In Partnership with

codiscope

MANICODE

SECURE CODING EDUCATION

OAuth 2.0 Security Introduction

JIM MANICO Secure Coding Instructor *www.manicode.com*

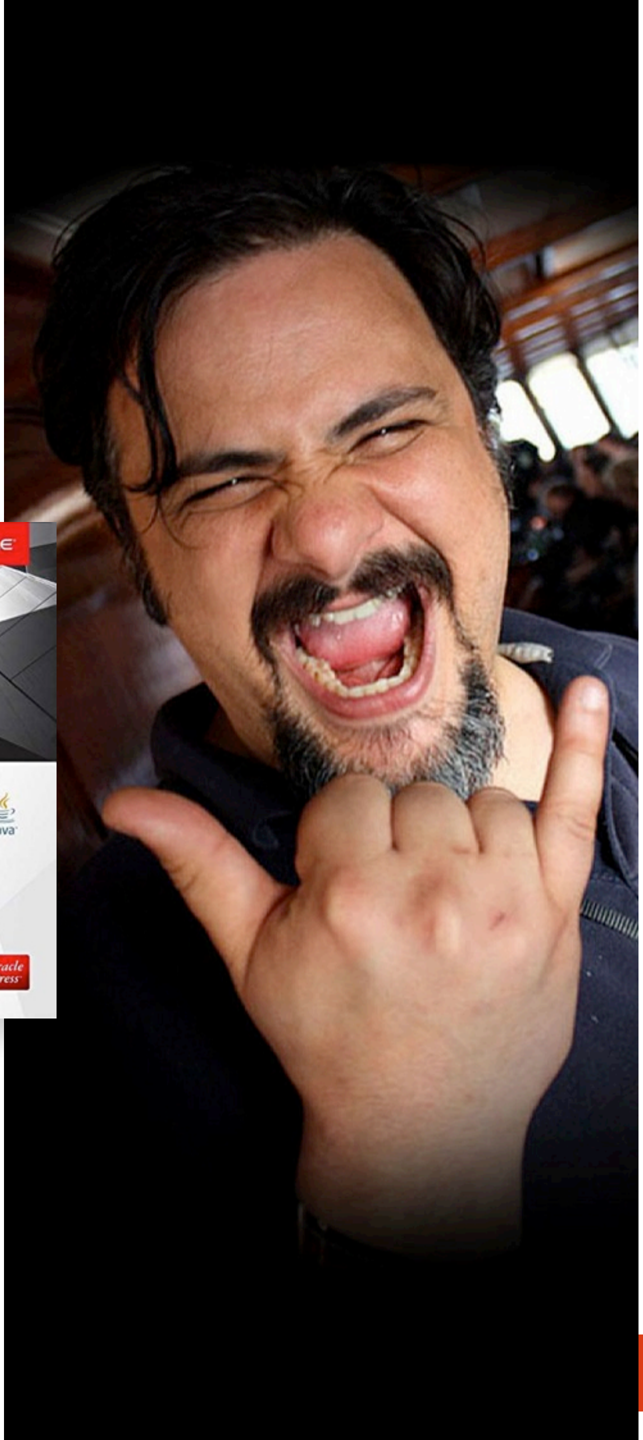
A little background dirt…

jim@manico.net

@manicode

- 19+ years of software development experience
- Former OWASP Global Board Member
- Project manager of the OWASP Cheat Sheet Series and several other OWASP projects
- Author of "Iron-Clad Java, Building Secure Web Applications" from McGraw-Hill/Oracle-Press
- Kauai, Hawaii Resident

ORACLE
Iron-Clad Java: Building Secure Web Applications
Best Practices for Secure Java Web Application Development
Jim Manico
August Detlefsen
Contributing Author, Kevin Kenan
Technical Editor, Milton Smith
Oracle Press



OAuth 2.0: Where are we going?

OAuth Terms

Client Registration

OAuth Grant Types

OAuth Threat Model

OAuth Countermeasures and Controls

So, what is OAuth 2.0?

OAuth is like a valet key.

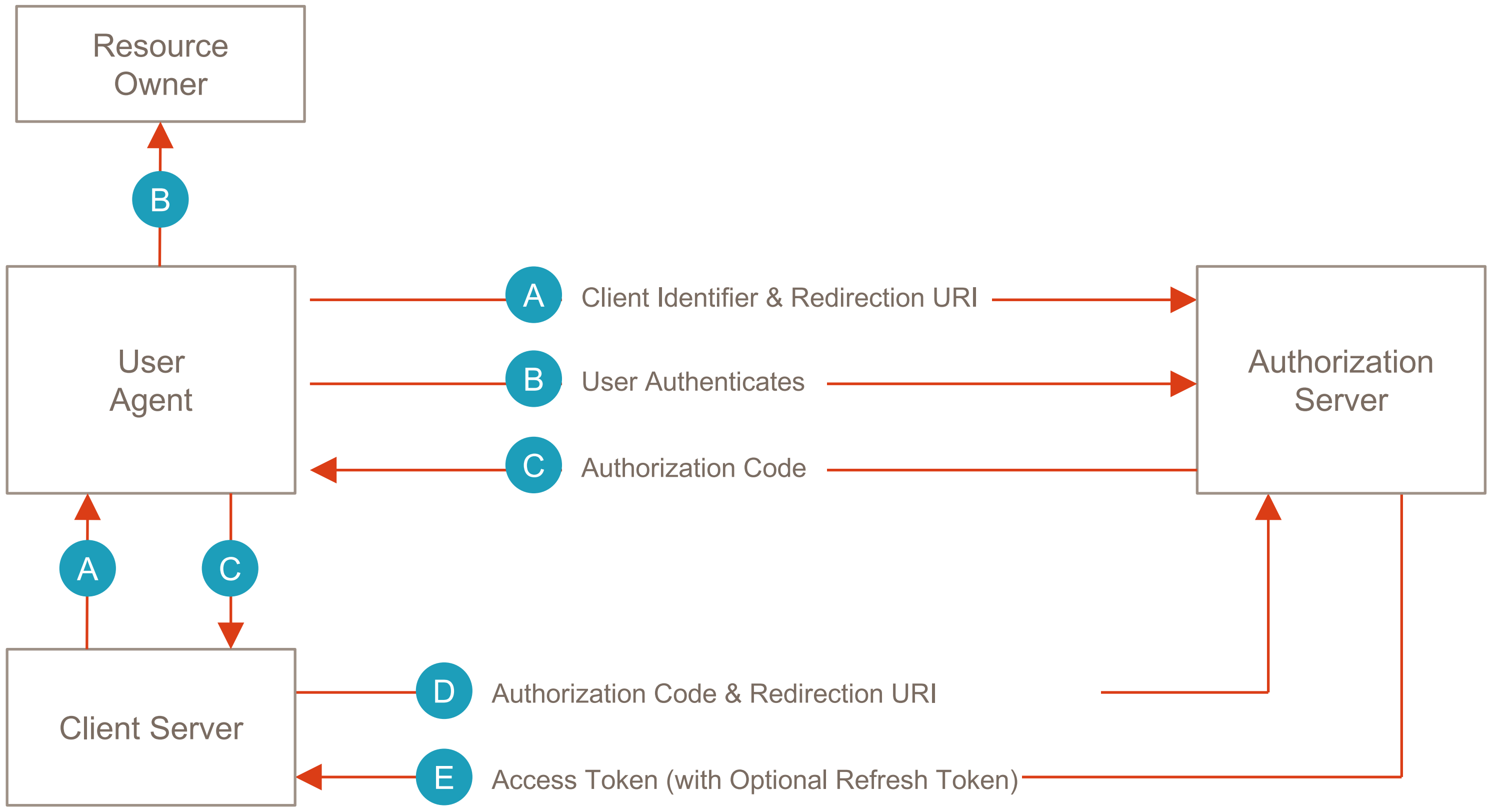
It provides another domain delegated access to your application server resources.

What should OAuth NOT be used for?

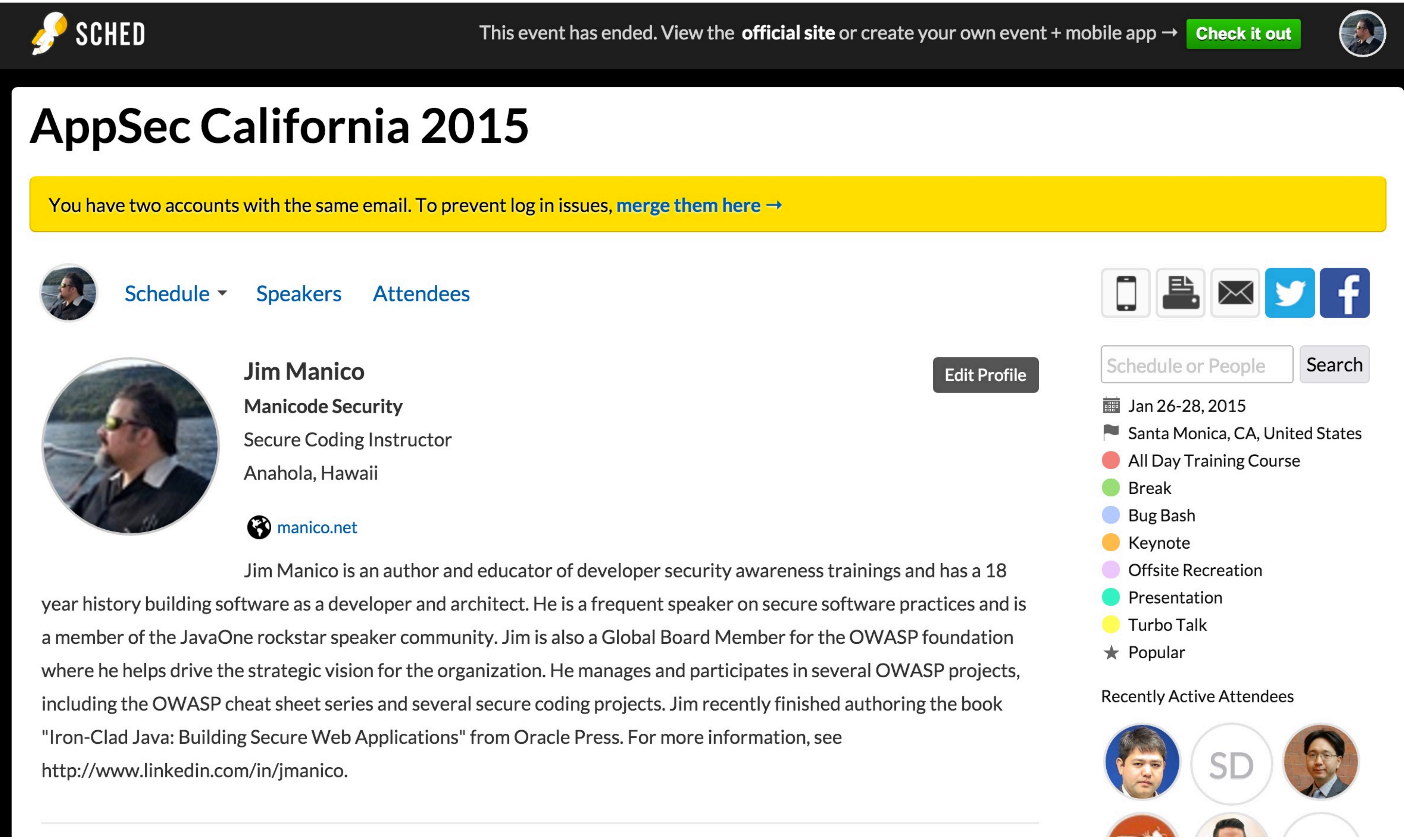
- OAuth should not be used for traditional access control.
- OAuth should not be used for authentication.
- OAuth should not be used for federation.

OAuth should be used for delegation!

OAuth 2.0 Authorization Code Grant


Resource Owner
B
User Agent
A
Client Identifier & Redirection URI
B
User Authenticates
C
Authorization Code
Authorization Server
A
C
Client Server
D
Authorization Code & Redirection URI
E
Access Token (with Optional Refresh Token)

http://tools.ietf.org/html/rfc6749



SCHED

This event has ended. View the **official site** or create your own event + mobile app → Check it out

AppSec California 2015

You have two accounts with the same email. To prevent log in issues, merge them here →

Schedule Speakers Attendees

Jim Manico

Manicode Security

Secure Coding Instructor

Anahola, Hawaii

Edit Profile

manico.net

Jim Manico is an author and educator of developer security awareness trainings and has a 18 year history building software as a developer and architect. He is a frequent speaker on secure software practices and is a member of the JavaOne rockstar speaker community. Jim is also a Global Board Member for the OWASP foundation where he helps drive the strategic vision for the organization. He manages and participates in several OWASP projects, including the OWASP cheat sheet series and several secure coding projects. Jim recently finished authoring the book "Iron-Clad Java: Building Secure Web Applications" from Oracle Press. For more information, see http://www.linkedin.com/in/jmanico.

Schedule or People Search

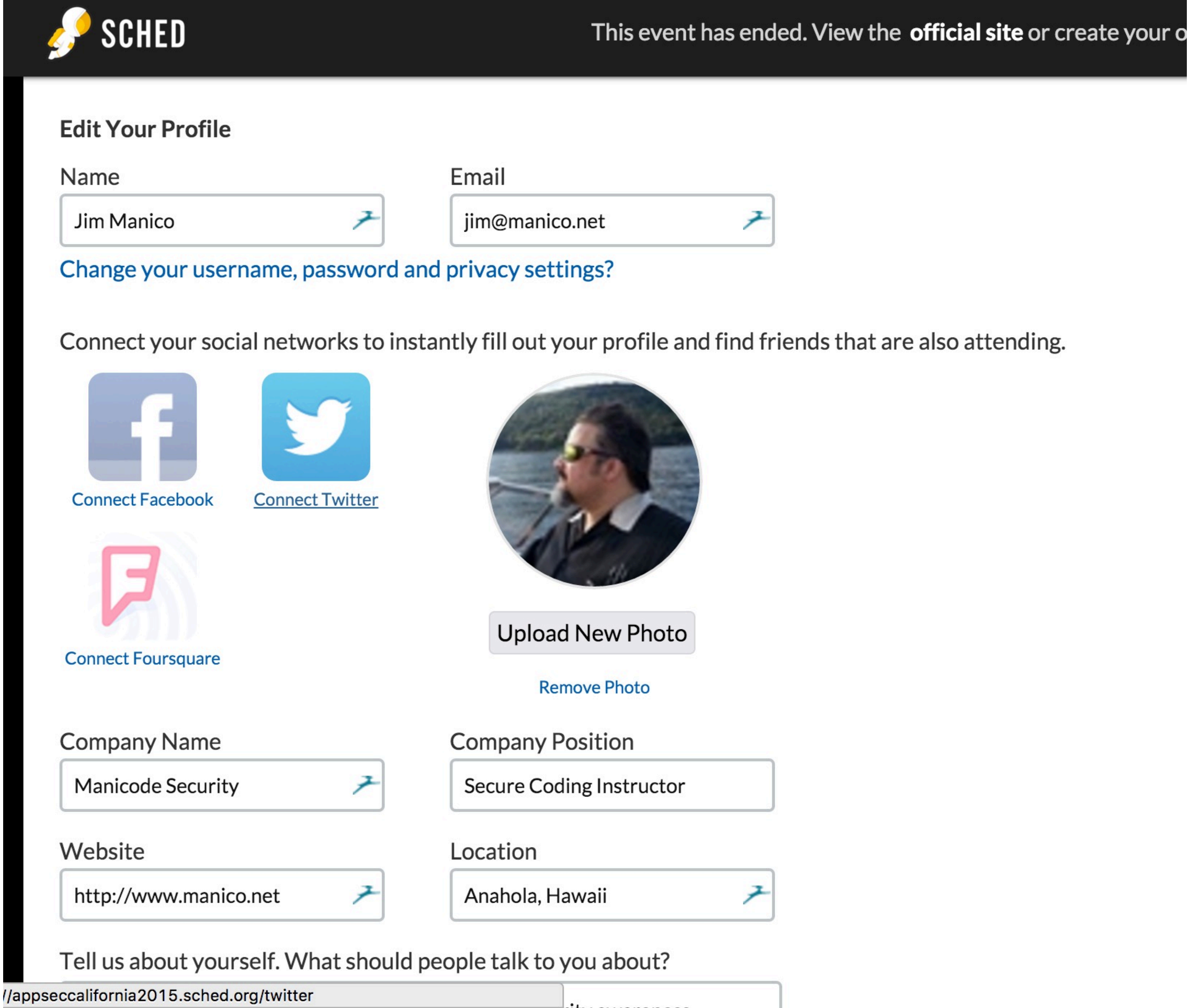
Jan 26-28, 2015

Santa Monica, CA, United States

● All Day Training Course
● Break
● Bug Bash
● Keynote
● Offsite Recreation
● Presentation
● Turbo Talk
★ Popular

Recently Active Attendees

SD



SCHED

This event has ended. View the **official site** or create your o

Edit Your Profile

Name	Email
Jim Manico	jim@manico.net

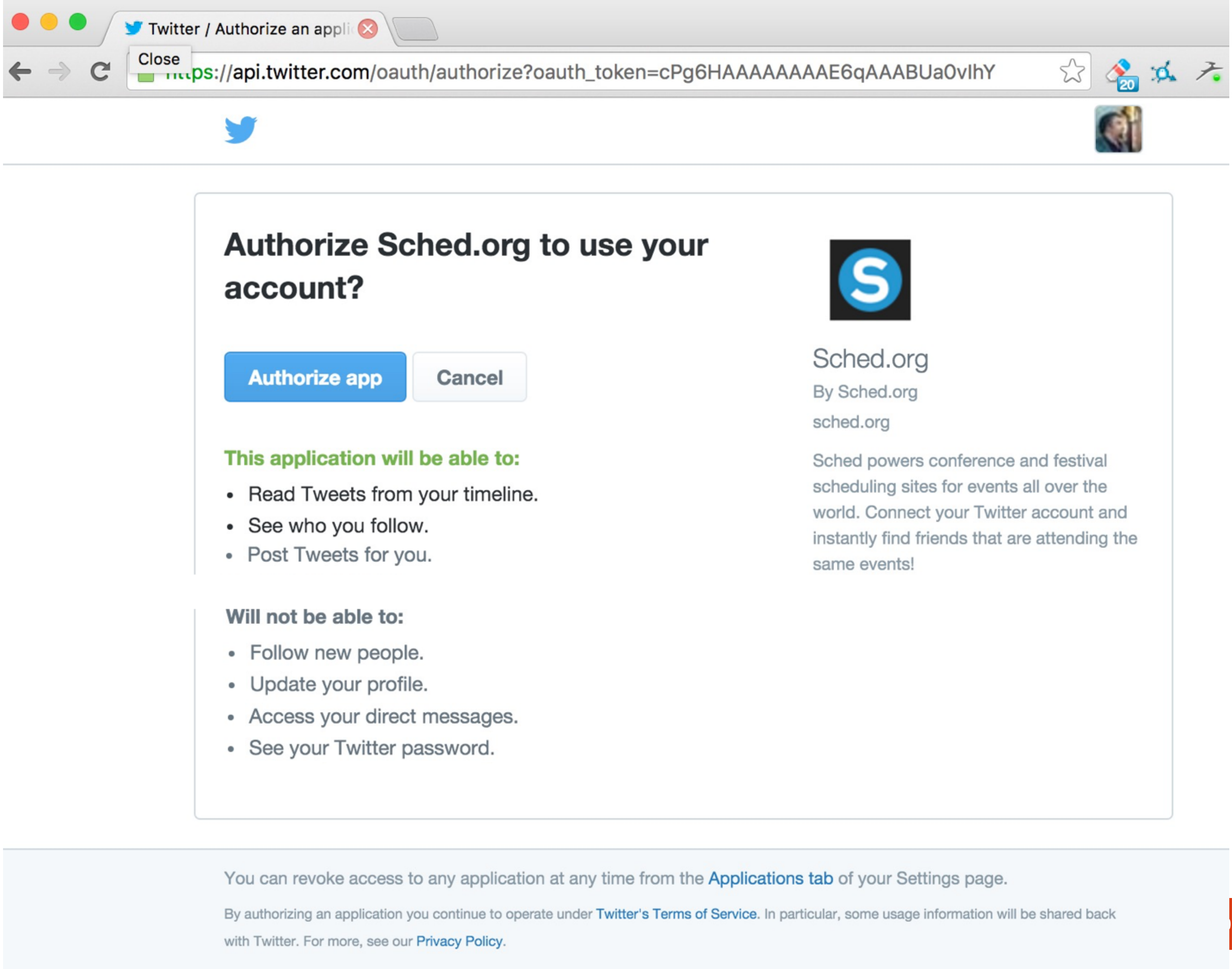
Change your username, password and privacy settings?

Connect your social networks to instantly fill out your profile and find friends that are also attending.

Company Name	Company Position
Manicode Security	Secure Coding Instructor
Website	Location
http://www.manico.net	Anahola, Hawaii

Tell us about yourself. What should people talk to you about?

/appseccalifornia2015.sched.org/twitter


Twitter / Authorize an appli
Close
https://api.twitter.com/oauth/authorize?oauth_token=cPg6HAAAAAAAE6qAAABUa0vlhY
20

Authorize Sched.org to use your account?

This application will be able to:

- Read Tweets from your timeline.
- See who you follow.
- Post Tweets for you.

Will not be able to:

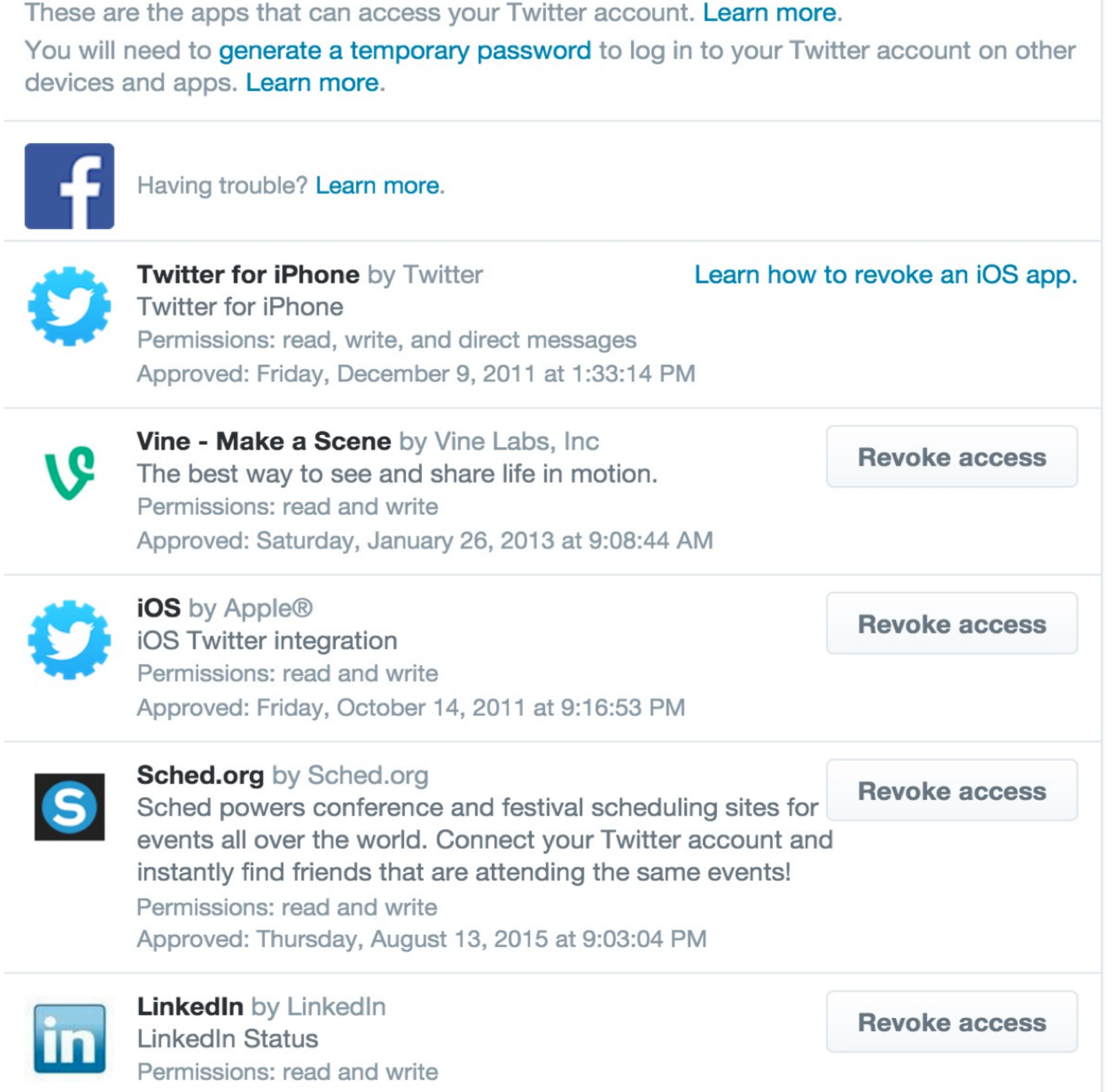
- Follow new people.
- Update your profile.
- Access your direct messages.
- See your Twitter password.

Sched.org
By Sched.org
sched.org

Sched powers conference and festival scheduling sites for events all over the world. Connect your Twitter account and instantly find friends that are attending the same events!

You can revoke access to any application at any time from the Applications tab of your Settings page.

By authorizing an application you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.



These are the apps that can access your Twitter account. Learn more.

You will need to generate a temporary password to log in to your Twitter account on other devices and apps. Learn more.

Having trouble? Learn more.

Twitter for iPhone by Twitter
Twitter for iPhone
Permissions: read, write, and direct messages
Approved: Friday, December 9, 2011 at 1:33:14 PM
Learn how to revoke an iOS app.

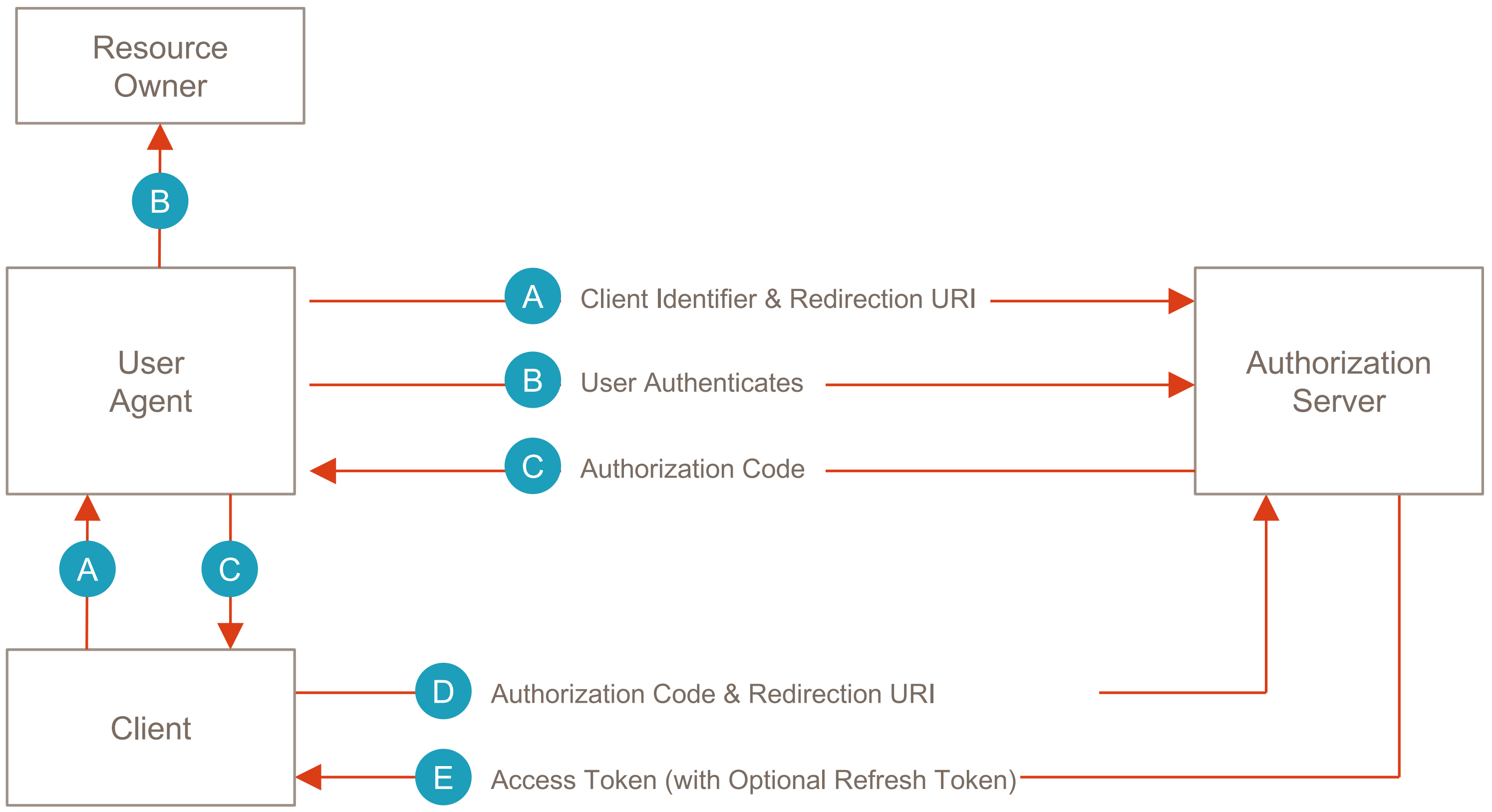
Vine - Make a Scene by Vine Labs, Inc
The best way to see and share life in motion.
Permissions: read and write
Approved: Saturday, January 26, 2013 at 9:08:44 AM
Revoke access

iOS by Apple®
iOS Twitter integration
Permissions: read and write
Approved: Friday, October 14, 2011 at 9:16:53 PM
Revoke access

Sched.org by Sched.org
Sched powers conference and festival scheduling sites for events all over the world. Connect your Twitter account and instantly find friends that are attending the same events!
Permissions: read and write
Approved: Thursday, August 13, 2015 at 9:03:04 PM
Revoke access

LinkedIn by LinkedIn
LinkedIn Status
Permissions: read and write
Revoke access

OAuth 2.0 Authorization Code Grant


Resource Owner
B
User Agent
A
Client Identifier & Redirection URI
B
User Authenticates
C
Authorization Code
Authorization Server
A
C
Client
D
Authorization Code & Redirection URI
E
Access Token (with Optional Refresh Token)

http://tools.ietf.org/html/rfc6749

OAuth 2.0 Terminology

Client Application Definition

Client Application

Application requesting access to protected resource on behalf of resource owner. The application that the resource owner is providing access to.

Client Application Types

- Mobile applications
- Web browsers
- Desktop applications
- Web server

Confidential vs Public Client Applications

Confidential Clients

An application that must register with authorization servers. Authorization servers give unique client secrets to confidential clients when they successfully register.

Public Clients

An application that anyone can download and use. Mobile apps and native applications are example of public clients.

High Level Concepts

Protected Resource
Valuable data or features protected by the service provider

OAuth 2.0 Actors
Resource owner/user, client application, resource server, authorization server, user-agent

Token Types
- Refresh token
- Access token
- Authorization code token

OAuth 2.0 Grant
Types authorization code, implicit, resource-owner password credentials, client credentials, extension.

Extension Grants
An important extension used in OIDC is the ID token. This is very useful as no access is delegated but identity is established.

Token Types

Access Token

OAuth token used to directly access protected resources on behalf of a user or service.

Refresh Token

Refresh tokens, when given to the authorization server, will provide a new active access token. Refresh tokens themselves cannot access resources. While access tokens should be short lived, refresh tokens are long lived or simply never expire until the user revokes them. Refresh tokens also provide more scalable patterns.

Authorization Code Token

Authorization code tokens that are specific and exclusive to the authorization code grant type used to retrieve access and/or refresh tokens.

Additional OAuth 2.0 Terms

Resource Owner or End-User

User and account owner of resource (the end-user)

Resource Server/Service Provider

Server hosting protected resources owned by the end-user. Accepts access tokens for protected resources.

Authorization Server/Service Provider

Server issuing access tokens to provide other clients access to protected resources. Often same server as resource server. One authorization server may issue access tokens to many resource servers.

Other OAuth Term

Client Identifier

Unique ID used in part to authenticate a client application to an authorization server.

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)."

— *https://tools.ietf.org/html/rfc6750#section-1.2*

Danny (resource owner) has an account with Twitter (service provider). Danny is also a regular customer of the website Ono Ono Lau Lau Hawaiian Cooking (confidential client application). Danny can grant Ono Ono Lau Lau access to Danny's protected ability to tweet at Twitter (resource server), without sharing Danny's user name and password with Ono Ono Lau Lau's website (client application).

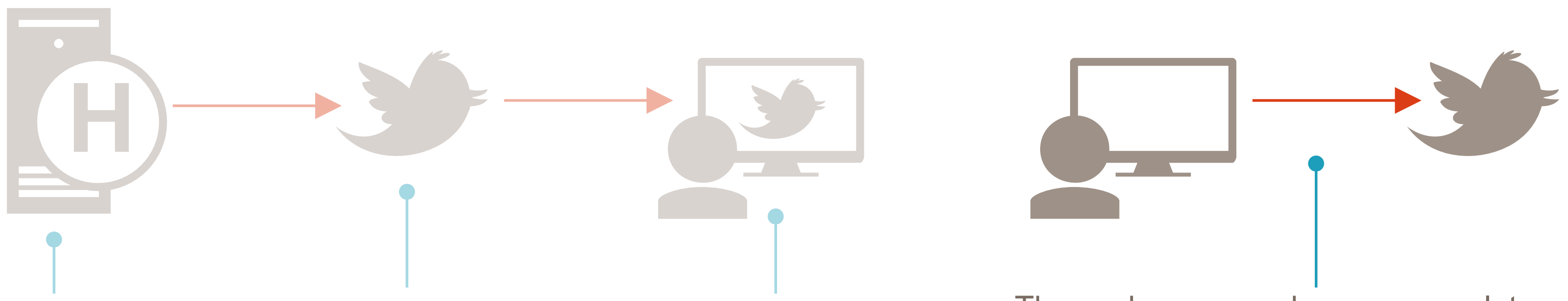
Instead, Danny authenticates directly with Twitter (authorization server), which issues the Ono Ono Lau Lau website an access token and a refresh token that will let the Ono Ono Lau Lau website tweet (access protected resources) on behalf of the user every time they upload a new recipe.

Introduction to OAuth 2.0

Sample OAuth Workflow

Using OAuth, your eCommerce server can now tweet on behalf of the user even when the user is not logged on.

How does this happen?



First, the user logs into the eCommerce server with his account and edits his account profile.

Next, the eCommerce server redirects the user to Twitter.

The user logs onto Twitter and authorizes the eCommerce server to tweet on her behalf.

Then, whenever orders are complete the eCommerce tweets a little note about how awesome the eCommerce company is - even when the user is not logged onto the eCommerce server.

OAuth v1

Founded in Cryptography	…especially digital signatures. A signed message is tied to its origin.
OAuth 1.0 Messages are Individually Signed	If a single message within the communication is constructed or signed improperly, the entire transaction will be invalidated
Limited Device Support Beyond the Web	Mobile? Native? Sad Panda?
Platform Interoperability and Implementation Challenges	Crypto is hard.

OAuth 2.0

OAuth 2 Transport Security	Security is delegated to HTTPS/TLS.
Centered around bearer tokens	These are easy for integration but not great for security. ID Tokens, where only user info is required and no delegated access is provided, are also key in OAuth 2.0.
Much easier to work with	OAuth 2.0 is much more usable, but much more difficult to build securely.
Much more flexible	OAuth 2.0 considers non-web clients as well.
Better separation of duties	Resource requests and user authorization can be decoupled in OAuth 2.0.

OAuth v1 and v2: Which should you use?

Google moved away from OAuth 1.0 in April 2012.

Twitter still supports OAuth 1.0.

It’s rare for new server implementations to support OAuth 1.0.

Plenty of OAuth 2.0 “add-on” RFC’s to support crypto if needed.

So 2.0 in almost all situations in 2017+.

OAuth and Interoperability...

Because OAuth 2.0 is more of a framework than a protocol like OAuth 1.0, OAuth 2.0 implementations are less likely to be naturally interoperable with any other OAuth 2.0 implementations.

Unfortunately, the OAuth 2.0 specification leaves a few required components partially or fully undefined (for example, client registration, authorization server capabilities, and endpoint discovery).

OAuth 2.0 Grant Types

Authorization Code Grants can hide long-lived tokens from the user. This is the only way a client would never get to see user credentials.

Implicit Grants can only activate a short-lived token in the browser when the user is currently logged on.

Resource-Owner Password Credentials Grants can grant and expose long-lived tokens directly to the user via a trusted client. This means the client captures the users credentials and presents these to the Identity Provider to authenticate. It's best for the client to only store the refresh and access tokens and NOT store the credentials.

Client Credentials Grants can grant and expose a long-lived token directly to a client application that needs to access data not associated with a specific user. Should only be used when the user cannot be redirect to IDP login pages, or login pages cannot be embedded in client. Typical example is a agent in ADFS or a Radius Frontend.

OAuth 2.0 Tokens

Access Tokens

Access tokens, simply put, are OAuth tokens used to access protected resources.

- Access Token Fields
  - token_type (required). The type of the token issued (mac token, bearer token, etc).
  - expires_in (recommended).The lifetime in seconds of the access token.
  - scope (required). The level of access permitted when retrieving user resources. It is critical to build OAuth solutions that limit the scope via principle of least privilege. Also ensure that the scope is messages properly to user so the message and actual delegated access are in sync.
- Access tokens when used as a bearer token
  - Bearer tokens are defined by RFC6750
  - https://tools.ietf.org/html/rfc6750

Access Token General Countermeasures

- Review treat analysis from https://tools.ietf.org/html/rfc6819
- Ensure access tokens are all short-lived
  (low session time or even one use per token).
- Consider supporting token revocation at the authorization server per RFC 7009 (OAuth 2.0 Token Revocation).
- Ensure scope of access tokens are as limited as possible.
- Protect access tokens on authorization servers in a one-way fashion like other credentials used for authentication.
- Protect access tokens in client applications in a temporary, secure storage mechanism.

Access Token General Countermeasures

- Ensure the authorization server provides only necessary grants.
- Ensure hardening of the authorization server from token theft.
- Ensure basic web server and web application security best practices.
- Use well-configured transport layer security (TLS) in all aspects of OAuth 2.0 communication.
- Ensure high-entropy access tokens that cannot be guessed or brute-forced.
- Consider strong authentication between the client application and authorization server.

Refresh Tokens

- An active refresh token can request a new and active access token from the authorization server repeatedly.
- Another distinction between a refresh token and an access token is who consumes them.
  - The refresh token is used only by the authorization server, and the access token is used only by a resource server.
- Refresh tokens can be long-lived: *they will persist until a user invalidates them.*
- Other than Social deployment an application is not likely to need a refresh token. This is a fundamental question in order to understand the use cases of the application. Do not allow a refresh token if the client does not need it!

Refresh Token General Countermeasures

- Consider not supporting refresh tokens. They are long-lived active sessions and are inherently dangerous.
- Ensure refresh tokens can be easily listed and revoked by the resource owner (user).
- Consider supporting token revocation at the authorization server per RFC 7009 (OAuth 2.0 Token Revocation).
- Ensure the scope of refresh tokens is as limited as possible (refresh tokens should only be able to request specific and scope-limited access tokens).
- Protect refresh tokens on authorization servers in a one-way fashion like other credentials used for authentication.
- Protect refresh tokens in client applications using secure long-term storage mechanisms.

Refresh Token General Countermeasures

- Ensure hardening of the authorization server and client server from token theft.
- Ensure web application security best practices for all servers.
- Use well-configured TLS in all aspects of OAuth 2.0 communication.
- Ensure high-entropy refresh tokens that cannot be guessed or brute-forced.
- Bind refresh tokens to client ID and client secret.
- Consider strong authentication between the client application and authorization server.
- Consider using proof-based tokens instead of bearer tokens for refresh tokens.

Client Registration

Client Registration Overview

- Client registration occurs when the client application successfully registers with a service provider.
  (authorization server and resource server).
- OAuth 2.0 authorization servers can require client applications to successfully register before other OAuth communication (like resource server requests) can occur.

Client Types

There are two major categories of client applications defined by OAuth 2.0 – Confidential Clients and Public Clients

Confidential Clients

A confidential client can be a web server, web service, Cron job, or some legacy system.

- Web Applications or Web Services

Public Clients

A public OAuth client is a client that does not require registration.

- Mobile or Native Applications
- *Trusted or Third-Party Public*

What Clients Provide Authorization Server At Registration Time

- Client must tell the service provider what kind of client it is.
  - Confidential Clients (i.e., web application)
  - Public Clients (i.e., browser or native clients)
- Client should provide a client redirection URI (or URIs) to the authorization server. The redirect URI or URIs are used by the authorization code and implicit grant types.
- Client must provide the authorization server with other needed info such as description, website, logo, etc.

What Authorization Server Create and Provide at Registration Time

When clients successfully register with an authorization server, the following happens:

- The authorization server provides each confidential client a unique client identifier for future communication.
- Confidential clients are also given a unique client secret (credentials like passwords or keys) that is required (along with the client identifier) for future communications. THIS IS A massive weakness that should be replaced with mutual TLS if possible.

Client Registration Threats

- During the OAuth 2.0 registration process between a client application and an authorization server, attackers can take advantage of aspects of the OAuth 2.0 client application registration process to execute these attacks.
- They can use the authorization server as an open redirector or redirect the user to a malicious page based on weak client registration.
- They can steal data from the network that OAuth communication happens on due to weak or missing encryption in transit.
- They can use a malicious client application to steal data from the authorization server based on weak client registration.
- They can set up a fraudulent authorization server meant to steal credentials or hijack access from a genuine service.

Client Registration Controls

The following defenses are critical when considering security issues that can arise due to poorly built client registration security.

1. Guarantee network communication confidentiality and integrity; authorization server and resource server authenticity.
2. Ensure only genuine confidential clients can register with the authorization server.
3. Ensure proper management of public clients.
4. Ensure authorization server protection from malicious or open redirection via weak registration.

Grant Types

OAuth 2.0 Grant Types

You can hide long lived tokens token from the user **(authorization code grant)**

You can only activate a short-lived token in the browser when the user is currently logged on (**implicit grant**)

You can grant and expose a long-lived tokens directly to the user via a trusted client (**password grant**) but never do this in a browser!

You can grant and expose a long-lived token directly to other services that need to access data not associated with a specific user (**client credentials grant**)

Authorization Server Security

- TLS for everything (Authenticity, Confidentiality, Integrity)
- Authorization servers should not automatically process repeat authorizations to public clients unless the client is validated using a pre-registered redirect URI (Section 5.2.3.5).
- Authorization servers can mitigate the risks associated with automatic processing by limiting the scope of access tokens obtained through automated approvals (Section 5.1.5.1).
- Explain the scope (resources and the permissions) the user is about to grant in an understandable way (Section 5.2.4.2).
- Narrow the scope as much as possible (Section 5.1.5.1).
- Don't redirect to a redirect URI if the client identifier or redirect URI can't be verified (Section 5.2.3.5).

https://tools.ietf.org/html/rfc6819

Authorization Code Grant

OAuth 2.0 Authorization Code Grant

The whole purpose of the authorization code grant type is to provide a client application (web application or native client) access to users' protected resources from another service.

In short, the user delegates (usually limited) access to a client application.

3-Legged OAuth

Authorization Code Grant is often referred to as "three-legged OAuth 2.0”

- The first round trip redirects the user from the client application to the authorization server, where the user logs into the authorization server and is redirected back to the client application with a proper authorization code. The user credentials used in this step by the user are never exposed to the client application.
- The second round trip from the client application to the authorization server contains the authorization code. This code is used to gain access to an access or refresh token. No active refresh or access tokens are exposed to the user.
- The third round trip uses an active access token to request protected resources.

The user is never exposed or has access to the access token!

OAuth 2.0 Authorization Code Grant

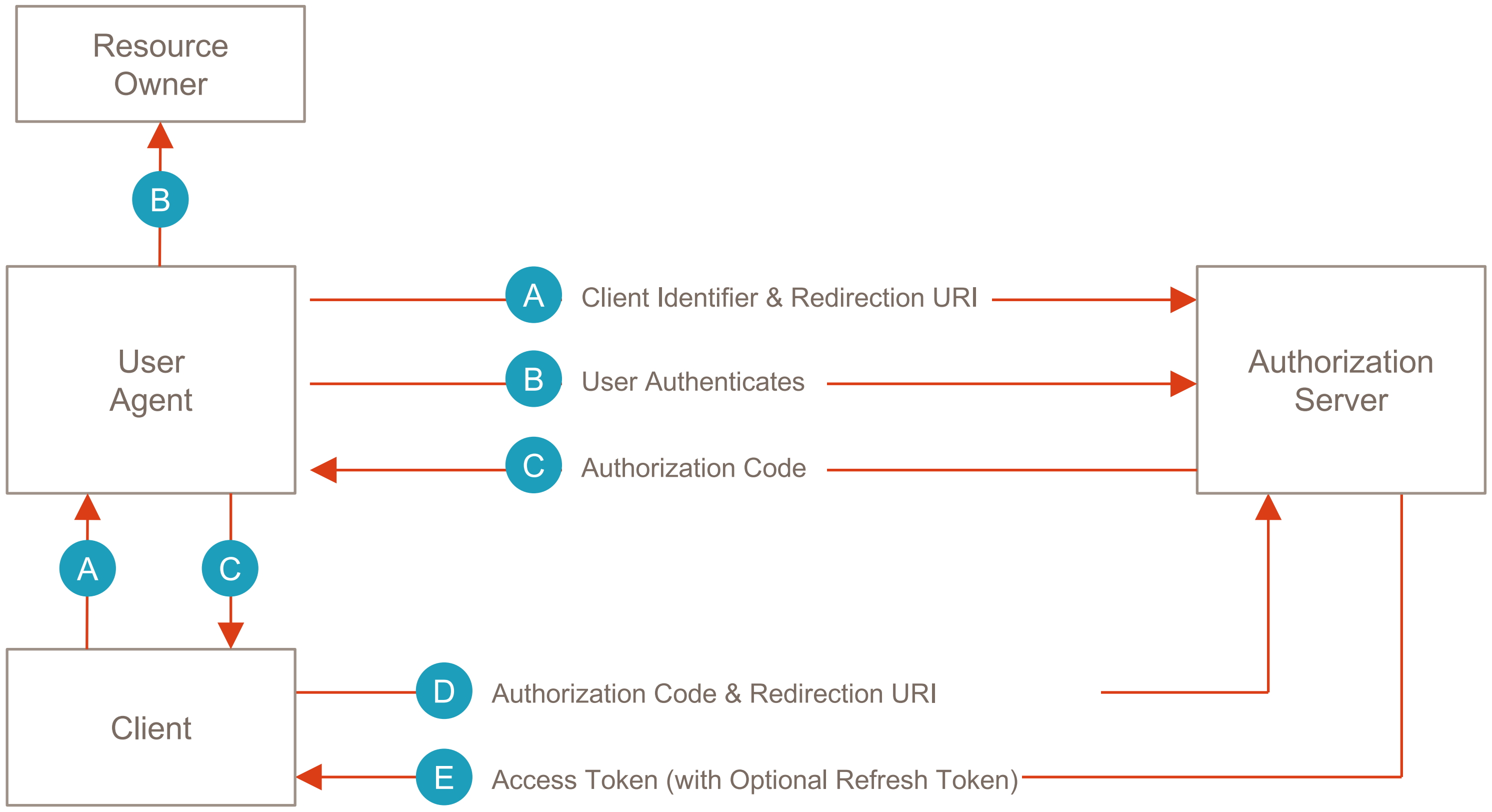
The User (Resource Owner) never has access to actual access token

The User (Resource Owner) never has access to actual access token

The Client application can use the access token even when the resource owner is not present

Authorization Code Refresh Tokens are often long lived or permanent until the User (Resource Owner) revokes this access through the Client UI.

OAuth 2.0 Authorization Code Grant


Resource Owner
B
User Agent
A
Client Identifier & Redirection URI
B
User Authenticates
C
Authorization Code
Authorization Server
A
C
Client
D
Authorization Code & Redirection URI
E
Access Token (with Optional Refresh Token)

http://tools.ietf.org/html/rfc6749

Authorization Code Variables

The client starts the "authorization code" workflow by redirecting the user to the authorization server with the right request data. This initial client request includes:

response_type : this is required by "authorization code" grant type and should contain the value "code"

client_id: this is the client identifier assigned to the client at client registration time. This is unique for every client for authorization code grants.

scope: level of access requested, domain specific

redirection URI: Where the authorization server redirects the user after access is granted or denied

OAuth 2.0 Authorization Code Grant: General Risks

- Threat agent can read authorization codes, tokens, client secrets, and other sensitive data via network sniffing, referrer header leakage, browser history, server logs, or unvalidated redirects.
- Threat agent can cause denial of service against service provider or client application that is a web application or web service.

OAuth 2.0 Authorization Code Grant: Authorization/Resource Server Risks

- Threat agent can read an authorization code, access token, client ID, or other client secret directly from the authorization server database due to SQL Injection, weak access control, or poor database security.
- Threat agent can take advantage of authorization server redirects to conduct open or malicious redirects.
- Threat agent can guess or brute force authorization codes, tokens, client ids or other client secrets.
- Malicious client applications can exploit existing trust to gain malicious access to protected resources.
- Authorization server provides a scope that is much more permissive than needed, leading to a variety of access control weaknesses.
- Attacker can change scope of an existing active token for data theft.
- Malicious or counterfeit authorization server can steal credentials.
- CSRF attack against authorization code can force service provider to authorize a user to access the threat agent's resources. From there the user may upload data into the threat agent's account.

OAuth 2.0 Authorization Code Grant: Client Application Risks

- Threat agent steals client ID, client secrets, access tokens, or refresh token directly from a client application. This is especially high impact when the token is a bearer token and is not tied to the client application with stronger authentication.
- Threat agent can take advantage of client application redirects to conduct open or malicious redirects.
- Threat agent can use a malicious client application to phish for user credentials.

OAuth 2.0 Authorization Code Grant Server Controls

General Controls

- TLS Everywhere
- Ensure all standard webserver and standard web security controls are implemented for all servers.
- Educate your users about the risk of phishing.

Resource Server Controls

- Ensure only active access tokens with proper scope are granted access to protected resources
- Consider limiting the number of uses (or one-time usage) for access tokens.

Client Application Controls

- Consider strong client authentication between the client application and the service provider.
- When receiving a URL redirected from the the authorization server with sensitive data, reload the user to a client URL that lets the client application consume the sensitive data (like an authorization code) but remove it from the user-agent.
- Ensure that all tokens are stored in a secure fashion.

OAuth 2.0 Authorization Code Grant Server Controls

During Client Registration

- Issue client secrets and client IDs only to clients with proper security policy.
- Ensure that all clients applications are given a unique and strong client ID/client secret pair after successful registration.
- Ensure all client applications are forced to register a full redirect URI with the authorization server.

Requesting an Authorization Code

- Verify that the pairing of client ID and client secret is valid.
- If the user is already authenticated at the service provider when being challenged to provide delegated access, force reauthentication.
- Explain scope to users as much as possible before they authorize access.
- Store authorization codes as you would other credentials in the authorization server.
- Only redirect to a URI that is registered with the authorization server.

OAuth 2.0 Authorization Code Grant Server Controls

Requesting Access Token or Refresh Token

- Verify authorization codes like other credentials (see credential storage module) in the authorization server.
- Limit the scope of all provided access tokens as much as possible.
- Force reauthorization when scope or other aspects of token changes.
- Ensure the validity of access tokens is short (the length of your client application session or less).
- Ensure that client secrets and all tokens for a specific user can be revoked by that user.
- Consider limiting authorization codes to only one use.
- Use the "state" parameter to avoid CSRF.

Detecting Malicious Activity

- Ensure that client ID, client secrets, and all active tokens for that client that can be easily revoked by the authorization server.

CSRF attacks against OAuth

CSRF Attacks against Oauth: Part 1

1. Attacker assumes that Victim is currently logged in at Client Site
   - *https://consumer-site.example/* (The OAuth Client Application)
2. Attacker goes through registration/login workflow at Client Site
   - *https://consumer-site.example/login* and uses that account to trigger a Oauth workflow with the Provider Service (The OAuth authorization/resource server)

1) Client Site redirects attacker to Provider Site login interface.
   - This is called the Authorization Request. *https://provider-site.example/login*

2) Attacker successfully logs in with Provider Site

3) Provider Site responds with redirect URL which contains the authorization code in the code parameter.
   - This is called the Authorization Grant *http://consumer-site.example/auth?code=1a2s3d4f5g6h*

CSRF Attacks against Oauth: Part 2

3. Instead of visiting or redirecting to the Authorization Grant redirect URL, Attacker copies the URL and places a reference to it in an image tag on a web page
   - *(<img src="http://consumer-site.example/auth?code=1a2s3d4f5g6h" />) (https://evil-page.example/)*

4. Attacker gets Victim to visit *https://evil-page.example/*.
   This in turn gets Victim to request the Authorization Grant URL http://consumer-site.example/auth?code=1a2s34f5g6h

5. By visiting the Authorization Grant URL, the Victim's Consumer Site account is now attached via OAuth to the attackers Service account. Any action that effects the service account by the Victim is accessible to the Attacker.

OAuth 2 and avoiding CSRF

1	**Consumer** generates unique random state value, and stores it in server side session variable. JSON Web Tokens are good for state values.
2	**Consumer** sends "state" parameter with Authorization Request
3	On successful authorization, **Provider Site** includes "state" parameter in Authorization Grant redirect URI
4	When **Victim** visits redirect URI, the "state" parameter is compared against the "state" parameter stored in server side session variable.

Use the "state" parameter, it is essentially a CSRF token!

What is an OAuth Code Flow Open Redirector Attack? How is it stopped?

OAuth 2 Authorization Code Flow
Open Redirector: Attack

1. Victim goes through login workflow at Consumer Site (https://consumer-site.example/login) using Provider Site for authorization.
2. Attacker constructs an Authorization Request URL for Provider Site
    1. redirect_uri is set to https://evil-site.example/
3. Attacker either embeds evil URL in an image tag or constructs a clickable link at Consumer Site.
4. When the evil URL is loaded, the provider will 302 redirect back to redirect_uri since user was already logged in.
5. When the redirect occurs, the evil site can read the HTTP Referrer to get the Authorization Code.
6. Using this Authorization Code, Attacker can login as user

OAuth 2 Authorization Code Flow
Open Redirector: Remediation

Use a server side redirect_uri that enforces strict policy

At the very least, server side whitelist should enforce what the redirect_url must start with for added flexibility.

What Happens if an Access Token is Stolen?

OAuth 2 Implicit Flow
Access Token Reuse: Attack

1. Victim authorizes with Evil Consumer Site for Provider Site using access_token
2. Acme Widgets Consumer Site uses Implicit Flow for authentication.
3. Attacker authenticates as Victim with the Evil Consumer Site access_token using https://acme-widgets.example/callback#access_token=access_token

"One Token to Rule Them All"

Conclusion

OAuth 2.0 Summary

- It takes massive efforts to build secure OAuth 2 solutions
- The core standard barely addresses security
- Major providers with PHDs to spare are overall doing a reasonable job of build secure solutions
- Clients are at risk because they are likely to build less security implementations than providers
- Buckle up, read the threat model several times and follow it's many many many recommendations

Authentication: Where we’ve been

OAuth Terms

Client Registration

OAuth Grant Types

OAuth Threat Model

OAuth Countermeasures and Controls

It's been a pleasure.

jim@manicode.com