Engineer

CSRF

атака, що змушує браузер надіслати небажаний запит

Що це таке

CSRF, або підробка міжсайтових запитів (англійською cross-site request forgery), — це атака, за якої сторінка, яку жертва випадково відкриває, змушує власний браузер жертви надіслати запит на інший сайт, той, де жертва увійшла в обліковий запис. Браузер, як завжди, додає cookie цього сайту, тож запит надходить із сесією жертви й виглядає як дія, яку людина обрала сама. Common Weakness Enumeration веде цю ваду як CWE-352 і описує її з боку сервера: застосунок не перевіряє, або не може перевірити, достатньо надійно, чи запит навмисно надіслав той користувач, від якого він нібито надходить.

Шпаргалка OWASP із запобігання CSRF (cheat sheet) перелічує, звідки може початися атака, від шкідливого вебсайту до електронного листа, допису в блозі, миттєвого повідомлення чи програми, і чого вона може досягти: переказу коштів, зміни пароля, покупки, надання привілеїв. Кожне з цього — дія на сайті, зміна стану, і саме там шпаргалка ставить захист: токен у кожному запиті, що змінює стан. CWE додає, що масштаб обмежують лише привілеї жертви, бо зловмисник позичає її особу. Якщо жертва — адміністратор, наслідком може стати повний контроль над вебзастосунком.

CWE розглядає CSRF як складену ваду, що існує лише там, де сходяться кілька інших. Серед них — помилка перевірки походження, CWE-346, і ненавмисний проксі, відомий як «confused deputy», CWE-441: браузер тримає облікові дані жертви й витрачає їх від імені сторінки, яка власних не має. Це відрізняє CSRF від міжсайтового скриптингу, який CWE наводить як споріднену ваду того самого рівня. За XSS код зловмисника виконується всередині цільового сайту; за CSRF він лишається на сторінці зловмисника й лише надсилає запити. Це відрізняє його й від CORS, який визначає, що сторінка може читати з іншого походження. OWASP зазначає, що звичайна HTML-форма надсилає простий запит, тобто такий, що не потребує

попереднього запиту (preflight), а браузери дозволяють такі запити до будь-якого походження. У назві запису CWE термін пишеться з великих літер, Cross-Site Request Forgery, а сам запис наводить інші назви того самого явища, серед них XSRF і Session Riding.

Застосуйте поняття до рішення

Застосуйте «Що таке CSRF?» 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

CSRF тримається в обох найвідоміших списках вад. CWE-352 входив до CWE Top 25 у 2009, 2010 і 2011 роках і в кожну редакцію з 2019 до 2025 року. OWASP виділяв для нього окрему категорію в Top 10 2007 і 2010 років, як A5, і 2013 року, як A8. Від 2021 року він входить до A01, Broken Access Control, категорії, що стоїть першою в редакції 2025 року. OWASP повідомляє, що в усіх без винятку протестованих застосунках знайшлася та чи інша форма порушення контролю доступу, і називає CWE-352 серед помітних вад цієї категорії.

Атаку важко викорінити, бо роботу виконує браузер. Чернетка W3C про Fetch Metadata виходить саме з цього: браузери легко змусити надсилати запити на ендпоінти сайту й додавати до них облікові дані, які користувач уже має, наприклад cookie чи привілейоване становище в інтранеті, тож застосункам доводиться дуже уважно стежити за тим, як ці ендпоінти поводяться, а це виявляється складно. CWE зазначає, що атаку можна влаштувати майже з будь-якого тегу чи конструкції HTML, зокрема з тегів зображень і посилань. Зафіксовані приклади в записі читаються як перелік адміністративних дій, виконаних із чужої сторінки: облікові записи користувачів, додані через URL у тегу `img` (CVE-2004-1703), права адміністратора, здобуті так само (CVE-2004-1842), і вебінтерфейс, у якому підроблений запит міг змінити паролі або зупинити віртуальну машину (CVE-2009-3759).

Сканери тут мало допомагають. CWE пише, що наразі CSRF важко надійно виявити автоматичними методами, бо кожен застосунок має власну неявну політику щодо того, які запити може спричинити сторонній, а які потребують твердої певності, що користувач справді цього хотів. Інструмент не може знати, що одна форма нешкідлива, а наступна переказує гроші. Тож CSRF — питання проєктування: вирішуйте для кожного запиту окремо, що доводить намір користувача.

Браузери зрушили на бік захисників. Шпаргалка OWASP каже, що сучасні поширені браузери за замовчуванням застосовують до cookie SameSite=Lax, а це не пускає їх у більшість міжсайтових запитів `POST`, і що всі основні браузери надсилають заголовки Fetch Metadata з березня 2023 року. Обидва механізми допомагають, і жоден не закриває питання. Чернетка про cookie, яка визначає SameSite, називає Lax розумним додатковим рівнем захисту від атак, що покладаються на небезпечні методи на кшталт `POST`, але не надійним захистом від CSRF як класу атак, і радить не вважати навіть режим Strict усім захистом сайту. Користувачі старіших або вбудованих браузерів, додає OWASP, можуть отримати cookie, які поводяться так, наче значення SameSite не задано зовсім.

Як це виглядає на практиці

Візьмімо вебзастосунок на `app.example.com`, де користувач, який увійшов у систему, змінює адресу електронної пошти свого облікового запису формою, що надсилається методом `POST` на `/account/email`. Cookie сесії має атрибути `SameSite=None; Secure`, бо та сама панель вбудована ще й у портал партнера, а None — це значення, яке дозволяє браузеру надсилати cookie в міжсайтовому контексті. Зловмисник публікує сторінку на `prizes.example.net` і домагається, щоб жертва її відкрила:

```
<form action="https://app.example.com/account/email" method="post">
  <input type="hidden" name="email" value="mallory@example.net">
</form>
<script>document.forms[0].submit();</script>
```

Браузер надсилає форму, щойно сторінка завантажиться, і оскільки cookie це дозволяє, сесія йде разом із запитом:

```
POST /account/email HTTP/1.1
Host: app.example.com
Origin: https://prizes.example.net
Sec-Fetch-Site: cross-site
Content-Type: application/x-www-form-urlencoded
Cookie: session=...

email=mallory%40example.net
```

Для сервера це користувач, який увійшов у систему й змінює свою адресу. Якщо більше нічого не перевіряти, адреса зміниться, а лист для скидання пароля, надісланий на нову адресу, віддасть обліковий запис зловмиснику.

Два заголовки в цьому запиті належать браузеру, а не зловмиснику. Стандарт Fetch відносить `Origin` до заборонених заголовків запиту разом з усіма заголовками, назва яких починається з `Sec-`, тож скрипт на сторінці не може ні задати, ні змінити їх. `Origin` називає походження, з якого надіслано запит, а чернетка Fetch Metadata визначає для `Sec-Fetch-Site` чотири значення: `cross-site`, `same-site`, `same-origin` і `none` для навігації, яку почав сам користувач, ввівши адресу або обравши закладку. Перевірка може спиратися на обидва заголовки й однаково мати запасний варіант із токеном для браузерів, що не надсилають жодного з них. Ескіз на Python:

```
import hmac
import secrets

SAFE_METHODS = {"GET", "HEAD", "OPTIONS", "TRACE"}
OUR_ORIGIN = "https://app.example.com"


def new_csrf_token(session: dict) -> str:
    session["csrf_token"] = secrets.token_urlsafe(32)
    return session["csrf_token"]


def csrf_ok(method: str, headers: dict, form: dict, session: dict) -> bool:
    if method in SAFE_METHODS:
        return True
    if headers.get("Sec-Fetch-Site") == "cross-site":
        return False
    origin = headers.get("Origin")
    if origin is not None and origin != OUR_ORIGIN:
        return False
```

```
    expected = session.get("csrf_token", "").encode()
    sent = form.get("csrf_token", "").encode()
    return bool(expected) and hmac.compare_digest(sent, expected)
```

Токен дає `secrets.token_urlsafe(32)`. Документація Python каже, що модуль `secrets` бере найбезпечніше джерело випадковості, яке надає операційна система, і що 32 байти, тобто 256 біт, вважаються достатніми. Сервер кладе токен у приховане поле кожної форми, яку генерує, і зберігає копію в сесії. Токен ніколи не потрапляє в cookie, URL чи журнал: саме ці місця шпаргалка OWASP виключає для токена синхронізації (synchronizer token). `hmac.compare_digest` навмисно не зупиняється на першому байті, що відрізняється, тож час порівняння нічого не каже про те, яку частину здогадки вгадано. Обидва значення спершу кодуються в байти, бо рядки тексту функція приймає лише тоді, коли вони суто ASCII. Запит без заголовка `Origin` переходить до перевірки токена, а `Origin: null` її не проходить, бо пропускається лише точне походження.

У справжньому застосунку спершу беріть захист, вбудований у фреймворк, а не пишіть власний; це перший пункт у списку OWASP. Проміжний шар (middleware) Django, наприклад, вимагає і свій CSRF-cookie, і поле `csrfmiddlewaretoken` для кожного методу, крім `GET`, `HEAD`, `OPTIONS` і `TRACE`, відповідає кодом 403, коли чогось із цього бракує або воно хибне, і звіряє `Origin` з хостом і з налаштуванням `CSRF_TRUSTED_ORIGINS`. Там, де нікому не потрібно вбудовувати застосунок, cookie сесії можна також надсилати як `__Host-session=...; Path=/; Secure; SameSite=Lax`. OWASP зазначає, що cookie з префіксом `__Host-` не можна перезаписати з іншого піддомену.

Налаштування браузера — це шар, а не сам захист. У портфоліо є опис того, як посилено захист застосунку за допомогою CSP на основі nonce, і там сказано, що проміжний шар на фронтенді встановлює HSTS і cookie SameSite разом із Content Security Policy, а права доступу сервер повторно перевіряє під час кожного запиту, тоді як обмеження на фронтенді вважаються суто питанням UX.

Де виникають проблеми

Зміна стану через GET. SameSite=Lax однаково надсилає cookie під час міжсайтової навігації верхнього рівня з безпечним методом, тож посилання, на яке клацає жертва, або сторінка, що перенаправляє все вікно на `https://app.example.com/account/delete?confirm=yes`, приходить уже з сесією.

OWASP називає це найпоширенішим способом, у який захист на основі SameSite підводить на практиці, а Django будує свій захист на тому самому засновку: `GET` та інші безпечні методи не повинні мати побічних ефектів. Тримайте `GET`, `HEAD`, `OPTIONS` і `TRACE` без побічних ефектів, а кожну зміну ставте за метод, який покривають ваші перевірки.

SameSite як увесь захист. Чернетка про cookie пояснює, чому Lax самого по собі мало: зловмисник однаково може відкрити нове вікно або запустити навігацію верхнього рівня, щоб запит вважався same-site, а попередній рендеринг (prerendering) може зробити те саме непомітно для користувача. OWASP додає, що SameSite прив'язаний до домену, який можна зареєструвати, а не до походження. Запит із будь-якого хоста під example.com вважається same-site для `app.example.com`, тож забутий піддомен, який хтось перехопив, уже опиняється всередині огорожі. Вважайте SameSite другим шаром, а рішення ухвалюйте на сервері за токеном або перевіркою походження.

Токен, який може підкинути зловмисник. У наївному шаблоні double-submit сервер лише перевіряє, що cookie й поле форми містять те саме випадкове значення. OWASP тепер радить цей шаблон не використовувати, бо будь-хто, хто може записувати cookie для домену, через вразливий сусідній піддомен, захоплення DNS або cookie, впроваджений через звичайний HTTP, може задати обидві половини. Запис CWE досі наводить простий подвійно надісланий cookie серед заходів пом'якшення, тож тут джерела розходяться, і слід дотримуватися застереження OWASP. Зберігайте токен у сесії на сервері або використовуйте підписаний double-submit cookie від OWASP, прив'язаний до сесії через HMAC.

Міжсайтовий скриптинг на тому самому сайті. Скрипт, що виконується на ваших власних сторінках, може прочитати токен із форми й надіслати запит, який за будь-якою мірою є same-origin. Шпаргалка OWASP каже про це прямо: XSS може звести нанівець усі техніки захисту від CSRF. CWE і Django кажуть те саме, а найвідоміший приклад у CWE показує, як ці дві вади працюють разом. Хробак Samy через XSS вбудував себе в профіль MySpace і додав автора атаки в друзі, а друзі, які переглядали профіль, отримували таку саму зміну у власних профілях; оскільки жодна жертва цієї зміни не обирала, CWE вважає CSRF однією з першопричин. Вважайте кожну знахідку XSS також знахідкою CSRF і виправляйте її першою.

Перевірка походження, що пропускає забагато. Перевірка на кшталт `origin.startswith("https://app.example.com")` пропускає й `https://app.example.com.attacker.example`, і саме від цієї пастки застерігає шпаргалка OWASP. Значення `null` — ще одна. За стандартом Fetch форма, надіслана зі сторінки з політикою референта `no-referrer`, приходить з `Origin: null`, а OWASP додає, що його можуть дати й переспрямування між різними походженнями та фрейми в пісочниці. Порівнюйте походження повністю з фіксованим списком точних рядків, а відсутній заголовок чи `null` вважайте невідомим, ніколи не вашим власним походженням.

Надмірна довіра до preflight. Власний заголовок запиту — надійний захист для API, бо браузери надсилають попередній запит (preflight) перед кожним запитом до іншого походження, що має такий заголовок. Але це працює, лише якщо API відкидає все інше. OWASP застерігає, що до JSON API, який приймає й `text/plain`, можна дістатися з форми, бо це один із трьох типів вмісту, дозволених простому запиту, а політика CORS, що дозволяє облікові дані з усіх піддоменів через регулярний вираз, дає кожному, хто перехопив піддомен, змогу надіслати власний заголовок. Відкидайте типи вмісту, яких ендпоінт не очікує, і перелічуйте точні походження, яким дозволено надсилати запити з обліковими даними.

Забута форма входу. CSRF під час входу (login CSRF) працює навпаки: зловмисник змушує браузер жертви увійти в обліковий запис самого зловмисника. У прикладі OWASP з інтернет-магазином жертва потім вводить дані своєї картки, і зловмисник згодом може ними розраховуватися. Документація Django називає login CSRF серед атак, від яких захищає її механізм. Захистіть форму входу токеном, прив'язаним до попередньої сесії, а коли користувач увійде, знищте цю сесію й почніть нову, як радить OWASP.

Запити, зібрані з даних зловмисника. CSRF на боці клієнта (client-side CSRF) виникає, коли власний JavaScript застосунку збирає запит із вхідних даних, які контролює зловмисник, наприклад з URL, імені вікна чи `postMessage`. Скрипт сам додає токен, а запит є same-site, тож його не зупиняють ні токени, ні SameSite; OWASP описує це як повернення проблеми confused deputy. Не давайте вхідним даним зловмисника визначати адресатів і параметри запитів або обирайте їх із фіксованого списку в коді.

Що почитати далі

- CWE-352: Cross-Site Request Forgery (CSRF) — запис про ваду: складена структура, наслідки, заходи пом’якшення й зафіксовані CVE.
- OWASP Cheat Sheet Series: Cross-Site Request Forgery Prevention — токени, Fetch Metadata, перевірки походження, межі SameSite і login CSRF.
- W3C: Fetch Metadata Request Headers — заголовок `Sec-Fetch-Site` і чотири значення, які може надіслати браузер.
- IETF draft: Cookies: HTTP State Management Mechanism (draft-ietf-httpbis-rfc6265bis) — атрибут SameSite і чому Lax — додатковий рівень захисту, а не ліки; досі Internet-Draft.
- Django documentation: Cross Site Request Forgery protection — вбудований захист фреймворку, що він передбачає і де піддомени його обходять.
- OWASP Top 10:2025: A01 Broken Access Control — де CWE-352 стоїть у редакції 2025 року.

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 CSRF?» 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/csrf/