



세션과 토큰 기반 인증 방식

☰ 분류

암호화

세션 인증 방식과 토큰 인증 방식에 대한 정리

쿠키

쿠키란 클라이언트가 어떤 웹 사이트를 방문할 경우, 사이트의 서버를 통해 클라이언트 브라우저에 설치되는 정보 기록입니다.

쿠키는 Key-value 형태를 띄고 있으며, 해당 클라이언트는 앞으로 요청을 보낼 때마다 **Request header**에 쿠키를 담아 전송하게 됩니다. 서버는 쿠키에 담긴 정보를 통해 현재 **Request**를 보낸 클라이언트가 누구인지 식별할 수 있습니다.

쿠키의 단점

1. 쿠키는 보안에 취약합니다. 요청시 내부 정보를 그대로 보내고, 유출 및 조작의 위험이 존재합니다.
2. 쿠키에는 용량 제한이 있어 많은 정보를 담을 수 없습니다.
3. 웹 브라우저마다 쿠키에 대한 지원 형태가 다르기 때문에 브라우저 간 공유가 불가능합니다.
4. 쿠키의 사이즈가 커질수록 네트워크에 부하가 심해집니다.

세션

쿠키를 통해 브라우저의 로그인 상태를 유지할 수 있었지만, 보안의 위험이 있었습니다.


이를 해결하기 위해 세션은 비밀번호 등 클라이언트의 중요한 정보는 서버 측에서 관리하는 방식입니다. 클라이언트가 서버의 요청을 보낼 때, 서버는 저장해둔 세션 ID를 활용합니다. 로그인 상황을 예시로 들면, 로그인 요청이 성공하는 경우 서버는 세션 ID를 서버에 저장해둡니다.

그리고 클라이언트 브라우저의 쿠키에 세션 ID를 저장합니다. 이제 서버는 다음 요청이 들어올 때마다, 쿠키의 세션 ID의 유효성 검사를 실시합니다.

장단점

1. 쿠키를 포함한 요청이 외부에 노출되더라도 세션 id 자체는 유의미한 개인정보를 담고 있지 않습니다.
2. 그러나 해커가 이를 중간에 탈취하여 클라이언트로 위장할 수 있다는 한계가 존재합니다.
3. 각 사용자마다 고유한 세션 ID가 발급되기 때문에, 요청이 들어올 때마다 회원 정보를 확인할 필요가 없습니다.
4. 서버에서 세션 저장소를 사용하므로 요청이 많아지면 서버에 부하가 심해집니다.

JWT(Json Web Token)

Jwt는 인증에 필요한 정보들을 암호화 시킨 토큰을 의미합니다. 클라이언트의 세션 상태를 저장하지 않고 필요한 정보를 토큰에 저장합니다. 서버는 이 토큰을 클라이언트에게 전달하고, 그것을 증명서처럼 사용합니다. JWT는  을 구분자로 세 가지로 나뉘는 구조를 지닙니다.

JWT.IO

JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.

 <https://jwt.io/>



JSON Web Tokens are an open, industry standard **RFC 7519** method for representing claims securely between two parties.
JWT.IO allows you to decode, verify and generate JWT.

jwt debugger

JWT의 구조




1. 헤더 (HEADER)

Header 의 alg과 typ는 각각 정보를 암호화할 해싱 알고리즘 및 토큰의 타입을 지정합니다.



JWT 토큰 암호화 알고리즘 - HS256과 RS256

이 글에선 JWT 토큰의 전체적인 동작 방식에 대한 내용은 다루지 않고, 암호화 알고리즘에 대해서만 다루도록 하겠습니다. JWT 토큰 암호화 알고리즘 중 대표적인 HS256, RS256에 대해서만 다루겠습니다.

 <https://velog.io/@ddangle/JWT-토큰-암호화-알고리즘-HS256과-RS256>

velog

2. 페이로드 (PAYLOAD)

Payload 는 토큰에 담을 정보를 지니고 있습니다. 주로 클라이언트의 고유 ID 값 및 유효 기간 등이 포함되는 영역입니다. Key-value 형식으로 이루어진 한 쌍의 정보를 Claim이라고 칭합니다. **Payload** 의 내용은 해독이 가능하기 때문에 중요한 정보는 포함해서는 안됩니다.



3. 시그니처 (SIGNATURE)

Signature 는 인코딩된 **Header** 와 **Payload** 를 더한 뒤 비밀키로 해싱하여 생성합니다.
Header 와 **Payload** 는 단순히 인코딩된 값이기 때문에 제 3자가 복호화 및 조작할 수 있지만,
Signature 는 서버 측에서 관리하는 비밀키가 유출되지 않는 이상 복호화 할 수 없습니다. 따라서 **Signature** 는 토큰의 위변조 여부를 확인하는데 사용됩니다.



JWT 인증 과정

1. 클라이언트 로그인 요청이 들어오면, 서버는 검증 후 클라이언트 고유 ID 등의 정보를 **Payload**에 담습니다.
2. 암호화 할 비밀키를 사용해 **Access Token(JWT)**을 발급합니다.
3. 클라이언트는 전달 받은 토큰을 저장해두고, 서버에 요청할 때 마다 토큰을 요청 헤더 **Authorization**에 포함 시켜 함께 전달합니다.
4. 서버는 토큰의 **Signature**를 비밀키로 복호화한 다음, 위변조 여부 및 유효 기간 등을 확인합니다.
5. 유효한 토큰이라면 요청에 응답합니다.

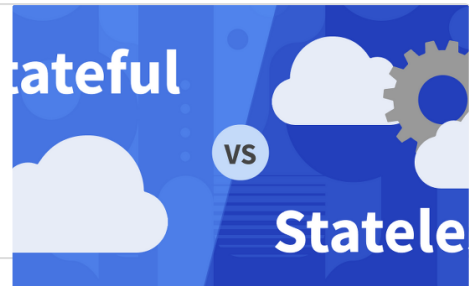
장점

이 방식을 사용하면 토큰을 제 3자가 변경하려 하여도 서버의 비밀키를 알 수 없어서 변경이 어렵고, 변경을 하더라도 서명이 다르기 때문에 요청을 받지 않게 될 것입니다. 이를 통해 데이터의 위변조를 막을 수 있으며, 인증 정보에 대한 별도의 저장소가 필요 없습니다. 클라이언트 인증 정보를 저장하는 세션과는 다르게 서버가 **Stateless** 가 됩니다. 또한 확장성이 매우 좋고, 모바일 애플리케이션에서도 잘 동작합니다.

🌐 아주 쉽게 이해하는 Stateful / Stateless 차이

Stateful 과 Stateless 차이점 웹 공부를 하다보면 클라이언트 (Client)와 서버(Server)간의 통신을 상태유지(Stateful) 하느냐, 상태유지하지않음(Stateless) 으로 하느냐 라는 말귀를 한번쯤은 들어

🖥️ <https://inpa.tistory.com/entry/WEB-Stateful-Stateless> -정리



단점

위에서 설명한 대로 **Payload** 자체는 조희가 가능하기 때문에 중요한 정보를 담을 수는 없습니다. 또한, 토큰의 길이가 길어 인증 요청이 많아질 수록 네트워크 부하가 심해질 것입니다. 또한 토큰은 발급하면 만료될 때까지 계속 사용이 가능하기 때문에 토큰이 탈취 당하면 대처하기가 어렵습니다.

대안

1. 이와 같은 단점들을 보완하기 위해, 토큰의 만료 시간을 짧게 설정하는 방법이 있습니다. 토큰이 탈취 되더라도 빠르게 만료되기 때문에 피해를 최소화할 수 있다는 장점이 있으나, 사용자가 자주 로그인해야 하는 불편함이 수반됩니다.
2. 글을 작성하는 도중 토큰이 만료가 된다면 **Form Submit** 요청을 보낼 때 작업이 정상적으로 처리 되지 않고, 이전에 작성한 글이 날아가는 등의 불편함이 존재합니다. **Sliding Session** 은 서비스를 지속적으로 이용하는 클라이언트에게 자동으로 토큰 만료 기한을 늘려주는 방법입니다. 글 작성 혹은 결제 등을 시작할 때 새로운 토큰을 발급해줄 수 있습니다. 이를 통해 사용자는 로그인을 자주 할 필요가 없어집니다.
3. **Access Token** 과는 별개로 **Refresh Token** 을 사용하는 방법이 있습니다. **Access Token** 에는 짧은 유효기간을 설정하고, 그보다는 긴 유효기간을 가진 **Refresh Token** 을 함께 발급해줍니다. 서버는 **DB** 에 저장된 **Refresh Token** 과 비교하여 유효성 검사를 실시하고, 유효한 경우 새로운 **Access Token** 을 발급해줍니다.

대안 3번의 경우, 서버에서도 Refresh Token을 관리해줘야 하기 때문에 Jwt의 장점을 완전히 누릴 수 없습니다.

그러나 서버에서 원치 않을 경우 Refresh Token을 강제 만료시킬 수 있다는 장점도 존재합니다.

소스 코드 구현 블로그

[Spring Security] JWT(JSON Web Token) 의 코드 구현

깃허브 : 전체 코드 Spring Boot에 Spring Security와 JWT를 사용해 로그인 구현 1. 동작과정 Generating JWT Client : 로그인 요청 POST (id, pw) Server : id, pw가 맞는지 확인 후 맞다면 JWT를

<https://aonee.tistory.com/72>

