1. *Чому клієнтський JavaScript?*
2. **Багата інтерактивність**  
   JavaScript може динамічно змінювати вміст сторінки без перезавантаження сторінки, реагувати на події, такі як кліки миші або натискання клавіш, та взаємодіяти з користувачем через форми та інші елементи веб-сторінки.
3. **Швидший час завантаження**

Коли ми переносимо завдання на браузер клієнта, він бере на себе частину обчислювального навантаження, що дозволяє серверу більш ефективно обробляти інші запити і підтримувати велику кількість користувачів одночасно.

1. **Покращена взаємодія з користувачем**

За допомогою JavaScript ми можемо створювати чудові елементи інтерфейсу користувача та адаптивні інтерфейси, які захоплюють користувачів.

1. *Огляд обмежень безпеки*
2. **Same-Origin Policy**

Політика сховища одного походження є важливим механізмом безпеки, який обмежує доступ JavaScript до даних на сторонніх доменах. Тобто, політика дозволяє скриптам, які працюють на сторінках, що згенеровані на одному сайті, отримати доступ до методів та атрибутів один одного без особливих обмежень, але забороняє доступ до більшості методів та властивостей на сторінках інших сайтів.

Початок форми

* **Розуміння блокування**

Основні принципи та функції SOP включають таке:

1. **Спільне походження**: SOP вимагає, щоб веб-сторінки завантажували ресурси (наприклад, JavaScript, cookies, інші дані) тільки з веб-сайтів, які мають те саме походження (схему, домен та порт) як і вихідна сторінка.
2. **Захист від Cross-Site Scripting (XSS)**: SOP допомагає запобігти атакам XSS. Зловмисник не може виконувати JavaScript на іншому домені, тому він не може отримати доступ до даних на цьому домені.

[XSS (Cross-Site Scripting) - це вразливість, яка дозволяє зловмисникам вставляти скрипти на сторінки веб-сайтів, що може призвести до крадіжки cookies користувачів та інших конфіденційних даних](https://inuasparwil.ru/vidpovidi-na-zapitannja/6908-php-bezpeka-shho-take-xss.html)[1](https://bing.com/search?q=%D1%89%D0%BE+%D1%82%D0%B0%D0%BA%D0%B5+XSS+%D0%B0%D1%82%D0%B0%D0%BA%D0%B8)[2](https://inuasparwil.ru/vidpovidi-na-zapitannja/6908-php-bezpeka-shho-take-xss.html).  [XSS-атаки можуть бути здійснені через різні канали, такі як форми, коментарі, пошукові запити та інші 3](https://techlila.com/uk/web-application-attack/).

1. **Захист від Cross-Site Request Forgery (CSRF)**: SOP допомагає захистити від атак CSRF, оскільки відправлені запити на інші домени заборонені, якщо вони виконуються через браузер. Це заважає виконанню небажаних дій на ім'я користувача без його відома.

Cross-Site Request Forgery (CSRF) атака - це тип атаки, який змушує користувача виконувати небажані дії на веб-сайті, на якому він вже авторизований. Атака ґрунтується на соціальному інжинірингу, наприклад, надсиланні посилання через електронну пошту або чат, щоб обманом натиснути на зловмисний запит. Атака можлива тому, що веб-браузери автоматично надсилають деякі типи токенів автентифікації з кожним запитом до веб-сайту. [Атака відрізняється від атаки перехоплення міжсайтових сценаріїв (XSS), оскільки не потребує впровадження шкідливого коду на веб-сторінку 1](https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-6.0).

1. **Захист кукі**: SOP вимагає, щоб куки (файли зберігання даних на боці клієнта) були призначені лише для веб-сайтів того самого походження. Це перешкоджає веб-сайтам інших доменів у викраденні або зміні ваших кукі.

* **Винятки та обхідні шляхи**

Існують деякі спеціальні методи та заголовки, такі як JSONP, CORS (Cross-Origin Resource Sharing) і деякі інші, які дозволяють об'єктам з інших доменів взаємодіяти з об'єктами на вашому домені, але вони мають бути дозволені явно. Обидва методи призначені для забезпечення взаємодії між веб-сайтами, розташованими на різних доменах, але вони мають важливі різниці. Тут я розкажу про JSONP, а про CORS розкажу згодом:

1. **JSONP (JSON with Padding)**:
   * **Опис**: JSONP - це метод, який дозволяє завантажувати дані з інших доменів за допомогою вставлення тегу **<script>** з URL іншого домену на сторінку, який завантажує дані.
   * **Переваги**:
     + Простий і підтримується багатьма браузерами.
     + Відсутність необхідності налаштувань на сервері.
   * **Недоліки**:
     + Менш безпечний, оскільки вставка скрипта з іншого домену може призвести до XSS-атак.
     + Підтримує тільки HTTP GET-запити.
     + Обмежена можливість обробки помилок та обробки статусів HTTP.

Рекомендується використовувати CORS, якщо це можливо, оскільки цей метод є безпечнішим та більш гнучким. JSONP слід використовувати лише тоді, коли CORS недоступний, і тільки з надійних джерел.

* **Використання слабких місць**

Ось деякі з найвідоміших вразливостей:

1. **JSONP (JSON with Padding)**
2. **Clickjacking**:
   * **Вразливість**: Сайти вставляються в інші сторінки так, що користувачі не бачать їх і можуть випадково виконувати дії на цих сторінках.
   * **Пом'якшення**: Використовуйте заголовки X-Frame-Options або Content Security Policy для обмеження встраивання вашого контенту на інших сторінках.

**2. Cross-Origin Resource Sharing (CORS)**

**Cross-Origin Resource Sharing** (**CORS**, спільне використання ресурсів з різних джерел) — механізм безпеки сучасних браузерів, який дозволяє вебсторінкам використовувати дані, що знаходяться на інших доменах. Тобто реалізує захист для [вебсторінок](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1%D1%81%D1%82%D0%BE%D1%80%D1%96%D0%BD%D0%BA%D0%B0" \o "Вебсторінка), які не відповідають [політиці одного походження](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%96%D1%82%D0%B8%D0%BA%D0%B0_%D1%82%D0%BE%D0%B3%D0%BE_%D0%B6_%D0%BF%D0%BE%D1%85%D0%BE%D0%B4%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F)[[1]](https://uk.wikipedia.org/wiki/Cross-Origin_Resource_Sharing#cite_note-mozhacks_cors-1).

Для реалізації цього механізму, [вебсервер](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80" \o "Вебсервер) з якого завантажується вебсторінка додає в [HTTP-заголовок](https://uk.wikipedia.org/wiki/HTTP#%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8) окремий тег, в якому перераховано домени, з яких вебсторінка може отримувати дані.

* **Походження та мета CORS**

**Походження CORS**: CORS став актуальним завдяки розвитку сучасних веб-додатків та тому, що багато ресурсів стали доступними через API, які розташовані на інших доменах. Відмова в доступі до цих ресурсів обмежувала можливості розробки веб-додатків, тому була потреба в стандартизованому механізмі для обміну ресурсами між джерелами.

**Мета CORS**: Мета CORS полягає в забезпеченні безпеки і контролю при взаємодії веб-сайтів на різних доменах. Головні цілі включають:

1. **Безпека**: CORS має захищати користувачів від небажаних атак, таких як Cross-Site Request Forgery (CSRF) та Cross-Site Scripting (XSS).
2. **Контроль доступу**: CORS дозволяє серверам встановлювати правила доступу для конкретних доменів. Це дозволяє веб-сайтам контролювати, хто може і хто не може отримувати доступ до їхніх ресурсів.
3. **Зручність розробки**: CORS дозволяє використовува ресурси з інших доменів без потреби обходити браузерні обмеження безпеки.

Початок форми

* **Роль Preflight Requests**

Запити перед початком (preflight requests) є частиною механізму CORS (Cross-Origin Resource Sharing) і використовуються для прокладання шляху для безпечного перехресного зв'язку між веб-сайтами. Ось як це працює:

1. **Попередній запит (Preflight Request)**:
   * Коли веб-сторінка робить запит до іншого джерела з різним доменом, браузер може виконати попередній запит перед основним запитом.
   * Попередній запит - це запит типу OPTIONS, який містить деякі життєво важливі метадані.
2. **Життєво важливі метадані**:
   * У попередньому запиті передаються життєво важливі метадані, такі як заголовки запиту, які містять інформацію про методи, заголовки та дозволені джерела, з яких браузер відправить основний запит.
3. **Серверна обробка**:
   * Сервер, до якого робиться запит, перевіряє ці метадані та вирішує, чи можна довіряти відправнику запиту.
   * Якщо сервер вважає, що запит є безпечним і відповідає CORS політиці, він відправляє відповідь зі статусом 200 (OK), і браузер продовжує виконувати основний запит.
4. **Основний запит (Main Request)**:
   * Після успішного попереднього запиту і відповіді від сервера, браузер відправляє основний запит, який вже містить фактичний запит до ресурсу на іншому джерелі.

* **Захист Cross-Origin запитів**

Ось кілька кроків, які можна вжити для цього:

1. **Встановіть відповідні заголовки CORS на сервері**:

Налаштуйте сервер так, щоб він включав відповідні CORS-заголовки у відповідях на запити. Основні CORS-заголовки, які слід налаштовувати, це:

1. **Origin**: Цей заголовок відправляється браузером разом із кожним запитом і вказує на походження (домен) веб-сайту, що відправляє запит.
2. **Access-Control-Allow-Origin**: Цей заголовок вказує, які домени мають доступ до цих ресурсів.
3. **Access-Control-Allow-Methods**: Цей заголовок вказує дозволені HTTP-методи, які можуть використовуватися для доступу до ресурсів.
4. **Access-Control-Allow-Headers**: Цей заголовок вказує, які HTTP-заголовки можуть бути включені в запити.
5. **Використовуйте мінімальні дозволи**:

Не надавайте надмірних дозволів. Встановлюйте лише необхідні дозволи для конкретних доменів та ресурсів.

1. **Валідація вхідних даних**:

Перевіряйте та валідуйте всі вхідні дані, щоб запобігти SQL-ін'єкціям, XSS-атакам та іншим видам атак на ваш додаток.

1. **Використання Content Security Policy (CSP)**:

Встановіть CSP, який дозволяє вам обмежити, з яких джерел можна завантажувати ресурси, і які типи скриптів можна виконувати.

1. **Використовуйте HTTPS**:

Використовуйте шифроване з'єднання HTTPS, щоб захистити дані, які передаються між клієнтом і сервером.

Правильно налаштована політика CORS разом із зазначеними вище заходами допоможе забезпечити безпеку вашого веб-додатка і захистити конфіденційні дані користувачів від несанкціонованого доступу.

1. **Content Security Policy (CSP)**

Content Security Policy (CSP) - це механізм, який дозволяє веб-розробникам встановлювати жорсткі правила щодо того, які ресурси можуть бути завантажені на сторінку, і звідки вони можуть бути завантажені.

Якщо сторонній JavaScript-код не знаходиться в списку дозволених джерел, браузер буде блокувати його виконання, навіть якщо він намагається вставити його на сторінку через XSS. Ця механіка сприяє збереженню безпеки та цілісності веб-додатків.

* **Введення в CSP**

Основні принципи та функціональність CSP:

1. **Відповідність засобами забезпечення**:
   * CSP дозволяє вам встановити політику безпеки, яка відповідає вашим потребам і ресурсам.
   * Вона допомагає забезпечити, що лише безпечний та авторизований контент буде завантажено та виконано на сторінці.
2. **Заборона атак XSS**:
   * CSP може блокувати вбудову зловмисного JavaScript-коду на сторінку через відмову виконувати скрипти, якщо вони не відповідають налаштованій політиці.
   * Видаляє велику частину потенційних вразливостей XSS.
3. **Можливість повідомлення про помилки**:
   * Якщо скрипт або ресурс не відповідає політиці безпеки, CSP може встановити заголовок повідомлення про помилку, який допомагає виявити та усунути проблеми.

Загалом, CSP є потужним інструментом безпеки, який забезпечує важливий щит від атак і цю політику варто розглядати як необхідну частину захисту вашого додатка.

* **Захист від Code Injection**

Існують різні методи застосування директив CSP для захисту веб-програм від атак із впровадженням коду. Ось кілька з них:

* **default-src**: Вказує джерела, з яких можна завантажувати будь-який контент.
* **script-src**: Вказує джерела, з яких можна завантажувати скрипти.
* **style-src**: Вказує джерела, з яких можна завантажувати таблиці стилів.
* **img-src**: Вказує джерела, з яких можна завантажувати зображення.
* **Забезпечення безпеки скриптів**

Ось кроки для налаштування жорсткої політики безпеки JavaScript:

1. **Відключіть внутрішні скрипти і inline-код**:

Додайте директиви, щоб вимкнути внутрішні скрипти і inline-код, оскільки ці елементи вважаються менш безпечними. Це дозволить лише виконання скриптів з власного домену.

1. **Додайте надійні джерела до білого списку**:

Додайте надійні джерела, з яких ви дозволяєте завантажувати скрипти або ресурси. Наприклад, якщо ви хочете дозволити завантажувати скрипти з Google CDN, додайте їх у директиву. Ця директива дозволить завантажувати скрипти лише з власного домену та з Google CDN.

1. **Блокування небезпечних дій**:

Додайте директиви, щоб блокувати небезпечні дії, такі як використання **eval**, вбудову скриптів з **data:** або **blob:** URI, а також інші можливі джерела вразливостей. Ця директива блокує використання об'єктів, а також скрипти з небезпечних джерел.

1. **Регулярно перевіряйте і оновлюйте політику**:

Періодично перевіряйте і оновлюйте політику CSP, оскільки ваш додаток може розвиватися і змінюватися, а зміни в коді можуть вплинути на потребу в додаванні нових надійних джерел або блокуванні нових потенційних загроз.

1. *Найкращі методи захисту JavaScript на стороні клієнта*

* **Input Validation**

(Щоб запобігти вразливостям системи безпеки, можна використовувати наступні методи перевірки введених користувачем даних на стороні клієнта:

1. **Валідація даних**: Перевірка правильності введених даних на стороні клієнта перед їх відправкою на сервер.
2. **Капча**: Це технологія, яка дозволяє перевірити, чи є користувачем людина, а не бот.
3. **Токени**: Токени можуть бути використані для перевірки автентичності користувача. Токен - це унікальний ідентифікатор, який генерується сервером і передається клієнту. Клієнт повинен повернути токен на сервер для підтвердження своєї автентичності.
4. **Хешування паролю**: Пароль може бути захешований на стороні клієнта перед відправкою на сервер.

* **Secure Communications**

Щоб забезпечити безпечне спілкування між клієнтом і сервером, можна використовувати шифрування та надійні протоколи.

Один з найбільш поширених протоколів для забезпечення безпеки спілкування між клієнтом і сервером - це **TLS (Transport Layer Security)**

Можна також використовувати **симетричне шифрування**. [При симетричному шифруванні обидва боки спілкування використовують один і той же ключ для шифрування та розшифрування даних 3](https://www.f5.com/glossary/ssl-tls-encryption).

Іншим протоколом, який можна використовувати для забезпечення безпеки спілкування між клієнтом і сервером, є **HTTPS (Hypertext Transfer Protocol Secure)**.

* **Code Minification**

Видалення непотрібних коментарів та пробілів може зменшити розмір коду JavaScript, що може зменшити поверхню атаки. Це означає, що зловмисники матимуть менше можливостей для використання вразливостей у вашому коді. Однак, це не є гарантією безпеки.

* **Code Obfuscation**

Обфускація - це процес перетворення зрозумілого коду на менш зрозумілий шляхом заміни імен функцій та змінних на складночитаємий текст або короткі символи. Це може ускладнити розуміння коду для зловмисників, але не є гарантією безпеки.

1. *Тематичні дослідження та приклади*

* **Найважчі битви веб-гіганта**

[Наприклад, у 2014 році eBay став жертвою атаки XSS, яка була успішною завдяки тому, що зловмисник вставив шкідливий код на стороні клієнта через форму пошуку 1](https://www.geeksforgeeks.org/difference-between-xss-and-csrf/). [У 2018 році GitHub став жертвою атаки CSRF, яка була успішною завдяки тому, що зловмисник використовував попередньо отримане cookie для виконання шкідливих запитів на сторонньому сайті](https://portswigger.net/web-security/csrf/xss-vs-csrf)

* **Впровадження безпечного JavaScript**

(Ось кілька компаній, які успішно впровадили безпечний клієнтський JavaScript у своїх веб-додатках:

Великі компанії активно використовують клієнтський JavaScript. Такі компанії як Гугл, Фейсбук, Твітер, ГітХаб використовують Content Security Policy для захисту від атак і вразливостей. Також Фейсбук використовує React, який автоматично екранує вхідні дані.

Ці компанії є прикладами успішної реалізації безпечного клієнтського JavaScript у своїх веб-додатках.

* **Поради з лінії фронту**

Також деякі передові практики та поради від досвідчених розробників, які орієнтуються в складнощах клієнтської безпеки JavaScript:

* + 1. **Використовуйте Content Security Policy (CSP)**:
  + Налаштуйте директиви CSP для обмеження виконання JavaScript та завантаження ресурсів з ненадійних джерел.
  + Слідкуйте за звітами про помилки CSP і реагуйте на них швидко.

1. **Уникайте використання eval і Function конструктора**:
   * Використовуйте ці функції обережно, оскільки вони можуть виконувати довільний код і створювати вразливості.
2. **Валідуйте та фільтруйте вхідні дані**:
   * Всі дані, які потрапляють на сторінку або оброблюються на клієнті, повинні бути обов'язково валідовані та очищені від потенційно небезпечного коду.
3. **Використовуйте бібліотеки та фреймворки з вбудованими механізмами безпеки**:
   * Багато популярних бібліотек і фреймворків, таких як React або Angular, мають вбудовані заходи безпеки для обмеження атак XSS та інших загроз.
4. **Захищайте конфіденційні дані користувачів**:
   * Зашифровуйте конфіденційні дані перед їх передачею через мережу.
   * Використовуйте токени аутентифікації та сесійні куки для контролю доступу.
5. **Перевіряйте свій код на вразливості з регулярністю**:
   * Використовуйте інструменти та сканери безпеки для автоматичної перевірки коду на наявність вразливостей.
6. **Постійно оновлюйте залежності та бібліотеки**:
   * Перевіряйте, чи всі використовувані бібліотеки мають оновлені безпеку версії і оновлюйте їх, якщо необхідно.
7. *Запитання до аудиторії*

* Що таке XSS атаки?:

Це атаки, де зловмисник вставляє шкідливий код, за допомогою якого може викрадати конфеденційні дані користувачів.

* Що таке CORS?:

Це по суті такий механізм безпеки, який дозволяє вебсторінкам використовувати дані, що знаходяться на інших доменах