Podatność CSRF

Marek Miśkiewicz

UMCS 2024

Ostrzeżenie: Prawne Konsekwencje Ataków na Aplikacje Webowe

Zwracam uwagę, że **symulowanie ataków na aplikacje webowe, takie jak SQL Injection, XSS czy CSRF, w środowiskach produkcyjnych lub bez wyraźnej zgody właściciela systemu, jest naruszeniem prawa.** Zgodnie z **polskim Kodeksem Karnym (art. 267–269)** takie działania mogą być uznane za przestępstwo zagrożone karą grzywny, ograniczenia wolności lub pozbawienia wolności.

Proszę pamiętać, że analiza podatności i testowanie bezpieczeństwa powinny być przeprowadzane wyłącznie w środowiskach testowych lub na podstawie **jasnej zgody właściciela systemu**. Wszelkie działania bez upoważnienia są niezgodne z prawem i mogą prowadzić do poważnych konsekwencji prawnych.

Wykorzystujcie zdobytą wiedzę wyłącznie w celach edukacyjnych i zgodnie z etyką zawodową!

Kodeks karny

Art. 267.

§ 1. Kto bez uprawnienia uzyskuje dostęp do informacji dla niego nieprzeznaczonej, otwierając zamknięte pismo, podłączając się do sieci telekomunikacyjnej lub przełamując albo omijając elektroniczne, magnetyczne, informatyczne lub inne szczególne jej zabezpieczenie, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 2.
§ 2. Tej samej karze podlega, kto bez uprawnienia uzyskuje dostęp do całości lub części systemu informatycznego

Art. 268.

§ 1. Kto, nie będąc do tego uprawnionym, niszczy, uszkadza, usuwa lub zmienia zapis istotnej informacji albo w inny sposób udaremnia lub znacznie utrudnia osobie uprawnionej zapoznanie się z nią, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 2.
§ 2. Jeżeli czyn określony w § 1 dotyczy zapisu na informatycznym nośniku danych, sprawca podlega karze pozbawienia wolności do lat 3.

Art. 269a.

Kto, nie będąc do tego uprawnionym, przez transmisję, zniszczenie, usunięcie, uszkodzenie, utrudnienie dostępu lub zmianę danych informatycznych, w istotnym stopniu zakłóca pracę systemu informa- tycznego, systemu teleinformatycznego lub sieci teleinformatycznej, podlega karze pozbawienia wolności od 3 miesięcy do lat 5.

Art. 269c.

Nie podlega karze za przestępstwo określone w art. 267 § 2 lub art. 269a, kto działa wyłącznie w celu zabezpieczenia systemu informatycznego, systemu teleinformatycznego lub sieci teleinformatycznej albo opracowania metody takiego zabezpieczenia i niezwłocznie powiadomił dysponenta tego systemu lub sieci o ujawnionych zagrożeniach, a jego działanie nie naruszyło interesu publicznego lub prywatnego i nie wyrządziło szkody.

Bibliografia

1. **Bezpieczeństwo aplikacji webowych**, SECURITIUM, Kraków 2019
2. https://owasp.org/www-community/attacks/csrf
3. https://cwe.mitre.org/data/definitions/352.html

Definicja CRSF (Cross-Site Request Forgery)

Cross-Site Request Forgery (CSRF lub XSRF) to rodzaj podatności aplikacji webowych, która pozwala atakującemu na wykonanie nieautoryzowanych akcji w imieniu zalogowanego użytkownika. Podatność ta wykorzystuje fakt, że przeglądarka automatycznie dołącza ciasteczka sesyjne do żądań wysyłanych do danej witryny, nawet jeśli żądanie pochodzi z innej strony.

OWASP

> Cross-Site Request Forgery (CSRF) to atak, który zmusza użytkownika końcowego do wykonania niepożądanych działań w aplikacji internetowej, w której jest on aktualnie uwierzytelniony. Z niewielką pomocą socjotechniki (np. wysyłając link za pośrednictwem poczty e-mail lub czatu), atakujący może nakłonić użytkowników aplikacji internetowej do wykonania działań wybranych przez atakującego. Jeśli ofiara jest zwykłym użytkownikiem, udany atak CSRF może zmusić użytkownika do wykonania żądań zmiany stanu, takich jak przelew środków, zmiana adresu e-mail itp. Jeśli ofiarą jest konto administracyjne, CSRF może narazić na szwank całą aplikację internetową.

https://owasp.org/www-community/attacks/csrf

CWE

> Gdy serwer WWW jest zaprojektowany do odbierania żądań od klienta bez żadnego mechanizmu weryfikacji, czy zostały one wysłane celowo, atakujący może oszukać klienta, aby wykonał niezamierzone żądanie do serwera WWW, które będzie traktowane jako autentyczne żądanie. Można to zrobić za pomocą adresu URL, ładowania obrazu, XMLHttpRequest itp. i może to spowodować ujawnienie danych lub niezamierzone wykonanie kodu.

https://cwe.mitre.org/data/definitions/352.html

Mechanizm działania

Mechanizm działania ataku CSRF opiera się na kilku kluczowych elementach. Po pierwsze, użytkownik **musi być zalogowany** do atakowanej aplikacji, a jego sesja musi być aktywna. Następnie, atakujący musi nakłonić ofiarę do **odwiedzenia złośliwej strony lub kliknięcia w specjalnie spreparowany link**. Strona atakującego zawiera kod, który **automatycznie wysyła żądanie** do atakowanej aplikacji. Ponieważ przeglądarka dołącza ciasteczka sesyjne do tego żądania, aplikacja traktuje je jako legalne żądanie od zalogowanego użytkownika.

Szczególnie narażone na ataki CSRF są aplikacje, które:

- Nie implementują żadnych mechanizmów ochrony przed CSRF
- Używają tylko autoryzacji opartej na ciasteczkach sesyjnych
- Wykonują ważne operacje poprzez żądania GET
- Nie weryfikują pochodzenia żądań

W kontekście bezpieczeństwa aplikacji webowych, CSRF jest klasyfikowany jako jedna z dziesięciu najpoważniejszych podatności według OWASP (Open Web Application Security Project).

Wektory ataku

Atakujący ma kilka głównych metod nakłonienia ofiary do wejścia w interakcję ze złośliwym kodem:

1. Poprzez phishing emailowy - wysyłając wiadomości zawierające:
    - Linki do złośliwych stron wyglądające jak legalne URL-e
    - Przyciski "call-to-action" zachęcające do kliknięcia
    - Treść wzbudzającą ciekawość lub poczucie pilności
2. Poprzez złośliwe strony internetowe, które mogą zawierać:
    - Ukryty formularz HTML z predefiniowanymi polami
    - Kod JavaScript wykonujący się automatycznie po załadowaniu strony
    - Obrazy ze spreparowanymi atrybutami `src` wykonującymi żądania

Przykładowy kod złośliwej strony może wyglądać następująco:

```
<form action="https://bank.example.com/transfer" method="POST" id="csrf-form">
    <input type="hidden" name="amount" value="1000">
    <input type="hidden" name="accountTo" value="attacker-account">
</form>
<script>
    document.getElementById("csrf-form").submit();
</script>
```

```
<img src="https://bank.example.com/transfer?amount=1000&to=attacker" style="display:none">
```

W bardziej zaawansowanych scenariuszach, atakujący może użyć:

- `XMLHttpRequest` lub Fetch API do wykonania żądań
- WebSocket do nawiązania połączenia
- Ramek `iframe` do ukrycia złośliwej zawartości

Co istotne, żądanie może być wysłane na różne sposoby:

- GET: poprzez tagi img, script, link
- POST: poprzez automatyczne wysłanie formularza
- Inne metody HTTP: wykorzystując JavaScript i AJAX

Kluczowe elementy skutecznego ataku to:

- Poprawne określenie endpointu atakowanej aplikacji
- Właściwe przygotowanie parametrów żądania
- Ukrycie złośliwego kodu przed użytkownikiem
- Zapewnienie automatycznego wykonania żądania

Scenariusze ataku

Forum i administrator

1. Komentujący dodaje komentarz:

```
<img src="http://forum.test.org/admin/addUser?login=support&pass=123456&type=adm">
```

2. Administrator loguje się na stronie i wchodzi na forum (uwierzytelnienie)
3. Do przeglądarki administratora ładuje się kod HTML z przesłanym wcześniej tagiem `<img>` - podczas próby pobrania obrazka przeglądarka realizuje żądanie HTTP do panelu administratora i w konsekwencji powoduje dodania nowego konta.

Cechy ataku

- nie wykorzystano JavaScript'u
- nie została wykorzystana podatność XSS (choć można by ją było wykorzystać)
- atakujący nie zna loginu i hasła administratora
- logi systemowe nie zawierają numeru IP atakującego - atak przeprowadzono z komputera ofiary - administratora
- atakujący nie widzi odpowiedzi serwera, co nie ma wpływu na skuteczność ataku
- atak odbywa się w ramach jednej domeny (OSRF - One-Site Request Forgery)

Aplikacja bankowa

Mamy aplikację bankową "SecureBank", która umożliwia wykonywanie przelewów. Użytkownik po zalogowaniu może wykonać przelew poprzez formularz dostępny pod adresem: https://securebank.com/transfer

Formularz przelewu w aplikacji wygląda następująco:

```
<form action="/transfer" method="POST">
    <input type="text" name="recipient" placeholder="Numer konta odbiorcy">
    <input type="number" name="amount" placeholder="Kwota">
    <input type="text" name="title" placeholder="Tytuł przelewu">
    <button type="submit">Wykonaj przelew</button>
</form>
```

1. Atakujący przygotowuje złośliwą stronę (some-site.com):

```
<html>
  <body>
    <h1>Wygraj iPhone 16 Pro!</h1>
    <img src="iphone.jpg" style="width:300px"/>
    <form id="csrf-form" action="https://securebank.com/transfer" method="POST"
                         style="display:none">
        <input type="hidden" name="recipient" value="12345678">
        <input type="hidden" name="amount" value="1000">
        <input type="hidden" name="title" value="Zwrot nadpłaty">
    </form>
    <script>
        document.getElementById("csrf-form").submit();
    </script>
  </body>
</html>
```

2. Ofiara jest zalogowana do aplikacji SecureBank w swojej przeglądarce (ma ważną sesję)
3. Atakujący wysyła ofierze email z linkiem do złośliwej strony:

```
Temat: Gratulacje! Wygrałeś iPhone 16 Pro!
Treść: Kliknij tutaj, aby odebrać swoją nagrodę: http://some-site.com/winners
```

4. Ofiara klika w link i otwiera złośliwą stronę

W tle, bez wiedzy ofiary:

- Przeglądarka ładuje stronę some-site.com
- JavaScript automatycznie wysyła formularz
- Przeglądarka dołącza ciasteczka sesyjne do żądania
- SecureBank odbiera żądanie jako legitymowane
- Przelew zostaje wykonany

Zaawansowana wersja ataku

```
<html>
<body>
    <h1>Wygraj iPhone 16 Pro!</h1>
    <script>
        function performCSRF() {
            // Utworzenie żądania XHR
            const xhr = new XMLHttpRequest();
            xhr.open('POST', 'https://securebank.com/transfer', true);
            xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

            // Przygotowanie danych
            const data = new URLSearchParams({
                'recipient': '12345678',
                'amount': '1000',
                'title': 'Zwrot nadpłaty'
            });

            // Wysłanie żądania
            xhr.withCredentials = true; // Ważne - wymusza wysłanie ciasteczek
            xhr.send(data.toString());
        }

        // Wykonaj atak po załadowaniu strony
        window.onload = performCSRF;
    </script>
</body>
</html>
```

Opis

Mechanizm:

- Żądanie jest realizowane programowo za pomocą `XMLHttpRequest`.
- Dane są konstruowane dynamicznie w JavaScript, a następnie wysyłane w żądaniu `POST`.

Działanie:

- Funkcja `performCSRF` tworzy żądanie AJAX, ustawia nagłówki i wysyła dane na serwer.
- Użycie `xhr.withCredentials = true` wymusza dołączenie ciasteczek użytkownika (np. sesyjnych) do żądania, co pozwala serwerowi rozpoznać użytkownika.

Zalety dla atakującego:

- Możliwość dynamicznego tworzenia i manipulacji danymi żądania.
- Może działać z bardziej złożonymi formatami danych (np. JSON).
- Może być używany do interakcji z odpowiedzią serwera (jeśli jest taka potrzeba).
- Większa elastyczność — można wysyłać żądania na inne porty, dodawać niestandardowe nagłówki itp.

Ograniczenia:

- Wymaga znajomości JavaScriptu i technik AJAX.
- Może być blokowany przez niektóre mechanizmy ochrony (np. CSP, SameSite).

Metody ochrony przed CSRF

1. **Użycie tokenów CSRF (CSRF Tokens)**
    - Należy wygenerować unikalny token dla każdej sesji użytkownika.
    - Token powinien być dołączany do formularzy i przesyłany jako część żądania POST/PUT/DELETE.
    - Na serwerze konieczna jest weryfikacja, czy token jest poprawny i zgodny z sesją użytkownika.
    - W przypadku front-endu korzystającego z JavaScript (np. SPA), token CSRF można przesyłać w nagłówkach żądań AJAX.

2. **Weryfikacja nagłówka Referer lub Origin**
    - Sprawdzanie, czy nagłówek Referer lub Origin pochodzi z zaufanego źródła (np. Twojej domeny).
    - Dla dodatkowego bezpieczeństwa, należy korzystać z Origin, ponieważ Referer może być usuwany przez przeglądarki w niektórych sytuacjach.

3. **Używanie nagłówka SameSite dla ciasteczek**
    - Należy ustawić flagę SameSite na ciasteczkach sesyjnych:
    - `SameSite=Lax`: Ogranicza wysyłanie ciasteczek do żądań GET z innych stron.
    - `SameSite=Strict`: Wysyła ciasteczka tylko w obrębie tej samej domeny.
    - Przykład ustawienia nagłówka:

```
Set-Cookie: sessionId=abc123; SameSite=Strict; Secure; HttpOnly;
```

4. **Użycie uwierzytelniania opartego na tokenach**
    - Zamiast ciasteczek sesyjnych, należy stosować stosuj tokeny uwierzytelniające przesyłane w nagłówkach (np. JWT).
    - Token powinien być przechowywany w bezpiecznych miejscach, takich jak HttpOnly cookies.
5. **Ograniczenie metod HTTP**
    - Należy korzystać tylko z bezpiecznych metod HTTP w miejscach, gdzie to możliwe:
        - GET do pobierania danych.
        - POST, PUT, DELETE z dodatkowymi mechanizmami weryfikacji, takimi jak tokeny CSRF.

6. **Sprawdzanie tożsamości użytkownika**
    - Przed wykonaniem krytycznych operacji (np. zmiana hasła, wykonanie przelewu) powinno się wymagać ponownego uwierzytelnienia użytkownika.
7. **Wymuszenie specyficznych nagłówków w żądaniach AJAX**
    - Należy wymagać, aby każde żądanie zawierało niestandardowy nagłówek, np.:

      ```
      X-Requested-With: XMLHttpRequest
      ```

    - Na serwerze weryfikuj obecność tego nagłówka.
8. **Implementacja Content Security Policy (CSP)**
    - CSP nie chroni bezpośrednio przed CSRF, ale ogranicza możliwość wykonania złośliwego kodu na stronie, który mógłby pomóc w realizacji ataku.

9. **Segmentacja uprawnień**
    - Pomocne jest ograniczenie dostępu do krytycznych funkcji tylko do zaufanych użytkowników.
    - Niezbędna jest weryfikacja uprawnienia użytkownika przy każdej operacji na serwerze.
10. **Regularne testy bezpieczeństwa**
    - Testowanie aplikacji pod kątem podatności na CSRF, wykorzystując narzędzia takie jak OWASP ZAP czy Burp Suite.
    - Wdrożenie automatycznych testów bezpieczeństwa (np. CI/CD) pomoże wychwycić luki na wczesnym etapie.