Programming

쉽게 알아보는 서버 인증 1편(세션/쿠키 , JWT)

이호연 자유로운 오랑우탄 | 2018. 7. 21. 21:53

앱 개발을 처음 배우게 됐을 때, 각종 화면을 디자인해보면서 프론트엔드 개발에 큰 흥미가 생겼습니다. 한때 프론트엔드 개발자를 꿈꾸기도 했었죠(~~현실은 ...~~) 그러나 서버와 통신을 처음 배웠을 때 마냥 쉬운 일은 아니구나라고 생각했습니다. 항상 당연하게 생각해왔던 기능(데이터를 보내고 받는 작업)들을 직접 구현하려고 하니 헷갈리는게 한둘이 아니더군요. 그 중에서 초창기에 가장 어려움을 겪었던 부분은 바로 '인증' 부분이었습니다.

인증이 필요한 이유

인증은 프론트엔드 관점에서 봤을 때 사용자의 로그인, 회원가입과 같이 사용자의 도입부분을 가리키곤 합니다. 반면 서버사이드 관점에서 봤을 때는 모든 API 요청에 대해 사용자를 확인하는 작업입니다.

사용자 A와 사용자 B가 앱을 사용한다고 가정하겠습니다. 두 사용자는 기본적으로 정보가 다르고 보유하고 있는 컨텐츠도 다릅니다. 따라서 서버에서는 A,B가 요청을 보냈을 때 누구의 요청인지를 정확히 알아야 합니다. 만일 그렇지 못한다면, 자신의 정보가 타인에게 유출되는 최악의 상황이 발생하겠죠? 그렇기에 앱(프론트 엔드)에서는 자신이 누구인지를 알만한 단서를 서버에 보내야 하며, 서버는 그 단서를 파악해 각 요청에 맞는 데이터를 뿌려주게 됩니다.

HTTP 요청에 대해

현재 모바일이나 웹 서비스에서 가장 많이 쓰이는 통신 방식은 HTTP 통신입니다. HTTP 통신은 응답 후 연결을 끊기게 되며 과거에 대한 정보를 전혀 담지 않습니다. 이 말은 지금 보낼 HTTP 요청은 지난 번에 내 정보를 담아 보냈던 HTTP 요청과 전혀 관계가 없다는 말입니다. 따라서 각각의 HTTP 요청에는 주체가 누구인지에 대한 정보가 필수적입니다(인증이 필요없다면 필요없을 수도 있음).

서버에 요청을 보내는 작업은 HTTP 메세지를 보내는 것입니다. HTTP 메세지의 구조는 다음과 같습니다. 일반적으로 헤더와 바디 두가지로 구성되며, 공백은 헤더와 바디를 구분짓는 역할을 합니다. 여기서 헤더에는 기본적으로 요청에 대한 정보들이 들어갑니다. 바디에는 서버로 보내야할 데이터가 들어가게 됩니다. 보통 모바일/웹 서비스의 인증은 HTTP 메세지의 헤더에 인증 수단을 넣어 요청을 보내게 됩니다.

지금부터 아래의 방식들을 통해 현재 HTTP 기본 인증이 어떻게 이뤄지는지를 알아보도록 하겠습니다. 추후에 Node.js를 통해 각각의 인증 방식을 구현하는 포스팅도 올리겠습니다.

인증 방식

1. 계정정보를 요청 헤더에 넣는 방식

가장 보안이 낮은 방식은 계정정보를 요청에 담아 보내는 방식입니다. 위에서 언급한 HTTP 요청에 인증할 수단에 비밀번호를 넣습니다.

네. 정말 최악의 인증방식입니다. 데이터를 요청할 때마다 사용자의 프라이빗한 정보를 계속해서 보낸다는 건 상당히 보안에 안좋습니다. 보통 앱에서는 서버로 HTTP 요청을 할 때 따로 암호화되지 않습니다. 따라서 해커가 마음만 먹으면 HTTP 요청을 가로채서(intercept) 사용자의 계정정보를 알 수 있습니다. 본 방식은 절대로 실제 서비스에서는 쓰이지 않습니다(HTTP 요청을 암호화해서 보안을 높이는 방식으로 HTTPS가 있습니다만 그래도 안쓰임). 개발 단계에서는 인증절차 귀찮을 때나 쓸만한 듯합니다.

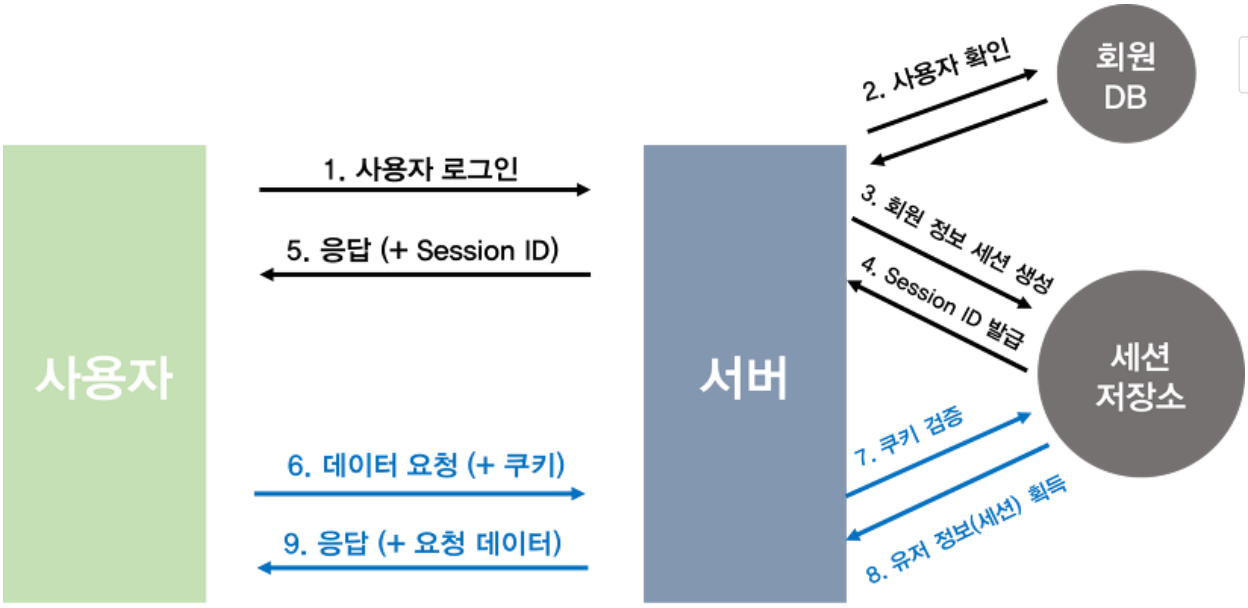
(장점)
1. 인증을 테스트 할 때 빠르게 시도해볼 수 있다.

(단점)

1. 보안에 매우 취약하다.
2. 서버에서는 신호가 올때마다 Id,Pw를 통해 유저가 맞는지 인증해야 한다. 이는 비효율적이다.

2. Session / Cookie 방식

위의 계정 정보를 매번 요청에 넣어서 보내기엔 보안에 너무 취약합니다. 따라서 나온 인증 방식이 Session / Cookie 입니다.



순서는 요약하면 다음과 같습니다.

1. 사용자가 로그인을 한다.

2. 서버에서는 계정정보를 읽어 사용자를 확인한 후, 사용자의 고유한 ID값을 부여하여 세션 저장소에 저장한 후, 이와 연결되는 세션ID를 발행합니다.

3 사용자는 서버에서 해당 세션ID를 받아 쿠키에 저장을 한 후, 인증이 필요한 요청마다 쿠키를 헤더에 실어 보냅니다.

4. 서버에서는 쿠키를 받아 세션 저장소에서 대조를 한 후 대응되는 정보를 가져옵니다.

5. 인증이 완료되고 서버는 사용자에 맞는 데이터를 보내줍니다.

세션 쿠키 방식의 인증은 기본적으로 세션 저장소를 필요로 합니다(Redis를 많이 사용). 세션 저장소는 로그인을 했을 때 사용자의 정보를 저장하고 열쇠가 되는 세션ID값을 만듭니다. 그리고 HTTP 헤더에 실어 사용자에게 돌려보냅니다. 그러면 사용자는 쿠키로 보관하고 있다 인증이 필요한 요청에 쿠키(세션ID)를 넣어

보낼 것입니다. 웹 서버에서는 세션 저장소에서 쿠키(세션ID)를 받고 저장되어 있는 정보와 매칭시켜 인증을 완료합니다.

**세션ID를 쿠키라고 봐도 동일합니다. 쿠키가 사용자 개념에서 더 큰 범주입니다. 세션ID를 쿠키로 저장하는 셈이죠.

**세션과 쿠키 개념이 헷갈리시는 분들이 있으신데, 세션은 서버에서 가지고 있는 정보이며 쿠키는 사용자에게 발급된 세션을 열기 위한 열쇠(SESSION ID)를 의미합니다. 쿠키만으로 인증을 사용한다는 말은 서버의 자원은 사용하지 않는다는 것이며, 이는 즉 클라이언트가 인증 정보를 책임지게 됩니다. 그렇게 되면 위의 첫번째 방식처럼 HTTP 요청을 탈취당할 경우 다 털리게 됩니다. 따라서 보안과는 상관없는 단순히 장바구니나 자동로그인 설정 같은 경우에는 유용하게 쓰입니다.

결과적으로 인증의 책임을 서버가 지게하기 위해 세션을 사용하는 겁니다(사용자가 해킹당하는 것보단 서버가 해킹당하는게 훨씬 어려우니까요!) 사용자(클라이언트)는 쿠키를 이용하고, 서버에서는 쿠키를 받아 세션의 정보를 접근하는 방식으로 인증을 합니다.

(장점)

1. 세션/쿠키 방식은 기본적으로 쿠키를 매개로 인증을 거칩니다. 여기서 쿠키는 세션 저장소에 담긴 유저 정보를 얻기 위한 열쇠라고 보시면 됩니다. 따라서 쿠키가 담긴 HTTP 요청이 도중에 노출되더라도 쿠키 자체(세션 ID)는 유의미한 값을 갖고있지 않습니다(중요 정보는 서버 세션에) 이는 위의 계정정보를 담아 인증을 거치는 것보단 안전해 보입니다.

2. 사용자 A는 1번, 사용자 B는 2번 이런식으로 고유의 ID값을 발급받게 됩니다. 그렇게 되면 서버에서는 쿠키 값을 받았을 때 일일이 회원정보를 확인할 필요 없이 바로 어떤 회원인지를 확인할 수 있어 서버의 자원에 접근하기 용이할 것입니다.

(단점)

1. 장점 1에서 쿠키를 탈취당하더라도 안전할 수 있다고 언급했습니다. 그러나 문제가 하나 있습니다. 만일 A 사용자의 HTTP 요청을 B 사용자(해커)가 가로챘다면 그 안에 들어있는 쿠키도 충분히 훔칠 수 있습니다. 그리고 B 사용자는 그 훔친 쿠키를 이용해 HTTP 요청을 보내면 서버의 세션저장소에서는 A 사용자로 오인해 정보를 잘못 뿌려주게 되겠죠(세션 하이재킹 공격이라고 합니다)

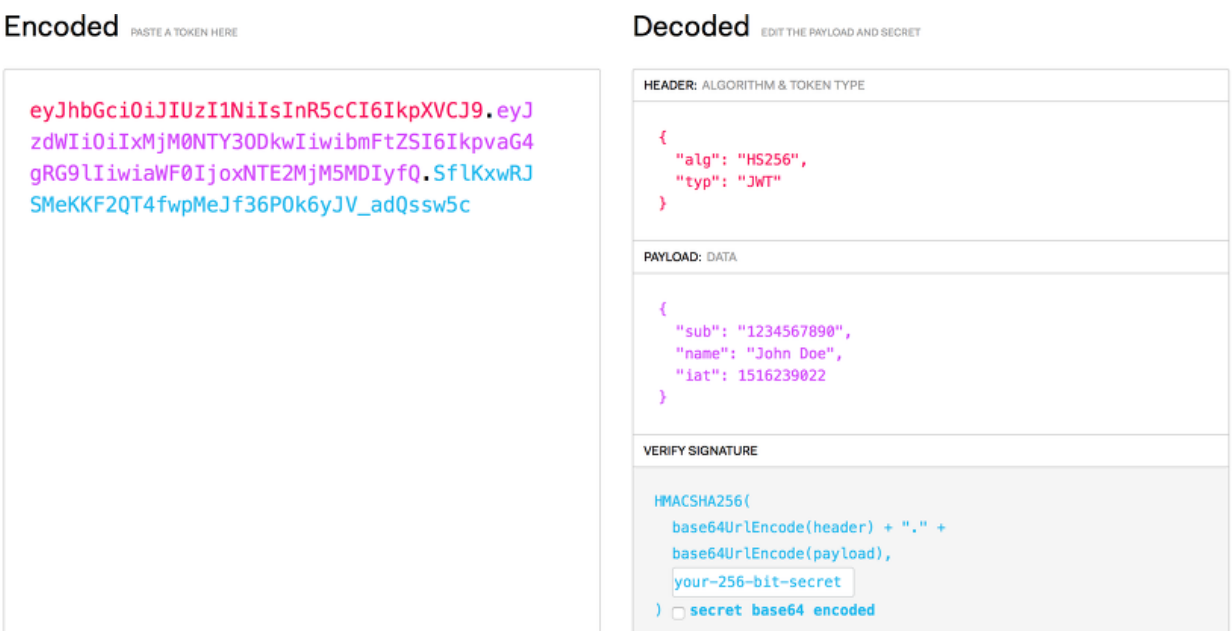
-> 해결책

1. HTTPS를 사용해 요청 자체를 탈취해도 안의 정보를 읽기 힘들게 한다. 2. 세션에 유효시간을 넣어준다.

2. 서버에서 세션 저장소를 사용한다고 했습니다. 따라서 서버에서 추가적인 저장공간을 필요로 하게되고 자연스럽게 부하도 높아질 것입니다.

3. 토큰 기반 인증 방식 (ft. JWT)

JWT는 세션/쿠키와 함께 모바일과 웹의 인증을 책임지는 대표주자입니다. JWT는 Json Web Token의 약자로~~(외국에서 'JOT'으로 읽는다네요 호호)~~ 인증에 필요한 정보들을 암호화시킨 토큰을 뜻합니다. 위의 세션/쿠키 방식과 유사하게 사용자는 Access Token(JWT 토큰)을 HTTP 헤더에 실어 서버로 보내게 됩니다.



<jwt.io사이트 캡쳐>

인증순서 전에 간단하게 JWT에 대해 알아보도록 하겠습니다. https://jwt.io 를 들어가보면 암호화된 토큰을 볼 수 있습니다.

토큰을 만들기 위해서는 크게 3가지, Header,Payload, Verify Signature가 필요합니다.

Header : 위 3가지 정보를 암호화할 방식(alg), 타입(type) 등이 들어갑니다.

Payload : 서버에서 보낼 데이터가 들어갑니다. 일반적으로 유저의 고유 ID값, 유효기간이 들어갑니다.

Verify Signature : Base64 방식으로 인코딩한 Header,payload 그리고 SECRET KEY를 더한 후 서명됩니다.

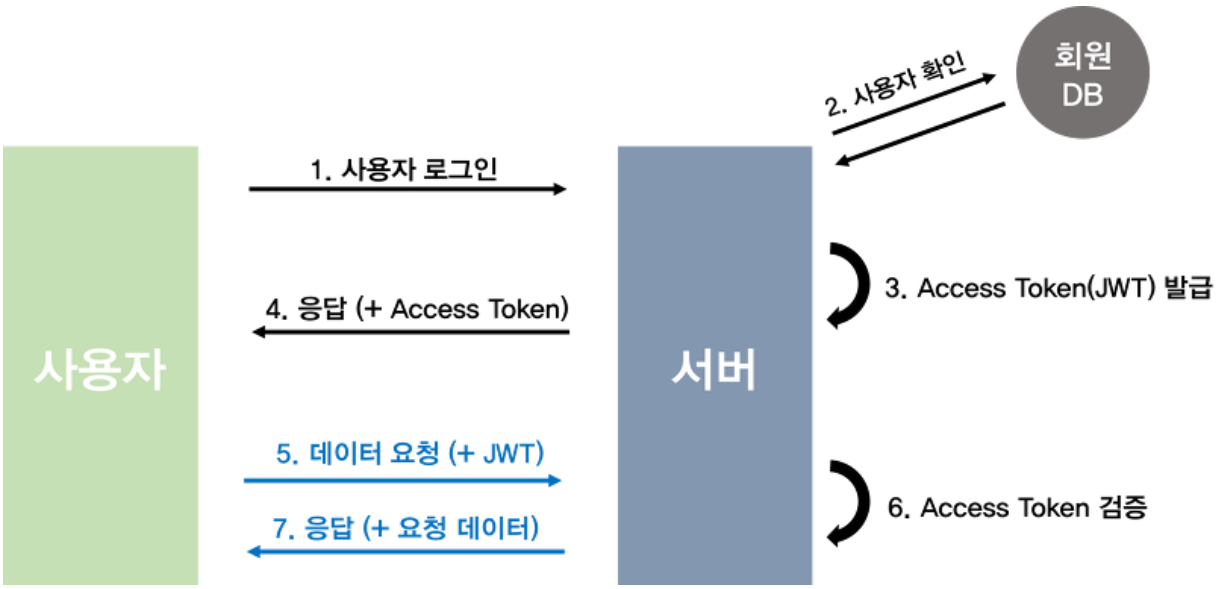
최종적인 결과 : Encoded Header + "." + Encoded Payload + "." + Verify Signature

Header, Payload는 인코딩될 뿐(16진수로 변경), 따로 암호화되지 않습니다. 따라서 JWT 토큰에서 Header, Payload는 누구나 디코딩하여 확인할 수 있습니다. 여기서 누구나 디코딩할 수 있다는 말은 Payload에는 유저의 중요한 정보(비밀번호)가 들어가면 쉽게 노출될 수 있다는 말이 됩니다.

하지만 Verify Signature는 SECRET KEY를 알지 못하면 복호화할 수 없습니다.

A 사용자가 토큰을 조작하여 B 사용자의 데이터를 훔쳐보고 싶다고 가정하겠습니다. 그래서 payload에 있던 A의 ID를 B의 ID로 바꿔서 다시 인코딩한 후 토큰을 서버로 보냈습니다. 그러면 서버는 처음에 암호화된 Verify Signature를 검사하게 됩니다. 여기서 Payload는 B사용자의 정보가 들어가 있으나 Verify Signature는 A의 Payload를 기반으로 암호화되었기 때문에 유효하지 않는 토큰으로 간주하게 됩니다. 여기서 A사용자는 SECRET KEY를 알지 못하는 이상 토큰을 조작할 수 없다는 걸 확인할 수 있습니다.

이제부터 JWT가 어떻게 인증에 사용되는지 알아보도록 하겠습니다.



1. 사용자가 로그인을 한다.

2. 서버에서는 계정정보를 읽어 사용자를 확인 후, 사용자의 고유한 ID값을 부여한 후, 기타 정보와 함께 Payload에 넣습니다.

3. JWT 토큰의 유효기간을 설정합니다.

4. 암호화할 SECRET KEY를 이용해 ACCESS TOKEN을 발급합니다.

5. 사용자는 Access Token을 받아 저장한 후, 인증이 필요한 요청마다 토큰을 헤더에 실어 보냅니다.

6. 서버에서는 해당 토큰의 Verify Signature를 SECRET KEY로 복호화한 후, 조작 여부, 유효기간을 확인합니다.

7. 검증이 완료된다면, Payload를 디코딩하여 사용자의 ID에 맞는 데이터를 가져옵니다.

세션/쿠키 방식과 가장 큰 차이점은 세션/쿠키는 세션 저장소에 유저의 정보를 넣는 반면, JWT는 토큰 안에 유저의 정보들이 넣는다는 점입니다. 물론 클라이언트 입장에서는 HTTP 헤더에 세션ID나 토큰을 실어서 보내준다는 점에서는 동일하나, 서버 측에서는 인증을 위해 암호화를 하냐, 별도의 저장소를 이용하냐는 차이가 발생합니다.

(장점)

1. 간편합니다. 세션/쿠키는 별도의 저장소의 관리가 필요합니다. 그러나 JWT는 발급한 후 검증만 하면 되기 때문에 추가 저장소가 필요 없습니다. 이는 Stateless 한 서버를 만드는 입장에서는 큰 강점입니다. 여기서 Stateless는 어떠한 별도의 저장소도 사용하지 않는, 즉 상태를 저장하지 않는 것을 의미합니다. 이는 서버를 확장하거나 유지,보수하는데 유리합니다.

2. 확장성이 뛰어납니다. 토큰 기반으로 하는 다른 인증 시스템에 접근이 가능합니다. 예를 들어 Facebook 로그인, Google 로그인 등은 모두 토큰을 기반으로 인증을 합니다. 이에 선택적으로 이름이나 이메일 등을 받을 수 있는 권한도 받을 수 있습니다.

여기까지의 글만 봤을 때는 JWT가 세션/쿠키 방식보다 더 효율적으로 보입니다. 하지만 JWT도 단점들이 존재합니다.

(단점)

1. 이미 발급된 JWT에 대해서는 돌이킬 수 없습니다. 세션/쿠키의 경우 만일 쿠키가 악의적으로 이용된다면, 해당하는 세션을 지워버리면 됩니다. 하지만 JWT는 한 번 발급되면 유효기간이 완료될 때 까지는 계속 사용이 가능합니다. 따라서 악의적인 사용자는 유효기간이 지나기 전까지 신나게 정보들을 털어갈 수 있습니다.

-> 해결책

기존의 Access Token의 유효기간을 짧게 하고 Refresh Token이라는 새로운 토큰을 발급합니다. 그렇게 되면 Access Token을 탈취당해도 상대적으로 피해를 줄일 수 있습니다. 이는 다음 포스팅에 나올 Oauth2에 더 자세히 다루도록 하겠습니다.

2. Payload 정보가 제한적입니다. 위에서 언급했다시피 Payload는 따로 암호화되지 않기 때문에 디코딩하면 누구나 정보를 확인할 수 있습니다. (세션/쿠키 방식에서는 유저의 정보가 전부 서버의 저장소에 안전하게 보관됩니다) 따라서 유저의 중요한 정보들은 Payload에 넣을 수 없습니다.

3. JWT의 길이입니다. 세션/쿠키 방식에 비해 JWT의 길이는 깁니다. 따라서 인증이 필요한 요청이 많아질 수록 서버의 자원낭비가 발생하게 됩니다.

지금까지 크게 세션/쿠키, JWT에 대해 알아보았습니다. 인증을 처음 구현해본다면 두가지 방식을 모두 구현해보고 차이점을 직접 느껴보시는 걸 추천드립니다!