Engineer

CORS

дозвіл сервера іншим походженням читати його відповіді

Що це таке

CORS, повністю Cross-Origin Resource Sharing, — це правила, за якими сервер повідомляє браузеру, що сторінкам інших сайтів дозволено читати його відповіді. Браузер тримає різні походження окремо одне від одного. Походження — це схема, хост і порт разом, тож `https://app.example.com`, `https://api.example.com` і `http://app.example.com` — три різні походження, і скрипт на одному з них не може прочитати відповідь іншого, доки той не погодиться. CORS і є способом погодитися: невелика родина HTTP-заголовків, майже всі з яких починаються з `Access-Control-`. Сервер додає їх до відповідей, а браузер перевіряє, перш ніж віддати хоч щось скрипту, що зробив запит.

Протокол описано у Fetch Standard від WHATWG, живій специфікації того, як браузери завантажують ресурси. Там же прямо сказано, навіщо він потрібен. Сторінка має змогу користуватися даними з іншого походження й надсилати запити, багатші за ті, що вміє HTML-форма, але лише тоді, коли інша сторона сама на це погодилася. Без такої згоди будь-яка сторінка з інтернету могла б читати сервери, що стоять за корпоративним файрволом, або приватні дані користувача на іншому сайті, ще й з його cookies.

Сам обмін короткий. Коли скрипт просить ресурс з іншого походження, браузер додає заголовок `Origin` із походженням сторінки. Сервер відповідає заголовком `Access-Control-Allow-Origin`, значення якого — або те саме походження, записане точно, або `*` для всіх. Якщо значення збігається, скрипт отримує відповідь. Якщо ні, скрипт отримує мережеву помилку і жодного байта тіла.

Деякі запити потребують дозволу ще до того, як їх надіслано. `GET`, `HEAD` або `POST` із кількома звичайними заголовками й тілом типу `application/x-www-form-urlencoded`, `multipart/form-data` чи `text/plain` вирушає одразу, адже таке HTML-форма завжди могла надіслати з будь-якого сайту. Усе, що понад це, як-от `PUT`, `DELETE`, тіло в JSON або заголовок `Authorization`, спершу проходить попередній запит, який усі називають preflight: це запит `OPTIONS`, що називає метод у `Access-Control-Request-Method`, а додаткові заголовки — в `Access-Control-Request-Headers`. Справжній запит іде лише тоді, коли відповідь має успішний статус, наприклад 200 чи 204, і дозволяє і те, і те.

Застосуйте поняття до рішення

Застосуйте «Що таке CORS?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Облікові записи, паролі й права налаштовані так, щоб доступ відповідав роботі, а звільнені втрачали його того ж дня.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Написала 1 000 правил RBAC для Linux-серверів — 1,000 правил доступу написано, щоб усунути вразливість

Написала 1 000 правил RBAC для Linux-серверів Зв'язатися з нами →

Чому це важливо

Майже кожен вебзастосунок з окремим API стикається з CORS першого ж дня. Фронтенд віддається з одного походження, API — з іншого, і перший `fetch` між ними падає в консолі браузера, доки API не почне надсилати правильні заголовки. Команди, які знайомляться з CORS саме так, запам’ятовують його як перешкоду й налаштовують відповідно: дозволяють будь-що, аби помилка зникла. Саме тут і ховається ризик.

Ці заголовки вирішують, хто може читати дані ваших користувачів через їхні ж браузери. Користувач, що увійшов у ваш застосунок, має сесійну cookie. Якщо API дозволяє будь-якому походженню надсилати запити з cookies і читати відповіді, то кожна сторінка, яку відвідає цей користувач, може викликати API від його імені й прочитати результат. Fetch Standard каже про це без евфемізмів: ділитися відповідями, отриманими з обліковими даними користувача, треба з надзвичайною обережністю, щоб не створити проблему заплутаного заступника (confused deputy). Браузер, який несе cookies користувача, і є тим заступником, а недбале налаштування CORS дає чужій сторінці ним керувати.

Так само важливо розуміти, чого CORS не робить. Він не захищає сервер. Простий запит доходить до сервера й виконується незалежно від того, чи можна потім поділитися відповіддю, тож `POST`, який змінює дані, уже змінив їх, перш ніж браузер гляне на перший CORS-заголовок. Поза браузером заголовок `Origin` містить те, що впише клієнт, і посібник OWASP з тестування попереджає, що його можна підробити. Хто може викликати ендпойнт і що там робити, вирішують автентифікація та авторизація на сервері, щоразу. CORS вирішує лише, яким вебсторінкам дозволено прочитати відповідь усередині браузера.

Як це виглядає на практиці

Уявімо односторінковий застосунок на `https://app.example.com`, який зберігає нотатку через API на `https://api.example.com` і користується сесійною cookie, яку API встановив під час входу:

```
await fetch("https://api.example.com/v1/notes/42", {
  method: "PUT",
  credentials: "include",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ text: "Реліз у п'ятницю" }),
});
```

`PUT` із тілом у JSON — це більше, ніж може надіслати форма, тож браузер спершу питає. Попередній запит не несе cookies, а лише називає те, чим користуватиметься справжній:

```
OPTIONS /v1/notes/42 HTTP/1.1
Host: api.example.com
Origin: https://app.example.com
Accept: */*
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: content-type
```

API звіряє походження зі своїм списком і відповідає:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: https://app.example.com
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: PUT
```

```
Access-Control-Allow-Headers: content-type
Access-Control-Max-Age: 600
Vary: Origin
```

Тепер браузер надсилає сам `PUT` з cookie та заголовком `Origin`, а відповідь API знову містить `Access-Control-Allow-Origin` і `Access-Control-Allow-Credentials: true`. Якщо бракує хоча б одного з них, нотатку на сервері збережено, але скрипт про це так і не дізнається. `Access-Control-Max-Age` дозволяє браузеру повторно використовувати відповідь на попередній запит до 600 секунд замість питати перед кожним викликом; без цього заголовка стандарт тримає її п'ять секунд. `Vary: Origin` стоїть тут тому, що відповідь залежить від того, хто питає, і кеш, який цього не врахує, може віддати відповідь для одного походження іншому.

На платформі з професійними та організаційними даними на рівні API працюють обмеження частоти запитів, CORS, ліміти на розмір запиту й валідація вхідних даних ще до того, як щось потрапить у базу даних. Та сама сторінка пояснює, чому CORS — лише один пункт цього переліку, а не весь захист: платформа виходить з того, що клієнт ворожий, і сервер повторно перевіряє права доступу під час кожного запиту, хай що там дозволив браузер.

Де виникають проблеми

Сервер копіює будь-який `Origin` назад в `Access-Control-Allow-Origin`. Це найшвидший спосіб прибрати помилку з консолі, а разом з `Access-Control-Allow-Credentials: true` він відкриває кожному сайту в інтернеті те, що бачить в API користувач. Тримайте явний список походжень, яким потрібен доступ, порівнюйте вхідний `Origin` з ним точно й надсилайте заголовки лише за збігу.

Список перевіряють за префіксом або підрядком. Правило, яке приймає будь-яке походження, що починається з `https://app.example.com`, прийме й `https://app.example.com.attacker.example`, а перевірка підрядка ще гірша. Порівнюйте серіалізовані походження цілком. Серіалізоване походження не має ні шляху, ні скісної риски в кінці, тож `https://app.example.com/` у списку ніколи ні з чим не збіжиться.

Від `*` чекають роботи разом з cookies. Коли запит несе облікові дані, браузер відкидає `*` в `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers` та `Access-Control-Expose-Headers`, а `Access-Control-Allow-Credentials` має бути точно `true`, малими літерами. Публічний API лише для читання й без cookies може спокійно відповідати `*`; API, що спирається на сесію, мусить назвати походження.

`null` є в списку дозволених. Сторінки, завантажені з URL зі схемою `data:`, отримують непрозоре походження, а запит, який перенаправили між походженнями, теж повідомляє `Origin: null`, тож багато зовсім різних документів показують одне й те саме значення. Дозволити `null` означає дозволити їх усі. Не додавайте його.

Бракує `Vary: Origin`. Якщо API надсилає `Access-Control-Allow-Origin` лише тоді, коли в запиті є відповідний `Origin`, кеш може зберегти відповідь без цього заголовка й віддати її пізнішому CORS-запиту або зберегти відповідь для одного походження й віддати іншому. Fetch Standard вимагає `Vary: Origin` щоразу, коли відповідь залежить від походження. Якщо значення — `*` або одне незмінне походження, надсилайте його в кожній відповіді, а `Vary` не додавайте.

`Access-Control-Allow-Headers: *` вважають таким, що покриває все. Символ-замінник ніколи не покриває `Authorization`, його треба назвати явно, а для запитів з обліковими даними він не покриває взагалі нічого. Називайте ті заголовки, які клієнт справді надсилає.

CORS використовують як контроль доступу. Сервер, який відхиляє запити, бо `Origin` виглядає не так, а решту пропускає, написав перевірку, яку будь-який скрипт поза браузером просто обходить. Автентифікуйте клієнта, авторизуйте дію на сервері, а CORS залиште його єдину роботу: вирішувати, що браузер покаже сторінці.

Що почитати далі

- Fetch Standard: the CORS protocol — саме визначення: заголовки запиту й відповіді, попередній запит, таблиця комбінацій з обліковими даними й порада щодо `Vary`.
- RFC 6454: The Web Origin Concept — що таке походження, як його визначають схема, хост і порт і як воно записується в заголовок `Origin`.

- OWASP Web Security Testing Guide: Testing Cross Origin Resource Sharing — як тестувальники шукають походження-замінник `*` або походження, яке сервер повертає без жодної перевірки.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке CORS?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/cors/