# Ataki CSRF na aplikacje webowe

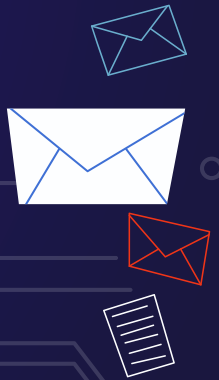
Michał Chmiel
Dawid Gruszecki
Mateusz Sznurawa

# Co to jest CSRF ?

CSRF (Cross-Site Request Forgery) to rodzaj ataku, który zmusza użytkownika do wykonania niechcianej akcji na stronie internetowej, na której jest aktualnie zalogowany.

Atak CSRF wykorzystuje autoryzację sesji użytkownika, aby wykonać nieautoryzowane żądanie do aplikacji, np. zmiana hasła, usunięcie konta.

Przykłady ataków: zmiana danych konta, wykonanie operacji bankowych, publikacja postów w mediach społecznościowych.



# Jak działa atak CSRF ?

**01** Użytkownik jest zalogowany do aplikacji webowej (np. banku).

**02** Atakujący przygotowuje złośliwą stronę lub e-mail z linkiem do aplikacji ofiary.

**03** Kliknięcie w link powoduje wykonanie żądania HTTP (np. przelewu, zmiany hasła) do aplikacji, wykorzystując sesję użytkownika.

**04** Aplikacja nie weryfikuje, czy żądanie pochodzi od użytkownika, więc akcja jest wykonywana.

# Typy ataków CSRF

# Ataki oparte na formularzach

## Ataki oparte na formularzach:

Formularz w aplikacji wymaga autoryzacji (np. zmiana hasła, adresu e-mail, przelew pieniędzy). Aplikacja sprawdza tylko, czy użytkownik jest zalogowany, ale nie weryfikuje pochodzenia żądania.

## Złośliwa strona atakującego:

Atakujący tworzy stronę, która zawiera formularz wysyłający dane do aplikacji ofiary. Formularz jest wysyłany automatycznie (np. za pomocą JavaScript) lub po kliknięciu użytkownika.

## Eksploatacja zaufania aplikacji:

Użytkownik jest zalogowany na stronie ofiary i odwiedza stronę atakującego. Aplikacja ofiary wykona operację (np. zmieni dane, przeleje środki) na podstawie formularza, ponieważ ufa, że użytkownik sam wysłał żądanie.

```html
<form action="https://bank.com/update_email" method="POST">
    <label for="email">Nowy e-mail:</label>
    <input type="email" id="email" name="email">
    <button type="submit">Zmień e-mail</button>
</form>
```

```html
<html>
<body>
    <h1>Witaj na naszej stronie!</h1>
    <form action="https://bank.com/update_email" method="POST" style="display:none;">
        <input type="hidden" name="email" value="attacker@example.com">
    </form>
    <script>
        // Automatyczne wysłanie formularza
        document.forms[0].submit();
    </script>
</body>
</html>
```

# Ataki oparte na linkach

W aplikacji ofiary użytkownik może usunąć swoje konto klikając w odpowiedni link, który wywołuje metodę GET.

Aplikacja przetwarza żądanie GET i na podstawie parametrów URL (*account_id* oraz *confirm=true*) usuwa konto użytkownika.

```
<!-- Link do usunięcia konta użytkownika -->
<a href="https://victimwebsite.com/delete_account?account_id=12345&confirm=true">Usuń konto</a>
```

# Ataki oparte na JavaScript

Ataki CSRF oparte na JavaScript są bardziej zaawansowane, ponieważ wykorzystują manipulację skryptami w celu wykonania nieautoryzowanych działań w aplikacjach webowych. Złośliwy skrypt może wysyłać żądania do aplikacji ofiary, wykorzystując aktywną sesję użytkownika (np. ciasteczka sesyjne), by przeprowadzić atak bez jego wiedzy.

Atakujący wykorzystuje obiekt XMLHttpRequest w JavaScript do wysłania żądania POST lub GET do aplikacji ofiary.


Podobnie jak w przypadku XMLHttpRequest, atakujący może użyć nowoczesnej funkcji fetch() w JavaScript do wysyłania żądań do aplikacji ofiary.

# Wykorzystanie obiektu XMLHttpRequest

```
<script>
    // Złośliwy skrypt, który wysyła żądanie POST do aplikacji ofiary
    document.getElementById('attackButton').onclick = function() {
        var xhr = new XMLHttpRequest();
        xhr.open('POST', 'https://victimsite.com/transfer', true);
        xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

        // Dane do żądania (np. transfer środków)
        var data = 'amount=1000&to_account=attacker_account&csrf_token=123456';

        // Wysłanie żądania (z automatycznie załadowanymi ciasteczkami sesji)
        xhr.send(data);
    };
</script>
```

# Funkcja fetch

```
<script>
    // Złośliwy skrypt, który używa funkcji fetch() do wysłania żądania POST
    document.getElementById('attackButton').onclick = function() {
        fetch('https://victimsite.com/transfer', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/x-www-form-urlencoded'
            },
            body: 'amount=1000&to_account=attacker_account'
        });
    };
</script>
```

# Konsekwencje ataku CSRF

Użytkownik nieświadomie wykonuje akcję, którą normalnie by odrzucił.

Atakujący może przejąć kontrolę nad ważnymi operacjami (np. zmiana hasła, transfer środków).

Może prowadzić do utraty danych, pieniędzy i dostępu do kont użytkowników.

# Jak wykrywać ataki

- Brak zabezpieczeń na żądaniach: Aplikacje, które nie implementują mechanizmów ochrony, są podatne na CSRF.
- **Brak unikalnych tokenów w formularzach**: Jeśli formularze nie zawierają tokenów CSRF, atakujący może w łatwy sposób przeprowadzić atak.
- **Brak weryfikacji źródła żądania**: Aplikacje, które nie weryfikują pochodzenia żądania (nagłówki Referer lub Origin), mogą być narażone na CSRF.

# Dobre praktyki w tworzeniu bezpiecznych aplikacji webowych

| |
|---|
| Stosowanie walidacji zarówno po stronie klienta, jak i serwera - aby zapewnić, że dane wprowadzane przez użytkowników są poprawne. |
| Dostosowywanie poziomu uprawnień użytkowników, aby niektóre czynności były dostępne tylko dla uprzywilejowanych użytkowników. |
| Regularne aktualizacje i usuwanie luk bezpieczeństwa. |
| Stosowanie sprawdzonych bibliotek oraz frameworków. |

# Jak chronić aplikacje przed CSRF?

**Tokeny CSRF**: Stosowanie unikalnych tokenów w formularzach, które są sprawdzane przy każdym żądaniu. Token jest generowany przez serwer i weryfikowany przy każdym żądaniu.  Przykład:

**<input type="hidden" name="csrf_token" value="random_generated_token">**

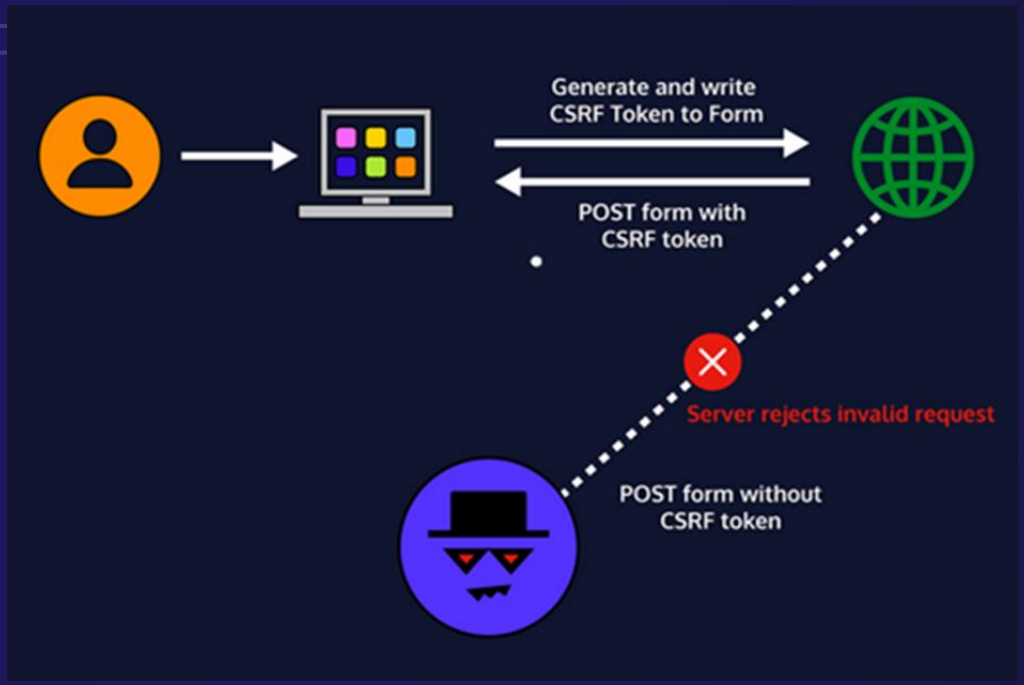
**Weryfikacja nagłówków**: Sprawdzanie nagłówków Referer i Origin w celu upewnienia się, że żądanie pochodzi z autoryzowanego źródła.

**SameSite Cookies**: Użycie atrybutu SameSite w ciasteczkach, aby ograniczyć możliwość wysyłania ciasteczek przez inne strony.

**Set-Cookie: sessionid=xyz; SameSite=Strict**

# Tokeny CSRF - jak działają i dlaczego są skuteczne

| |
|---|
| Serwer generuje unikalny token przy każdej sesji użytkownika (lub operacji). |
| Token jest przesyłany do klienta (np. w ukrytym polu formularza lub nagłówku żądania AJAX). |
| Serwer weryfikuje token w każdym żądaniu, aby upewnić się, że pochodzi ono od autoryzowanego użytkownika. |
| Nawet jeśli napastnik zna sesję użytkownika, nie może wygenerować poprawnego tokena (nie przewidzi go). |
| Każdy token ma dwie kopie (jedna zapisana na serwerze, druga przekazywana jako ukryte pole w formularzu albo nagłówek żądania HTTP). |


Generate and write
CSRF Token to Form
POST form with
CSRF token
Server rejects invalid request
POST form without
CSRF token

# Przykład implementacji ochrony CSRF

```php
if (password_verify(password: $_POST['password'], hash: GLOBAL_PASSWORD_HASH)) {
    $_SESSION['id'] = generateSessionId();
    if (empty($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = bin2hex(string: random_bytes(length: 32));
    }
```

```php
<form action="delete_account.php" method="post">
    <input type="hidden" name="csrf_token" value="<?php echo htmlspecialchars($_SESSION['csrf_token']); ?>">
    <button type="submit">Delete Account</button>
</form>
```

```php
if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    // Token CSRF nieprawidłowy - podejrzane żądanie
    die("Invalid CSRF Token");
}
```

# Same Site cookies

Jest to atrybut ciasteczek, który ogranicza ich przesyłanie w żądaniach między stronami. Ogranicza automatyczne przesyłanie ciasteczek w żądaniach cross-origin, co blokuje większość ataków CSRF.

**Rodzaje wartości SameSite:**

## 01 Strict

- Ciasteczka są przesyłane **tylko** podczas żądań z tej samej domeny (origin).
- Idealne dla wrażliwych operacji, np. logowania.
- Przykład: Żądania nawigacyjne z innych stron nie zawierają ciasteczek.

## 02 Lax

- Ciasteczka są przesyłane podczas nawigacji z linków zewnętrznych (GET, HEAD, SAFE).
- Ograniczone do prostych żądań (np. otwarcia linku), ale chronią przed CSRF dla innych metod (POST, PUT).

## 03 None

- Ciasteczka są przesyłane zawsze, nawet w żądaniach cross-origin.
- Muszą być oznaczone jako **Secure**, co wymaga HTTPS.
- Używane, gdy aplikacje działają w wielu domenach.

# Weryfikacja nagłówków

- Serwer analizuje **nagłówki HTTP** przesyłane w żądaniach od klienta, aby upewnić się, że pochodzą one z zaufanego źródła.

- Kluczowe nagłówki używane do ochrony przed CSRF to Referer i Origin.

# Referer

- Zawiera adres URL strony, z której pochodzi żądanie.
- Weryfikacja: Sprawdzenie, czy Referer zgadza się z domeną aplikacji.
- Przykład: Referer: https://moja-strona.pl/formularz

# Origin

- Dokładniejszy niż Referer, zawiera tylko schemat, domenę i port (bez ścieżki.
- Używany w żądaniach typu CORS oraz w większości żądań POST.
- Przykład: Origin: https://moja-strona.pl