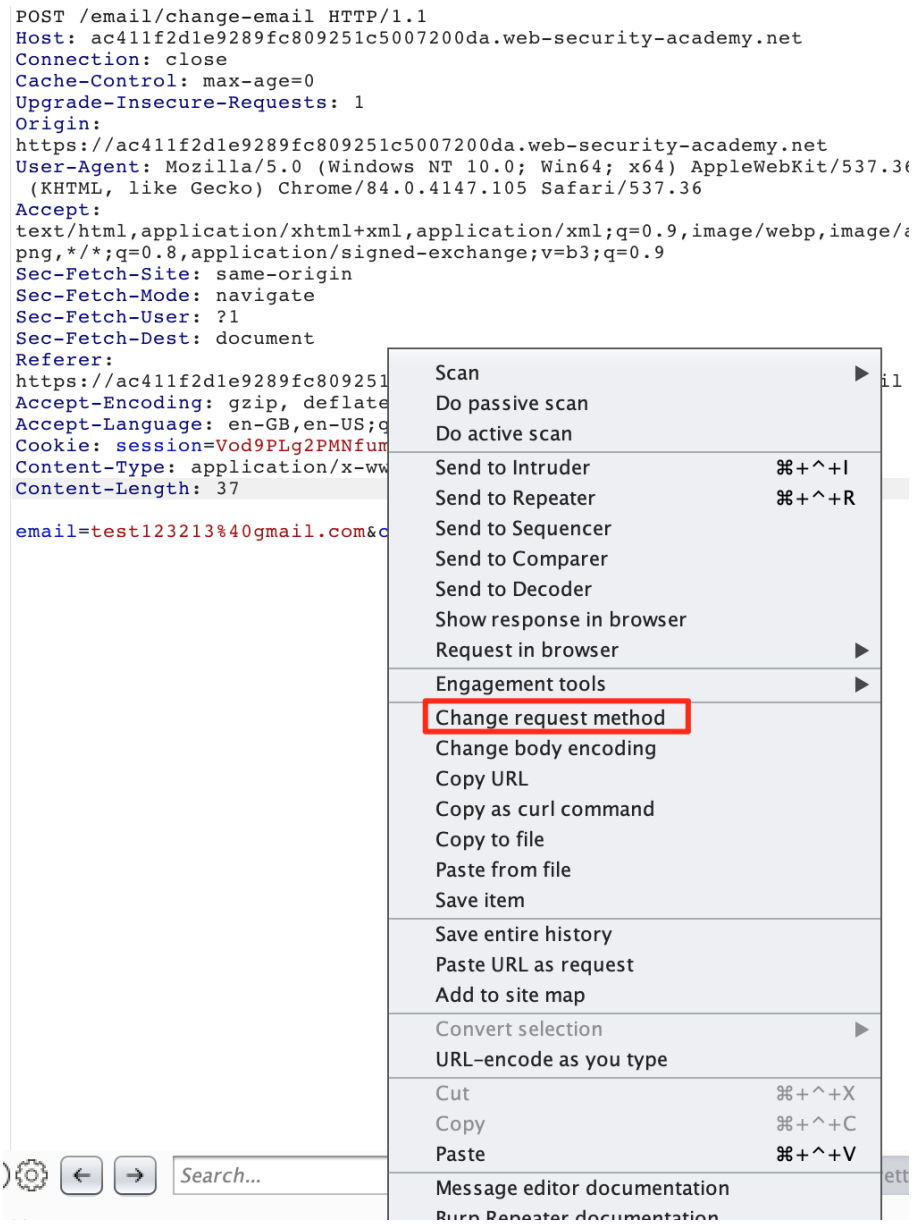
## CSRF where token validation depends on request method

We repeat the same steps as before until we change our email one time. We should now see another request to POST /email/change-email. Let's right click this request and send it to the repeater.

If we change our provided CSRF token, we can see that our request is not accepted.



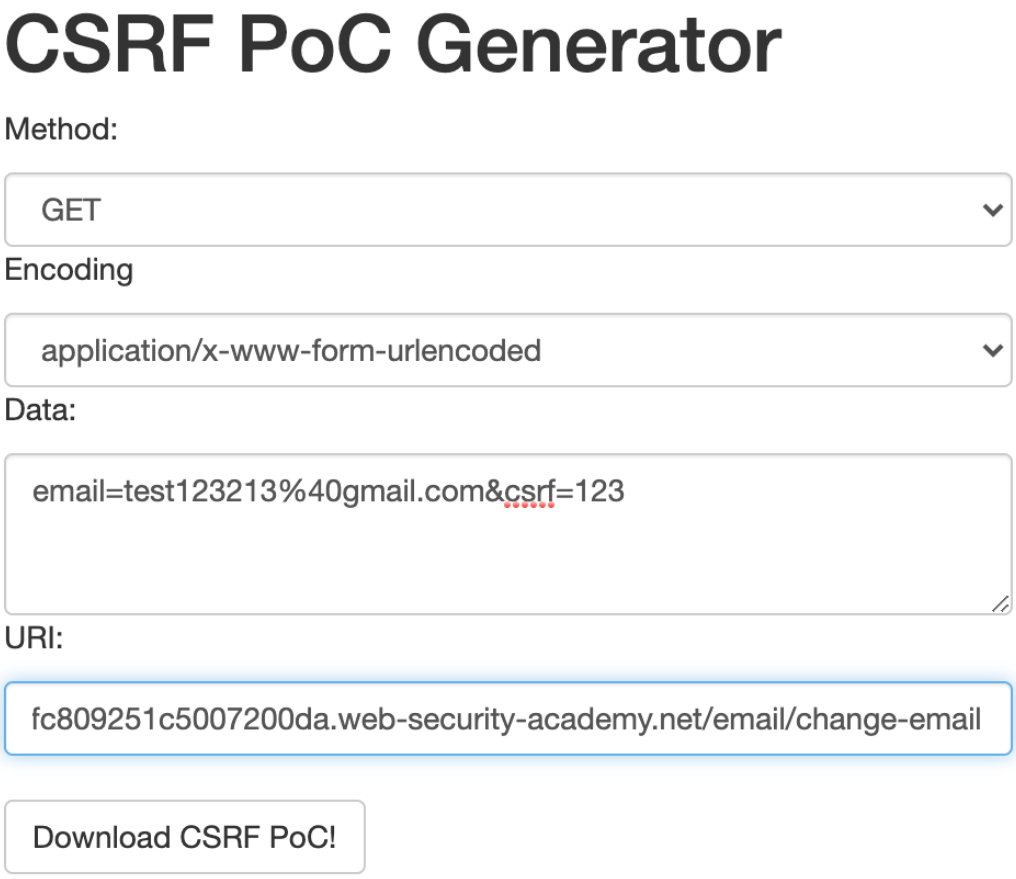

However we can also send our POST parameters as GET parameters in a GET request. Some servers also accept GET requests instead of POST request. Burp can easily transform our method by right clicking the request and picking "change request method" from the context menu.




This will give us a GET request:

```
GET /email/change-email?email=test123213%40gmail.com&csrf=123 HTTP/1.1
Host: ac411f2d1e9289fc809251c5007200da.web-security-academy.net
Connection: close
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: https://ac411f2d1e9289fc809251c5007200da.web-security-academy.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.105 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://ac411f2d1e9289fc809251c5007200da.web-security-academy.net/email
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: session=Vod9PLg2PMNfumbuefWpMLCn78dZItGq
```

Which we can again try to enter in our CSRF PoC generator to prove impact.




To solve this lab, we need to make some changes to the payload from the previous lab.

```
<form method="GET" action="https://ac1d1fda1ed2a20f80ed507400ce0053.web-security-academy.net/email/change-email">
     <input type="hidden" name="email" value="test231324@gmail.com">
</form>
<script>
      document.forms[0].submit();
</script>
```

We can again paste this payload in our exploit server and execute the exploit to complete the lab.

## Validation of CSRF token depends on token being present

For this lab, we need to repeat the same steps as before, but this time, we need to remove the CSRF token. Sometimes servers validate a token when it's there and give an error when it's not correct but they might not validate any token if it's not there.