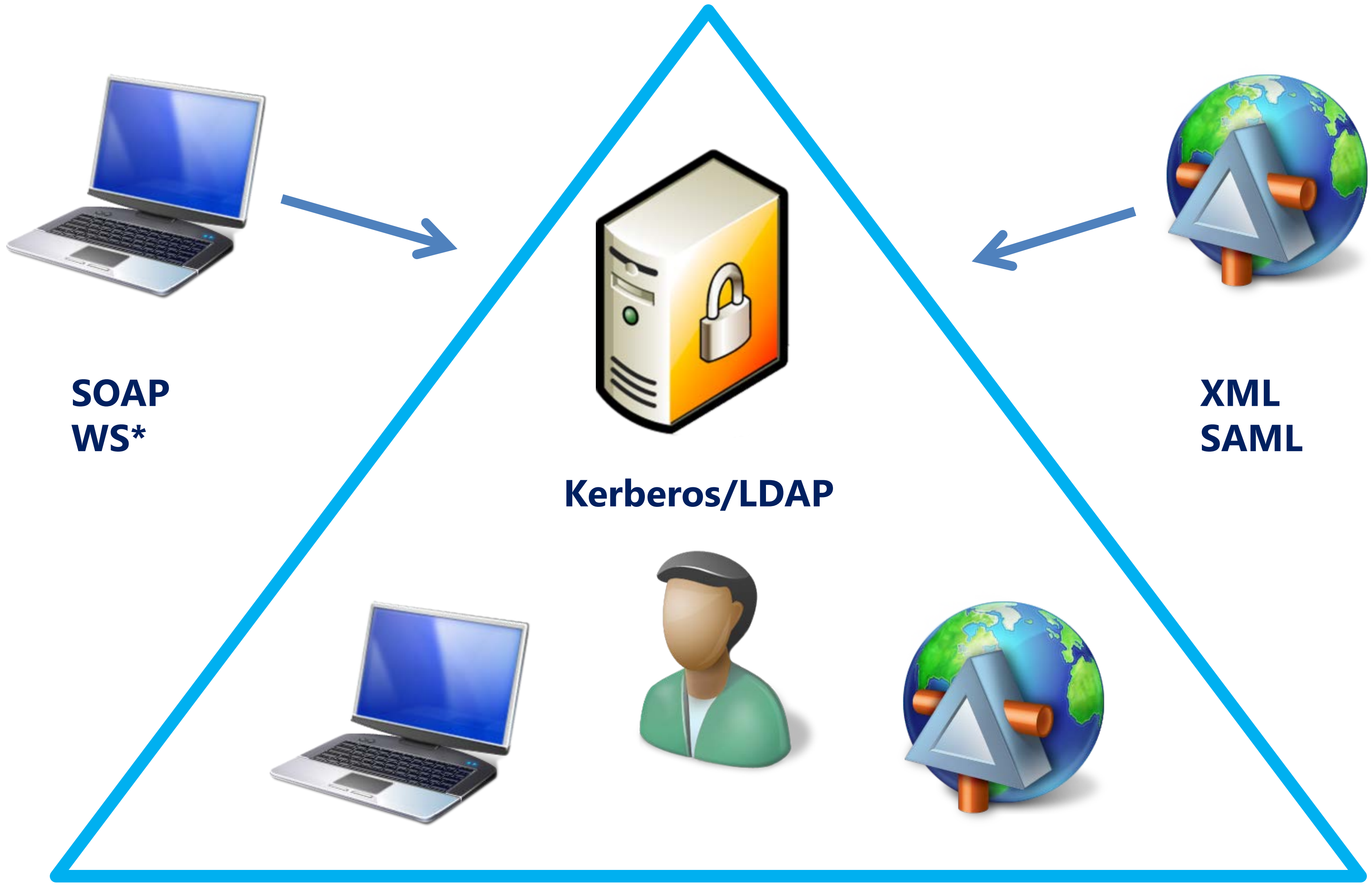
OAuth2, OpenID Connect and JSON Web Tokens (JWT)

Outline

- The new security stack for modern applications
- JSON Web Tokens
- OAuth2
- OpenID Connect

- OAuth2 security discussion
- Resources

Enterprise Security



The mobile Revolution

No SOAP
No SAML
No WS*

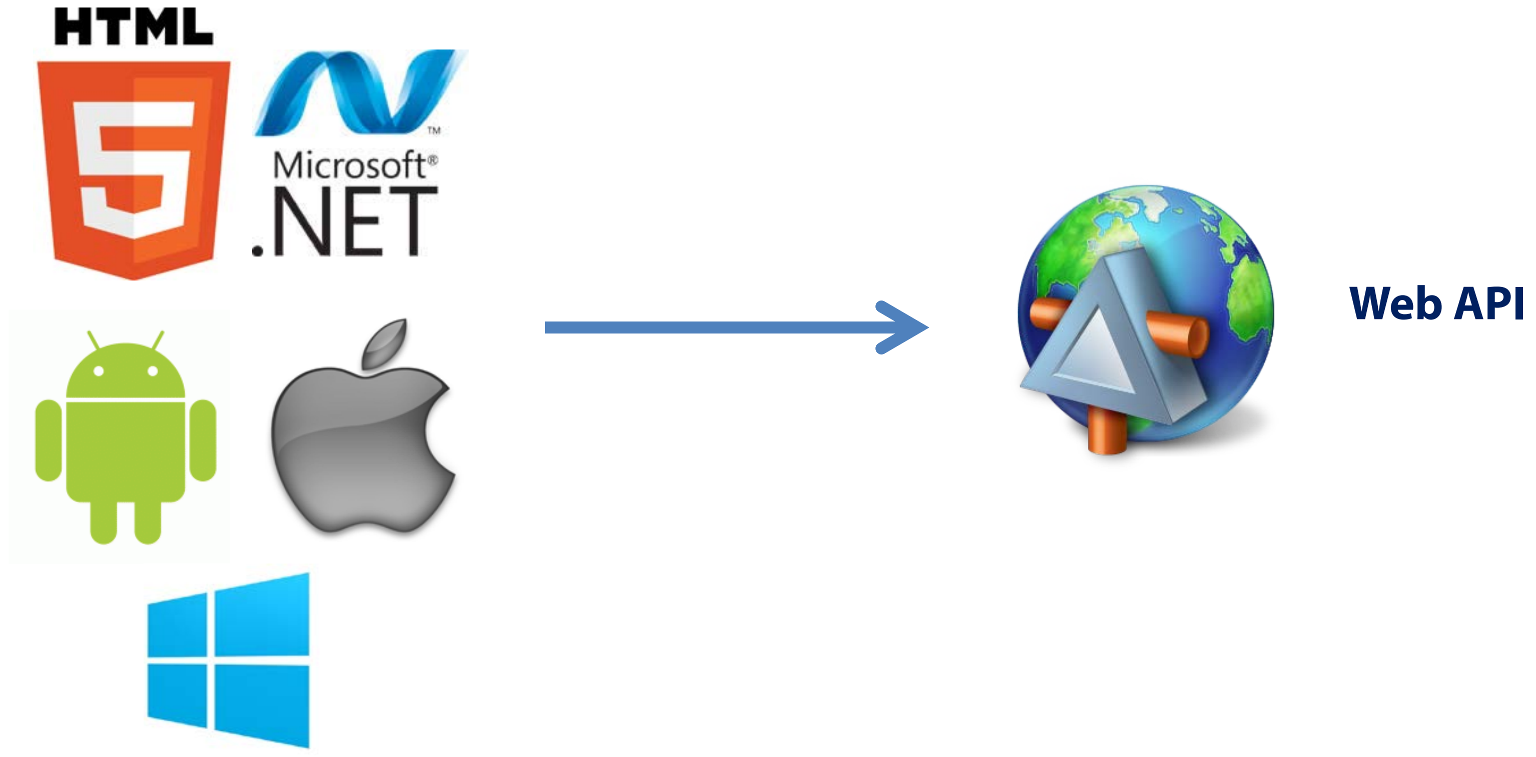
HTTP
JSON

Scenario 1: Mobile Enterprise Apps

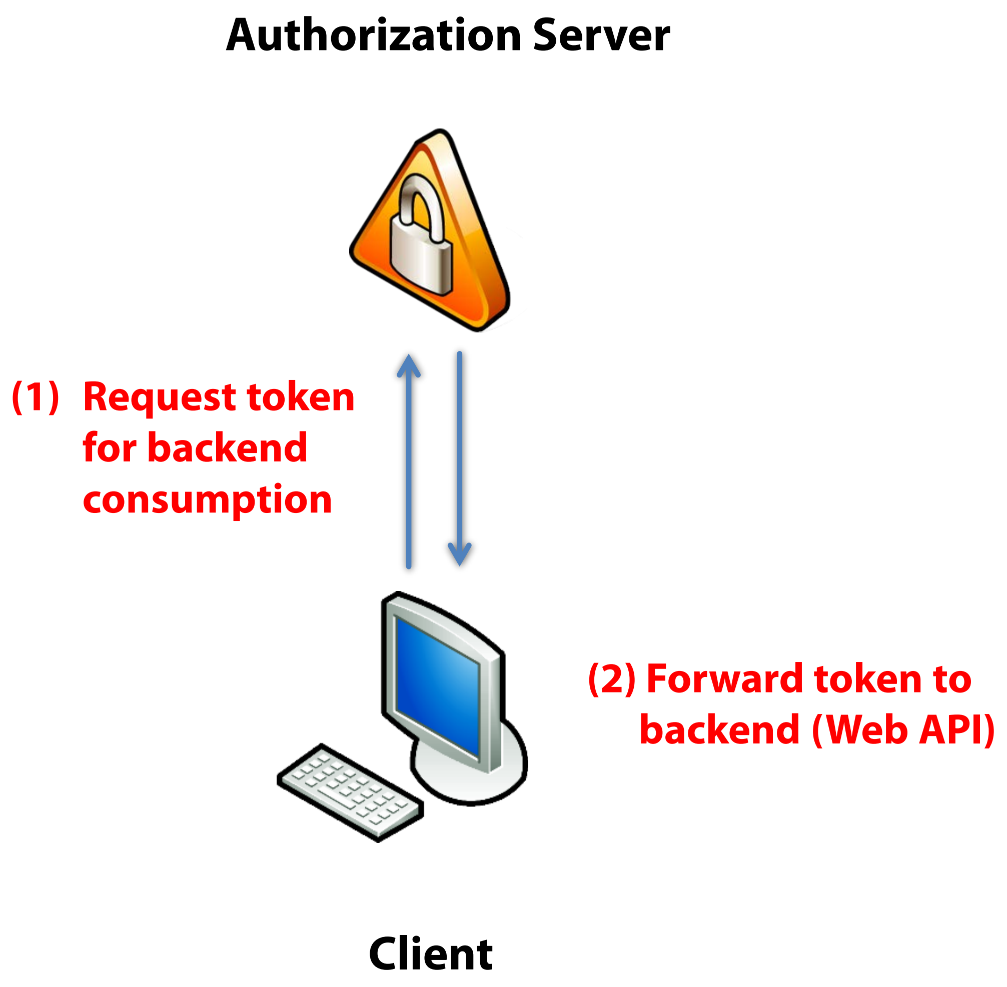
AT&T 3G
9:42 AM
Text
Calendar
Photos
Camera
YouTube
Stocks
Maps
Weather
Clock
Calculator
Notes
Settings
iTunes
App Store
Phone
Mail
Safari
iPod

Scenario 2: Business to Customer

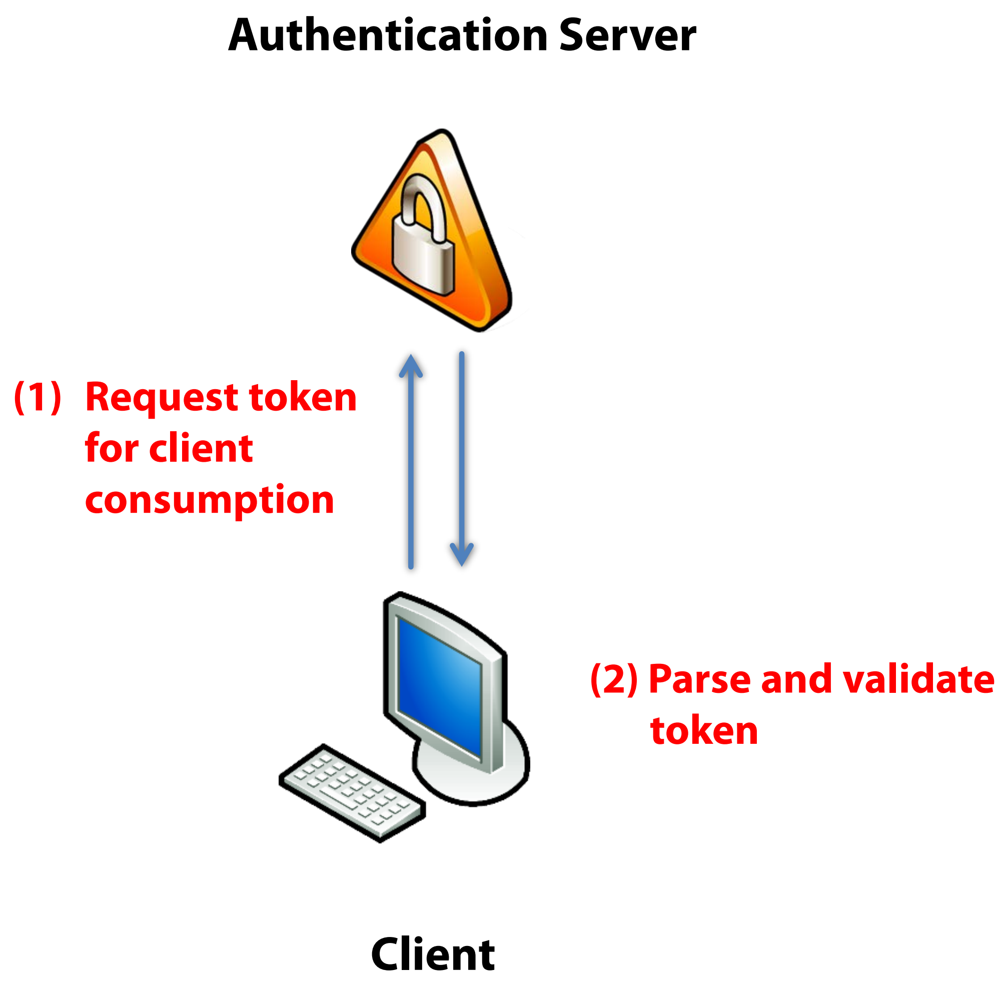
- Software vendors jump on the "apps bandwagon"
- Reach and cross-platform design becomes much more important



OAuth2


Authorization Server
(1) Request token for backend consumption
(2) Forward token to backend (Web API)
Client

OpenID Connect


Authentication Server
(1) Request token for client consumption
(2) Parse and validate token
Client

- **"Classic" security is intranet-only**
  - plus maybe special customer facing (web) applications in the DMZ
- **B2B federation using protocols like WS-Federation, SAML2p and WS-Trust**

- **Mobile devices are a game changer**
  - no "enterprise security" integration
  - less powerful
  - …but increasingly popular and business criticial
- **New "common denominator" technologies**
  - presentation (e.g. HTML5)
  - authentication & authorization

Purpose of a security token

- **Security tokens are (protected) data structures**
  - contain information about issuer and subject (claims)
  - signed (tamper proof & authenticity)
  - typically contain an expiration time

- **A client requests a token**
- **An issuer issues a token**
- **A resource consumes a token**
  - has a trust relationship with the issuer

History

- **SAML 1.1/2.0**
  - XML based
  - many encryption & signature options
  - very expressive
- **Simple Web Token (SWT)**
  - Form/URL encoded
  - symmetric signatures only
- **JSON Web Token (JWT)**
  - JSON encoded
  - symmetric and asymmetric signatures (HMACSHA256-384, ECDSA, RSA)
  - symmetric and asymmetric encryption (RSA, AES/CGM)
  - (the new standard)

JSON Web Token

- **On its way to official standardization**
  - http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html

- **Header**
  - metadata
  - algorithms & keys used
- **Claims**
  - Issuer (iss)
  - Audience (aud)
  - IssuedAt (iat)
  - Expiration (exp)
  - Subject (sub)
  - …and application defined claims

Structure

Header

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

Claims

```
{
  "iss": "http://myIssuer",
  "exp": "1340819380",
  "aud": "http://myResource",
  "sub": "alice",

  "client": "xyz",
  "scope": ["read", "search"]
}
```

$$\underbrace{\texttt{eyJhbGciOiJub25lIn0}}_{\textbf{Header}}\texttt{.}\underbrace{\texttt{eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD}}_{\textbf{Claims}}\texttt{.}\underbrace{\texttt{4MTkzODAsDQogImh0dHA6Ly9leGFt}}_{\textbf{Signature}}$$

Producing a token

- **Microsoft library on Nuget**
  - http://nuget.org/packages/Microsoft.IdentityModel.Tokens.JWT/

```
var token = new JWTSecurityToken(
     issuer: "http://myIssuer",
     audience: "http://myResource",
     claims: GetClaims(),
     signingCredentials: GetKey(),
     validFrom: DateTime.UtcNow,
     validTo: DateTime.UtcNow.AddHours(1));


// serialize
var tokenString =
 new JWTSecurityTokenHandler().WriteToken(token);
```

Consuming a token

- **Retrieve serialized token**
  - from HTTP header, query string etc…
- **Validate token**
  - and turn into claims

```
var token = new JWTSecurityToken(tokenString);
var validationParams = new TokenValidationParameters
{
     ValidIssuer = "http://myIssuer",
     AllowedAudience = "http://myResource",
     SigningToken = GetSigningKey()
};

var handler = new JWTSecurityTokenHandler();
var principal = handler.ValidateToken(token, validationParams);
```

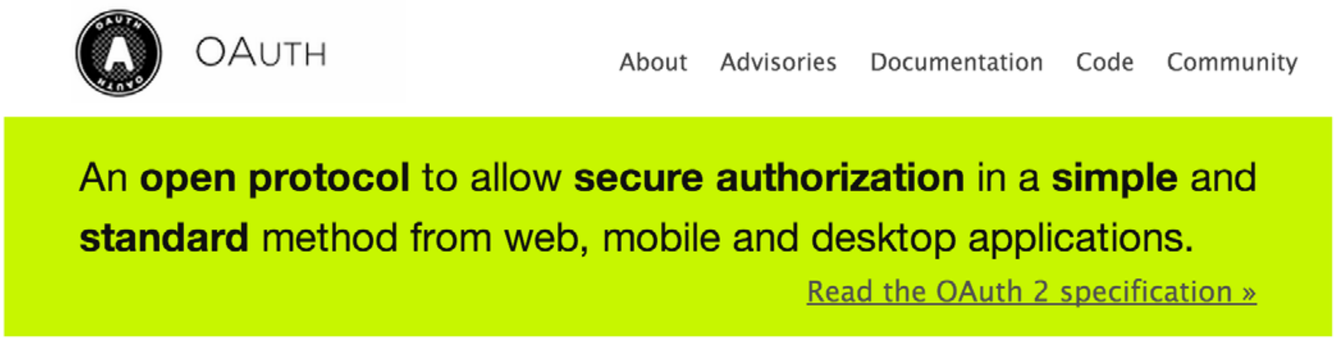
- **JWT is easy to**
    - create
    - transmit
    - parse
    - validate

- **Quickly becomes the standard for web based tokens**
- **Mandatory in OpenID Connect**

Outline

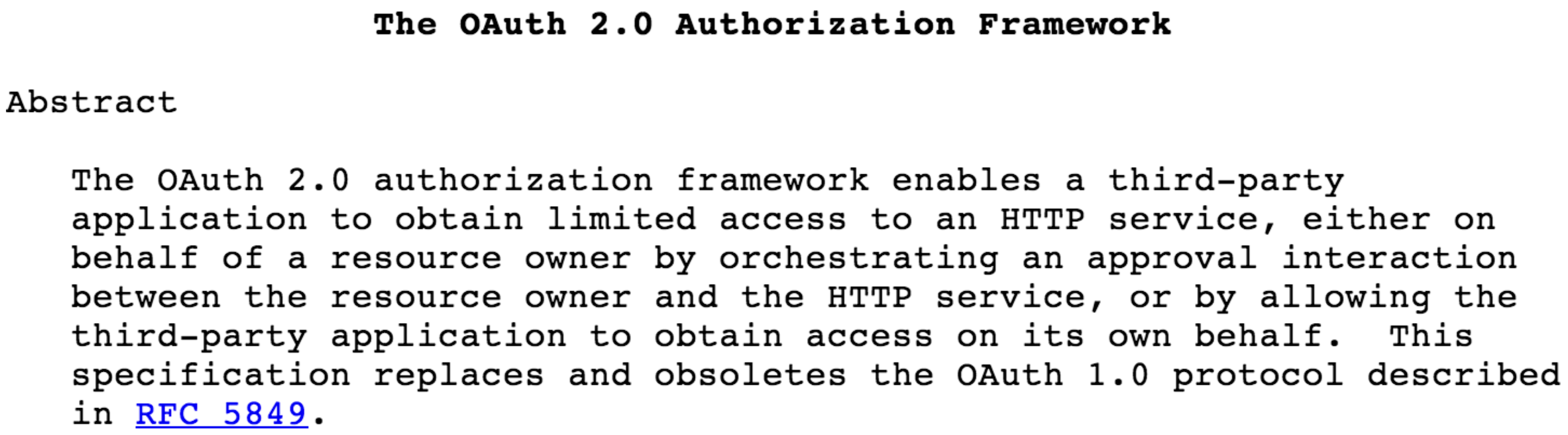
- Overview
- History
- Flows

What is OAuth2 ?



An **open protocol** to allow **secure authorization** in a **simple** and **standard** method from web, mobile and desktop applications.

Read the OAuth 2 specification »

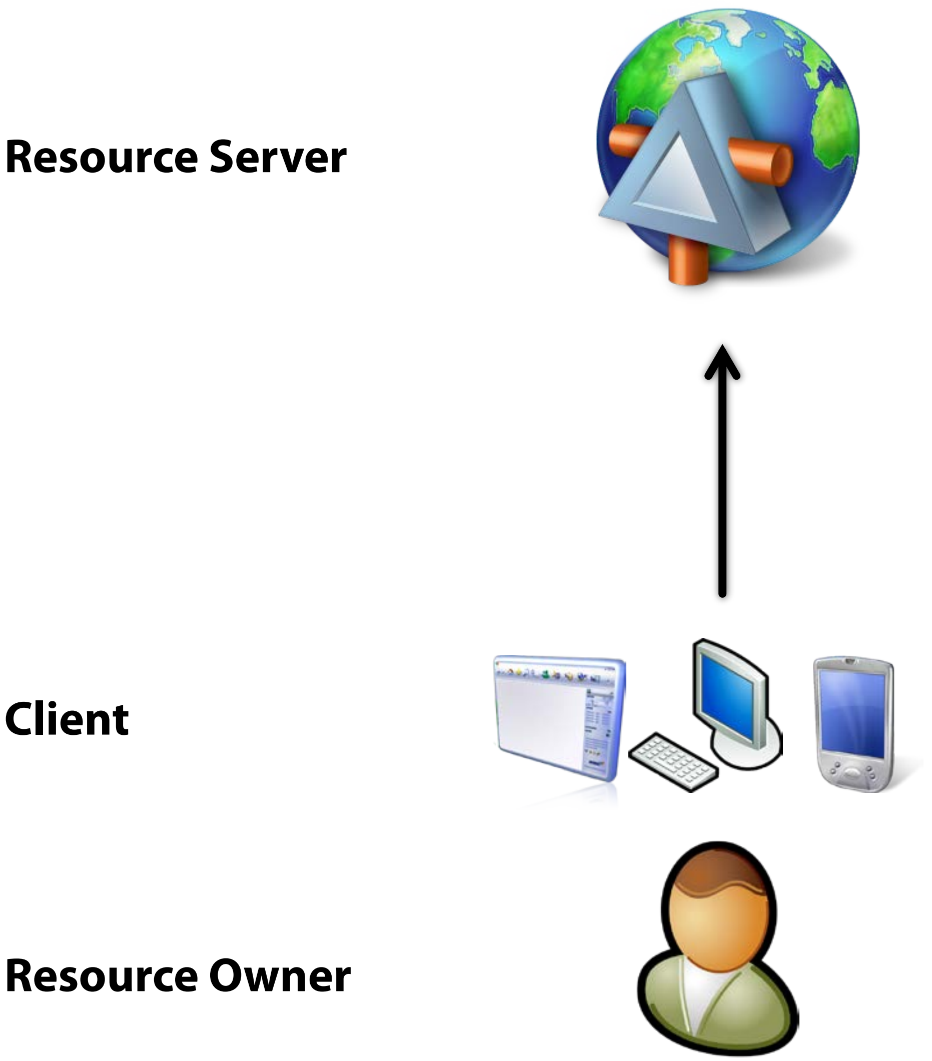


The OAuth 2.0 Authorization Framework

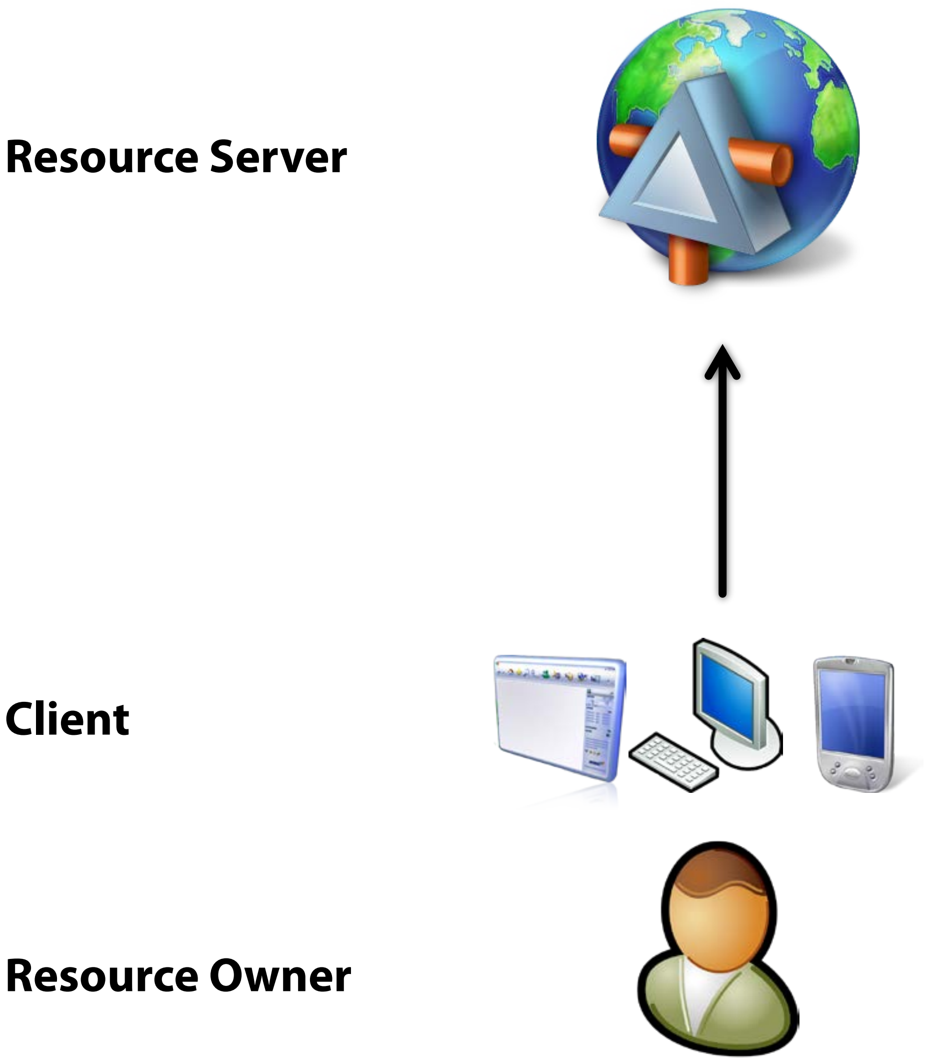
Abstract

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849.

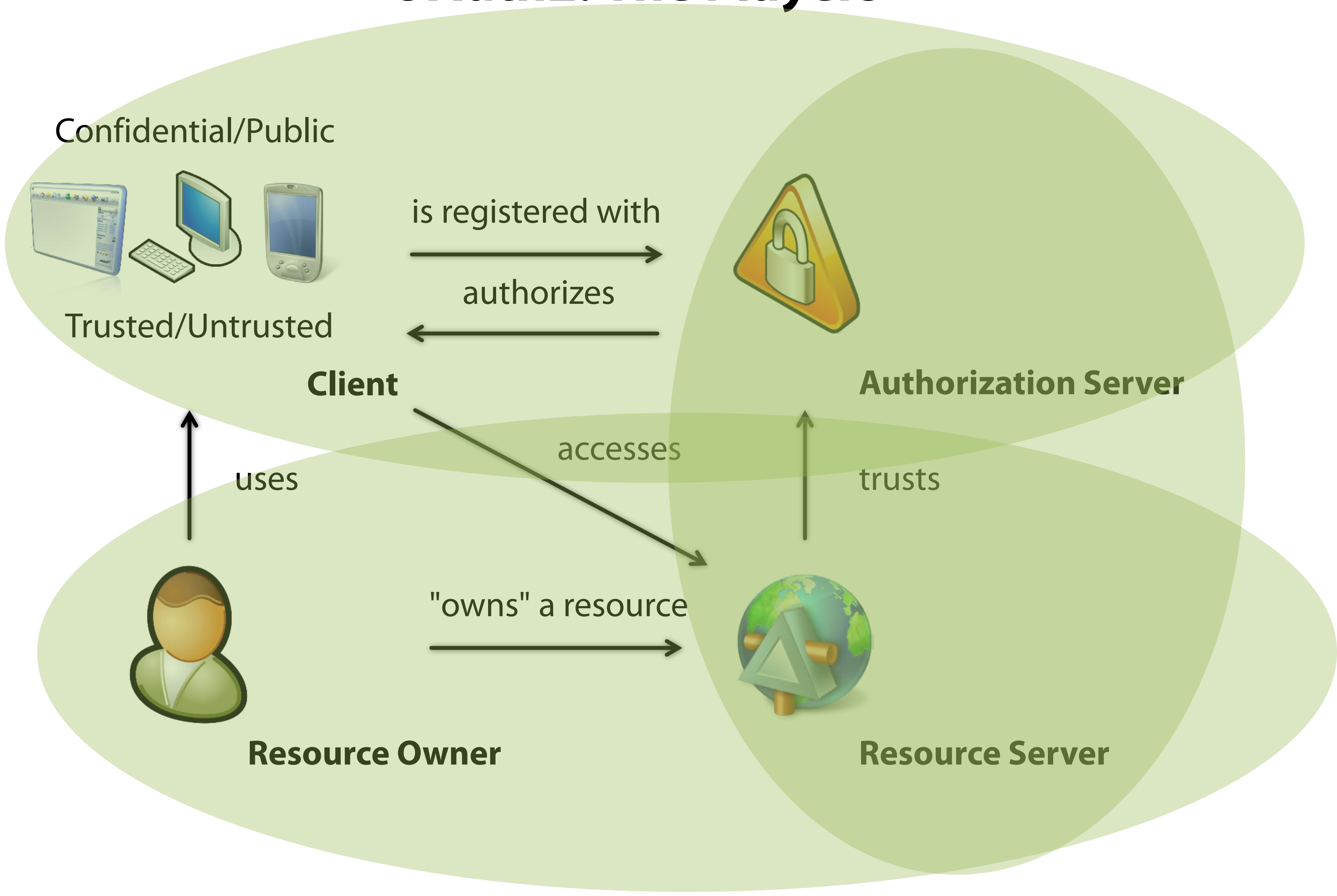
High level overview



High level overview



OAuth2: The Players



OAuth2 Flows - with User Interaction

- **Authorization Code Flow**
  - Web application clients
    1. Request authorization
    2. Request token
    3. Access resource
- **Implicit Flow**
  - Native / local clients
    1. Request authorization & token
    2. Access resource

OAuth2 Flows - no User Interaction

- **Resource Owner Password Credential Flow**
  - "Trusted clients"
    1. Request token with resource owner credentials
    2. Access resource
- **Client Credential Flow**
  - Client to Service communication
    1. Request token with client credentials
    2. Access resource

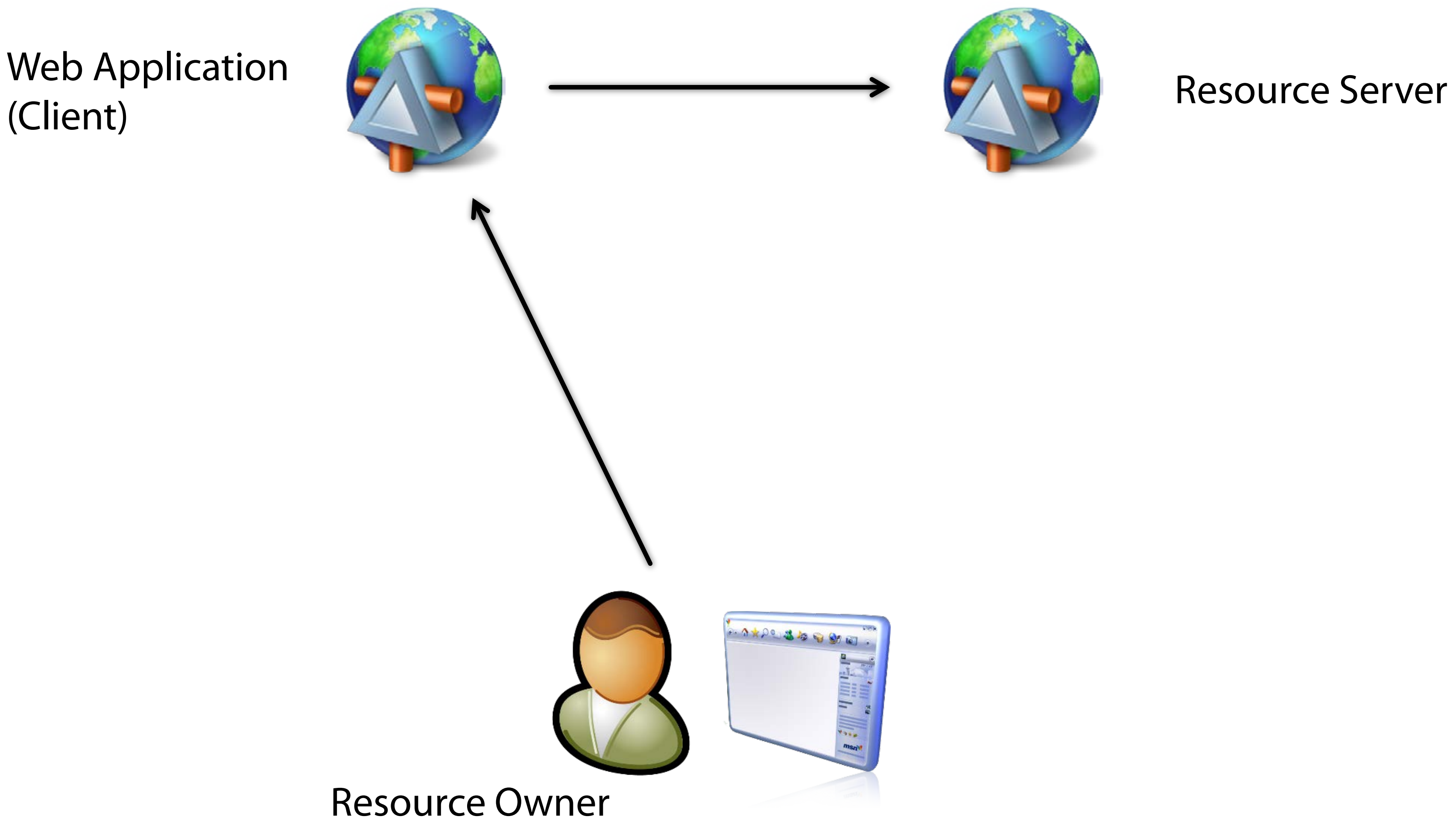
- **OAuth2 makes it HTTP/JSON friendly to request and transmit tokens**
  - typically for delegated authorization (access tokens)
- **Takes "multiple client" architectures into account**
  - clients can have varying trust levels

- **Since v2 of the spec is quite new, there's currently quite a discussion about its pros & cons. See Appendix A**

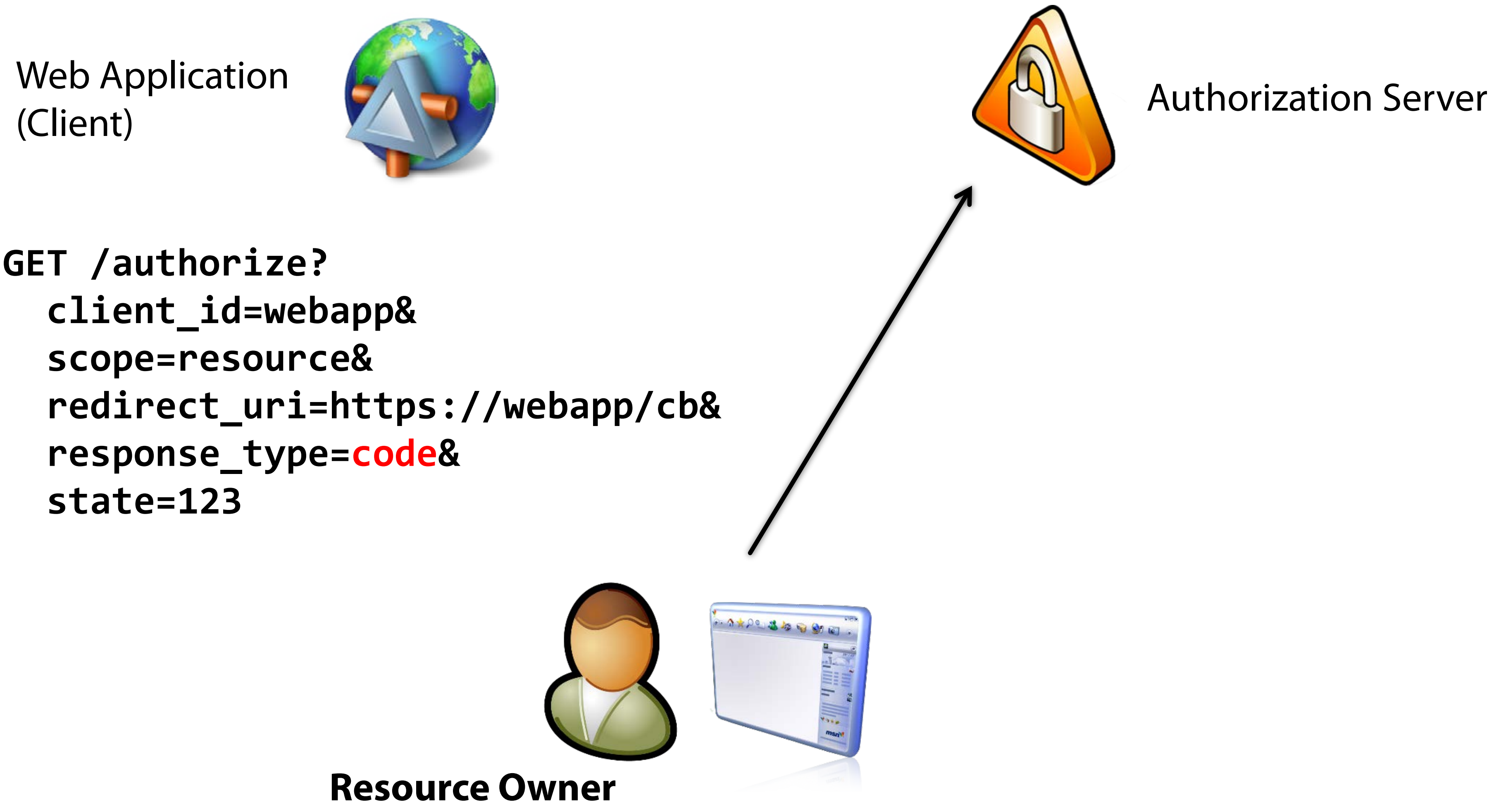
Outline

- Authorization Code Flow
- Implicit Flow
- Resource Owner Credential Flow
- Client Credential Flow

Authorization Code Flow (Web Application Clients)

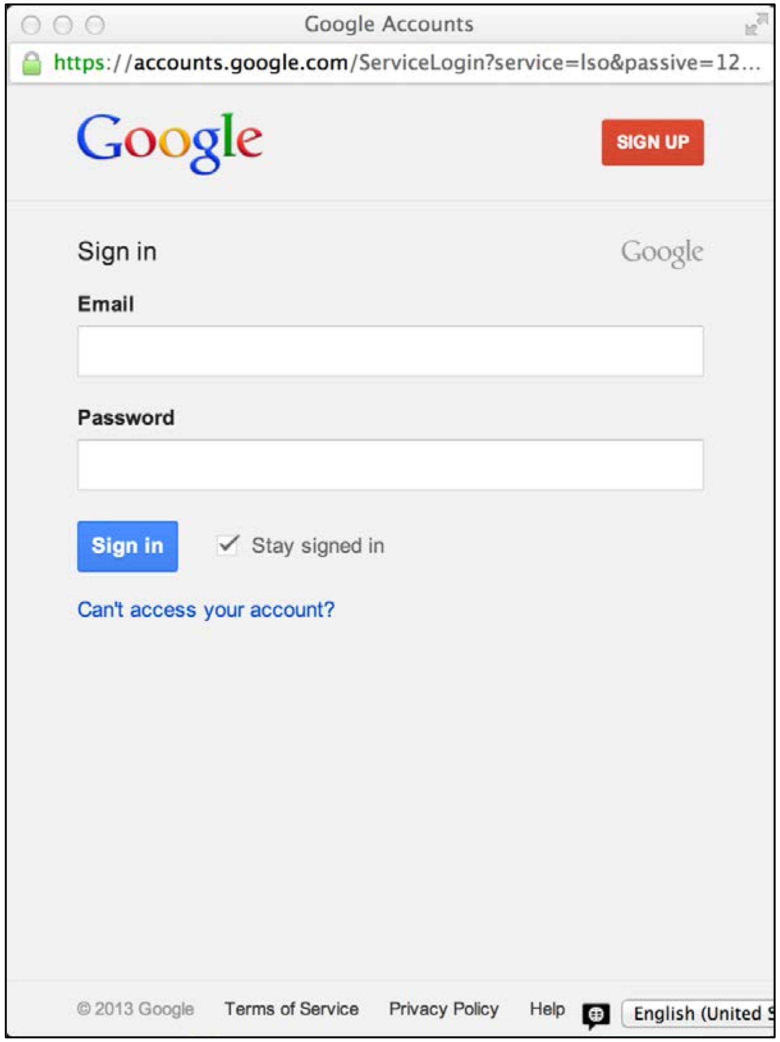


Step 1a: Authorization Request



```
GET /authorize?
  client_id=webapp&
  scope=resource&
  redirect_uri=https://webapp/cb&
  response_type=code&
  state=123
```

Step 1b: Authentication


Google Accounts

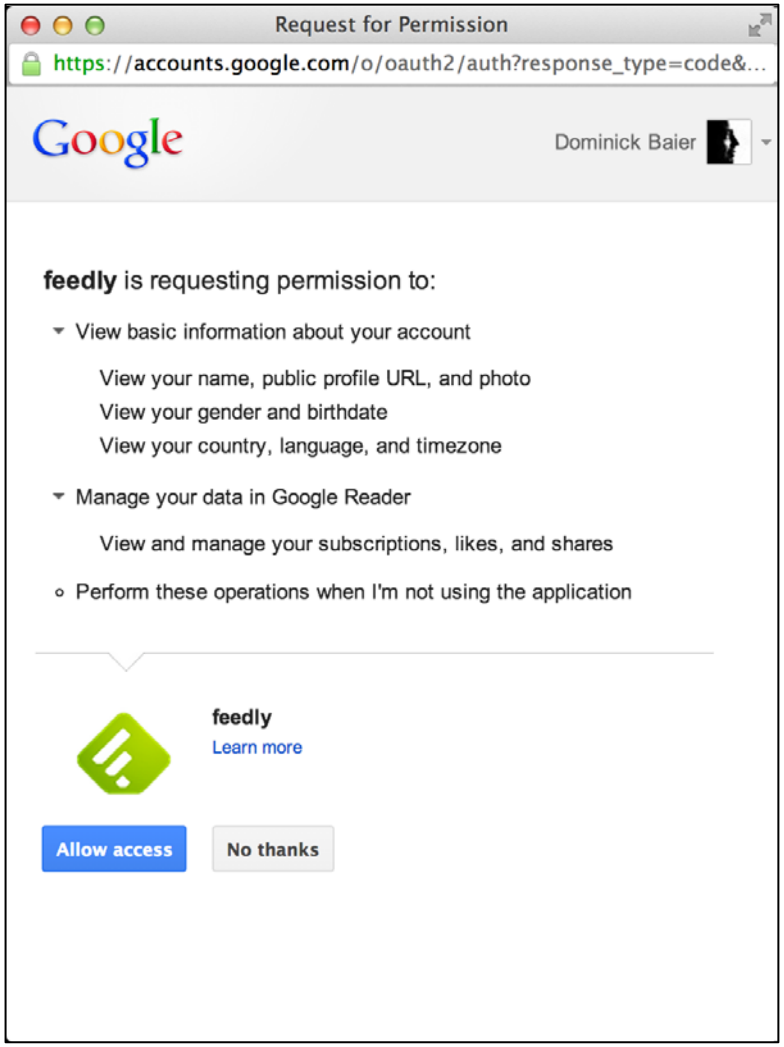
https://accounts.google.com/ServiceLogin?service=lso&passive=12...

Sign in

Email

Password

Step 1c: Consent


Request for Permission

https://accounts.google.com/o/oauth2/auth?response_type=code&...

Dominick Baier

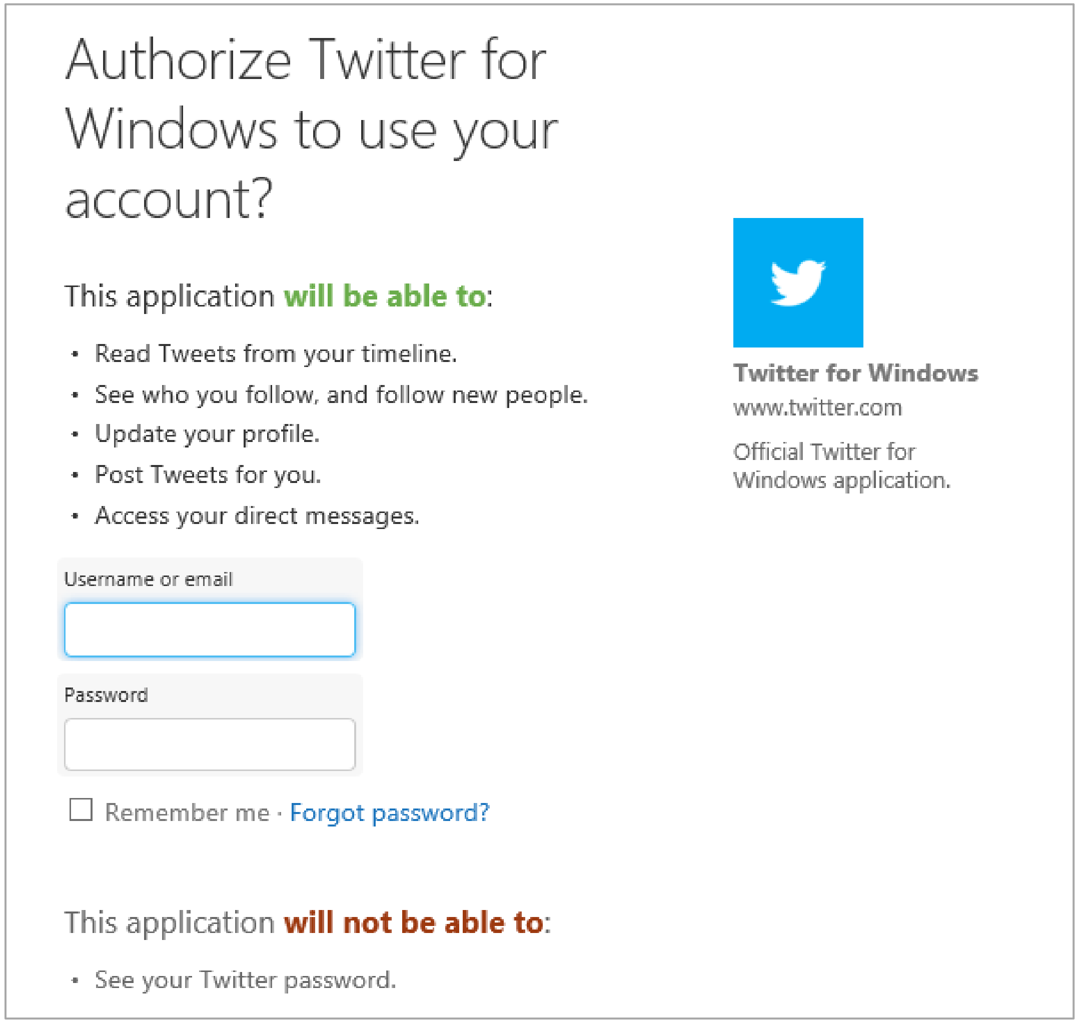
feedly is requesting permission to:

- View basic information about your account
  - View your name, public profile URL, and photo
  - View your gender and birthdate
  - View your country, language, and timezone
- Manage your data in Google Reader
  - View and manage your subscriptions, likes, and shares
- Perform these operations when I'm not using the application

feedly

Learn more

Twitter Consent



Authorize Twitter for Windows to use your account?

This application **will be able to**:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.
- Access your direct messages.

Username or email

Password

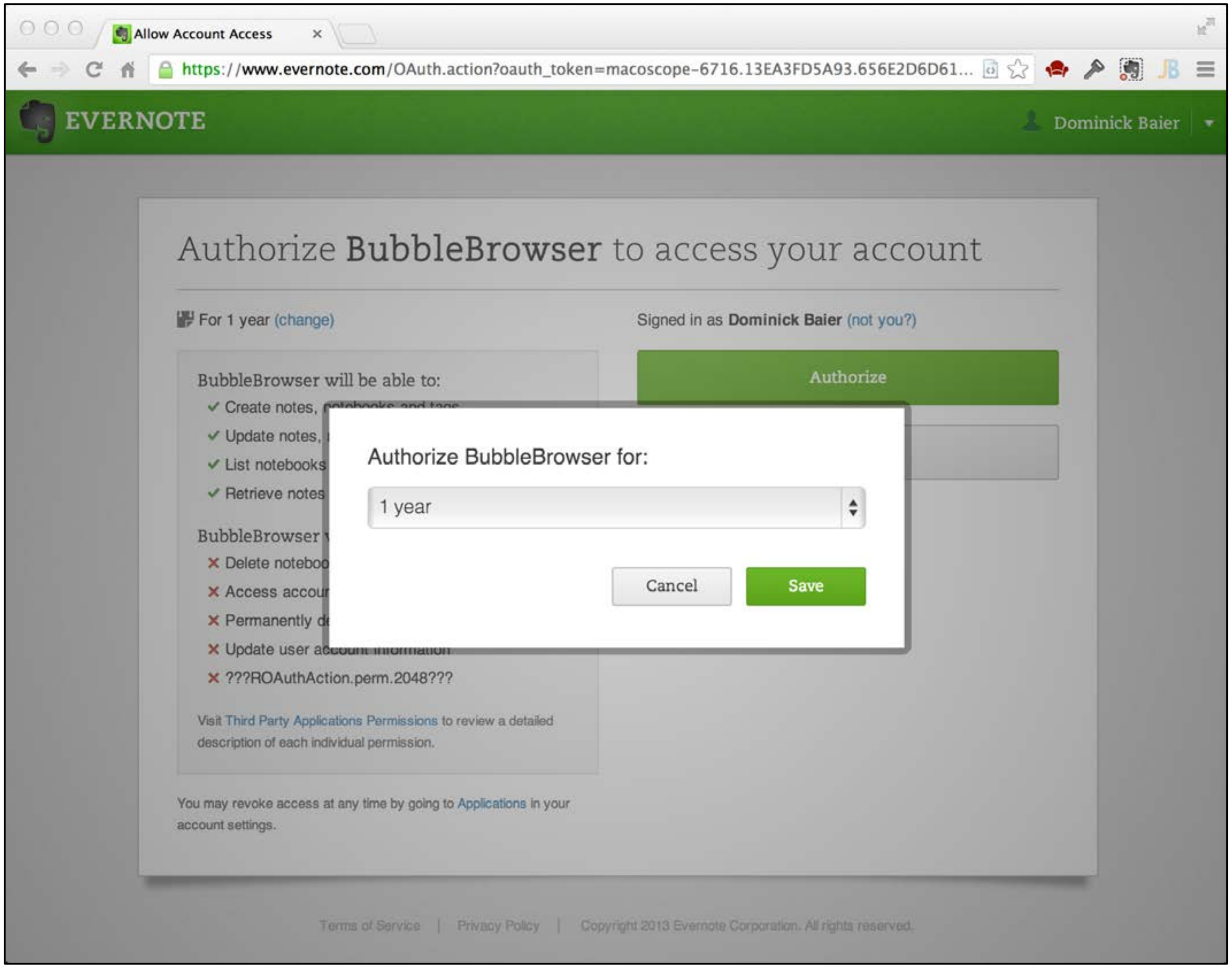
☐ Remember me · Forgot password?

This application **will not be able to**:

- See your Twitter password.

Twitter for Windows
www.twitter.com
Official Twitter for Windows application.

Evernote Consent



Allow Account Access

https://www.evernote.com/OAuth.action?oauth_token=macoscope-6716.13EA3FD5A93.656E2D6D61...

EVERNOTE

Dominick Baier

Authorize **BubbleBrowser** to access your account

For 1 year (change)

Signed in as **Dominick Baier** (not you?)

Authorize

BubbleBrowser will be able to:

✓ Create notes, notebooks and tags

✓ Update notes,

✓ List notebooks

✓ Retrieve notes

BubbleBrowser

✗ Delete notebo

✗ Access accou

✗ Permanently d

✗ Update user account information

✗ ???ROAuthAction.perm.2048???

Visit Third Party Applications Permissions to review a detailed description of each individual permission.

You may revoke access at any time by going to Applications in your account settings.

Authorize BubbleBrowser for:

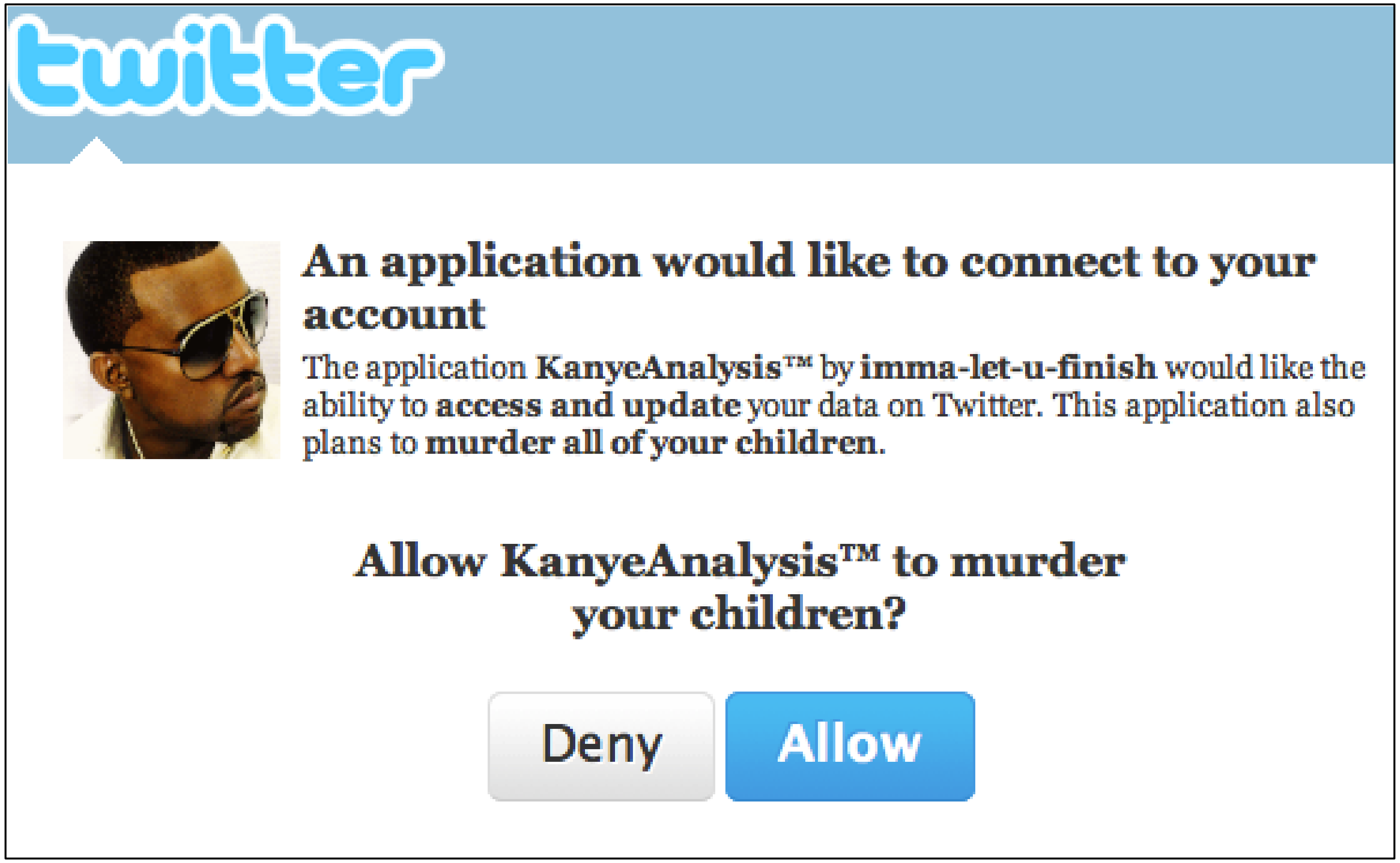
1 year

Cancel

Save

Terms of Service | Privacy Policy | Copyright 2013 Evernote Corporation. All rights reserved.

The Consent Screen is important!



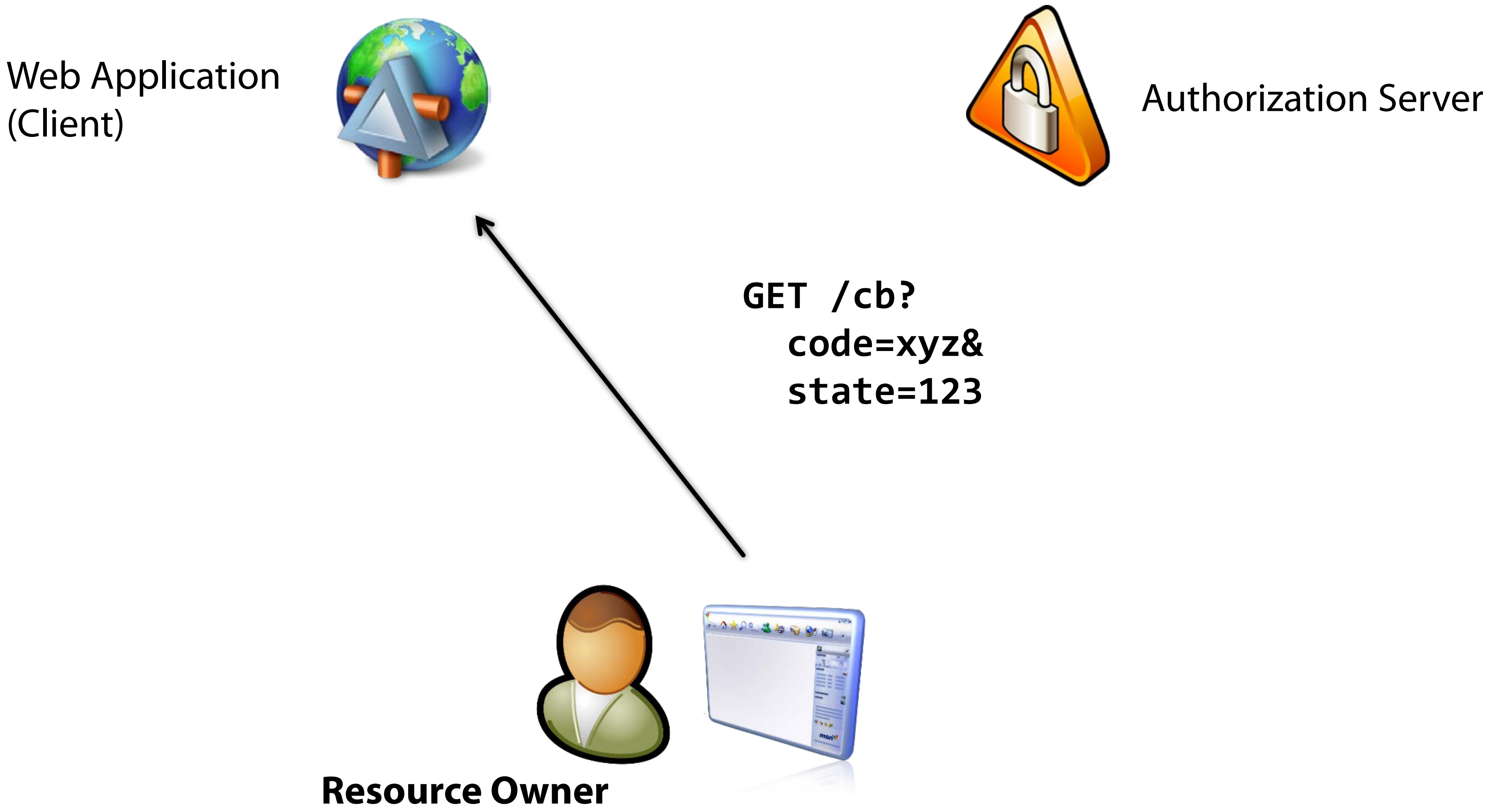
An application would like to connect to your account

The application **KanyeAnalysis™** by **imma-let-u-finish** would like the ability to **access and update** your data on Twitter. This application also plans to **murder all of your children**.

Allow KanyeAnalysis™ to murder your children?

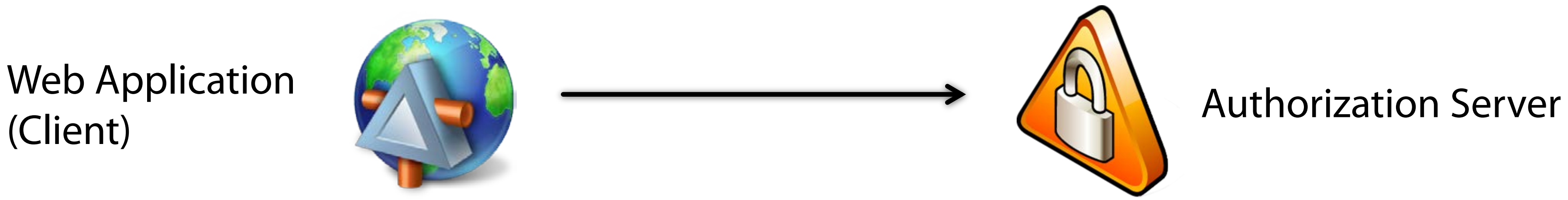
http://zachholman.com/2011/01/oauth_will_murder_your_children/

Step 1d: Authorization Response

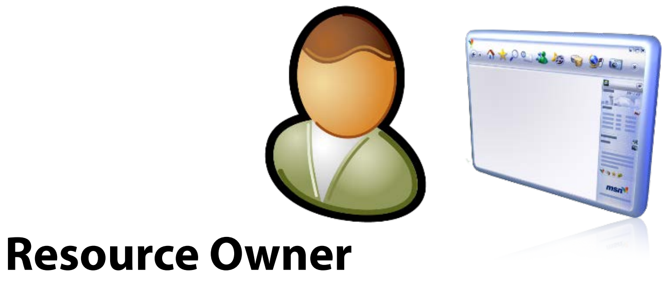


```
GET /cb?
  code=xyz&
  state=123
```

Step 2a: Token Request

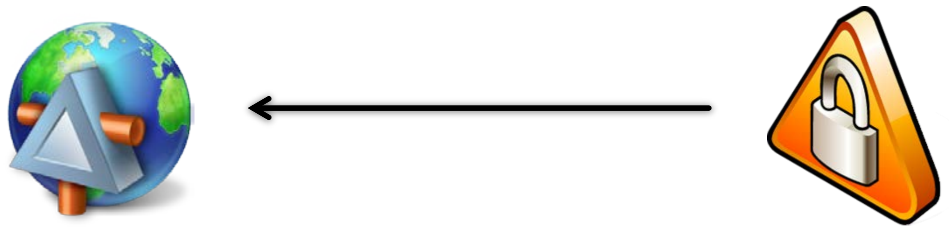


```
POST /token
 Authorization: Basic (client_id:secret)

grant_type=authorization_code&
authorization_code=xyz&
redirect_uri=https://webapp/cb
```



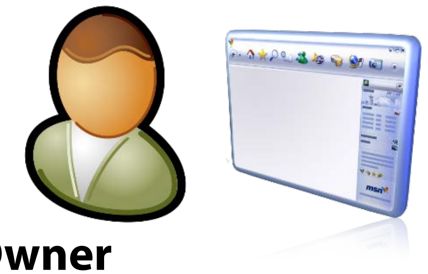
Step 2b: Token Response

Web Application
(Client)



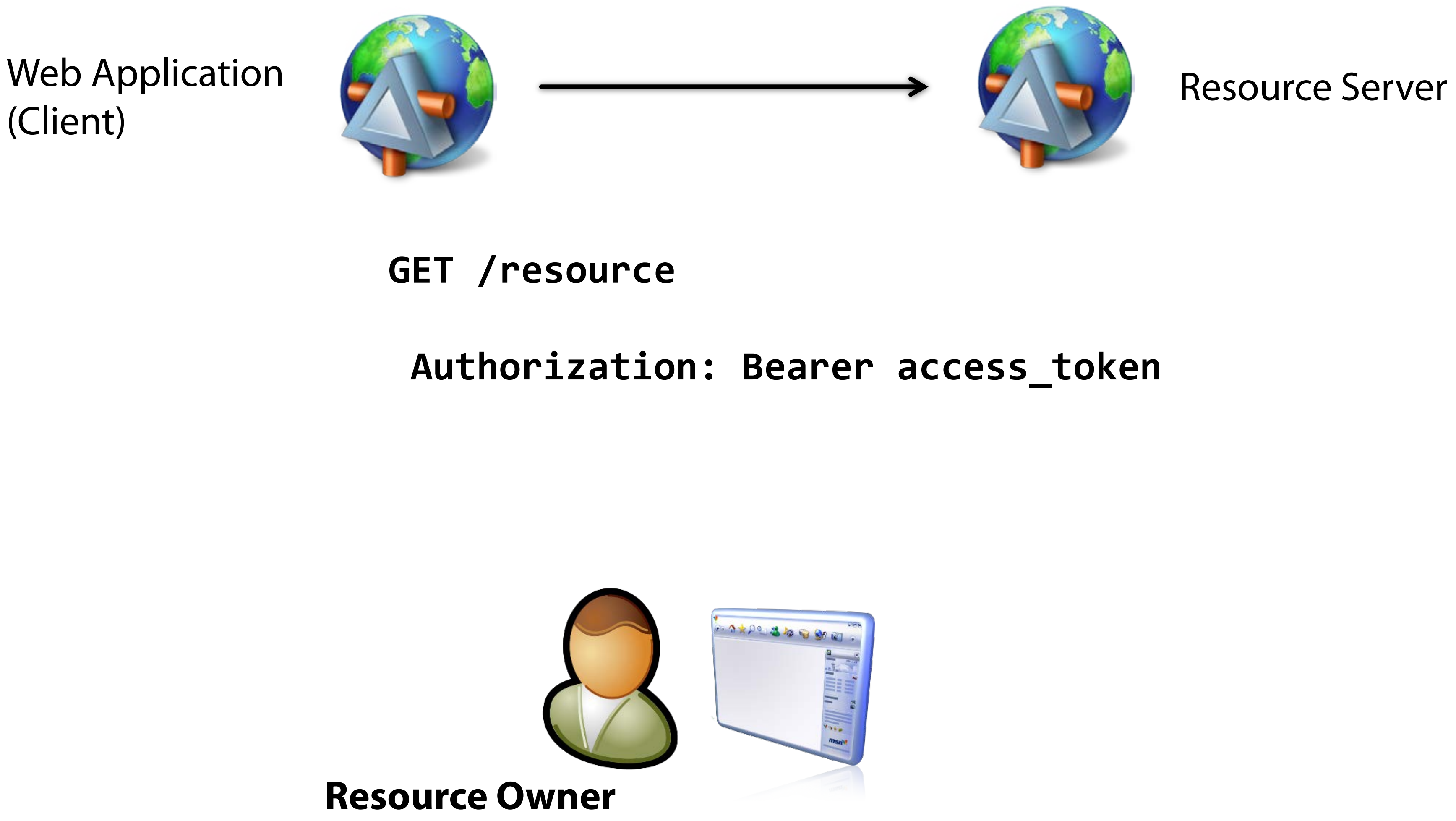
Authorization Server

```
{
  "access_token" : "abc",
  "expires_in" : "3600",
  "token_type" : "Bearer",
  "refresh_token" : "xyz"
}
```



Resource Owner

Step 3: Resource Access



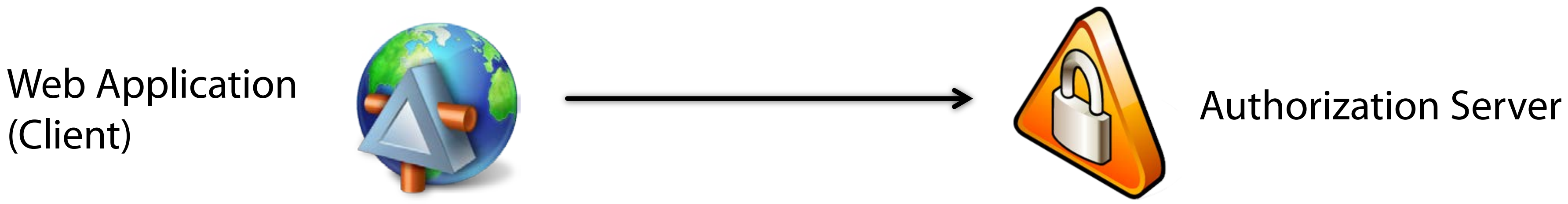
```
GET /resource

 Authorization: Bearer access_token
```

Access Token

- **The resource server will authorize the client & resource owner based on the contents of the access token**
  - after validation of issuer, signature and expiration

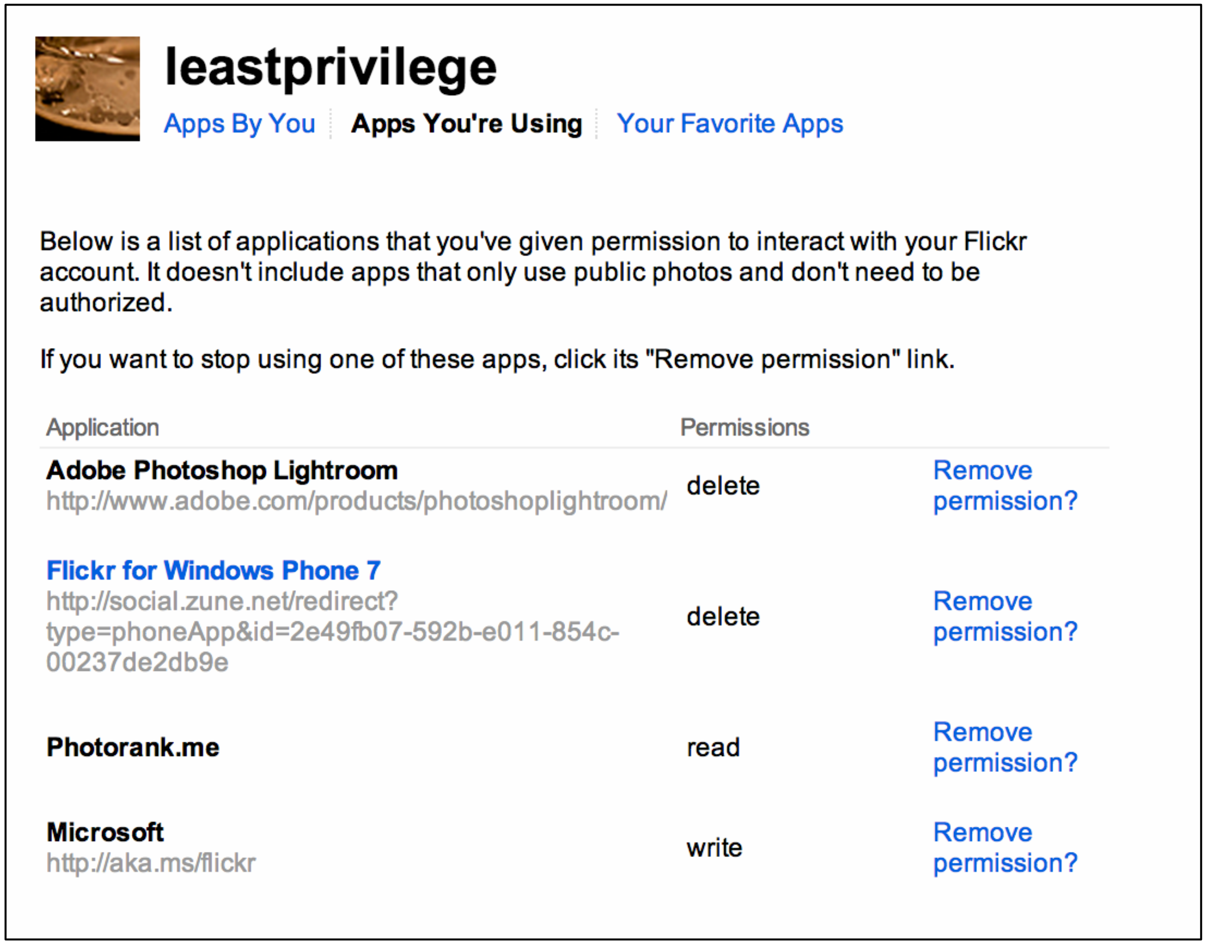
- **Typical claims for an access token are**
  - resource owner identifier
  - client identifier
  - granted scopes
  - …anything additional that makes sense for your application

(Step 4: Refreshing the Token)



```
POST /token
 Authorization: Basic (client_id:secret)

grant_type=refresh_token&
refresh_token=xyz
```

Client Management (Flickr)



leastprivilege

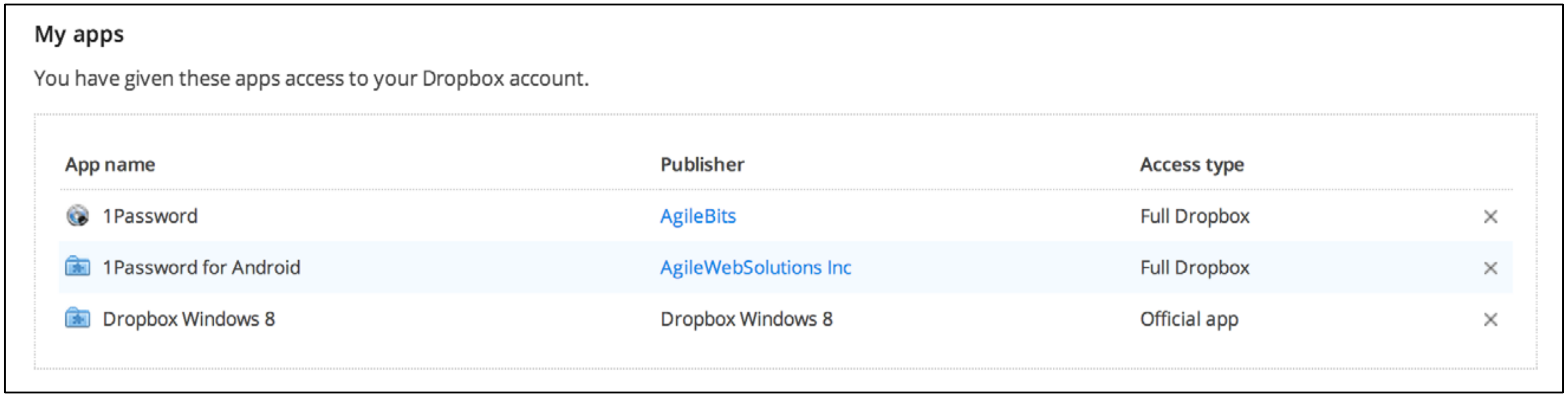
Apps By You | **Apps You're Using** | Your Favorite Apps

Below is a list of applications that you've given permission to interact with your Flickr account. It doesn't include apps that only use public photos and don't need to be authorized.

If you want to stop using one of these apps, click its "Remove permission" link.

Application	Permissions	
Adobe Photoshop Lightroom http://www.adobe.com/products/photoshoplightroom/	delete	Remove permission?
Flickr for Windows Phone 7 http://social.zune.net/redirect?type=phoneApp&id=2e49fb07-592b-e011-854c-00237de2db9e	delete	Remove permission?
Photorank.me	read	Remove permission?
Microsoft http://aka.ms/flickr	write	Remove permission?

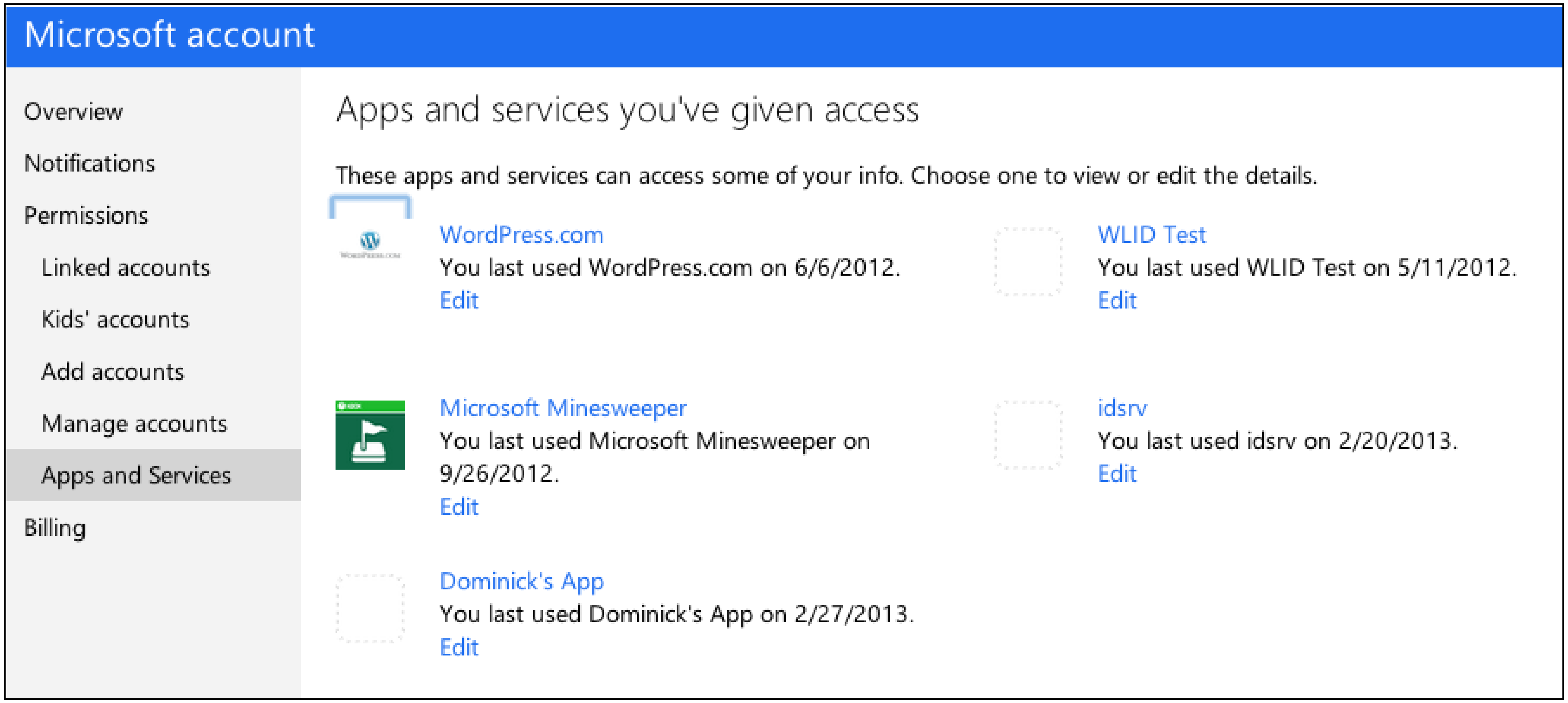
Client Management (Dropbox)


My apps

You have given these apps access to your Dropbox account.

App name	Publisher	Access type	
1Password	AgileBits	Full Dropbox	×
1Password for Android	AgileWebSolutions Inc	Full Dropbox	×
Dropbox Windows 8	Dropbox Windows 8	Official app	×

Client Management (Microsoft Live)



Microsoft account

- Overview
- Notifications
- Permissions
  - Linked accounts
  - Kids' accounts
  - Add accounts
  - Manage accounts
  - Apps and Services
- Billing

Apps and services you've given access

These apps and services can access some of your info. Choose one to view or edit the details.

WordPress.com
You last used WordPress.com on 6/6/2012.
Edit

WLID Test
You last used WLID Test on 5/11/2012.
Edit

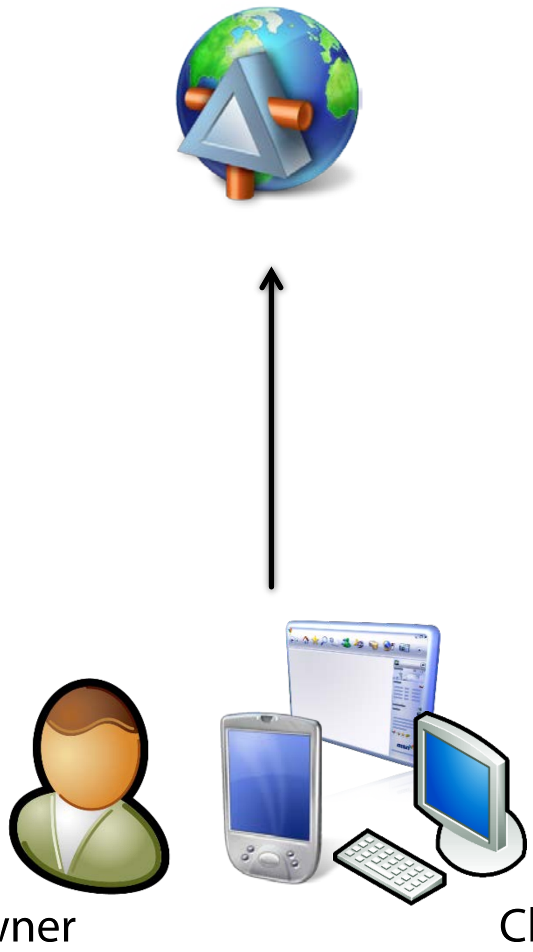
Microsoft Minesweeper
You last used Microsoft Minesweeper on 9/26/2012.
Edit

idsrv
You last used idsrv on 2/20/2013.
Edit

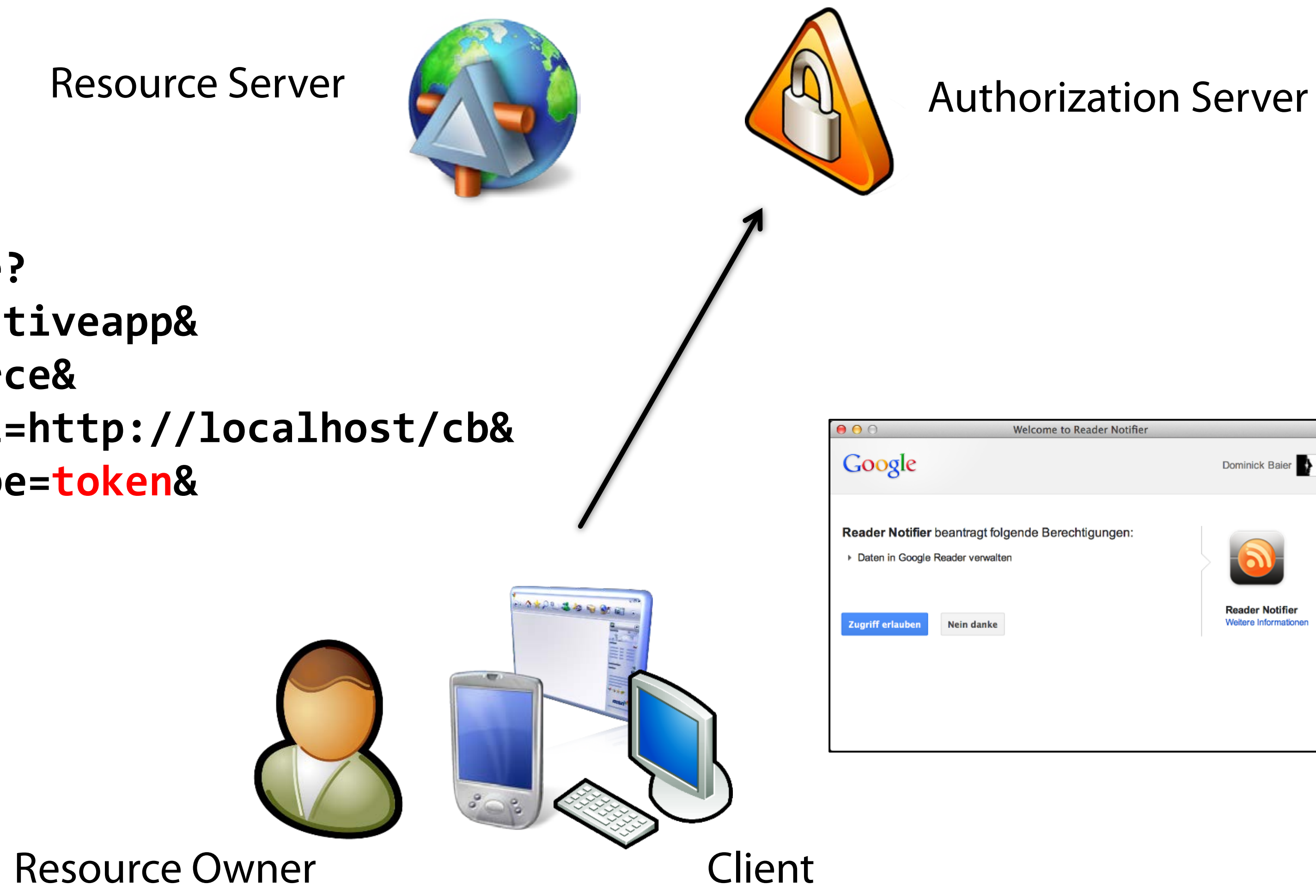
Dominick's App
You last used Dominick's App on 2/27/2013.
Edit

- **Designed for server-based applications**
  - Client can store secret securely on the server
- **Accountability is provided**
  - access token never leaked to the browser
- **Long-lived access can be implemented**

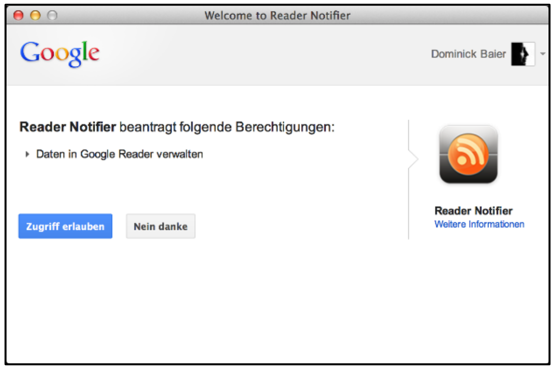
Implicit Flow (Native / Local Clients)



Step 1a: Authorization Request



```
GET /authorize?
  client_id=nativeapp&
  scope=resource&
  redirect_uri=http://localhost/cb&
  response_type=token&
  state=123
```


Welcome to Reader Notifier

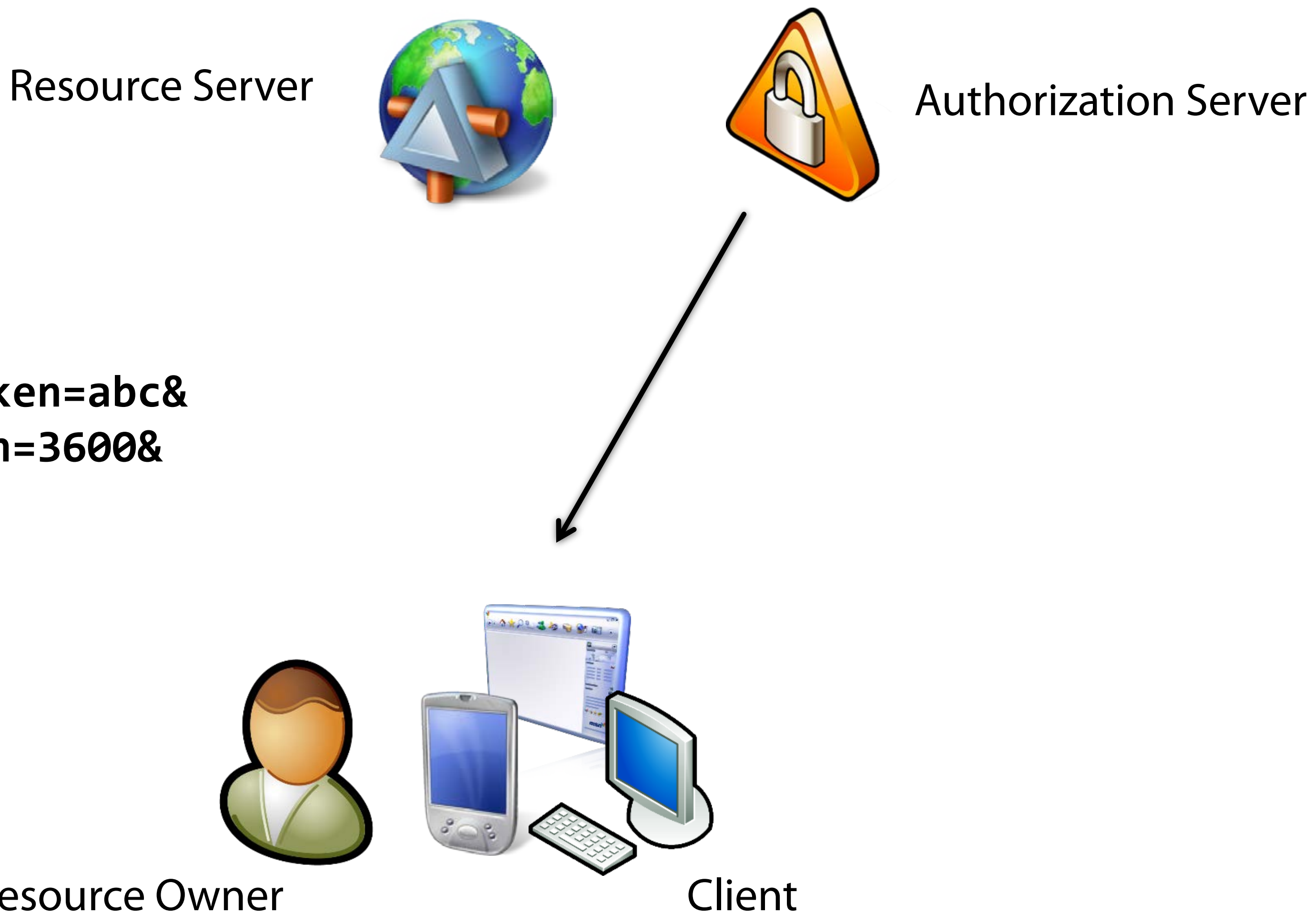
Google

Dominick Baier

Reader Notifier beantragt folgende Berechtigungen:

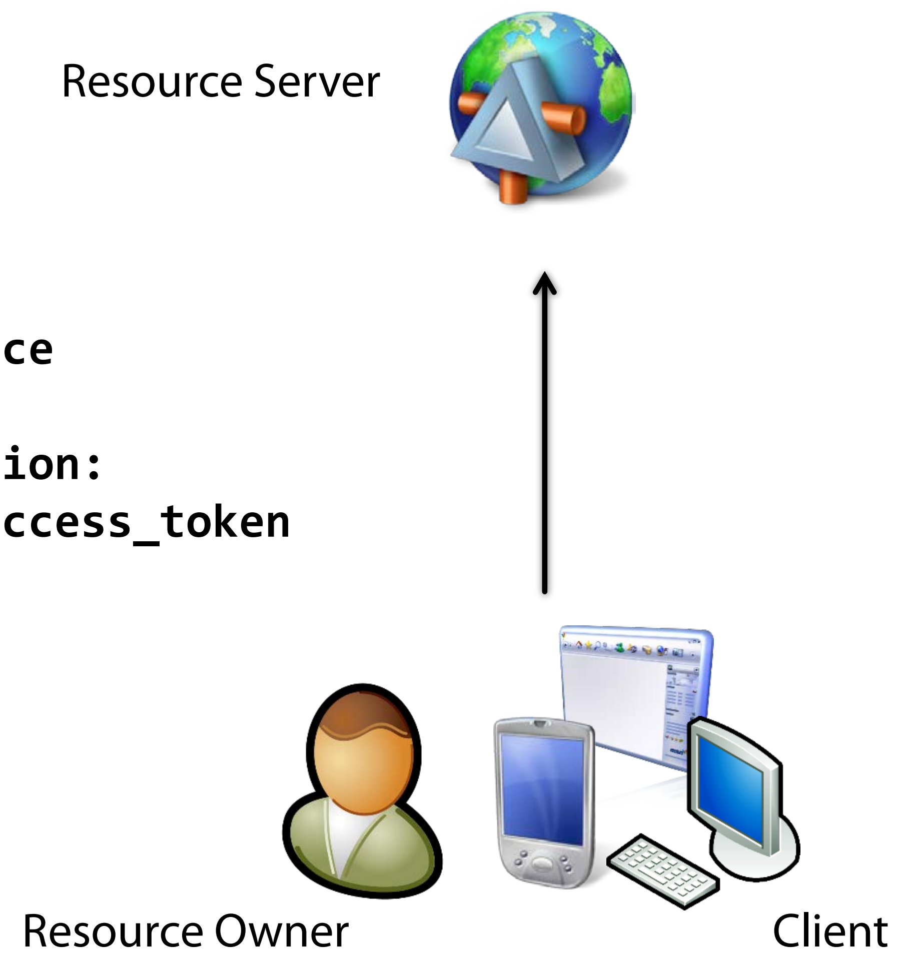
- Daten in Google Reader verwalten

Step 1b: Token Response



```
GET /cb#
  access_token=abc&
  expires_in=3600&
  state=123
```

Step 2: Resource Access

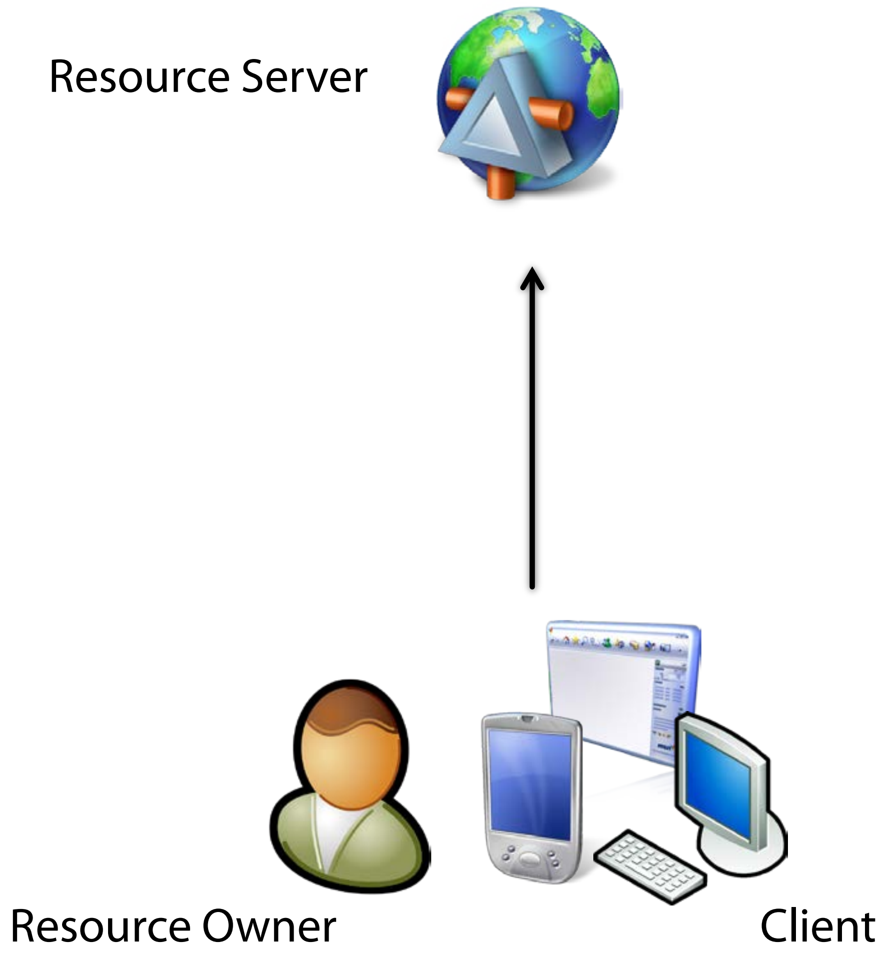


```
GET /resource

 Authorization:
   Bearer access_token
```

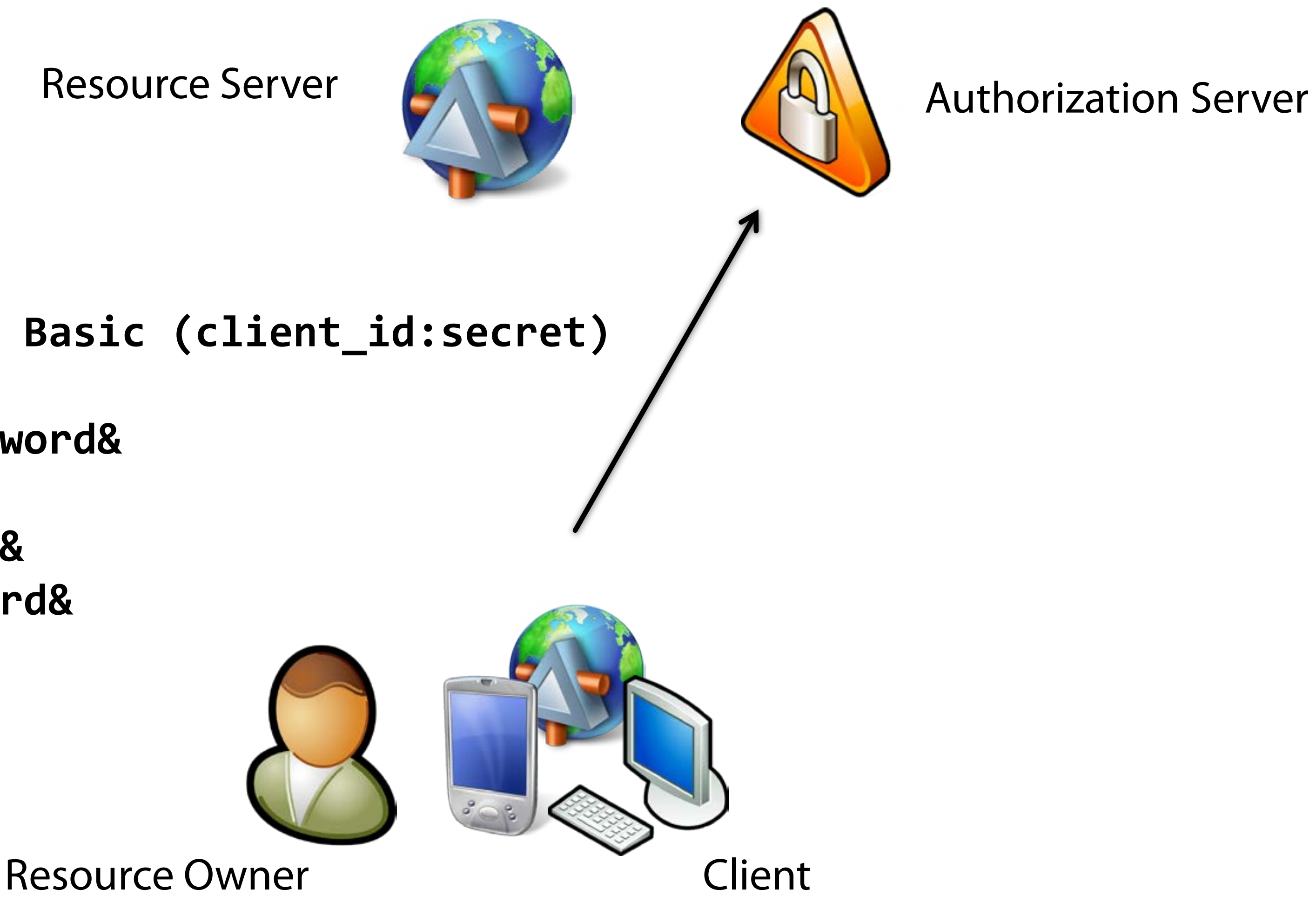
- **Simplified handshake**
  - no authorization code
- **Token is exposed to browser / local OS**
- **No client authentication**
  - no refresh tokens

- **Heavily debated and many "non-standard" variations**

Resource Owner Password Credential Flow (Trusted Application)

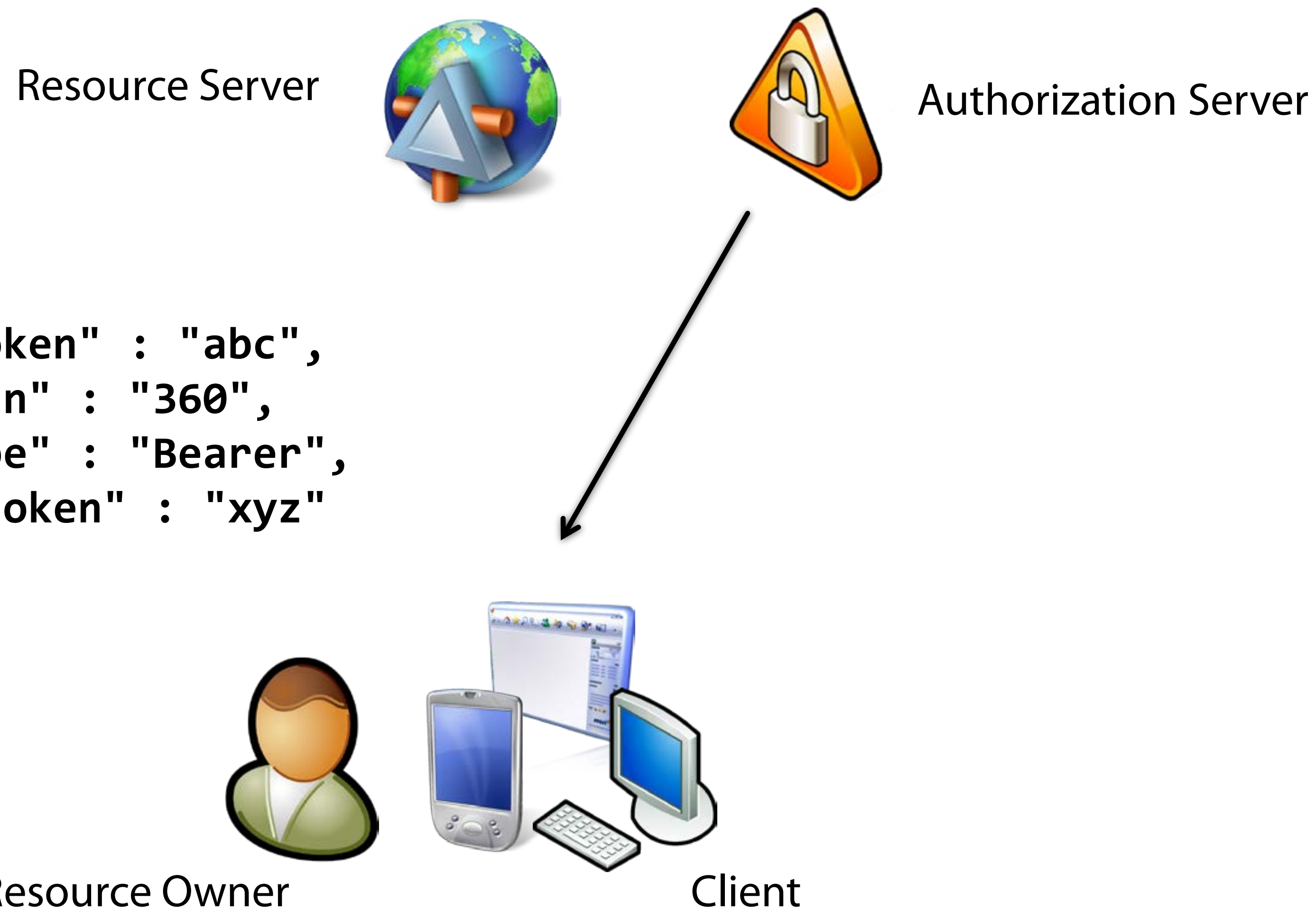


Step 1a: Token Request



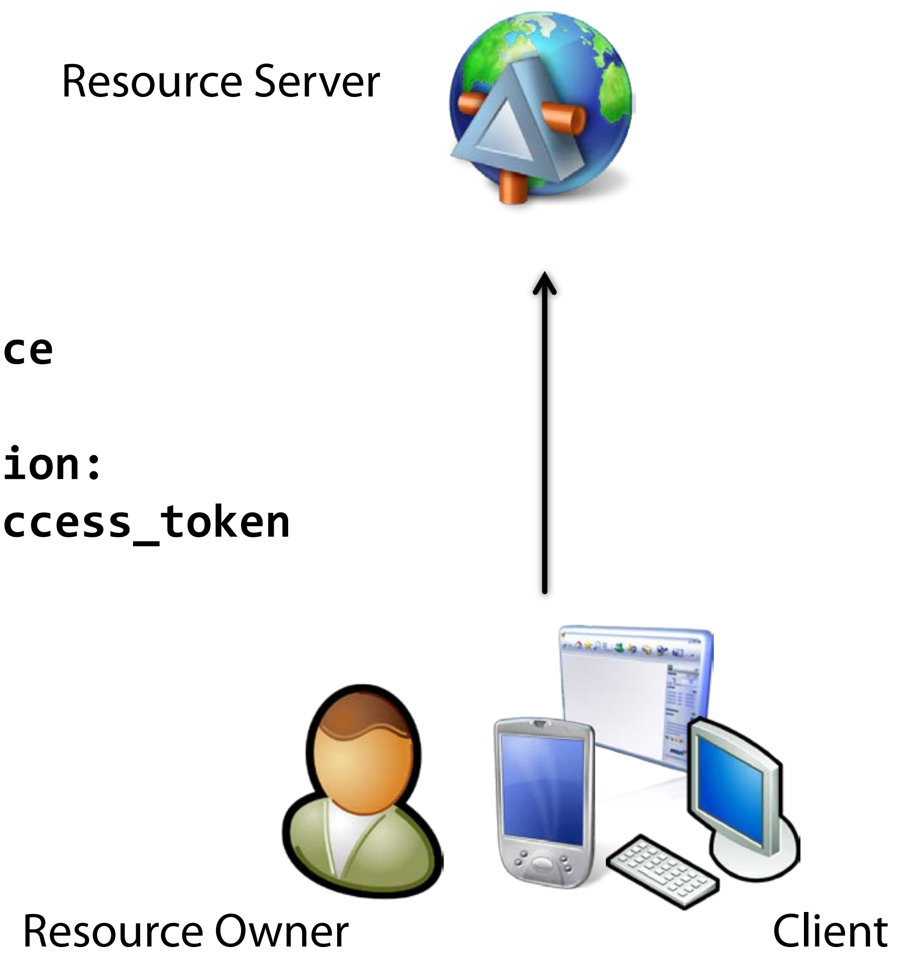
```
POST /token
 Authorization: Basic (client_id:secret)

grant_type=password&
scope=resource&
user_name=owner&
password=password&
```

Step 1b: Token Response



```
{
  "access_token" : "abc",
  "expires_in" : "360",
  "token_type" : "Bearer",
  "refresh_token" : "xyz"
}
```

Step 2: Resource Access

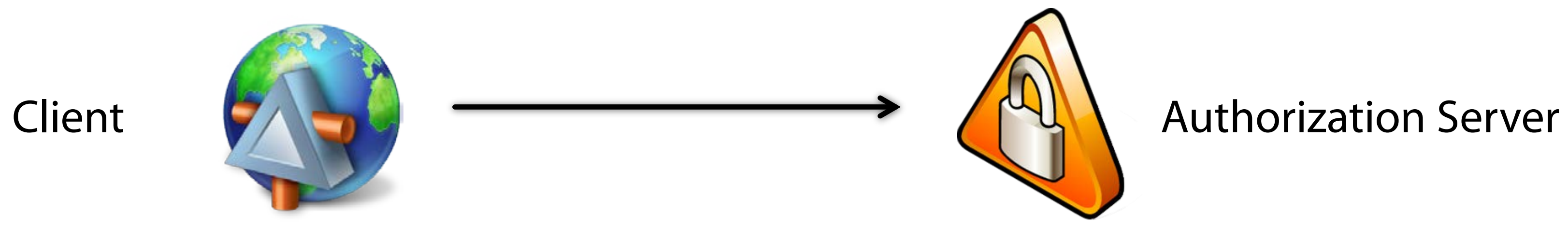


```
GET /resource

 Authorization:
   Bearer access_token
```

Summary –
Resource Owner Credential Flow

- **Resource owner credentials are exposed to client**
  - users should not become accustomed to that
- **Still better to store access/refresh token on device than password**
  - if the developer is using that feature

Client Credentials Flow – No human involved at all



```
POST /token
 Authorization: Basic (client_id:secret)

grant_type=client_credentials&
scope=resource
```

- **The OAuth2 flows describe the various options for**
  - request authorization
  - request tokens

- **A separate spec (RFC 6750) describes how to transmit bearer access tokens to the resource server**

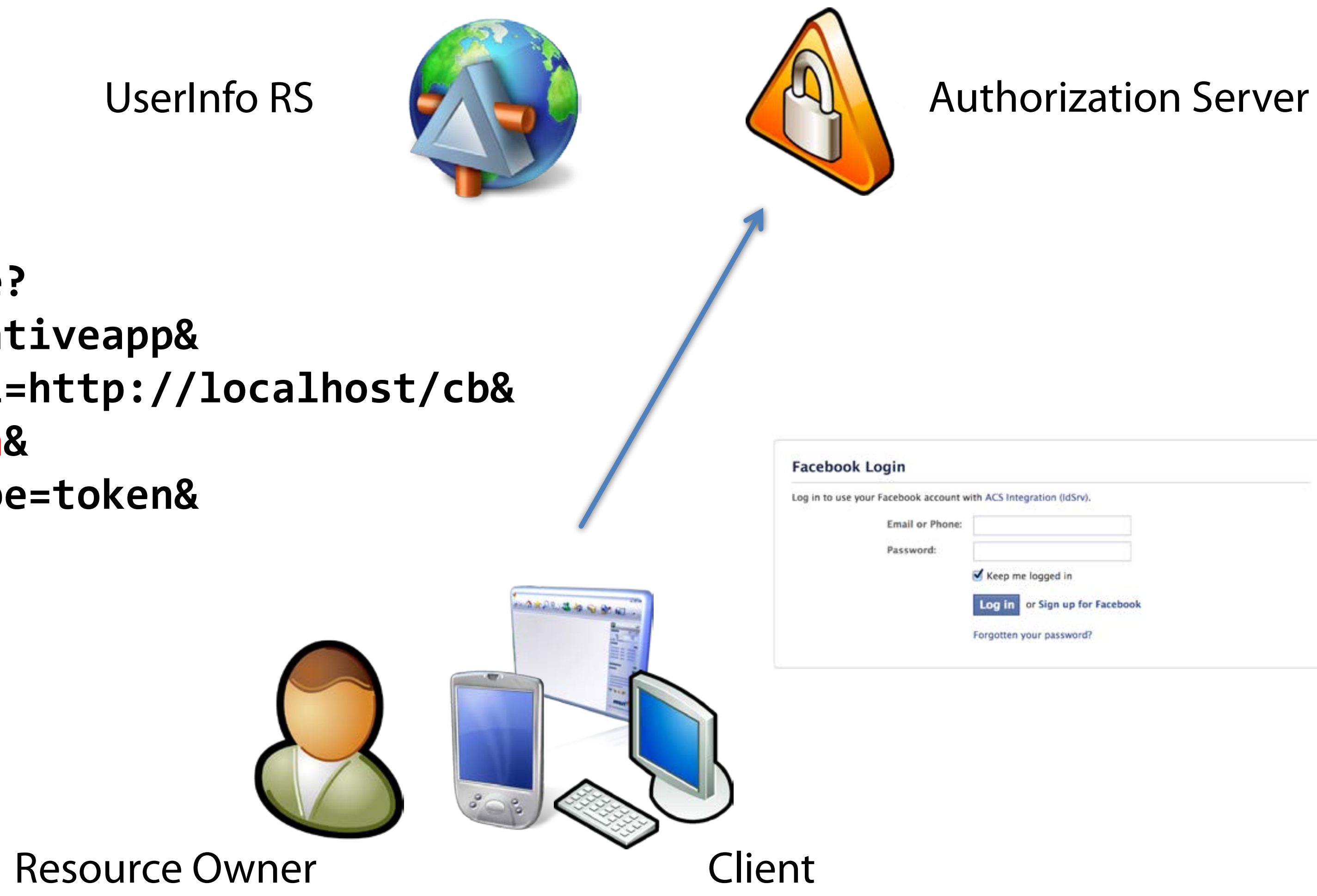
Outline

- OAuth2 and authentication
- OpenID Connect
- Flows

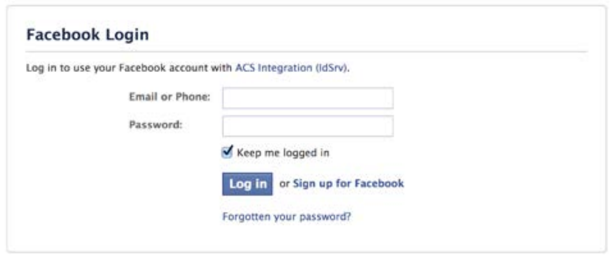
OAuth2 & Authentication

- **OAuth2 is for (delegated) authorization**
  - authentication is a pre-requisite for that
  - access token is for some back-end service
- **Sometimes you "just" need authentication**
  - (at least to begin with)
  - identify user in an application
  - control access to application features
- **OAuth2 is regularly "abused" for that**

OAuth2 for Authentication: Request



```
GET /authorize?
  client_id=nativeapp&
  redirect_uri=http://localhost/cb&
  scope=signin&
  response_type=token&
  state=123
```


Facebook Login

Log in to use your Facebook account with ACS Integration (IdSrv).

Email or Phone:

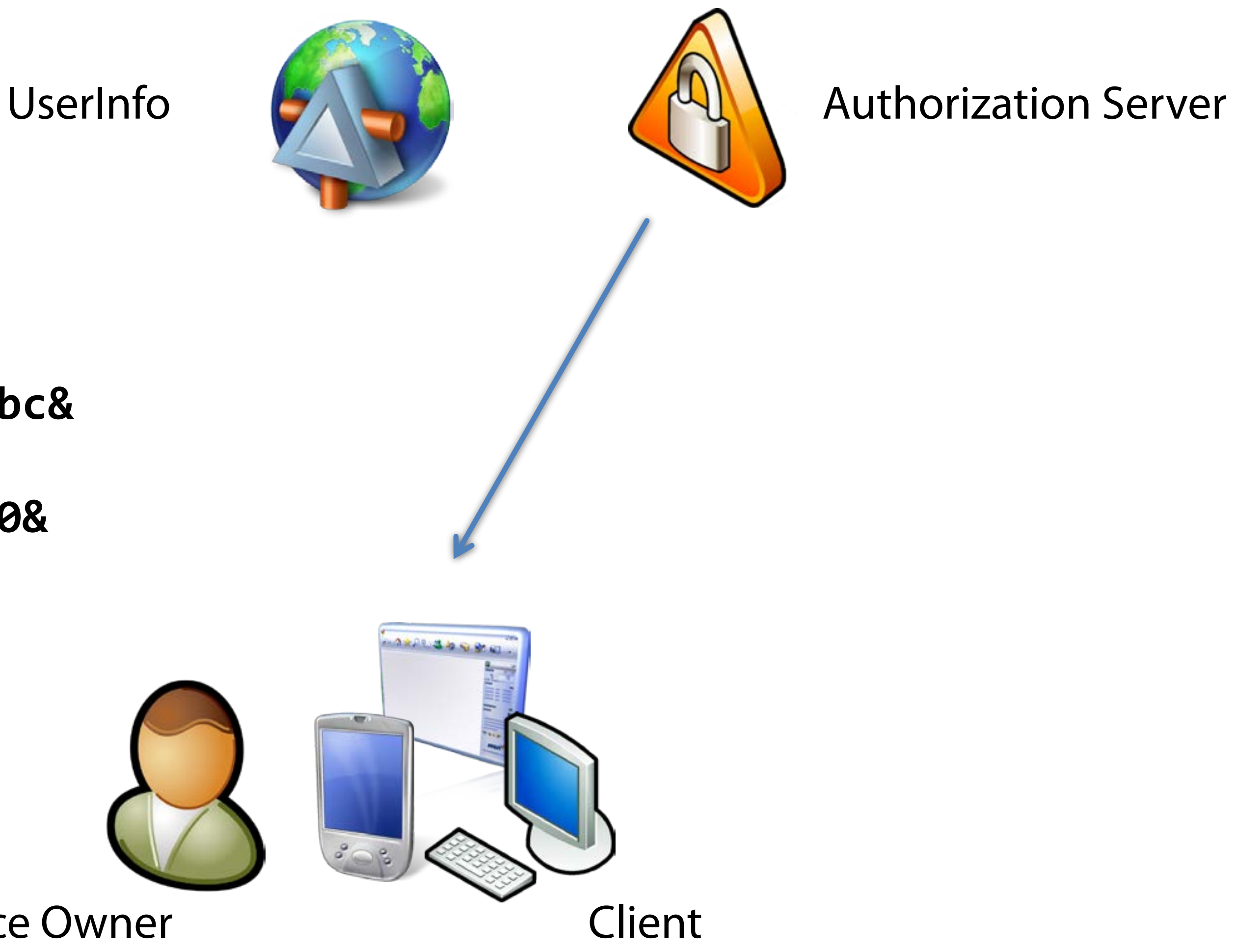
Password:

✓ Keep me logged in

Log in or Sign up for Facebook

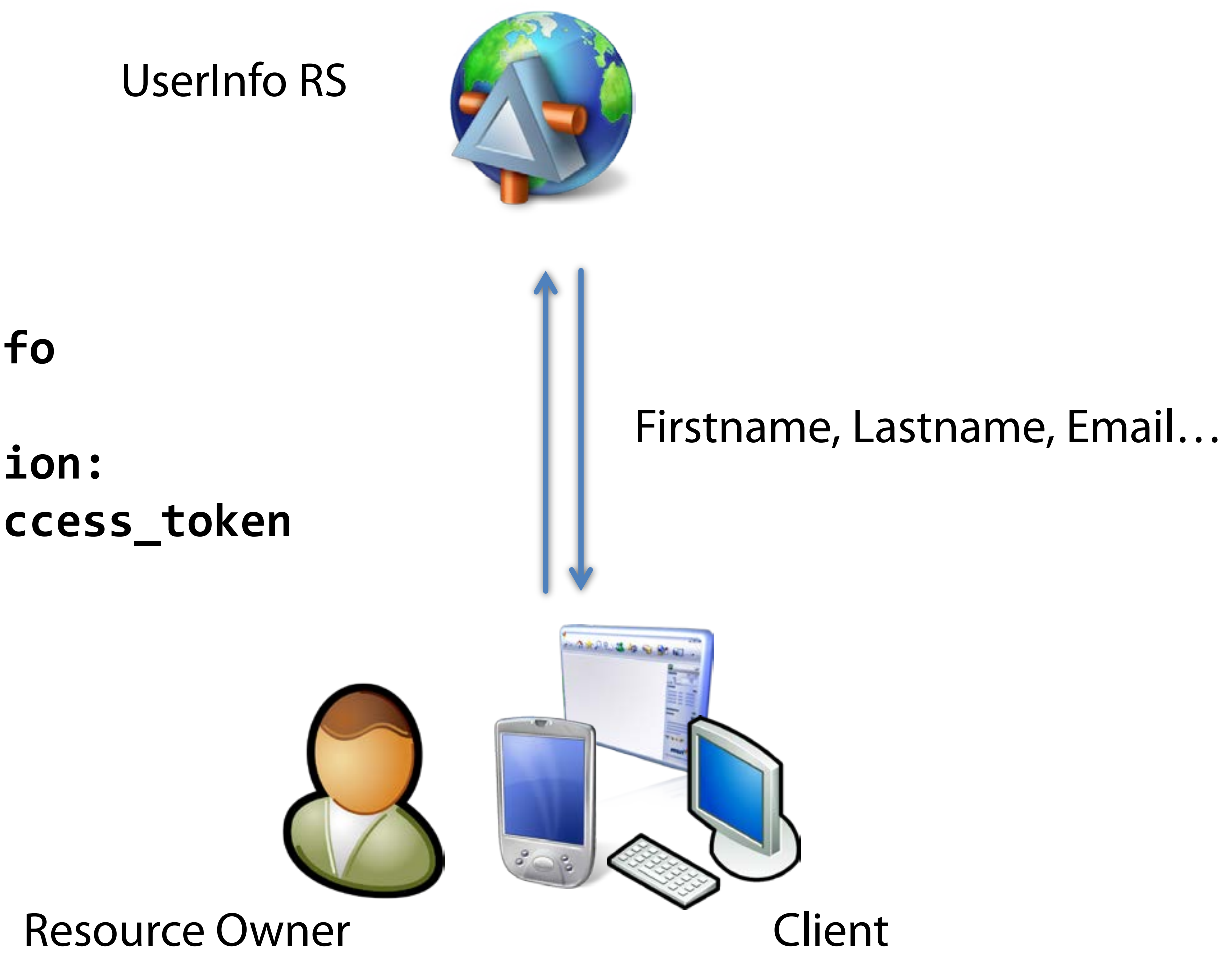
Forgotten your password?

OAuth2 for Authentication: Response



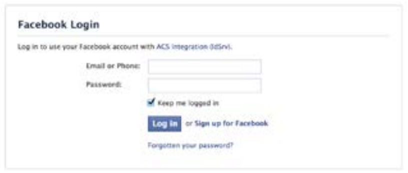
```
GET /cb?
  access_token=abc&
  userid=123&
  expires_in=3600&
  state=123
```

OAuth2 for Authentication: Accessing User Data



```
GET /userinfo

 Authorization:
   Bearer access_token
```

The Problem



Facebook Login

Log in to use your Facebook account with ACS Integration (IdSrv).

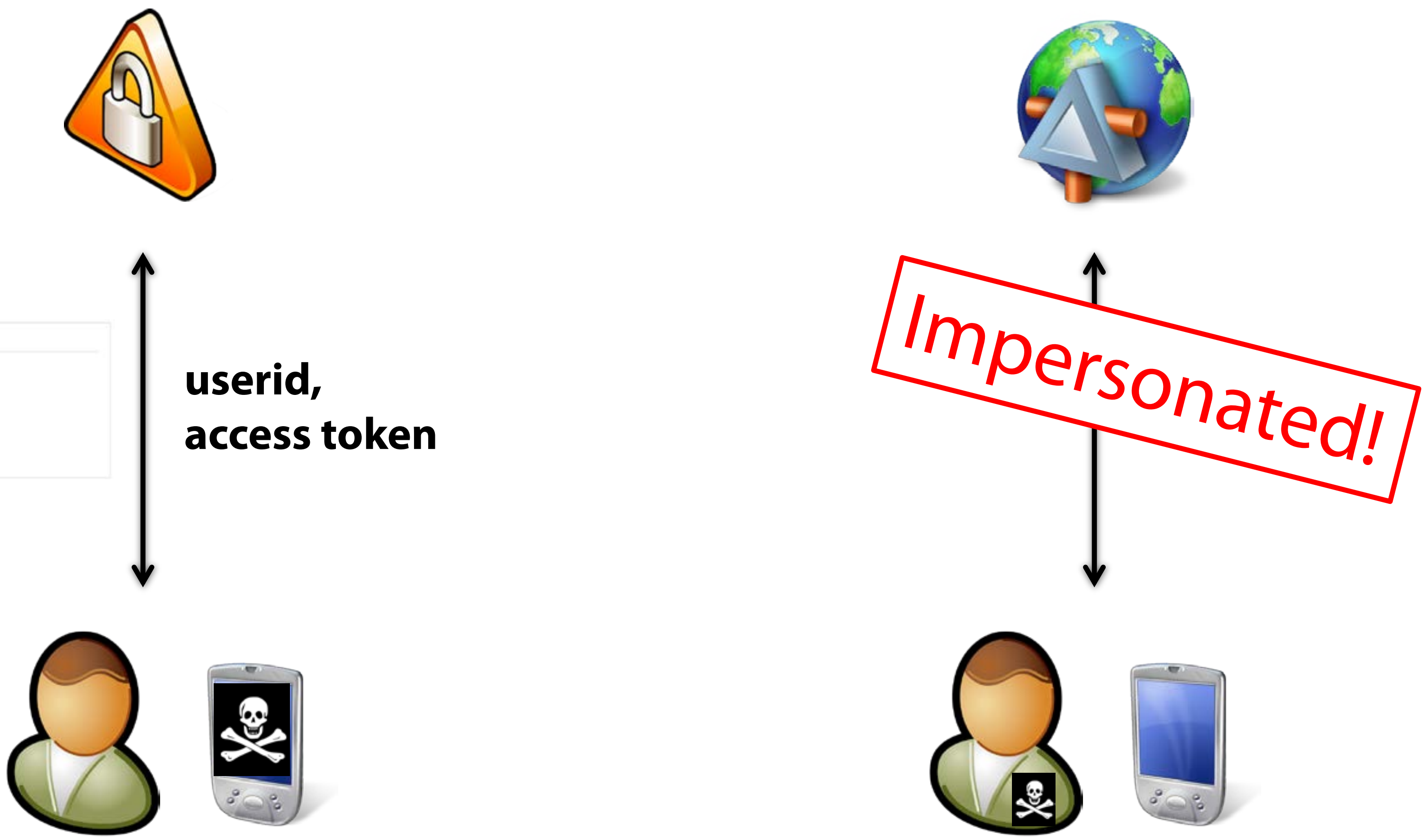
Email or Phone:

Password:

[✓] Keep me logged in

Log in or Sign up for Facebook

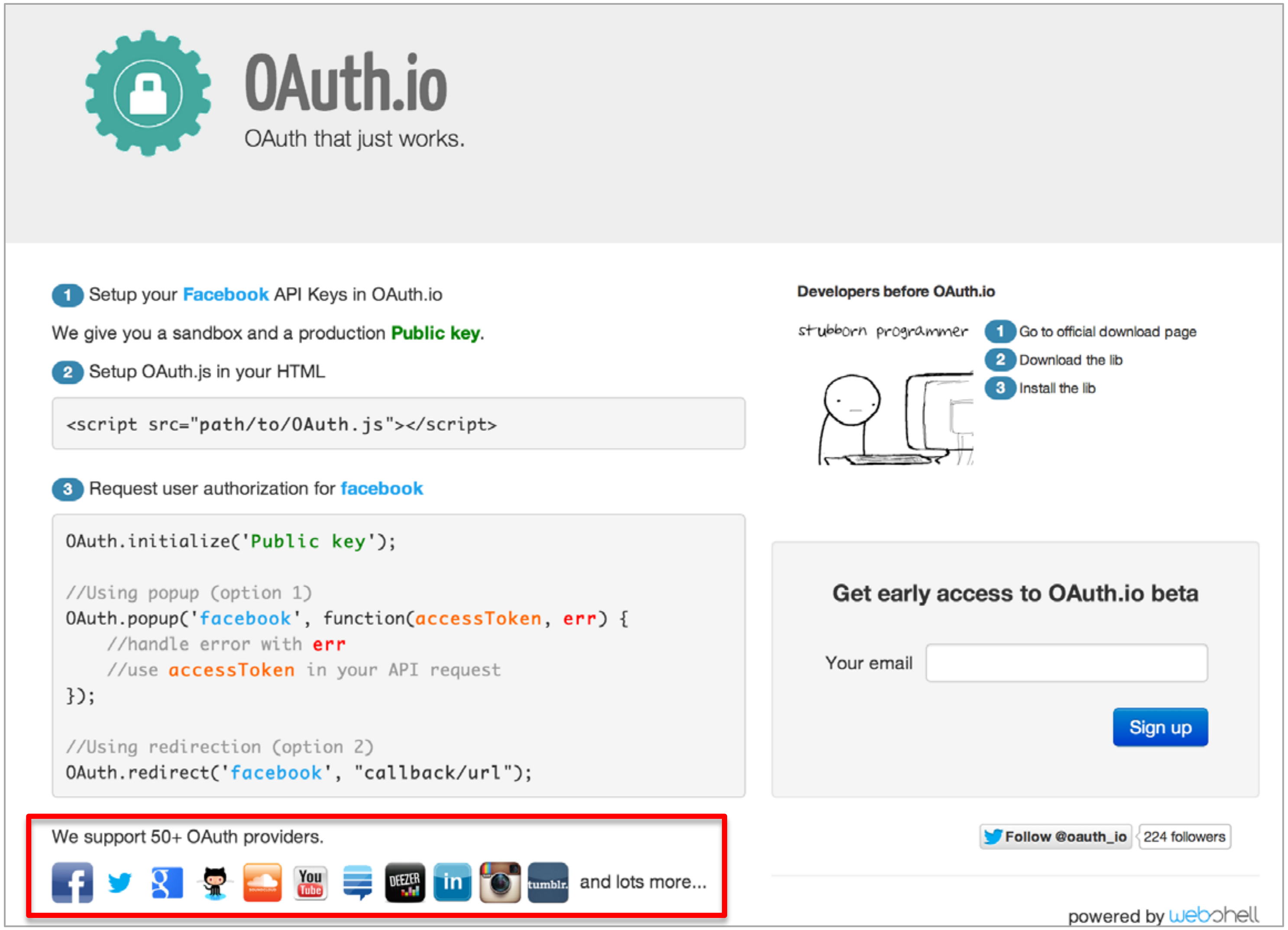
Forgotten your password?



1. User logs into malicious app (app steals token)

2. Malicious developer uses stolen access token in legitimate app

The Solution?



OAuth.io

OAuth that just works.

1 Setup your **Facebook** API Keys in OAuth.io

We give you a sandbox and a production **Public key**.

2 Setup OAuth.js in your HTML

```
<script src="path/to/OAuth.js"></script>
```

3 Request user authorization for **facebook**

```
OAuth.initialize('Public key');

//Using popup (option 1)
OAuth.popup('facebook', function(accessToken, err) {
    //handle error with err
    //use accessToken in your API request
});

//Using redirection (option 2)
OAuth.redirect('facebook', "callback/url");
```

We support 50+ OAuth providers.

and lots more...

Developers before OAuth.io

1 Go to official download page

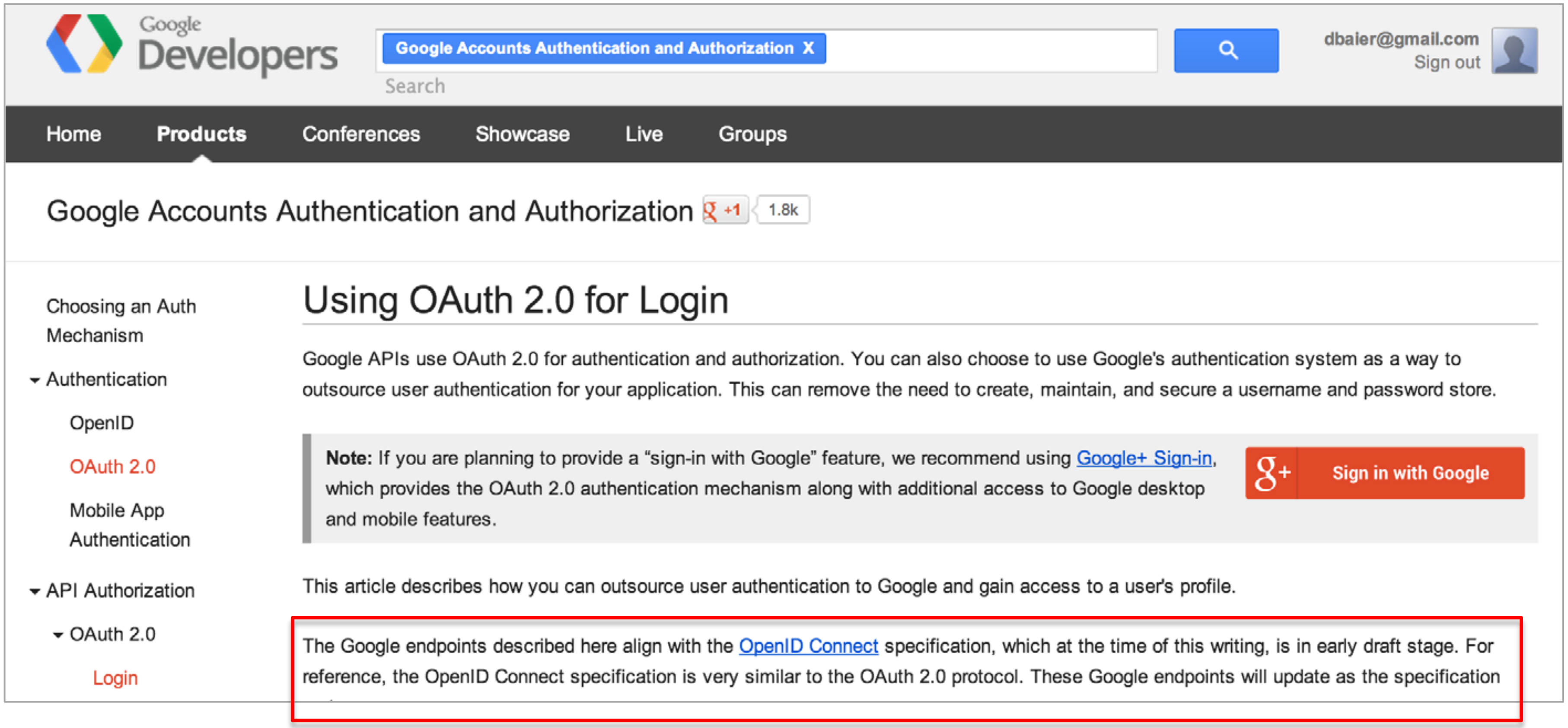
2 Download the lib

3 Install the lib

Get early access to OAuth.io beta

Your email

OpenID Connect



Google Accounts Authentication and Authorization

Using OAuth 2.0 for Login

Google APIs use OAuth 2.0 for authentication and authorization. You can also choose to use Google's authentication system as a way to outsource user authentication for your application. This can remove the need to create, maintain, and secure a username and password store.

Note: If you are planning to provide a "sign-in with Google" feature, we recommend using Google+ Sign-in, which provides the OAuth 2.0 authentication mechanism along with additional access to Google desktop and mobile features.

This article describes how you can outsource user authentication to Google and gain access to a user's profile.

The Google endpoints described here align with the OpenID Connect specification, which at the time of this writing, is in early draft stage. For reference, the OpenID Connect specification is very similar to the OAuth 2.0 protocol. These Google endpoints will update as the specification

OpenID Connect Flows

- **OpenID Connect builds on top of OAuth2**
  - Authorization Code Flow
  - Implicit Flow
- **Adds some new concepts**
  - ID Token
  - UserInfo endpoint
- **..and some additional protocols, e.g.**
  - discovery & dynamic registration
  - session management

http://openid.net/connect/

OpenID Connect: The Players

Identity Provider

Authorization Endpoint

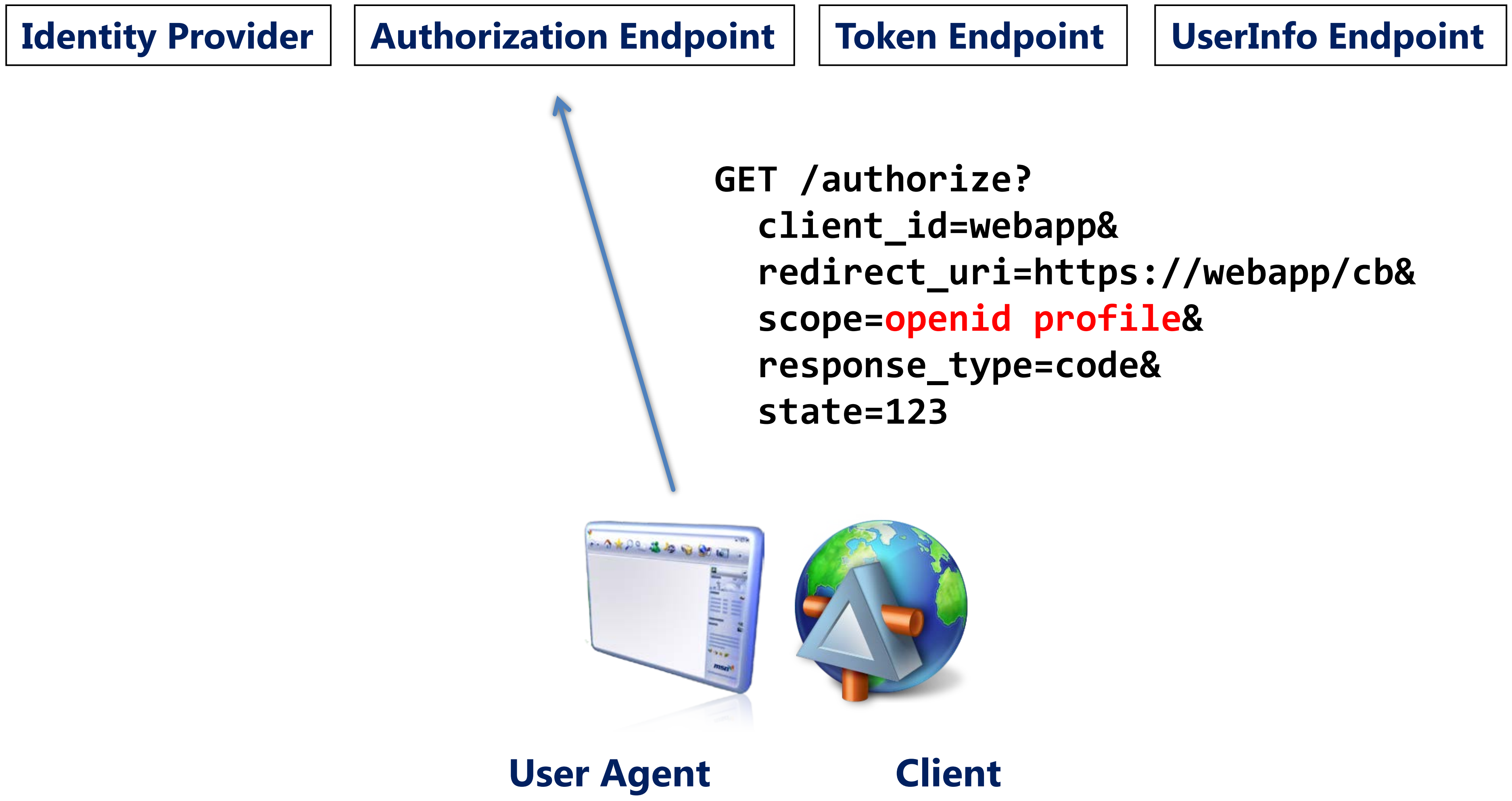
Token Endpoint

UserInfo Endpoint

User Agent

Client

Step 1a: Authorization Request



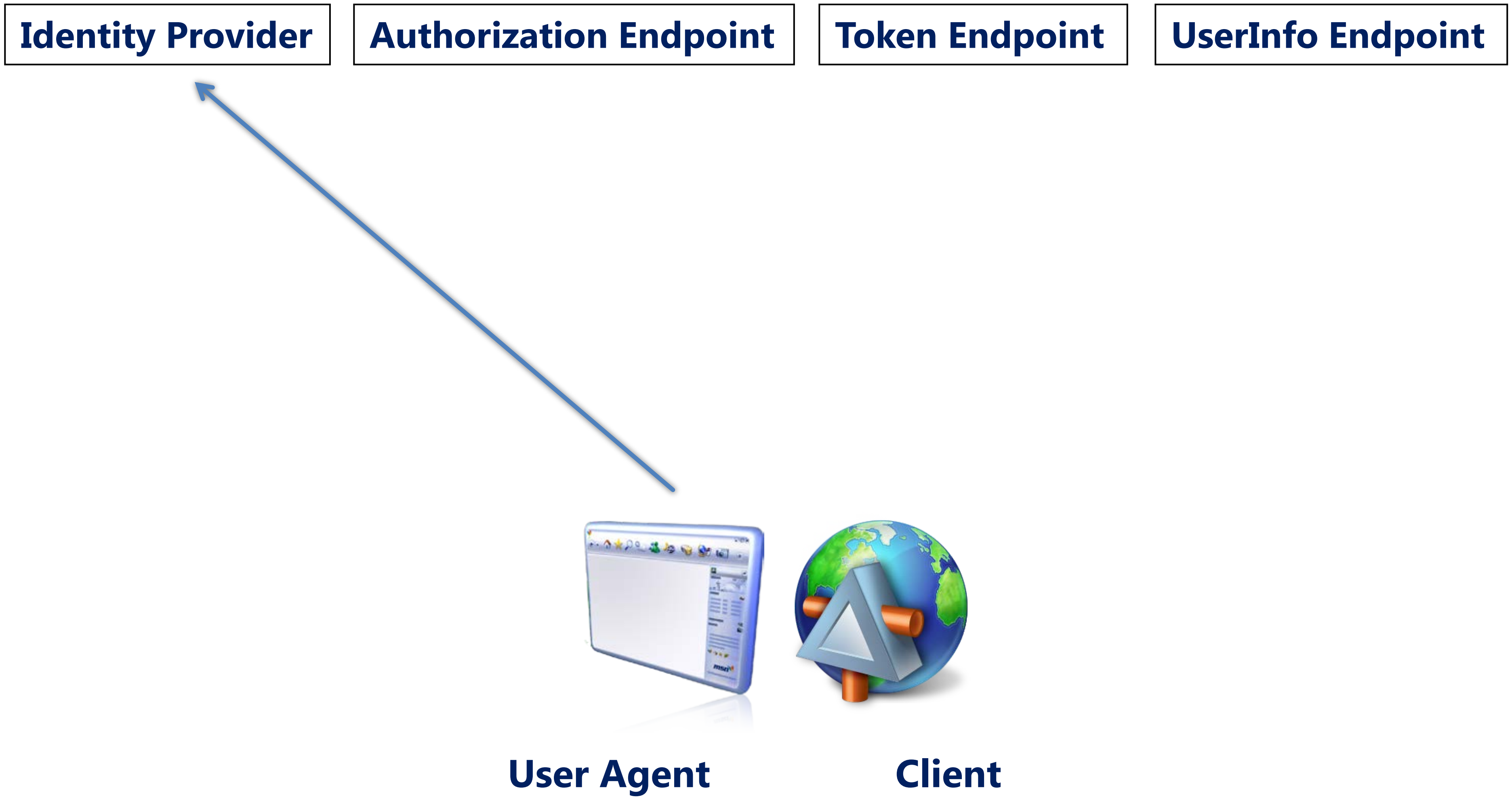
```
GET /authorize?
  client_id=webapp&
  redirect_uri=https://webapp/cb&
  scope=openid profile&
  response_type=code&
  state=123
```

Scopes & Claims

- **OpenID defines a set of standard scopes and claims**

Scope	Claims
profile	name, family_name, given_name, middle_name, nickname, preferred_username, profile, picture, website, gender, birthdate, zoneinfo, locale, and updated_at.
email	email, email_verified
address	address
phone	phone_number, phone_number_verified
offline_access	requests refresh token

Step 1b: Authentication



Step 1c: Consent

Identity Provider

Authorization Endpoint

Token Endpoint

UserInfo Endpoint

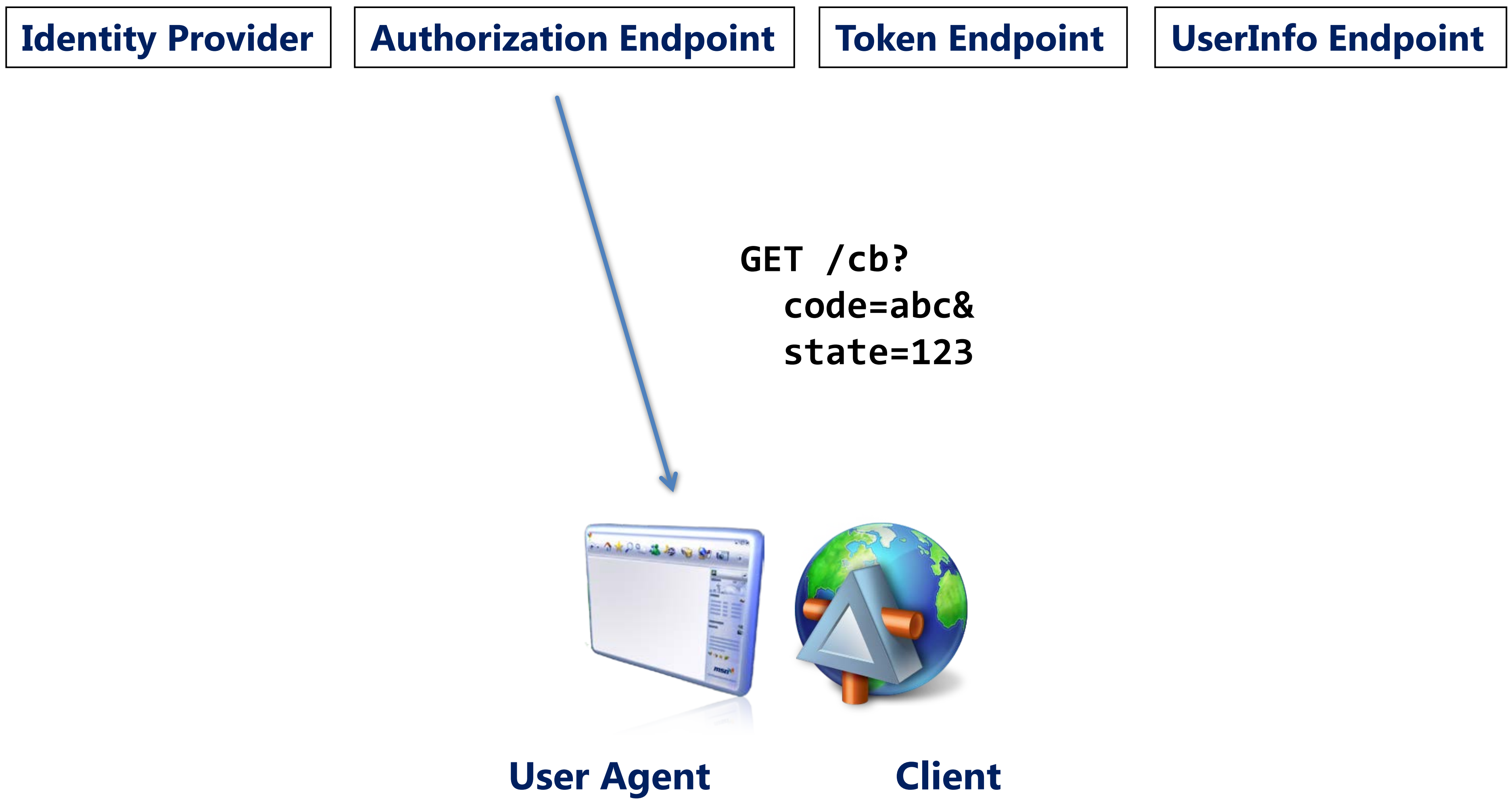
OpenID Connect

Application **WebApp** asks
for permission to access your profile

User Agent

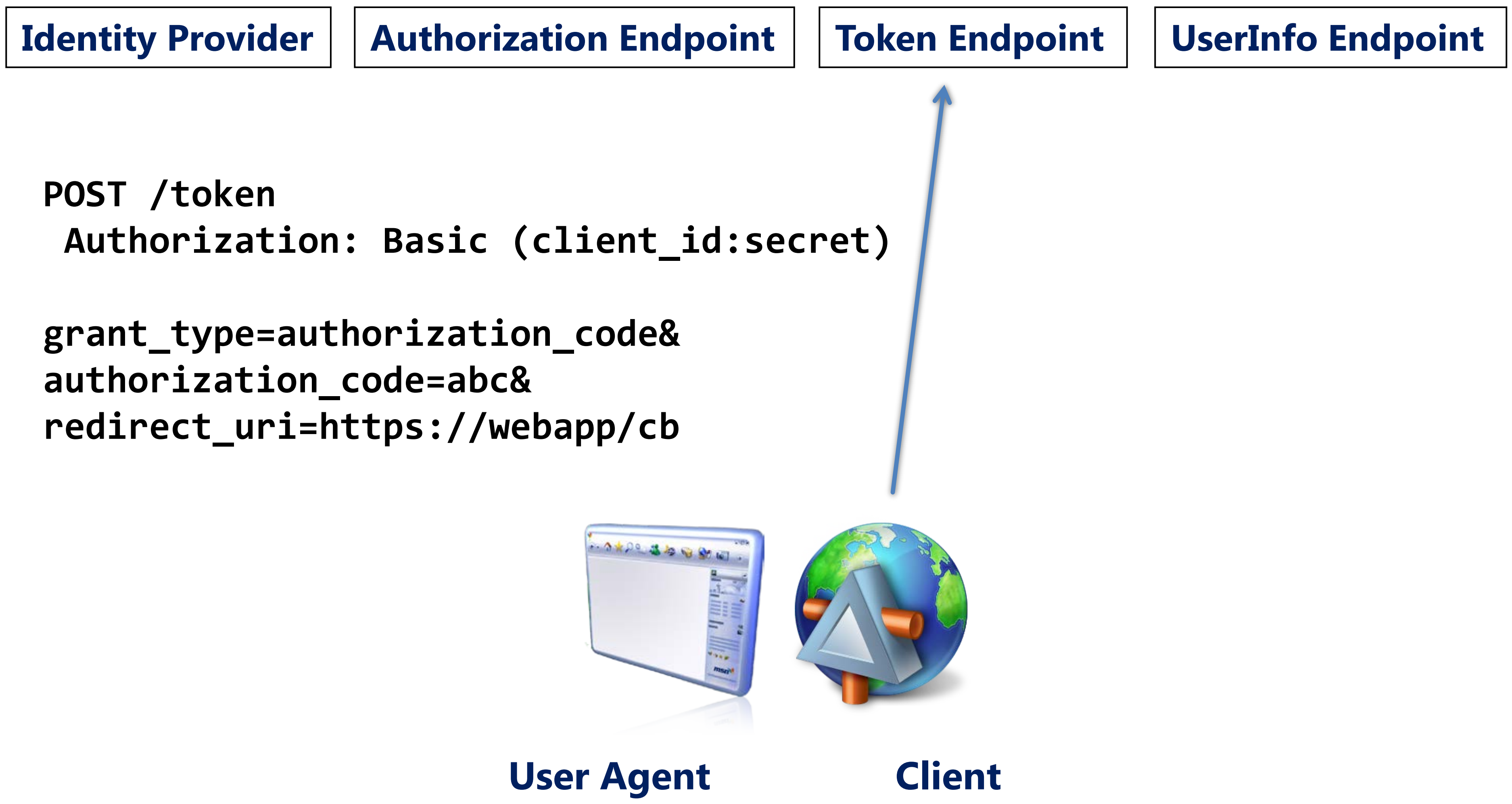
Client

Step 1d: Authorization Response



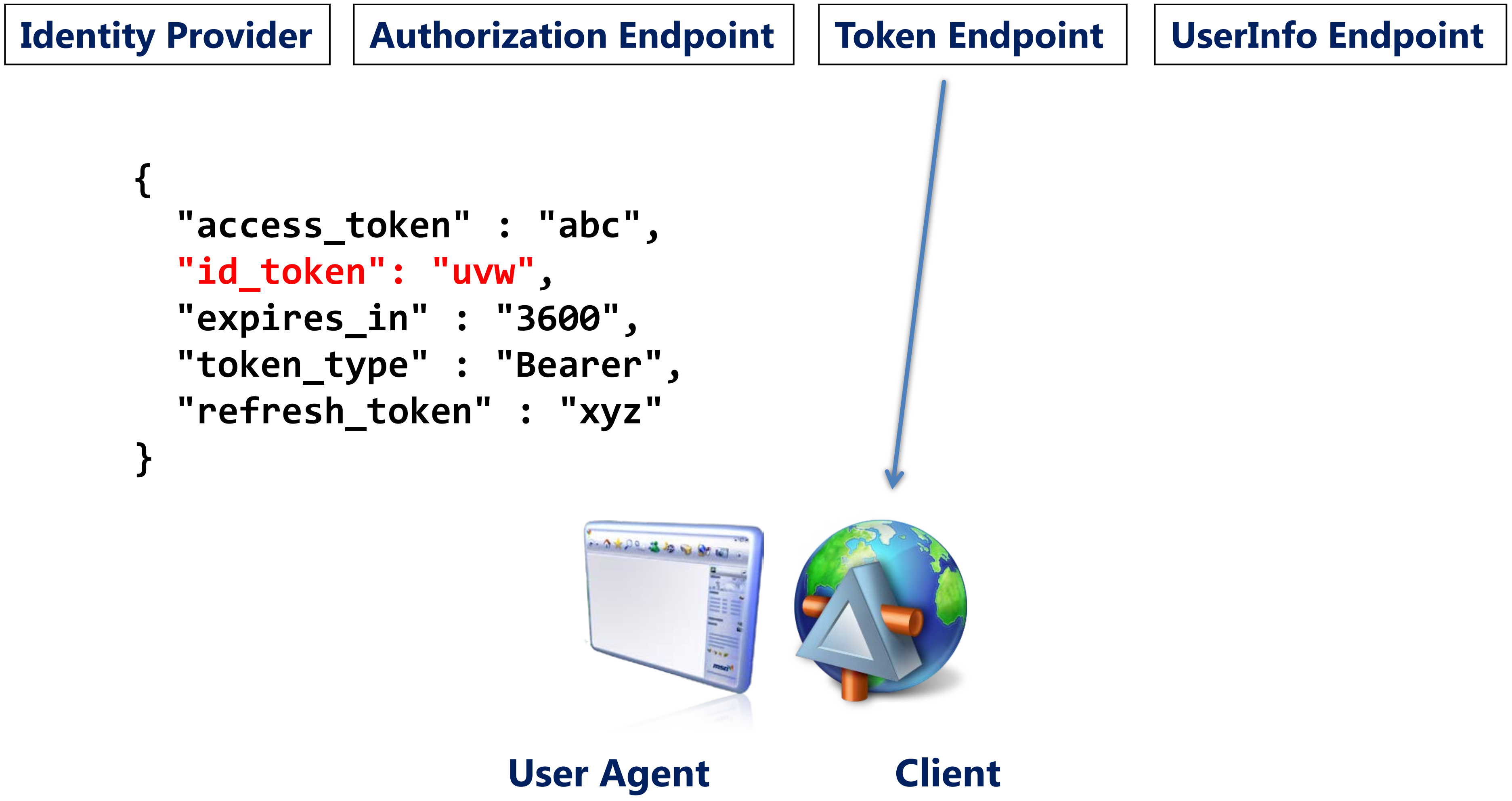
```
GET /cb?
  code=abc&
  state=123
```

Step 2a: Token Request



```
POST /token
 Authorization: Basic (client_id:secret)

grant_type=authorization_code&
authorization_code=abc&
redirect_uri=https://webapp/cb
```

Step 2b: Token Response

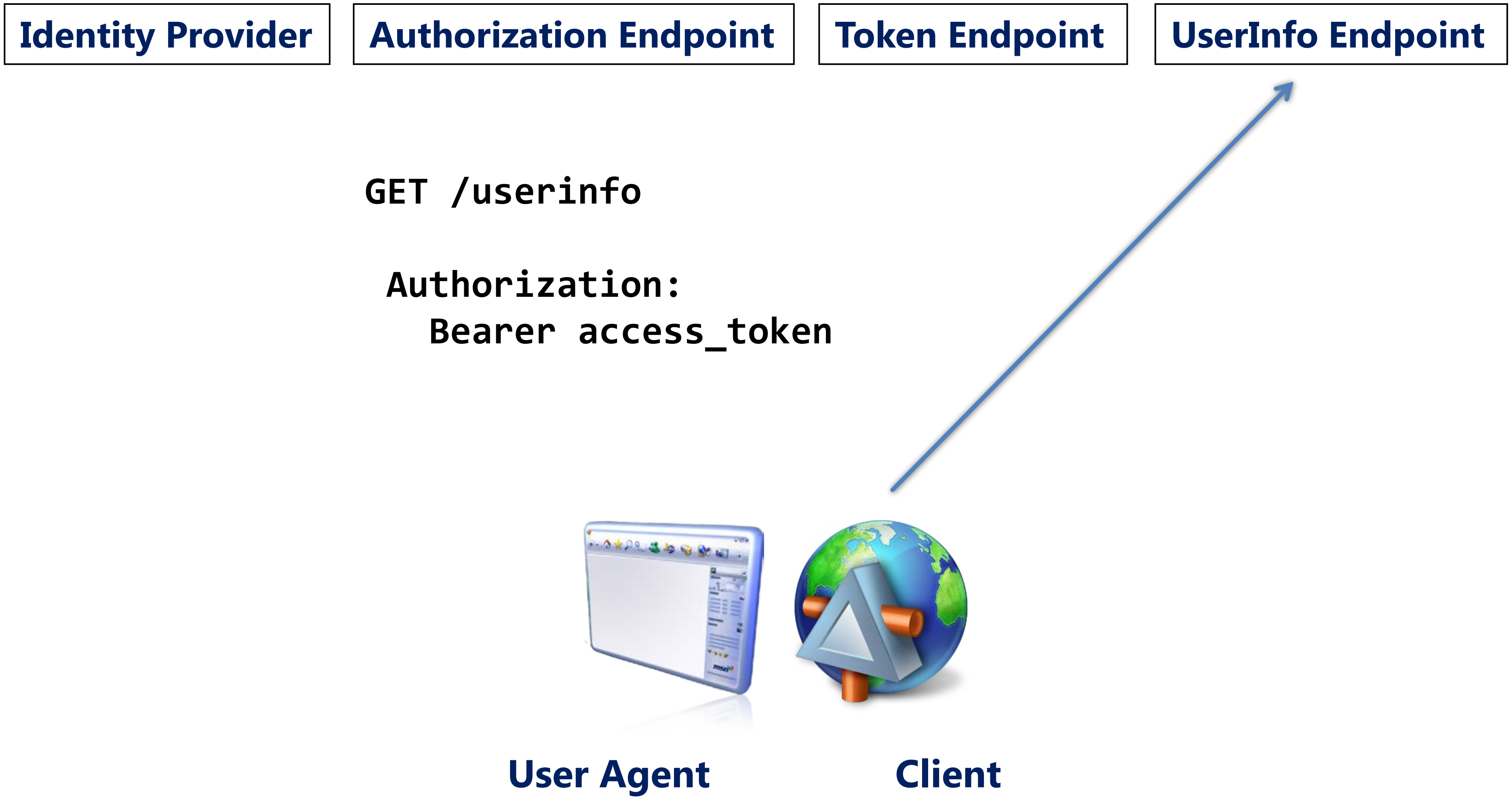


```
{
  "access_token" : "abc",
  "id_token": "uvw",
  "expires_in" : "3600",
  "token_type" : "Bearer",
  "refresh_token" : "xyz"
}
```

ID Token

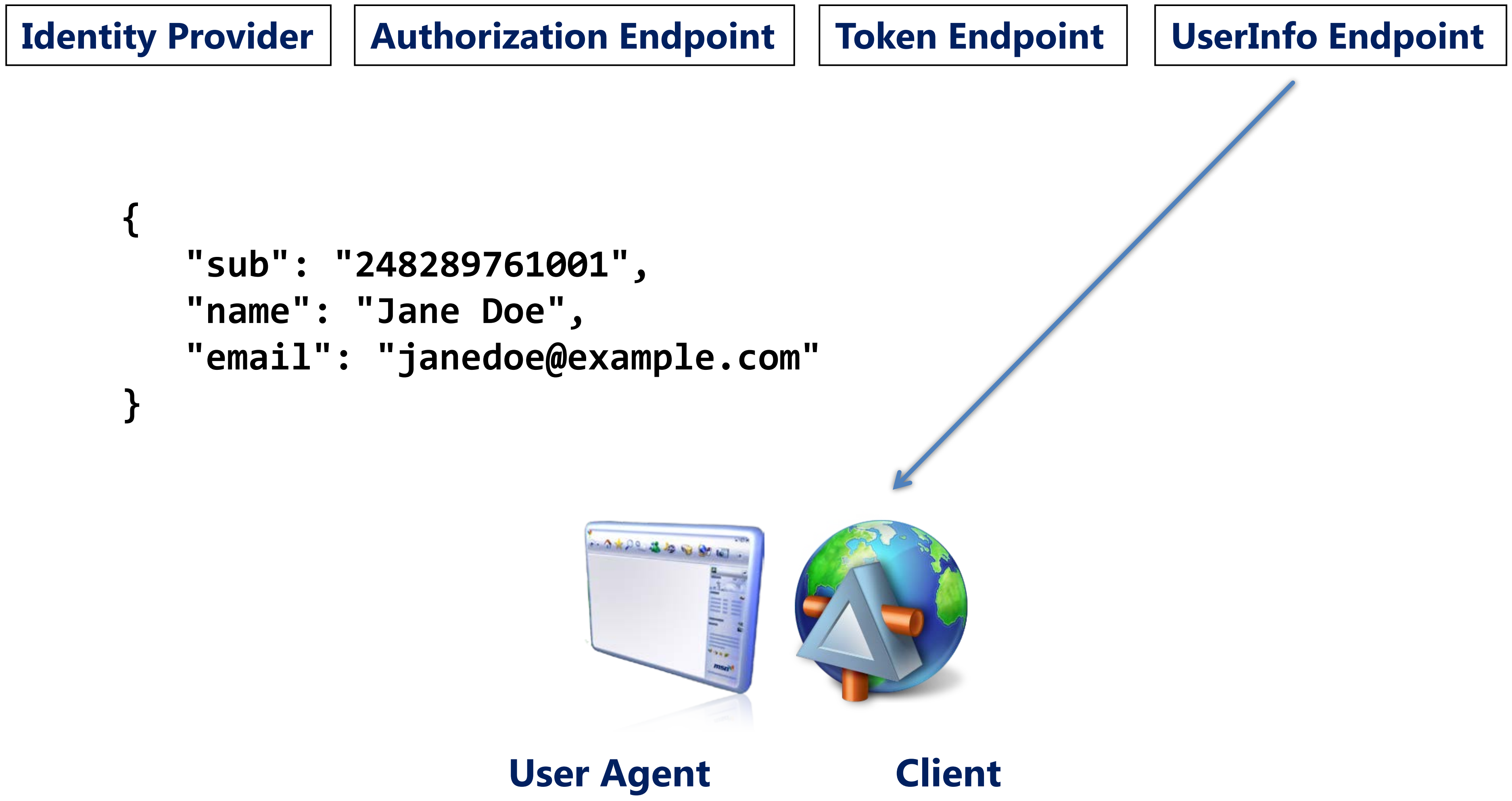
- **JWT that contains claims about the authentication event**
  - Issuer (iss)
  - Subject (sub)
  - Audience (aud)
  - Expiration (exp)
- **Client must validate the ID token at this point**

Step 3a: UserInfo Request



```
GET /userinfo

 Authorization:
   Bearer access_token
```

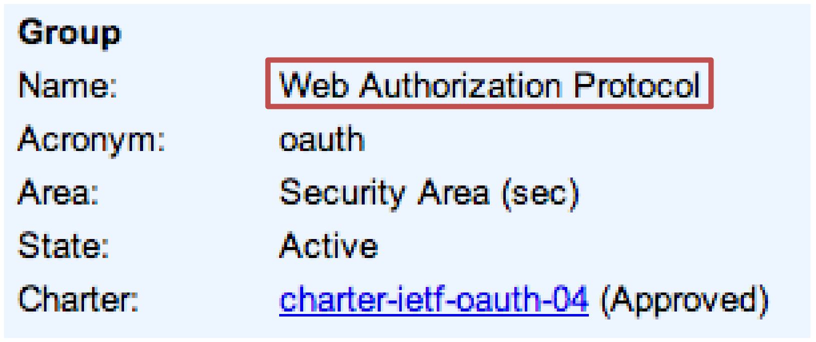
Step 3b: UserInfo Response



```
{
   "sub": "248289761001",
   "name": "Jane Doe",
   "email": "janedoe@example.com"
}
```

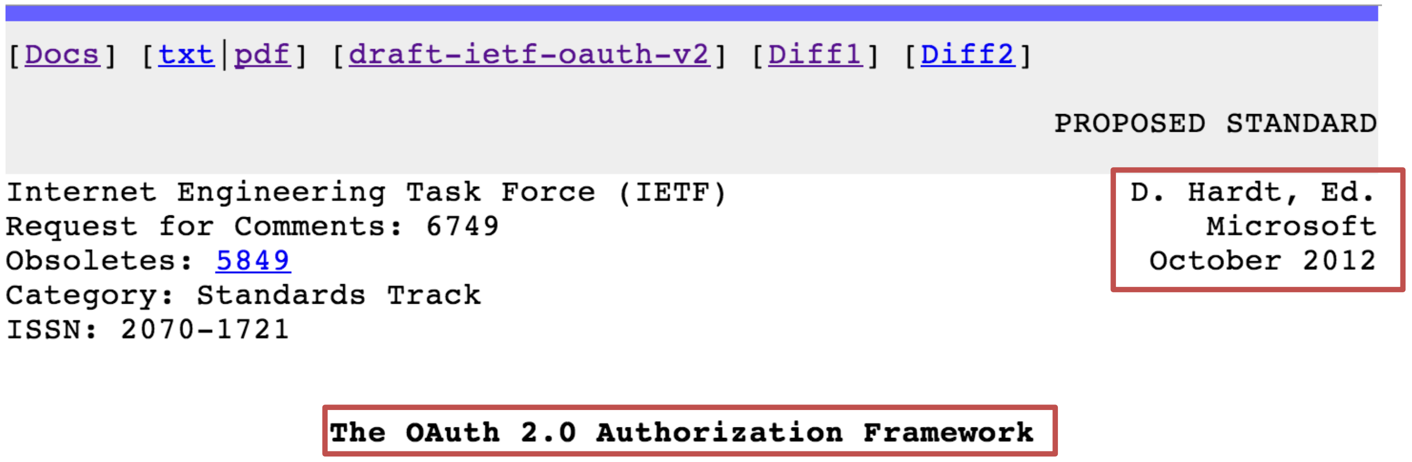
- **OpenID Connect standardizes how authentication with OAuth2 works**
  - standard scopes and claims
  - token type is JWT
  - ID token
  - UserInfo endpoint
- **Goal is to allow a client to use an arbitrary OpenID Connect provider without code modifications**
  - as opposed to how it works with homegrown OAuth2 authentication today

- **Not done yet, but "basic profile" pretty stable**
- **Additional specs are under development**



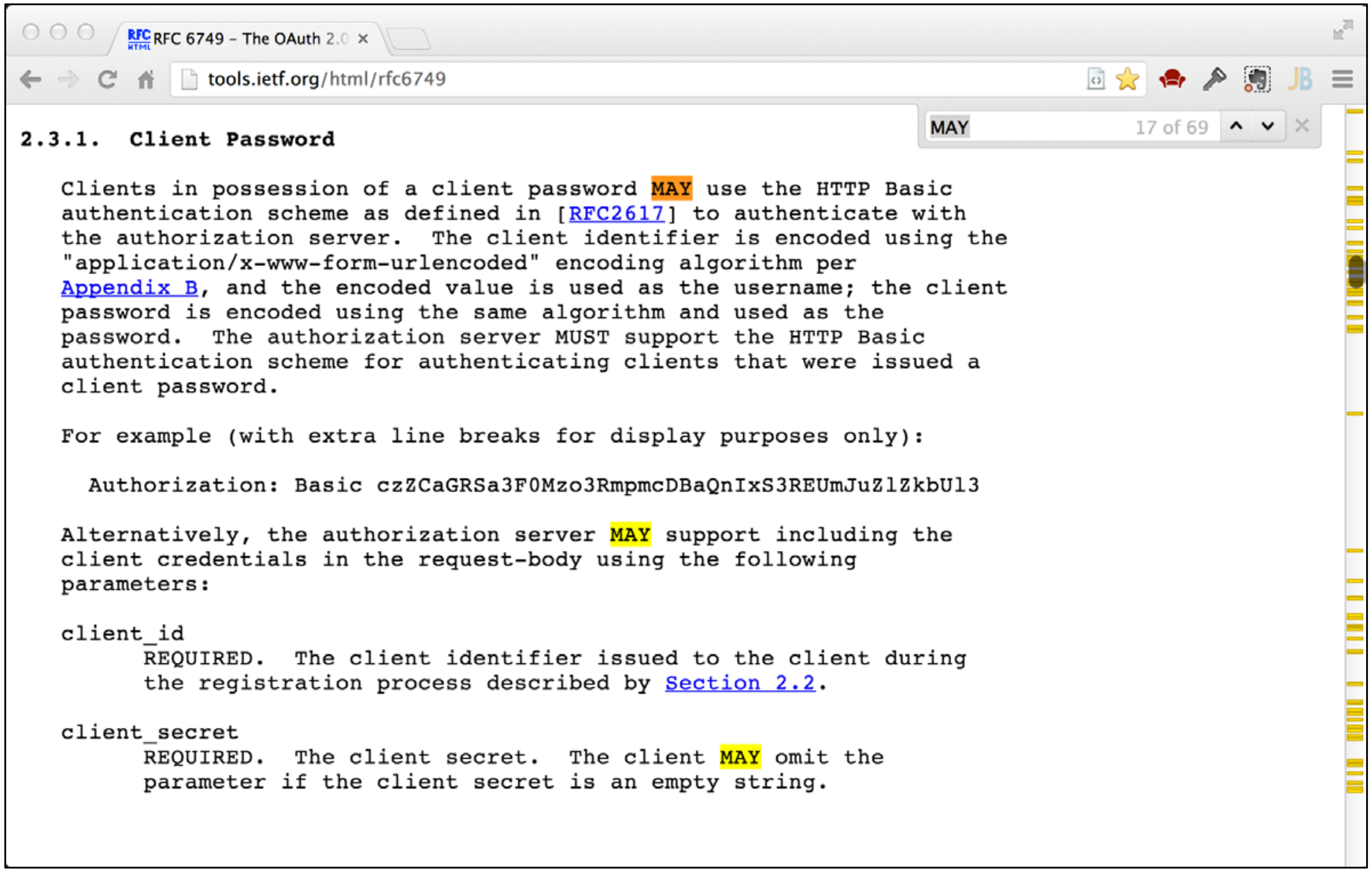
Group

Name:	Web Authorization Protocol
Acronym:	oauth
Area:	Security Area (sec)
State:	Active
Charter:	charter-ietf-oauth-04 (Approved)



```
[Docs] [txt|pdf] [draft-ietf-oauth-v2] [Diff1] [Diff2]

                                                      PROPOSED STANDARD

Internet Engineering Task Force (IETF)                    D. Hardt, Ed.
Request for Comments: 6749                                    Microsoft
Obsoletes: 5849                                            October 2012
Category: Standards Track
ISSN: 2070-1721


                 The OAuth 2.0 Authorization Framework
```

"A Framework to build Protocols"



```
2.3.1.  Client Password

   Clients in possession of a client password MAY use the HTTP Basic
   authentication scheme as defined in [RFC2617] to authenticate with
   the authorization server.  The client identifier is encoded using the
   "application/x-www-form-urlencoded" encoding algorithm per
   Appendix B, and the encoded value is used as the username; the client
   password is encoded using the same algorithm and used as the
   password.  The authorization server MUST support the HTTP Basic
   authentication scheme for authenticating clients that were issued a
   client password.

   For example (with extra line breaks for display purposes only):

     Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

   Alternatively, the authorization server MAY support including the
   client credentials in the request-body using the following
   parameters:

   client_id
         REQUIRED.  The client identifier issued to the client during
         the registration process described by Section 2.2.

   client_secret
         REQUIRED.  The client secret.  The client MAY omit the
         parameter if the client secret is an empty string.
```

JSON Web Token (JWT)

JSON Web Encryption (JWE)
JSON Web Signatures (JWS)
JSON Web Algorithms (JWA)

Assertion Framework for OAuth2
JWT Bearer Token Profiles
SAML 2.0 Bearer Token Profiles
Token Revocation
MAC Tokens

The OAuth2 Authorization Framework
(RFC 6749)

OAuth2 Bearer Token Usage
(RFC 6750)

Threat Model and Security Considerations
(RFC 6819)

Core (proposed standards)

Informational

OAuth2 Resource Set Registration
Dynamic Client Registration
User-Managed Access
Chaining and Redelegation
Metadata & Introspection

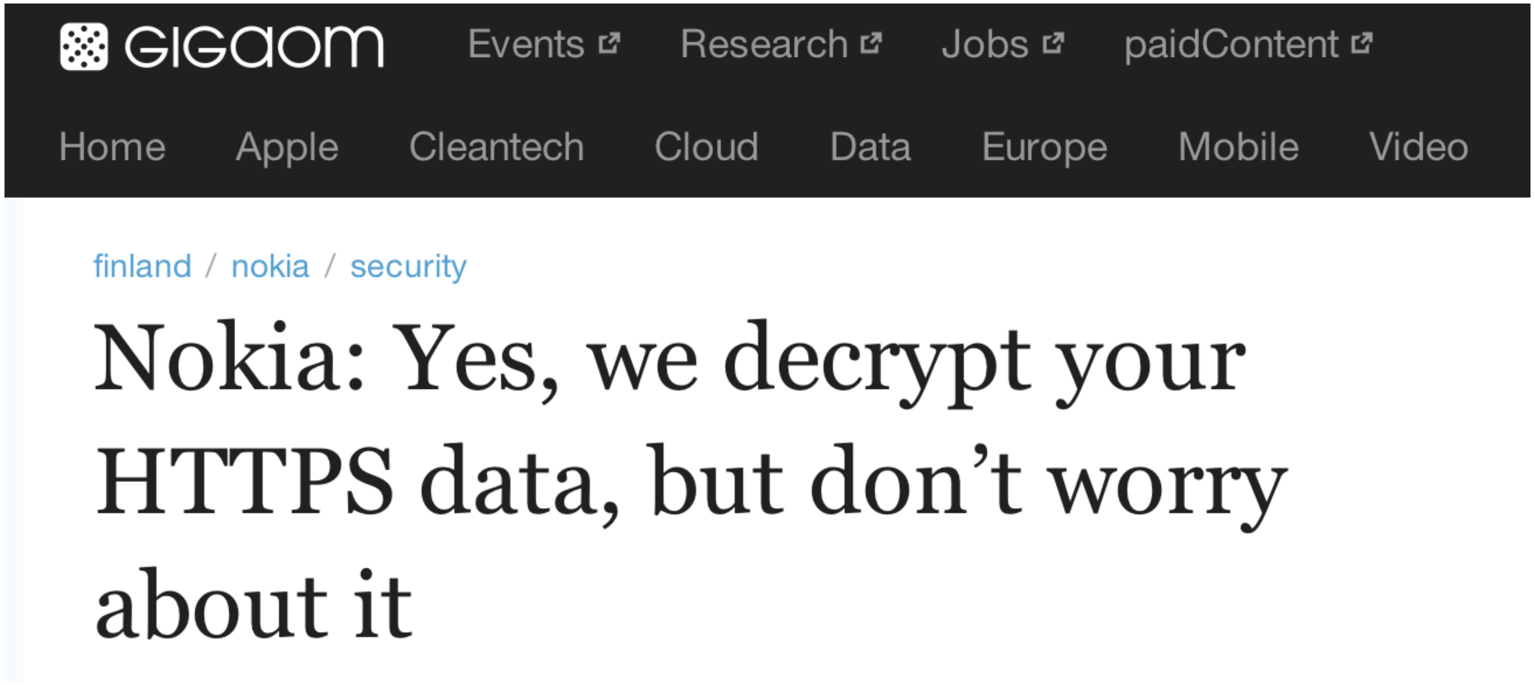
http://openid.net/specs/openid-connect
basic-1_0-23.html
implicit-1_0-06.html
messages-1_0-15.html
standard-1_0-16.html
discovery-1_0-12.html
registration-1_0-14.html
session-1_0-11.html

http://datatracker.ietf.org/wg/oauth/

Bearer Token

A security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can. Using a bearer token does not require a bearer to prove possession of cryptographic key material (proof-of-possession).

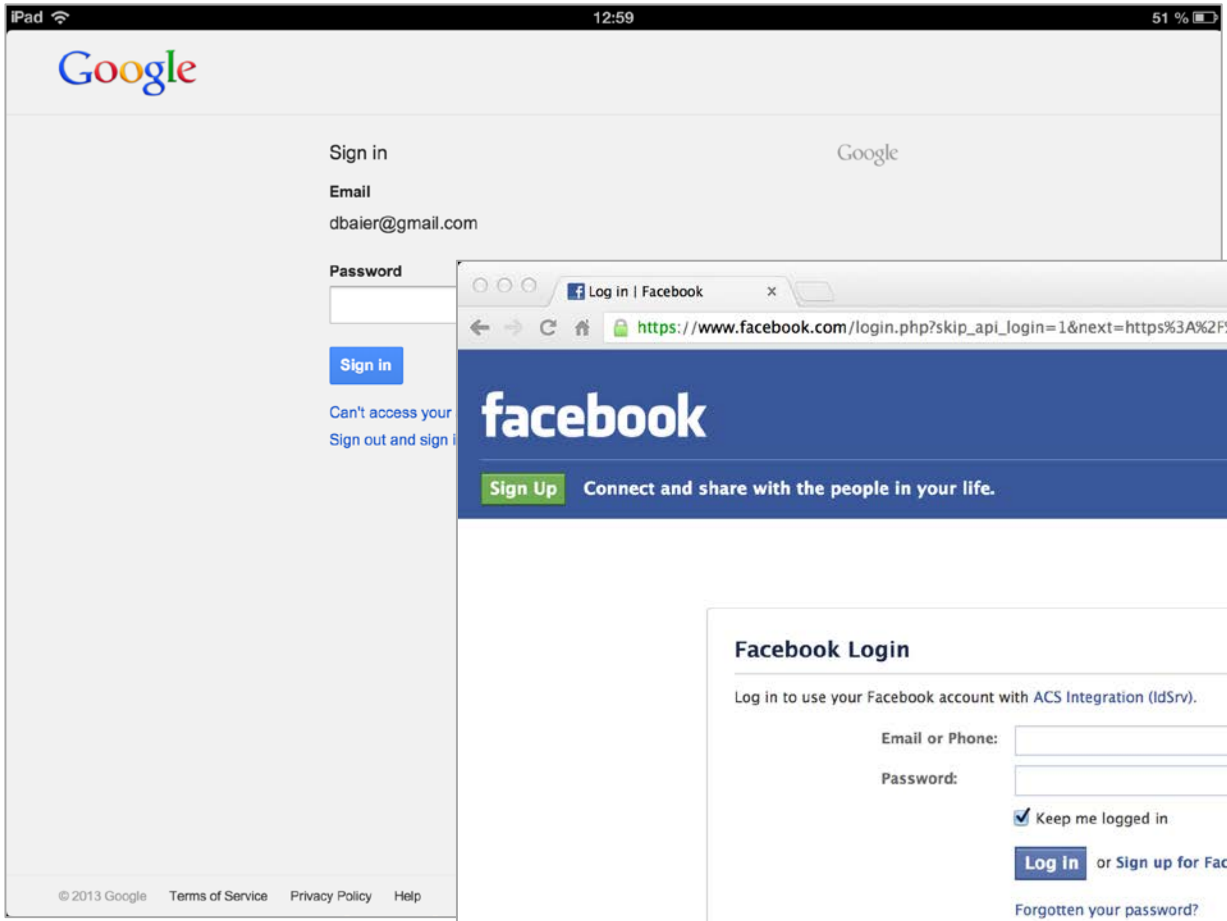
Infrastructure & SSL



Nokia: Yes, we decrypt your HTTPS data, but don't worry about it

http://gigaom.com/2013/01/10/nokia-yes-we-decrypt-your-https-data-but-dont-worry-about-it/

Security Theater



iPad 12:59 51 %

Google

Sign in

Google

Email

dbaier@gmail.com

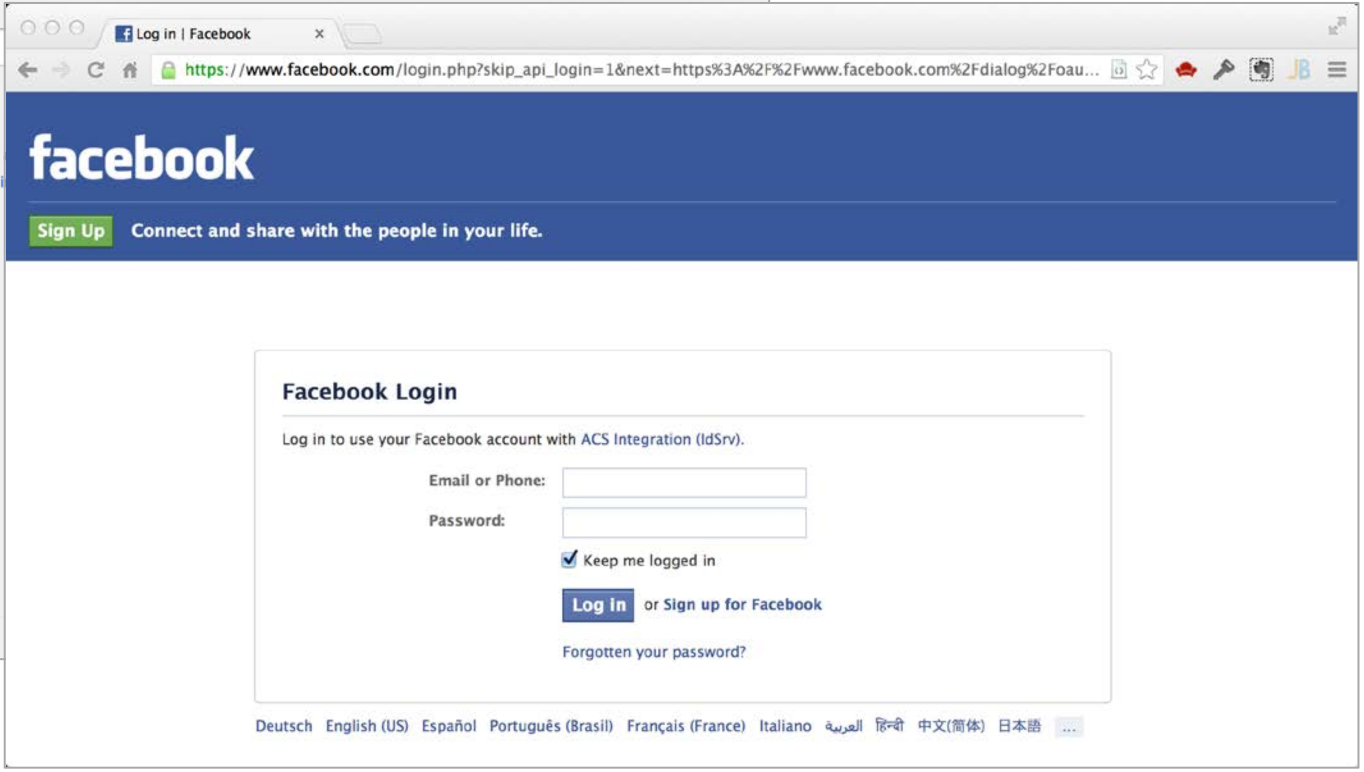
Password

Sign in

Can't access your

Sign out and sign i

© 2013 Google Terms of Service Privacy Policy Help



Log in | Facebook

https://www.facebook.com/login.php?skip_api_login=1&next=https%3A%2F%2Fwww.facebook.com%2Fdialog%2Foau...

facebook

Sign Up Connect and share with the people in your life.

Facebook Login

Log in to use your Facebook account with ACS Integration (IdSrv).

Email or Phone:

Password:

✓ Keep me logged in

Log in or Sign up for Facebook

Forgotten your password?

Deutsch English (US) Español Português (Brasil) Français (France) Italiano العربية हिन्दी 中文(简体) 日本語 ...

OAUTH2
ATTACK SURFACE EVERYWHERE

Attack Surface

```
GET /authorize?
  client_id=nativeapp&
  redirect_uri=http://localhost/cb&
  scope=resource&
  response_type=token&
  state=123
```

http://leastprivilege.com/2013/03/15/common-oauth2-vulnerabilities-and-mitigation-techniques/
http://leastprivilege.com/2013/03/15/oauth2-security/
http://homakov.blogspot.de/2012/08/saferweb-oauth2a-or-lets-just-fix-it.html

Some Facebook Hacks

- http://www.darkreading.com/blog/240148995/
  the-road-to-hell-is-authenticated-by-facebook.html
- http://homakov.blogspot.no/2013/02/hacking-facebook-with-oauth2-and-chrome.html
- www.nirgoldshlager.com/2013/03/
  how-i-hacked-any-facebook-accountagain.html

- **The OAuth2 "approach" is useful for many typical applications scenarios**

- **Spec needs some refinement**
  - "basic profile"
  - MAC tokens
- **Current implementations are lacking**
  - even by the big guys
  - let alone the myriad of DIY implementations

- **Very good & balanced view**
  - https://www.tbray.org/ongoing/When/201x/2013/01/23/OAuth

JWT

- **JWT debugger**
  - http://openidtest.uninett.no/jwt
- **Microsoft JWT**
  - http://nuget.org/packages/Microsoft.IdentityModel.Tokens.JWT/
- **Specs**
  - http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-08
  - http://tools.ietf.org/html/draft-ietf-jose-json-web-signature-11
  - http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-11
- **Java JWT**
  - https://bitbucket.org/nimbusds/nimbus-jose-jwt/wiki/Home

OAuth2

- **Specs**
  - http://tools.ietf.org/html/rfc6749
  - http://tools.ietf.org/html/rfc6750
- **Threat Model**
  - http://tools.ietf.org/html/rfc6819

- **Thinktecture.IdentityModel**
  - https://github.com/thinktecture/Thinktecture.IdentityModel.45
- **Thinktecture.IdentityServer**
  - https://github.com/thinktecture/Thinktecture.IdentityServer.v2
- **DotNetOpenAuth**
  - http://dotnetopenauth.net/

OpenID Connect

- **Specs**
  - http://openid.net/connect/
- **Google "OpenID Connect" signin**
  - https://developers.google.com/accounts/docs/OAuth2Login
  - https://developers.google.com/accounts/cookbook/technologies/OpenID-Connect
- **Reference implementation**
  - https://github.com/mitreid-connect/OpenID-Connect-Java-Spring-Server