OAuth 2.0: Theory and Practice

Daniel Correia
Pedro Félix
whoami

• Daniel Correia
  • Fast learner Junior Software Engineer
  • Passionate about everything Web-related
  • Currently working with the SAPO SDB team

• Pedro Félix
  • Teacher at ISEL – the engineering school of the Lisbon Polytechnic Institute
  • Independent consultant working with the SAPO SDB team
OAuth History

• OAuth started circa 2007
• 2008 - IETF normalization started in 2008
• 2010 - RFC 5849 defines OAuth 1.0
• 2010 - WRAP (Web Resource Authorization Profiles) proposed by Microsoft, Yahoo! And Google
• 2010 - OAuth 2.0 work begins in IETF
• 2012
  • RFC 6749 - The OAuth 2.0 Authorization Framework
  • RFC 6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage
An use case

• The cast of characters
  • [www.storecode.example](http://www.storecode.example) – code repository service (e.g. github.com)
  • [www.checkcode.example](http://www.checkcode.example) – code analysis service (e.g. travis-ci.org)
  • Alice – a fictional developer

• The problem
  • How can Alice allow checkcode to access her private code stored at storecode?
The password anti-pattern

• A solution: Alice shares her password with checkcode

• Problems:
  • Unrestricted access – checkcode has all of Alice’s permissions
    • read and write on all code repositories, issues, wiki, ...
  • No easy revocation
    • Changing password implies revoking all other client applications
  • Password management
    • Changing password implies updating all the delegated applications
The protocol

Alice's authentication and authorization delegation to checkcode

Authorization request

Authorization response code

Token request client creds code

Token response access_token

Service request access_token

Service response Alice's resource representation
A demo would be nice

Accessing GitHub
Developer experience

• Manage Clients (Applications)
  • client_id
  • client_secret
  • redirect_uri
User experience

• Grant authorizations
• Manage authorization
The OAuth 2.0 roles

Resource Owner (aka User)

Client

Authorization Server

Token response

access_token

service request

access_token

service response

Alice’s resource representation

Resource Server (aka Service)

Authorization response

code

Token request

client creds
code

www.checkcode.example

www.storecode.example
A matter of trust
Client Types

• **Confidential**
  “Clients capable of maintaining the confidentiality of their credentials”
  (e.g. client implemented on a secure server)

• **Public**
  “Clients incapable of maintaining the confidentiality of their credentials”
  (e.g. clients executing on the device used by the resource owner)
Client Types

• 3 implementation scenarios
  • Single client – all the users (web app)
  • One client per user (native mobile app)
  • One client per multiple users (family shared tablet, IPTV Box)

• Dynamic Client Registration
  • Client Registration Endpoint – still in draft
  • Turning public clients into private client instances
  • Not a closed problem
Authorization and Token Endpoints

Alice’s authentication and authorization delegation to checkcode

Authorization Endpoint

Authorization Server

Token Endpoint

www.checkcode.example

www.storecode.example

Front Channel

Back Channel

authorization request

authorization response

code

token request

client creds

code

token response

access_token

Back Channel

Authorization and Token Endpoints

www.checkcode.example

www.storecode.example

Front Channel

Back Channel

authorization request

authorization response

code

token request

client creds

code

token response

access_token
Front and back channels

• Front channel
  • Authorization Endpoint (AE)
  • Authorization request – redirect from Client to AE via the User-agent
  • Human interface – User authentication and authorization delegation
  • Authorization response – redirect from AE to Client via the User-agent

• Back channel
  • Token Endpoint (TE)
  • Direct request-response between Client and TE
  • No User interaction
  • No human interface
Scopes

• scope
  • “scope of the access request”
  • Parameter on the authorization request or token request
    • Set of space-delimited strings
  • E.g `https://www.googleapis.com/auth/calendar.readonly`

• Usages
  • Client – Must find the required scopes for each service interaction – docs
  • User – AS translates the scopes into friendly User messages
  • Service – Maps a scope into (URIs, methods) or (service, operation)

• Granted scope may differ from requested scopes
  • No provision for mandatory and optional scopes
The *grant* concept

- Represents the logical outcome of the User’s authorization
  - User identity
  - Client identity
  - Scope

- Core domain concept

- Bound to all the tokens
  - Code
  - Access token
  - Refresh token
Not (Keep It Simple)

Internet Engineering Task Force (IETF)
Request for Comments: 6749
Obsoletes: 5849
Category: Standards Track
ISSN: 2070-1721

D. Hardt, Ed.
Microsoft
October 2012

The OAuth 2.0 Authorization Framework
OAuth 2.0: a framework not a protocol

• The previous protocol is just a one of many options

• Three parts
  1. Obtaining user authorization
  2. Issuing access tokens
  3. Using access tokens to authorize service requests

• Multiple protocol flows
  • Different User authorization

• Critique
  • Complexity
  • Compromises interoperability
  • WS-* again?
Obtaining authorization

• Authorization Code **Grant**
  • The previous protocol

• Implicit **Grant**
  • Authorization Endpoint returns the access token directly
  • Javascript Clients running on the browser

• Resource Owner Password Credentials **Grant**
  • User gives password to Client, Client uses it to obtain access token

• Client Credentials **Grant**
  • No User, Client access on its own behalf

• Extensions
  • Identity federation, SAML assertions
Implicit Grant

Alice’s authentication and authorization delegation to checkcode

authorization request

authorization response

access_token

service request

access_token

service response

Alice’s resource representation
Resource Owner Password Credentials Grant

- Grant access
- User password

www.checkcode.example

- Token request
  - Client creds
  - User password

- Token response
  - Access_token

www.storecode.example

- Service request
  - Access_token

- Service response
  - Alice's resource representation
Client Credentials Grant

www.checkcode.example

<table>
<thead>
<tr>
<th>token request</th>
<th>client creds</th>
</tr>
</thead>
<tbody>
<tr>
<td>token response</td>
<td>access_token</td>
</tr>
<tr>
<td>service request</td>
<td>access_token</td>
</tr>
<tr>
<td>service response</td>
<td>Alice’s resource representation</td>
</tr>
</tbody>
</table>

www.storecode.example
Accessing the Token Endpoint

POST /token_endpoint HTTP/1.1
Host: as.storecode.example
Content-Type: application/x-www-form-urlencoded
Authorization: Basic <client_id:client_secret>

grant_type=authorization_code
code=AbCdEf...
redirect_uri=https://redirect.checkcode.example
client_id=...&
client_secret=..

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"Bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3J0kF0XG5Qx2T1KWIA",
  "example_parameter":"example_value"
}
Accessing the service (Resource Server)

• How to associate the access token to the request message?
• Bearer – just append the token to the request message – RFC 6750
  • Just like “bearer checks” or HTTP cookies
• MAC (holder-of-key) – prove the possession of a key – still draft
  • Similar to OAuth 1.0 or to AWS (used in S3)

```
GET /resource HTTP/1.1
Host: api.storecode.example
Authorization: Bearer <access_token>
```

```
GET /resource HTTP/1.1
Host: api.storecode.example
Authorization: MAC id="...",
               nonce="...",
               mac="..."
```
Bearer vs. MAC

- **Bearer**
  - Simpler – no signatures
  - Require HTTPS
    - Incorrect use
  - RFC 6750
  - Similar to cookie usage
    - Behare of the fallacy
    - Same origin policies
  - Discoverability

- **MAC**
  - Safer
  - More complex – signature
    - Client library integration
Token structure

• Not covered by the RFCs
• Token content options
  • Artifact (reference/handle) – reference to stored data
    • Store Hash(artifact) and not artifact directly
    • At least 128 bits of entropy
    • Revocation – just clear the stored data
  • Assertions – contains the (cryptographically protected) data
    • JWT – JSON Web Token
    • Revocation – harder (e.g. maintain revocation list)
• Token data
  • Validity period
  • Grant (User, Client, Scopes)
  • Type ({code, access_token, refresh_token})
  • Usage (e.g. code should be used only once)
Refresh tokens

• Two lifetimes

• Access tokens – short lifetime
  • Bearer usage

• Refresh tokens – long lifetime
  • Usage requires client credentials
  • Useful for revocation

• Token Endpoint - obtain new access token given a refresh token

• Critique: state management on the client
Security: authorization request

- Request-response correlation
  - **state** parameter - unpredictable
  - Session-fixation attack
- Code search
  - At least 128 bit of entropy
  - Small usage period (e.g. 5 minutes)
  - Code bound to a client_id
  - Code usage throttled by client_id
Security: code exchange

Alice’s authentication and authorization delegation to checkcode

Authorization request: response_type=code, client_id, redirect_uri, scope, state

Authorization response: code, ...

Token request: client_secret, redirect_uri, code, ...

Token response: access_token
Mobile: authorization request

- Use a "web view"
  - e.g. Windows 8 `WebAuthenticationBroker`
- Use an external browser - how to obtain the response parameters?
- Redirect
  - Use localhost
  - Special redirect URI `urn:ietf:wg:oauth:2.0:oob` (Google uses it but not on RFC)
  - Custom redirect URI scheme
OAuth 2.0: for authorization not authentication

- Not safe for authentication in the general case
- OpenID Connect – OAuth 2.0 + authentication
SDB - Service Delivery Broker

- Brokering between service clients and service enablers (*implementations*)
  - Access Control (OAuth 1.0, API keys, ...)
  - Caching, protocol and format translation, ...
- Public market place - [https://store.services.sapo.pt](https://store.services.sapo.pt)
- Multi-tenant
References

  • RFCs
  • Drafts

• Eran Hammer