OAuth

Services Oriented Architectures Security Project

Della Rossa Edoardo, Magoni Martina, Rendo Salvatore

OAuth 1.0

The first protocol

History

Blaine Cook, Chris Messina, Larry Halff and David Recordon

→ OpenID implementation with Twitter and Magnolia APIs

- April 2007: OAuth discussion group
- July 2007: initial specification
- October 3rd, 2007: **OAuth Core 1.0** final draft
- June 24th, 2009: **OAuth Core 1.0 Revision A**
- April 2010: **RFC 5849**, OAuth 1.0 Protocol
- October 2012: **RFC 6749**, OAuth 2.0 Authorization Framework
- Currently: **OAuth 2.1**

Model

Terminology:

- Client - Consumer
- Server - Service provider
- Resource Owner - User
- Protected Resource
- Client Credentials - Consumer Key and Secret
- Temporary Credentials - Request Token and Secret
- Token Credential - Access Token and Secret

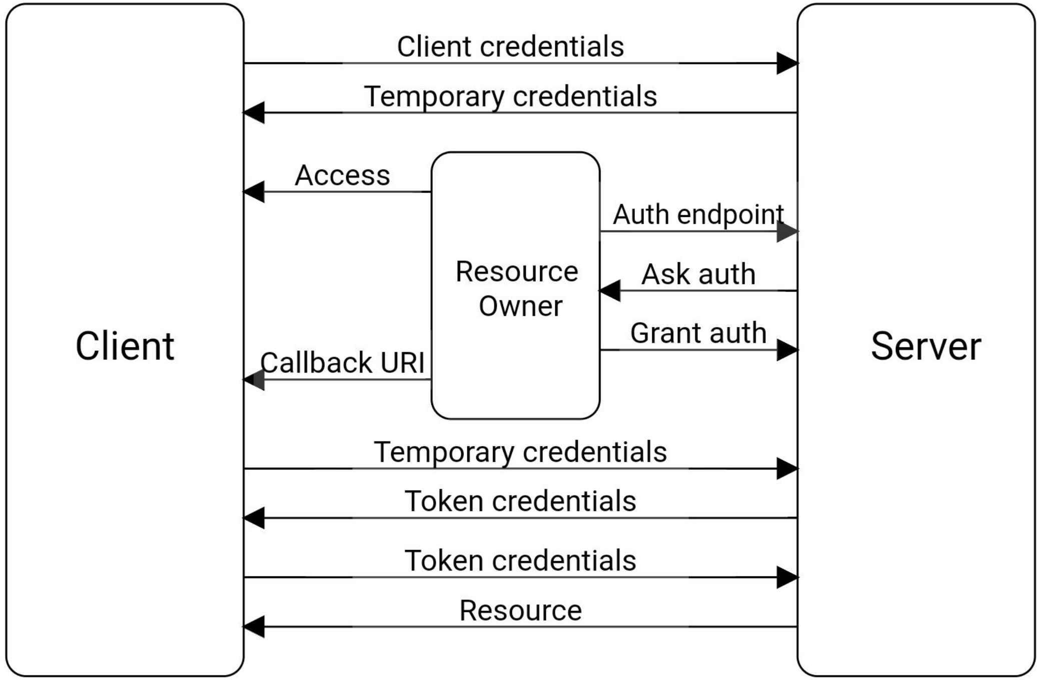
→ *“enabling delegated access to protected resources”*

Redirection-based user-agent process: end-user authorizes client access to his resources

+

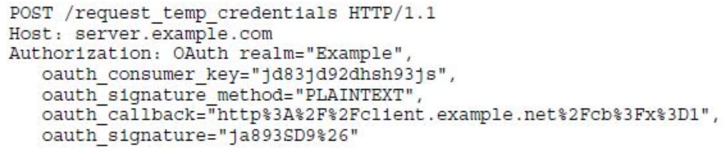
Authenticated Requests: request sent using credentials identifying the client making the request and the resource owner who authorized the request

Flow

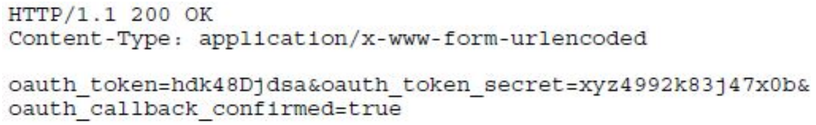

Client
Server
Client credentials
Temporary credentials
Access
Resource Owner
Auth endpoint
Ask auth
Grant auth
Callback URI
Temporary credentials
Token credentials
Token credentials
Resource

Redirection-Based Authorization

1. The client obtains a set of **temporary credentials** from the server



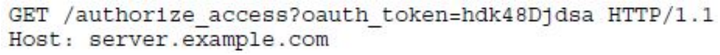
```
POST /request_temp_credentials HTTP/1.1
Host: server.example.com
Authorization: OAuth realm="Example",
   oauth_consumer_key="jd83jd92dhsh93js",
   oauth_signature_method="PLAINTEXT",
   oauth_callback="http%3A%2F%2Fclient.example.net%2Fcb%3Fx%3D1",
   oauth_signature="ja893SD9%26"
```



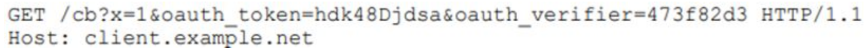
```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded

oauth_token=hdk48Djdsa&oauth_token_secret=xyz4992k83j47x0b&
oauth_callback_confirmed=true
```

2. The resource owner authorizes the server to **grant the client's access** request

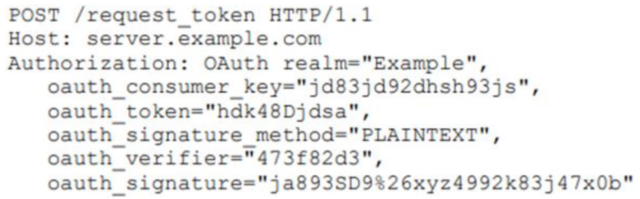


```
GET /authorize_access?oauth_token=hdk48Djdsa HTTP/1.1
Host: server.example.com
```

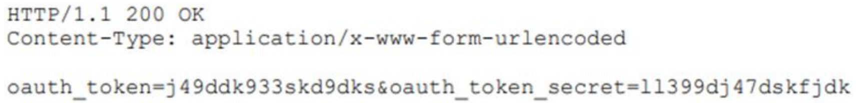


```
GET /cb?x=1&oauth_token=hdk48Djdsa&oauth_verifier=473f82d3 HTTP/1.1
Host: client.example.net
```

3. The client uses the temporary credentials to request a set of **token credentials** from the server



```
POST /request_token HTTP/1.1
Host: server.example.com
Authorization: OAuth realm="Example",
   oauth_consumer_key="jd83jd92dhsh93js",
   oauth_token="hdk48Djdsa",
   oauth_signature_method="PLAINTEXT",
   oauth_verifier="473f82d3",
   oauth_signature="ja893SD9%26xyz4992k83j47x0b"
```



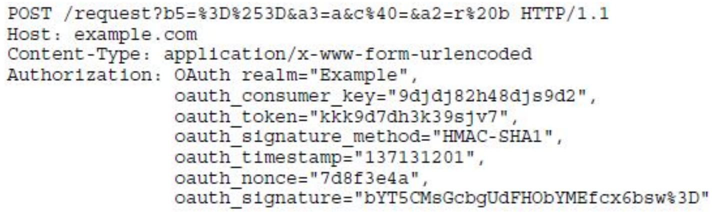
```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded

oauth_token=j49ddk933skd9dks&oauth_token_secret=ll399dj47dskfjdk
```

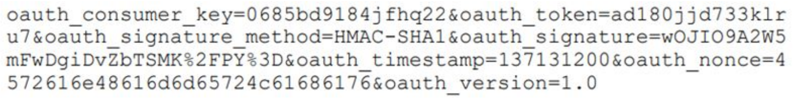
Authenticated Requests

Client makes authenticated requests

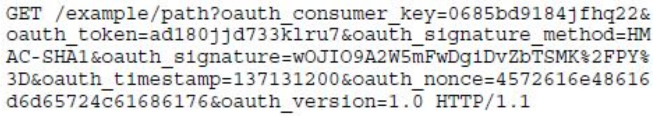
- Calculate the values of **protocol parameters**
- Add them to the **HTTP request** in the**:**
  - HTTP "Authorization" header field
  - HTTP request entity-body
  - HTTP request URI query
- Send the **authenticated request** to the server



```
POST /request?b5=%3D%253D&a3=a&c%40=&a2=r%20b HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Authorization: OAuth realm="Example",
               oauth_consumer_key="9djdj82h48djs9d2",
               oauth_token="kkk9d7dh3k39sjv7",
               oauth_signature_method="HMAC-SHA1",
               oauth_timestamp="137131201",
               oauth_nonce="7d8f3e4a",
               oauth_signature="bYT5CMsGcbgUdFHObYMEfcx6bsw%3D"
```



```
oauth_consumer_key=0685bd9184jfhq22&oauth_token=ad180jjd733klr
u7&oauth_signature_method=HMAC-SHA1&oauth_signature=wOJIO9A2W5
mFwDgiDvZbTSMK%2FPY%3D&oauth_timestamp=137131200&oauth_nonce=4
572616e48616d6d65724c61686176&oauth_version=1.0
```



```
GET /example/path?oauth_consumer_key=0685bd9184jfhq22&
oauth_token=ad180jjd733klru7&oauth_signature_method=HM
AC-SHA1&oauth_signature=wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%
3D&oauth_timestamp=137131200&oauth_nonce=4572616e48616
d6d65724c61686176&oauth_version=1.0 HTTP/1.1
```

Server verifies requests

- Recalculate the **request signature** (oauth_signature parameter)
- Ensure that the combination of **nonce + timestamp + token** has not been used before
- Verify the **scope and status** of the client authorization

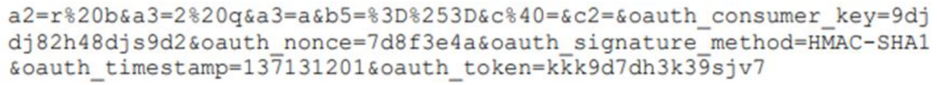
If the request fails verification:
400 - Bad Request or **401** - Unauthorized

Signature

Client can include two sets of credentials with each request to **identify itself and the RO**

The **shared-secret part** (or the client’s private key) is used to prove the **client’s rightful ownership**

HMAC-SHA1 and RSA -SHA1 use as an **input** a concatenation of HTTP request elements normalized in a string called **Signature base string**



```
a2=r%20b&a3=2%20q&a3=a&b5=%3D%253D&c%40=&c2=&oauth_consumer_key=9dj
dj82h48djs9d2&oauth_nonce=7d8f3e4a&oauth_signature_method=HMAC-SHA1
&oauth_timestamp=137131201&oauth_token=kkk9d7dh3k39sjv7
```

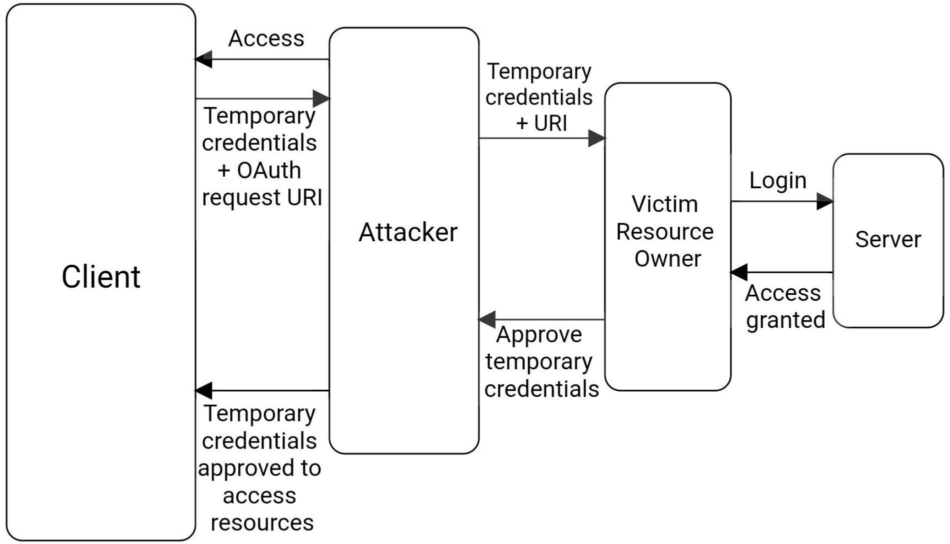
“oauth_**signature_method**”:

- **HMAC-SHA1**:
  digest = HMAC-SHA1 (key, text)
- **RSA-SHA1**:
  S = RSASSA-PKCS1-V1_5-SIGN (K, M)
- **PLAINTEXT**:
  client shared-secret encoded + "&" + token shared-secret encoded

Security Considerations: Possible attacks

Signature methods	RSA-SHA1 Signature Method SHA-1 Cryptographic Attacks Signature Base String Limitations
Requests	Confidentiality of Requests Scoping of Access Requests Entropy of Secrets
Servers	Spoofing by Counterfeit Servers Proxying and Caching of Authenticated Content Plaintext Storage of Credentials Denial-of-Service / Resource-Exhaustion Attacks Automatic Processing of Repeat Authorizations
Clients	Secrecy of the Client Credentials
Resource owner	Cross-Site Request Forgery (CSRF) User Interface Redress Phishing Attacks

OAuth Security Advisory: 2009.1


Client
Access
Temporary credentials + OAuth request URI
Attacker
Temporary credentials + URI
Victim Resource Owner
Login
Server
Access granted
Approve temporary credentials
Temporary credentials approved to access resources

The attacker could use the **knowledge of the approved temporary credentials** to **masquerade as a legitimate** user via the client application to the server application

PROBLEM:
lack of a mechanism to ensure that the party that **started the authorization flow** with the client is the same as the one that **authorized it** with the server

List of OAuth providers using OAuth 1.0

Dropbox	1.0 - 2.0
Etsy	1.0
Evernote	1.0a
Flickr	1.0a
MySpace	1.0a
Netflix	1.0a
Tumblr	1.0a
Twitter	1.0a - 2.0
Wordpress	1.0a
Yahoo!	1.0a - 2.0

OAuth 2.0

Differences and advantages over OAuth 1.0

OAuth 2.0 Key Concepts and workflow

1. **Clients:**

   OAuth 2.0 introduces the concept of clients, which can be applications or services requesting access to user resources. Clients can be web-based, mobile, or desktop applications.

2. **Resource Owner:**

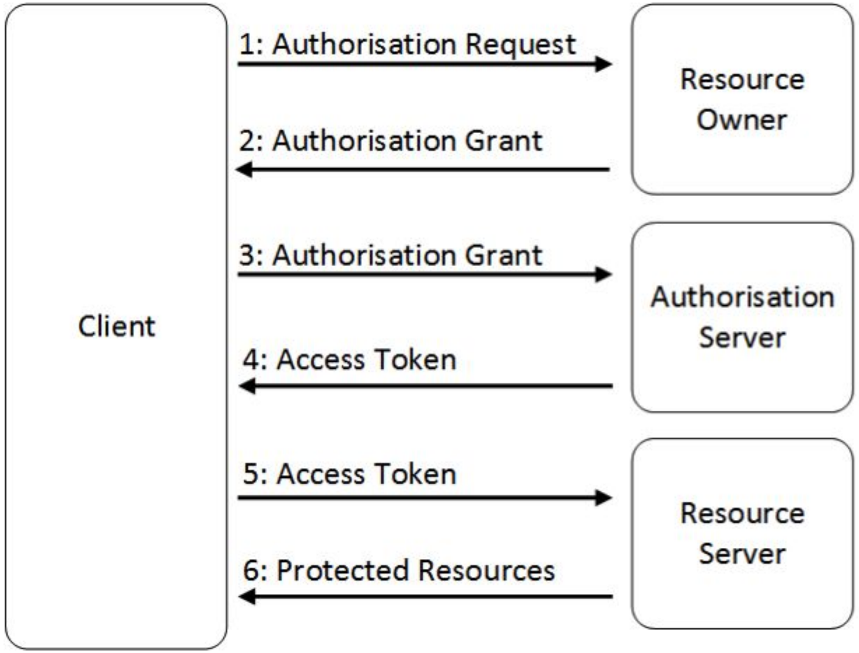
   The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

3. **Authorization Server:**

   The Authorization Server is responsible for authenticating users and issuing access tokens to authorized clients.

4. **Resource Server:**

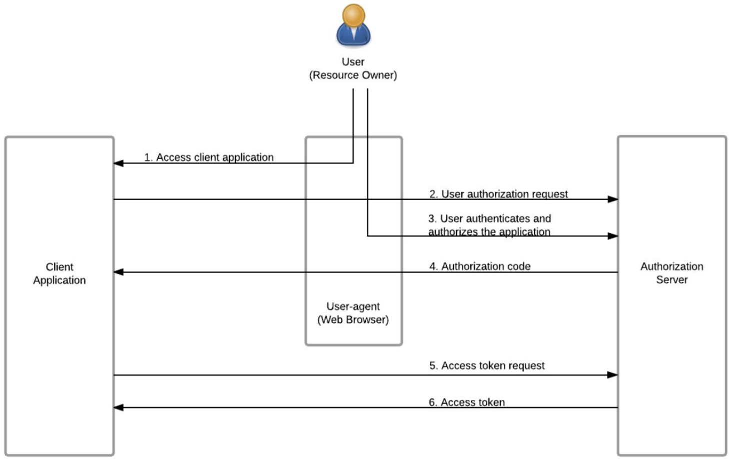
   An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.



Grant types

Authorization Code Grant

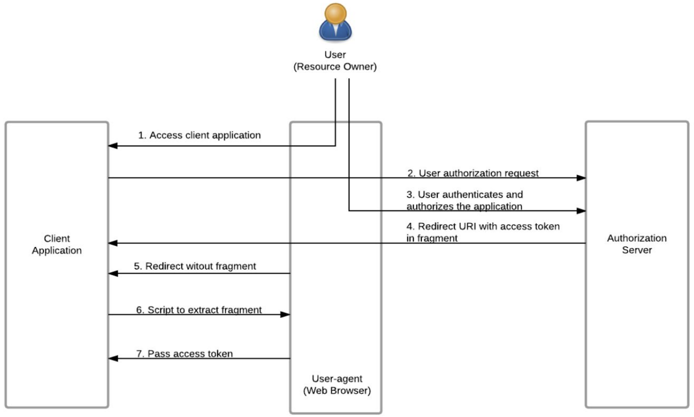
A single-use code that the authorization server returns to the client to be exchanged for an access token.



Implicit Grant

Simplified authorization code flow optimized for clients implemented in a browser using a scripting language such as JavaScript. Instead of issuing the client an authorization code, the client is issued an access token directly.

The access token is encoded into the redirect URI therefore should be treated as public knowledge and must have very limited permissions.

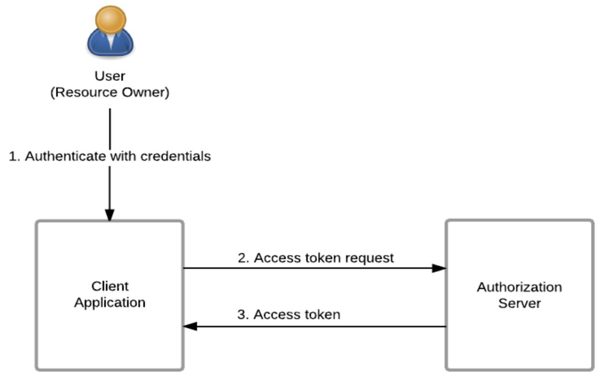

User
(Resource Owner)
1. Access client application
2. User authorization request
3. User authenticates and authorizes the application
4. Redirect URI with access token in fragment
5. Redirect witout fragment
6. Script to extract fragment
7. Pass access token
Client Application
User-agent (Web Browser)
Authorization Server

Grant types

Resource Owner Password Credentials Grant

The client itself asks the user for the resource owner's username and password. The client will then send these credentials to the authorization server along with the client's own credentials.

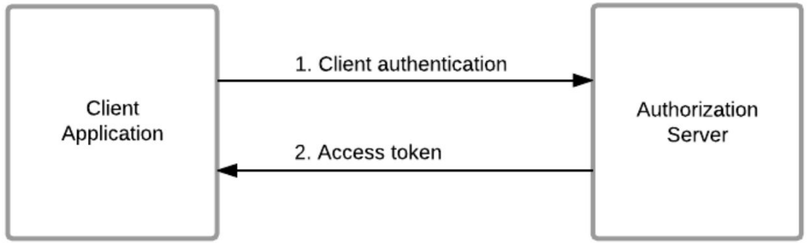
Should only be used when there is a high degree of trust between the resource owner and the client.



Client Credentials Grant

The client can request an access token using only its client credentials with this grant type.

It's used when the client is acting on its own behalf (the client is also the resource owner) or is requesting access to protected resources based on an authorization previously arranged with the authorization server.



Access Token and Refresh Token

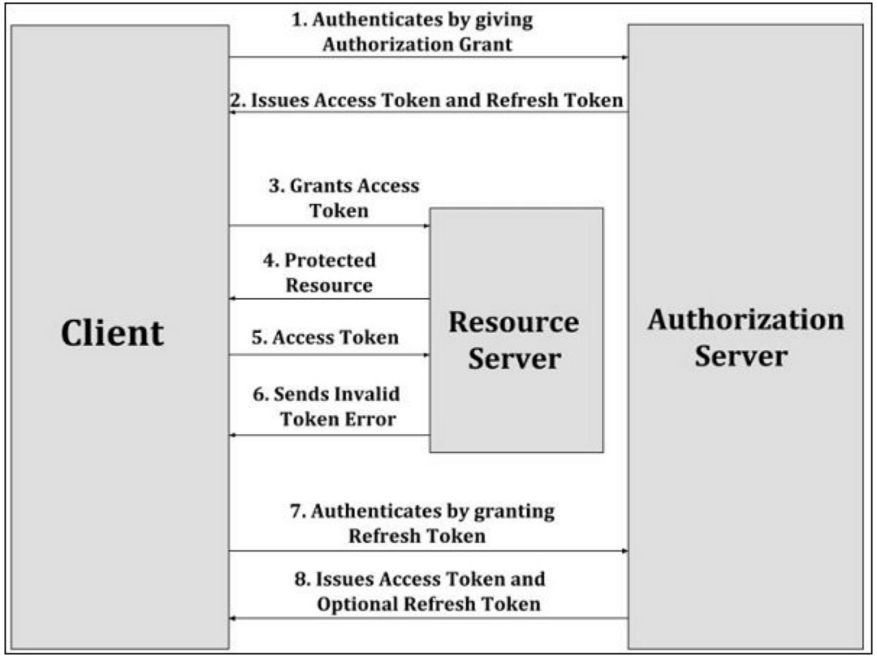
Access tokens are credentials used to access protected resources.

It is a string representing an authorization issued to the client, usually opaque to him.

Access Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server.

Refresh tokens are credentials used to obtain access tokens.

Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope.


1. Authenticates by giving Authorization Grant
2. Issues Access Token and Refresh Token
3. Grants Access Token
4. Protected Resource
5. Access Token
6. Sends Invalid Token Error
7. Authenticates by granting Refresh Token
8. Issues Access Token and Optional Refresh Token
Client
Resource Server
Authorization Server

Implementation & Possible attacks

Crucial points for a secure OAuth 2.0 implementation are:

- **Client Authentication**
- **Access token and refresh token:**
  - Storing
  - Exchange
  - Expiration
- **Authorization Code**
- **Requests Confidentiality**
- **Ensuring Endpoint Authenticity**

The lack of a secure implementation of OAuth 2.0 could lead to attacks like:

- **Client Impersonation**
- **Token Eavesdropping**
- **Redirection URI Manipulation**
- **Credentials-Guessing Attacks**
- **Cross-Site Request Forgery (CSRF)**

Fixing OAuth 1.0 with Version 2.0

OAuth 1.0

The main problems about version 1.0 were:

1. **No clear separation of duties**
2. **Managing of signatures, timestamps and nonce**
3. **Managing credential storing and the exchange of them**
4. **Complex design**

OAuth 2.0

The solution to OAuth 1.0 issues are solved by version 2.0 with:

1. **Clear separation of duties**
2. **Token based approach**
3. **Different types of authorization grant**
4. **Simplified workflow and Flexible Design**

OAuth 2.0 vs 1.0: Simplicity and workflow

OAuth 1.0:

1. **Request Signing:** In Version 1.0 request signing involves the creation of a signature for each request. It can be a daunting task for developers, adding additional layers of complexity that must be implemented correctly to ensure security.
2. **Complex Signatures:** Version 1.0 relies on cryptographic signatures for request authentication, requiring the various credentials and parameters in each request, making it complex and error-prone to implement.
3. **Multiple Credentials:** Developers need to manage multiple sets of credentials. The handling of these credentials correctly adds to the complexity of version 1.0.
4. **Nonce and Timestamp:** Version 1.0 introduces the requirement for a nonce and a timestamp in requests, preventing replay attacks. Managing these values can be confusing.

OAuth 2.0:

1. **Token-Based:** Version 2.0 simplifies the entire authentication and authorization process by relying on tokens. The absence of cryptographic signatures for every request makes the overall implementation less error-prone and more simple.
2. **Clear Separation of Duty:** Version 2.0 separates the roles of the entities inside the protocol. The separation between authorization server, client, and resource server makes it easier to understand the protocol's workflow, reducing developer's confusion during implementation.
3. **Flexibility of Design:** Version 2.0 is designed to be adaptable to various use cases and platforms, allowing developers to choose from different grant types, based on their application's needs.
4. **Widely Used:** Version 2.0 has become a widely adopted standard with well-established libraries and tools available for various programming languages and platforms. This ecosystem of resources further simplifies implementation.

OAuth 2.0 vs 1.0: Scalability and performance

OAuth 1.0:

1. **Request Signing Overhead:** Version 1.0 involves the generation of cryptographic signatures for each request, which adds computational overhead to both the client and the server, impacting the scalability of the system.
2. **Timestamp and Nonce Checks:** The server needs to check and validate Nonces and timestamps values for each incoming request, which can consume processing resources.
3. **Resource Server Complexity:** Version 1.0 introduces complexity at the resource server, which must verify request signatures and needs to manage token secrets for both request and access tokens, affecting heavily the performance and scalability of the resource server.

OAuth 2.0:

1. **Simplified Workflow:** The simplified workflow can lead to more efficient interactions between the protocol's entities, improving the overall performance.
2. **Reduced Processing:** With Version 2.0's focus on tokens and simplified authentication and authorization processes, there is less need for extensive, and computational expensive, nonce and timestamp checks.
3. **Caching Opportunities:** Resource servers can cache access tokens, reducing the need for repeated authorization and token retrieval, especially in scenarios with a high volume of similar requests.

OAuth 2.0 vs 1.0: Web integration and mobile

OAuth 1.0:

1. **Complexity for Web Integration:** OAuth 1.0's reliance on cryptographic signatures can be complex and tedious for web applications.
2. **Token Storage Challenges:** in Version 1.0 it is mandatory to store access tokens and secrets securely on the client side, presenting security and management challenges, particularly in web environments.
3. **Limited Mobile Support:** The complexity and secret management requirements makes it less suitable for non-browser based application clients and less mobile-friendly, especially for resource-constrained mobile devices

OAuth 2.0:

1. **Simplified Integration both for Web and Mobile Integrations.**
2. **Mobile-Friendly:** OAuth 2.0 excels in mobile application integration through secure flows like authorization code and implicit flows. Native SDKs and libraries simplify the mobile integration, and its token-based approach enhances mobile performance, even on resource-constrained devices.
3. **Unified Compatibility:**Version 2.0 provides a unified framework for both web and mobile environments, simplifying scalability across various user interfaces.

OAuth 2.1

Differences and advantages over OAuth 2.0

Fixing OAuth 2.0 vulnerabilities

draft-ietf-oauth-v2-1-09

Published: 10 July 2023

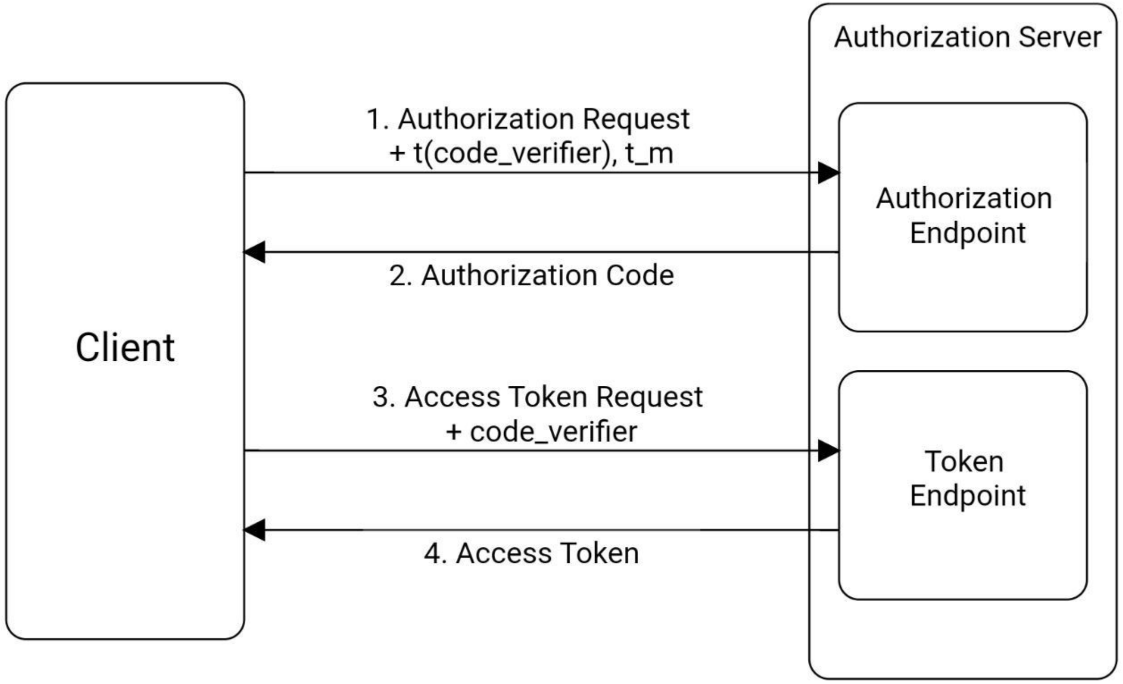
Expires: 11 January 2024

Draft created by IETF to update and fix OAuth 2.0

6 major changes:

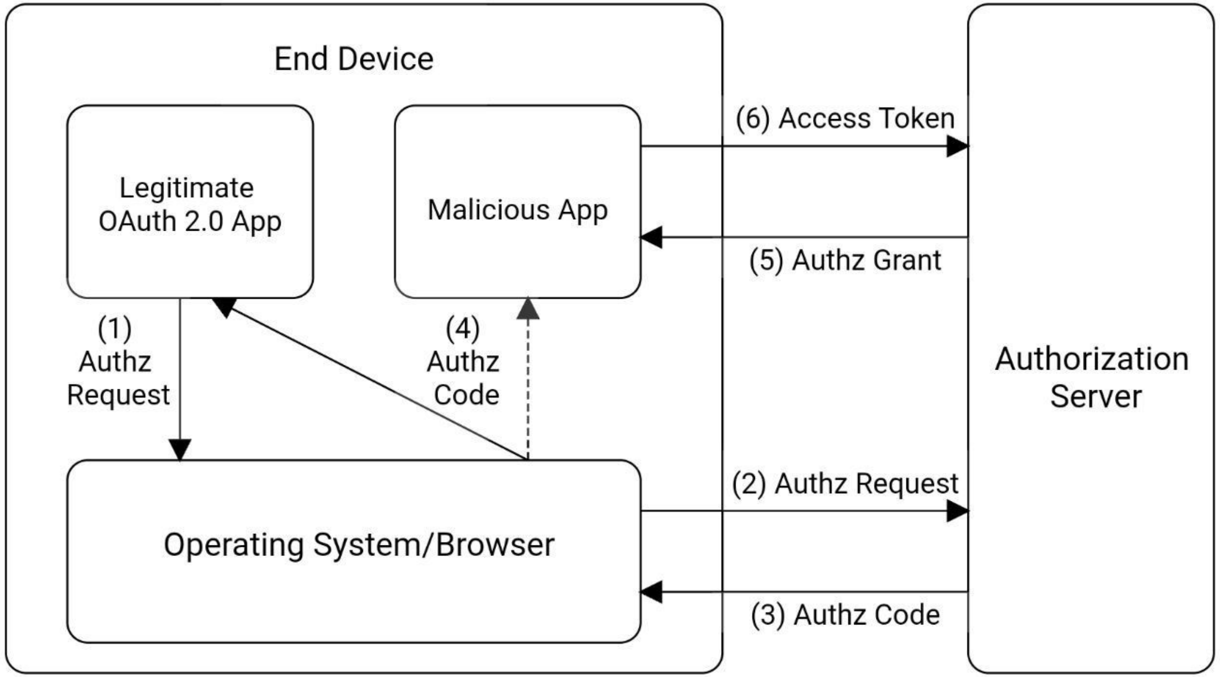
- PKCE Implementation (RFC 7636)
- Exact string matching on redirection URI
- Implicit grant removal
- Resource Owner Password Credentials grant removal
- No more bearer tokens in query strings
- Limiting refresh token to one-time use or sender-constrained

PKCE Implementation (RFC 7636)



Fonte: RFC7636

PKCE - Authz Code interception attack



Exact URI string matching

Wildcards in redirect URI validation are more complex to manage and more error prone.

```
https://sample-application-*.myapp.com/oauth-callback
https://sample-application-v1.myapp.com/oauth-callback?*
```

Security problems occur when the attacker manage to redirect the traffic to malicious host

Implicit grant removal

Access token is sent via Authorization Response URL fragment, it is stored in browser history and accessible to the JavaScript code running on the page.

Redirect URI example

```
https://example-app.com/redirect
  #access_token=g0ZGZmNj4mOWIjNTk2Pw1Tk4ZTYyZGI3
  &token_type=Bearer
  &expires_in=600
  &state=xcoVv98y2kd44vuqwye3kcq
```

Clients should use response_type=code instead of response_type=token (Implicit grant).

Resource Owner Password Credentials grant removal

- Implemented to simplify migrating to OAuth.
- User credentials are forwarded to the Authorization Server, and user explicit authorization to AS is avoided.
- Does not allow to use Two-Factor Authentication
- The AS has full access to user credentials, that is what OAuth is designed to avoid.

Bearer Tokens in query strings

Token usage in query string is not allowed, to avoid token leakage via browser history, or other attacks using URL reading (JavaScript malicious elements)

```
provider.com/get_user_profile?access_token=abcdef
```

Token usage is still allowed in TLS protected information (HTTP header or POST body).

Limiting refresh token - More security measures

Refresh token demand more security measures, because they can be used to gain infinite access to the resource.
OAuth 2.1 draft provides two possibilities:

One time use refresh token

The client has to store a new refresh token every refresh

Cryptographically binded token

Refresh token mechanism uses a cryptographic binding that links the token to the client.

This bind is checked at every refresh.

References

OAuth 1.0

- OAuth Core Workgroup, "*OAuth Core 1.0*"
- OAuth Core Workgroup, "*OAuth Core 1.0 Revision A*"
- OAuth Core Workgroup, "*OAuth Security Advisory 2009.1*"
- Internet Engineering Task Force, "*RFC 5849: The OAuth 1.0 Protocol*"
- *Introduce OAuth 1.0 - Authlib 1.2.1 documentation*
- *OAuth - Wikipedia*
- *Why OAuth 1.0a?*
- *List of OAuth providers - Wikipedia*

OAuth 2.0

- OAuth 2.0
- RFC 6749. OAuth 2.0
- Differences Between OAuth 1 and 2 - OAuth 2.0 Simplified
- OAuth 2.0 Grant Types - WSO2 Identity Server Documentation

OAuth 2.1

- Internet Engineering Task Force, "*The OAuth 2.1 Authorization Framework*", Internet-Draft n. *draft-ietf-oauth-v2-1-09*