API Key

서비스들이 거대해짐에 따라 기능들을 분리하기 시작하였는데 이를위해 Module이나 Application들간의 공유와 독립성을 보장하기 위한 기능들이 등장하기 시작했다. 그 중 제일 먼저 등장하고 가장 널리 보편적으로 쓰이는 기술이 API Key이다.

동작방식

1. 사용자는 API Key를 발급받는다. (발급 받는 과정은 서비스들마다 다르다. 예를들어 공공기관 API같은 경우에는 신청에 필요한 양식을 제출하면 관리자가 확인 후 Key를 발급해준다.
2. 해당 API를 사용하기 위해 Key와 함께 요청을 보낸다.
3. Application은 요청이 오면 Key를 통해 User정보를 확인하여 누구의 Key인지 권한이 무엇인지를 확인한다.
4. 해당 Key의 인증과 인가에 따라 데이터를 사용자에게 반환한다.

문제점

API Key를 사용자에게 직접 발급하고 해당 Key를 통해 통신을 하기 때문에 통신구간이 암호화가 잘 되어 있더라도 Key가 유출된 경우에 대비하기 힘들다. 그렇기때문에 주기적으로 Key를 업데이트를 해야하기 때문에 번거롭고 예기치 못한 상황(한쪽만 업데이트가 실행되어 서로 매치가 안된다는 등)이 발생할 수 있다. 또한, Key한가지로 정보를 제어하기 때문에 보안문제가 발생하기 쉬운편이다.

OAuth2

API Key의 단점을 메꾸기 위해 등작한 방식이다. 대표적으로 페이스북, 트위터 등 SNS 로그인기능에서 쉽게 볼 수 있다. 요청하고 요청받는 단순한 방식이 아니라 인증하는 부분이 추가되어 독립적으로 세분화가 이루어졌다.

동작방식

1. 사용자가 Application의 기능을 사용하기 위한 요청을 보낸다. (로그인 기능, 특정 정보 열람 등 다양한 곳에서 쓰일 수 있다. 여기에서는 로그인으로 통일하여 설명하겠다.)
2. Application은 해당 사용자가 로그인이 되어 있는지를 확인한다. 로그인이 되어 있지 않다면 다음 단계로 넘어간다.
3. Application은 사용자가 로그인되어 있지 않으면 사용자를 인증서버로 Redirection한다.
4. 간접적으로 Authorize 요청을 받은 인증서버는 해당 사용자가 회원인지 그리고 인증서버에 로그인 되어 있는지를 확인한다.
5. 인증을 거쳤으면 사용자가 최초의 요청에 대한 권한이 있는지를 확인한다. 이러한 과정을 Grant라고 하는데 대체적으로 인증서버는 사용자의 의지를 확인하는 Grant처리를 하게 되고, 각 Application은 다시 권한을 관리 할 수도 있다. 이 과정에서 사용자의 Grant가 확인이 되지않으면 다시 사용자에게 Grant요청을 보낸다.

> *Grant 란?* Grant는 인가와는 다른 개념이다. 인가는 서비스 제공자 입장에서 사용자의 권한을 보는 것이지만, Grant는 사용자가 자신의 인증정보(보통 개인정보에 해당하는 이름, 이메일 등)를 Application에 넘길지 말지 결정하는 과정이다.

6. 사용자가 Grant요청을 받게되면 사용자는 해당 인증정보에 대한 허가를 내려준다. 해당 요청을 통해 다시 인증서버에 인가 처리를 위해 요청을 보내게 된다.

7. 인증서버에서 인증과 인가에 대한 과정이 모두 완료되면 Application에게 인가코드를 전달해준다. 인증서버는 해당 인가코드를 자신의 저장소에 저장을 해둔다. 해당 코드는 보안을 위해 매우 짧은 기간동안만 유효하다.
8. 인가 코드는 짧은 시간 유지되기 때문에 이제 Application은 해당 코드를 Request Token으로 사용하여 인증서버에 요청을 보내게된다.
9. 해당 Request Token을 받은 인증서버는 자신의 저장소에 저장한 코드(7번 과정)과 일치하지를 확인하고 긴 유효기간을 가지고 실제 리소스 접근에 사용하게 될 Access Token을 Application에게 전달한다.
10. 이제 Application은 Access Token을 통해 업무를 처리할 수 있다. 해당 Access Token을 통해 리소스 서버(인증서버와는 다름)에 요청을 하게된다. 하지만 이 과정에서도 리소스 서버는 바로 데이터를 전달하는 것이 아닌 인증서버에 연결하여 해당 토큰이 유효한지 확인을 거치게된다. 해당 토큰이 유효하다면 사용자는 드디어 요청한 정보를 받을 수 있다.

문제점

기존 API Key방식에 비해 좀 더 복잡한 구조를 가진다. 물론 많은 부분이 개선되었다. 하지만 통신에 사용하는 Token은 무의미한 문자열을 가지고 기본적으로 정해진 규칙없이 발행되기 때문에 증명확인 필요하다. 그렇기에 인증서버에 어떤 식이든 DBMS 접근이든 다른 API를 활용하여 접근하는 등의 유효성 확인 작업이 필요하다는 공증 여부 문제가 있다. 이러한 공증여부 문제뿐만 아니라 유효기간 문제도 있다.

JWT

JWT는 JSON Web Token의 줄임말로 인증 흐름의 규약이 아닌 Token 작성에 대한 규약이다. 기본적인 Access Token은 의미가 없는 문자열로 이루어져 있어 Token의 진위나 유효성을 매번 확인해야 하는 것임에 반하여, JWT는 인증여부 확인을 위한 값, 유효성 검증을 위한 값 그리고 인증 정보 자체를 담고 있기 때문에 인증서버에 묻지 않고도 사용할 수 있다. 토큰에 대한 자세한 내용과 인증방식은 JWT문서를 참조하자.

문제점

서버에 직접 연결하여 인증을 학인하지 않아도 되기 때문에 생기는 장점들이 많다. 하지만 토큰 자체가 인증 정보를 가지고 있기때문에 민감한 정보는 인증서버에 다시 접속하는 과정이 필요하다.

참고사이트

https://www.sauru.so/blog/basic-of-oauth2-and-jwt/