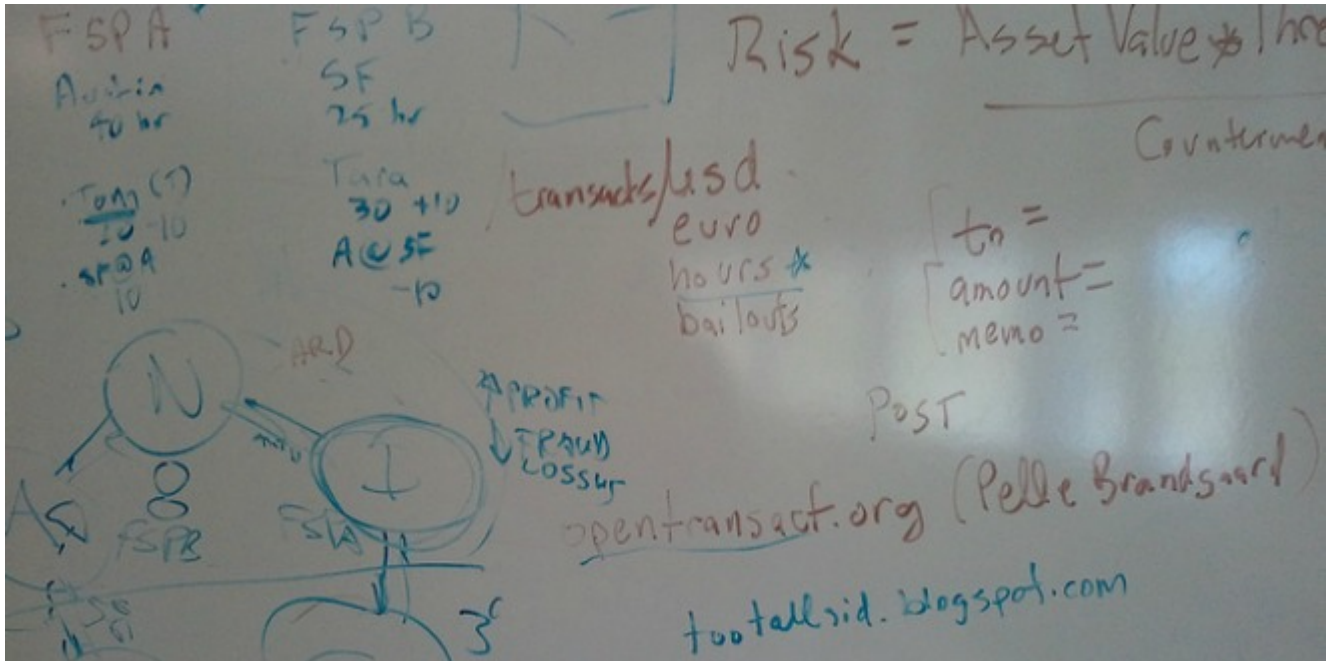


# WHEN THE ANSWER IS OAUTH, WHAT WAS THE QUESTION?



Behold: "Beautiful payments with OAuth." Image: tombrown91/Flickr

**OAUTH IS NEW.** OAuth is cool. OAuth also suffers from overinflated expectations — one can find related issues frequently. But OAuth is too important to leave it to propaganda. So let's get real.

## SYNOPSIS

OAuth (RFC 6749 and its predecessors) does care about 2 simple things — this is important to remember when one dives deeper:

- How to use a token to access resources?
- How to obtain such tokens?

## AUTHORIZATION

OAuth aims at establishing and enforcing authorization for resource accesses.

# HTTP

OAuth is confined to HTTP. It cannot help if other protocols are used to access resources.

# TOKEN ZOO

There is a variety of token types (bearer vs. proof tokens, self-contained vs. identifier tokens, refresh vs. access tokens), means of presenting tokens (HTTP Authorization request headers, URL query parameters, form-encoded body parameters) and obtaining them (various combinations of protocol exchanges – called flows). This is where the specification and framework complexity actually comes from. This text tries to avoid detailing the token zoo as far as possible.

# PARTIES IN GAME

OAuth addresses the following 3 parties:

- **Resource owner:** the owner of a resource e.g. the GMail user John Doe
- **Resource provider:** the provider of a resource e.g. Google
- **Resource consumer:** the (wannabee) consumer of a resource, aka client in OAuth

In particular, OAuth assumes resource owner and consumer to be distinct. OAuth defines how these parties interact in order to let a resource provider decide whether a request by a resource consumer is to be allowed or denied – backed by the resource owner's intent/consent.

# RESOURCE GRANULARITY

The resource granularity natively used with OAuth is **Web API access** e.g.:

May a distinguished resource consumer call the API published by Google's Calendar service for the GMail user John Doe

Extensions/refinements are needed to differentiate according particular API methods or specific (subsets of) resources managed through an API. For example, OpenID Connect and Kantara User-Managed Access define such extensions/refinements.

## RESOURCE PROVIDER ENDPOINTS

OAuth defines 3 resource provider endpoints and their corresponding exchanges:

- **Authorization endpoint:** used by resource consumers to obtain authorization from resource owners – depends on (classical) user authentication, provides OAuth grant objects e.g. authorization codes
- **Token endpoint:** used by resource consumers to obtain OAuth tokens – depends on (classical) client authentication and OAuth grant objects, provides OAuth tokens
- **Resource endpoint:** used by resource consumers to request resources – depends on client authentication by means such as HTTP Bearer or MAC authentication with OAuth tokens

The combination of authorization and token endpoints is known as OAuth **authorization server**.

## DEGREES OF FREEDOM

To identify the OAuth degrees of freedom read its exchanges in reverse order:

1 **Resource endpoint exchange:** 2-party request/response exchange between resource consumer and provider. There are little degrees of freedom in this exchange: by challenging for e.g. HTTP Bearer or MAC authentication the resource provider asks for an OAuth token. The resource consumer can either present a valid token (request granted) or not (denied).

2 **Token endpoint exchange:** 2-party token request/response exchange between resource consumer and provider. In this exchange there are more degrees of freedom: OAuth defines 4 flows (cf. below) to obtain a token and allows further ones to be added.

3 **Authorization endpoint exchange:** 3-party authorization request/response exchange among resource consumer, owner and provider. It is conducted via redirection and it is optional

## PROTOCOL FLOWS

OAuth combines these exchanges to following protocol flows:

- **Authorization code flow** conducting authorization/token/resource endpoint exchanges
- **Implicit flow** conducting authorization/resource endpoint exchanges: tokens are provided in response from authorization endpoints; no dedicated exchange with token endpoints
- **Client credentials flow** conducting token/resource endpoint exchanges: token endpoint callers perform (classical) client authentication
- **Resource owner password credentials flow** conducting token/resource endpoint exchanges: token endpoint callers utilize resource owners' credentials for authentication

## RESOURCE OWNER PRESENCE AND ATTENTION

OAuth assumes the individual resource owner to be present and attentive – unless the token endpoint can decide whether to supply an OAuth token in a token endpoint request from own, local context. There are 2 ways to establish such context:

- An earlier dialogue conducted between OAuth authorization endpoint and resource owner

- Some arrangement that is out-of-band to OAuth

So although OAuth separates the resource owner and consumer roles, these roles are still implicitly coupled with respect to presence/timeliness/responsiveness.

## FITNESS FOR ENTERPRISE IAM

Classical enterprise IAM (legal entity-owned resources) addresses authorization according a 2-party requestor/responder model which collapses resource providers/owners into responders:

- Such solutions do not address the OAuth use case – this is why OAuth was invented
- Vice versa: OAuth does not fit enterprise IAM use cases depending on such 2-party models

Even when a Cloud scenario splits resource provider and owner roles (e.g. SaaS subscribers as owners of resources served by SaaS providers) OAuth is not a natural solution. This is due to the implied constraints and refers to classical OAuth according its original use case: the authorization endpoint exchange confines OAuth to consumer IAM (individually-owned resources) since it assumes the presence of individual request owners when resource consumers make initial requests.

To utilize OAuth for enterprise IAM one needs avoid the authorization endpoint exchange:

- Use an out-of-band means to equip the OAuth token endpoint backend with context that matches the given business case as well as the given protocol constraints
- Use the client credentials flow: this flow allows to resemble the 2-party enterprise IAM model under an OAuth umbrella

## AUTHENTICATION AND SSO

OAuth produces noteworthy side-effects with respect to authentication and SSO:

- **HTTP request authentication:** OAuth extends the HTTP authentication framework to challenge resource consumers for OAuth tokens and to present them to resource providers. OAuth 2.0 defines the new HTTP authentication methods Bearer and MAC while OAuth 1.x defines OAuth.
- **Pseudo user authentication:** non-error responses from OAuth authorization endpoints allow resource consumers to conclude that requestors are authenticated at resource providers. OpenID Connect further exploits this basic property.
- **SSO:** equipping mobile apps with OAuth tokens allows them to access serves on behalf of a user. This allows transferring SSO state on client-side.

So, OAuth combines authorization, authentication and SSO in way that may result in a need to reiterate the “sorting-out matters in authn/SSO and authz“-work that was done in the enterprise IAM camp over the last decade.

## SUMMARY

OAuth does a trick that is mandatory in consumer IAM: it allows individual resource owners to delegate resource access rights to third-parties in a discretionary fashion with a limited scope based on a user dialogue.

In that respect, OAuth is unprecedented and fundamentally important. The original flavor of OAuth comprises a 3-party exchange requiring the presence of the individual resource owner which does confine OAuth to the consumer IAM space.

There are also other flavors of OAuth that better fit enterprise IAM. Modulo details the features facilitated by these other flavors do not deviate much from that in classical enterprise IAM solutions. So OAuth mainly contributes to consumer IAM. It may also be used in enterprise IAM but is rather no game-changer for that space.

So what was the question again? If the question was something like:

We run Web-based applications which host resources owned by our users/subscribers.

There are other Web-based applications (composite applications, mash-ups) that want to

offer value-added services and need to access their resources for that reason.

How can we make sure that resource owners are happy with that?

then exploring OAuth is a safe harbor. Note that identity data present a special type of resources.


If the question was anything (significantly) different then someone should throw a “Whoops – Are We on the Right Track” exception.

## APPENDIX: AUTHORIZATION CODE FLOW DIAGRAM

### APPENDIX: IMPLICIT FLOW DIAGRAM

### APPENDIX: CLIENT CREDENTIALS FLOW DIAGRAM

### APPENDIX: USER PASSWORD CREDENTIALS FLOW DIAGRAM

 *Oliver Pfaff, Senior Consultant IAM, Security and Cloud at iC Consult (www.ic-consult.com), the IAM Excellence company in the German-speaking region. He currently works with customers in automotive with respect to their strategies, architectures and services for consumer IAM. Leading-edge technologies esp. OAuth and OpenID Connect present one foundation of this work.*