Refresh Token

Access Token(JWT)를 통한 인증 방식의 문제점

- 제 3자에게 탈취당할 경우 보안에 취약하다.
- 위를 해결하기 위해 유효기간을 짧게 만들면 그만큼 사용자는 로그인을 자주 다시 해서 새로 토큰을 발급받아야 한다.

이를 해결하기 위해 생각해낸 것이 Refresh Token이다.

용도

- Access Token과 똑같은 형태의 JWT
- 처음 로그인을 완료했을때 Access Token과 동시에 발급. 다만, Refresh Token은 긴 유효시간을 가짐(보통 2주정도)
- Access Token이 만료됐을 때 새로 발급해주는 열쇠로 쓰임.

Refresh Token 등장 배경

```
1. [Front] ID와 비밀번호를 준다.

2. [Back] ID와 비밀번호를 검증하고 Access Token을 반환한다.

3. [Front] Access Token을 받아 다음 api호출부터 헤더에 붙여준다.

4. [백엔드] api호출시 Access Token이 유효한지 확인하여 처리한다.
```

- 이렇게 간단하게 구현한 인증 방식은 Access Token이 탈취당했을 때 문제가 생긴다.
- 공격자는 제한없이 API를 호출해 정보를 채갈 수 있다.
- 그래서 나온 개념이 만료시간.

```
1. [Front] ID와 비밀번호를 준다.

2. [Back] ID와 비밀번호를 검증하고 Access Token을 반환한다. 이 때, 만료시간을 설정한다.

3. [Front] Access Token을 받아 다음 api호출부터 헤더에 붙여준다.

4. [Back] api호출시 Access Token이 유효한지, 만료시간이 지나지 않았는지 확인하여 처리한다.
```

- 탈취당했을 때의 문제는 해결되었지만 또다른 문제가 생긴다.
- 사용자의 Access Token이 주기적으로 만료되어 자주 다시 로그인을 해야한다.

- 이를 보완하기 위해 Refresh Token을 통해 Access Token을 재발급 할 수 있도록 한다.

```
1. [Front] ID와 비밀번호를 준다.

2. [Back] ID와 비밀번호를 검증하고 Access Token, Refresh Token을 반환한다. 이 때 Access Token의 만료시간을 설정한다.

3. [Front] Access Token을 받아 다음 api호출부터 헤더에 붙여준다. Refresh Token은 안전한 곳에 보관한다.

4. [Back] api호출시 Access Token이 유효한지, 만료시간이 지나지 않았는지 확인하여 처리한다.

5. [Front] 만약 Access Token이 만료되어 api 동작이 실패하였다면, Refresh Token을 Back에 줘서 Reissue를 건다.

6. [Back] Reissue요청이 오면, Access Token을 새로 만들어 반환한다.
```

- Front에서 Refresh Token을 안전한 곳에 보관하는데, Access Token를 안전한 곳에 보관을 안하는 이유는?
- 안전한 곳일수록 접근하는데 시간이 오래 걸릴 것이다. Access Token은 API요청 시 마다 사용되므로 빈번하게 접근해야한다.
- 따라서 비교적 덜 안전한 곳에 Access Token을 저장하고, Refresh Token은 만료 되었을 때(30분 ~ 2시간)에만 접근해 사용한다.
- 최신방법 중 하나는 Refresh Token을 DB에 저장하는 방식을 채택한다.

```
1. [Front] ID와 비밀번호를 준다.

2. [Back] ID와 비밀번호를 검증하고 Access Token, Refresh Token을 반환한다. 이 때 Access Token의 만료시간을 설정한다.
Refresh Token의 경우 DB에 {ID, Refresh Token} 식으로 저장한다.

3. [Front] Access Token을 받아 다음 api호출부터 헤더에 붙여준다. Refresh Token은 안전한 곳에 보관한다.

4. [Back] api호출시 Access Token이 유효한지, 만료시간이 지나지 않았는지 확인하여 처리한다.

5. [Front] 만약 Access Token이 만료되어 api 동작이 실패하였다면, Refresh Token을 Back에 줘서 Reissue를 건다.

6. [Back] Reissue요청이 오면, Refresh Token이 DB에 있는 것과 같은지 비교하고, 맞다면 Access Token을새로 만들어 반환한다.
```

- 마지막 문제점은 API 요청 -> Access Token이 만료 -> 재발급 -> 다시 API요청 순으로 진행되므로 불필요한 API 호출이 일어난다.
- 이를 해결하기 위해 Access Token의 만료시간을 저장해두고 만료시간이 적게 남았을 경우 or 만료되었을 경우 재발급을 요청한다.

```
1. [Front] ID와 비밀번호를 준다.

2. [Back] ID와 비밀번호를 검증하고 Access Token과 Refresh Token, Access Token의 만료시간을 반환해준다. 이 때 생성한 Refresh Token은 DB에 {ID,Refresh Token}으로 저장한다.

3. [Front] 반환받은 Access Token을 매 api 호출마다 헤더에 붙여서 전송한다.

4. [Back] api호출시 헤더의 Access Token을 확인하고 유효한지, 만료기간이 지났는지를 체크 후 api를 동작시킨다.

5. [Front] Access Token의 만료 기간이 지나거나, 30초 미만으로 남았다면, Back에 Refresh Token을 붙여 Reissue 요청을 보낸다.

6. [Back] Reissue요청이 들어올 경우, Refresh Token이 DB에 있는 것인지 확인한 후, 맞다면 Access Token과 새로운 Access Token 만료 시간을 반환한다.

7. [Front] Reissue결과 반환된 Access Token과 만료기간을 저장하여 다음 api호출에 사용한다.
```

Refresh Token 저장 장소는 어디?

이곳 저곳 검색해 보아도 명확한 해답을 얻지는 못했다.

Front의 저장소에 Refresh Token을 저장하더라도 재발급을 위해서는 Server에 토큰 값을 담아 요청을 보내야한다.

그렇다면 탈취의 위협이 있긴 하다는 것이 아닌가?

최근에는 DB에만 저장한다고 하지만 그런다고 해결이 되는 문제일까?

```
 DB에 저장되어 만료된 Access Token을 받을 경우 재발급 시켜 준다면 무제한으로 탈취당하는 것은 똑같은것이 아닌가

 또는

Refresh Token을 DB에도 Front에도 저장한다고 하면
Refresh Token을 탈취 당했을때 DB에 저장하더라도 Access Token을 발급받을 수 있게 되는 게 아닌가 ..
```

Stack Overflow에서는 **http-only 속성이 부여된 쿠키에 저장하는 것을 권장** 한다고 한다.

이유는 해당 속성이 부여된 쿠키는 자바 스크립트 환경에서 접근할 수 없기 때문이다.

그래서 XSS나 CSRF가 발생하더라도 토큰이 누출되지 않는다.

일반 쿠키나 브라우저의 로컬스토리지는 자바스크립트로 자유롭게 접근할 수 있기 때문에 보안 측면에서는 권장되지 않는다.

물론 쿠키에 담는것은 자바 스크립트로 접근이 불가하기 때문에 XSS 공격으로부터 안전하지만 쿠키의 특성 상 request에 자동으로 실리기 때문에 CSRF공격에 취약하다.

MDN은 저장소로 쿠키를 추천하지 않는다고 한다. MDN Cookie

참고

개발일지

하루히즘 Velog