



What the Heck is OAuth and OIDC?

Matt Raible | [@mraible](https://twitter.com/mraible)

October 25, 2018

Photo by [l1mey](https://flickr.com/photos/l1mey) flickr.com/photos/l1mey/4510426390



okta



Who is Matt Raible?

Father, Skier, Mountain Biker,
Whitewater Rafter

Open Source Connoisseur

Web Developer and Java Champion

Okta Developer Advocate



Bus Lover

Blogger on raibledesigns.com and
developer.okta.com/blog







PHILLIPS
66



developer.okta.com

Authentication Standards



What about You?

Have you heard of OAuth or OIDC?

Do you consider yourself a developer?

Have you ever written authentication
from scratch?

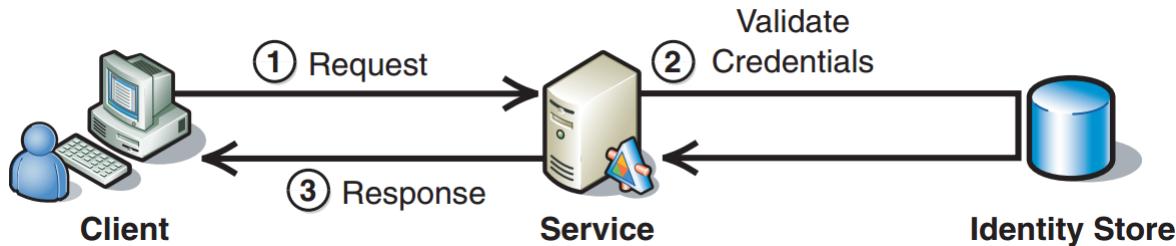
Have you implemented OAuth or OIDC?

Why are you here?



OAuth 2.0 Overview

Web Authentication



Authentication Required
http://localhost:5000

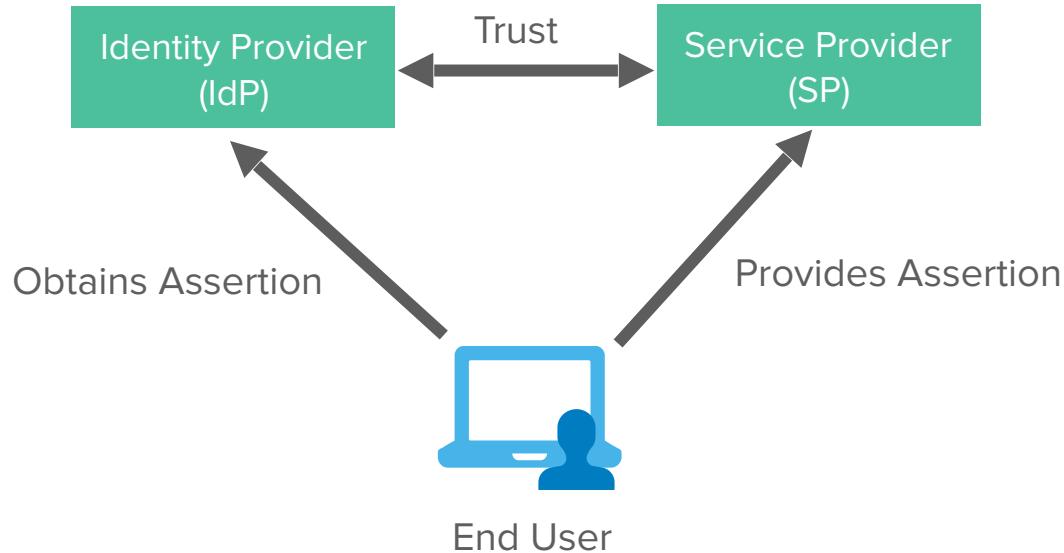
Username

Password

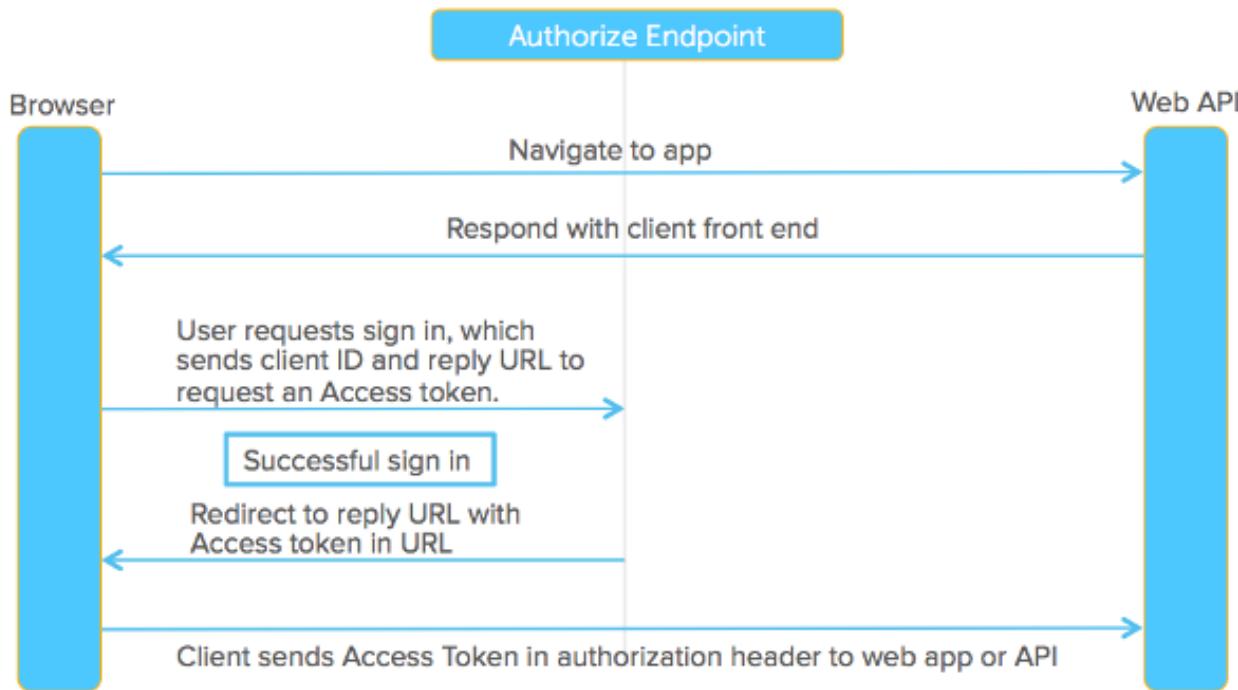
GET /index.html HTTP/1.1
Host: www.example.com
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==



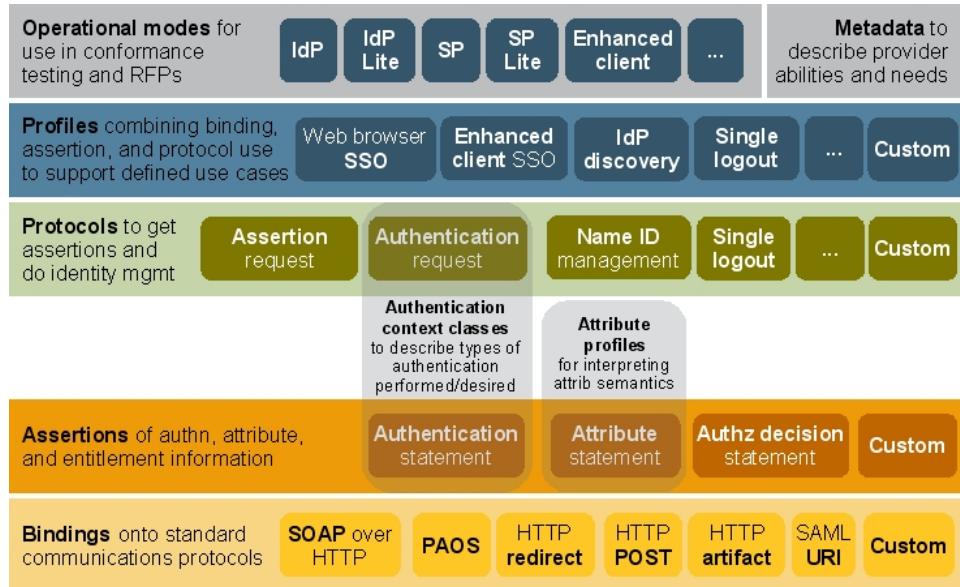
Federated Identity



How Federated Identity Works



SAML 2.0



Authentication Request Protocol

Assertion

OASIS Standard, 15 March 2005

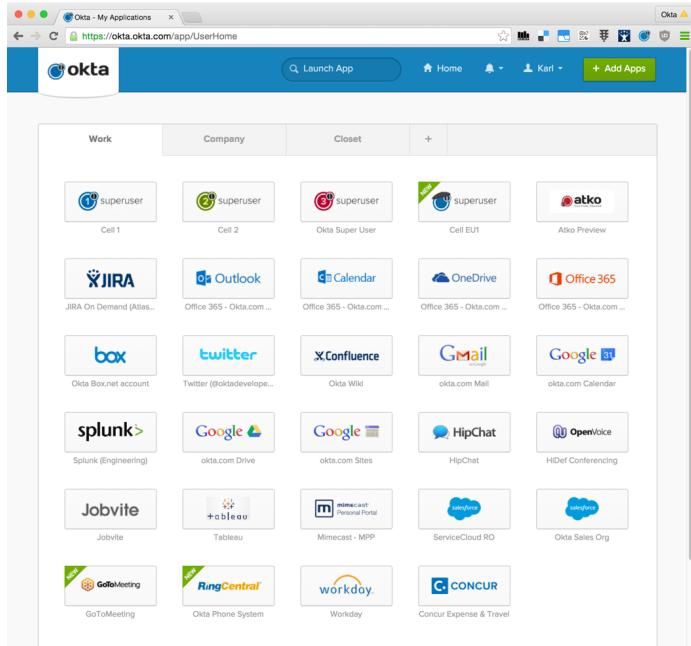


SAML 2.0 Assertion

```
<Assertion xmlns="urn:oasis:names:tc:SAML:2.0:assertion" ID="b07b804c-7c29-ea16-7300-4f3d6f7928ac" Version="2.0"
IssueInstant="2004-12-05T09:22:05"
<Issuer>https://example.okta.com</Issuer>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</ds:Signature>
<Subject>
  <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:unspecified">
    matt@example.com
  </NameID>
  <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
  </SubjectConfirmation>
</Subject>
<Conditions NotBefore="2004-12-05T09:17:05" NotOnOrAfter="2004-12-05T09:27:05">
  <AudienceRestriction>
    <saml:Audience>https://sp.example.com/saml2/sso</saml:Audience>
  </AudienceRestriction>
</Conditions>
<AuthnStatement AuthnInstant="2004-12-05T09:22:00" SessionIndex="b07b804c-7c29-ea16-7300-4f3d6f7928ac">
  <AuthnContext>
    <AuthnContextClassRef>
      urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
    </AuthnContextClassRef>
  </AuthnContext>
</AuthnStatement>
<AttributeStatement>
  <Attribute Name="displayName">
    <AttributeValue>Matt Raible</AttributeValue>
  </Attribute>
</AttributeStatement>
</Assertion>
```

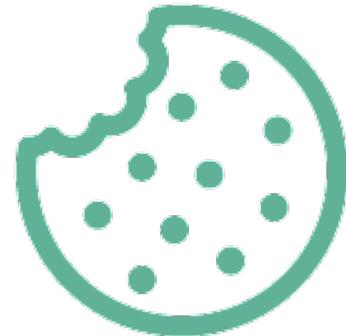


SAML = Web SSO



The screenshot shows the Okta My Applications dashboard. At the top, there's a search bar labeled "Launch App", a "Home" button, a user dropdown for "Karl", and a "+ Add Apps" button. Below the header, there are four tabs: "Work" (selected), "Company", "Closet", and a "+" button. The main area displays a grid of application icons:

- Row 1: superuser (Cell 1), superuser (Cell 2), superuser (Okta Super User), superuser (Cell EU!), atko (atko Preview)
- Row 2: JIRA (JIRA On Demand (Attest...)), Outlook (Office 365 - Okta.com ...), Calendar (Office 365 - Okta.com ...), OneDrive (Office 365 - Okta.com ...), Office 365 (Office 365 - Okta.com ...)
- Row 3: Box (Okta Box.net account), Twitter (Twitter (@oktadevelop...)), Confluence (Okta Wiki), Gmail (okta.com Mail), Google Calendar (okta.com Calendar)
- Row 4: Splunk (Splunk (Engineering)), Google Drive (okta.com Drive), Google Sites (okta.com Sites), HipChat (HipChat), OpenVoice (H2Def Conferencing)
- Row 5: Jobvite (Jobvite), Tableau (+ableau), Mimecast (Mimecast - MPP), ServiceCloud RO (ServiceCloud RO), Okta Sales Org (Okta Sales Org)
- Row 6: GoToMeeting (GoToMeeting), RingCentral (RingCentral Phone System), Workday (Workday), Concur (Concur Expense & Travel)







What's
Changed
Since
2005?



Confusion



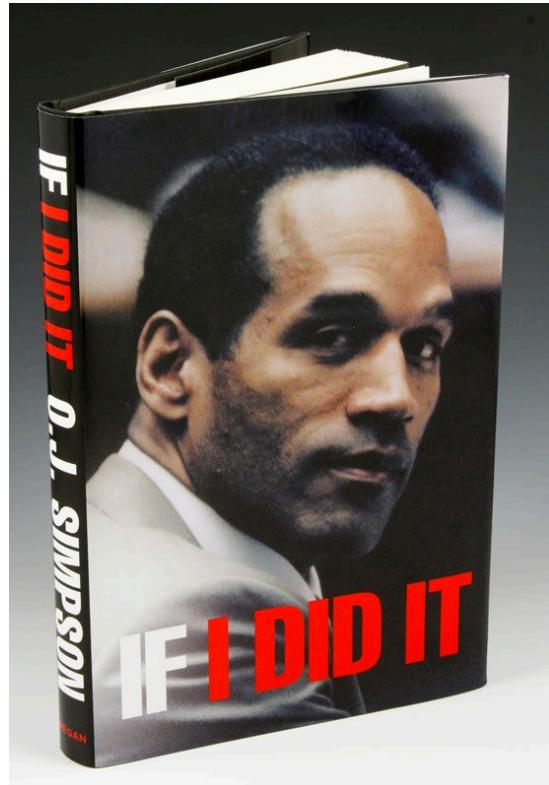
Identity Use Cases (circa 2006)

Simple login — basic, forms, and cookies

Single sign-on across sites — SAML

Mobile app login — N/A

Delegated authorization — N/A



The Delegated Authorization Problem

How can you let a website access your data
(without giving it your password)?



Don't do it this way!

Are your friends already on Yelp?

Many of your friends may already be here, now you can find out. Just log in and we'll display all your contacts, and you can select which ones to invite! And don't worry, we don't keep your email password or your friends' addresses. We loathe spam, too.

Your Email Service

 msn Hotmail  YAHOO! MAIL  AOL Mail  Gmail

Your Email Address

(e.g. bob@gmail.com)

Your Gmail Password

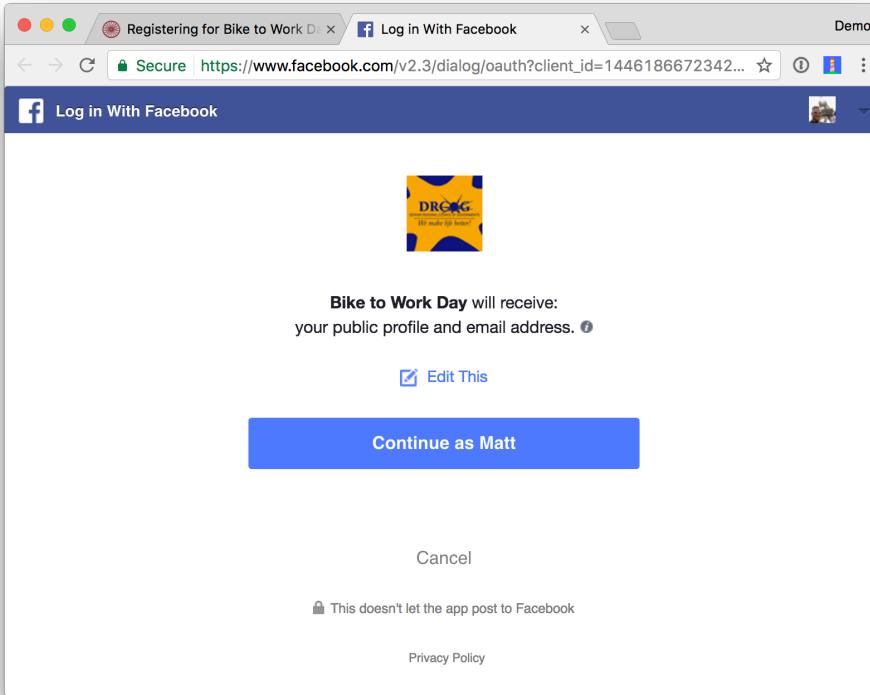
(The password you use to log into your Gmail email)

[Skip this step](#)

Check Contacts



Have you ever seen one of these?





Hotel Key Cards, but for Apps



Hotel Key Cards, but for Apps



OAuth Authorization Server



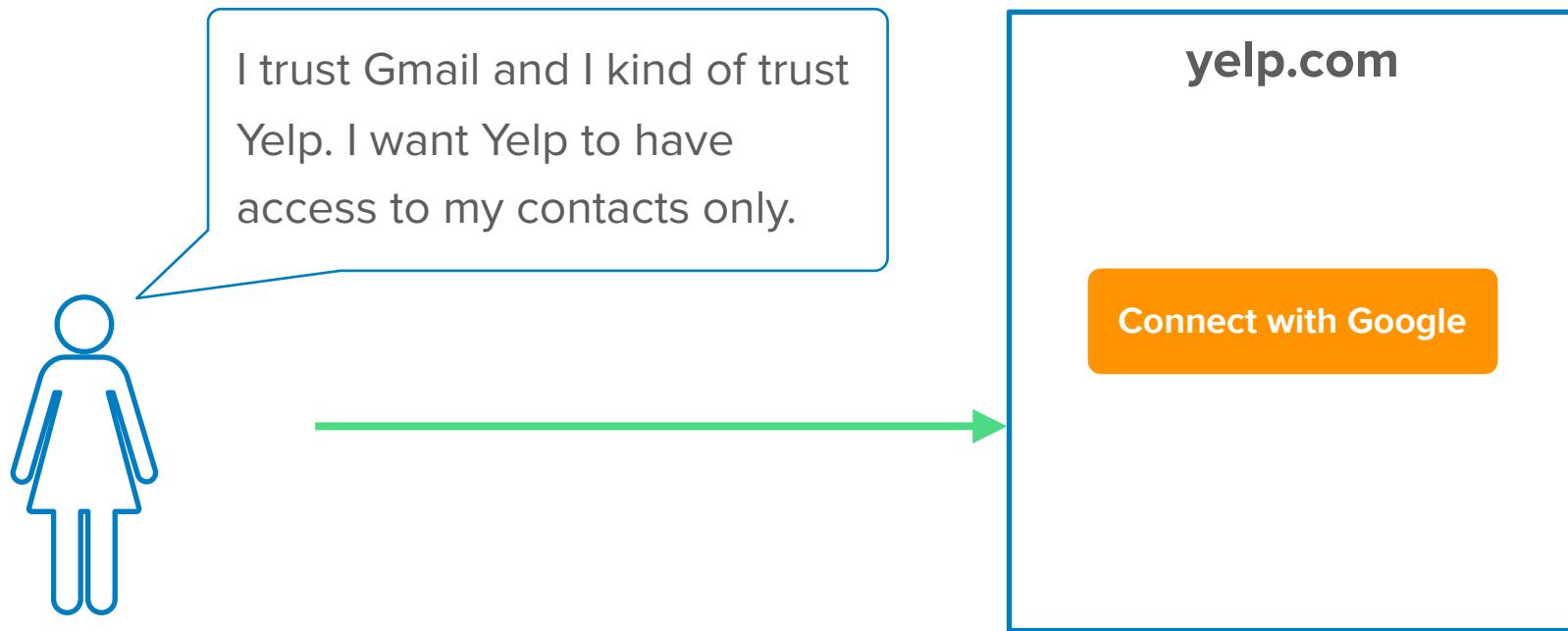
Access Token



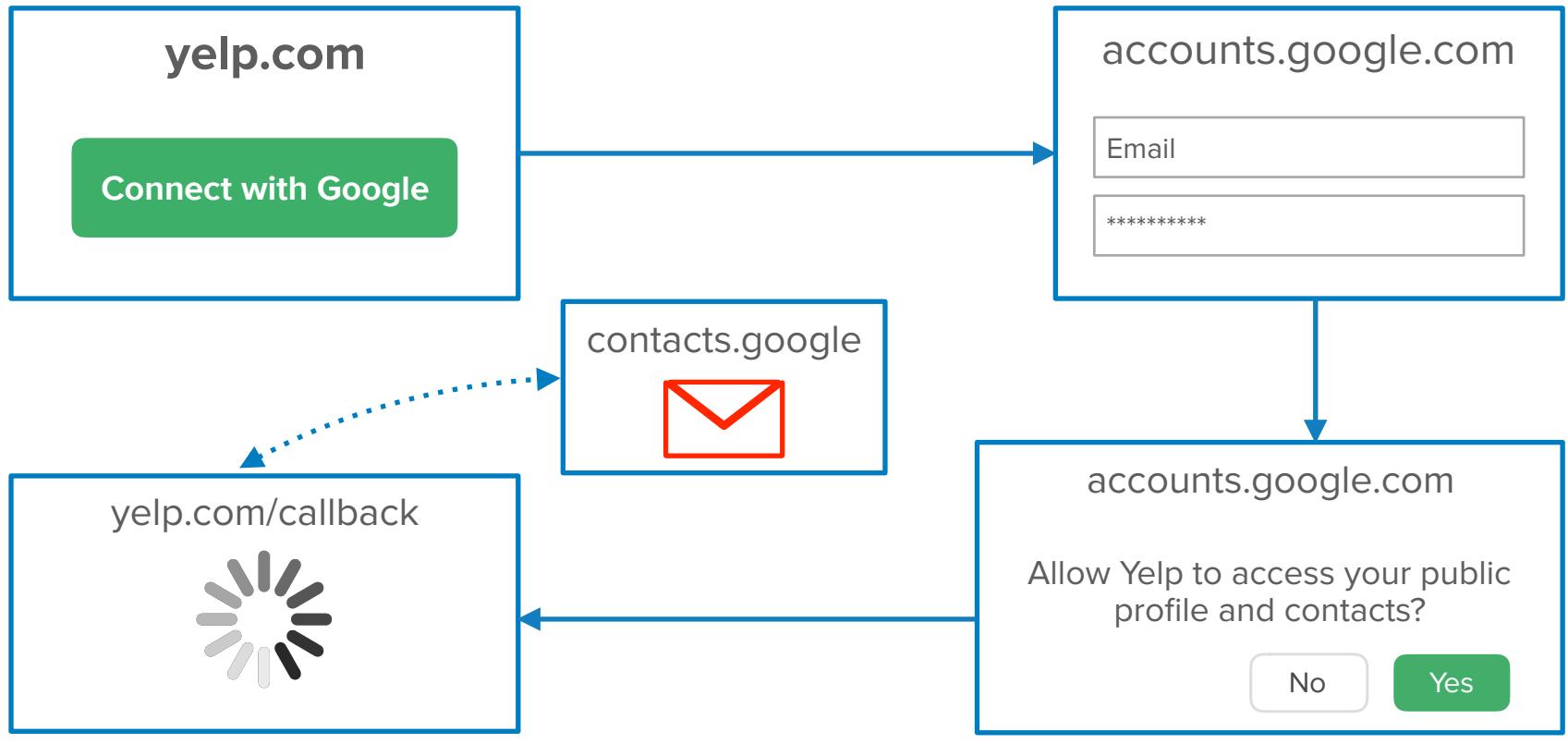
Resource (API)



Delegated Authorization with OAuth 2.0



Delegated Authorization with OAuth 2.0



OAuth 2.0 Terminology

Actors

Clients

Authorization Server

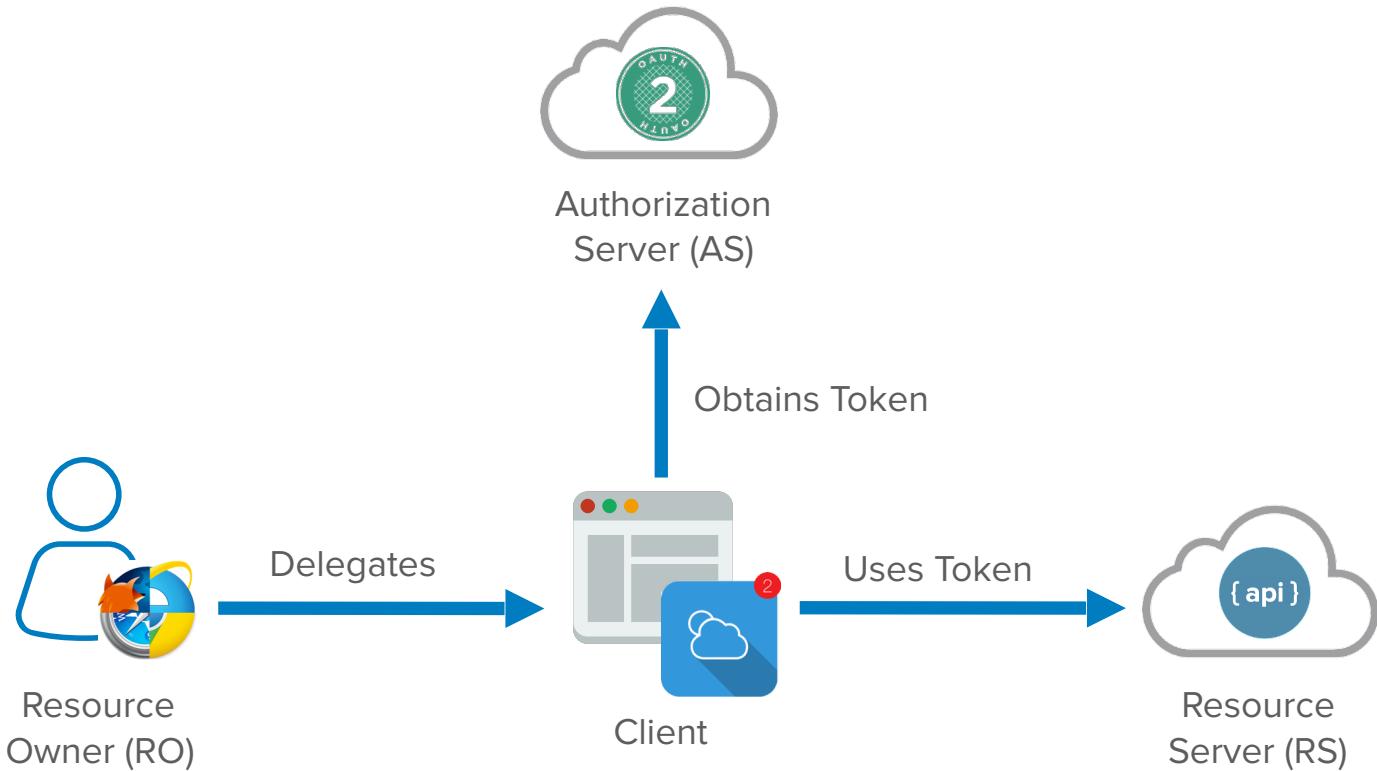
Resource Server

Access Tokens

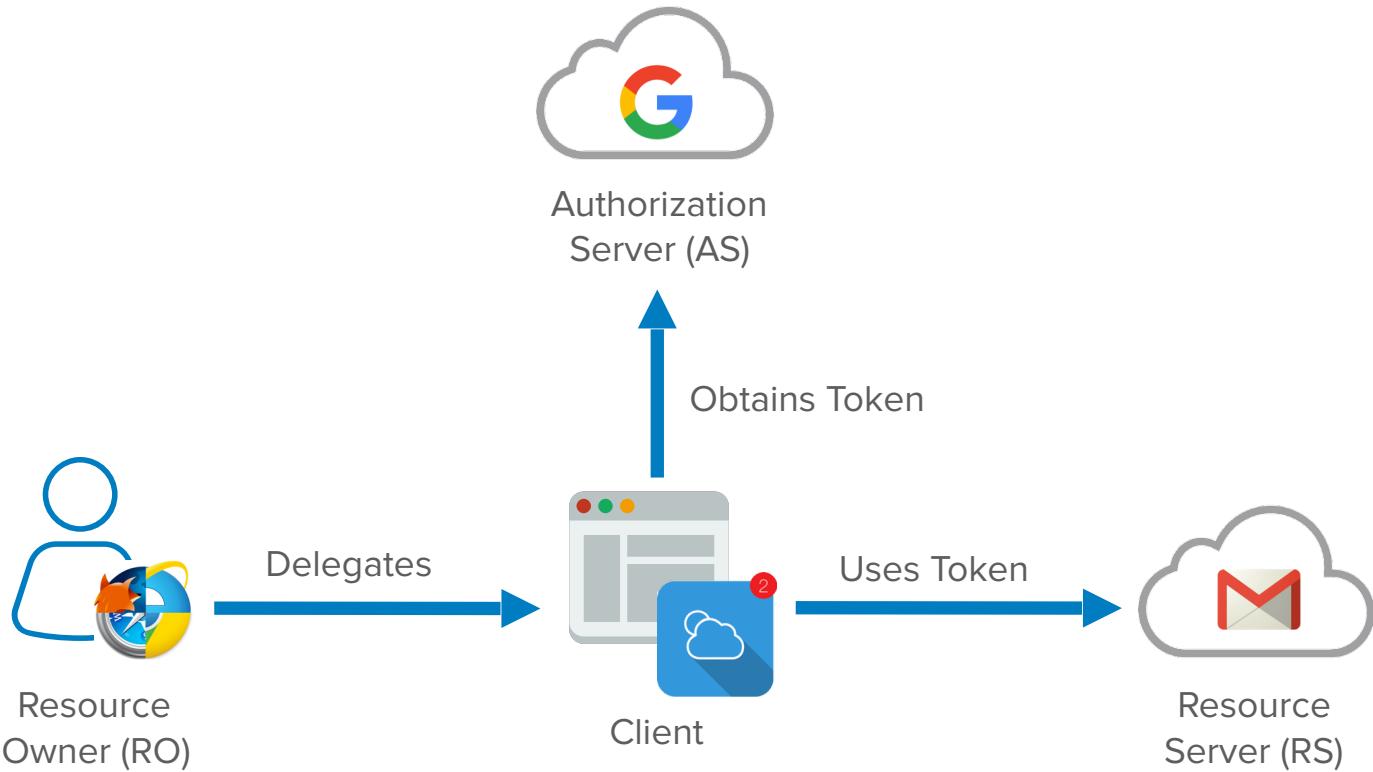
Redirect URI



Actors



Actors

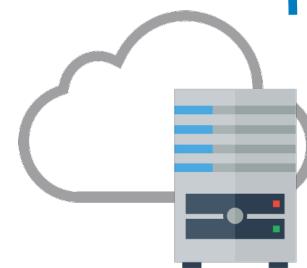


Clients

Public
(Client Identification)



Confidential
(Client Authentication)



Clients

Client Registration is the DMV of OAuth



Create client ID

Application type

Web application
 Android [Learn more](#)
 Chrome App [Learn more](#)
 iOS [Learn more](#)
 PlayStation 4
 Other

Name

My Web App

Authorized JavaScript origins

Enter JavaScript origins here or redirect URIs below (or both) [?](#)
Cannot contain a wildcard (`http://*.example.com`) or a path (`http://example.com/subdir`).

`http://www.example.com`

Authorized redirect URIs

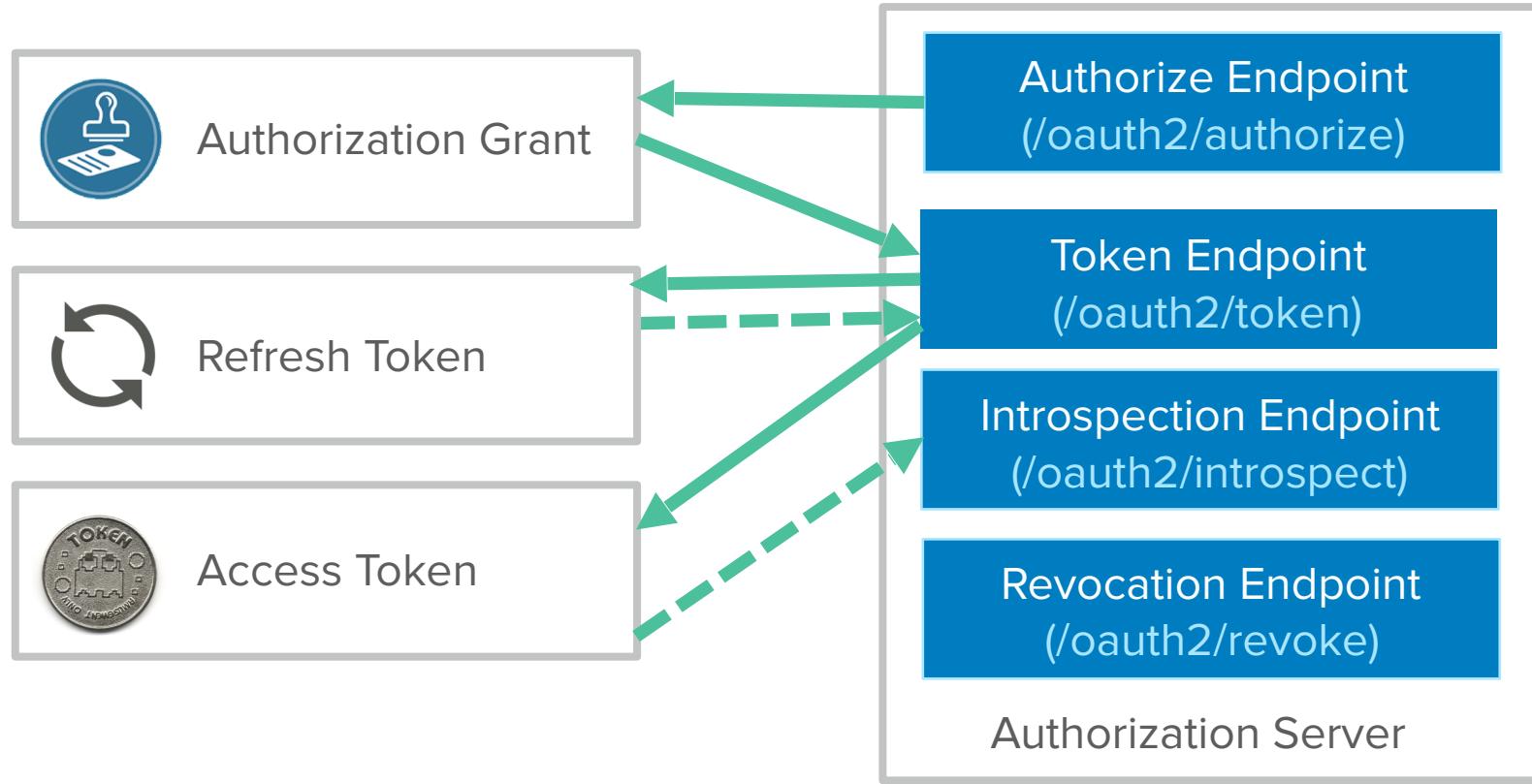
Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

`http://www.example.com/oauth2callback`

[Create](#) [Cancel](#)



Authorization Server



Tokens



Access Token (Required)

- Short-lived token used by **Client** to access **Resource Server** (API)
- Opaque to the **Client**
- **No client authentication required (Public Clients)**
- Optimized for scale and performance
- Revocation is dependent on implementation

Refresh Token (Optional)

- Long-lived token that is used by **Client** to obtain new access tokens from **Authorization Server**
- **Usually requires Confidential Clients with authentication**
- Forces client to rotate secrets
- Can usually be revoked

OAuth doesn't define the format of a token!



Access Token Types

Self-encoded tokens

Protected, time-limited data structure agreed upon between Authorization Server and Resource Server that contains metadata and claims about the identity of the user or client over the wire.

Resource Server can validate the token locally by checking the signature, expected issuer name and expected audience or scope.

Commonly implemented as a signed JSON Web Tokens (JWT)

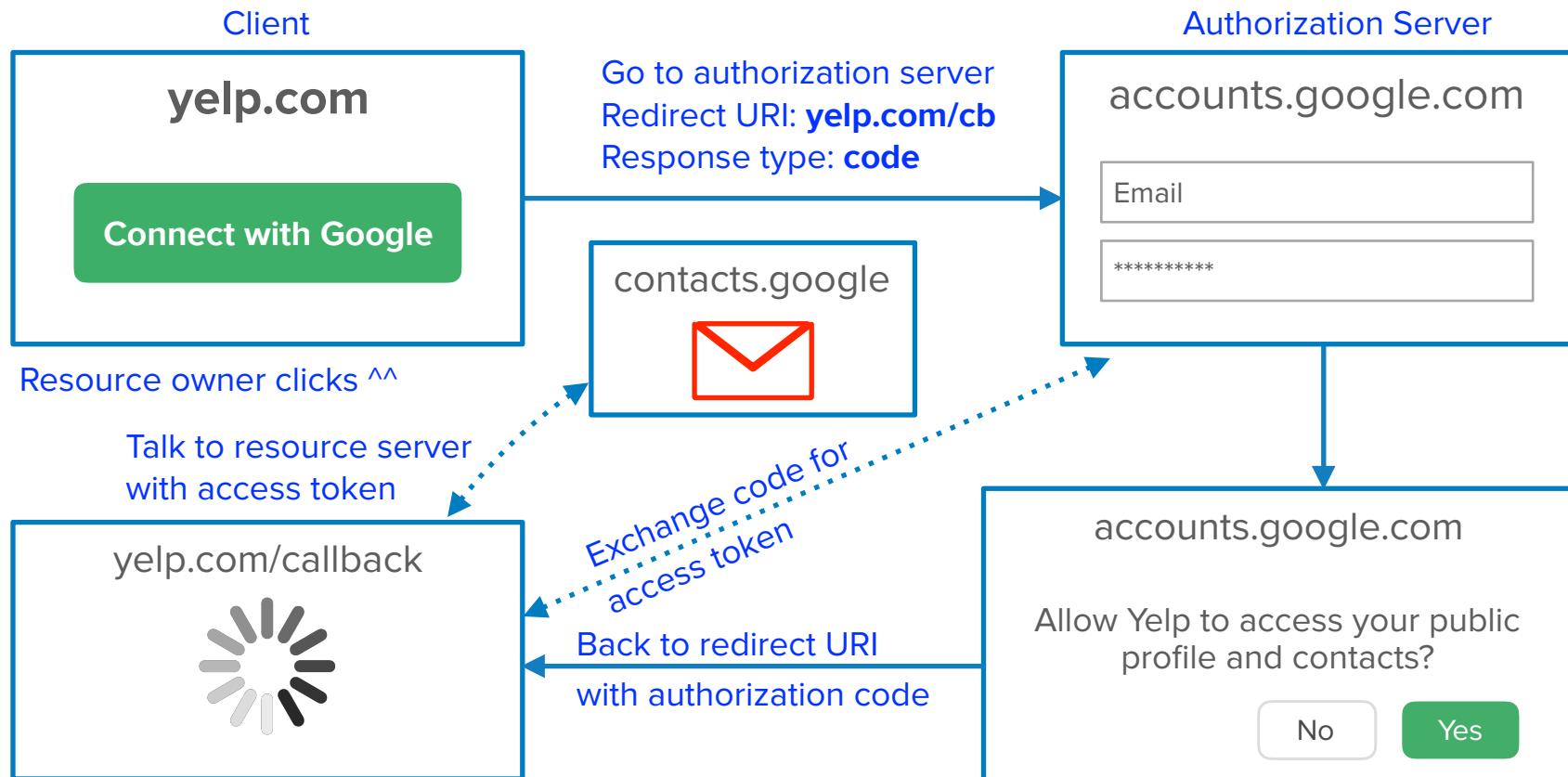
Reference tokens (aka opaque tokens)

Infeasible-to-guess (secure-random) identifier for a token issued and stored by the OAuth 2.0 Authorization Server

Resource Server must send the identifier via back-channel to the OAuth 2.0 Authorization Server's token introspection endpoint to determine if the token is valid and obtain claims/scopes



OAuth 2.0 Authorization Code Flow



More OAuth 2.0 Terminology

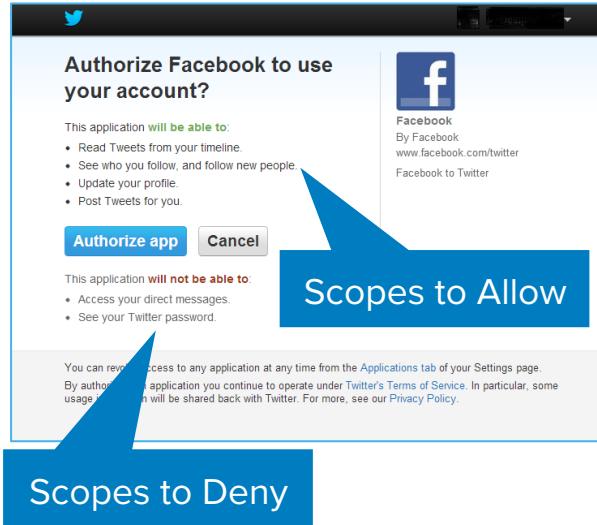
Scopes

Consent

Grants



Scopes



Additive bundles of permissions asked by client when requesting a token

Decouples authorization policy decisions from enforcement

Who owns the data? End user or the target service

Who gets to specify the authorization policy? End user or application owner



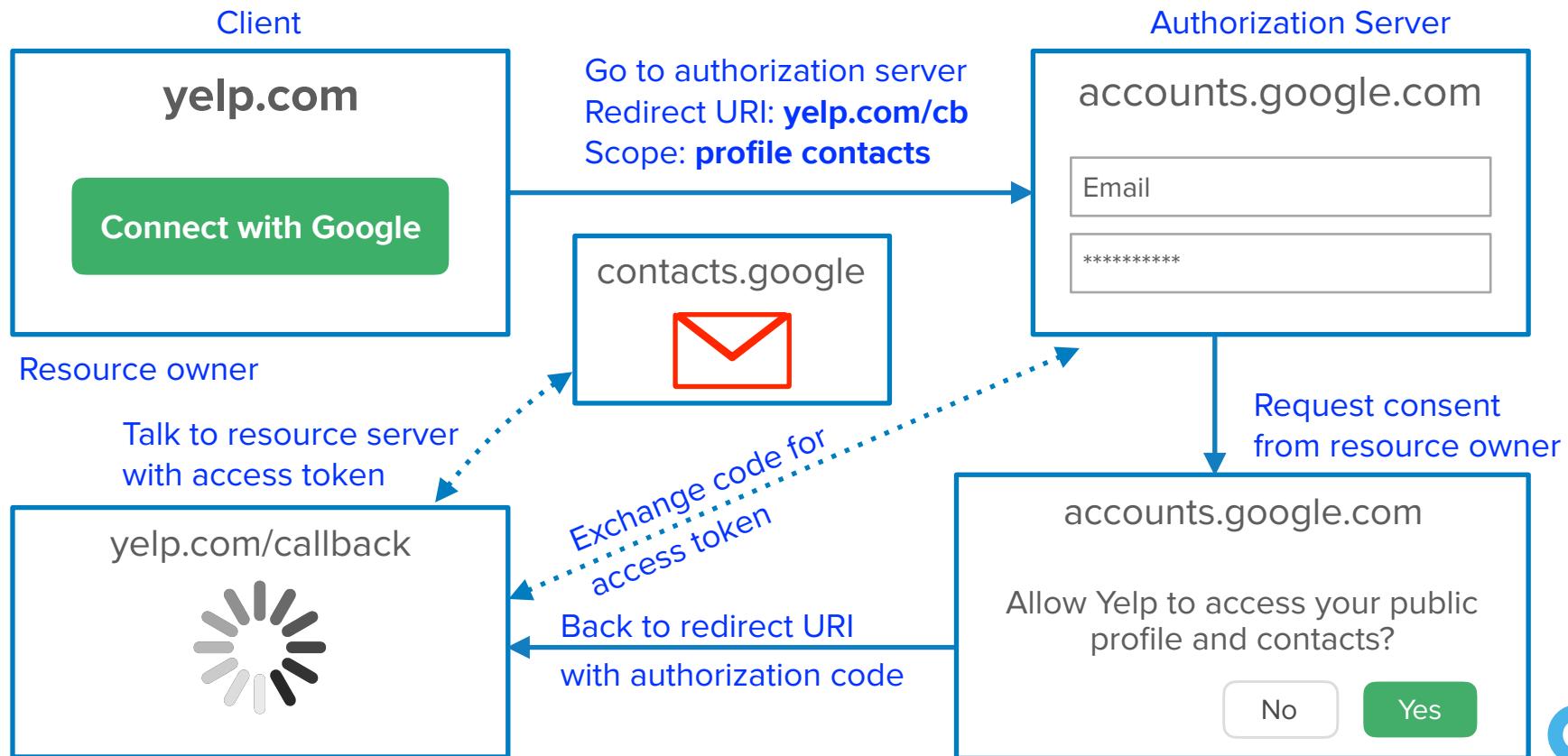
Capturing User Consent

Authorization Grant (Trust of First Use)

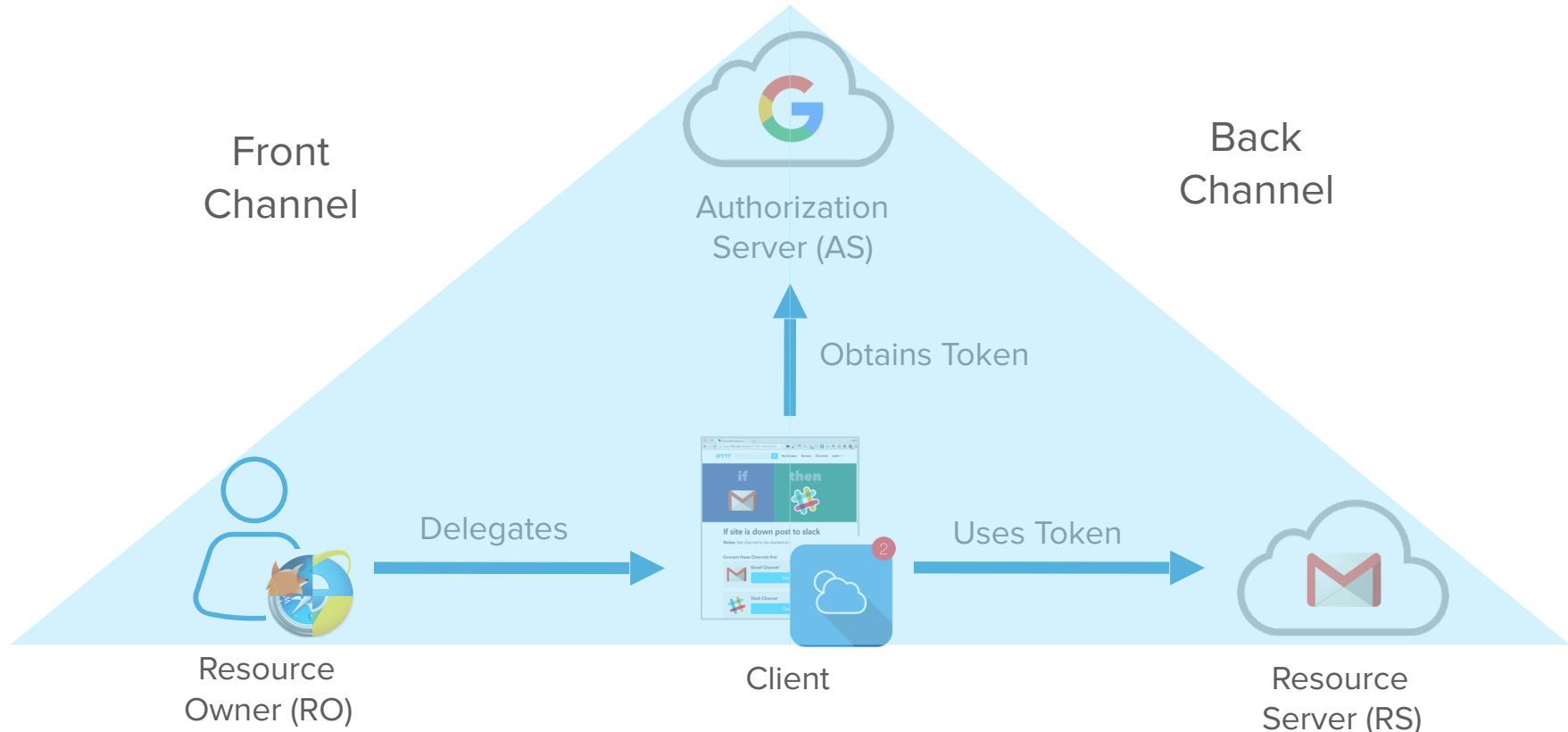
The image displays three screenshots illustrating the process of capturing user consent through authorization grants:

- Mobile App Authorization Screen:** A screenshot of a mobile application interface showing the "Authorize" screen. It asks for permission to access the account for various durations: "For 1 year", "For 30 days", "For 1 week", and "For 1 day". The "For 30 days" option is selected. Below the duration options are two buttons: "Authorize" (in green) and "Decline".
- Evernote Account Settings Screen:** A screenshot of the Evernote web interface under "Account & Privacy". It shows a list of devices and accounts granted access to the user's account:
 - Karl's iPhone: Grants access to apps on this iOS device
 - Google Chrome: Has full access to your Google Account
 - Asana: Has access to Google Contacts, basic account info
 - Google Voice iOS: Has access to Google Contacts, Google Voice, basic account info
 - Project Default Service Account: Has access to Google Services
 - Sunrise Calendar: Has access to Google Calendar, Google Contacts, basic account info
 - feedly: Has access to basic account info
 - PBS: Has access to basic account info
 - Postman: Has access to basic account info
 - Sharelock: Has access to basic account infoA note at the bottom states: "You may revoke access at any time by going to Applications in your account settings."
- LinkedIn Permission Request Screen:** A screenshot of the LinkedIn permission request interface. It shows a summary of the requested permissions:
 - YOUR FULL PROFILE:** Full profile including experience, education, skills, and recommendations
 - YOUR EMAIL ADDRESS:** The primary email address you use for your LinkedIn account
 - YOUR CONNECTIONS:** Your 1st and 2nd degree connections
 - NETWORK UPDATES:** Retrieve and post updates to LinkedIn as you
 - INVITATIONS AND MESSAGES:** Send messages and invitations to connect as youA large button at the bottom says "Allow access" (in blue), and there is also a "Cancel" button.

OAuth 2.0 Authorization Code Flow



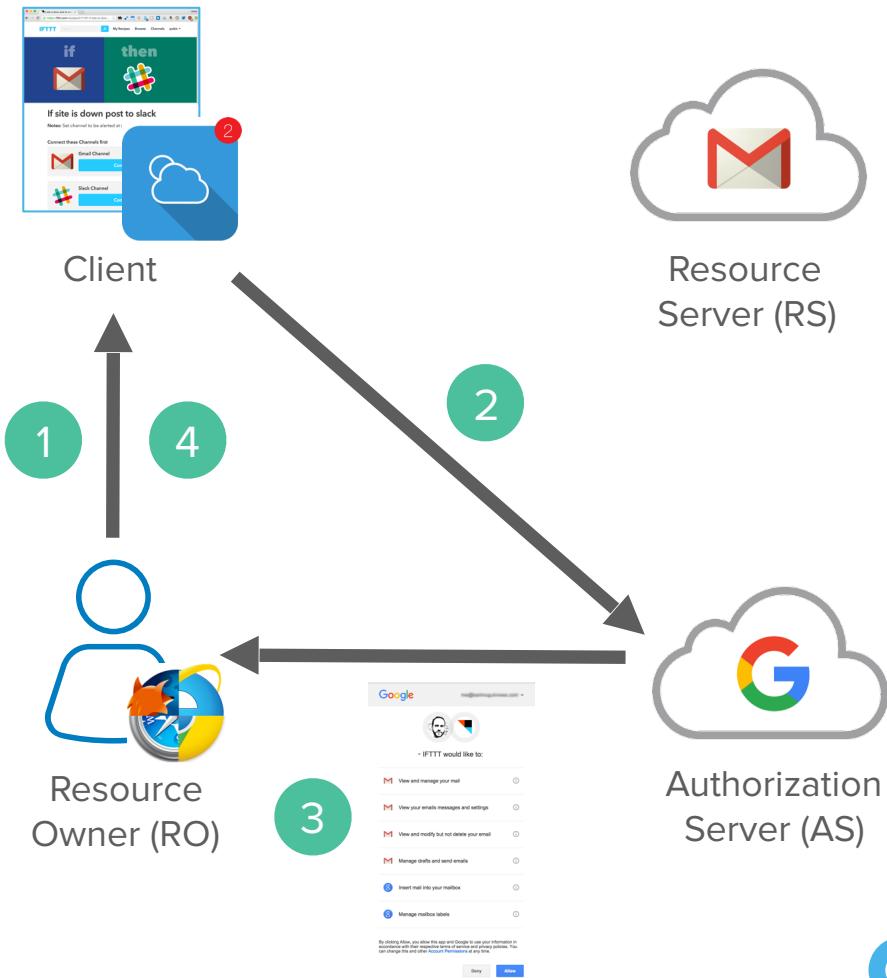
Flow Channels



Front Channel Flow

Authorize via User Agent

- 1 Resource Owner starts flow to delegate access to protected resource
- 2 Client sends authorization request with desired scopes via browser redirect to Authorize Endpoint on Authorization Server
- 3 User authenticates and consents to Delegated Access (Grant)
- 4 Authorization Code Grant or Access Token is returned to Client via browser redirect



Authorization Request

Request

```
GET https://accounts.google.com/o/oauth2/auth?  
scope=gmail.insert gmail.send&  
redirect_uri=https://app.example.com/oauth2/callback&  
response_type=code&  
client_id=812741506391&  
state=af0ifjsldkj
```

Response

```
HTTP/1.1 302 Found  
Location: https://app.example.com/oauth2/callback?  
code=MsCeLvIaQm6bTrgtp7&  
state=af0ifjsldkj
```

Note: Parameters are not URL-encoded for example purposes

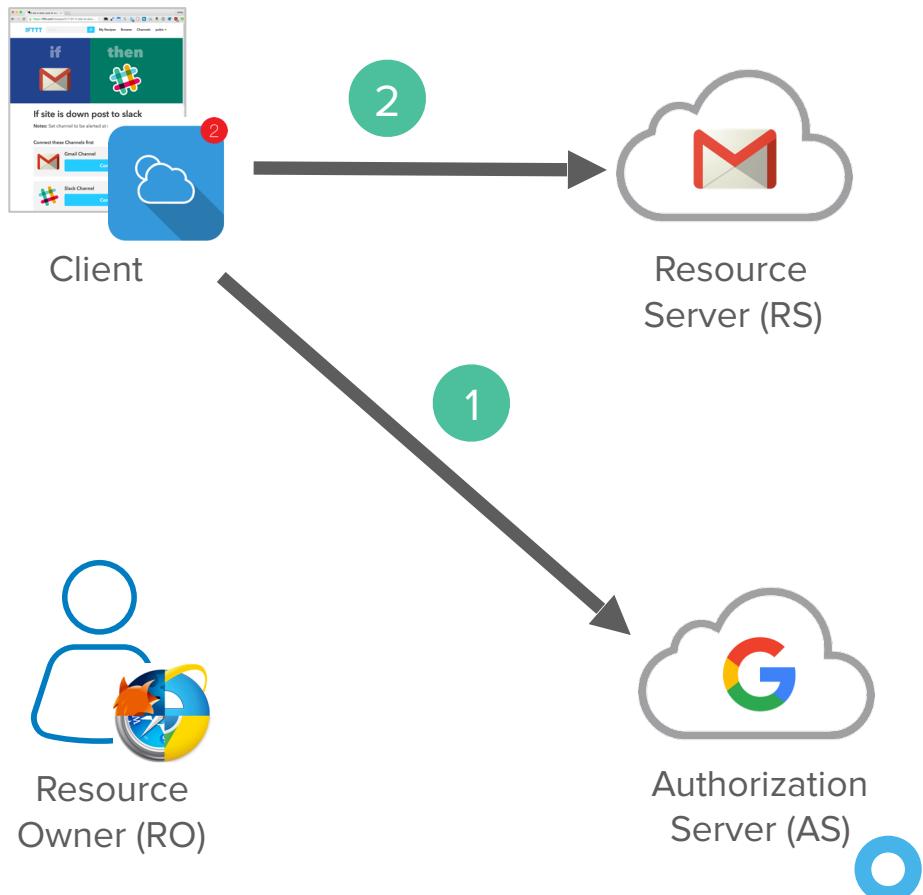


Back Channel Flow

Exchange Grants for Tokens

1 Client exchanges **Authorization Code Grant** with token endpoint on Authorization Server for an **Access Token** and optionally **Refresh Token**

2 Client accesses protected resource with **Access Token**



Token Request

POST /oauth2/v3/token HTTP/1.1

Host: www.googleapis.com

Content-Type: application/x-www-form-urlencoded

code=MsCeLvIaQm6bTrgtp7&

client_id=812741506391&

client_secret={client_secret}&

redirect_uri=https://app.example.com/oauth2/callback&

grant_type=authorization_code

Note: Parameters are not URL-encoded for example purposes



Token Response

```
{  
    "access_token": "2YotnFZFEjr1zCsicMWpAA",  
    "token_type": "Bearer",  
    "expires_in": 3600,  
    "refresh_token": "tGzv3J0kF0XG5Qx2T1KWIA"  
}
```

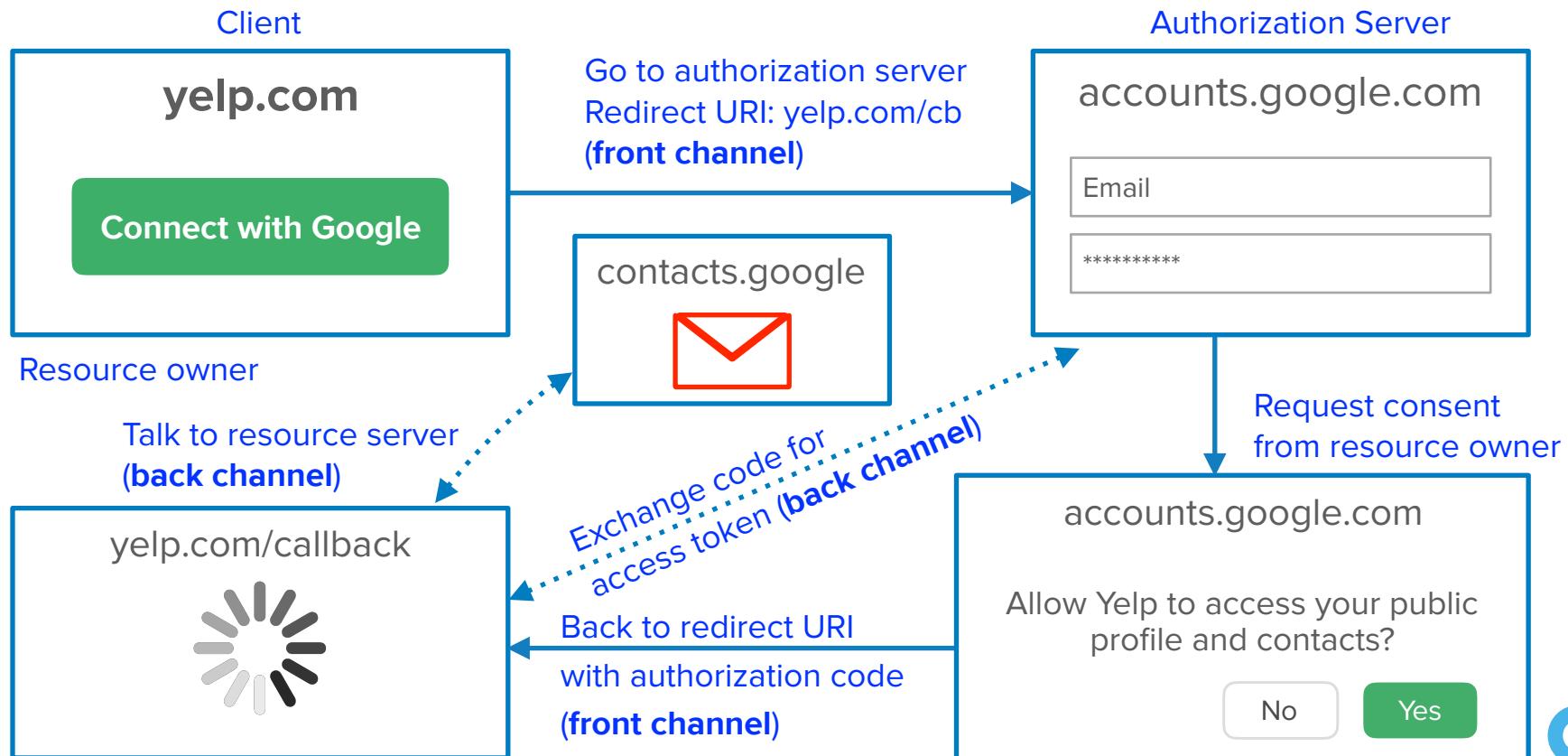


Making Protected Resource Requests

```
curl -H "Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA" \  
https://www.googleapis.com/gmail/v1/users/1444587525/messages
```



OAuth 2.0 Authorization Code Flow



OAuth 2.0 Grant Types (Flows)

Implicit (2 Legged)

- Optimized for browser-only **Public Clients**
- **Access token** returned directly from authorization request (Front-channel only)
- *Does not support refresh tokens*
- Assumes **Resource Owner** and **Public Client** are on the same device
- Most vulnerable to security threats

Authorization Code (3 Legged)

- Front channel flow used by **Client** to obtain **authorization code grant**
- Back channel flow used by **Client** to exchange **authorization code grant** for **access token** and optionally **refresh token**
- Assumes **Resource Owner** and **Client** are on separate devices
- Most secure flow as tokens never passes through user-agent

Client Credential

- Optimized for server-only **Confidential Clients** acting on behalf of itself or a user
- Back-channel only flow to obtain an **access token** using the **Client's** credentials
- Supports shared secrets or assertions as **Client** credentials signed with either symmetric or asymmetric keys



OAuth 2.0 Grant Types (Flows)

Resource Owner Password

- Legacy grant type for native username/password apps such as desktop apps
- Username/password is **authorization grant** to obtain **access token** from **Authorization Server**
- *Does not support refresh tokens*
- Assumes **Resource Owner** and **Public Client** or on the same device

Assertion

- Allows **Authorization Server** to trust **authorization grants** from third party such as SAML IdP (Federation)
- **Assertion** is used to obtain **access token** with token request
- *Does not support refresh tokens*

Device

- Optimized for devices that do not have access to web-browsers
- **User code** is returned from authorization request that must be redeemed by visiting a URL on a device with a browser to authorize
- Back channel flow used by **Client** to poll for authorization approval for **access token** and optionally **refresh token**



OAuth Flows

Six different flows

Necessary because of:

How you get consent from client?

Who is making consent?

Adds a lot of complexity to OAuth

When people ask if you support OAuth, are they asking for all six?

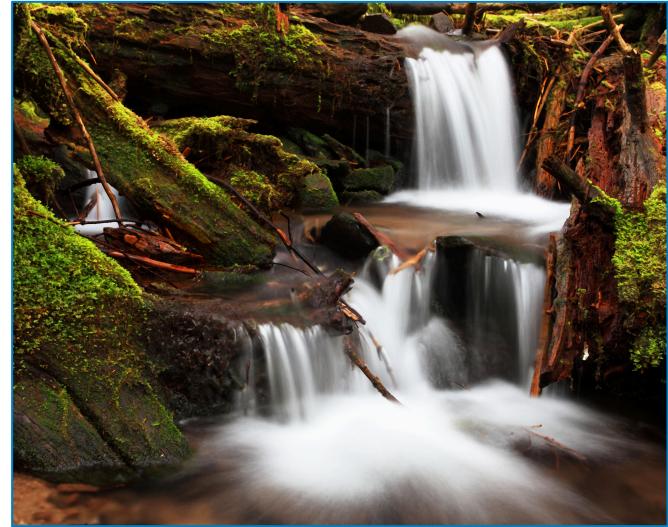
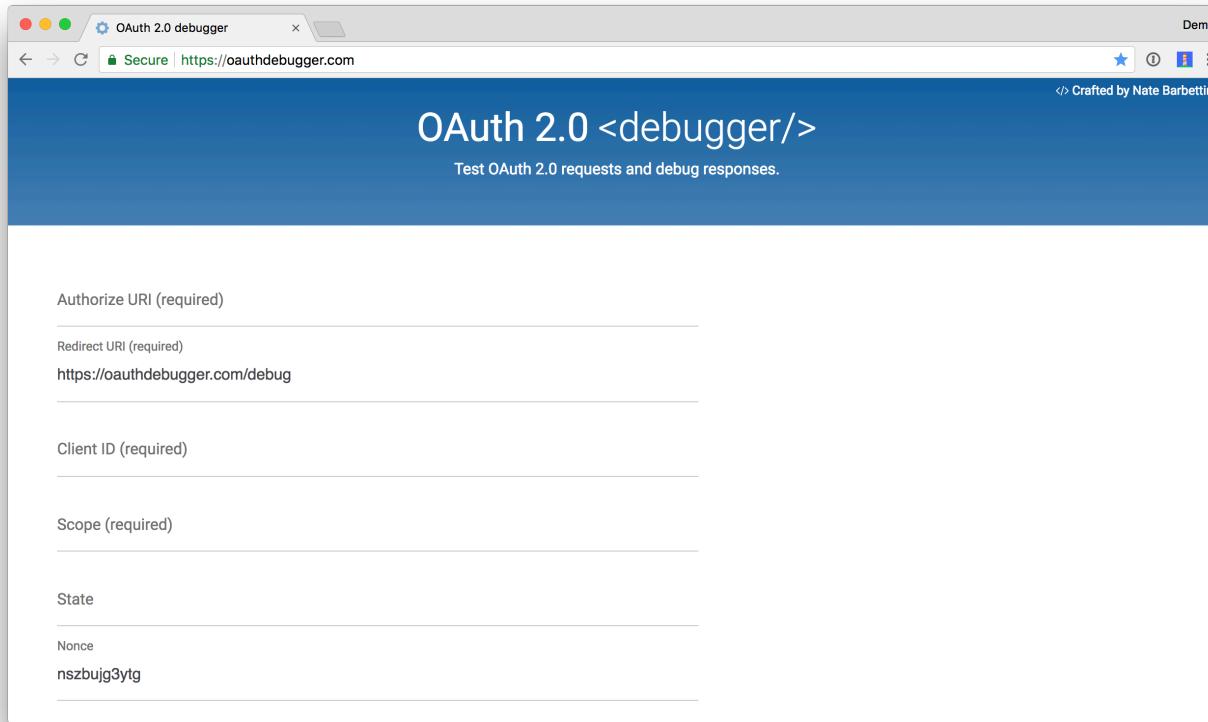


Image: Ian Sane, Spring Runoff
<https://www.flickr.com/photos/31246066@N04/4620052369>



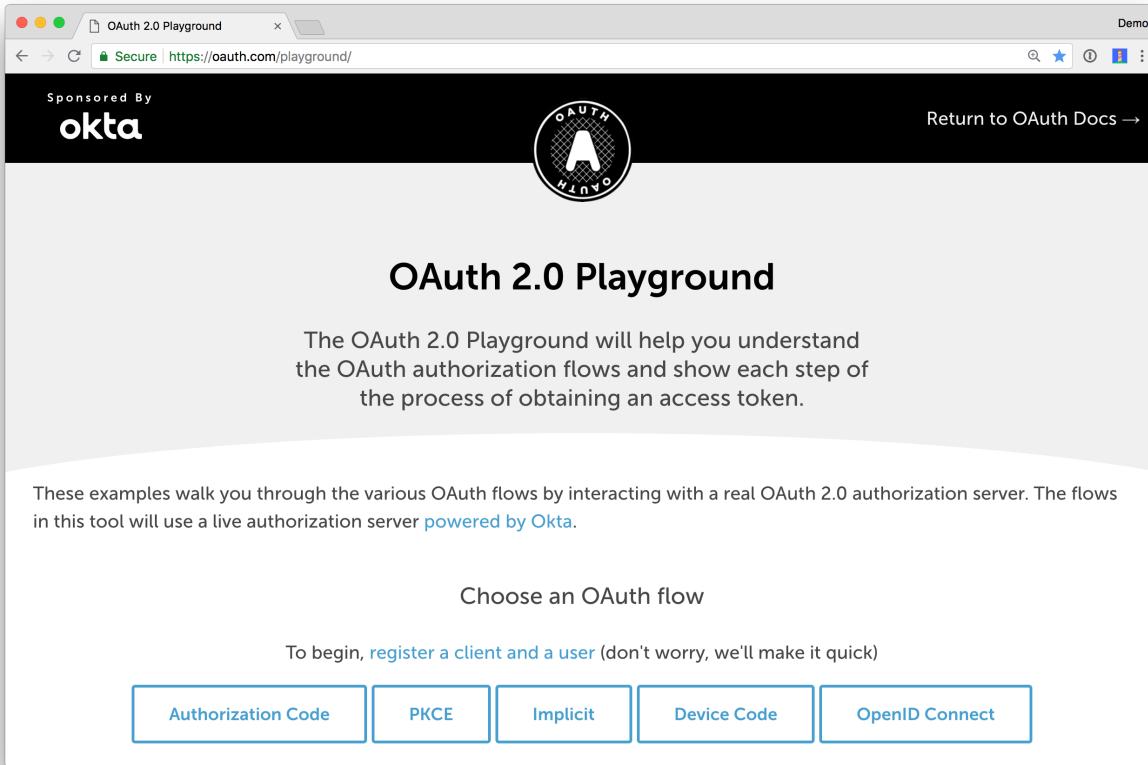
OAuth 2.0 Debugger

<https://oauthdebugger.com>



OAuth 2.0 Playground

<https://oauth.com/playground>



The screenshot shows a web browser window for the OAuth 2.0 Playground. The title bar reads "OAuth 2.0 Playground". The address bar shows a secure connection to "https://oauth.com/playground/". The page header includes a "Sponsored By" logo for Okta and a large circular "A" logo with "OAUTH" text. A "Return to OAuth Docs →" link is also present. The main content area features a heading "OAuth 2.0 Playground" and a descriptive paragraph about the tool's purpose: "The OAuth 2.0 Playground will help you understand the OAuth authorization flows and show each step of the process of obtaining an access token." Below this, a note states: "These examples walk you through the various OAuth flows by interacting with a real OAuth 2.0 authorization server. The flows in this tool will use a live authorization server [powered by Okta](#)". A section titled "Choose an OAuth flow" offers five options: "Authorization Code", "PKCE", "Implicit", "Device Code", and "OpenID Connect".

Sponsored By
okta

Return to OAuth Docs →

OAuth 2.0 Playground

The OAuth 2.0 Playground will help you understand the OAuth authorization flows and show each step of the process of obtaining an access token.

These examples walk you through the various OAuth flows by interacting with a real OAuth 2.0 authorization server. The flows in this tool will use a live authorization server [powered by Okta](#).

Choose an OAuth flow

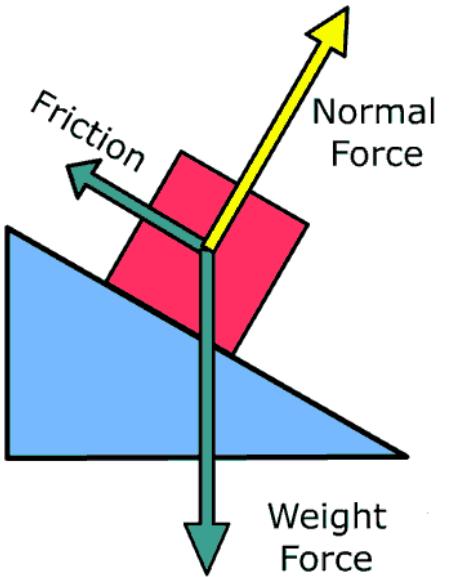
To begin, [register a client and a user](#) (don't worry, we'll make it quick)

Authorization Code PKCE Implicit Device Code OpenID Connect



Token State Management

Developer Friction



Common OAuth 2.0 Security Issues

Too many inputs that need validation

Token hijacking with CSRF

Always use CSRF token with state parameter to ensure OAuth flow integrity

Leaking authorization codes or tokens through redirects

Always whitelist redirect URLs and ensure proper URI validations

Token hijacking by switching clients

Bind the same client to authorization grants and token requests

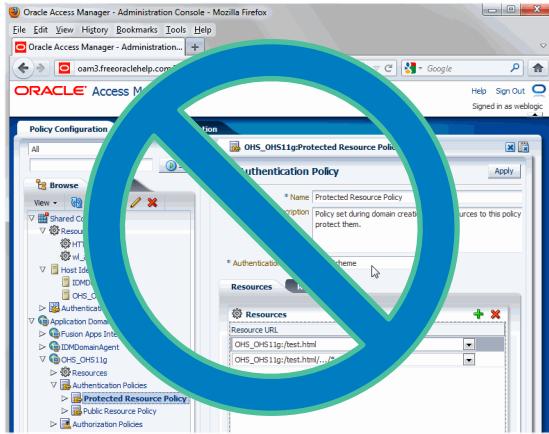
Leaking client secrets

Unbounded & Bearer Tokens

See draft specification of OAuth Proof-of-Possession Token Extension



Key Enterprise OAuth 2.0 Use Cases



Decouples authorization policy decisions from enforcement

Enables the right blend of fine & coarse grained authorization

Replaces traditional Web Access management (WAM) Policies

Restrict and revoke which apps can access specific APIs

Ensure only managed and/or compliant devices can access specific APIs

Deep integration with identity deprovisioning workflow to revoke all tokens for a user and device

Federation with an IdP



OAuth 2.0 Facts

Not backward compatible with OAuth 1.0

Interoperability issues exists as its not a protocol but rather an authorization framework

OAuth 2.0 is not an authentication protocol

OAuth 2.0 alone says absolutely nothing about the user

The screenshot shows a web browser window with the title "End User Authentication with C" and the URL "https://oauth.net/articles/authentication/". The page content includes a logo for "OAUTH" and navigation links for "OAuth 2.0", "Code", "Articles", "Security", "Books", and "About". The main heading is "User Authentication with OAuth 2.0". Below it, a paragraph explains that OAuth 2.0 is a delegation protocol used for authorization decisions across applications. A red rectangular box highlights the sentence: "OAuth 2.0 is not an authentication protocol." At the bottom, another paragraph clarifies that confusion arises because OAuth is used inside authentication protocols.

The [OAuth 2.0](#) specification defines a *delegation* protocol that is useful for conveying *authorization decisions* across a network of web-enabled applications and APIs. OAuth is used in a wide variety of applications, including providing mechanisms for user authentication. This has led many developers and API providers to incorrectly conclude that OAuth is itself an *authentication* protocol and to mistakenly use it as such. Let's say that again, to be clear:

OAuth 2.0 is not an authentication protocol.

Much of the confusion comes from the fact that OAuth is used *inside* of authentication protocols, and developers will see the OAuth components and interact with the OAuth flow and assume that by simply using OAuth, they can accomplish user authentication. This turns out to be not only untrue, but also dangerous for service providers, developers, and end users.



Identity Use Cases (circa 2012)

Simple login — OAuth 2.0

Single sign-on across sites — OAuth 2.0

Mobile app login — OAuth 2.0

Delegated authorization — OAuth 2.0



Not an
Authentication
Protocol?



OAuth 2.0 as Pseudo-Authentication

As made famous by Facebook Connect and Twitter

Client accessing a <https://api.example.com/me> resource with an access token is not authenticating the user

Access tokens just prove the Client was authorized, are opaque, and intended to only be consumed by the Resource Server

Who is the user (claims)?

When did the user authenticate?

Does the user still have an active or expired session?

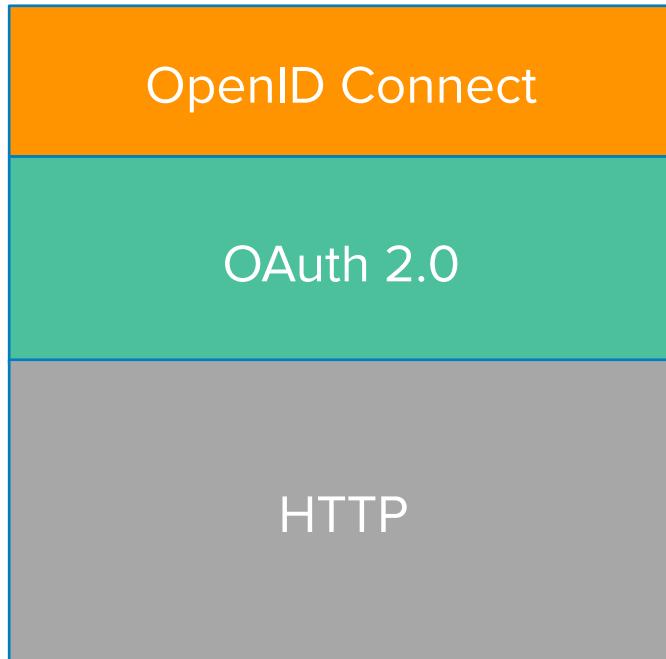
How did the user authenticate?

Just password or password + second factor



OpenID Connect

OAuth 2.0 and OpenID Connect



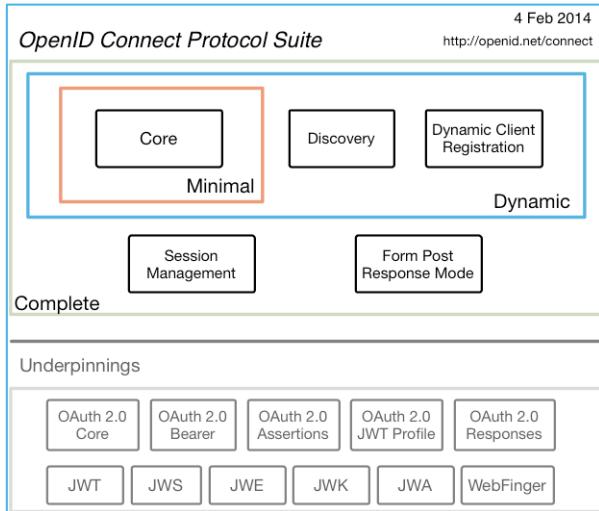
OpenID Connect is for
authentication

OAuth 2.0 is for
authorization



OpenID Connect

OAuth 2.0 + Facebook Connect + SAML 2.0 (good parts)



Extends OAuth 2.0 with new signed **id_token** for the Client and UserInfo endpoint to fetch user attributes

Provides a standard set of scopes and claims for identities

profile

email

address

phone

Built-in registration, discovery & metadata for dynamic federations

Bring Your Own Identity (BYOI)

Supports high assurance levels and key SAML use cases (enterprise)



Authorization Request

Request

```
GET https://accounts.google.com/o/oauth2/auth?  
  scope=openid email&  
  redirect_uri=https://app.example.com/oauth2/callback&  
  response_type=code&  
  client_id=812741506391&  
  state=af0ifjsldkj
```

Response

```
HTTP/1.1 302 Found  
Location: https://app.example.com/oauth2/callback?  
  code=MsCeLvIaQm6bTrgtp7&  
  state=af0ifjsldkj
```

Note: Parameters are not URL-encoded for example purposes



Token Request

POST /oauth2/v3/token HTTP/1.1

Host: www.googleapis.com

Content-Type: application/x-www-form-urlencoded

code=MsCeLvIaQm6bTrgtp7&

client_id=812741506391&

client_secret={client_secret}&

redirect_uri=https://app.example.com/oauth2/callback&

grant_type=authorization_code

Note: Parameters are not URL-encoded for example purposes



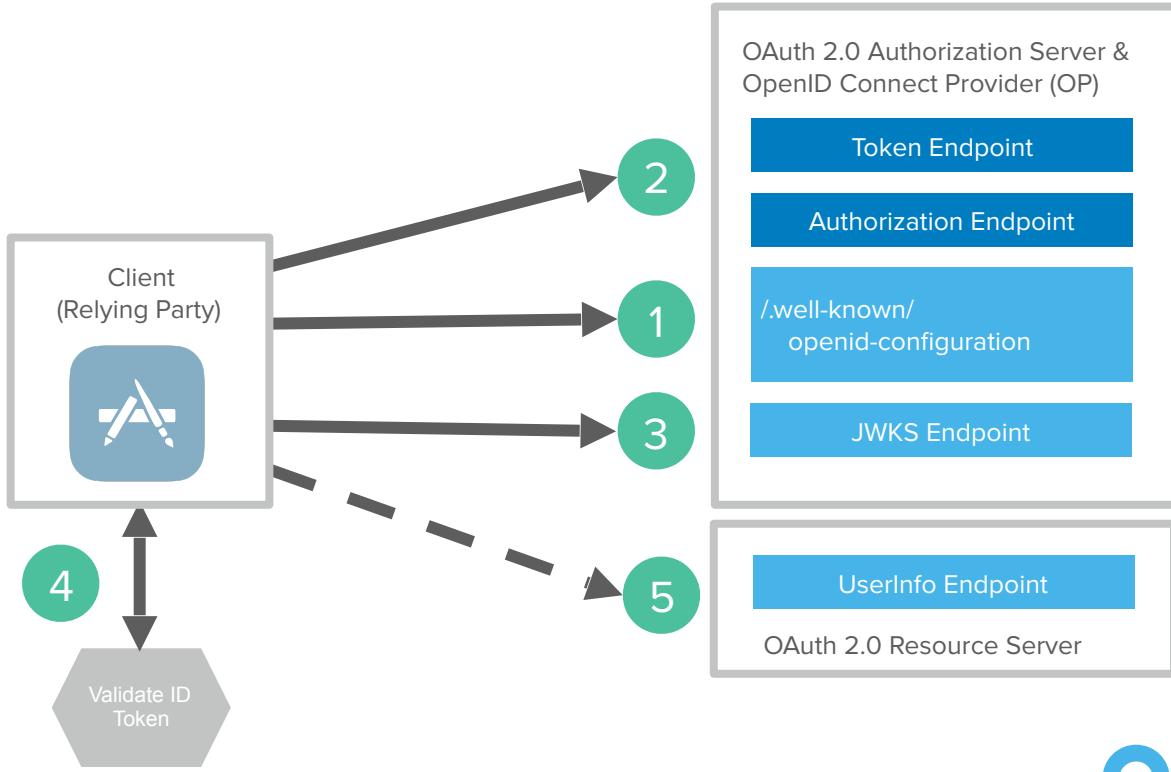
Token Response

```
{  
  "access_token": "2YotnFZFEjr1zCsicMWpAA",  
  "token_type": "Bearer",  
  "expires_in": 3600,  
  "refresh_token": "tGzv3J0kF0XG5Qx2T1KWIA",  
  "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ..."  
}
```

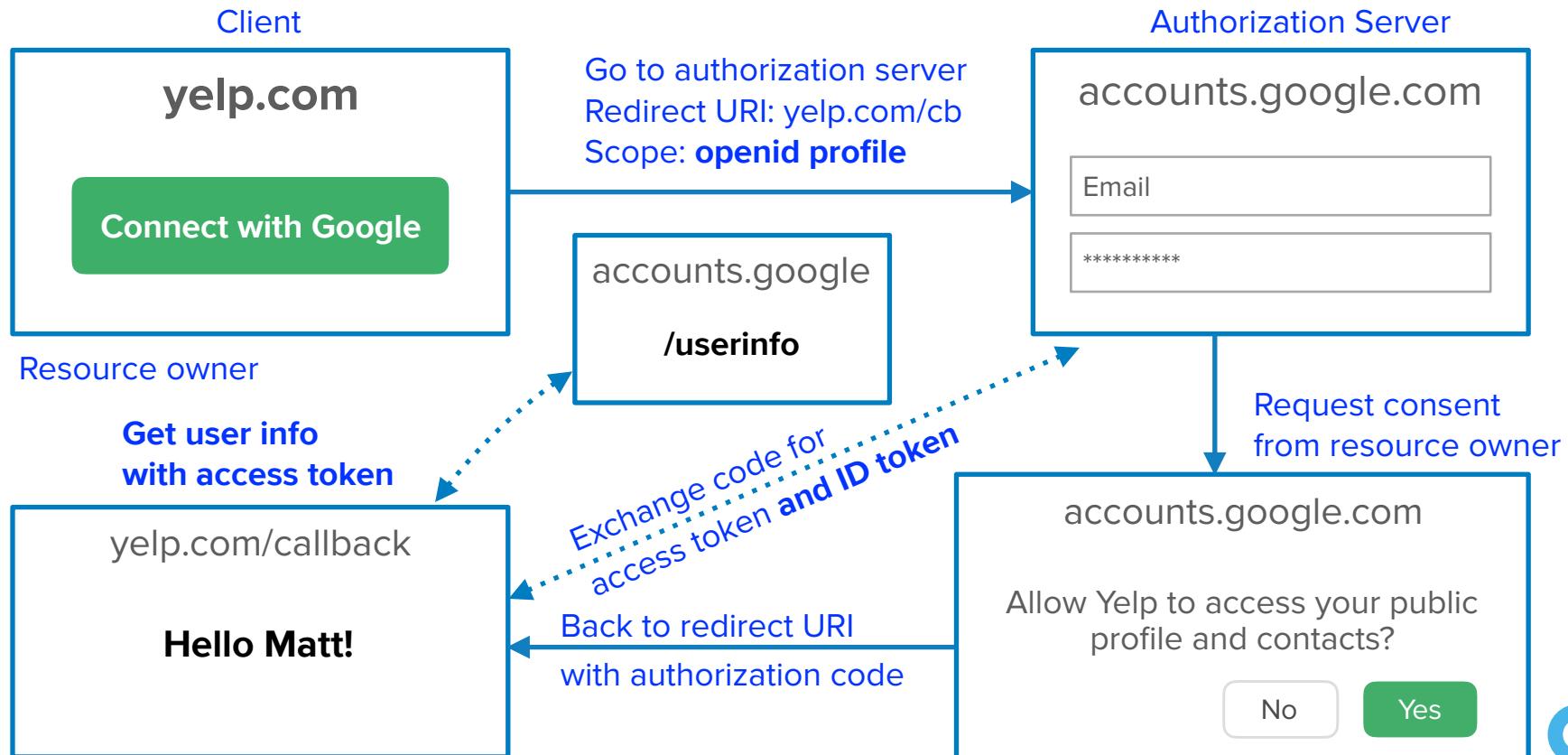


OpenID Connect

- 1 Discover OpenID Provider Metadata
- 2 Perform OAuth flow to obtain a **ID token** and/or **access token**
- 3 Get JSON Web Key Set (JWKS) for signature keys
- 4 Validate **ID token** (JSON Web Token)
- 5 Get additional user attributes with **access token** from UserInfo endpoint



OIDC Authorization Code Flow



JSON Web Token (JWT)

base64url(Header) + ":" + base64url(Claims) + ":" + base64url(Signature)

Header

```
eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2V4YW1wbGUub2t0YS5jb20iLCJzdWIiOiIwMHVncmVuTWVxdllsYTRIVzBnMyIsImF1ZCI6IncyNTVIRVdpU1U0QXVOeEVqZWlqIiwiaWF0IjoxNDQ2MzA1MjgyLCJleHAiOjE0NDYzMDo4ODIsImFtcii6WyJwd2QiXSwiYXV0aF90aW1lIjoxNDQ2MzA1MjgyLCJ1bWFpbCI6ImthcmxAZXhhbXBsZS5jb20iLCJ1bWFpbF92ZXJpZml1ZCI6dHJ1ZX0.XcNXs4C7DqpR22LLti777AMMVcxM7FjEPKZQnd-AS_Cc6R54wuQ5EApuY6GVFCkIlnfbNmYSbHMk04HL3uoexVOPQmcqhNPDLLEChj00jQwZDjhPD9uBoNwGyiZ9_YKwsRpzb9NEeY8xEwXJFIdk6SRktTFrVNHAOIhEQsgm8
```

Signature

Header

```
{  
  "alg": "RS256"  
  "kid": "123456789"  
}
```

Claims

```
{  
  "iss": "https://example.okta.com",  
  "sub": "00ugrenMeqvYla4HW0g3",  
  "aud": "w255HEWiSU4AuNxEjeij",  
  "iat": 1446305282,  
  "exp": 1446308882,  
  "amr": [  
    "pwd"  
  ],  
  "auth_time": 1446305282,  
  "email": "matt@example.com",  
  "email_verified": true  
}
```



jsonwebtoken.io

<https://wwwjsonwebtoken.io>

The screenshot shows the homepage of jsonwebtoken.io. At the top, there's a navigation bar with tabs for "JSON WEB TOKEN", "JWT INSPECTOR", and "JWT101". Below the navigation, a large blue header area contains the text "Encode or Decode JWTs" and a sub-instruction: "Paste a JWT and decode its header, payload, and signature, or provide header, payload, and signature information to generate a JWT". A modal window is open in the foreground, titled "JWT String". Inside the modal, there's a text input field containing a long JWT string: `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOIlxMjM0NTY3ODkwliwibmFtZSI6Ikpvag4gRG9IliwiYWRTaW4iOnRydWUsImp0aSI6IjNhMWQ3YWE4LTcxYmQtNDJhNC04NWY2LTIVOWZmOWU0MzY5ZlslmhCl6MTUzMjA5ODQwMiwiZhwljoxNTMyMTAyMDAyfQ.NGATp5PLICm0clZ09g-vi6B0y7UJCB_Eb0b2Hi1vhsU`. Below the modal, two sections are visible: "Header" and "Payload". The "Header" section contains the JSON object:

```
{"typ": "JWT", "alg": "HS256"}
```

. The "Payload" section contains the JSON object:

```
{"sub": "1234567890", "name": "John Doe", "admin": true, "jti": "3a1d7aa8-71bd-42a4-85f6-5b9ff9e4369f", "iat": 1532098402, "exp": 1532102002}
```

.



Which grant type is right for you?



Native

iOS, Android



SPA

Angular, React, etc.



Web

.NET, Java, etc.



Service

Machine-to-Machine

Authorization
Code

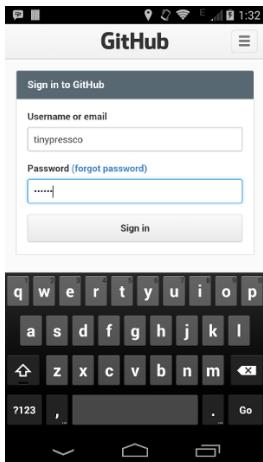
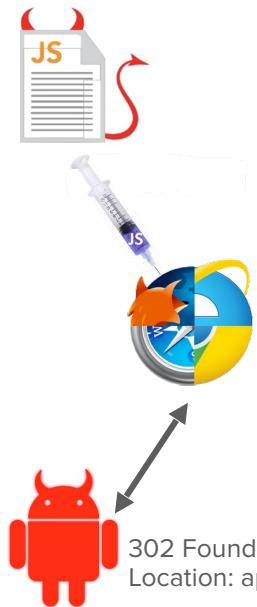
Implicit

Authorization
Code

Client Credentials



Native App Best Practices



Do not use an embedded web views for authenticating users!

App (or 3rd party library/script) can directly obtain the user's credentials which goes against OAuth's raison d'être

Users can't reuse their existing session with their IdP (SSO) increasing friction for sign-in/sign-up

IdPs can't implement new authentication methods

Do not store client secrets in apps that are distributed via App Stores!

Use PKCE (RFC 7636) to protect authorization code from interception

Follow guidelines outlined in **OAuth 2.0 for Native Apps Best Current Practice**

<https://tools.ietf.org/html/draft-ietf-oauth-native-apps-12>

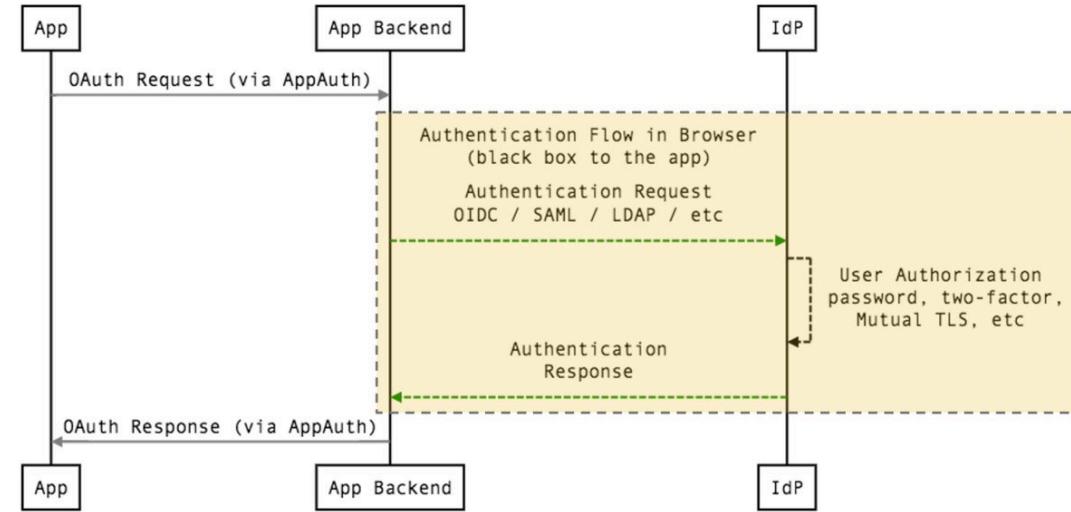


Just Use AppAuth!

<https://appauth.io>



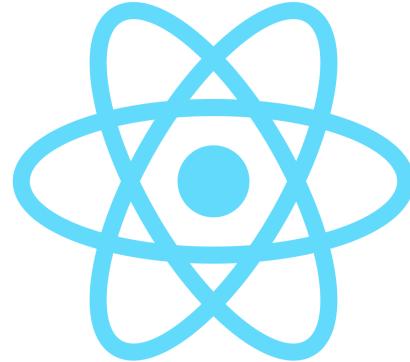
iOS
macOS



Implements **OAuth 2.0 for Native Apps** Best Current Practice

OAuth and OIDC Demos

github.com/oktadeveloper



KEYCLOAK



OAuth and OIDC Libraries

Google OAuth Client Library

ScribeJava

Spring Security OAuth

Nimbus OAuth SDK

List of client and server libraries for many languages:

<https://oauth.net/code/>



Open Source OAuth and OIDC Servers

ORY Hydra

<https://github.com/ory/hydra>



Apereo CAS

<https://github.com/apereo/cas>



Keycloak

<https://github.com/keycloak/keycloak>



JHipster UAA

<https://jhipster.github.io/using-uaa/>



Additional Resources

OAuth Specification

oauth.net

OAuth 2.0 Servers

[oauth.com](https://www.oauth.com)

The image displays two web browser windows side-by-side. The left window shows the homepage of oauth.net, featuring a large yellow banner with the text "An open protocol to allow secure authorization in a standard method from various clients and servers". Below this, there's a section for "Consumer developers..." and another for "For you're building...". The right window shows the homepage of [OAuth.com - OAuth 2.0 Server](https://www.oauth.com), which is sponsored by Okta. It features a prominent "OAuth 2.0 Servers" heading and a sub-section titled "OAuth 2.0 is the modern standard for securing access to APIs." Both sites include navigation menus like "OAuth 2.0", "Code", "Articles", "Security", "Books", and "About".

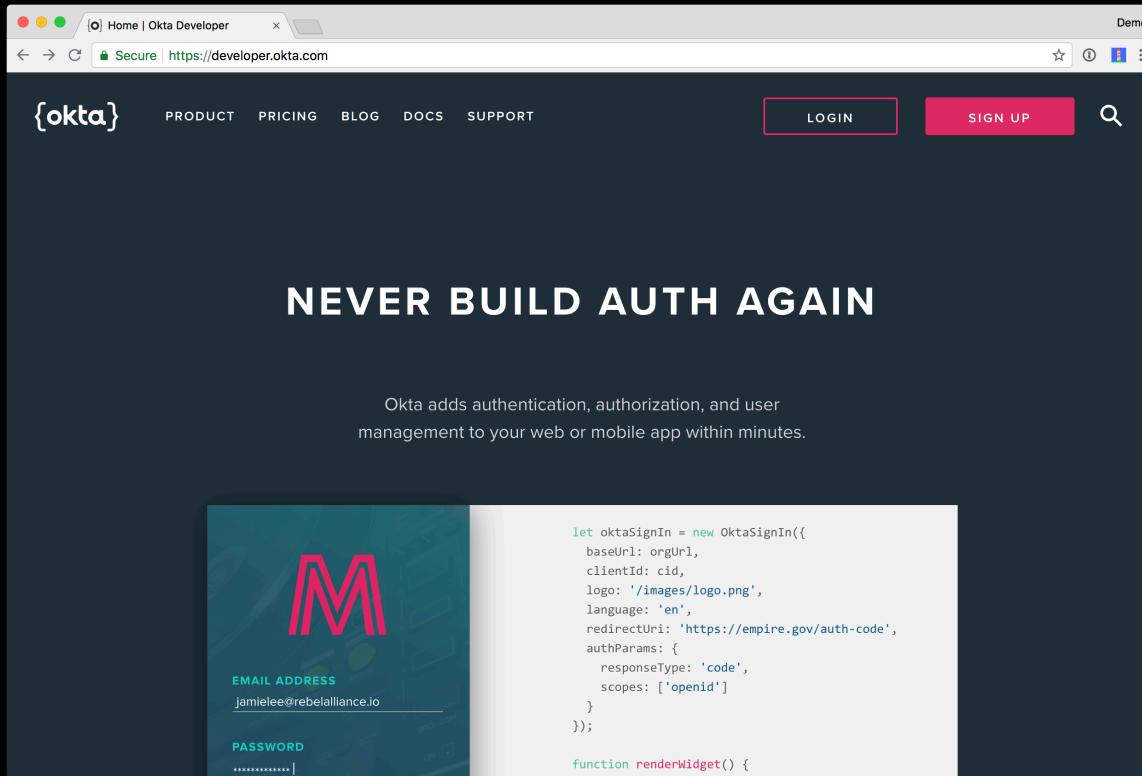


An aerial photograph of a modern city, likely Dubai, showing a complex network of elevated highways and a dense cluster of skyscrapers in the background. The image has a blue-toned overlay.

developer.okta.com/blog

@oktadev

Play with OAuth 2.0 and OIDC



The screenshot shows the Okta Developer homepage. At the top, there's a navigation bar with links for PRODUCT, PRICING, BLOG, DOCS, SUPPORT, LOGIN (in a pink box), SIGN UP (in a pink box), and a search icon. Below the navigation, a large banner features the text "NEVER BUILD AUTH AGAIN". Underneath the banner, a subtext reads: "Okta adds authentication, authorization, and user management to your web or mobile app within minutes." To the left, there's a placeholder image of a login form with fields for EMAIL ADDRESS and PASSWORD. To the right, a block of sample JavaScript code is displayed:

```
let oktaSignIn = new OktaSignIn({
  baseUrl: orgUrl,
  clientId: cid,
  logo: '/images/logo.png',
  language: 'en',
  redirectUri: 'https://empire.gov/auth-code',
  authParams: {
    responseType: 'code',
    scopes: ['openid']
  }
});

function renderWidget() {
```



YouTube: OAuth 2.0 in plain English



<https://youtu.be/996OiexHze0>



Questions?

Keep in touch!

 raibledesigns.com

 [@mraible](https://twitter.com/mraible)

Presentations

 speakerdeck.com/mraible

Code

 github.com/oktadeveloper





@oktadev