Wzorce bezpieczeństwa

Zaawansowane Techniki Programowania

Politechnika Krakowska, 2022

Wzorce bezpieczeństwa

- Koncept bezpieczeństwa (Wzorzec Single-sign-on)
- Mechanizmy uwierzytelniania
- Interceptory uwierzytelniania

Koncept bezpieczeństwa

Bezpieczeństwo jest ważne.

Dane

- **Poufność** - dane powinny być dostępne tylko dla uprawnionych do tego użytkowników
- **Integralność** - dane nie powinny być modyfikowane przez nieuprawnionych do tego użytkowników

- **Dostępność** - dane powinny być dostępne, gdy jest to potrzebne
- **Niezaprzeczalność** - użytkownicy nie mogą odrzucać ani zaprzeczyć relacji przy użyciu danych ani żadnych innych procesów

Po co nam wzorce bezpieczeństwa?

Aby aplikacje, których używamy były bezpieczne, powinny zostać spełnione podstawowe zasady bezpieczeństwa.

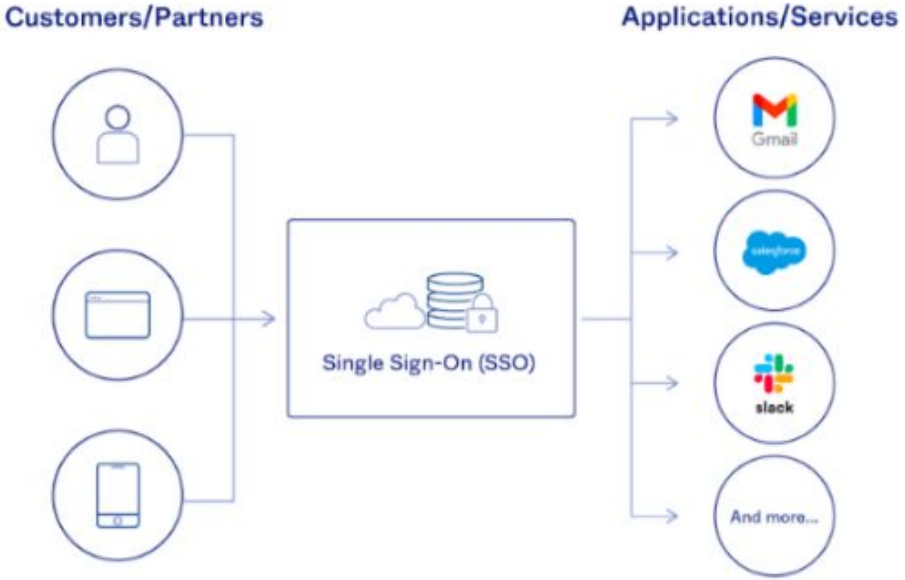
Co zrobić, żeby spełniać te zasady?

Wzorce bezpieczeństwa możemy traktować jako rozwiązanie dla problemów z bezpieczeństwem, na które są często spotykane, przy użyciu zweryfikowanych rozwiązań.

Koncept single-sign-on

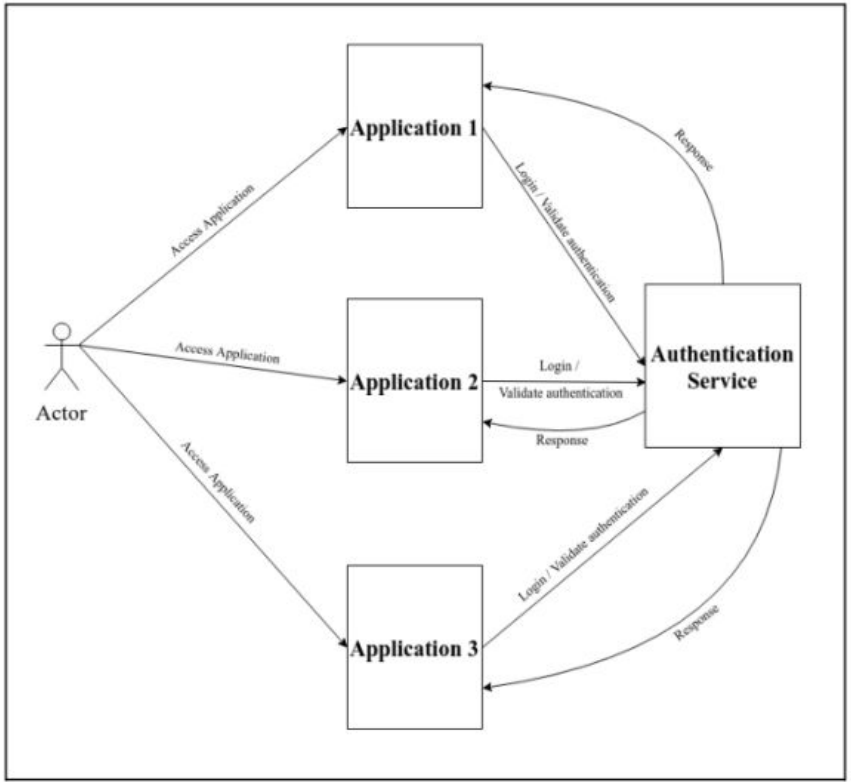
Single-sign-on jest wzorcem bezpieczeństwa, który udostępnia jeden serwis uwierzytelnienia wielu aplikacjom.

Wykorzystanie SSO wiąże się przede wszystkim z ułatwieniem użytkownikowi dostępu do szeregu aplikacji, zachowując przy tym wyższy poziom bezpieczeństwa.



Proces działania

1. Użytkownik wysyła zapytanie do aplikacji.
2. Aplikacja wysyła zapytanie do usługi uwierzytelniającej.
3. Serwis uwierzytelniający zwraca:
   a. Stronę logowania - jeżeli użytkownik nie jest uwierzytelniony
   b. Dostęp do aplikacji - jeżeli jest



Mechanizmy uwierzytelniania użytkowników

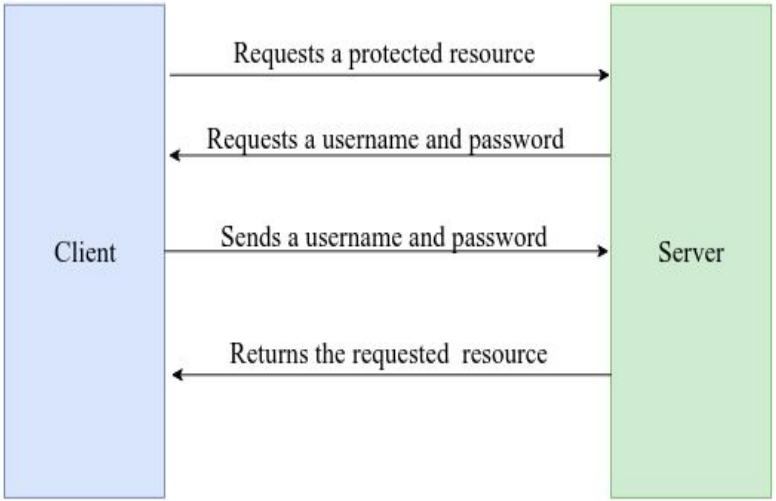
- W środowisku korporacyjnym każda aplikacja lub zasób musi sprawdzać dostęp użytkownika i chronić przed nieautoryzowanym dostępem.

- Mechanizmy uwierzytelniania umożliwiają programiście uwierzytelnianie i weryfikację użytkowników w szybki i łatwy sposób. Aby skorzystać z mechanizmu uwierzytelniania, należy skonfigurować go za pomocą pliku deskryptora wdrożenia (np. za pomocą adnotacji).

Uwierzytelnianie podstawowe	Uwierzytelnianie oparte na formularzach	Uwierzytelnianie szyfrowane	Uwierzytelnianie klienta	Wzajemne uwierzytelnianie

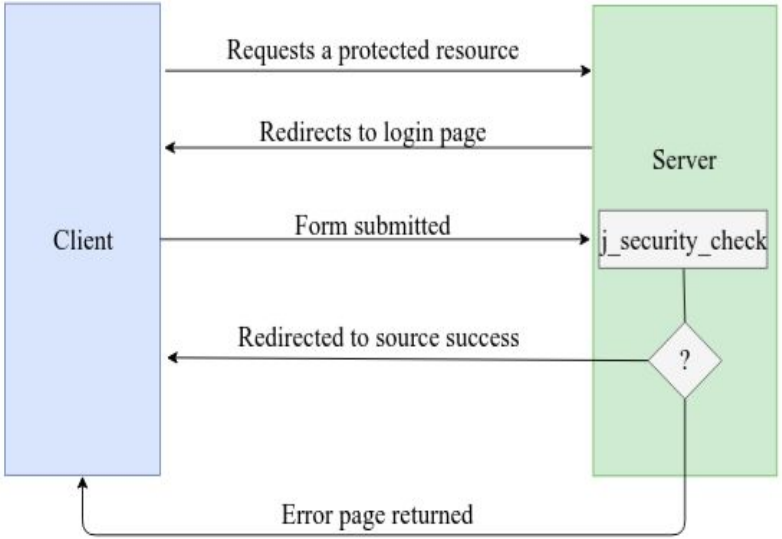
Uwierzytelnianie podstawowe

- Uwierzytelnianie podstawowe jest mechanizmem domyślnym.
- Jeśli użytkownik nie jest uwierzytelniany podczas wysyłania żądania, zwracane jest okno dialogowe z żądaniem podania nazwy użytkownika i hasła.



Uwierzytelnianie oparte na formularzach

- Mechanizm wykorzystuje formularze do żądania nazwy użytkownika i hasła, umożliwiając programiście dostosowanie ekranu logowania i błędu.



Uwierzytelnianie szyfrowane

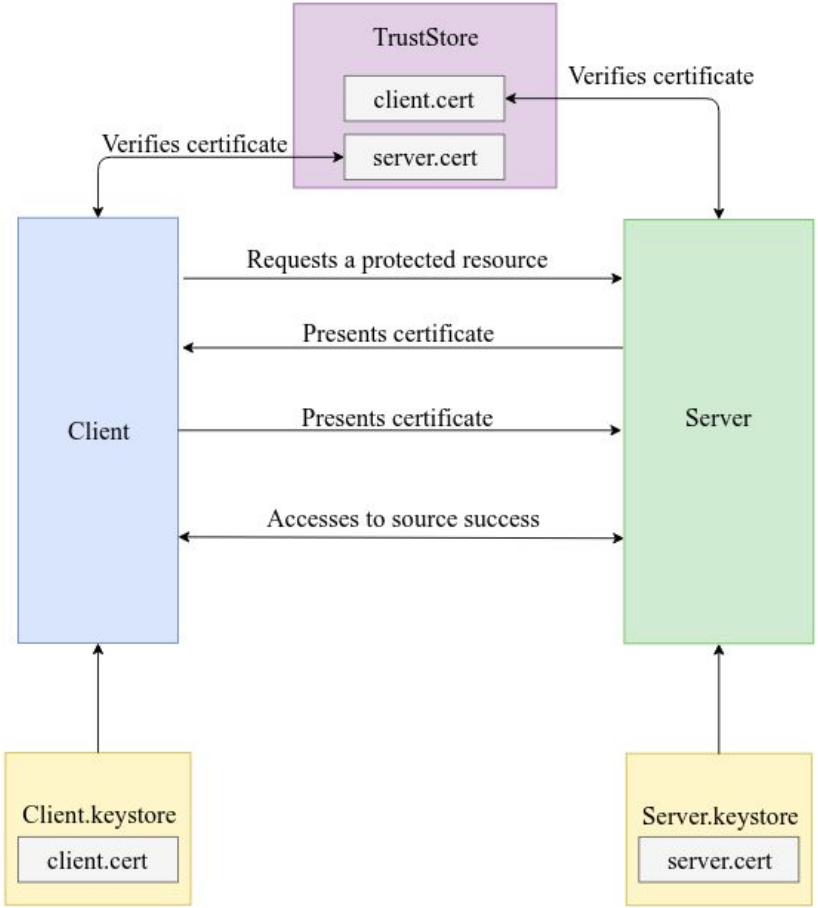
- Mechanizm wykorzystuje jednokierunkowy skrót kryptograficzny hasła i dodatkowych danych.
- Gdy użytkownik wysyła skrót do serwera, uwierzytelnianie skrótu wymaga hasła w postaci zwykłego tekstu, aby uzyskać dostęp do skrótu i zweryfikować dostęp.

Uwierzytelnianie klienta

- Mechanizm uwierzytelniania klienta przy użyciu jego certyfikatu klucza publicznego.
- Ten mechanizm jest uważany za bezpieczniejszy, ponieważ używa protokołu HTTP przez SSL (HTTPS) oraz certyfikatu klucza publicznego klienta.

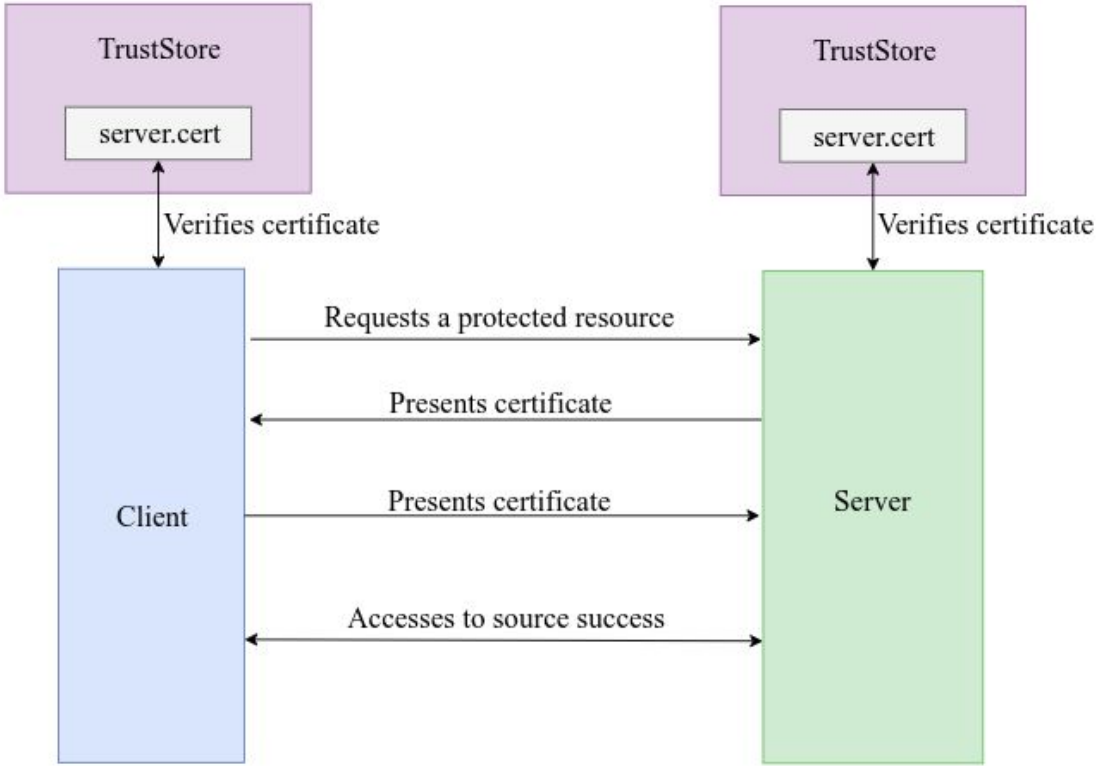
Wzajemne uwierzytelnianie - typ 1

- Weryfikacja certyfikatu serwera przez klienta
- Weryfikacja certyfikatu klienta przez serwer



Wzajemne uwierzytelnianie - typ 2

- Weryfikacja certyfikatu serwera przez klienta
- Weryfikacja wysłanych danych logowania klienta przez serwer



Sposoby wdrożenia mechanizmu uwierzytelniania

- Deskryptor wdrożenia
  - plik konfiguracyjny ze wspólną konfiguracją dla zbiorów zasobów

- Adnotacja
  - zdefiniowanie odmiennej konfiguracji dla określonych zasobów

- Konfiguracja programowa
  - dynamiczne określenie zasad uwierzytelniania

Elementy pliku deskryptora

Określenie chronionych zasobów:

```
<web-resource-collection>
        <web-resource-name>helloworld</web-resource-name>
        <url-pattern>/resources/helloworld/*</url-pattern>
</web-resource-collection>
```

Określenie ról bezpieczeństwa:

```
<auth-constraint>
        <role-name>user</role-name>
</auth-constraint>
```

Określenie mechanizmu autentyfikacji:

```
<login-config>
        <auth-method>BASIC</auth-method>
</login-config>
```

Interceptory uwierzytelniania

- Wzorzec przechwytujący jest jednym z technik autoryzacji występującej w Java EE 8.
- Funkcjonuje jak programowanie zorientowane na aspekt (AOP).
- Aby zaimplementować go, możemy użyć interceptora EJB lub interceptora CDI.
- Dobrym rozwiązaniem jest użycie przechwytywacza EJB z logiką biznesową w klasie EJB, ale lepiej jest używać przechwytywacza CDI wraz z klasą CDI w warstwie webowej.

Implementacja wzorca Authentication Interceptor

- Aplikacja, która przechwytuje request i zwraca “hello world”.
- “Hello world” jest chronionym zasobem i użytkownik musi przejść autentykacje, aby się do niego dostać.
- CDI Interceptor przechwytuje próbę dostępu do zasobu i bada dane podane przez użytkownika.
- Na potrzeby przykładu, źródło danych nie znajduje się w bazie danych, lecz w jednej z klas.

1.Annotation

```java
@Inherited
@InterceptorBinding
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD, ElementType.TYPE})
public @interface Authentication {

    2 usages
    @Nonbinding String[] roles() default {};
```

- Kwalifikator, który mówi, że metoda ma zostać przechwycona.
- Parametr roles określa, którzy użytkownicy mają dostęp do zasobu.

2. Interceptor

```java
@Authentication
@Interceptor
public class AuthenticationInterceptor implements Serializable{

    2 usages
    @Inject
    private Auth auth;

    @AroundInvoke
    public Object authentication(InvocationContext context) throws IOException {

        HttpServletRequest request = getHttpServletRequest( context );
        HttpServletResponse response = getHttpServletResponse( context );

        String[] credentials = AuthUtils.readBasicAuthHeader( request );

        String login;
        String password;
```

- Klasa Authentication Interceptor jest używana jako Interceptor.
- Służy ona do przechwytywania wszystkich wywołań metod oznaczonych kwalifikatorem @Authentication.
- Wywołuje metodę oznaczoną kwalifikatorem @AroundInvoke.
- Metoda ta potrzebuje parametrów dotyczących żądań HTTP.

3. Implementacja interfejsu do autoryzacji

```java
@Stateless
public class AuthImpl implements Auth {


    @Inject
    private DataSource dataSource;


    @Override
    public Boolean isAuthorized(String login,String role) {
        return dataSource.readUserRoles( login ).contains( role );
    }


    @Override
    public Boolean isAuthenticated(String login, String password) {
        return dataSource.readUserPassword( login ).contains( password );
    }
}
```

- W tej klasie walidowane są przekazane dane użytkownika.

4. Źródło danych o użytkownikach

```java
@ApplicationScoped
public class DataSource {

    4 usages
    private Map<String, List<String>> roles;
    4 usages
    private Map<String, String> passwords;

    @PostConstruct
    public void init(){
        roles = new HashMap<>();
        roles.put("rhuan", Arrays.asList("user","admin"));
        roles.put("joao", Arrays.asList("user","admin"));

        passwords = new HashMap<>();
        passwords.put("rhuan", "123456");
        passwords.put("joao", "123456");
    }

    1 usage
    public List<String> readUserRoles(String login) { return roles.get( login ); }

    1 usage
    public String readUserPassword(String login) { return passwords.get( login ); }
}
```

- Klasa zawierająca logikę do czytania danych użytkownika.
- Dane przechowywane są w mapach.

5. Ekstrakcja loginu i hasła z żądania

```java
public class AuthUtils {

    1 usage
    public static final int INDEX_LOGIN = 0;
    1 usage
    public static final int INDEX_PASSWORD = 1;

    1 usage
    public static String[] readBasicAuthHeader( HttpServletRequest request ){

        final String authorization = request.getHeader( s: "Authorization");

        if (authorization != null && authorization.startsWith("Basic")) {

            String base64Credentials = authorization.substring("Basic".length()).trim();
            String credentials =
                    new String(
                            Base64.getDecoder().decode(
                                    base64Credentials),
                            Charset.forName( charsetName: "UTF-8"));

            return credentials.split( regex: ":", limit: 2);
        }

        return new String[0];
    }
}
```

- Narzędzie do czytania loginu i hasła zawartego w żądaniu.
- Autentykacja musi być ustawiona na basic.

6. Zabezpieczony zasób

```java
@Path("helloworld")
public class HelloWorld {

    @GET
    @Authentication(roles = {"user"})
    public Response helloWorld(@Context HttpServletRequest request,
                               @Context HttpServletResponse response){

        return Response
                .ok( entity: "Hello World. Welcome to App with validation by authentication " +
                        "interceptor!")
                .build();

    }

}
```

- Klasa do której kierowane są żądania.
- Jej metoda response jest oznaczona kwalifikatorem @Authentication.
- Dzięki niemu jest ona przechwytywana przez Authentication Interceptor.

Agnieszka Rejczak
Edyta Mróz
Małgorzata Zieleń
Kamil Wiśniewski
Mateusz Jamrozik
Szymon Barszcz