AUTHLETE

OAuth 2.0

Authorization Code flow

Security concerns

Security aspect

There are many layers…

(but nothing fancy…)

User centricity of the protocol

Consent to clients

AS rely on consent of the user and the sanity of the authorization request to grant authorization code to clients.

Consent to clients

AS rely on consent of the user and the sanity of the authorization request to grant authorization code to clients.

Consent to clients

Whenever there is a person, an unconscious error or a social engineer attack can take place

Vectors

- Complexity of the scopes
- User behavior
- Deceiving clients

Complexity of the scopes

- Scopes are not segregated
  - "View and manage the files"
  - "View and manage your mail"
  - "Read Consumer" and "Write Consumer"

Complexity of the scopes

- Scopes for processing sensitive data
  - Mailbox scanners
  - Monitoring tools for documents

Complexity of the scopes

- Scopes for configuring tenants
  - Configure federation
  - Configure directories, instances, rules, etc.

(Everything that can be user for persistent access)

Recent attacks

Many recent attacks have too broad permissions granted as root cause

Users tend to over consent

- Lack of awareness on what is been granted
- Play down the risk of granting access to the client
- Blame AS in case of abuse by the client

Consent to client

The permission granting UI needs to be explicit, prevent clickjacking and scopes should be planned to implement a segregation of permissions

Deceiving clients

- Close or exact same name
- Non printable characters
- Same graphics

Deceiving clients

Workflow with verification steps needs to take place when registering or changing client data

Taking public client as confidential

Confidential and public clients

- Confidential clients have a credential established.
- Public clients don’t have a credential.
  - Single page apps, Native apps or mobile apps can’t be shipped to customer with a credential

Common mistake

A common security problem is to take public clients as confidential, with the assumption that it was implemented by the company, or the secret is obfuscated on the app.

On top of that

Very often, the AS never ask for consent for those clients and too broad permissions are granted

The security problem

Too much trust on an instance that can’t be verified

The security problem

If you are implementing a client identification, or plan to implement it, you might be missing one simple option

Confidential and public clients

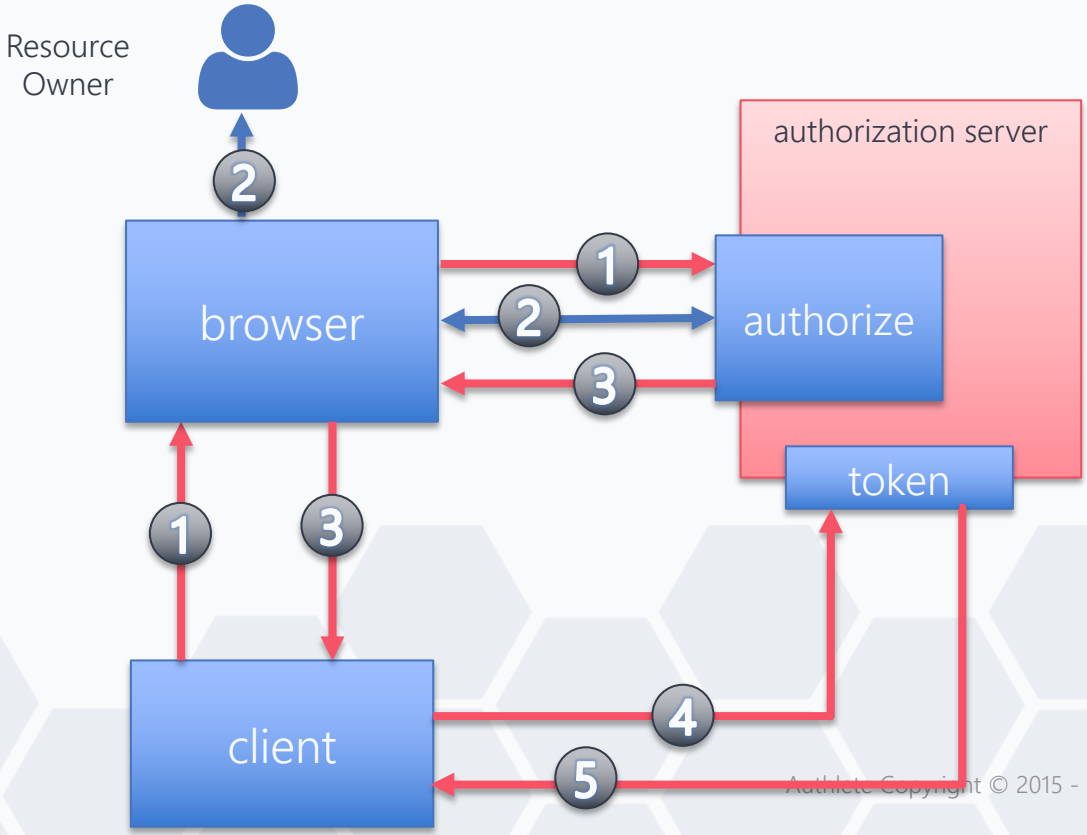
individual instances of public client can register a credential and become confidential

Confidential and public clients

the provisioning process can rely on Dynamic client registration (RFC 7591)

Protecting authorization request and code

Authorization request and code protection



1 – Redirect to authorize endpoint with response_type=code

2 – The user login and grant the permission

3 – redirect to client with the authorization code

4 – client send the authorization code to token with credentials

5 – AS returns the access token, refresh token, granted scopes and time to live of the token

Authorization request disclosure or tamper evident

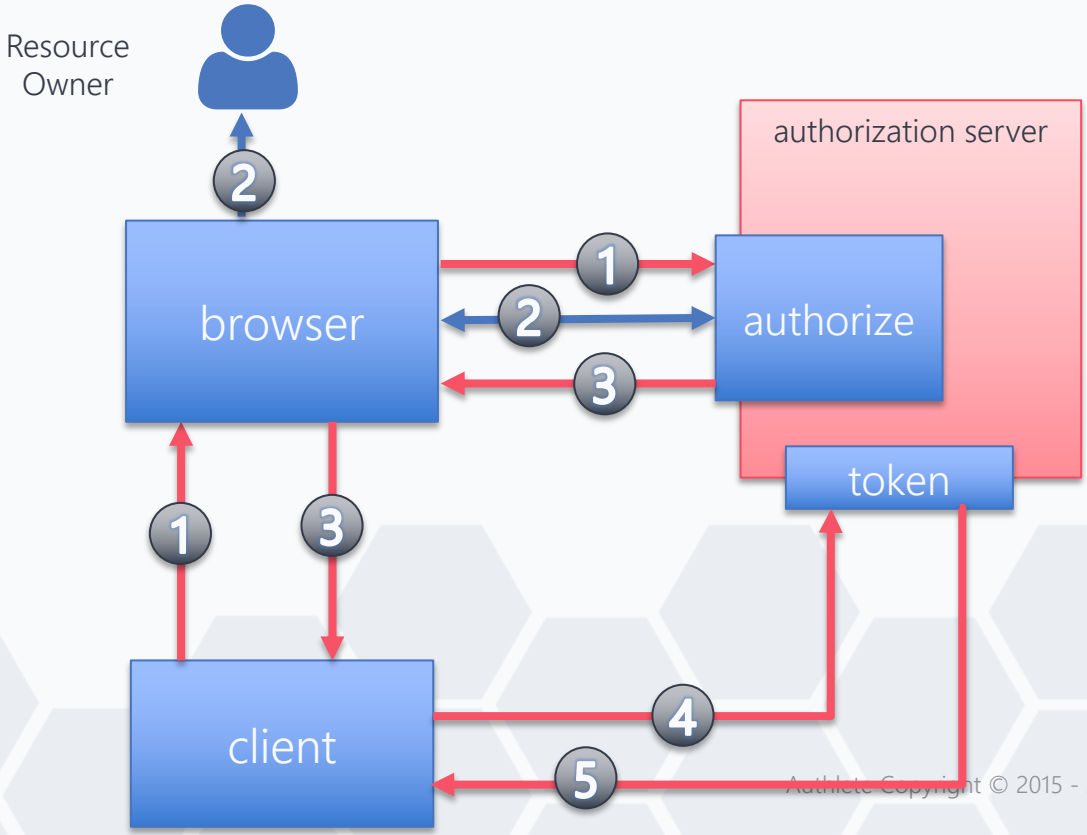
- The authorization parameters are introspected by the browser
- There are some approaches that server can use to prevent that:
  - Pushed Authorization Requests
  - Request Objects or JAR

Authorization code protection

- The code is required to be short living and single usage
- In case of public client, additional measures are required to prevent code interception
  - Proof Key for Code Exchange (RFC7636)

Open redirector

Open Redirector



1 – Redirect to authorize endpoint with response_type=code

2 – The user login and grant the permission

3 – redirect to client with the authorization code

4 – client send the authorization code to token with credentials

5 – AS returns the access token, refresh token, granted scopes and time to live of the token

Security concern – Open Redirector

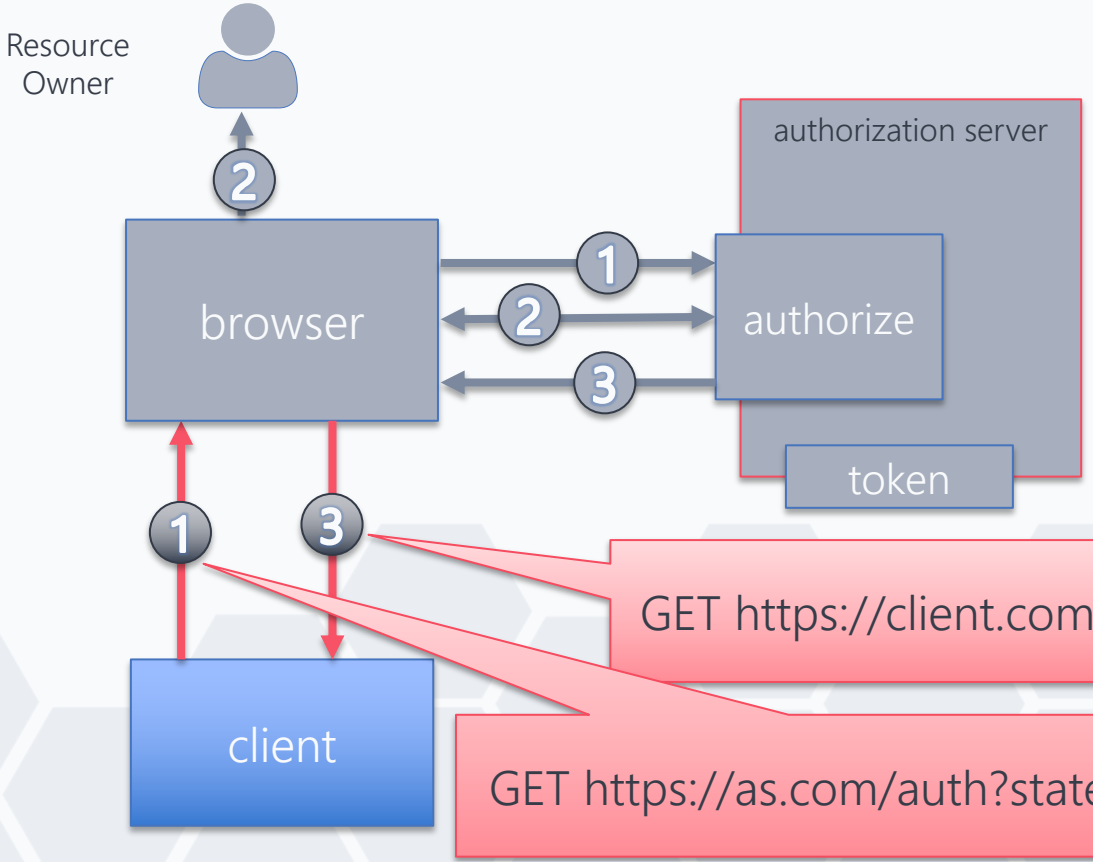
- The AS needs to validate the redirect_uri on the request against a set of registered uris

- Some security profiles require the redirect uri to be compared as exactly and in full

Security

Affecting clients

Redirect uri and CSRF



1 – Redirect to authorize endpoint with response_type=code

2 – The user login and grant the permission

3 – redirect to client with the authorization code

Redirect uri and CSRF

The client should use the state parameter. It can be used in Double submit cookie or as Synchronizer Token patterns

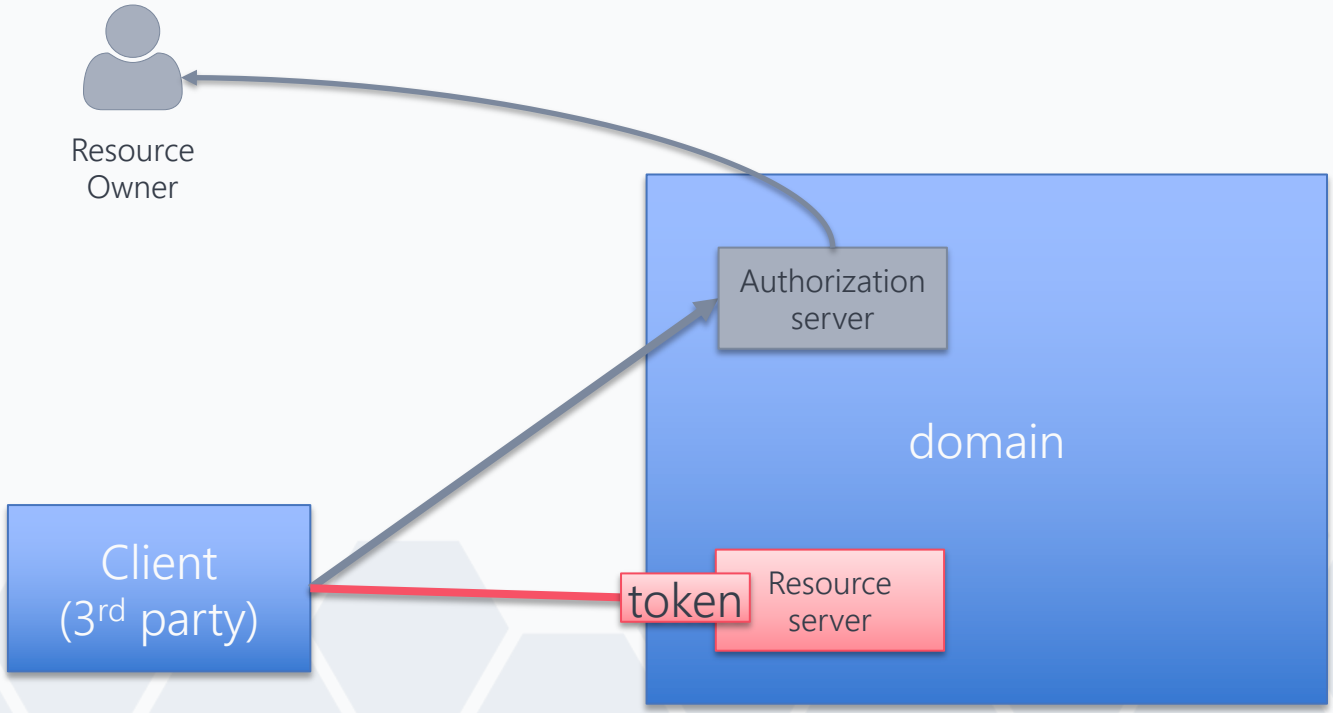
state support in practice

- When using it in Double submit cookie pattern the size might became a problem
  - The constraint is the size of the url: 2048 chars

- When implementing as Synchronizer Token it is sufficient

Bearer token

Client impersonation



Client impersonation

- Access token is sent back and forth between client and resource servers
- Transport between client and resource server should have forward secrecy

Access token

- Specifications for locking the access token to the specific client instance
  - MTLS bound (RFC 8705)
  - DPOP (draft-04 just published)

Refresh token

- It is sent back and forth between client and authorization server
- Very often is long living
- Transport between client and AS should have forward secrecy

AUTHLETE

www.authlete.com

AUTHLETE

OAuth 2.0

Authorization Code flow

Security concerns