OAuth Login for Security.

[**Definitions**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#def-sec)

There are three roles involved in the OAuth process. These roles are not mutually exclusive; a single party may act in more than one role.

**Resource Owner (RO):**

The entity that has authorization at the beginning of the OAuth process, who delegates authorization to the Client

**Client:**

The entity to which authorization is delegated by the Resource Owner

**Server:**

The entity that enforces the authorizations in question (e.g., the server that stores a protected resource)

An OAuth Server issues three types of opaque tokens in the course of the authorization process. These values are random values that are unique within a defined scope.

**Request token:**

A temporary identifier for a specific Client for the duration of a single OAuth authorization transaction. An RO that requests a verification token using that request token indicates that the corresponding Client is the entity to be authorized.

**Verification token:**

A temporary identifier for an authorization, including the authorized Client, the authorizing RO, and the resources to which access is to be granted. A Client that requests an access token using a given verification token indicates that the RO has granted the corresponding authorization.

**Access token:**

A long-lived identifier for an authorization. A Client that requests access to resources using a given access token indicates that it has been authorized via the OAuth process to access those resources.

Many of the security properties of the OAuth process require protocol participants to associate, or "bind", certain other data to tokens, then subsequently verify these associations/bindings. In the below, when an entity is required to use the value "bound to a token" or "associated with a token", it is implied that the value in question is the one associated by the entity at hand.

[**Requirements and design goals**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#req-sec)

The fundamental goal of the OAuth model is to allow the Resource Owner to grant the Client authorization to access selected resources on the Server, while maintaining strict authentication and access control. The requirement for strict authentication means that no party should be able to authenticate as any other, whether a protocol participant (e.g., the Client authenticating as the RO) or a third party. The requirement for access control means that the delegation process should not grant the Client (or any other party) access to anything other than the resources designated for access by the Resource Owner.

The OAuth model must also ensure that the authorization process is followed: In order for an authorization to be delegated by a Resource Owner to a Client, it must be requested by the Client and authorized by the Resource Owner. In particular, the RO cannot unilaterally grant authorization to a Client without a request (since the Client may not be able to use that authorization), and of course, no authorization can be granted without the approval of the RO. In addition, the security model should prevent the Client from further delegating access without going through the same OAuth process.

It should be clear that these requirements already rule out the simplest models for delegating authorization. For example, a model in which the RO simply provides its authentication credentials to the Client is unacceptable for several reasons: It allows the Client to access all resources (not only any selected subset), and it requires the Client to authenticate as the RO.

Note also that this document defines only the security model for OAuth, i.e., how security-relevant data flow through the system, and what the security requirements are for these flows. In particular, this document does not define protocol mechanisms for carrying data (e.g., tokens) or providing security features (e.g., authentication, confidentiality). Such mechanisms should be defined in protocol-specific documents (e.g., a realization of OAuth in HTTP or XMPP).

[**Authorization flow**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#authz-flow-sec)

The OAuth authorization process is the process by which a Resource Owner can grant a Client permission to access resources protected by a Server. This process occurs in four high-level steps, illustrated in [Figure 1](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#proc-flow-fig):

1. The Client obtains a Request Token from the Server
2. The Client requests authorization from the Resource Owner
3. The Resource Owner validates the request and grants authorization
4. The Client exchanges the Request Token for an Access Token

If the process fails at any point, no authorization is granted.

Client Server Resource Owner

| | |

| Issue Req. Token | |

|<---------------------->| |

| | |

| | Request authorization |

|------------------------------------------------>|

| | |

| | Issue Ver. Token |

| |<---------------------->|

| | |

| Grant authorization | |

|<------------------------------------------------|

| | |

| Issue Acc. Token | |

|<---------------------->| |

| | |

~ ~ ~

| Access Resources | |

|<---------------------->| |

| | |

[**General Requirements and Assumptions**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#req-assm-sec)

There are a number of authentication relationships that need to be in place before OAuth can be used to delegate authorization. Client authentication is critical for preventing unauthorized parties from gaining access: The Client MUST be able to authenticate to both the Server and the Resource Owner. Moreover, the Client MUST be able to authenticate the same identifier to both parties (e.g., using the same certificate in the context of HTTPS), so that the Server and the Resource Owner can verify that the Client that the RO authorizes is the same one that the Server will enable to access the protected resources.

Likewise the Resource Owner MUST be able to authenticate both to Server and to the Client, and it MUST be able to authenticate the same identifier to both (it may have other identifiers with each). The Server MUST be able to authenticate both to the Client and the RO, but it is not required to use the same identity for both.

All messages in this process SHOULD be integrity- and confidentiality-protected; ones that MUST be so protected are noted below. The security of the authorization process relies only on the authentication relationships above, but access to some parameters passed between OAuth entities could allow third parties to disrupt the OAuth process.

[**Request token issuance**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#req-t-iss-sec)

Client Server Resource Owner

| | |

| req-token-request | |

1. |----------------------->| |

| | |

| req-token-issue | |

| (req-t) | |

2. |<-----------------------| |

| | |

The OAUth process is initiated by a Client that wishes to access a given set of resources on the Server. The authorization flow begins when the Client sends a request for a Request Token to the Server. In response to this request, the Server MUST authenticate the Client. If authentication is unsuccessful, the process fails. If authentication succeeds, the Server MUST generate a random request token and store a binding between the request token and authenticated identity of the Client (denoted Cl-ID below). During this transaction, the Client and the Server MAY negotiate the resources to which access is to be granted.

Upon receiving a request token from the Server, the Client MUST authenticate the Server and store a binding between the Server's authenticated identity and the request token.

The Request Token identifies a Client's request for authorization to access resources on the Server. At issuance, the token is bound to a Client identity and a Server identity. To uniquely identify an authorization transaction, the Server MUST ensure that each Request Token is unique within the set of such tokens that it issues. Token values SHOULD be difficult for an outside entity to guess. In order to avoid resource exhaustion at the Server, it is RECOMMENDED that Request Tokens have a limited lifetime.

[**Authorization of the request**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#authz-req-sec)

Before the Client can access protected resources, it must be authorized to access those resources: The Client must request authorization, the Resource Owner must notify the Server that it intends to grant access, and then the Resource Owner must actually grant access to the Client. In addition, before the Resource Owner grants access, it must assure that the Client's request is authentic by validating it with the Server.

[**Request for authorization**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#req-authz-sec)

Client Server Resource Owner

| | |

| | request-authz |

| | (req-t) |

3. |------------------------------------------------>|

| | |

The Client requests authorization to access resources by sending a message to the Resource Owner that contains an unused request token. After sending a request token in an authorization request, the Client MUST mark that request token as used, and MUST NOT use the request token in any subsequent authorization request.

If the Client wishes to verify that the RO that it interacts with on this step is the same entity that it interacts with later, then it MUST authenticate the RO and store a binding between the request token and the RO's authenticated identity (in addition to the binding to the Server's identity recorded earlier). It should be noted however, that these steps are purely optional. They provide no additional security guarantees with respect to the authorizations granted; they only allow the Client to track which request tokens have been used. Note that this transaction does not require the RO to authenticate the Client. The Client will be authenticated in a later step, after the RO is told by the Server which Client identity is bound to the request token

.

[**Verification token issuance**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#ver-t-iss-sec)

Client Server Resource Owner

| | |

| | ver-token-request |

| | (req-t) |

4. | |<-----------------------|

| | |

| | ver-token-issue |

| | (ver-t,cl-id,scope) |

5. | |----------------------->|

| | |

The Resource Owner validates the request for authorization and obtains the ability to grant authorization by requesting a verification token from the Server. This request MUST include the request token provided by the Client. Before sending this request, the RO SHOULD authenticate the Server, since a false Server can disrupt the authorization process (but not cause unintended authorization to be granted).

When the Server receives a request for a verification token, it MUST take the following actions, in order:

1. Authenticate the Resource Owner and verify that the RO owns the resources subject to authorization
2. Verify that the request token has not been used in a prior request. If so, the process fails; if not, the request token is marked as used.
3. Generate a random verification token that is unique in the scope of such tokens
4. Store a binding between the verification token, the Client identity bound to the request token, and the RO's authenticated identity
5. Establish an integrity- and confidentiality-protected channel to the RO (if one does not already exist, e.g., the channel used the request)
6. Send a message to the RO containing the following information:
   * The verification token
   * The Client identity bound to the request token
   * The scope of the authorization to be provided (i.e., the set of resources)

The Server and the RO MAY also perform further negotiation of the scope of the authorization at this point.

It should be emphasized that the Server MUST NOT transmit the verification token over any channel that is not authenticated, integrity-protected, and confidentiality-protected, since the verification token is highly sensitive information: Any third party that intercepts the verification token is capable of granting the Client access to the protected resources, and any party that can inject or modify a verification token can cause the RO to grant unintended accesses.

Upon receiving the Server's message (authenticating the Server again, if necessary), the RO stores a binding between the verification token and the Client identity provided by the Server.

Some might note that there is an asymmetry to the security requirements for the two halves of this transaction: The request for a verification token is only authenticated, while the response also needs to be confidentiality- and integrity-protected. This is because the injection of a false request token can only cause the process to fail (since the Client identity provided by the server will be wrong), while leakage of the verification token can cause used to grant an unintended authorization. For simplicity, it is RECOMMENDED that the same channel (with full protections) be used for the request and the response.

Note that at this point in the process, no authorization has been granted. The Server has only issued the Resource Owner a verification token so that the RO can grant authorization if desired. The Server MUST NOT provide the Client access to protected resources until the RO has provided the verification token to the Client, and Client has exchanged the verification token for an access token, as described below.

[**Notification of authorization**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#not-authz-sec)

Client Server Resource Owner

| | |

| authz-notify | |

| (ver-t,req-t) | |

6. |<------------------------------------------------|

| | |

If the Resource Owner decides to authorize the Client to access protected resources (the set bound to the verification token), then it signals this by sending a message to the Client containing the verification token. Before sending such a message, the RO MUST authenticate the Client and establish an integrity- and confidentiality-protected (i.e., encrypted) channel to the authenticated Client. If the authenticated identity of the Client does not match the Client identity bound to the verification token, or if the authentication fails, then the process fails and the RO MUST NOT send the verification token to the Client. As above, the RO MUST NOT transmit the verification token over any channel that is not authenticated, integrity-protected, and confidentiality-protected.

Upon receipt of an authorization notification message, the Client MUST authenticate the RO and store a binding between the RO's authenticated identity and the verification token. The Client MUST also store a binding between the verification token and the Server identity bound to the request token. The Client MAY verify that the RO's authenticated identity matches the one bound earlier to the request token.

[**Access token issuance**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#acc-t-iss-sec)

Client Server Resource Owner

| | |

| acc-token-request | |

| (ver-t) | |

7. |----------------------->| |

| | |

| acc-token-issue | |

| (acc-t,ro-id) | |

8. |<-----------------------| |

| | |

The Client obtains access to the designated resources by exchanging a verification token that it has received for an access token. To request an access token, the Client sends the verification token to the Server. Before sending this message, the Client MUST authenticate the Server and verify that the authenticated Server identity is the one bound to the verification token (i.e., the Server identity that the Client originally bound to the request token).

Upon receiving a request for an access token, the Server MUST authenticate the Client and verify (1) that the verification token provided is one that it issued, and (2) that the Client's authenticated identity matches the identity bound to the verification token. If both criteria are satisfied, then the Server MUST generate a random access token (unique within the scope of such tokens) and store a binding between the access token, the Client's authenticated identity, the RO identity bound to the verification token, and the resources bound to the verification token.

The Server completes the authorization process by sending to the Client a message containing (1) the access token and (2) an identifier for the RO to which the verification token was issued. The Client MUST authenticate the Server prior to accepting such a message and verify that the Server's identity matches that bound to the request token and verification token. The Client MUST also verify that the RO identity returned by the Server matches the RO identity bound to the verification token. If both verifications succeed, then the Client stores a binding between the access token and the protected resources.Note that because the Client received the verification token over a confidentiality-protected channel, the Server's acceptance of the verification token provides the Client assurance that the entity that provided the verification token was authorized (at the Server) to grant the specified authorization.

[**Accessing protected resources**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#acc-res-sec)

Client Server Resource Owner

| | |

| acc-res-request | |

| (acc-t) | |

9. |----------------------->| |

| | |

| resources | |

|<-----------------------| |

| | |

Finally, once a Client is authorized to access resources on the Server, it accesses them using the corresponding access token. When a Client wishes to access resources, it sends a request to the Server containing the access token. The Server MUST authenticate the Client and verify that the access token presented belongs to the authenticated Client before responding with the requested resources.

[**Summary**](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#authz-sum-sec)

All of the interactions above are summarized in [Figure 4](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#msg-flow-fig), below. [Figure 2](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#req-tok-life-fig) and [Figure 3](https://tools.ietf.org/id/draft-barnes-oauth-model-01.html#ver-tok-life-fig) illustrate the life-cycles of the request token and the verification token, respectively.

1. Issues token +--------+ +--------+ 2. Uses token

4. Provides info | Server |------------->| Client | request authz

to validate +--------+ +--------+

(Client ID) | ^ |

| | +----------+ |

| +-----| Resource |<----+

+------>| Owner |

+----------+

3. Uses token to request

verification token

**Figure 2: Life cycle of a request token**

1. Issues token +--------+ +--------+ 2. Uses token

4. Provides info | Server |<-------------| Client | request

to validate +--------+------------->+--------+ access tok.

(RO ID) | ^

| +----------+ |

+---->| Resource |-----+

| Owner |

+----------+

3. Uses token to

grant authz

**Figure 3: Life cycle of a verification token**

Client Server Resource Owner

| | |

| req-token-request | |

1. |----------------------->| |

| | |

| req-token-issue | |

| (req-t) | |

2. |<-----------------------| |

| | |

| | request-authz |

| | (req-t) |

3. |------------------------------------------------>|

| | |

| | ver-token-request |

| | (req-t) |

4. | |<-----------------------|

| | |

| | ver-token-issue |

| | (ver-t,cl-id,scope) |

5. | |----------------------->|

| | |

| authz-notify | |

| (ver-t,req-t) | |

6. |<------------------------------------------------|

| | |

| acc-token-request | |

| (ver-t) | |

7. |----------------------->| |

| | |

| acc-token-issue | |

| (acc-t,ro-id) | |

8. |<-----------------------| |

| | |

~ ~ ~

| acc-res-request | |

| (acc-t) | |

9. |----------------------->| |

| | |

| resources | |

|<-----------------------| |

| | |

**Figure 4: OAuth message flow**