УДК 004.021

ЗАЩИТА ДАННЫХ ПОЛЬЗОВАТЕЛЯ НА СТОРОНЕ СЕРВЕРА ПОСРЕДСТВОМ ФРЕЙМВОРКА SPRING SECURITY И ТЕХНОЛОГИИ JSON WEB TOKEN

П.Д. ТАЛАЙКО
(Представлено: канд. тех. наук, доц. И. Б. БУРАЧЁНОК)

Рассмотрены принципы работы фреймворка Spring Security и технологии JSON Web Token. Проведено основное описание реализации механизма авторизации и аутентификации пользователей в веб-приложении. Показаны реализация основных функций механизма защиты.

В современных веб-приложениях зачастую необходимо хранить пользовательскую информацию: настройки системы, личные данные, фильтры поиска и т.д. Поскольку веб-приложения являются многопользовательскими, то целесообразно идентифицировать пользователей посредством логина и пароля.

В представленной статье расссмотрим способ хранения личной информации пользователя с использованием фреймворка Spring Security и технологии JSON Web Token (JWT).

Spring Security – Java/Java EE фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для промышленных приложений, созданных с помощью Spring Framework. Spring Security позволяет сделать разграничение доступа пользователей приложения к ресурсам, предоставляет необходимые для хэширования паролей средства. При этом Spring Security используется вместе с JSON Web Token.

JWT (JSON Web Token) – это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на JSON формате. Как правило, используется для передачи данных авторизации в клиент-серверных приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности. Клиент в дальнейшем использует данный токен для обращения к ресурсам веб-приложения.

Каждый пользователь при входе в систему обязан ввести свой логин и пароль. Пароли хранятся в базе данных в хешированном виде, поэтому даже если злоумышленник каким-то образом получит доступ к таблице пользователей, он не сможет получить данные для входа в систему под именем другого пользователя.

Аутентификация – это процесс проверки подлинности каких-либо данных. Примером аутентификации может быть сравнение пароля, введенного пользователем, с паролем, который сохранен в базе данных сервера.

Пароли в базе данных не хранятся в открытом виде, а хешируется алгоритмом bcrypt, который позволяет сгенерировать разные хеш-значения для одинаковых данных. Если данные пользователя в базе данных совпадают с введенными, то пользователю выдается JWT-токен и процедуру аутентификации можно считать успешной.

Авторизация – предоставление определённому лицу или группе лиц прав на выполнение определённых действий. В разрабатываемом веб-приложении существуют две роли: роль пользователя (ROLE_USER) и роль администратора (ROLE_ADMIN). Соответственно существует необходимость разграничения доступа к возможностям веб-приложения и у пользователя не должно быть прав на выполнения части программного кода, если у него присутствует только роль пользователя. В качестве механизма разграничения ролей используется стандартная конфигурация Spring Security. Также каждый метод классов-контроллеров имеет аннотацию @PreAuthorize, которая позволяет либо допустить пользователя к исполнению программного кода, либо отвергнуть его запрос с ошибкой, если его роль не совпадает с той, что указана в аннотации.

Получение ролей сервером осуществляется из JWT токена, который хранится у пользователя локально в браузере в localStorage. При совершении какого-либо действия клиентское приложение передает JWT токен, в котором хранится информация о его ролях. Если пользователь локально изменил содержание токена, то при прохождении обработки на сервере и сравнения с секретным ключом он будет недействителен, что гарантируется спецификацией JWT.

При генерации JWT происходит установка времени жизни токена, после истечения которого токен становится не валидным и нуждается в обновлении. Построение токена можно приведено в листинге 1.

```
private String doGenerateToken(Map<String, Object> claims, String subject) {
        final Date createdDate = clock.now();
        final Date expirationDate = calculateExpirationDate(createdDate);
        return Jwts.builder()
            .setClaims(claims)
            .setSubject(subject)
            .setIssuedAt(createdDate)
            .setExpiration(expirationDate)
            .signWith(SignatureAlgorithm.HS512, secret)
            .compact();
}
```

Листинг 1. – Функция построения токена

Как можно видеть в функции doGenerateToken из листинга 1, происходит построение JWT String токена, который включает в себя встроенную информацию (claims), имя пользователя (subject), дату создания токена (createDate), дату истечения (expirationDate) и алгоритм шифрования с секретным значением. Как правило, секретным значением выступает текст, прописанный в контексте приложения.

По истечению времени токена необходимо его обновить, для чего будет использоваться функция refreshToken, приведённая в листинге 2.

```
public String refreshToken(String token) {
        final Date createdDate = clock.now();
        final Date expirationDate = calculateExpirationDate(createdDate);

        final Claims claims = getAllClaimsFromToken(token);
        claims.setIssuedAt(createdDate);
        claims.setExpiration(expirationDate);

        return Jwts.builder()
            .setClaims(claims)
            .signWith(SignatureAlgorithm.HS256, secret)
            .compact();
}
```

Листинг 2. – Функция обновления токена

В функцию refreshToken необходимо передать токен, время жизни которого уже истекло, установить с него claims и новую дату в обновлённый токен и вернуть.

Функция для проверки валидности токена приведена в листинге 3.

```
public Boolean validateToken(String token, UserDetails userDetails) {
        JwtUser user = (JwtUser) userDetails;
        final String username = getUsernameFromToken(token);
        final Date created = getIssuedAtDateFromToken(token);
        return (
            username.equals(user.getUsername())
                && !isTokenExpired(token)
                &&                      isCreatedAfteLastPasswordReset(created,
user.getLastPasswordResetDate())
        );
}
```

Листинг 3. – Функция проверки валидности токена

Spring Security необходим для получения данных пользователя UserDetails и фильтров HTTP запросов. Фильтор для проверки авторизации пользователя приведён в листинге 4.

```
     @Component
     public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint,
Serializable {
          private static final long serialVersionUID = -8970718410437077606L;
          @Override
          public void commence(HttpServletRequest request, HttpServletResponse re-
sponse,
                               AuthenticationException authException) throws IOEx-
ception {
               response.sendError(HttpServletResponse.SC_UNAUTHORIZED,      "Unauthor-
ized");
          }
     }
```

Листинг 4. – Фильтр HTTP запроса

Для использования токена пользователю предстоит передать логин и пароль на указанный URL, где вызовется функция doGenerateToken в случае, если пользователь будет найден в базе данных. Функция doGenerateToken вернёт сгенерированный токен, который необходимо будет передавать при обращении к API. Фреймворк Spring Security позаботится о проверке валидности токена и доступа к частям кода благодаря перехватчикам (фильтрам) HTTP запросов, которые проверяют наличие заголовка Autorization.

ЛИТЕРАТУРА

1. Token-Based Authentication. [Электронный ресурс] / Git hub gist. – Режим доступа: https://gist.github.com/zmts/802dc9c3510d79fd40f9dc38a12bccfc. – Дата доступа: 09.09.18.
2. Using JWT with Spring. [Электронный ресурс] / Baeldung. – Режим доступа: https://www.baeldung.com/spring-security-oauth-jwt. – Дата доступа: 09.09.18.
3. Spring Security. [Электронный ресурс] / Spring. – Режим доступа: https://spring.io/projects/spring-security. – Дата доступа: 10.09.18.
4. JWT. [Электронный ресурс] / JWT. – Режим доступа: https://jwt.io / – Дата доступа: 10.09.18.