CORS

A **CORS (Cross-Origin Resource Sharing)** egy olyan biztonsági mechanizmus, amelyet a böngészők alkalmaznak annak érdekében, hogy ellenőrizzék és szabályozzák a weboldalak közötti adatcserét, különösen akkor, amikor egy weboldal egy másik domain, protokoll vagy port erőforrásaihoz próbál hozzáférni.

# Alapfogalmak

## Mi az az "origin"?

* Egy weboldal **origin**-je az alábbi három részből áll:
  1. **Domain** (például: example.com)
  2. **Protokoll** (például: http:// vagy https://)
  3. **Port** (alapértelmezés szerint a 80-as port az HTTP-nél, és a 443-as port az HTTPS-nél)

Például a https://example.com:3000 URL originje:

* Protokoll: https
* Domain: example.com
* Port: 3000

## Mi az a "cross-origin"?

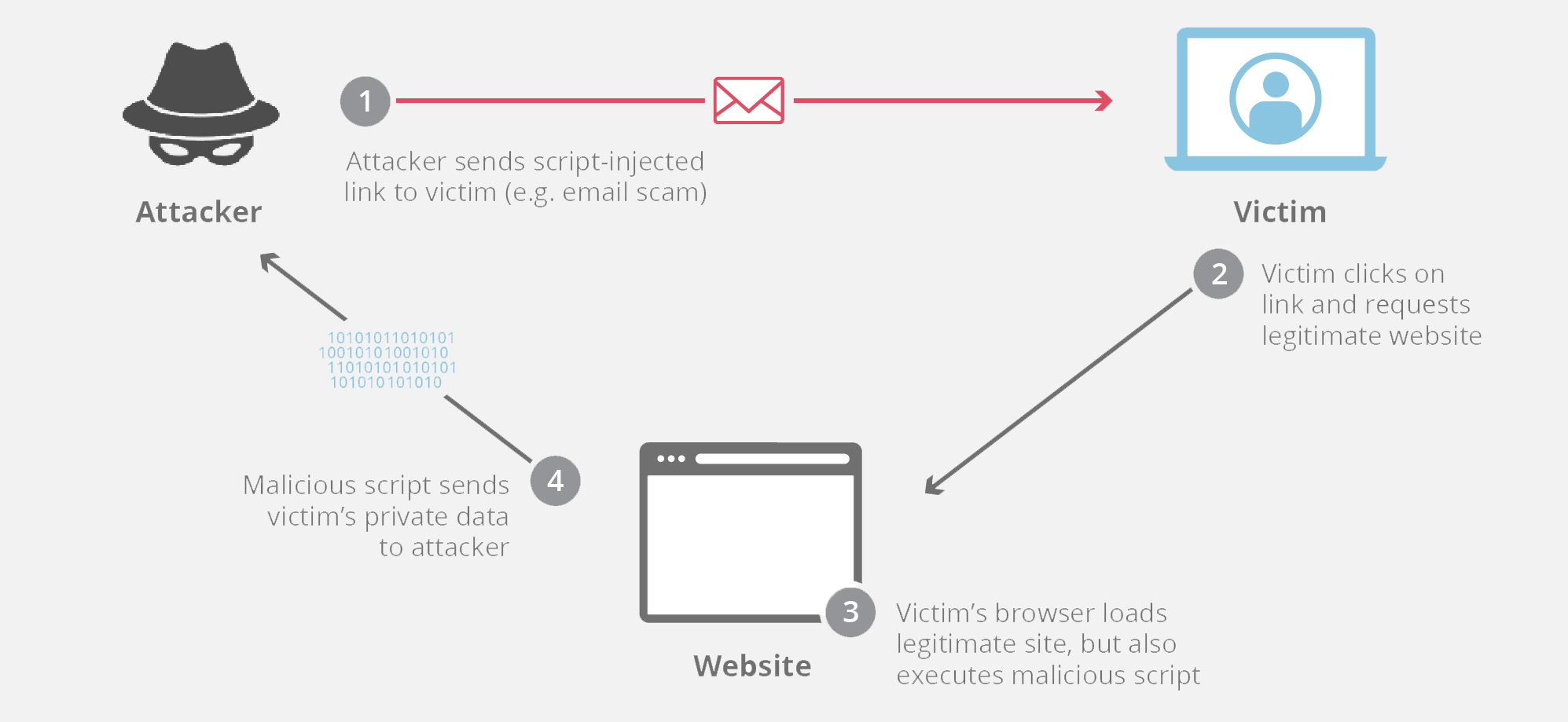
Egy "cross-origin" kérésről akkor beszélünk, ha egy weboldal egy másik originről (*domain-ről, portból vagy protokollból*) próbál betölteni adatokat. Például, ha a http://example.com weboldalról egy API-kérést küldünk a http://api.example.com címre, az már egy "cross-origin" kérés.

# CORS szükségessége

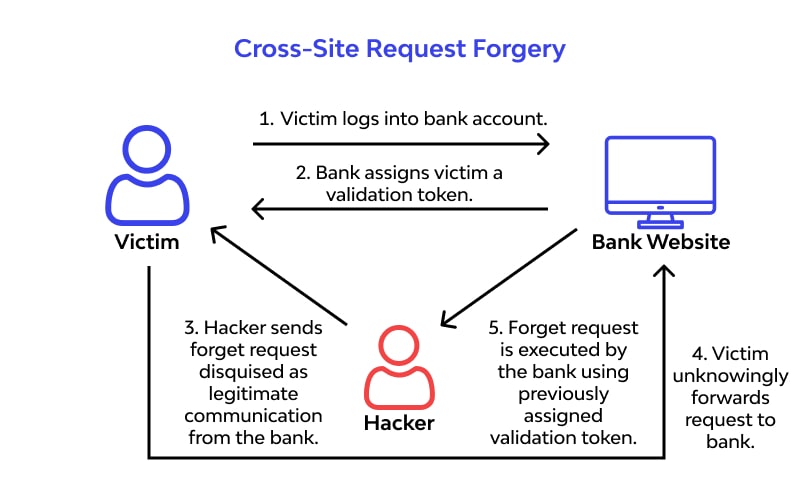
A böngészők alapvetően korlátozzák a "cross-origin" kéréseket, hogy megakadályozzák a **Cross-Site Scripting (XSS)** vagy **Cross-Site Request Forgery (CSRF)** típusú támadásokat. A CORS mechanizmus lehetővé teszi a szerverek számára, hogy megadják, melyik originről érkezhetnek biztonságosan kérések.

Mivel a fejlesztés során a szerver (*http://localhost:3000*) és kliens (*localhost:80* vagy *localhost:5500*) ugyanazon a gépen futnak, de különböző portokon, ezért a CORS problémája felmerülhet. Itt a "cross-origin" helyzet abból adódik, hogy a port számok eltérnek (3000 vagy 80 vagy 5500). A böngésző ilyenkor védi az adatokat azáltal, hogy megköveteli, hogy a szerver kifejezetten engedélyezze ezeket a kéréseket.

## Cross-Site Scripting



## Cross-Site Request Forgery



# Hogyan működik a CORS?

Amikor egy böngésző egy cross-origin kérést próbál küldeni, a CORS mechanizmus a következő lépéseket hajtja végre:

## Egyszerű kérés (*Simple Request*)

Egy kérés "egyszerű" (simple), ha az alábbi kritériumok mindegyike teljesül:

* **HTTP-módszerek** közül csak a GET, POST vagy HEAD van használatban.
* A kérésben csak alapvető HTTP-fejlécek vannak, mint például:
  + Accept
  + Content-Type (korlátozva text/plain, multipart/form-data, vagy application/x-www-form-urlencoded típusokra)
  + DPR, Width, Viewport-Width

Egy ilyen egyszerű kérés esetén a böngésző egyszerűen elküldi a kérést, és ha a szerver CORS fejlécekkel válaszol, akkor a böngésző ellenőrzi, hogy a válasz engedélyezett-e.

A szerver válaszában a következő fejléc szerepelhet, ami azt mondja a böngészőnek, hogy engedélyezett a kérés:

Access-Control-Allow-Origin: https://example.com

## Előzetes ellenőrzés (*Preflight Requests*)

Ha a kérés nem "egyszerű", a böngésző egy előzetes ellenőrzést (preflight) hajt végre egy **OPTIONS** kérés küldésével, mielőtt a tényleges kérést elküldené. Ez az ellenőrzés megkérdezi a szervert, hogy az engedélyezi-e az adott típusú kérést.

Az előzetes ellenőrzés kérdésre a szerver válasza tartalmazza a következő fejléceket:

* **Access-Control-Allow-Methods**: Mely HTTP-módszereket engedélyezi a szerver (pl.: GET, POST, PUT stb.).
* **Access-Control-Allow-Headers**: Mely egyéni HTTP-fejléceket engedélyezi a szerver (pl.: Content-Type, Authorization).
* **Access-Control-Allow-Origin**: Az origin, ahonnan a kérés érkezhet (vagy \* minden origin engedélyezésére).

Ha a szerver válasza megfelelő, a böngésző végrehajtja a tényleges kérést. Ha nem, a kérés elutasításra kerül.

Példa egy preflight kérésre:

OPTIONS /some/resource HTTP/1.1

Host: api.example.com

Origin: http://localhost:3000

Access-Control-Request-Method: POST

Access-Control-Request-Headers: Content-Type, apikey

És a szerver válasza:

HTTP/1.1 200 OK

Access-Control-Allow-Origin: http://localhost:3000

Access-Control-Allow-Methods: GET, POST, OPTIONS

Access-Control-Allow-Headers: Content-Type, apikey

## A tényleges kérés (Actual Request)

Ha az előzetes ellenőrzés sikeres, a böngésző elküldi a tényleges kérést az adatok lekéréséhez vagy módosításához.

# Gyakori CORS fejlécek

* **Access-Control-Allow-Origin**: Meghatározza, hogy mely originről érkező kérések engedélyezettek. Ha minden origin számára engedélyezett, az értéke lehet \*, de biztonsági okokból ez nem mindig ajánlott.
* **Access-Control-Allow-Methods**: A szerver által engedélyezett HTTP metódusok (pl. GET, POST, PUT, DELETE).
* **Access-Control-Allow-Headers**: Mely egyéni HTTP-fejléceket engedélyez a szerver.
* **Access-Control-Expose-Headers**: Fejlécek, amelyeket a kliens olvashat a válaszból (alapértelmezetten nem minden fejléc érhető el a böngésző számára).
* **Access-Control-Allow-Credentials**: Ha a szerver megköveteli, hogy a kérések tartalmazzanak hitelesítési adatokat (pl. sütiket), ez a fejlécnek true értéket kell tartalmaznia.

# CORS gyakori problémák

* **Nincs megfelelő CORS fejléc**: Ha a szerver nem válaszol a megfelelő CORS fejléc nélkül, a böngésző automatikusan blokkolja a kérést.
* **Előzetes ellenőrzés hibája**: Ha a böngésző egy preflight kérést küld, de a szerver nem adja meg a megfelelő válaszfejléceket (*például hiányzik az Access-Control-Allow-Headers vagy Access-Control-Allow-Methods*), a kérés blokkolódik.
* **Wildcard \* használata Access-Control-Allow-Origin-ben**: Ha hitelesítési adatokkal (*pl. sütik*) dolgozol, nem használhatsz \* értéket az Access-Control-Allow-Origin fejlécben, mivel biztonsági okokból csak konkrét origin lehet megadva.

# Hogyan oldhatod meg a CORS problémákat?

Több megoldás is lehetséges a böngészők beépített védelmének kikapcsolására az alkalmazásunk tesztelésénél

## Szerver oldali változtatások

A CORS szabályok beállítása a szerveren történik. Győződj meg arról, hogy a szerver válaszai tartalmazzák a megfelelő CORS fejléceket.

### CORS Middleware használata

const cors = require('cors');

app.use(cors({

origin: ['http://localhost:80', 'http://localhost:5500'], // Allowed origins

methods: ['GET', 'POST', 'OPTIONS'], // Allowed methods

allowedHeaders: ['Content-Type'] // Allowed headers

}));

### Kézi fejléc beállítása

app.use((req, res, next) => {

res.header('Access-Control-Allow-Origin', 'http://localhost:80');

res.header('Access-Control-Allow-Methods', 'GET, POST');

res.header('Access-Control-Allow-Headers', 'Content-Type');

next();

});

## CORS proxy használata

Ingyenes CORS proxy szerverek:

* CorsProxy.io (<https://corsproxy.io/>)
* CORS.SH (<https://proxy.cors.sh/>)
* HTMLDriven (<https://cors-proxy.htmldriven.com/>)
* …

## Böngésző bővítmények

Kikapcsolják a CORS ellenőrzést, de ez csak fejlesztés alatt ajánlott.