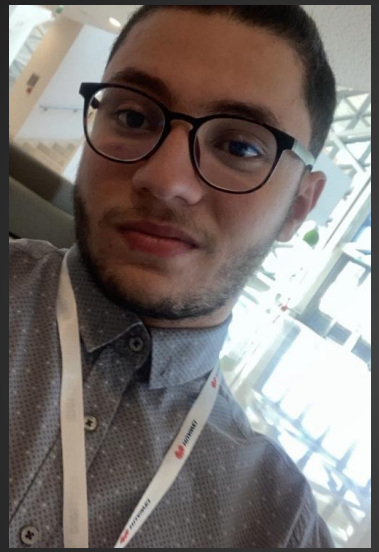
Client-side Security

Client-side web exploitation : attacks and mitigation

BSIDES ALGIERS

Mokrane Abdelmalek

3rd year CS student at ESI, Dev co-manager at GDG Algiers.



Oussama Merouan

Third year student at ESI, Technical department manager at Shellmates.

Cyber security enthusiasts and CTF players at *@ShellSec* and *@CyberErudites*

BSIDES ALGIERS

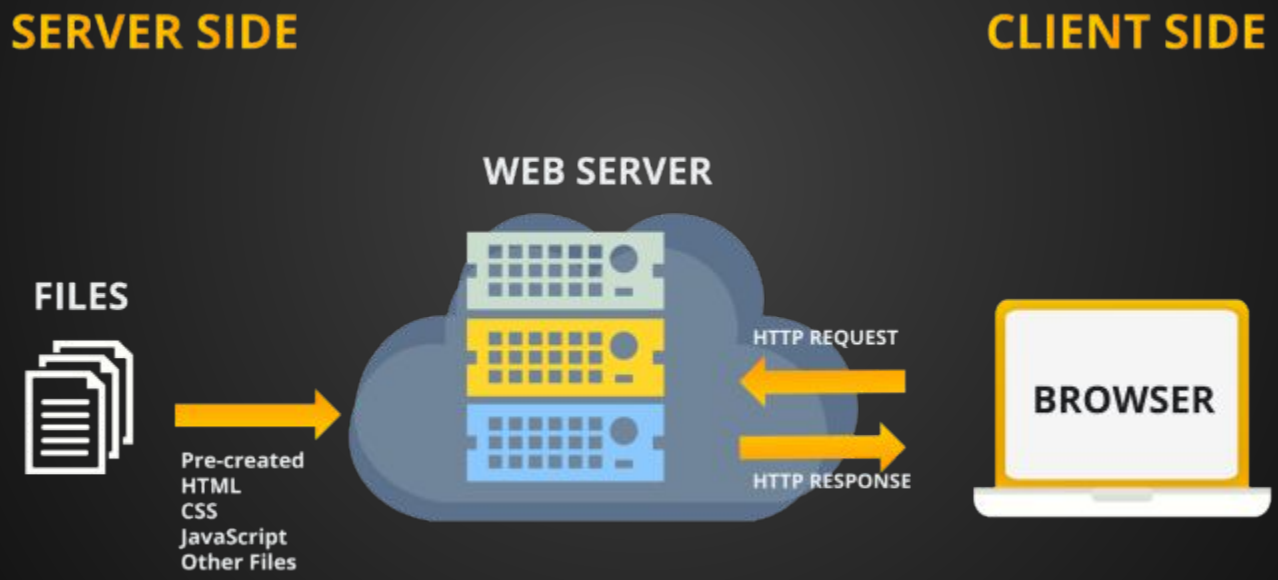
TABLE OF CONTENTS

BO ALGIERS OSIDES

SHELLMATES

Client-side Vs. Server-side

Client-side Vs. Server-side



Client-side vulnerabilities

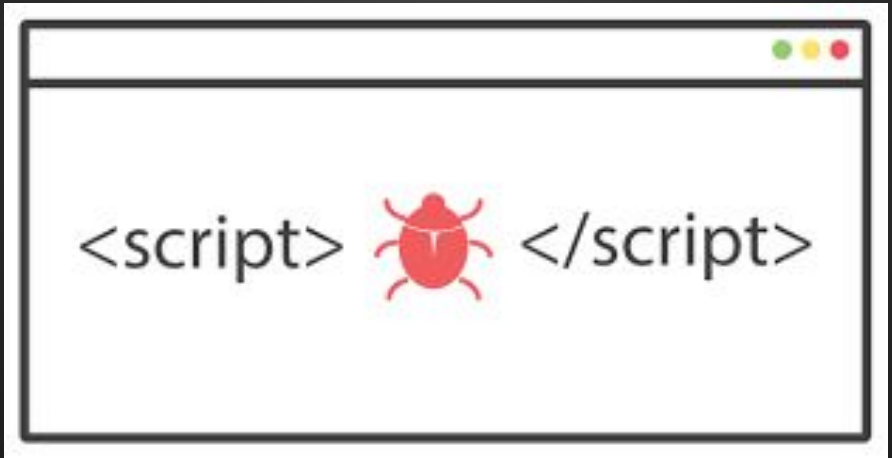
Security incidents and breaches

Client-side vulnerabilities

Cross Site Scripting

I. Cross Site Scripting (XSS)

A type of client side code injection attack. The attacker intends to run harmful scripts in the victim's web browser by embedding malicious code in a web page.



What you can do with XSS

01

Hijack a user's session

02

Perform unauthorized activities

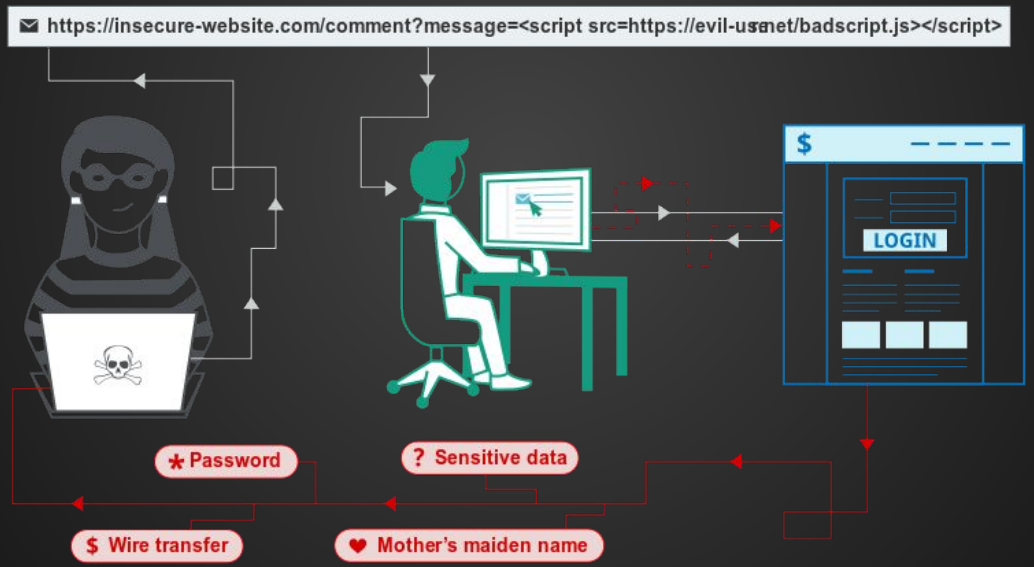
03

Perform phishing attacks

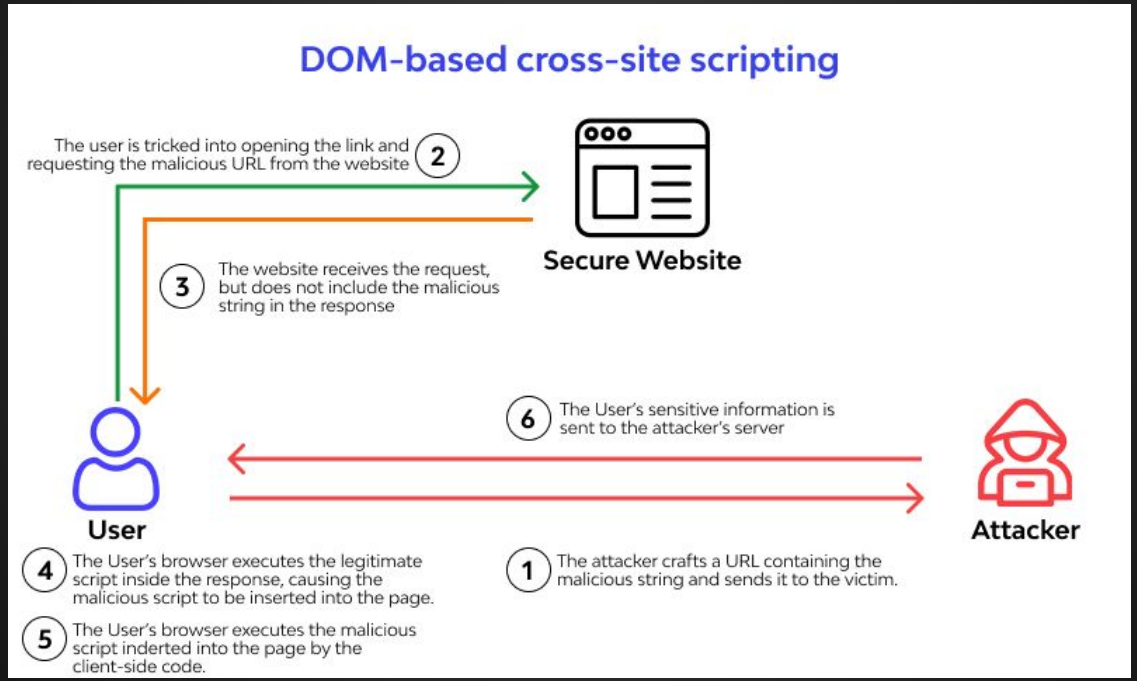
04

Steal sensitive information

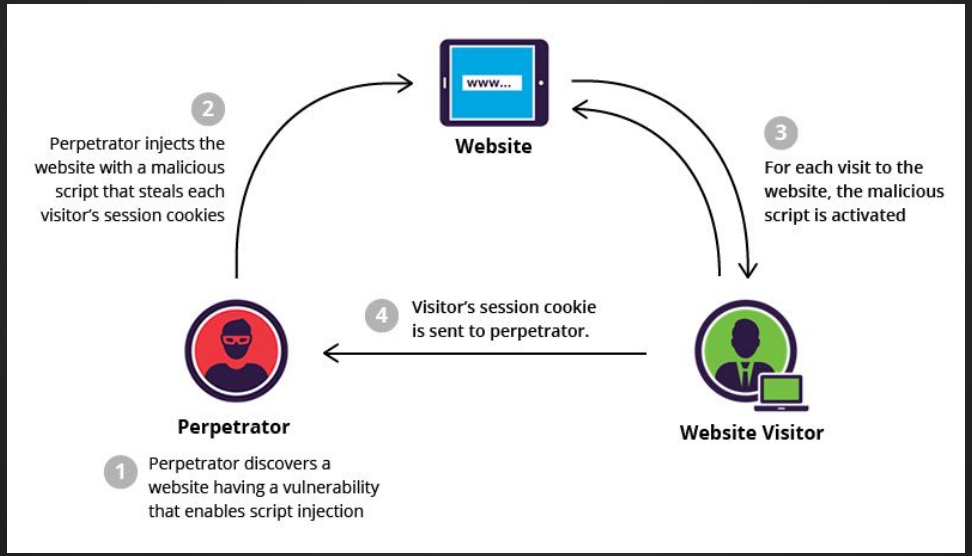
I. Cross Site Scripting - Reflected



I. Cross Site Scripting - DOM



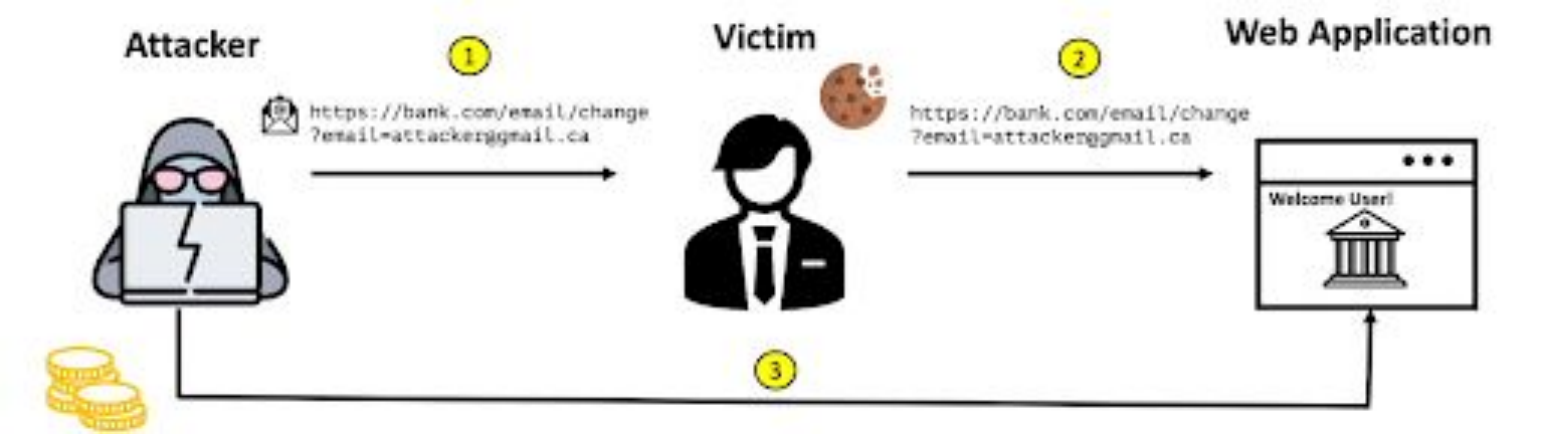
I. Cross Site Scripting - Stored



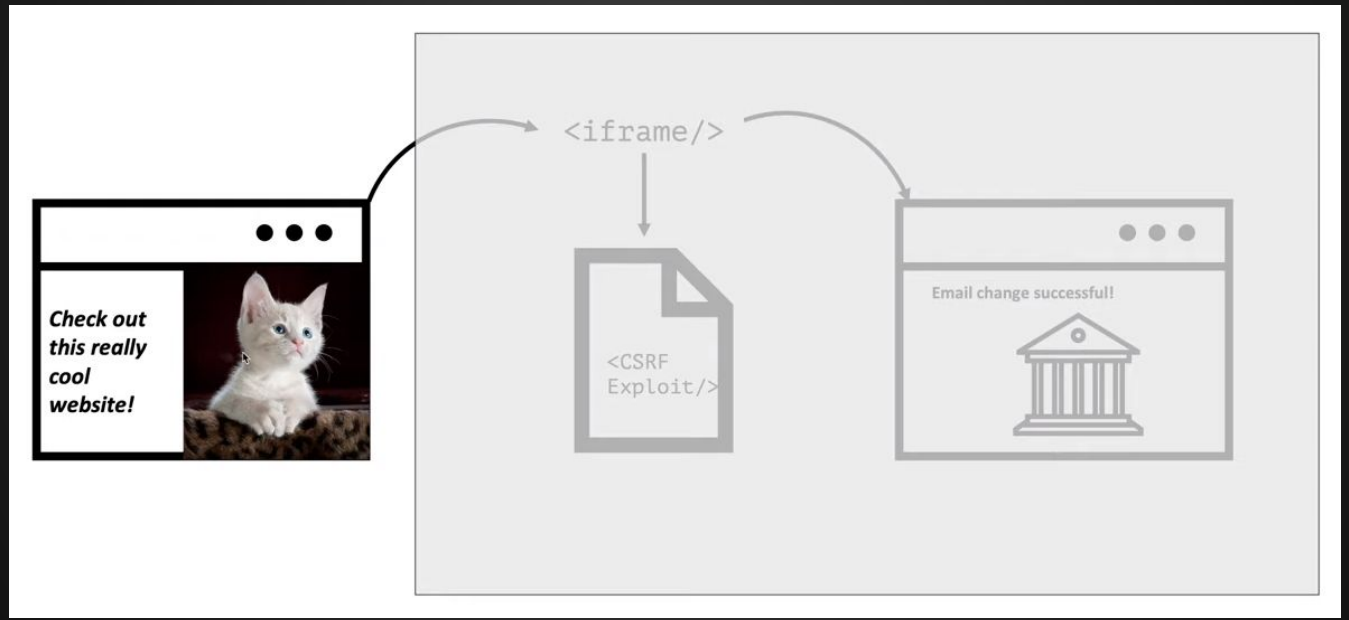
Cross Site Request Forgery

II. Cross Site Request Forgery

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



II. Cross Site Request Forgery



What you can do with CSRF

01

Change email

02

Transfer amount

03

Delete records (emails, blogs, etc)

04

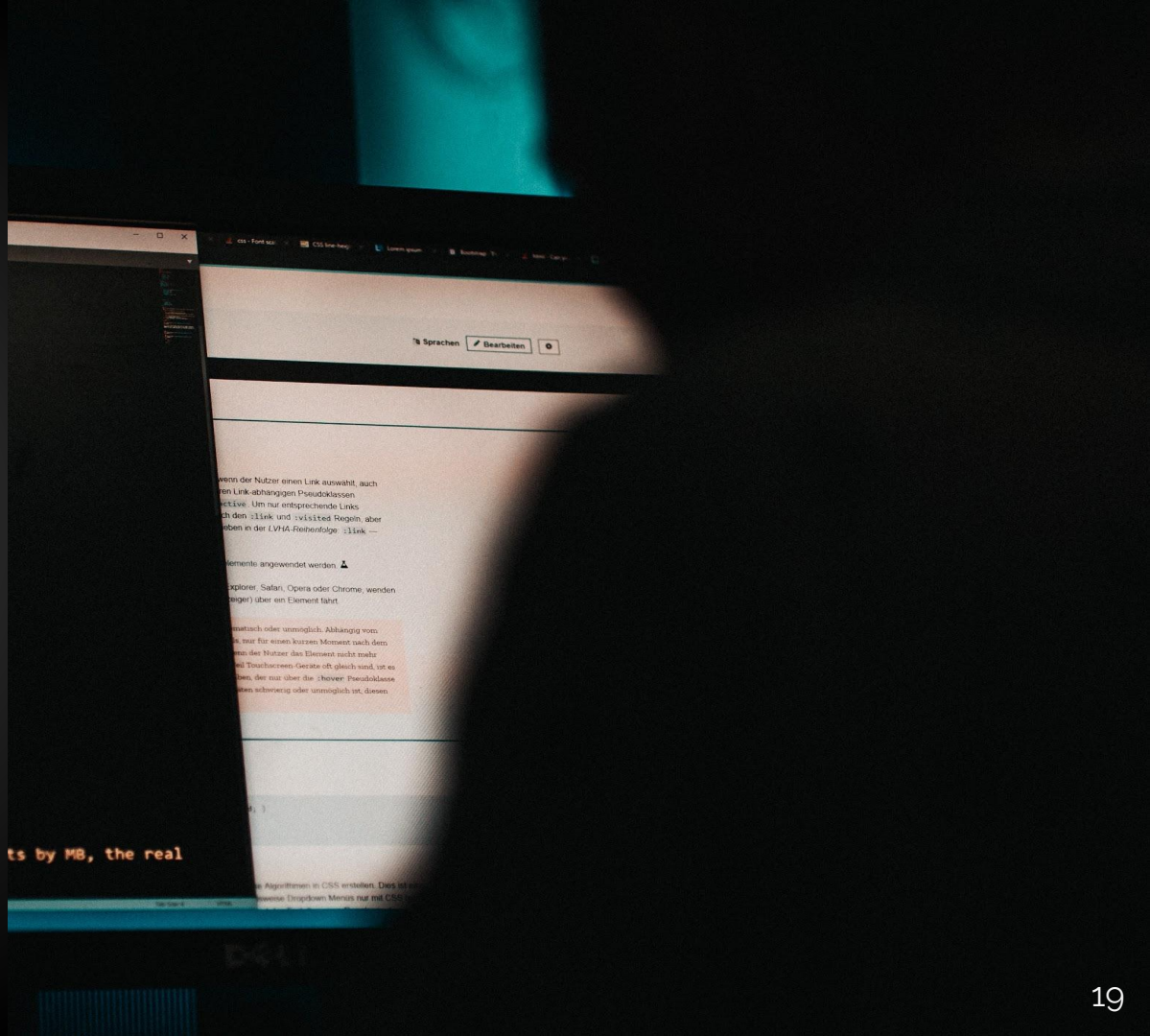
Perform high privilege actions

II. Cross Site Request Forgery

CSRF conditions:

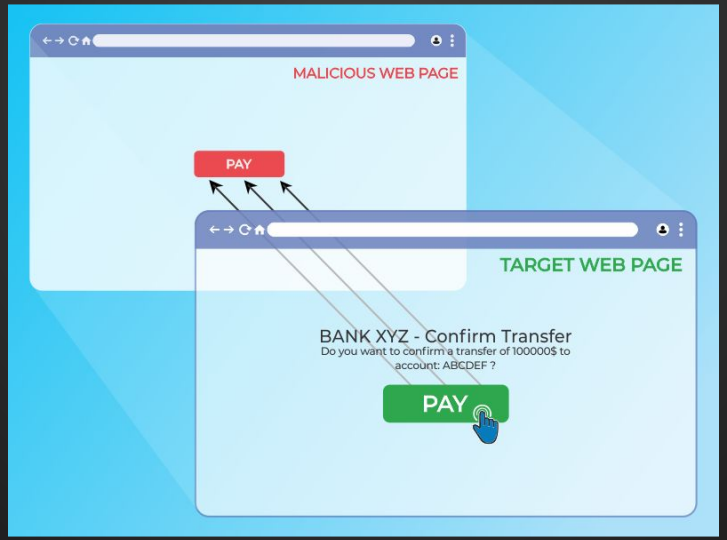
- A relevant action.
- Cookie-based session handling.
- No unpredictable request parameters.

Clickjacking



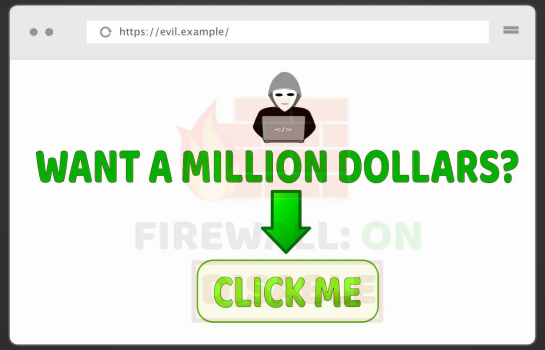
III. Clickjacking

Clickjacking is a type of attack in which a user is tricked into clicking on a webpage element that is either invisible or disguised as another element.

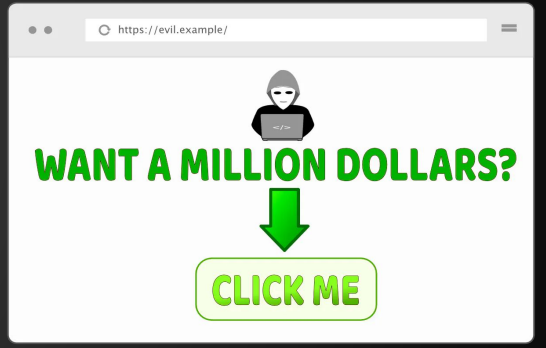


III. Clickjacking

01



02



03

III. Clickjacking - Types

Likejacking

Cookiejacking

Filejacking

Cursorjacking

Password manager attacks

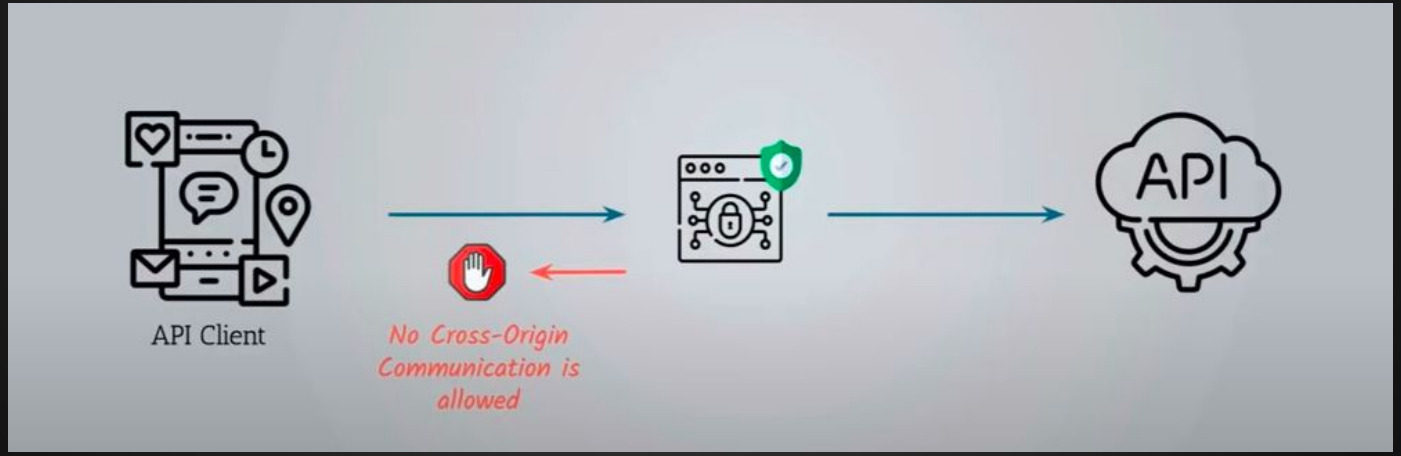
SOP / CORS

I. Same Origin Policy

Same Origin Policy is a security mechanism that limits how a document or script loaded from one origin interacts with a resource from another.

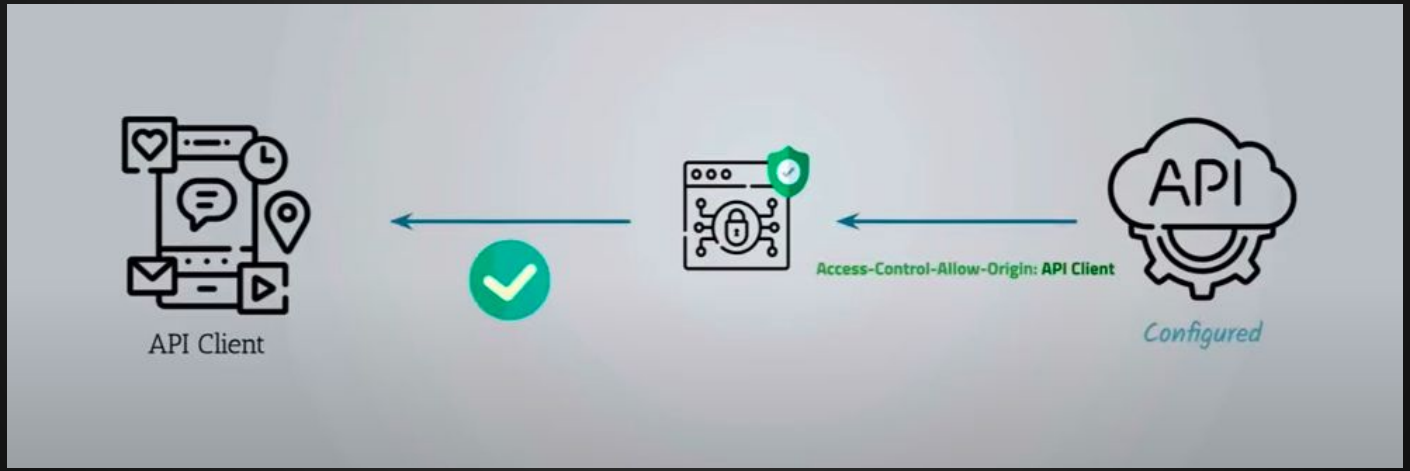
II. Cross-Origin Resource Sharing (CORS)

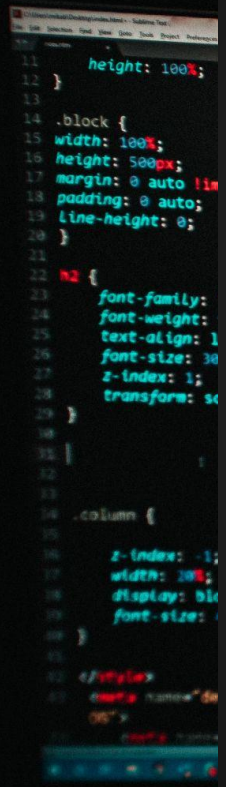
Cross Origin Resource sharing is another browser mechanism which enables controlled access to resources located outside of a given domain.



II. Cross-Origin Resource Sharing (CORS)

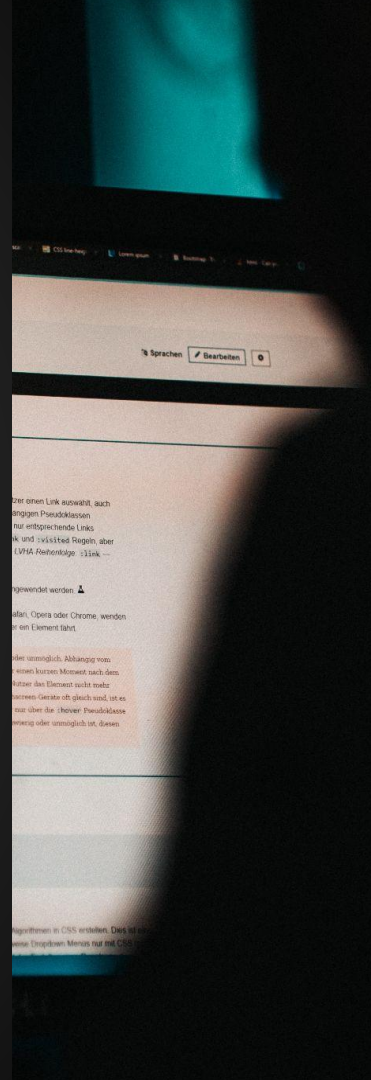
Access-Control-Allow-Origin header permits us to specify the origins that can fetch our API.





Mitigations

How to protect?



Mitigations

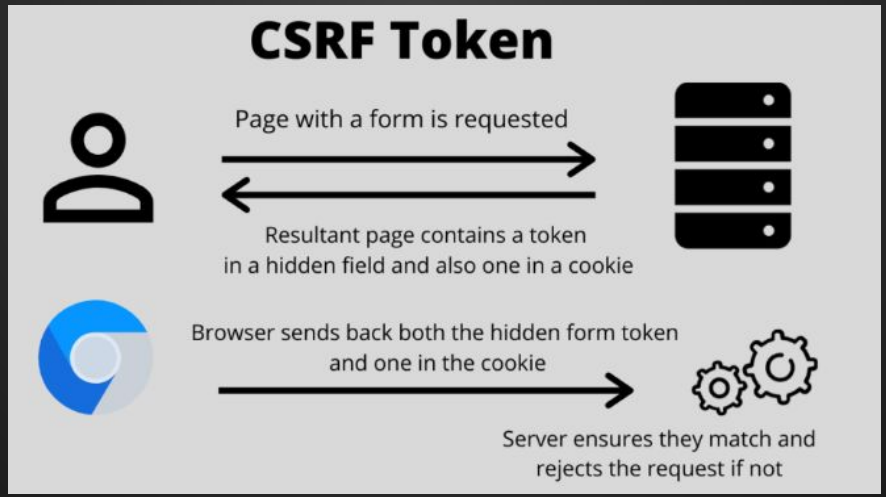
I. XSS

- **User Input Sanitization**
  - ```
    userInput.replace(/</g, "<").replace(/>/g, ">")
             .replace(/'/g, "2#39;").replace(/"/g, """)
             .replace(/&/g, "&");
    ```
  - Use DOMPurify
- **HttpOnly cookies**
  - ```
    resp.set_cookie("name", value = "value", httponly = True)
    ```
- **Using CSP**
  - ```
    Content-Security-Policy: script-src 'self'
    ```
    ```

II. CSRF

CSRF-TOKEN

- Unpredictable with high entropy, as for session tokens in general.
- Tied to the user's session.
- Strictly validated in every case before the relevant action is executed.



III. Clickjacking

- X-Frame-Options
  - `X-Frame-Options: DENY`
  - `X-Frame-Options: SAMEORIGIN`
- Using cookie's sameSite origin
  - `resp.set_cookie("name", value = "value", samesite="Strict")`
- Using CSP
  - `Content-Security-Policy: script-src-elem <source>`

Conclusion

IF you find a vulnerability, you go ahead and fix it. You don't base your security on:

- Luck
- Difficulty of exploitation
- Requiring intimate knowledge of your code which you assume no one has.

Thank you for your attention

Any questions?

 jm_abdelmalek@esi.dz

 /m0kr4n3/

 /m0kr4n3

 jo_merouan@esi.dz

 /ouxs11/

 /ouxs11