Spring Security

Maksymilian Świętoń
Tomasz Seruga
Witold Drożdżowski

1. Ogólnie o Spring Security, rodzaje zabezpieczeń

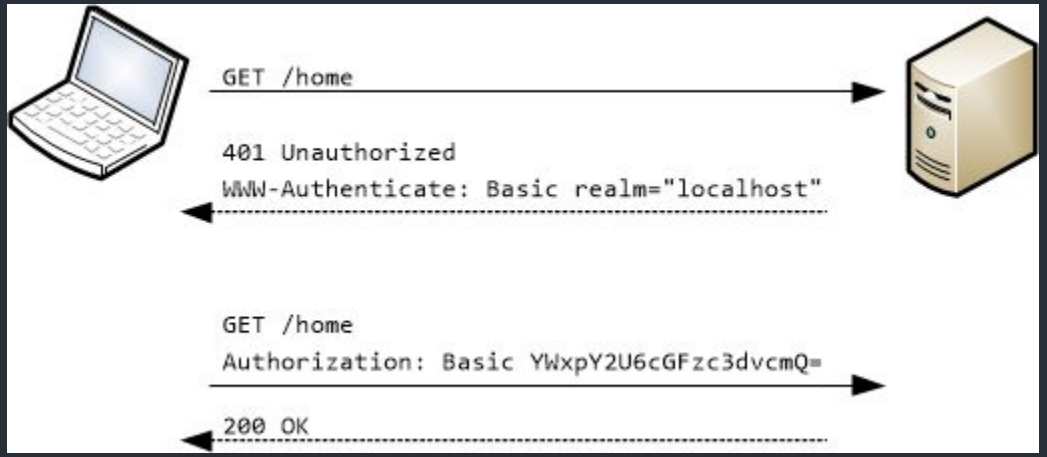
Czym jest Spring Security?

Największe zalety

- Elastyczność
- Wysoka skuteczność
- Społeczność

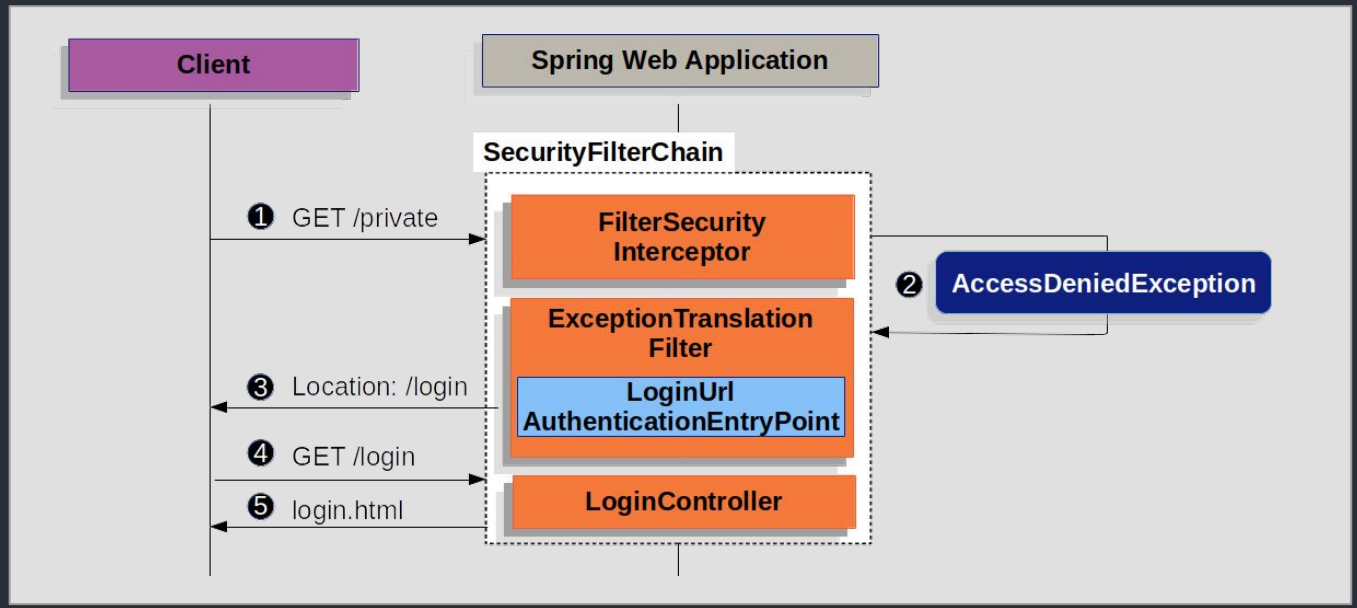
1.1 Mechanizmy zabezpieczeń Spring Security

Basic authorization



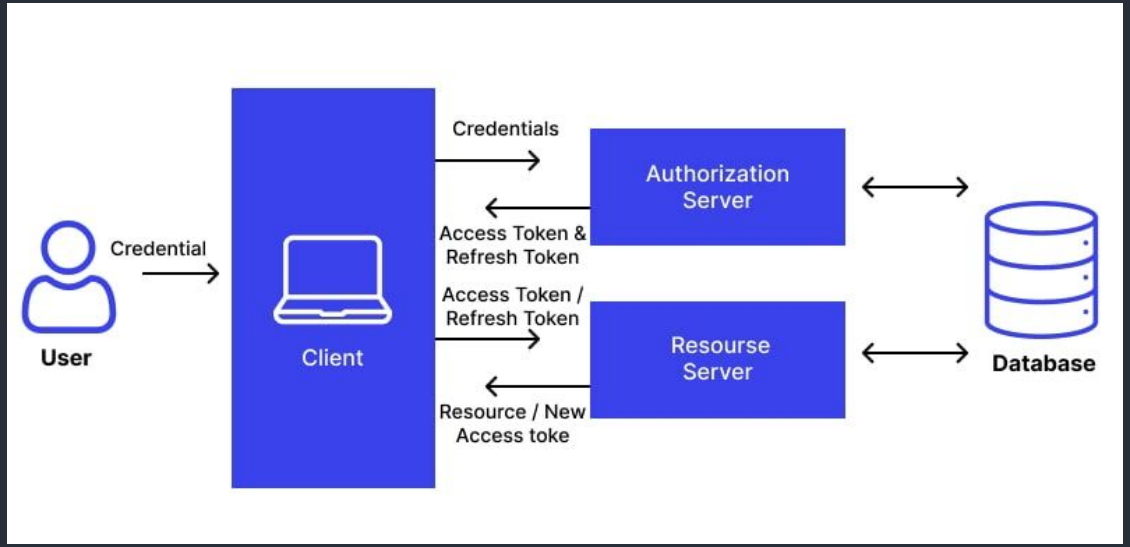
1.1 Mechanizmy zabezpieczeń Spring Security

Form login



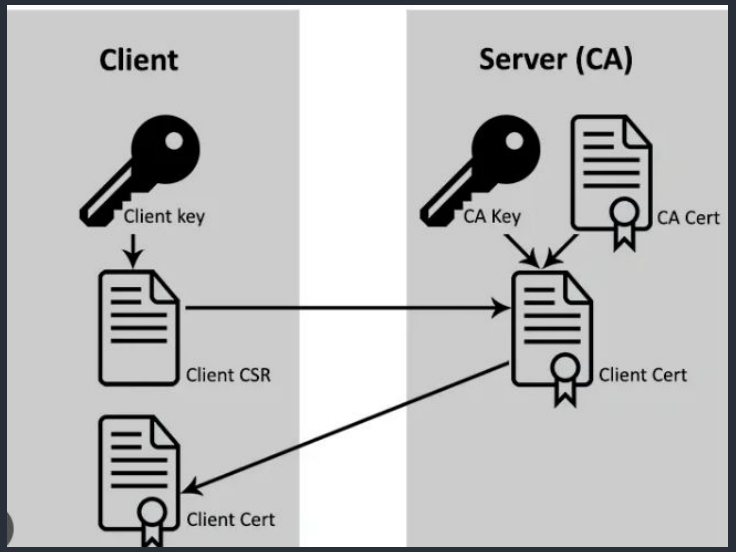
1.1 Mechanizmy zabezpieczeń Spring Security

Token-based Authentication



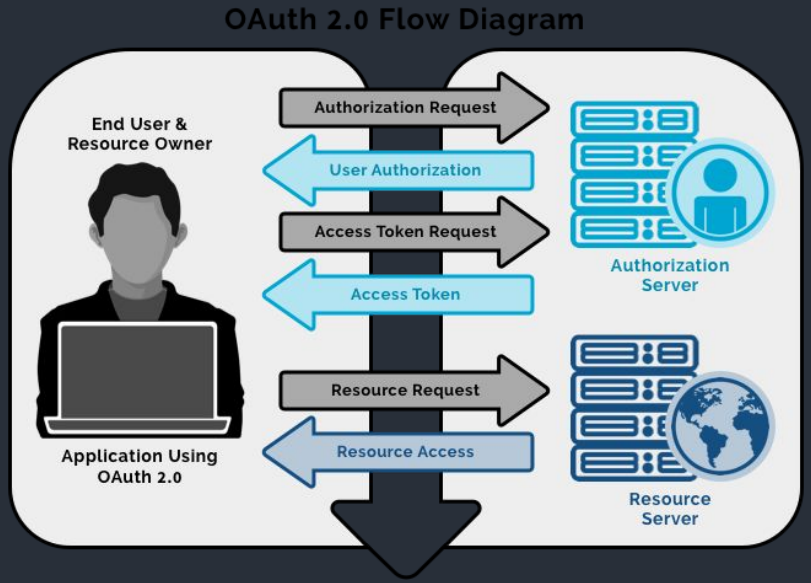
1.1 Mechanizmy zabezpieczeń Spring Security

Certificate Authentication



1.1 Mechanizmy zabezpieczeń Spring Security

OAuth 2



2.1 Specyfikacja działania

1. Umożliwia kontrolę dostępu do zasobów aplikacji oraz uwierzytelnianie użytkowników
2. Udostępnia różne metody zabezpieczeń
3. Posiada wiele przydatnych filtrów
4. Jest skonfigurowany przy pomocy klas Java
5. Jest realizowana poprzez klasę konfiguracji, która rozszerza klasę WebSecurityConfigurerAdapter
6. Dostarcza mechanizmy pozwalające na zwiększenie bezpieczeństwa aplikacji

2.1 Konfiguracja Spring Security

1. Konfiguracja Maven
2. Zabezpieczenia sieci z konfiguracją Java
3. Zabezpieczenia HTTP
4. Formularz logowania
5. Autoryzacja z rolami
6. Wylogowanie się
7. Uwierzytelnianie

1. Konfiguracja Maven

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>5.3.3.RELEASE</version>
</dependency>
```

2. Zabezpieczenia sieci z konfiguracją Java

```
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth)
      throws Exception {
        auth.inMemoryAuthentication().withUser("user")
          .password(passwordEncoder().encode("password")).roles("USER");
    }
}
```

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

3. Zabezpieczenia HTTP

```
@Bean
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.authorizeRequests()
      .anyRequest().authenticated()
      .and().httpBasic();
    return http.build();
}
```

```
<http>
    <intercept-url pattern="/**" access="isAuthenticated()"/>
    <form-login />
    <http-basic />
</http>
```

4. Formularz logowania

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.authorizeRequests()
      .anyRequest().authenticated()
      .and().formLogin()
      .loginPage("/login").permitAll();
    return http.build();
}
```

5. Autoryzacja z rolami

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.authorizeRequests()
      .antMatchers("/", "/home").access("hasRole('USER')")
      .antMatchers("/admin/**").hasRole("ADMIN")
      .and()
      // some more method calls
      .formLogin();
    return http.build();
}
```

6. Wylogowanie się

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.logout();
    return http.build();
}
```

7. Uwierzytelnianie

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth)
  throws Exception {
    auth.inMemoryAuthentication()
      .withUser("user").password(passwordEncoder().encode("password")).roles("USER")
      .and()
      .withUser("admin").password(passwordEncoder().encode("password")).roles("USER", "ADMIN");
}
```

7.1. Uwierzytelnianie JDBC (Java DataBase Conectivity)

```
@Autowired
private DataSource dataSource;

@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth)
  throws Exception {
    auth.jdbcAuthentication().dataSource(dataSource)
      .withDefaultSchema()
      .withUser("user").password(passwordEncoder().encode("password")).roles("USER")
      .and()
      .withUser("admin").password(passwordEncoder().encode("password")).roles("USER", "ADMIN");
}
```

3. Zabezpieczanie aplikacji REST

Jak możemy zabezpieczyć swoją aplikację REST ?

1. Implementacja uwierzytelnienia
2. Użycie protokołu TLS
3. Sprawdzenie parametrów interfejsów
4. Ograniczenie dostępu do zasobów
5. Użycie paginacji

3. Zabezpieczanie aplikacji REST

1. Implementacja uwierzytelnienia

Zawsze powinno się wiedzieć kto korzysta z aplikacji dlatego ważne jest użycie uwierzytelniania na przykład dzięki standardowi OAuth 2.0 który wykorzystuje token dostępu.

3. Zabezpieczanie aplikacji REST

2. Użycie protokołu TLS

Parametry uwierzytelniania mogą zostać naruszone podczas komunikacji serwer-klient. Dlatego ważne jest, aby używać TLS czyli Transport Layer Security które chroni informacje poprzez ich szyfrowanie.

3. Zabezpieczanie aplikacji REST

3. Sprawdzenie parametrów interfejsów

Parametry interfejsów muszą zostać zweryfikowane.
Można to zrobić ustanawiając schemat dla przychodzących parametrów i sprawdzając poprawność parametrów względem schematu.

3. Zabezpieczanie aplikacji REST

4. Ograniczenie dostępu do zasobów

Duża liczba żądań w krótkim czasie jest kosztowna obliczeniowo i może sprawić że interfejs przestanie odpowiadać. Dlatego ważne jest żeby ograniczyć liczbę żądań które może skierować klient w określonym przedziale czasowym.

3. Zabezpieczanie aplikacji REST

5. Użycie paginacji

Jeżeli aplikacja zwraca duże ilości danych, możemy wykorzystać paginację. Pozwala ona klientowi ograniczyć liczbę zasobów, do których może uzyskać dostęp w jednym żądaniu.