**Differences Between OAuth 1 and 2**

**https://www.oauth.com/oauth2-servers/differences-between-oauth-1-2/**

OAuth 2.0 is a complete rewrite of OAuth 1.0 from the ground. OAuth 2.0 is not backwards compatible with OAuth 1.0 or 1.1, and should be thought is a completely new protocol.

OAuth 1.0 was largely based on two existing proprietary protocols: Flickr’s authorization API and Google’s AuthSub. OAuth 1.0 was the became best solution at that time. For few years of many companies use OAuth 1 APIs, and many developers writing code to consume the APIs. There are some areas were identified as needing improvement because they were either limiting the abilities of the APIs, or were too challenging to implement.

OAuth 2.0 represents years of discussions between a wide range of companies and individuals including Yahoo!, Facebook, Salesforce, Microsoft, Twitter, Deutsche Telekom, Intuit, Mozilla and Google

**Terminology and Roles**

Where OAuth 2.0 defines four roles, (client, authorization server, resource server, and resource owner,) OAuth 1 uses a different set of terms for these roles. The OAuth 2.0 “client” is known as the “consumer,” the “resource owner” is known simply as the “user,” and the “resource server” is known as the “service provider”. OAuth 1 also does not explicitly separate the roles of resource server and authorization server.

The terms “two-legged” and “three-legged” have been replaced by the idea of grant types, such as the [Client Credentials](https://www.oauth.com/oauth2-servers/access-tokens/client-credentials/) grant type and the [Authorization Code](https://www.oauth.com/oauth2-servers/access-tokens/authorization-code-request/) grant type.

**Authentication and Signatures**

The majority OAuth 1.0 implementation attempts were unsuccessful due to the cryptographic requirements of the protocol. The complexity of OAuth 1.0 signatures was a major pain point for anyone coming from the simplicity of username/password authentication.

Developers used to be able to quickly write Twitter scripts to do things by using just their username and password. With the move to OAuth 1.0, these developers were forced to find, install, and configure libraries in order to make requests to the Twitter API since it requires cryptographic signing of each request.

With the introduction of OAuth 2.0 Bearer tokens, it can quickly make API calls from a cURL command. The access token is used instead of a username and password.

**Performance at Scale**

As larger providers started using OAuth 1.0, the community realized that the protocol had several limitations that made it difficult to scale to large systems. OAuth 1.0 requires state management across different steps and often across different servers. It requires generating temporary credentials which are often discarded unused, and typically requires issuing long lasting credentials which are less secure and harder to manage.

In addition, OAuth 1.0 requires that the protected resources endpoints have access to the client credentials in order to validate the request. This breaks the typical architecture of most large providers in which a centralized authorization server is used for issuing credentials, and a separate server is used for handling API calls. Because OAuth 1.0 requires the use of the client credentials to verify the signatures, it makes this separation very hard.

OAuth 2.0 addresses this by using the client credentials only when the application obtains authorization from the user. After the credentials are used in the authorization step, only the resulting access token is used when making API calls. This means the API servers do not need to know about the client credentials since they can validate access tokens themselves.