OAuth & OpenID Connect Lab

Owner: Amo Samuel
Reviewer: Thomas Darko

This report provides an overview of key security features configured in my Spring Boot application, including Cross-Site Request Forgery (CSRF) protection, OAuth 2.0 authentication, OpenID Connect (OIDC), and session management.

What is CSRF

Cross-Site Request Forgery (CSRF) is an attack where a malicious website tricks a user's browser into performing actions on a different site where the user is authenticated. CSRF attacks exploit the trust a site has in the user's browser.

CSRF Configuration

In the provided Spring Security configuration, CSRF protection is enabled with the following settings:

- **CSRF Token Repository**: CookieCsrfTokenRepository.withHttpOnlyFalse()

Code snippet

```
@Bean  samuelamo001 *
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http
            .csrf(csrf-> csrf
                    .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
            )
```

OAuth 2.0 Authentication

OAuth 2.0 is an authorization framework that allows third-party applications to obtain limited access to user accounts on an HTTP service. It is widely used for enabling single sign-on (SSO) and authorization.

OAuth 2.0 Configuration

The configuration specifies the use of OAuth 2.0 for login, particularly with Google as the identity provider.

- **Login Page**: /login
- **Success Handler**: Redirects to /profile after a successful login.

Code snippet

```
)

.oauth2Login(oauth2Login -> oauth2Login
            .loginPage("/login")
            .successHandler((request, response, authentication) -> response.sendRedirect( location: "/profile"))
)
```

OpenID Connect (OIDC)

OpenID Connect (OIDC) is an identity layer on top of OAuth 2.0 that provides authentication. It allows clients to verify the identity of users based on the authentication performed by an Authorization Server.

OIDC Integration

In this configuration, OAuth 2.0 is used for authentication, which implies the application can use OpenID Connect for identity management with a provider such as Google.

Session Management

Session management involves handling user sessions to ensure secure and consistent interactions. It includes creating, maintaining, and invalidating sessions.

Session Management Configuration

The provided configuration sets policies for session management as follows:

- **Session Creation Policy**: SessionCreationPolicy.ALWAYS

  Sessions are always created for requests.

- **Session Fixation Protection**: migrateSession()

  Ensures that the session ID changes after authentication to prevent session fixation attacks.

- **Maximum Sessions**: 1

  Limits the number of concurrent sessions per user.

- **Maximum Sessions Prevents Login**: true

  Prevents new logins if the maximum number of sessions is reached.

Code snippet

```
)

.sessionManagement(sessionManagement -> sessionManagement
        .sessionCreationPolicy(SessionCreationPolicy.ALWAYS) SessionManagementConfigurer<HttpSecurity>
        .sessionFixation().migrateSession()
        .maximumSessions(1) ConcurrencyControlConfigurer
        .maxSessionsPreventsLogin(true)

)
```

Conclusion

This security configuration demonstrates a comprehensive approach to managing security in a Spring Boot application, covering CSRF protection, OAuth 2.0 authentication, OpenID Connect, and session management.