Oauth Messages

Access URLs - Example

- Temporary Credential Request URI (Request Token URL)
  - https://photos.example.net/initiate
- Resource Owner Authorization URI (Authorize URL)
  - https://photos.example.net/authorize
- Token Request URI (Access token URL)
  - https://photos.example.net/token

Redirection-Based Authorization

- Visit Photo printing site
  - printer.example.com
- Request some photos to be printed and mailed to someone.
- Specify that photos are stored at
  - photos.example.net (server)
- User - paul

Redirection-Based Authorization

1. printer.example.com has previously registered with photos.example.net and obtained client credentials.
2. printer.example.com requests temporary credentials from photos.example.net and obtains these.
3. Customer redirected to photos.example.net.
4. Logon to photos.example.net.

Redirection-Based Authorization

5. Message that some printing service has request access to your photos. Do you want to allow such access. Yes.
6. Temporary credentials of printer.example.com updated to resource-owner-authorized.
7. Customer redirected back to Printing site.
8. printer.example.com obtains token credentials.
9. printer.example.com downloads the photos from photos.example.net and you are prompted which ones to print.

2. Client asks for Request Credentials

```
POST /initiate HTTP/1.1
    Host: photos.example.net
    Authorization: OAuth realm="Photos",
      oauth_consumer_key="dpf43f3p2l4k3l03",
      oauth_signature_method="HMAC-SHA1",
      oauth_timestamp="137131200",
      oauth_nonce="wIjqoS",
      oauth_callback="http%3A%2F%2Fprinter.example.com%2Fready",
      oauth_signature="74KNZJeDHnMBp0EMJ9ZHt%2FXKycU%3D"
```

- Notice that the client specifies a callback for this request.

2. Server Responds

```
HTTP/1.1 200 OK
    Content-Type: application/x-www-form-urlencoded
    oauth_token=hh5s93j4hdidpola&oauth_token_secret=hdhd0244k9j7ao03&
    oauth_callback_confirmed=true
```

- Server responds with Request Credentials
  - request id - hh5s93j4hdidpola
  - shared secret - hdhd0244k9j7ao03

3. Redirection

- Client redirects user's browser to
  - https://photos.example.net/authorize?oauth_token=hh5s93j4hdidpola
- Notice that the URL identifies the request.
- Resource owner is prompted to supply username and password to login.
- Then prompted to grant access to the client.

7. Redirect back to Callback URI

- http://printer.example.com/ready?oauth_token=hh5s93j4hdidpola&oauth_verifier=hfdp7dh39dks9884
- oauth_verifier – takes the place of the PIN for a desktop application.

8. Client requests Token Credentials

```
POST /token HTTP/1.1
   Host: photos.example.net
   Authorization: OAuth realm="Photos",
     oauth_consumer_key="dpf43f3p2l4k3l03",
     oauth_token="hh5s93j4hdidpola",
     oauth_signature_method="HMAC-SHA1",
     oauth_timestamp="137131201",
     oauth_nonce="walatlh",
     oauth_verifier="hfdp7dh39dks9884",
     oauth_signature="gKgrFCywp7rO0OXSjdot%2FIHF7IU%3D"
```

- over a secure Transport Layer Security (TLS) channel.

8. Client requests Token Credentials

- oauth_consumer_key="dpf43f3p2l4k3l03"
  - Identifies the consumer
- oauth_token="hh5s93j4hdidpola",
  - Identifies the request
- oauth_verifier="hfdp7dh39dks9884",
  - Takes the place of the PIN

8. Server supplies token credentials

```
HTTP/1.1 200 OK
    Content-Type: application/x-www-form-urlencoded
    oauth_token=nnch734d00sl2jdk&oauth_token_secret=pfkkdhi9sl3r4s00
```

- These are the token (access) credentials.

9. Client requests resources using Access Token Credentials

```
GET /photos?file=vacation.jpg&size=original HTTP/1.1
    Host: photos.example.net
    Authorization: OAuth realm="Photos",
      oauth_consumer_key="dpf43f3p2l4k3l03",
      oauth_token="nnch734d00sl2jdk",
      oauth_signature_method="HMAC-SHA1",
      oauth_timestamp="137131202",
      oauth_nonce="chapoH",
      oauth_signature="MdpQcU8iPSUjWoN%2FUDMsK2sui9I%3D"
```

Calculating the signature (step 9)

- text
  - signature base string
  - consists of a concatenation of a number of other request parameters.
- key
  - combination of
    - client shared secret
    - token shared secret
- Hash the combination of text and key.

Authentication of request

- Server recalculates the HMAC.
- Compares to the client value.
- This authenticates the request.

- [But it does not prevent replay attacks.]

Replay Attacks – Nonce & Timestamp

- A replay attack is when the same request is repeated.
- So send
  - timestamp
  - nonce

Replay Attacks – Nonce & Timestamp

- Timestamp
  - the number of seconds since January 1, 1970 00:00:00 GMT.
- Nonce
  - a number that is only ever used once
  - on every request you need to make you must make sure that the nonce you use **is not already used in the same second**.

Replay Attacks – Nonce & Timestamp

- Now the combination of nonce and timestamp is unique.
  - If a request is replayed within a second, the nonce is invalid.
  - If a request is replayed after a second, the timestamp is invalid.
- Note that neither can be changed as that would invalidate the signature.

Security Considerations

Confidentiality

- OAuth provides a mechanism for verifying the integrity of requests and authenticating the client.
- It provides no guarantee of request confidentiality.
- Transport layer security is required for that.
- Does not authenticate the server.
- Again HTTPS required for that.

Plaintext storage of client secrets

- Plaintext storage of client secrets on the server is required.
- So that the server can check signatures.
- [As opposed to storing a one way hash of the credentials.]