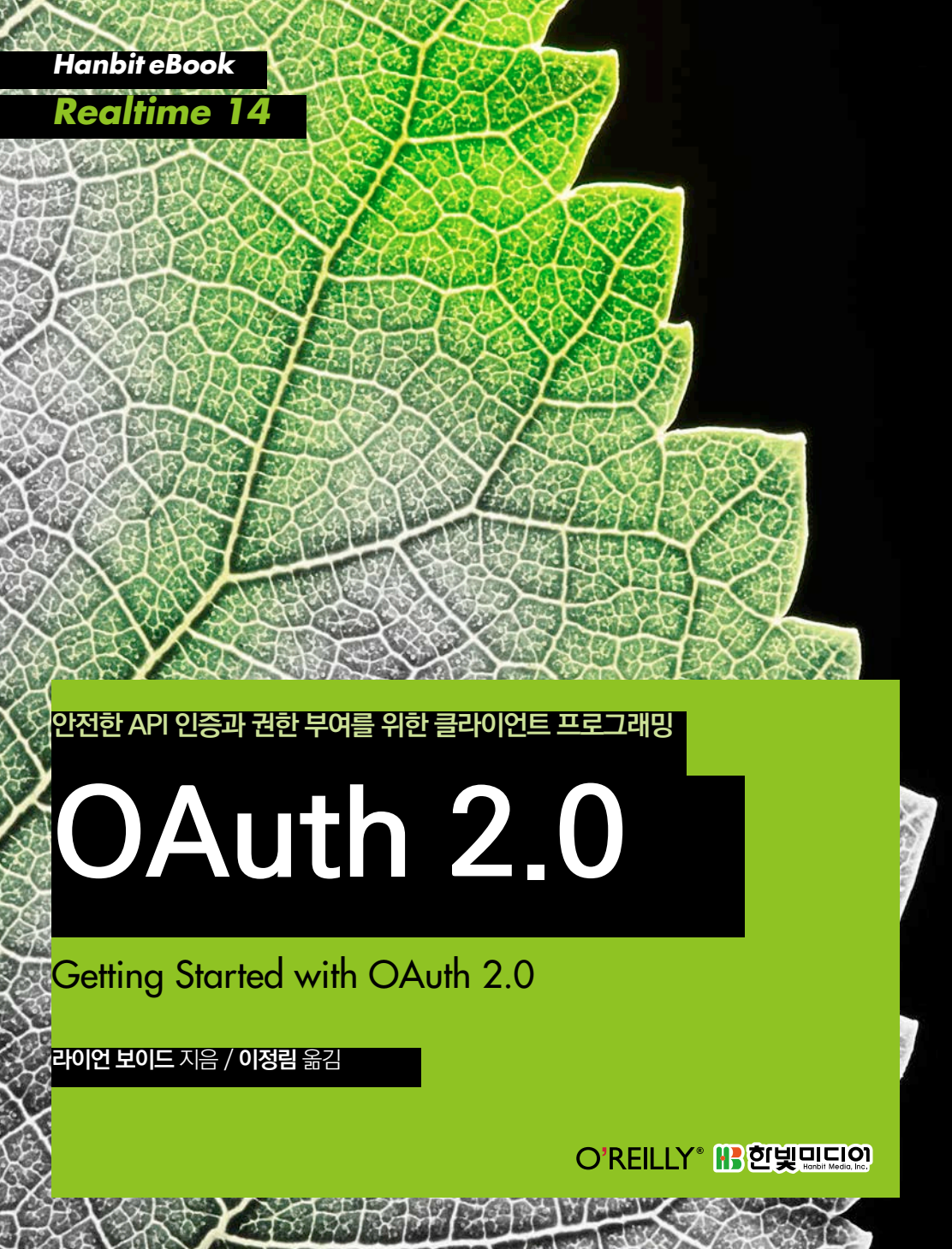
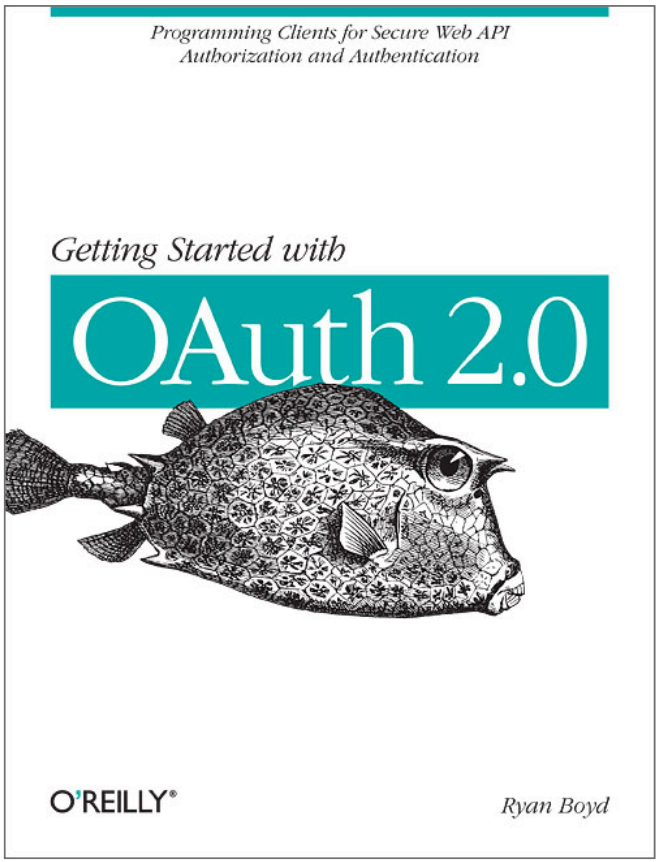

Hanbit eBook
Realtime 14
안전한 API 인증과 권한 부여를 위한 클라이언트 프로그래밍
OAuth 2.0
Getting Started with OAuth 2.0
라이언 보이드 지음 / 이정림 옮김
O'REILLY
한빛미디어
Hanbit Media, Inc.



이 도서는 O'REILLY의
Getting Started with OAuth 2.0의
번역서입니다.

안전한 API 인증과 권한 부여를 위한 클라이언트 프로그래밍

OAuth 2.0

안전한 API 인증과 권한 부여를 위한 클라이언트 프로그래밍 **OAuth 2.0**

초판발행 2012년 12월 28일

지은이 라이언 보이드 / **옮긴이** 이정림 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-7914-994-4 15560 / **정가** 11,000원

책임편집 배용석 / **기획** 김창수 / **편집** 김병희, 안선화
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 김현미
마케팅 박상용, 박주훈, 정민하

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanb.co.kr / **이메일** ask@hanb.co.kr

Published by HANBIT Media, Inc. Printed in Korea

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanb.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

지은이_ **라이언 보이드**

라이언 보이드는 개발자들이 구글 기술을 이용해 Google Apps를 확장하고 비즈니스를 만들 수 있도록 해주는 구글의 개발자다. OpenSocial에서 경력을 쌓았으며, 구글의 AtomPub API 관련 팀에서 개발자들을 이끌었다. 구글에서 일하기 전에는 RIT의 중앙 웹 호스팅 환경 구축을 위한 웹 아키텍터이자, 학사시스템을 만드는 웹 애플리케이션 개발자로서 고등교육 기관에서 일했다.

옮긴이_ **이정림**

SK C&C 솔루션 개발팀에서 NEXCORE 솔루션 간 콘텐츠 연계를 위한 협업 플랫폼을 개발하고 있다. 이전에는 BISTel R&D팀에서 제조 공정 자동화를 위한 EES 프레임워크를 개발하여 삼성전자, 독일 Siltronic, 싱가폴 GSMC 등의 제조 공정에 적용하였다. 노래 부르기를 좋아해서 가끔 회사 내, 홍대 등에서 공연활동을 하고 있다.

테크니컬 리뷰어_ **강대명**

오픈소스 컨트리뷰터로 NHN에서 잉여개발자로 일했다. 현재 영어 공부를 위해 해외에 머물고 있다.

저자 서문

필자는 1999년부터 웹 기반 API를 사용하여 애플리케이션을 SOAP 기반 웹 서비스로 만들었다. 수천 명의 개발자가 구글Google의 REST 기반 API를 사용하여 구글 캘린더, Picasa Web Albums, YouTube 등을 개발하고 있다. 각 API는 사용자를 대신할 권한이 필요했으며, 초기에는 ClientLogin이나 AuthSub 같은 소유권한 기술을 사용했다. 만약 Google API를 사용하는 개발자들이 야후Yahoo에서 제공하는 API와 통합하고자 한다면, 야후의 BBAuth를 애플리케이션에 추가해야 했다. 여러 업체에서 제공하는 API로 만든 애플리케이션에 소유권한 기술을 사용하기는 어려웠다.

OAuth 1.0의 개발은 많은 개발자의 수고를 덜어주었고 하나의 소유권한 기술로 수많은 웹 API를 사용할 수 있게 해주었다. 그러나 OAuth 1.0은 암호화 방식을 비롯하여, 서버에서 다른 서버로 웹 애플리케이션 플로우 없이 애플리케이션에 권한을 부여하는 것을 제약하는 등 몇 가지 문제점이 있었다. OAuth 2.0의 표준이 거의 완료되었다는 소식을 듣고 매우 기뻤다. OAuth 2.0은 다양한 용도로 사용할 수 있으며, 여러 애플리케이션에 적용하기 쉬운 권한 프로토콜을 제공하기 때문이다.

아마도 가장 흥미로운 것은 머지않은 OpenID Connect의 표준화다. OpenID Connect는 다양한 애플리케이션이 같은 인증 정보를 이용해 로그인(인증)할 수 있게 해주는 OAuth 2.0으로 만들어진 프로토콜이다. 나는 OpenID 초기 버전을 웹 애플리케이션에 적용했던 수백 명의 개발자들과 일하면서, 그것을 매끄럽게 적용하는 것이 쉽지 않다는 걸 알게 되었다. 그러나 OAuth 2.0의 개발로 개발자들은 권한 처리를 쉽게 할 수 있게 되었고 OpenID Connect를 통해 인증 처리를 쉽게 할 수 있게 되었다.

여러분이 이 책을 통해, 웹의 차세대 소유권한 기술인 OAuth 2.0과 OpenID Connect의 기본 지식을 습득하여 실무에 사용할 수 있기를 희망한다.

감사의 글

수년간 나를 이끌어주고 전문적인 의견을 제공해준 구글의 'the identity and auto' 팀과 이 책을 리뷰하고 조언해준 Eric Sachs와 Marius Scurtescu, Brena DE Me-Deimos에게 감사의 말을 전한다. 또한, 가족과 친구 그리고 Google's Developer Relations Group 동료의 아낌없는 지원에 감사하고 싶다. 마지막으로 OAuth를 만든 사람들과 관련 워킹그룹의 큰 도움이 아니었다면, 이 책을 집필하지 못했을 것이다.

지은이 **라이언 보이드**

역자 서문

요즘 들어 인터넷 사용자로서, 또한 웹 서비스를 제공해야 하는 플랫폼 개발자로서 웹 기술이 폭발적으로 성장하고 있다는 것을 새삼 느끼고 있다. 특히 스마트폰의 보급과 SNS의 활성화, 클라우드 컴퓨팅 기술의 발달, 차세대 웹 표준인 HTML5를 이용한 웹 사이트의 증가가 이를 뒷받침한다. 이러한 상황에서 OAuth의 등장은 사용자 인증을 통합하고 Open API의 권한 위임을 통해 다양한 벤더에서 제공하는 웹 서비스를 쉽게 연동하게 해준다. 이는 웹 사이트에서 사용자 회원 가입 없이 매시업된 서비스를 제공하여, 사용자는 종합선물세트라도 받은 듯할 것이다.

이 책을 접하기 전 나는 특정 웹 사이트에서 회원 가입 없이 구글 계정으로 로그인하고, 인터넷 기사를 읽은 후 클릭 한 번으로 페이스북의 담벼락에 기사를 공유하는 짜릿한 경험을 하고는, '대체 이건 뭐지?' 하는 궁금증을 가졌었다. 이 책을 읽는 독자들도 나와 같은 경험을 해보고 궁금증을 가졌으리라 생각한다. 특히 OAuth의 도입을 고민하는 분이라면 이 책을 강력히 추천하고 싶다. 이 책이 사례별 OAuth의 단계적 흐름과 장/단점, 그리고 사용 예제를 설명하기 때문이다.

마지막으로 번역을 시작하면서 태어난 아들 승빈이와 그럼에도 불구하고 물심양면으로 이해해준 착한 아내 은희에게 고마움을 전하고 싶다.

2012. 12 잠든 승빈이 옆에서

이정림

대상 독자 및 예제 파일

초급 | 초중급 | **중급** | 중고급 | 고급

- 웹 사이트를 통합 인증과 매시업 서비스에 관심이 많은 분
- OAuth를 도입하려는 서비스 제공자
- OAuth를 적용하려는 웹 개발자

예제 파일은 “http://examples.oreilly.com/0636920021810/”에서 받을 수 있다.

한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook 입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. **eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.**

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내시는 선배, 전문가, 고수분에게는 보다 쉽게 집필하실 기회가 되리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 준비 중이며, 조만간 선보일 예정입니다.

2. **무료로 업데이트되는, 전자책 전용 서비스입니다.**

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정한 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나, 저자(역자)와 독자가 소통하면서 보완되고 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위하여, DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT기기에서 자유롭게 활용하실 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해, 독자 여러분이 언제 어디서 어떤 기기를 사용하시더라도 편리하게 전자책을 보실 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

차례

1 | OAuth 소개

1.1 OAuth의 탄생 배경

영화 '페리스의 해방(Ferry Bueller's Day Off)'에는 한 주차 요원이 1961년식 페라리를 무단으로 훔쳐 타고 달아나는 장면이 나온다. 새로 구입한 머스탱[01]을 이처럼 도둑맞지 않으려면, 어떻게 보호해야 할까? 주차 요원이나 아이들이 사용할 수 없는 특별한 키(권한을 제한하는)를 제공하거나, 트렁크가 열리거나 과속운전을 하지 않도록 제어하면 될 것이다(보호한다).

OAuth는 온라인상에서 이와 동일한 이슈를 해결하기 위해 탄생했다.

구글Google이 'Google Calendar API'[02]를 처음 배포했을 때, 애플리케이션 개발자들은 사용자의 구글 캘린더Google Calendar를 읽고 조작할 수 있었다. 사용자가 권한을 위임하는 유일한 방법은 애플리케이션에 사용자 계정 ID와 비밀번호를 주는 것이었다. 그러면 애플리케이션은 구글이 제공하는 ClientLogin 프로토콜을 이용해 구글 캘린더를 사용할 수 있었다.

'고유 프로토콜proprietary protocol'[03]인 ClientLogin과 표준 프로토콜인 HTTP 기본 인증HTTP Basic Authentication은 데이터 접근을 위해 사용자에게 비밀번호를 요청하는 다양한 규모의 애플리케이션에 사용되었다. 데스크톱 애플리케이션을 제외한 웹에서 사용하는 모든 애플리케이션은 이런 인증이 필요하다. 온라인 사진 공유 사이트인 '플리커Flickr'[04]도 그 중 하나다. 원래 독자적인 회사였던 플리커는 구글이

01 (역자주) 미국 포드 사에서 만든 자동차다.

02 https://developers.google.com/google-apps/calendar/

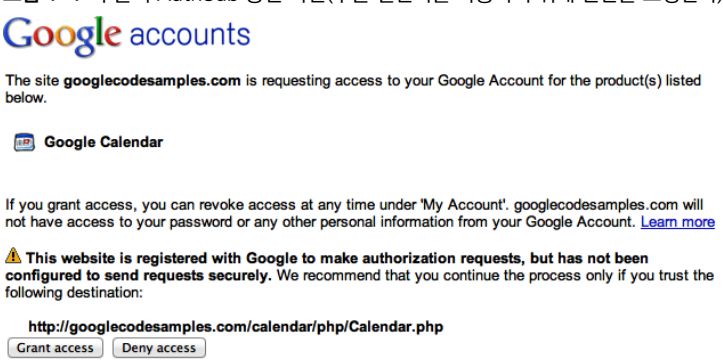
03 (역주자) 원문의 의미를 전달하기 위해서 'proprietary protocol'을 '고유 프로토콜'이라고 하였다. 고유 프로토콜은 특정 회사에서 제공하는 프로토콜로, 표준이 아니며 소유권이 제작한 회사에 있다.

04 http://www.flickr.com/

Blogger를 인수한 수년 후, 야후Yahoo에 인수되었다. 사실 야후는 사용자 비밀번호를 사용하여 웹 애플리케이션과 연동하는 구글과 플리커를 두려워했는데, 이들이 웹의 인증 문제를 개선하는 새로운 고유 프로토콜 개발을 주도해왔기 때문이었다.

사용자 데이터에 접근이 필요하면, 애플리케이션은 구글의 AuthSub[05](그림 1-1 참조)이나 야후의 BBAuth[06] 같은 프로토콜을 사용하여 '권한 요청 페이지'로 사용자를 리다이렉트할 것이다. 사용자가 자신의 계정으로 로그인하여 접근을 허가하면, 애플리케이션은 사용자 데이터에 접근할 때 사용하는 토큰을 얻는다.

그림 1-1 구글의 AuthSub 승인 화면(구글 캘린더를 사용하기 위해 권한을 요청한다)



이것은 일부 보안 문제를 해결하지만, 개발자들은 더 많은 노력을 기울여야 했다. 주요 API 제공 업체를 통합해야 하는 개발자로서는, 애플리케이션에 사용되는 '웹 기반 권한 프로토콜web-based authorization protocol'을 학습하고 구현해야 했기 때문이다.

새로운 API를 만드는 신생 기업들은 소유 인증 스키마를 구현하거나 사용자 정의 스키마를 개발하기가 쉽지 않았다. 그뿐만 아니라 개발한 API에 보안상 취약점이

05 https://developers.google.com/gdata/docs/auth/authsub
06 http://developer.yahoo.com/auth/

라도 있으면 큰 문제를 일으킬 수도 있었다. 그래서 이들 신생 기업이나 주요 API 제공 업체들은 웹 기반의 권한 플로우에 대한 일관성을 유지하기 위해, 표준 프로토콜을 만들기로 결의했다.

1.2 개발자들은 왜 OAuth에 관심을 가져야 하는가?

애플리케이션 개발자들은 협업 플랫폼과 소셜 네트워크를 사용하여, 웹 어디서든 데이터를 가진 사용자들과 연결할 수 있다. 데이터를 가진 사용자와의 연결은 보관하고 있던 낡은 데이터를 제거하고 개선하여 데이터 효율을 높이고, 경쟁자들의 애플리케이션과 차별되게 해준다.

OAuth는 애플리케이션이 안전하게 사용자 데이터에 접근할 수 있도록 제어하며, 사용자에게 계정 비밀번호를 요구하지 않는다. OAuth 기반 API가 제공하는 기능은 다음과 같다.

- 사용자의 소셜 네트워크에 접근하기 - 페이스북Facebook 친구들, 트위터Twitter 팔로어들, 또는 구글 주소록
- 다른 이의 페이스북 담벼락 또는 트위터에 글을 남기고, 자신의 웹 사이트에서 그들의 활동에 대한 정보 공유하기
- 데이터를 다른 이의 온라인 파일 시스템에 저장하기 위해, 그들의 구글 닥스Google Docs 또는 드랍박스Dropbox 계정에 접근하기
- 세일즈포스Salesforce의 CRM, 트립잇TripIt의 여행 계획 같은 여러 데이터 소스를 매시업mash up해서, 더 나은 의사결정을 위한 비즈니스 애플리케이션 통합하기

앞서 언급한 OAuth API로 개인 데이터에 접근하거나 업데이트하기 위해, 애플리케이션은 데이터의 소유권에 대한 권한 위임을 요구한다. API와 웹 환경에서 사용하는 300여 개 이상의 애플리케이션이 데이터에 접근하기 위해 OAuth를 지원한

다(Programmable Web in February 2012 참고[07]).

API의 권한을 다루는 공통 프로토콜common protocol은 개발자의 생산성을 향상한다. 새로운 API와 통합하는 데 드는 학습 시간을 줄여주기 때문이다. 동시에 권한 표준authorization standard은 API의 보안에 대한 신뢰성을 더욱 향상시킨다. 표준은 많은 커뮤니티에 의해 성숙되어왔기 때문이다.

1.3 왜 API는 비밀번호만으로 인증하는 방법을 사용하지 않는가?

웹에서 '사용자 이름'과 '비밀번호'는 일반적으로 인증과 권한에 대한 최소공분모다. 사용자 이름과 비밀번호는 수많은 로그인 페이지에서 HTTP Basic과 HTTP Digest 인증을 사용해왔다. 그러나 비밀번호를 물어보는 것은 수많은 부작용도 낳았다. 다음은 그 부작용이다.

신뢰

사용자는 애플리케이션에 비밀번호를 제공하기 꺼려한다.

피싱에 대해 둔감한 사용자

사용자가 애플리케이션에 쉬운 비밀번호를 사용하면 편할지 모르지만, 한편으로는 피싱을 당하기 쉽다.

접근 범위가 늘어남에 따른 위험부담

사용자가 애플리케이션에 비밀번호를 제공하면 애플리케이션에 필요한 데이터뿐만 아니라 사용자 계정 안에 있는 모든 데이터에 접근할 수 있게 된다. 따라서 애플리케이션은 사용자가 비밀번호를 안전하게 저장할 수 있도록, 외부로 비밀번호가 노출되지 않게 관리해야 한다. 대부분의 개발자는 보안 위험에 노출되어 추가적인 책임을 지는 것을 원하지 않는다.

---

07 http://www.programmableweb.com/apis/directory/1?auth=OAuth

신뢰성의 제한

사용자가 비밀번호를 바꾸면 애플리케이션은 더는 해당 데이터에 접근하지 못한다.

폐기 문제

애플리케이션이 데이터에 접근할 수 없도록 제한하는 유일한 방법은 비밀번호를 바꾸는 것이다. 비밀번호를 변경하면 모든 애플리케이션의 접근이 취소된다.

필수적인 비밀번호

API 제공 업체가 OpenID 또는 SAML("1.4.2 연합 인증" 참조) 같은 연합 인증 방식을 제공하면, 사용자는 비밀번호를 사용하지 않아도 된다. 그러나 이렇게 되면 사용자는 API를 이용한 애플리케이션을 사용할 수 없다.

더 강력한 인증 구현의 어려움

API 제공 업체가 API 인증을 위해 비밀번호를 요구할 경우, CAPTCHA[08]나 다중 인자 인증multifactor authentication(일회용 비밀번호 토큰) 같은 기술을 이용해 계정 보안을 향상해야 하는 문제가 생긴다.

1.4 OAuth 관련 용어

OAuth를 이해하기 위해 관련 용어를 알아두는 것이 중요하다. 먼저 핵심용어를 소개하고 책 전반에 있는 추가적인 용어들을 설명하겠다.

1.4.1 인증

인증Authentication은 원하는 사용자가 맞는지 '사용자의 신분'을 확인하는 과정이다. 실생활을 예로 들면, 경찰은 당신의 신분을 확인하기 위해 주민등록증이나 운전면

08 CAPTCHA(Completely Automated Public Turing test to tell Computers and Humans Apart)는 HIP(Human Interaction Proof) 기술의 일종으로, 사용자가 실제 사람인지 컴퓨터 프로그램인지를 구별하기 위해 사용되는 방법이다. [출처: 위키백과(ko.wikipedia.org)]

허증 같은 신분증을 요구한다. 그러고는 당신의 생김새와 신분증의 사진이 일치하는지를 보고 당신의 신분을 확인한다.

데스크톱 컴퓨터나 웹에서의 인증은 키보드를 치는 사용자가 계정의 소유자인지 확인하는 것이다. 인증을 위해 주로 사용자 이름과 비밀번호를 확인한다. 사용자 이름은 사용자 본인임을 나타내며, 소프트웨어 애플리케이션은 사용자가 제공한 비밀번호가 맞으면 올바른 사용자라고 가정한다.

1.4.2 연합 인증

일부 애플리케이션은 사용자의 신분을 확인하기 위해 시스템 계정(사용자 이름과 비밀번호를 포함하는)뿐만 아니라 다른 서비스를 사용하기도 한다. 이것을 '연합 인증'이라고 부른다.

기업 IT 환경에서 애플리케이션들은 사용자 인증을 위해 Active Directory 서버, LDAP 서버, SAML 제공 업체를 신뢰한다.

웹 환경에서는 사용자의 인증을 다루는 OpenID 제공 업체(구글 또는 야후)를 종종 신뢰한다. OpenID 제공 업체들은 연합해서 애플리케이션 개발자와 사용자 둘 모두에게 많은 혜택을 준다. OpenID는 연합된 인증을 다루기 위한 가장 일반적인 오픈 웹 프로토콜open web protocol이다.

이 책에서는 OAuth 2.0에 기반을 둔 OpenID의 차세대 버전인 OpenID Connect에 대해서만 설명할 것이다.

1.4.3 권한

권한Authorization은 문서 읽기나 이메일 계정에 접근하는 등의 일부 행위를 수행할 권리를 가진 사용자를 검증하는 과정이다. 주로 권한을 검증하기 전에 사용자 인증을 요구한다. 가령 과속 차량을 단속하는 경찰이 당신의 차량 속도를 측정할 경우,

그는 먼저 당신의 운전면허증으로 신분을 확인하고 운전이 허가된 사용자인지(만료일, 제한 조건 등) 검사한다.

위의 과정이 온라인에서도 똑같이 일어난다. 웹 애플리케이션은 먼저 사용자 로그인으로 당신의 신분을 확인한다. 그런 다음 각 동작에 대한 접근 제어 리스트를 확인하면서 어떤 데이터에 접근할 수 있고, 어떤 서비스를 사용할 수 있는지를 확인한다.

1.4.4 권한 위임

권한 위임은 본인을 대신하여 다른 사람이나 애플리케이션에 권한을 부여하는 행위다. 대리 주차 서비스를 제공하는 호텔에 주차하는 경우를 생각해보자. 이때 당신은 대리 주차 요원에게 자동차 키를 주고 당신 대신 운전할 것을 허가한다.

OAuth도 비슷하게 동작한다. 사용자는 본인 대신 액션을 수행하도록 애플리케이션에게 접근을 허가하고, 애플리케이션은 허가받은 액션만 수행할 수 있다.

1.4.5 역할

OAuth 프로토콜 플로우OAuth protocol flow 안에는 몇 가지 중요 액터actor들이 있다.

자원 서버 Resource Server

OAuth가 보호하는 사용자 소유의 자원을 호스팅하는 서버다. 자원 서버는 일반적으로 사진, 비디오, 달력, 주소록 같은 데이터를 보유하고 보호하는 API 제공업체다.

자원 소유자 Resource Owner

일반적으로 애플리케이션의 사용자인 자원 소유자는 자원 서버에서 호스팅되는 그들의 데이터에 접근을 허가할 수 있다.

클라이언트Client

애플리케이션은 권한을 가지고 있는 자원 소유자 대신, API를 이용하여 보호된 자원에 작업 수행을 요청한다.

권한 서버 Authorization Server

권한 서버는 자원 서버에서 보호되는 자원에 접근하기 위해 자원 소유자에게 동의를 얻고 클라이언트에 액세스 토큰을 발행한다. 소규모 API 제공 업체들은 권한 서버와 자원 서버 용도로 하나의 애플리케이션과 URL 체계를 사용하는 경우도 있다.

1.5 서명에 대한 큰 논쟁

OAuth 1.0은 클라이언트의 인증과 권한 검증을 위해 각 API 요청에 암호화 서명을 요구했다. 암호화 기술은 일반 개발자뿐만 아니라 고급 개발자에게도 어려운 문제였는데, 이로 인해 많은 개발자의 불만이 일었다. 다른 권한 프로토콜에 비해 OAuth의 사용률이 저조한 것도 그 때문이었다.

2007년 OAuth 1.0이 개발되었을 때, API의 보안을 위해 암호화 서명을 사용하기로 했다. 당시 주요 API 제공 업체들은 SSL/TLS를 사용하여 API를 보호하지 않고 일반 HTTP 엔드포인트endpoint[09]를 사용했다. 수년이 지난 후 SSL/TLS는 API를 보호하는 방법으로 대중화되었고, 보안 커뮤니티의 서명 요구는 점차 감소했다.

OAuth 1.0에서 사용하던 암호화 방식의 복잡성으로 인해 API 사용률이 낮아지고, API를 지원하는 SSL/TLS의 보급률이 높아지면서 OAuth WRAPWeb Resource Authorization Profile이 개발되었다. OAuth WRAP은 OAuth 2.0의 전 단계로, 복잡한 서명을 제거하고 토큰 전달 방식을 소개하였다.

09 엔드포인트는 끝점을 가리키며, HTTP 통신에서는 엔드포인트를 URL 주소로 지정한다.

OAuth 2.0의 개발이 마무리될 즈음, OAuth 스펙을 정의하던 에런 해머-라하브 Eran Hammer-Lahav를 비롯한 몇몇이 서명을 사용하지 않는 것을 강하게 반대했다. 에런은 자신의 블로그에 올린 "OAuth 2.0 (without Signatures) Is Bad for the Web"[10]란 글에서 서명의 복잡함에 대해 인정하면서도, 서명이 여전히 가치 있다고 항변했다. OAuth 2.0에서 서명을 없앨 경우 개발자의 실수를 야기하고, 사용자의 신용정보를 뜻하지 않게 악의적인 API 엔드포인트로 보낼 수 있으며, 서명으로 신용정보가 보호되지 않기 때문에 신용정보를 이용해 추가 요청을 만들어 악용할 수 있다고 하였다. 그는 서명을 없애는 것은 실현 가능성이 없다고 주장했으며, API와 OAuth 엔드포인트에 자동 탐지 기능을 추가하는 것은 치명적인 위험이 될 거라고 믿었다. 엔지니어들은 종종 보안성과 사용성 사이의 균형을 깨야 한다. 위의 경우도 마찬가지다.

1.5.1 전달 토큰을 이용한 위험 완화

서명의 주요 관심사 중 하나는, 서명을 제거할 경우 인증 서버나 자원 서버로 요청을 보낼 때 개발자들이 'SSL/TLS 인증서 체인certificate chain'[11]을 적절히 검증할 수 있느냐 하는 문제다. 이에 대해서는 OAuth 2.0 스펙에서도 요구하고 있으며, 'OAuth 2.0 위협 모델 및 보안 고려 사항'[12] 문서에도 언급되어 있다. 하지만 널리 알려진 라이브러리들로는 인증서와 인증서 권한 검증이 힘들고 이런 문제를 고치기도 어려워, 결과적으로 개발자가 검증을 무시함으로써 애플리케이션 보안을 위협하게 되었다. OAuth 2.0을 구현하고 API를 호출하여 라이브러리를 사용할 때는, 다음의 과정을 통해 SSL/TLS 인증서 체인을 적절하게 다루었는지 검증해야 한다.

- 서버에서 리턴되는 인증서의 호스트 이름과 접근되는 URL 내의 호스트 이름이 일치하는지 검증

10 http://hueniverse.com/2010/09/oauth-2-0-without-signatures-is-bad-for-the-web/
11 사용자가 다른 사용자의 인증서를 얻기 위한 연속된 인증 절차다.
12 http://tools.ietf.org/html/draft-ietf-oauth-v2-threatmodel-00#section-5.1.2

- 인증서 체인 내의 각 인증서가 신뢰받는 인증 기관(CA)에 적절하게 묶이는지 검증
- 서버 내의 인증서가 안전한지, 잠재적 공격에 따라 수정되지는 않는지 확인

1.5.2 OAuth 2.0 요청에 서명하기

API 제공 업체가 서명을 지원하거나 요구할 경우를 대비하여, MAC 접근 인증 스펙[13]에는 클라이언트가 OAuth 2.0 요청에 서명하는 방법이 정의되어 있다.

> **NOTE_** 만약 당신이 MAC을 컴퓨터 유형을 알아내는 용도로만 생각한다면, 왜 많은 개발자에게 서명이 어려운지를 MAC 접근 인증 스펙을 통해 알 수 있을 것이다. 암호 기술이 어렵다면 암호 기술에 대해 잘 설명한 책을 읽어보기 바란다.

Key 얻기

MAC 인증을 사용하여 요청에 서명하려면, 클라이언트는 우선 MAC 키를 얻어야 한다. MAC 키는 OAuth 권한 서버가 발급할 수 있다. OAuth 권한 서버가 키를 발급할 경우, 발급된 MAC 키는 권한 서버에서 access_token이 리턴될 때마다 리턴된다. MAC 키는 'hmac-sha-1'이나 'hmac-sha-256' 알고리즘을 사용하며, 개발자가 API 제공 업체를 이용해 애플리케이션을 등록할 때처럼 외부 프로세스를 통해 발급될 수도 있다. 발급 방법과 관계없이, 키는 항상 안전한 SSL/TLS 채널상에서 발급되고 기밀이 유지되어야 한다.

API 요청 만들기

서명을 요구하는 OAuth API에 연결할 때, 각 API 요청은 요청의 권한 헤더 authorization header 안에 MAC 서명이 있어야 한다. 이 서명의 생성 과정은 정상적인 요청 문자열(nonce, HTTP 메서드, 요청 URI, 호스트, 포트, 임의의 body hash 등)을 생성하고 암호화 서명을 수행한다. 필요하다면 OAuth MAC 서명을 다루

13 http://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-00

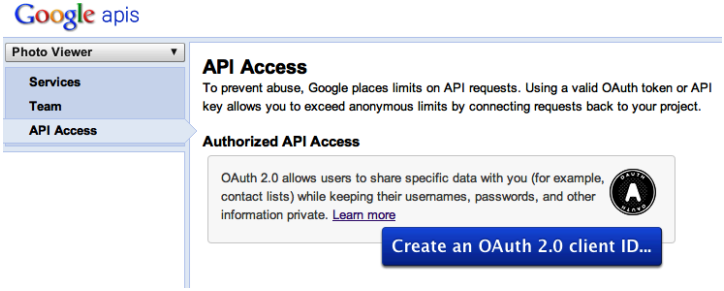
는 라이브러리 사용을 적극 추천한다. 사용자 라이브러리를 제작한다면, MAC 접근 인증 스펙[14]을 참조하라. 이 책에서는 관련 내용을 설명하지 않는다.

1.6 개발자와 애플리케이션 등록

OAuth는 API 요청을 식별하기 위해, 애플리케이션 서버를 이용해서 애플리케이션을 등록하기 원한다. OAuth는 자동화된 방법을 사용하여 애플리케이션을 등록하는 것을 허용하지만, 대다수의 API 제공 업체들은 개발자 웹 사이트에서 양식을 통해 수동으로 등록하길 원한다. 다음은 이 글을 쓰는 시점에 API 제공 업체들이 선호하는 애플리케이션 등록 방법이다.

- 구글은 그림 1-2처럼 API 콘솔[15]을 통해 클라이언트를 등록하길 요구한다.
- Microsoft Windows Live는 애플리케이션 관리 웹 사이트[16]를 사용하여 클라이언트를 등록하길 요구한다.
- 페이스북은 페이스북 개발자 웹 사이트[17]에서 클라이언트를 등록하길 요구한다.

그림 1-2 OAuth 애플리케이션 등록을 위한 구글의 API 콘솔



14 http://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-00#section-3
15 http://code.google.com/apis/console
16 https://manage.dev.live.com/
17 https://developers.facebook.com/apps

예를 들어, 다음은 구글의 API 콘솔을 통해 OAuth 클라이언트를 등록할 때 요구되는 정보다.

- 구글 계정
- 제품 이름
- 제품 로고(옵션)
- 리다이렉트 URI(웹 애플리케이션에만 적용)

등록이 완료되면 개발자에게 클라이언트의 인증서가 발급된다.

Client ID

자원 서버와 통신할 때 'client_id'를 사용한다.

Client Secret

권한 코드를 액세스 토큰Access Token으로 교환하고 서버사이드 웹 애플리케이션 플로우Web Application Flow를 사용하여 액세스 토큰을 재발급할 때는 'client_secret을 사용한다(그림 7-1 참조).

1.6.1 등록은 왜 필요한가?

애플리케이션을 등록하면 개발자에게 클라이언트 인증서를 준다. 이 인증서는 클라이언트의 요청을 권한 서버에서 인증할 때 사용한다. 또한 권한 코드를 액세스 토큰으로 교환하거나 액세스 토큰을 재발급할 때처럼, 요청의 신뢰성을 보장하는 데 사용된다(2장 참조) .

권한 처리 과정에서 사용자 경험을 향상하기 위해 등록은 API 제공 업체 정보를 제공한다. 데이터 액세스를 위해 사용자에게 애플리케이션의 요청을 보낼 때, API 제공 업체는 애플리케이션의 이름과 로고를 종종 표시할 것이다.

구글이 승인 화면에서 등록 정보를 사용하는 예는 그림 2-3에서 볼 수 있다.

1.7 클라이언트 프로파일, 액세스 토큰, 권한 플로우

OAuth의 초기 버전은 전통적인 클라이언트/서버 웹 애플리케이션의 API 권한을 다루기 위해 설계되었다. 이 스펙은 모바일/데스크톱/자바스크립트 애플리케이션과 브라우저 확장 등에 대한 권한을 다루는 방법을 정의하지 못했다. 그래서 OAuth 1.0으로 제작한 애플리케이션은 구현 메서드가 일관적이지 않았다. 이 때문에 OAuth 1.0으로 애플리케이션을 제작하는 방법이 종종 차선책으로 사용되었던 반면, OAuth 2.0은 다양한 사용 사례들을 고려하여 설계되었다.

1.7.1 클라이언트 프로파일

OAuth 2.0은 몇 가지 중요한 클라이언트 프로파일을 정의한다.

서버사이드 웹 애플리케이션

OAuth 클라이언트는 웹 서버에서 실행된다. 웹 애플리케이션은 자원 소유자(사용자)가 접근할 수 있으며 서버 프로그래밍 언어를 사용하여 적절한 API를 호출한다. 사용자는 OAuth client secret이나 권한 서버가 발급한 어떠한 액세스 토큰에도 접근하지 않는다.

웹 브라우저에서 실행되는 클라이언트사이드 애플리케이션

사용자의 웹 브라우저에서 실행되는 OAuth 클라이언트는 애플리케이션 코드와 API 요청에 접근할 수 있다. 애플리케이션은 웹 페이지 안의 자바스크립트나 웹 브라우저 확장 형태로 배포될 수 있고, 플래시와 같은 플러그인 기술을 사용하여 배포될 수도 있다. OAuth 인증서는 자원 소유자의 기밀을 유지하는 데 신뢰받지 못한다. 이 때문에 일부 API 제공 업체는 OAuth 클라이언트 프로파일을 사용하여 client secrets를 발급하지 않을 것이다.

네이티브 애플리케이션

OAuth 클라이언트는 인증서의 기밀을 유지하는 데 신뢰받지 못한다는 점에서

클라이언트사이드 애플리케이션과 매우 흡사하다. 로컬에 설치된 네이티브 애플리케이션은 웹 브라우저의 일부 기능에 국한될 수 있다.

1.7.2 액세스 토큰

OAuth 2.0을 반영한 대부분의 API는 허가된 요청을 만드는 데 전달 토큰만 있으면 된다. 전달 토큰은 간단한 토큰 값으로 보호되는 자원에 접근할 수 있게 해주는 액세스 토큰의 한 종류다. API 호출을 만드는 데 필요한 암호 키 같은 추가 정보는 없다.

서버사이드와 클라이언트사이드 웹 애플리케이션, 또는 네이티브 애플리케이션을 제작할 때 OAuth를 사용하는 최종 목표는 같다. 그것은 사용자나 애플리케이션 대신 API 요청을 수행하기 위해 OAuth 액세스 토큰을 얻는 것이다.

액세스 토큰을 얻은 후, 그것을 요청에 실어 보낼 수 있다. 액세스 토큰을 요청에 실어 보내는 다양한 방법 중 HTTP 권한 헤더에 액세스 토큰을 포함하는 방법을 선호한다.

```
GET /tasks/v1/lists/@default/tasks HTTP/1.1
Host: www.googleapis.com
Authorization: Bearer ya29.AHES6ZSzX
```

개발자들은 다음과 같은 이유로 HTTP 권한 헤더에 액세스 토큰을 포함하는 방법을 선호한다.

- 헤더는 프록시 서버나 웹 서버 접근 로그에 거의 로그를 남기지 않는다.
- 헤더는 거의 캐시되지 않는다.
- 헤더는 클라이언트에서 요청이 만들어질 때 브라우저 캐시에 저장되지 않는다.

스펙에 다른 메커니즘도 정의되어 있긴 하지만, API 제공업체가 굳이 구현할 필요는 없기 때문에 다음 방법 정도가 가능할 것이다.

질의 파라미터에 액세스 토큰을 포함하는 방법

URL 질의 파라미터에 액세스 토큰을 포함하는 방법은 디버깅이나 라이브러리가 권한 헤더를 수정하기 어려울 때 유용하다. 이 메커니즘은 클라이언트사이드 플로우를 사용할 때와 JSONP 요청 안에 액세스 토큰을 포함할 때 유용하다. 예를 들면, 다음과 같다.

```
https://www.googleapis.com/tasks/v1/lists/@default/tasks?
callback=outputTasks&access_token=ya29.AHES6ZTh00gsAn4
```

인코딩된 Form body 파라미터에 액세스 토큰을 포함하는 방법

이는 애플리케이션이 요청의 권한 헤더를 수정할 수 없을 때 사용하는 대체 메커니즘이다. HTTP body가 정상적으로 보내질 때만 사용되며, "application/x-www-form-urlencoded"의 바디body에 form 파라미터로 추가될 수 있다. 단, Google Tasks API는 이 메커니즘을 지원하지 않는다.

1.7.3 권한 플로우

각 클라이언트 프로파일에는 자원 소유자의 데이터에 접근할 수 있는 권한 획득을 위해 적절한 프로토콜 플로우를 적용해야 한다. OAuth 2.0 프로토콜에는 권한 획득을 위한 네 개의 주요 '허가 유형'과 확장 메커니즘이 정의되어 있다.

권한 코드

이 허가 유형은 서버사이드 웹 애플리케이션에 가장 적절하다. 자원 소유자가 데이터에 대한 접근을 허가하면, URL의 질의 파라미터에 의해 권한 코드와 함께 웹 애플리케이션으로 리다이렉트된다. 권한 코드는 클라이언트 애플리케이션에 의해 액세스 토큰으로 교환된다. 교환은 서버 간에 이루어지며 자원 소유자라도 액세스 토큰을 얻는 것을 막기 위해 client_id와 client_secret을 요구한다. 그리고 갱신 토큰refresh token을 이용하여 장기적인 API 접근을 허용한다.

브라우저 기반 클라이언트사이드 애플리케이션의 암묵적 허가

암묵적 허가는 브라우저에서 실행되는 클라이언트사이드 웹 애플리케이션에 최적화되어 있으며 가장 간단하다. 자원 소유자가 애플리케이션에 접근을 허가하면 새로운 액세스 토큰이 즉시 만들어져서, URL 안의 해시 프래그먼트에 포함되어 애플리케이션으로 전달된다. 애플리케이션은 즉시 해시 프래그먼트에서 액세스 토큰을 추출하고(자바스크립트 사용), API 요청을 만들 수 있다. 이 허가 유형은 허가 과정 중간에 권한 코드가 없어도 된다. 그러나 장기간 지속적으로 접근할 수 있도록 토큰을 재발급하지는 않는다.

자원 소유자의 비밀번호 기반 허가

이 허가 유형은 자원 소유자의 사용자 이름과 비밀번호를 OAuth 액세스 토큰으로 교환하는 것이 가능하다. API 제공 업체가 제작한 모바일 앱처럼 신뢰도가 높은 클라이언트에서만 사용된다. 사용자의 비밀번호가 클라이언트에게 노출되는 동안에도 비밀번호를 장치에 저장하지 않아도 된다. 초기 인증 후에 OAuth 토큰만 저장된다. 비밀번호가 저장되지 않기 때문에 사용자는 비밀번호 변경 없이 앱으로의 접근을 폐기할 수 있으며, 토큰은 데이터의 접근 범위가 제한된다. 그래서 이 허가 유형은 전통적인 사용자 이름/비밀번호 인증보다 개선된 보안을 제공한다.

클라이언트 인증서

클라이언트 인증서 허가 유형은 클라이언트가 자원을 소유하고 있거나 권한이 권한 서버에서 미리 준비되어 있을 때, 애플리케이션이 액세스 토큰을 획득하도록 허용한다. 이 허가 유형은 특정 사용자를 대신하는 것보다 저장 서비스나 데이터베이스를 대신하여 API에 접근하는 애플리케이션에 적절하다.

추가 플로우는 핵심 스펙 외부에 정의되어 있다.

장치 프로파일

장치 프로파일은 내장된 웹 브라우저가 없거나 게임 콘솔, 전자 포토 프레임 같은 제한된 입력 옵션 장치에서 OAuth를 사용할 수 있게 해준다. 일반적으로 사용자는 장치 내 플로우를 초기화하고, 웹 사이트에 접근할 컴퓨터를 선택하며, 장치 내에 표시되는 권한 코드를 삽입해서 접근을 승인받는다. 페이스북[18]은 이 플로우의 좋은 예[19]다.

SAML 전달 주장bearer assertion 프로파일

이 프로파일은 'SAML 2.0 주장'[20]을 OAuth 액세스 토큰으로 교환한다. 애플리케이션과 데이터 접근을 위해 이미 SAML 권한 서버가 설치된 엔터프라이즈 환경에서 유용하다.

18 http://developers.facebook.com/docs/howtos/login/devices/

19 http://oauth-device-demo.appspot.com/

20 주체에 대하여 수행되는 인증 행위, 주체에 대한 속성 정보 또는 명기된 자원에 대하여 주체가 행할 수 있는 인가 데이터 등에 SAML 기관이 생성한 데이터 조각이다. [출처: TTA(http://www.tta.or.kr)]

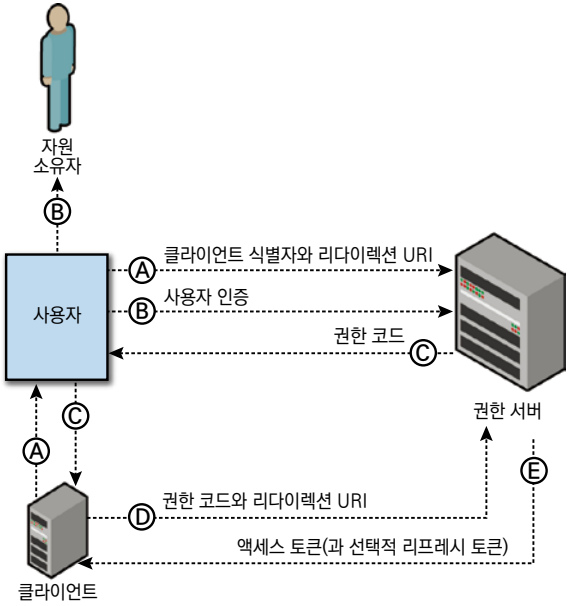
2 | 서버사이드 웹 애플리케이션 플로우

웹 애플리케이션 플로우(권한 코드 플로우Authorization Code Flow라고도 한다)에서 자원 소유자는 애플리케이션에 의해 API 제공 업체의 OAuth 권한 서버로 리다이렉트된다. OAuth 권한 서버는 사용자가 액티브 세션을 가졌는지 검사한 다음, 요청된 데이터에 접근해도 되는지 API 제공 업체에 알린다. 접근이 허가되면 API 제공 업체는 접근을 허용하고 웹 애플리케이션에 리다이렉트한다. 권한 코드는 다음과 같이 code 질의 파라미터로 URL에 포함된다.

```
http://www.example.com/oauth_callback?code=ABC1234
```

code가 질의 파라미터로 전달되기 때문에, 웹 브라우저는 OAuth 클라이언트처럼 활동하는 웹 서버를 통해 code를 보낸다. 이 권한 코드는 애플리케이션에서 권한 서버로의 서버 간 호출을 사용하여 액세스 토큰으로 교환된다. 액세스 토큰은 클라이언트에서 API를 호출하기 위해 사용된다. 그림 2-1은 OAuth 2.0 스펙 내의 다이어그램으로, 단계적 플로우를 설명한다.

그림 2-1 서버사이드 웹 애플리케이션 플로우: 단계적 흐름



2.1 권한 코드 플로우는 언제 사용하는가?

권한 코드 플로우는 다음 경우에 사용한다.

- 장기 접근이 요구될 때
- OAuth 클라이언트가 웹 애플리케이션 서버일 때
- API 호출에 대한 책임이 매우 중요하고 사용자가 접근하는 웹 브라우저에 OAuth 토큰이 노출되지 않아야 할 때

2.2 보안성

권한 코드 플로우는 자원 소유자의 웹 브라우저에 액세스 토큰을 노출하지 않는다. 대신 권한 허가는 웹 브라우저를 통해 전달되는 '권한 코드'라는 매개체를 사용한

다. 이 권한 코드는 API가 호출되기 전에 '액세스 토큰'으로 교환되어야 한다. 클라이언트 보안이 유지되는 한 액세스 토큰의 기밀이 보장되며, 이 교환 과정은 요청에 올바른 client_secret 값이 전달될 때만 성공한다.

3장에서 설명할 암묵적 플로우와 달리, 자원 소유자에게도 액세스 토큰이 노출되지 않는다. 액세스 토큰으로 만드는 API 요청은 OAuth 클라이언트와 해당 개발자들만 직접 관련되어 있다. 가장 중요한 것은 참조 헤더, 자바스크립트, 웹 브라우저 이력 등을 통해 액세스 토큰이 악의적인 코드에 노출되는 위험을 줄인다는 사실이다(액세스 토큰은 결코 웹 브라우저를 통해 전달되지 않는다).

웹 브라우저에 액세스 토큰이 드러나지 않기 때문에 누출의 위험이 적긴 하지만, 권한 코드 플로우를 사용하는 많은 애플리케이션이 데이타에 대한 오프라인offline 접근이 가능하도록 데이터베이스나 키 스토어에 '장기 재발급 토큰'을 저장한다. 이렇게 되면 사용자의 데이터에 접근할 수 있는 통로가 만들어지므로, 애플리케이션이 데이터에 장기간 오프라인 접근을 해야 하는 경우 추가적인 위험이 발생할 수 있다.

데이터베이스나 키 스토어에 '장기 재발급 토큰'을 저장하는 것은 클라이언트사이드 웹 애플리케이션 같은 플로우에서는 존재하지 않는다(3장 참고). 웹 사이트의 애플리케이션 아키텍처는 새로운 액세스 토큰을 획득하기 위해 사용자 웹 브라우저와 상호작용하기 어렵기 때문에, 많은 웹 사이트는 추가적인 위험이 있더라도 오프라인 데이터 접근을 선택할 것이다.

2.3 사용자 경험

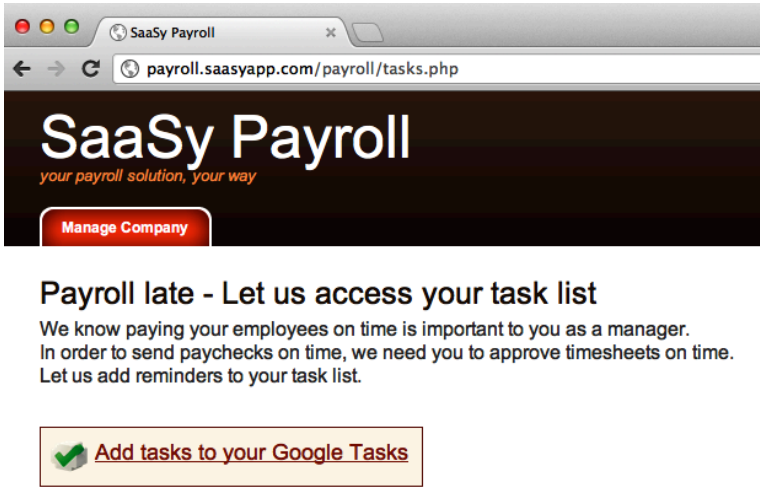
'급여 애플리케이션payroll application'의 예를 들어보자. 급여 애플리케이션은 관리자가 제때 직원들의 출퇴근 기록을 승인할 수 있도록 관리자 업무 리스트의 수정 권한에 접근하기를 원한다. 관리자가 매일 사용하는 업무 리스트에 알림기능을 추가

하면, 월급이 제때 입금되지 않아서 화를 내는 직원의 수나 인사부서에 전화하는 시간 낭비가 훨씬 감소할 것이다.

일반적인 사용자 경험은 매우 간단하다.

1. 급여 애플리케이션은 관리자에게 작업의 수정 권한에 대한 접근을 요청한다. 그러고 나서 업무 리스트 애플리케이션의 OAuth 권한 서버로 관리자를 리다이렉트한다(그림 2-2 참조) .

그림 2-2 급여 애플리케이션은 업무 리스트 애플리케이션의 승인 화면으로 다이렉트되는 것을 사용자에게 알린다.



2. OAuth 권한 서버는 사용자에게, 업무 리스트 애플리케이션의 API를 사용해서 급여 애플리케이션이 관리자의 작업을 업데이트하도록 권한을 허가할지 묻는다(그림 2-3 참조).