

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*



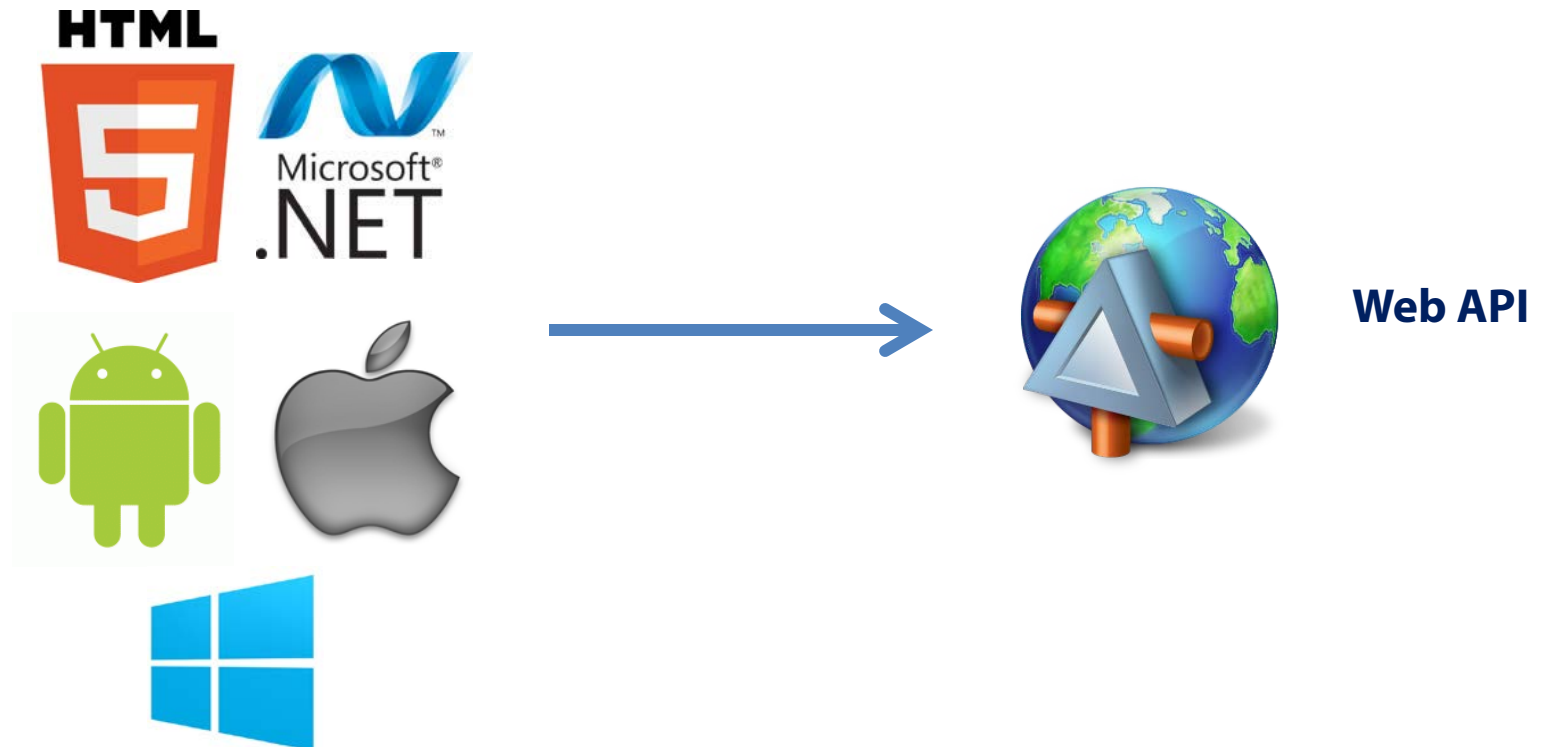
HTTP
JSON

Scenario 1: Mobile Enterprise Apps



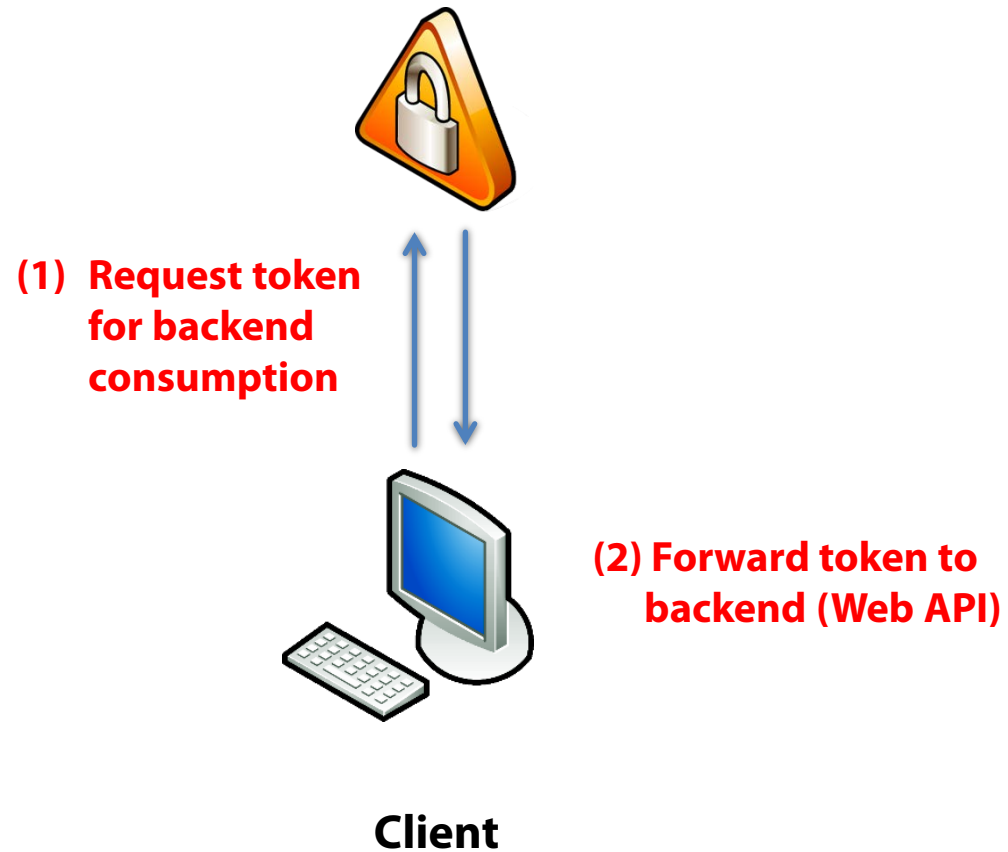
Scenario 2: Business to Customer

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



OAuth2

Authorization Server

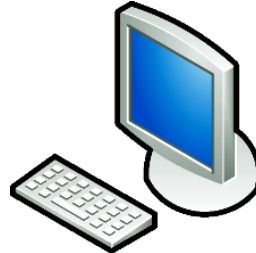


OpenID Connect

Authentication Server



**(1) Request token
for client
consumption**



**(2) Parse and validate
token**

Client

Summary

- **"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 critical
- **New "common denominator" technologies**
 - presentation (e.g. HTML5)
 - authentication & authorization

JSON Web Token (JWT)

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"]  
}
```

eyJhbGciOiJIub251In0.eyJpc3MiOiJqb2UiLA0KICJleHAiOiJlZzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt

Header

Claims

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);
```

Summary

- **JWT is easy to**
 - create
 - transmit
 - parse
 - validate
- **Quickly becomes the standard for web based tokens**
- **Mandatory in OpenID Connect**

What is OAuth2 ?



OAUTH

[About](#) [Advisories](#) [Documentation](#) [Code](#) [Community](#)

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](#).

History

- OAuth started circa 2007
- 2008 - IETF normalization started in 2008
- 2010 - RFC 5849 defines OAuth 1.0
- 2010 - WRAP (Web Resource Authorization Profiles) proposed by Microsoft, Yahoo! And Google
- 2010 - OAuth 2.0 work begins in IETF
- Working deployments of various drafts & versions at Google, Microsoft, Facebook, Github, Twitter, Flickr, Dropbox...
- Mid 2012 – Lead author and editor resigned & withdraws his name from all specs
- October 2012 – RFC 6749, RFC 6750

High level overview

Resource Server



Client



Resource Owner



A young man with short brown hair, wearing a red and white striped cardigan over a white t-shirt, is speaking and gesturing with his right hand. He is in a cluttered room with various items in the background, including a guitar, a computer monitor, and a red rotary phone.

FAKING OUT PARENTS

- 1. Fake a Stomach Cramp**
- 2. Moan and Wail**
- 3. Lick Palms**

<http://hueniverse.com/2007/09/explaining-oauth/>
<http://amzn.com/1449311601>





No problem. Trust me.



High level overview

Resource Server



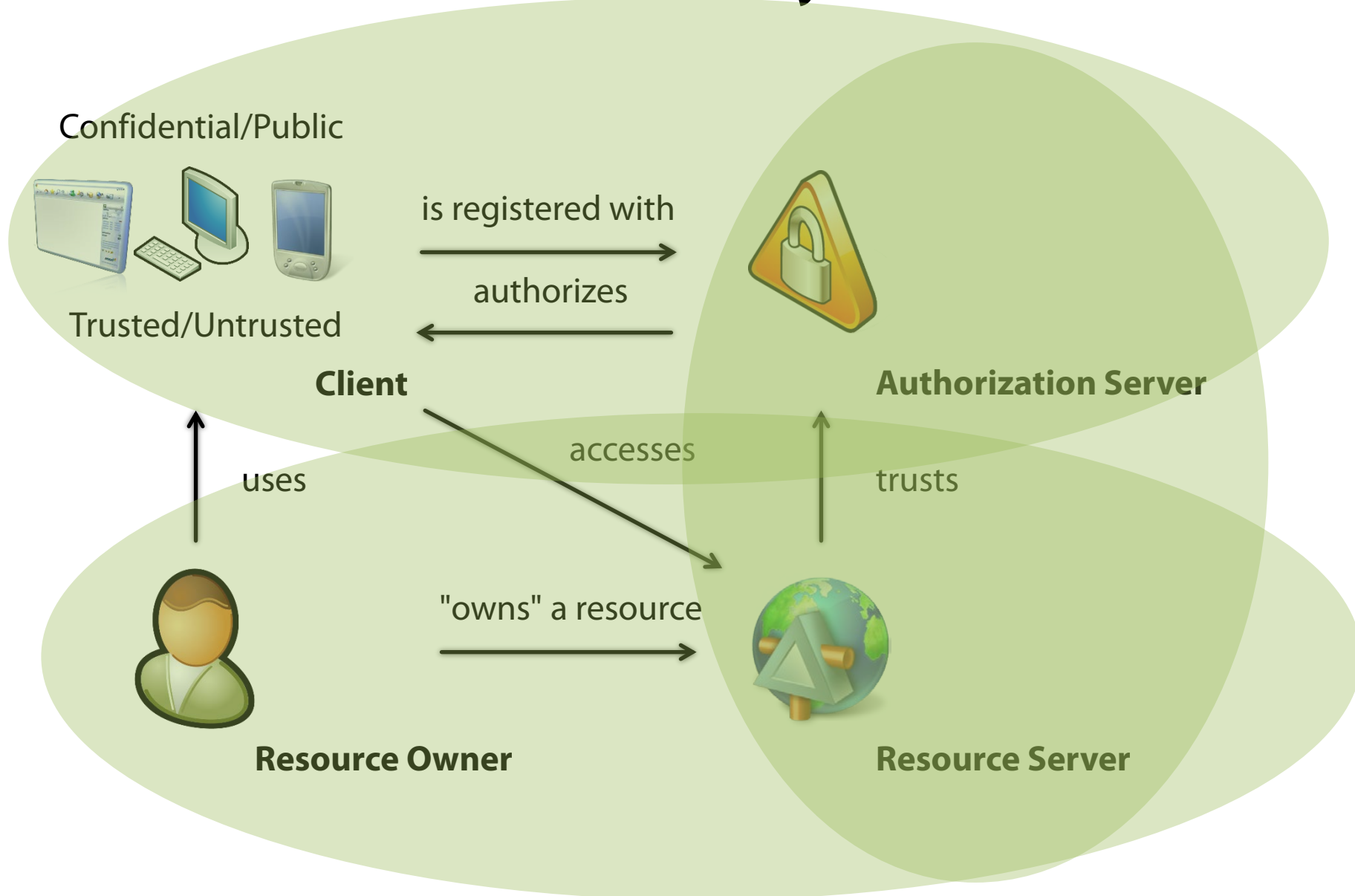
Client



Resource Owner



OAuth2: The Players



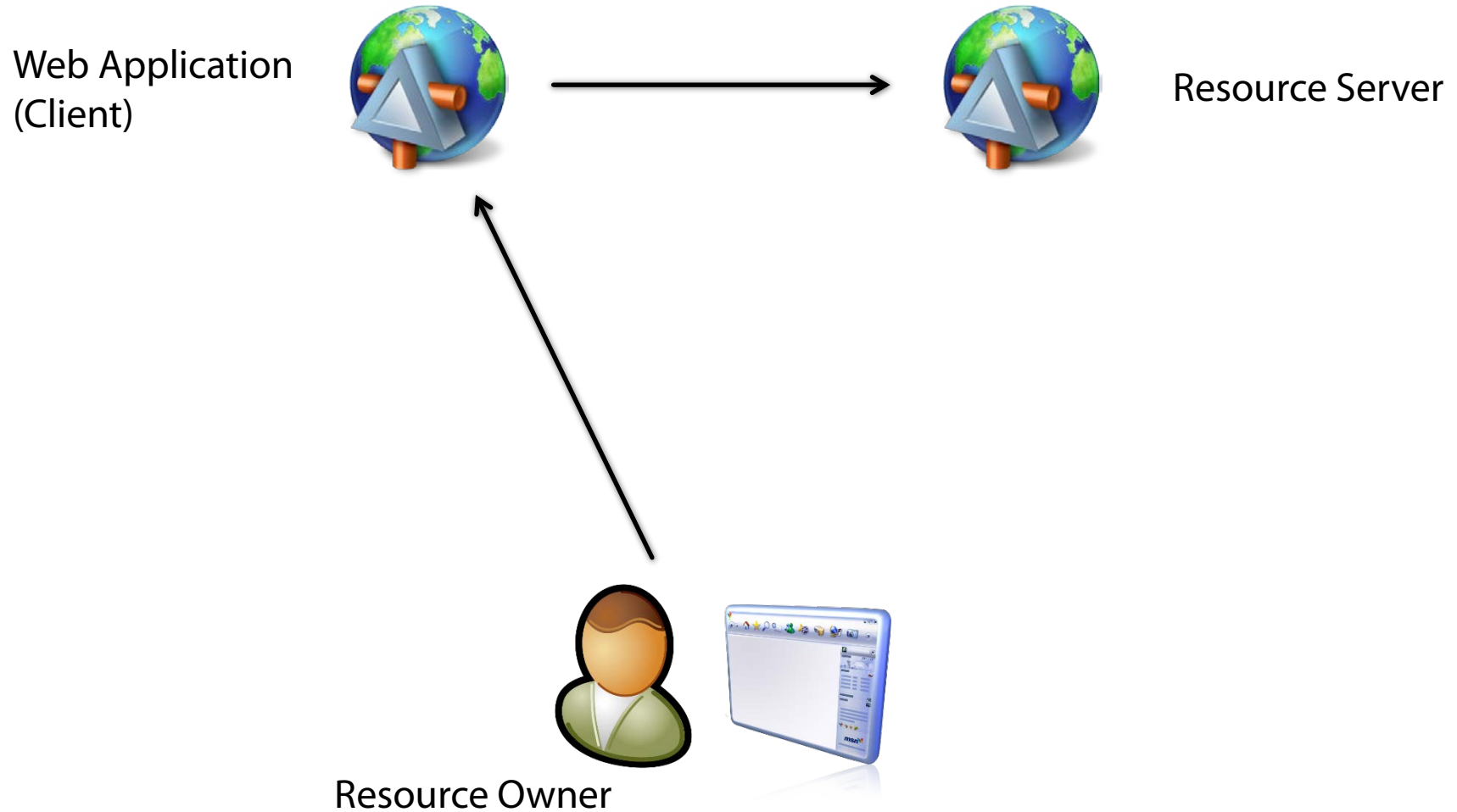
OAuth2 Flows - with User Interaction

- **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

Authorization Code Flow (Web Application Clients)



Step 1a: Authorization Request

Web Application
(Client)



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



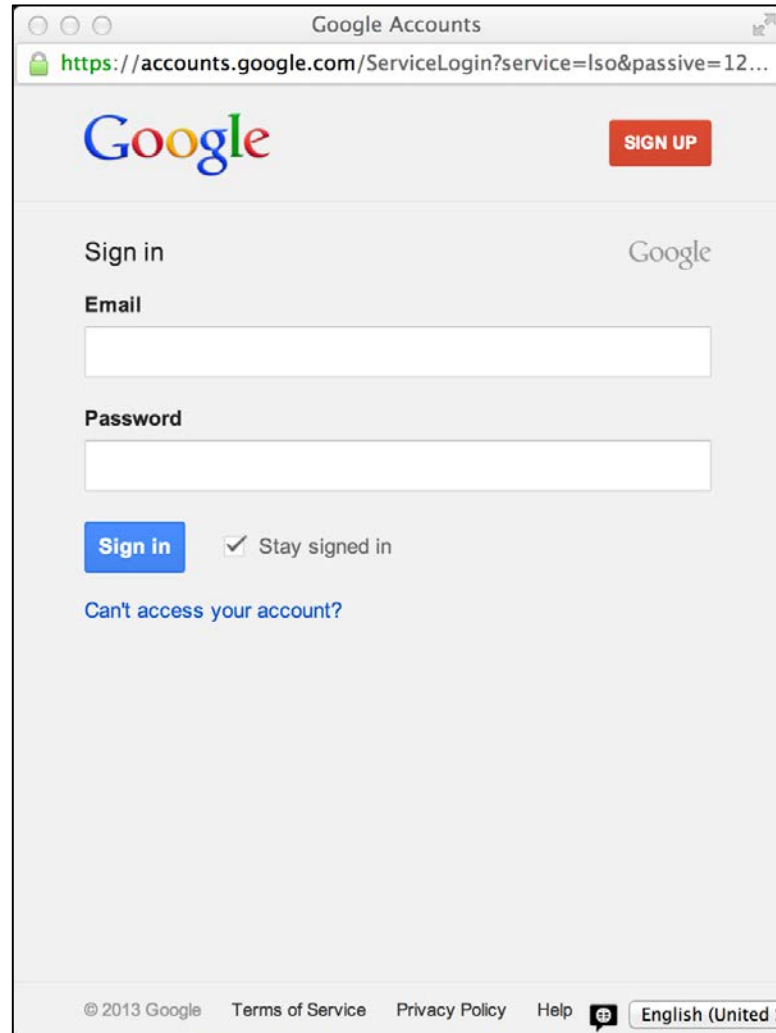
Authorization Server



Resource Owner



Step 1b: Authentication



A screenshot of a web browser window titled "Google Accounts". The address bar shows the URL <https://accounts.google.com/ServiceLogin?service=iso&passive=12...>. The page features the Google logo at the top left and a red "SIGN UP" button at the top right. Below the logo, the text "Sign in" is displayed, followed by a "Google" logo. The form includes an "Email" label and a text input field, a "Password" label and a text input field, a blue "Sign in" button, and a checkbox labeled "Stay signed in" which is checked. A link "Can't access your account?" is located below the sign-in button. The footer contains copyright information "© 2013 Google", links for "Terms of Service", "Privacy Policy", and "Help", along with a language selector set to "English (United S)".

Google Accounts

<https://accounts.google.com/ServiceLogin?service=iso&passive=12...>

Google

SIGN UP

Sign in

Google

Email

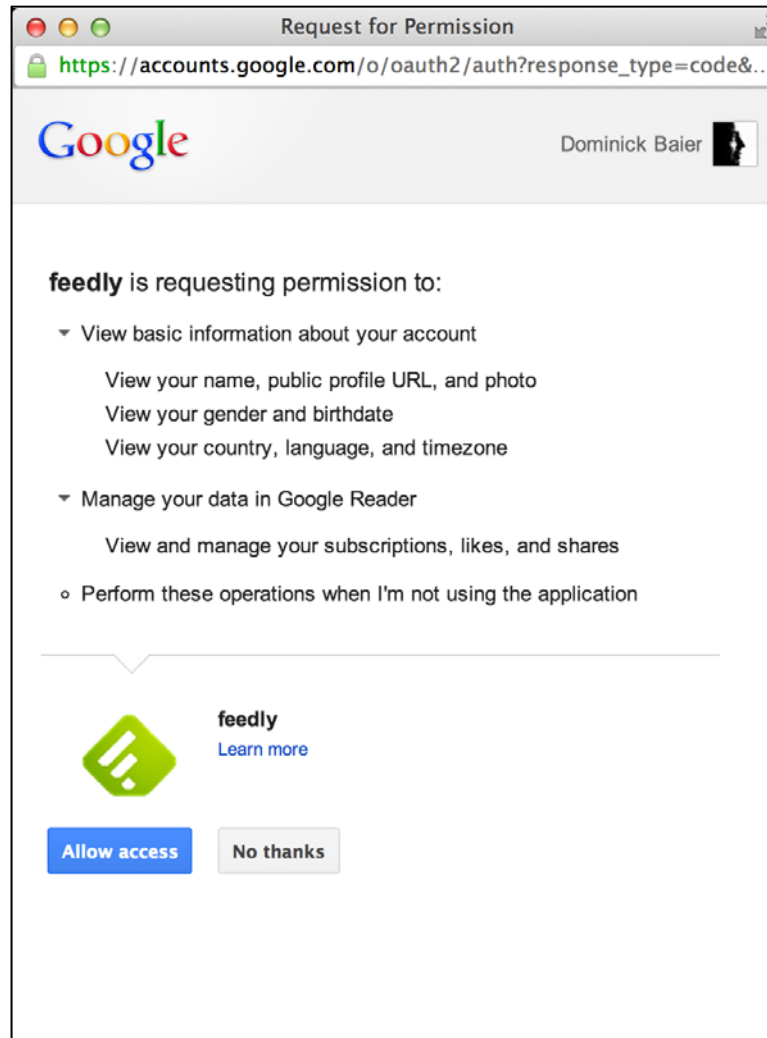
Password

Sign in ☒ Stay signed in

[Can't access your account?](#)

© 2013 Google Terms of Service Privacy Policy Help English (United S)

Step 1c: Consent



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.

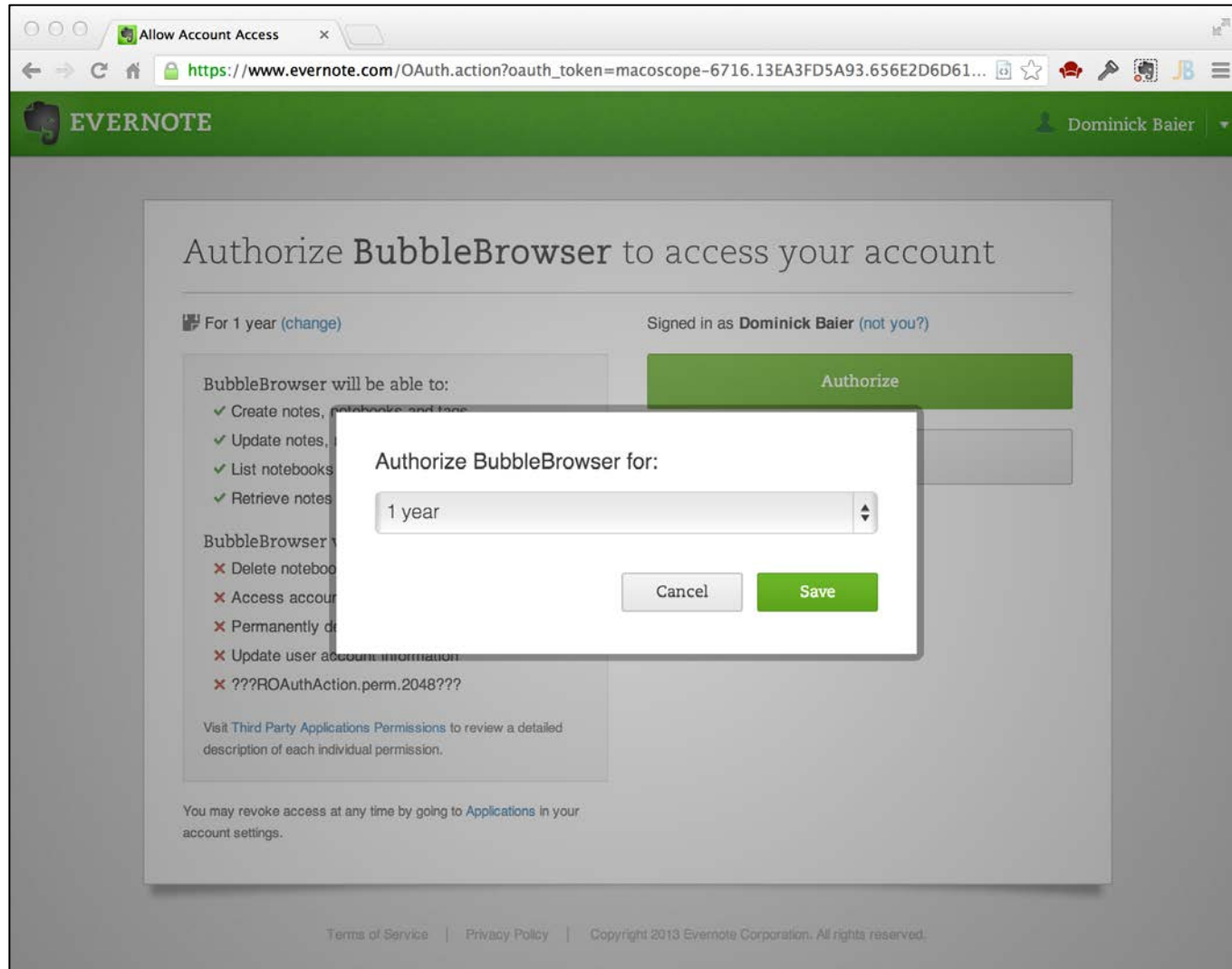


Twitter for Windows

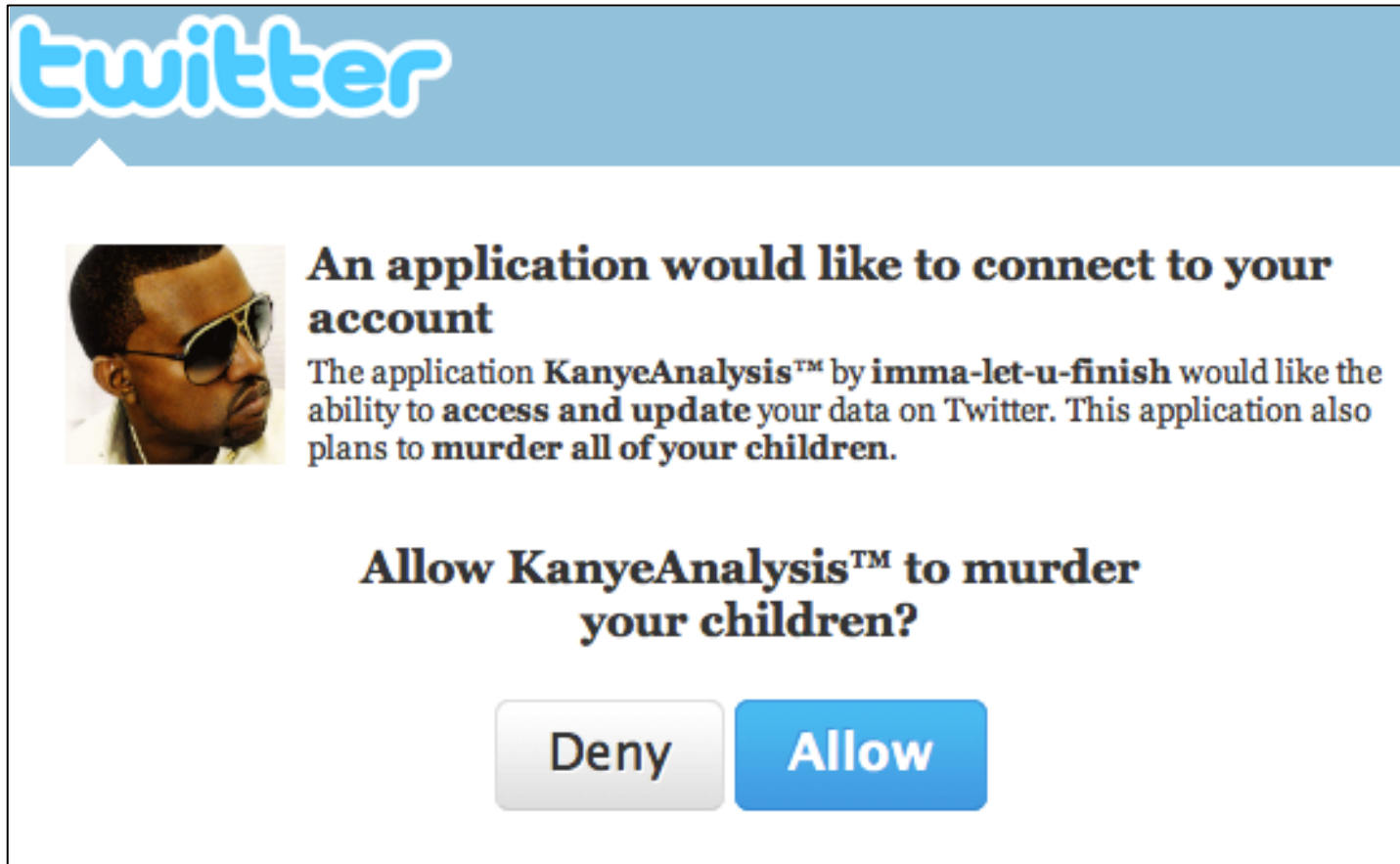
www.twitter.com

Official Twitter for Windows application.

Evernote Consent



The Consent Screen is important!



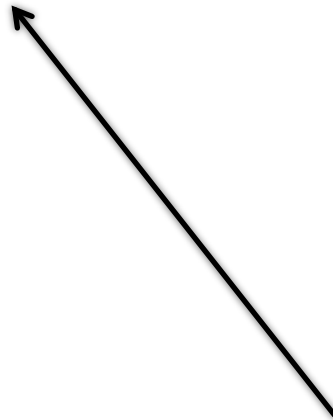
Step 1d: Authorization Response

Web Application
(Client)



Authorization Server

GET /cb?
code=xyz&
state=123



Resource Owner



Step 2a: Token Request

Web Application
(Client)



Authorization Server

POST /token

Authorization: Basic (client_id:secret)

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



Resource Owner

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

Web Application
(Client)



Resource Server

GET /resource

Authorization: Bearer access_token



Resource Owner

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)

Web Application
(Client)



Authorization Server

POST /token

Authorization: Basic (client_id:secret)

**grant_type=refresh_token&
refresh_token=xyz**



Resource Owner

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

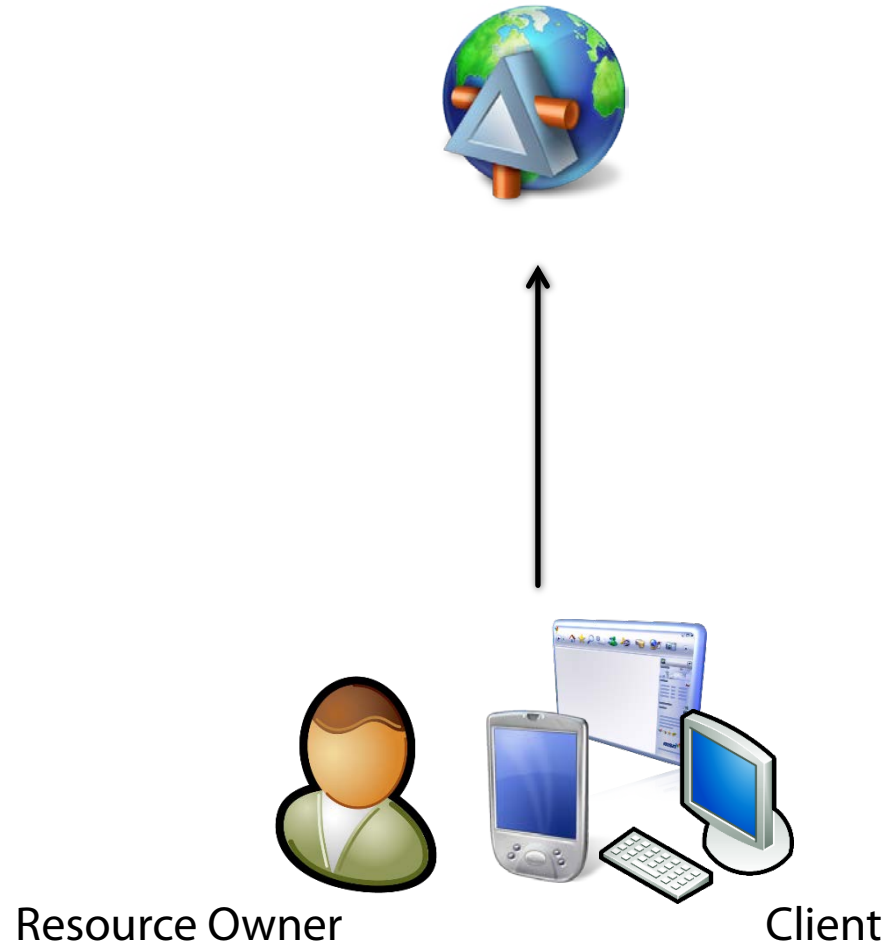
You last used Dominick's App on 2/27/2013.

[Edit](#)

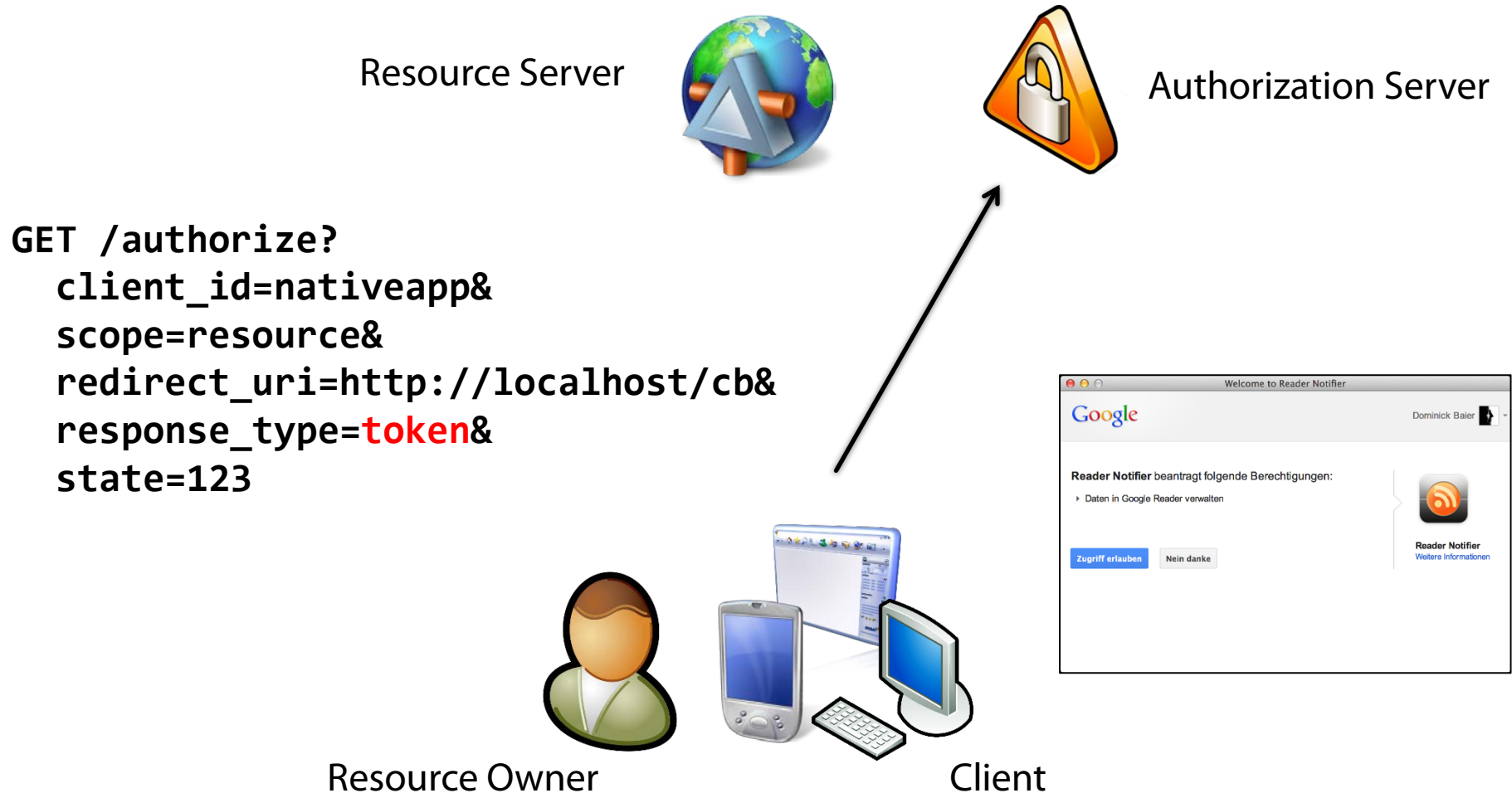
Summary – Code Flow

- **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**

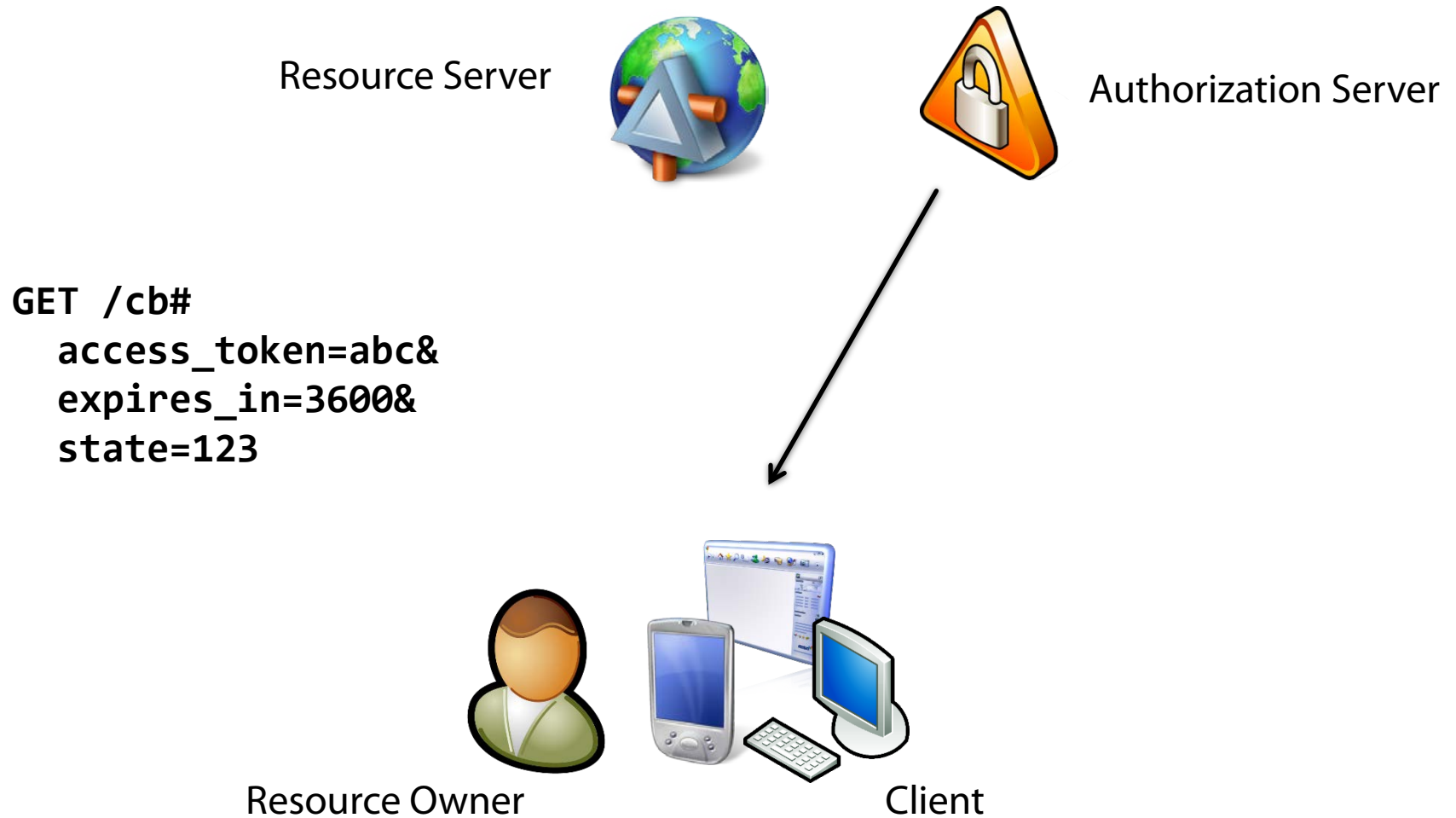
Implicit Flow (Native / Local Clients)



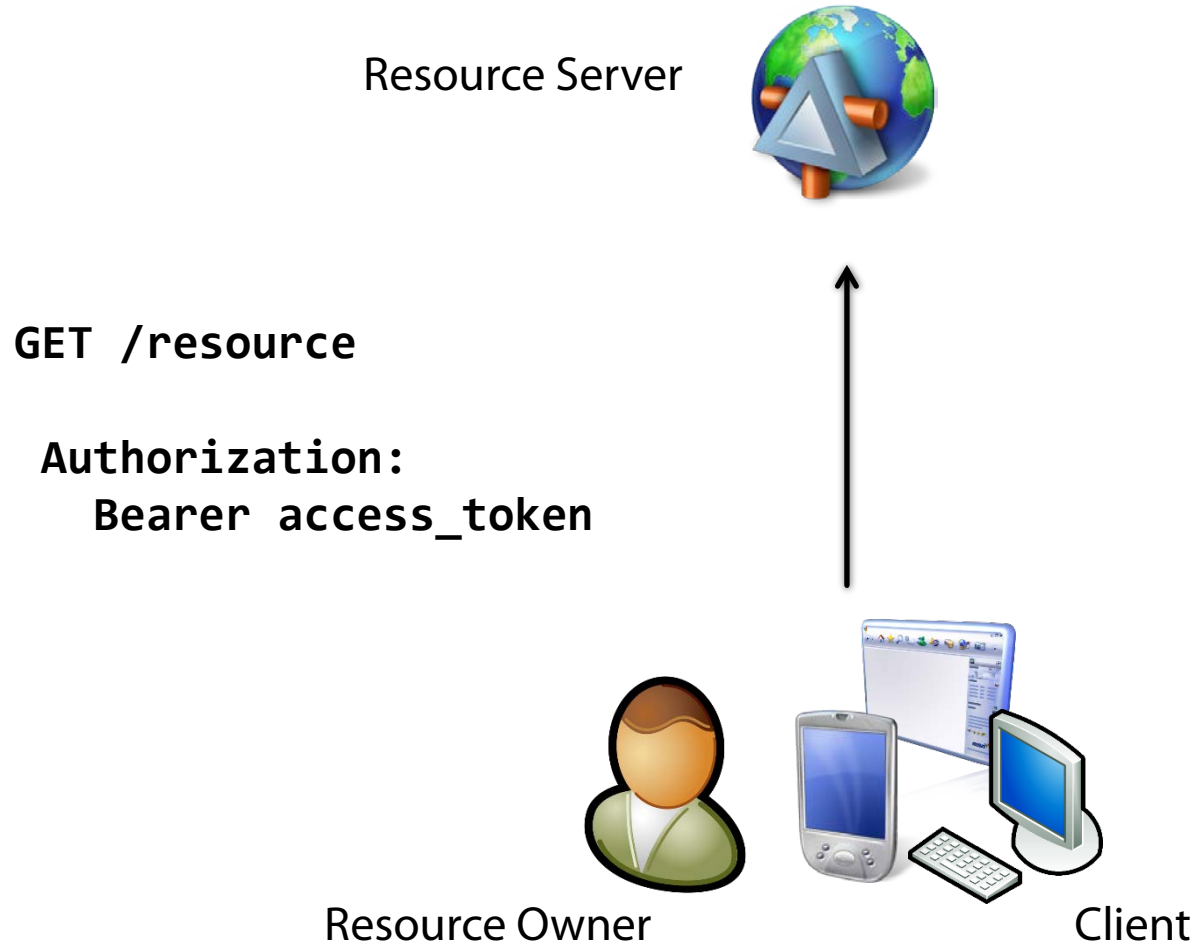
Step 1a: Authorization Request



Step 1b: Token Response



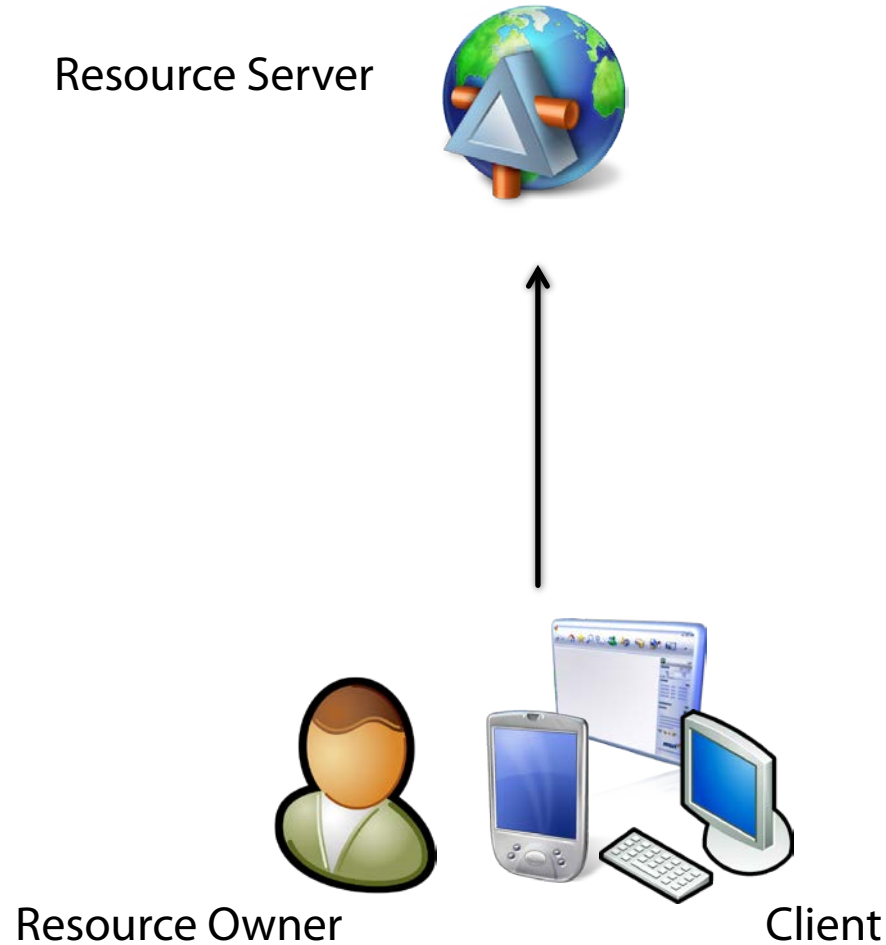
Step 2: Resource Access



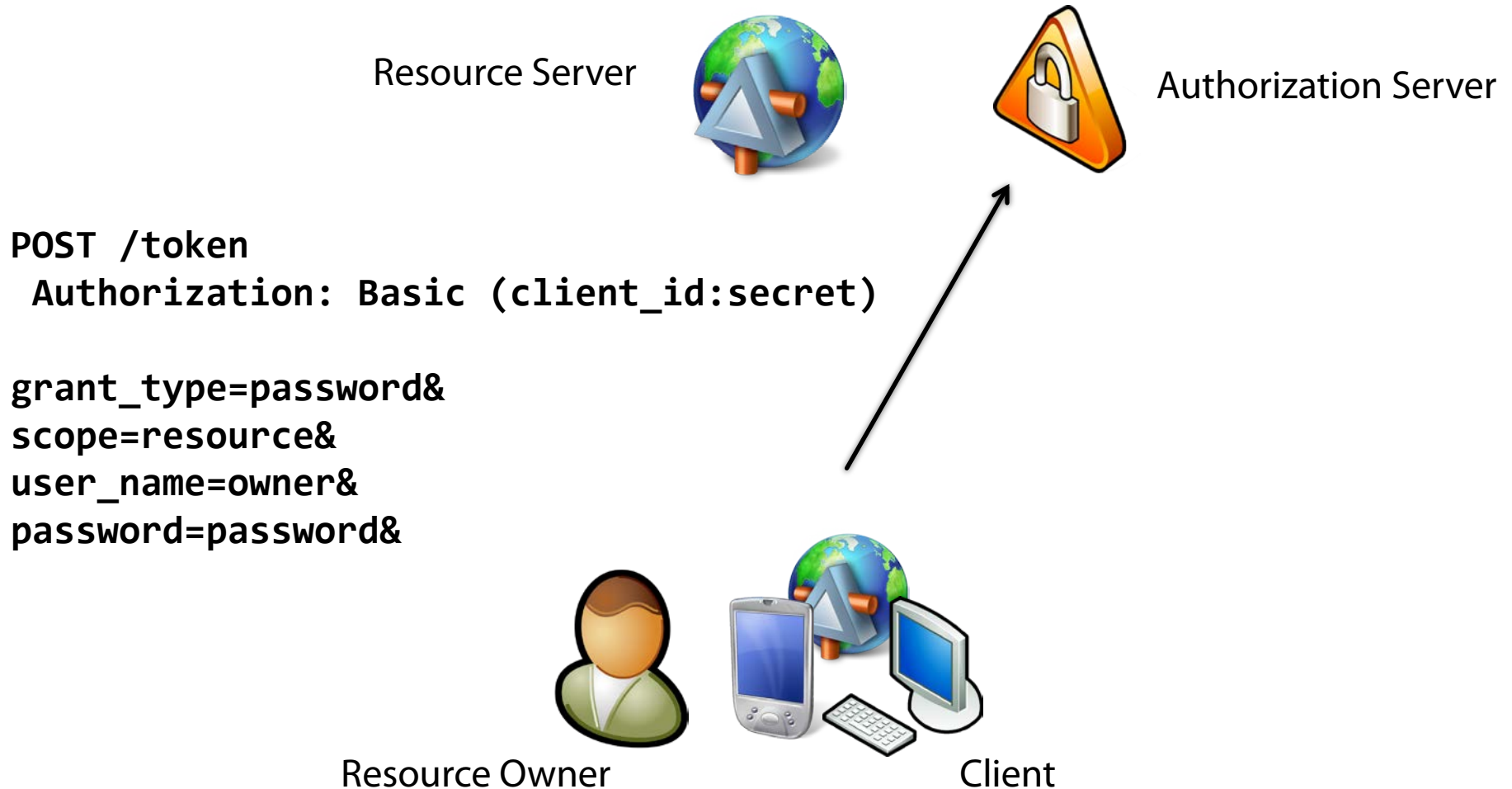
Summary – Implicit Flow

- **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



Step 1b: Token Response

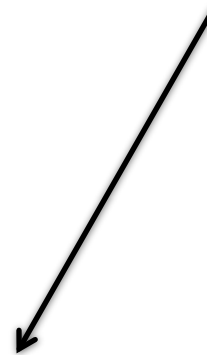
Resource Server



Authorization Server



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



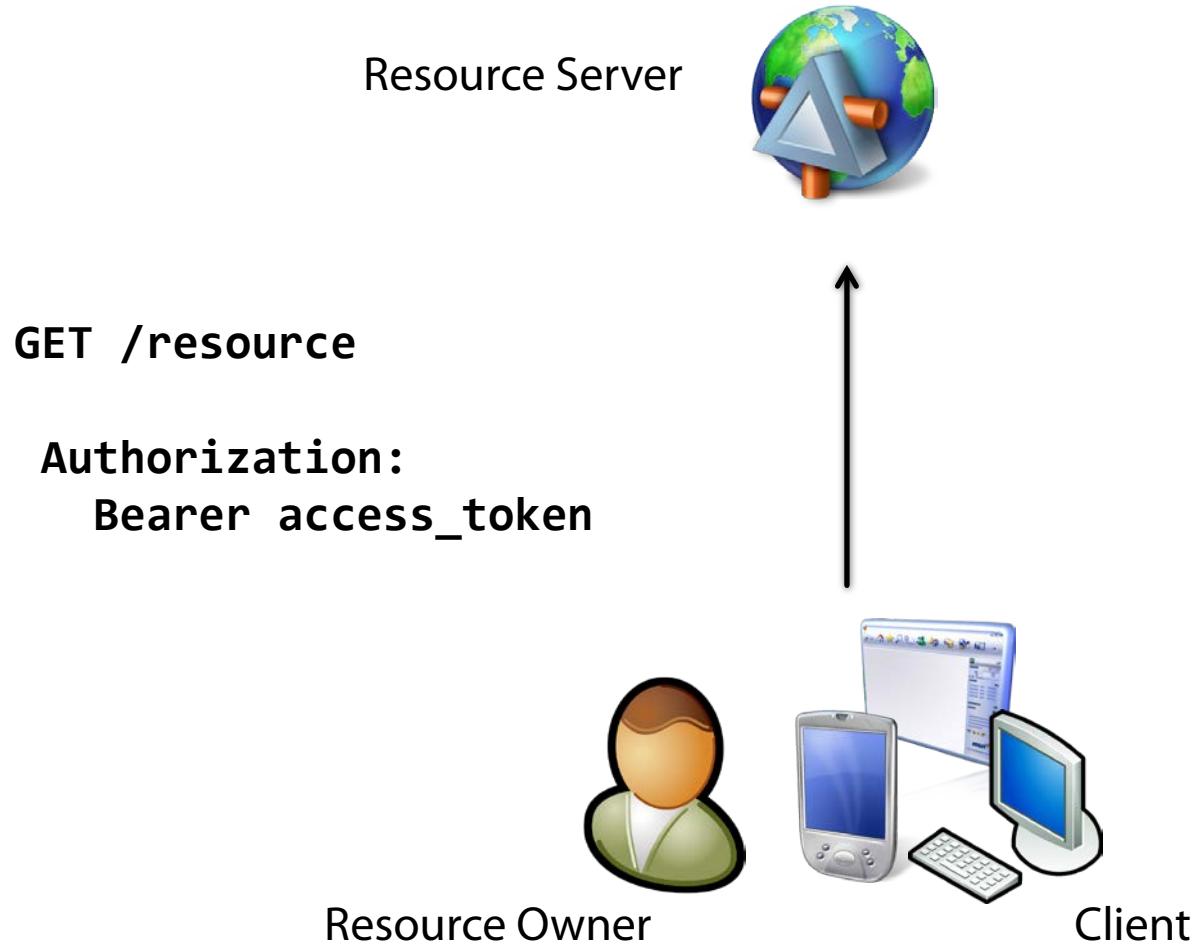
Resource Owner



Client



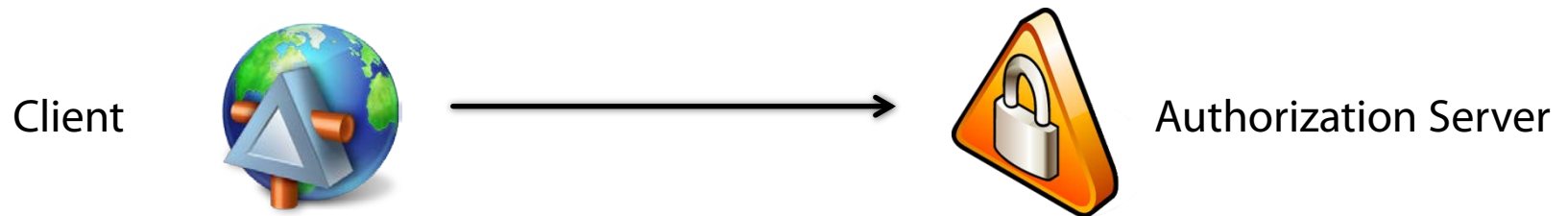
Step 2: Resource Access



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**

Outline

- **OAuth2 and authentication**
- **OpenID Connect**
- **Flows**

OAuth2 & Authentication

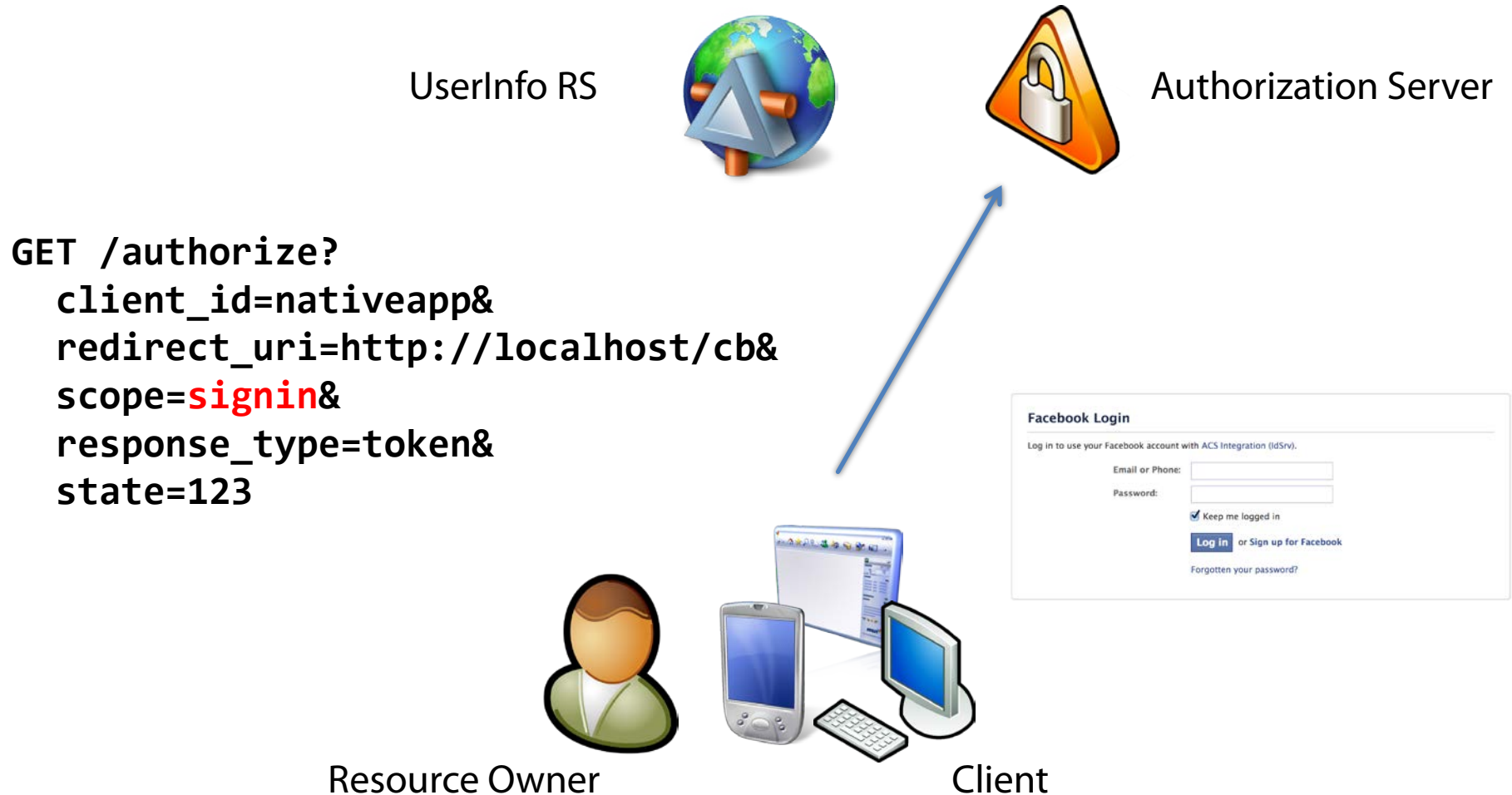
- **OAuth2 is for (delegated) authorization**
 - authentication is a pre-requisite for that
 - access token is for some back-end service
- **Sometimes you "just" need authentication**
 - (at least to begin with)
 - identify user in an application
 - control access to application features
- **OAuth2 is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



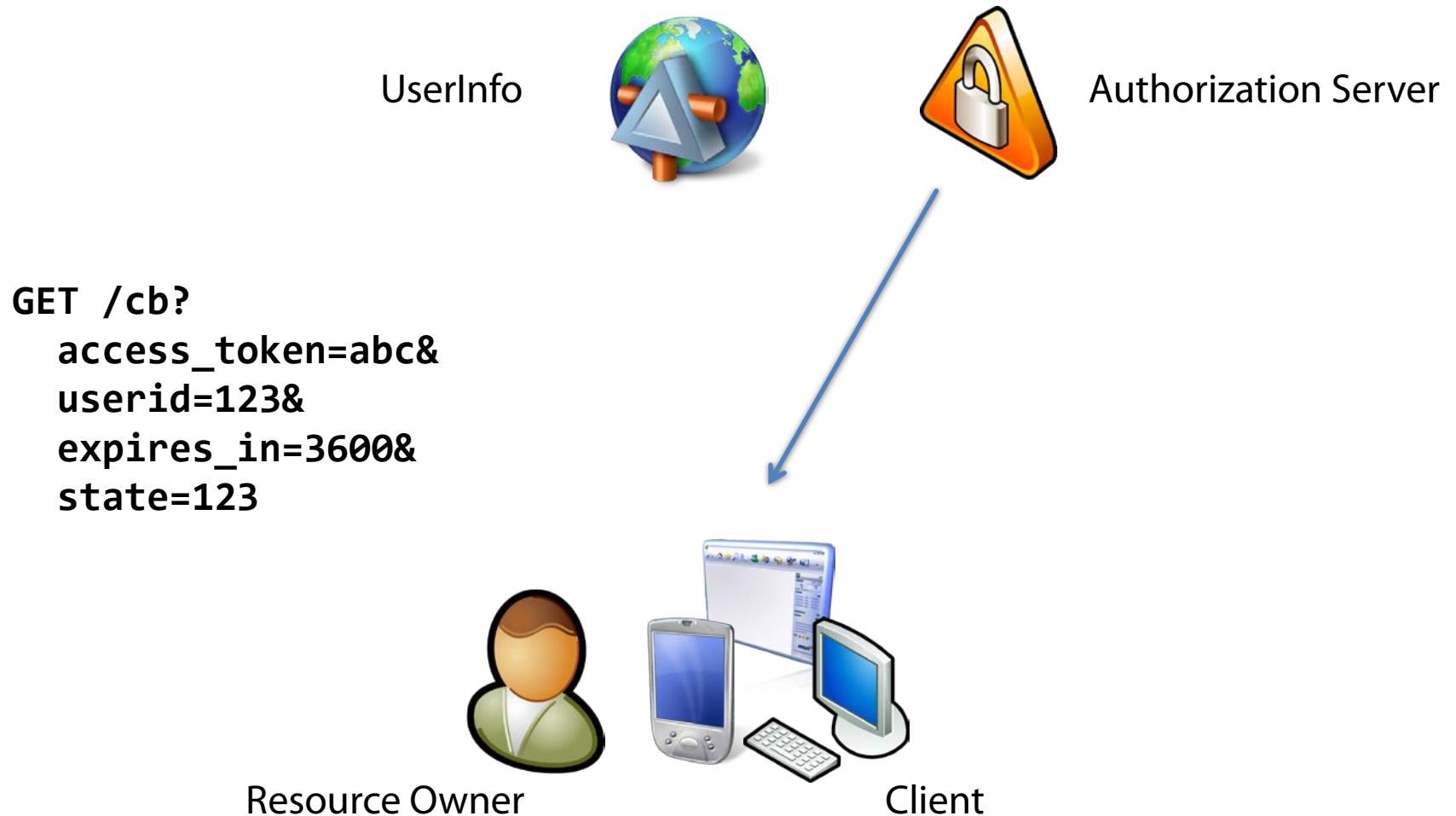
<http://www.thread-safe.com/2012/01/problem-with-oauth-for-authentication.html>

<http://www.cloudidentity.com/blog/2013/01/02/oauth-2-0-and-sign-in-4/>

OAuth2 for Authentication: Request



OAuth2 for Authentication: Response



OAuth2 for Authentication: Accessing User Data



The Problem



Facebook Login

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

Email or Phone:

Password:

☒ Keep me logged in

[Log In](#) or [Sign up for Facebook](#)

[Forgotten your password?](#)



**userid,
access token**



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



Impersonated!



**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.



Developers before OAuth.io

stubborn programmer

- 1 Go to official download page
- 2 Download the lib
- 3 Install the lib



Get early access to OAuth.io beta

Your email

Sign up

Follow @oauth_io 224 followers

powered by [webshell](#)



Google Developers

Google Accounts Authentication and Authorization X

Search

dbaler@gmail.com
Sign out

HomeProductsConferencesShowcaseLiveGroups

Google Accounts Authentication and Authorization +1 1.8k

Choosing an Auth Mechanism

- Authentication
 - OpenID
 - OAuth 2.0**
 - Mobile App Authentication
- API Authorization
 - OAuth 2.0
 - Login**

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.

Sign in with Google

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

Identity Provider

Authorization Endpoint

Token Endpoint

UserInfo Endpoint

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



User Agent



Client

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

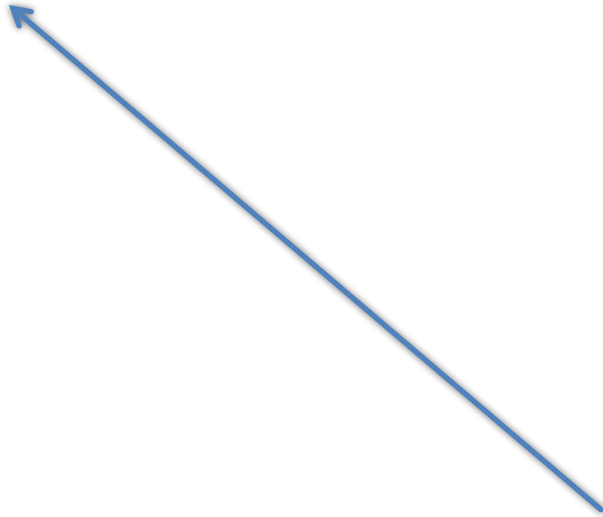
Step 1b: Authentication

Identity Provider

Authorization Endpoint

Token Endpoint

UserInfo Endpoint



User Agent



Client

Step 1c: Consent

Identity Provider

Authorization Endpoint

Token Endpoint

UserInfo Endpoint



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



User Agent



Client

Step 1d: Authorization Response

Identity Provider

Authorization Endpoint

Token Endpoint

UserInfo Endpoint

GET /cb?
code=abc&
state=123



User Agent



Client

Step 2a: Token Request

Identity Provider

Authorization Endpoint

Token Endpoint

UserInfo Endpoint

POST /token

Authorization: Basic (client_id:secret)

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



User Agent



Client



Step 2b: Token Response

Identity Provider

Authorization Endpoint

Token Endpoint

UserInfo Endpoint

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



User Agent



Client

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

Identity Provider

Authorization Endpoint

Token Endpoint

UserInfo Endpoint

GET /userinfo

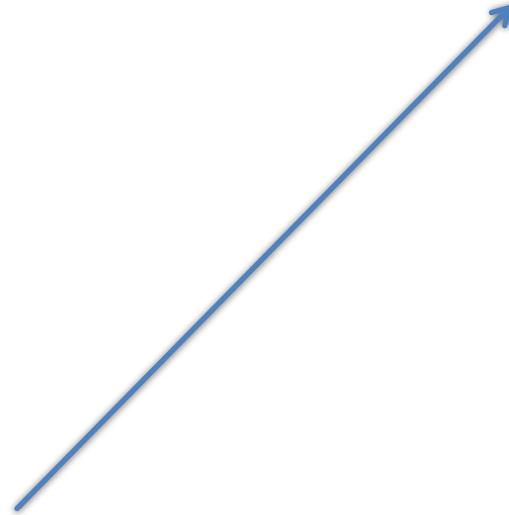
Authorization:
Bearer access_token



User Agent



Client



Step 3b: UserInfo Response

Identity Provider

Authorization Endpoint

Token Endpoint

UserInfo Endpoint

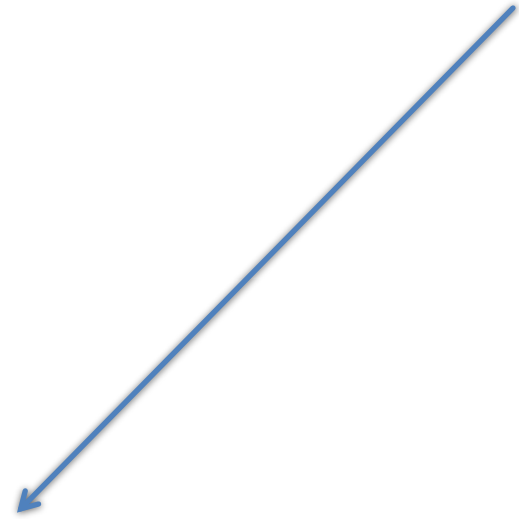
```
{  
  "sub": "248289761001",  
  "name": "Jane Doe",  
  "email": "janedoe@example.com"  
}
```



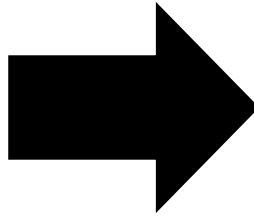
User Agent



Client



Criticism & Concerns



artwork by @ChrisMCarrasco

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](#)]

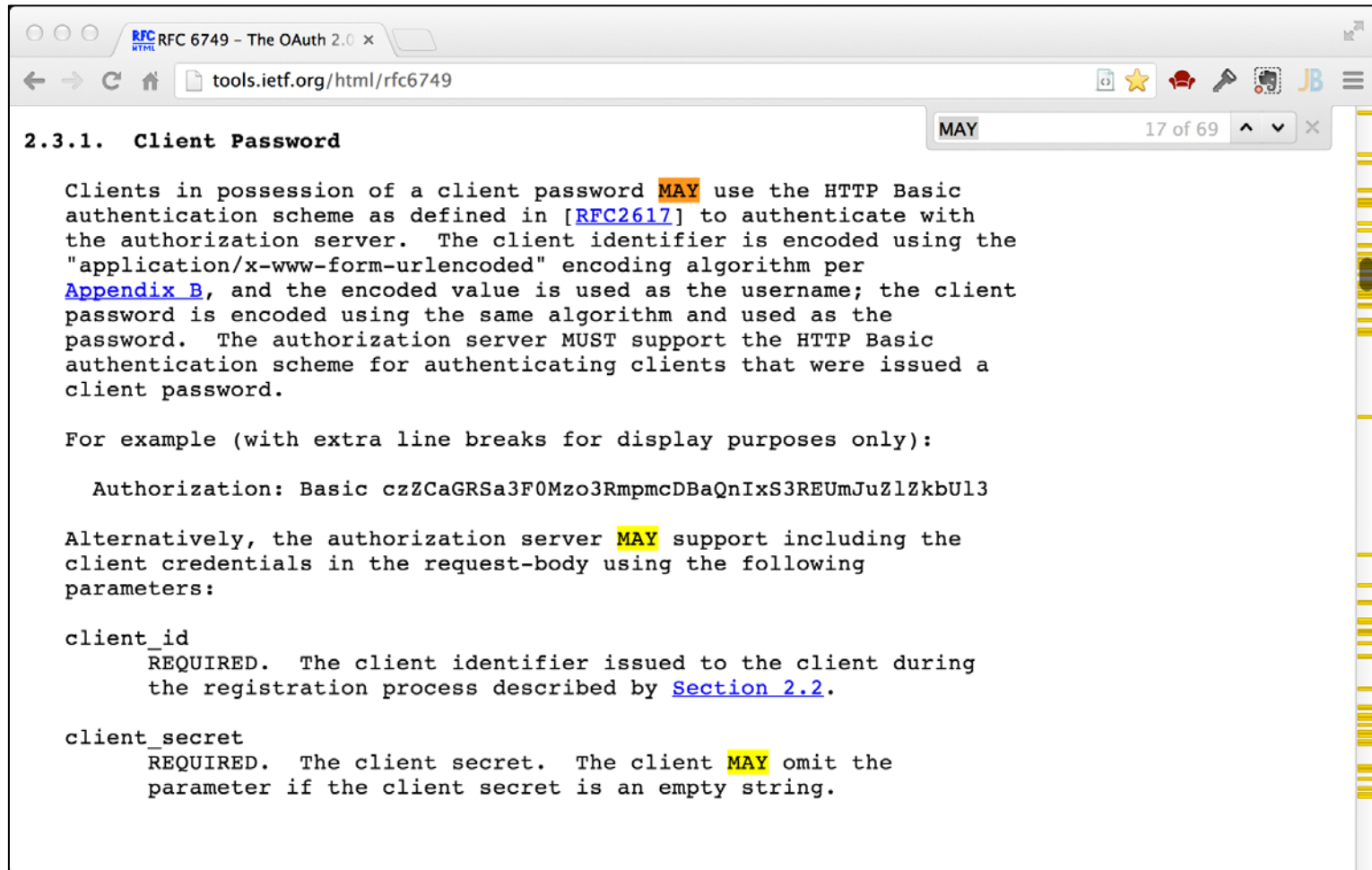
PROPOSED STANDARD

Internet Engineering Task Force (IETF)
Request for Comments: 6749
Obsoletes: [5849](#)
Category: Standards Track
ISSN: 2070-1721

D. Hardt, Ed.
Microsoft
October 2012

The OAuth 2.0 Authorization Framework

"A Framework to build Protocols"



The screenshot shows a web browser window with the address bar displaying `tools.ietf.org/html/rfc6749`. The page title is "RFC 6749 - The OAuth 2.0". The main content area is titled "2.3.1. Client Password". The text describes the use of client passwords for authentication, mentioning the HTTP Basic scheme and the encoding of client identifiers and passwords. It includes an example of an Authorization header and lists parameters for the request body: `client_id` and `client_secret`. The word "MAY" is highlighted in yellow in the original image.

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

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).

Developers & SSL



how to handle SSL validation error



[SSL Certificate Validation Error in .Net « Akbar's Blog](#)

blog.syedgakbar.com/.../ssl-certificate-validation-error-in-net/

Jul 17, 2012 – This callback method is used to **validate** the certificate in an **SSL** conversation // Changed the **handle** to ignore the **SSL Certificate errors** in the ...

[SSL Function Return Codes](#)

publib.boulder.ibm.com/infocenter/.../sssl2msg1000885.htm

The environment or **SSL handle** specified on a System **SSL** function call is not ...
Certificate **validation error**. ... An error is detected while validating a certificate.

[Ignoring SSL validation in Java - Stack Overflow](#)

stackoverflow.com/questions/.../ignoring-ssl-validation-in-java

2 answers - 20 Nov 2012

Foreword: I DO know that skipping **SSL validation** is really ugly. In this ...
ClientStateReceivedServerHello.**handle**(Unknown Source) at ... catch (
KeyManagementException e) { log.**error** ("No **SSL** algorithm support: " + e.

[How to handle invalid SSL certificates with Apache - Stack Overflow](#)

stackoverflow.com/.../how-to-handle-invalid-ssl-certificates-wi...

9 answers - 1 Dec 2009

... at sun.security.validator.Validator.**validate**(Validator.java:235) at sun.security.**ssl**. ...
When I go to mms.nw.ru, I get a **error** screen in Chrome.

Infrastructure & SSL



GIGAom

Events ↗

Research ↗

Jobs ↗

paidContent ↗

Home

Apple

Cleantech

Cloud

Data

Europe

Mobile

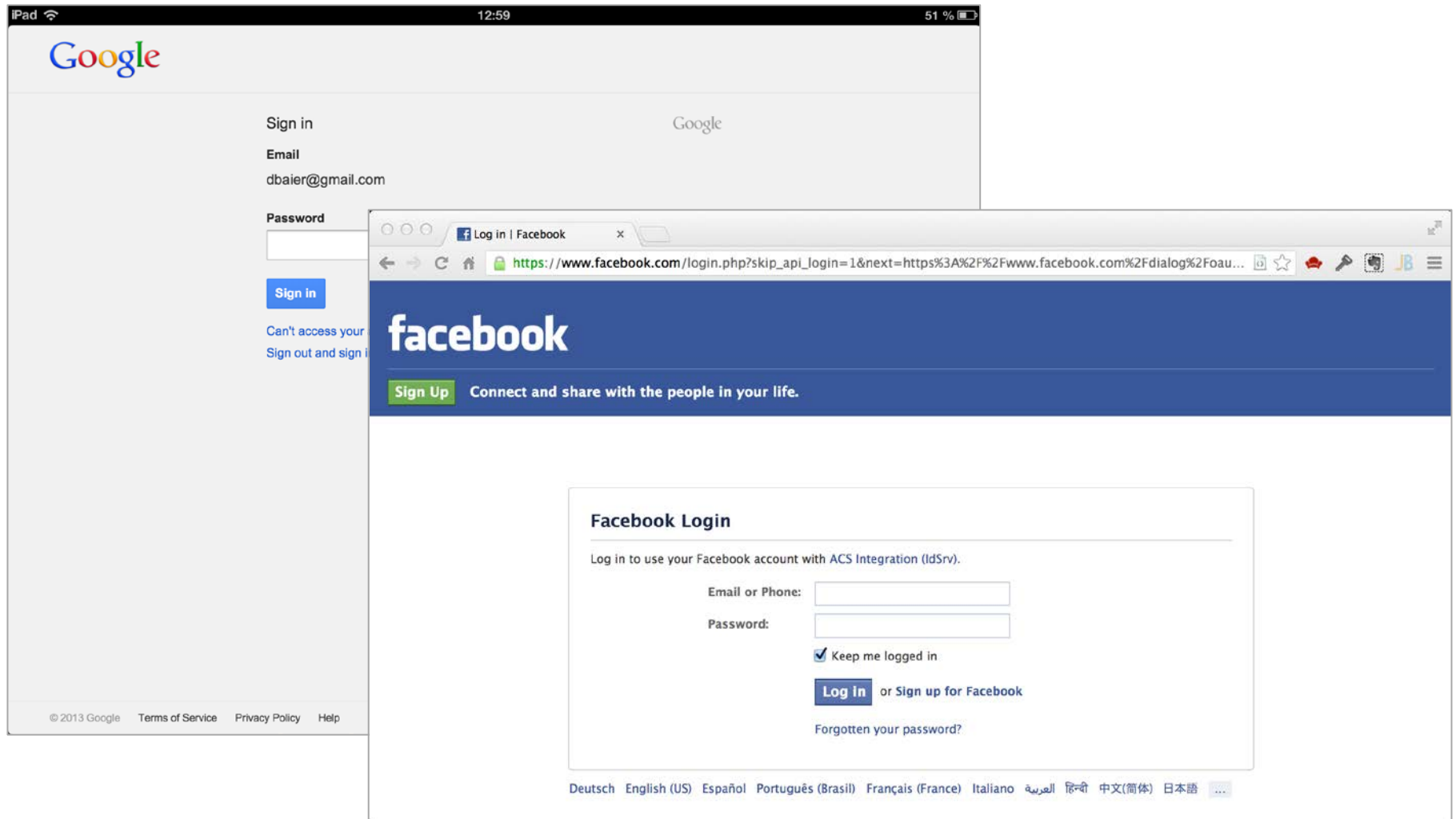
Video

[finland](#) / [nokia](#) / [security](#)

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





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

Summary

- **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>