CSRF Token Injection in AJAX

https://icarocomix.github.io/appuntidiprogrammazione

CSRF Token Injection in AJAX

ANALISI TECNICA

PROBLEMA

Le richieste asincrone falliscono con errore 403 perché non includono il segreto crittografico richiesto da Spring Security.

PERCHÉ

Esposizione sicura del token. Ho scelto di usare i meta-tag per centralizzare il segreto CSRF, rendendolo accessibile in modo trasparente a tutti i moduli JavaScript della pagina.

ESEMPIO IMPLEMENTATIVO - PART 1

```
<head>
    <meta name="_csrf" th:content="${_csrf.token}">
        <meta name="_csrf_header"
            th:content="${_csrf.headerName}">
            <!-- Oppure uso th:attr per sicurezza se il nome
                dell'attributo è dinamico: -->
            <meta
                th:attr="name=${_csrf.parameterName},content=${_c
                srf.token}">
            </head>
            /* Modulo JavaScript centralizzato (csrf.js): leggo i
                 meta-tag una sola volta e li rendo disponibili a
                 tutti i moduli fetch della pagina. */
            const CsrfUtils = (() => {
                // Leggo i valori dal DOM una sola volta al
                    caricamento della pagina
                const tokenElement =
                    document.querySelector('meta[name="_csrf"]');

    const headerElement = document.querySele
                    ctor('meta[name="_csrf_header"]');


                    if (!tokenElement || !headerElement) {

        console.error('Meta-tag CSRF non
                    trovati: le richieste POST falliranno con
                    403.');
    }
```

ESEMPIO IMPLEMENTATIVO - PART 2

```
const token =
                tokenElement?.content;
const headerName =
                 headerElement?.content;
            /* Wrapper fetch con CSRF automatico:
                sostituzione drop-in di window.fetch per
                tutte le richieste mutanti. */
            function secureFetch(url, options = {}) {

    const method = (options.method ||
                'GET').toUpperCase();
            // GET e HEAD non necessitano di CSRF
                (idempotenti per design)
            if (['GET', 'HEAD', 'OPTIONS'].includes(method))
                {
        return fetch(url, options);

    }
            // Aggiungo il token CSRF a tutte le richieste
                mutanti
            const headers = new Headers(options.headers ||
                {});
    headers.set(headerName, token);


    return fetch(url, {
                ...options, headers });
}
            /* Funzione helper per form AJAX: serializza il
                FormData e aggiunge il CSRF. */
```

ESEMPIO IMPLEMENTATIVO - PART 3

```
            function submitForm(form) {
        const
                formData = new FormData(form);
            // Spring Security accetta il token anche come
                parametro di form
            formData.append(document.querySelector('meta[name
                ="_csrf"]')?.getAttribute('name') || '_csrf',
                 token);

        return
                fetch(form.action, { method: form.method ||
                'POST', body: formData
            // NON imposto Content-Type: il browser lo fa
                automaticamente con il boundary multipart });
        }

    return {
        token,

            headerName,
        secureFetch,

            submitForm
    };
})();
        /* Esempio d'uso: */
        async function deleteProduct(productId) {
    const
            response = await CsrfUtils.secureFetch(`/api/prod
            ucts/${productId}`, { method: 'DELETE' });
```

ESEMPIO IMPLEMENTATIVO - PART 4

```
    if (response.ok) {
        document.get
                ElementById(`product-${productId}`).remove();

    } else if (response.status === 403) {

        console.error('CSRF token non valido o
                scaduto: ricarica la pagina.');
    }
}
            /* Per HTMX, configuro il token CSRF globalemente: */
            document.addEventListener('htmx:configRequest',
                (event) => {
                // HTMX legge i meta-tag automaticamente se
                    configurato correttamente
                event.detail.headers[CsrfUtils.headerName] =
                    CsrfUtils.token;
});
                /* Configurazione Spring Security per accettare
                    il token sia come header che come parametro:
                    */
                @Configuration
public class SecurityConfig {

                    @Bean
    public SecurityFilterChain
                    filterChain(HttpSecurity http) throws
                    Exception {
```

ESEMPIO IMPLEMENTATIVO - PART 5

```
        http

                    .csrf(csrf -> csrf
                .csrfToken
                    Repository(CookieCsrfTokenRepository.withHttp
                    OnlyFalse())
                // Permette la lettura del token via JS per SPA
                .csrfTokenRequestHandler(new
                    CsrfTokenRequestAttributeHandler())

            )
                // ... resto della configurazione
                return http.build();
    }
}
```

CONSOLE DI DEBUG

```
> Stato post: [utile]

> Azione: salva_post_ora()


> Requisito:
  Click sull'icona Segnalibro
```