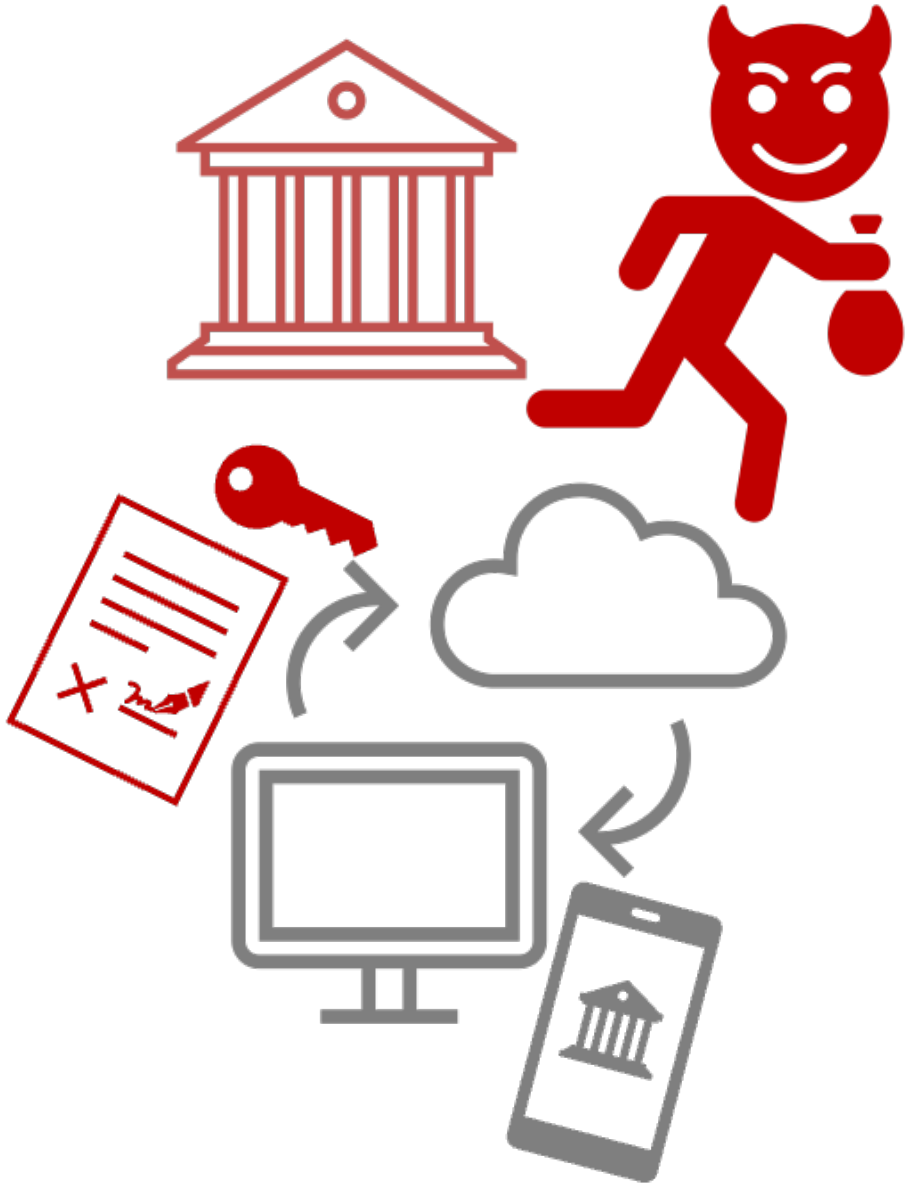
<WA/>
2024

Auth Security Issues

Do's and Don'ts

Enrico Masala
Antonio Servetti



Some Security Risks in Auth Operations

- Leaked secrets
  - Session IDs
  - Tokens
- Unauthorized operations
  - Performed by an unknowing (deceived or careless) user
  - E.g., Cross-Site Request Forgery (CSRF) attacks

- How to avoid or mitigate such risks?

How to Store Secret Info in the Browser

- Secret info (i.e., session IDs and tokens) are the way HTTP requests are authenticated/authorized on the server side
- Limit access to such secret info as much as possible
  - Use browser cookie storage
  - httpOnly flag (**inaccessible from Javascript**, prevents stealing via XSS attacks)
  - secure flag (send it only on **encrypted communication channels**)
- Sometimes it is not possible
  - When the info is to be sent via Javascript methods (e.g., fetch with HTTP headers including the secret)

HTML5 Web Storage API

- HTML5 introduced offline local storage in browsers to allow some applications work even without network connection
  - Example: offline Google docs
- Two storages: SessionStorage and LocalStorage
- Maintain a separate storage area for each origin
  - LocalStorage persists even when the browser is closed and reopened
  - SessionStorage lasts for the duration of a page session (i.e., browser tab open, including reloads)
- Such storage allows to store data in the form of key-value pairs
- You might be tempted to use it to store secret info: never do that!

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

LocalStorage and SessionStorage

- Available in Javascript via “window” object
- Use it only when access to the data is not assuming authentication or authorization
- A single Cross Site Scripting vulnerability can be used to steal all the data and/or to load malicious data there: do not consider such data as trusted!
- In short: never use them to store sensitive/secret information
  - Such as session ID, tokens, etc.
  - Use Cookie storage with *httpOnly* option whenever possible

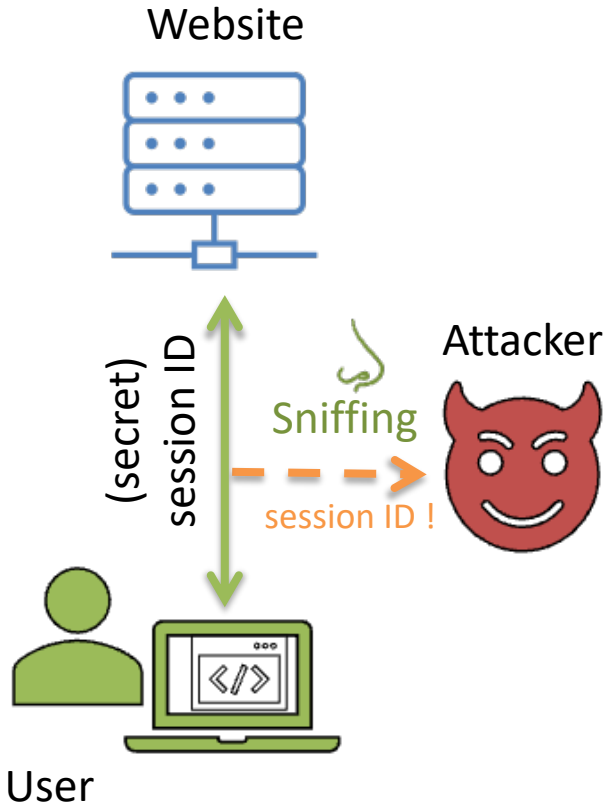
https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html

https://portswigger.net/web-security/dom-based/html5-storage-manipulation

Cookies vs Alternative Approach

	Cookies	Other approaches (e.g., tokens)
Pros	• Automatically handled by the browser • Can be made inaccessible to JS code (httpOnly option) to prevent access from malicious JS code (XSS) • Can be made to be sent on secure connections only (secure option)	• More flexibility (not restricted to cookie APIs) • Sent only when required, by JS
Cons	• Cannot be sent by third parties • **Always sent** for any request: expose to CSRF attack	• Need to explicitly manage storage (e.g., React state) • Can be **accessed** by any (malicious) script in the page (**XSS**)

How to Transmit Secret Info Securely?



- Use only encrypted channels (HTTPS)
- Provides
  - **Confidentiality**: the attacker cannot sniff/eavesdrop information
  - **Integrity**: attacker cannot modify traffic, i.e., reply requests towards the server
  - **Authentication**: client is connected to a legitimate server (via server certificate)
    - *Important, but out of the scope of this course*

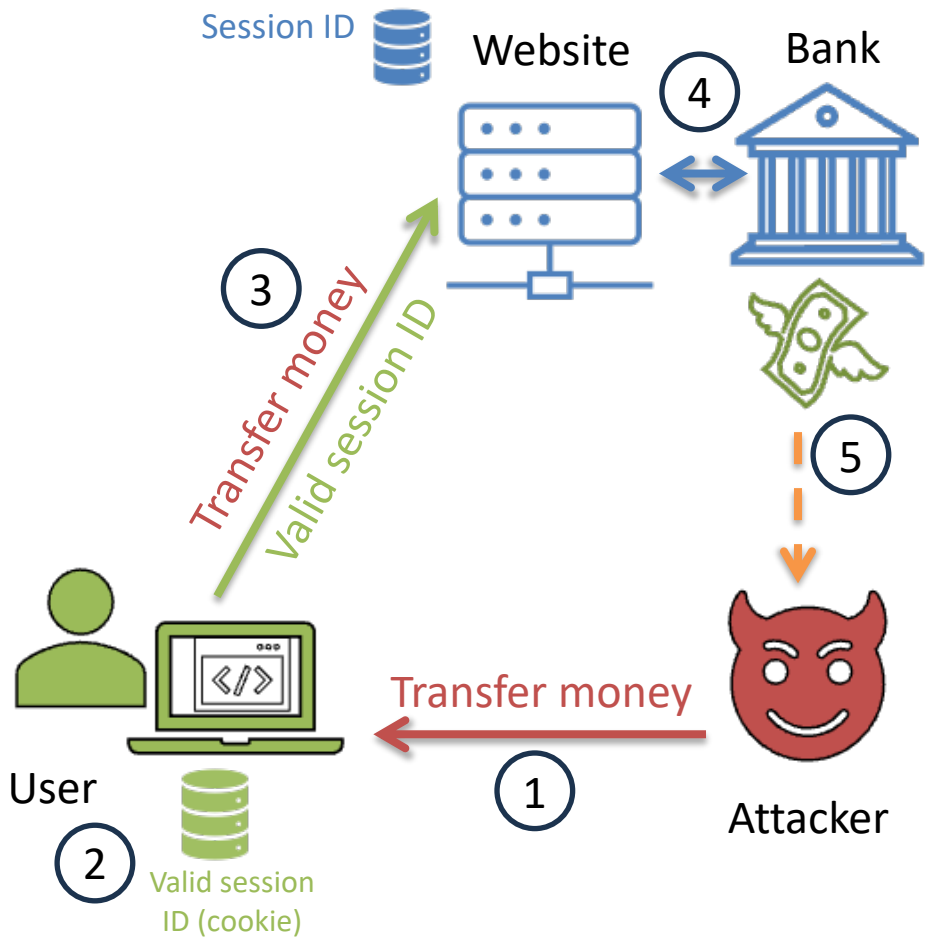
https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Security_Cheat_Sheet.html

How to Configure HTTPS Correctly?

- Check OWASP for best configuration: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Security_Cheat_Sheet.html

NB: **Only for this course, only for configuration simplicity and easier debugging at the exam**, HTTPS will NOT be used

Cross Site Request Forgery (CSRF) attack



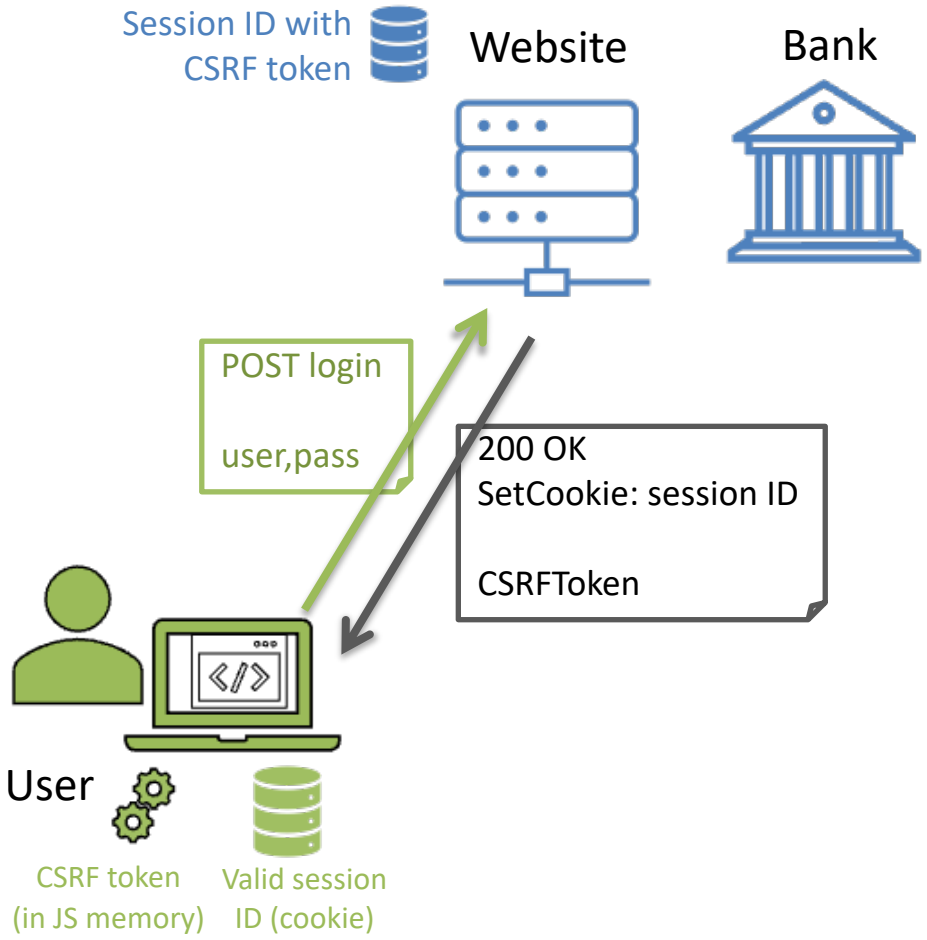
1. The attacker forges a malicious request link and sends it to a user (e.g., email)
2. The user may be already logged into the website
3. The user is deceived into sending the request to the website (e.g., click on a link)
4. The website validates the request since it comes with a valid session ID, and perform the operation
5. The attacker gains from the execution of the operation (e.g., money transfer)

CSRF Conditions for Success and Mitigation

- The user must have an authenticated session active (logged in with the bank website)
- The execution of a sensitive action relies only on the presence of a simple authentication factor (i.e., the presence of a session cookie)
- There is not a parameter in the request that the attacker cannot add because it is unknown

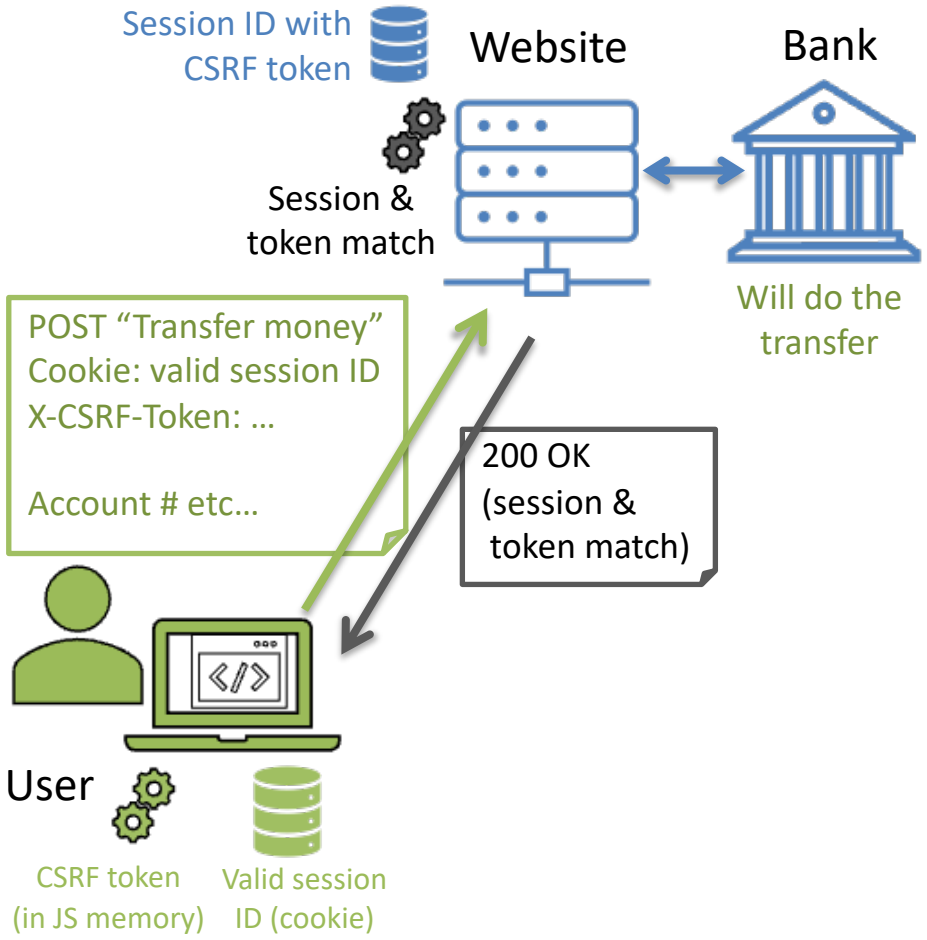
- Possible **mitigation strategy**: use an additional token, to be sent in session authenticated HTTP requests
  - Named “CSRF token” or “anti-forgery token” or “request verification token”

CSRF Token Creation



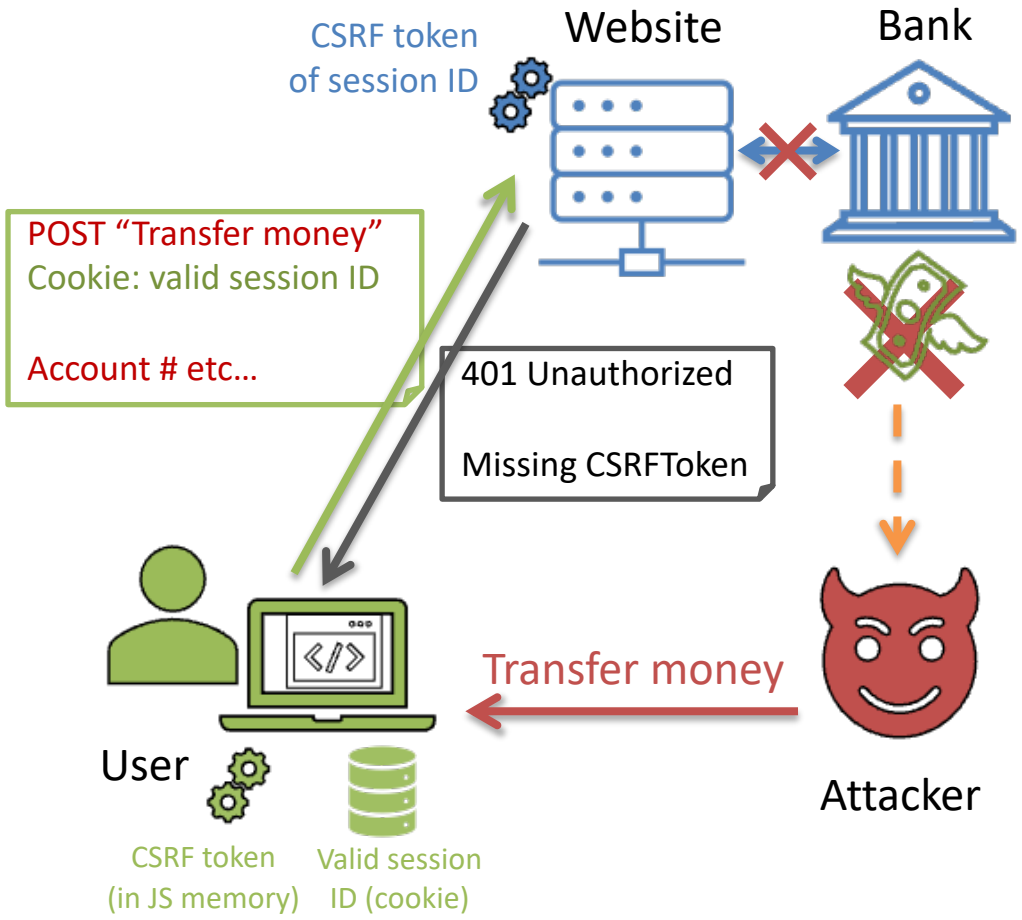
- When the session authentication is performed, the server also generates a random value (the CSRF token), sent to and stored by the client
  - The server keeps the value and associate it to session

CSRF Token Use



- Every time a session authenticated HTTP request is sent, the client adds the token via JS, typically as an additional HTTP header (`X-CSRF-Token:…`)
  - Do not store token as cookie, otherwise it would be always sent automatically by the browser
- The server checks the token presence and value before proceeding

CSRF Attack Prevention



- The attacker has no way of knowing the token value to use when creating the forged “link”
- If the token is missing or does not match, the server refuses the operation

CSRF Advantages and Limitations

- Since the CSRF token value is stored only in the client memory (e.g., in a Javascript variable) it will not be automatically added by the browser
  - The forged link/request will be refused
- Remember, this is just a mitigation strategy!
- Any successful XSS technique can defeat all CSRF mitigation techniques
  - Via Javascript, it could have access to the token and add it
- Recently, client-side CSRF attacks have also been developed
  - Trick the client-side JS code to send requests by manipulating input parameters
- How to treat these cases?

Multi-factor authentication

- Since it is so difficult to protect web applications (XSS, CSRF, etc.), a multi-factor authentication (MFA) approach could be used
  - Often implemented as a Two-Factor Authentication (2FA)
- The user is required to present more than one type of evidence to authenticate on a system
- See OWASP:

Factor	Examples
Something you know	Password and PINs, Security questions
Something you have	OTP Tokens, U2F Tokens, Certificates, Smart Cards, Email, SMS and Phone Calls
Something you are	Fingerprints, Facial recognition, Iris scan
Somewhere you are	Source IP Address, Geolocation, Geofencing
Something you do	Behavioral Profiling, Keystroke & Mouse Dynamics, Gait Analysis

https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html

Two-factor authentication

- The two factors must belong to different categories
  - E.g., not: password + PIN
- "*MFA is by far the best defense against the majority of password-related attacks*" [OWASP]
- Typically required at log in, but it may be also appropriate to require 2FA to perform other sensitive actions such as
  - Changing passwords, contact email, disabling 2FA, granting additional privileges, …
  - Performing sensitive operations (confirm money transfers, risky operations, …)

https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html

Example 2FA: Something You Have

- One Time Password Tokens (OTP)
  - Hardware and software based
- Hardware tokens
  - Almost impossible to compromise remotely, but handling them may be impractical (distribution costs, risk of losing them), may require special server hw
- Software OTP Tokens
  - Generally used to generate Time-based One Time Password (TOTP) codes
  - Typically via an App on the user's smartphone (e.g. Google authenticator)
  - Some are standardized (e.g., RFC 6238)

https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html

TOTP

- The server requires a TOTP when a sensitive action is required
- Pros
  - Cheap, no logistic issues, easily changed if user loses one
- Cons
  - Smartphones can be compromised, TOTP app might be on the same device used to authenticate, require the user to have a smartphone
- Note: a common secret should be established in order to verify TOTP
  - Usually done during the TOTP activation phase: the server chooses and sends the secret to the web client, e.g., as a QR-code, which is then memorized in the smartphone and never sent again by the server

https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html

Final Considerations

- Authentication and authorization are complex problems
- Much more than cookies + tokens + CSRF, for instance:
  - Third-party authentication: Google, Meta (ex Facebook),…
  - OAuth2
  - Different MFA forms
  - …
- Many possible attack approaches (XSS, CSRF, …)

Final Advice

- Never invent your own mechanism! Use standardized, well tested, ones!
- Recommended procedure
  - Identify an auth system which is well suited for your case
  - Study it in depth
  - Rely on advices and best practices from reputable sources, e.g., OWASP to implement it correctly
  - Maintain it over time: regularly consult advices from reputable security sources and make sure that newly discovered weaknesses are promptly addressed!

License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/