Simplified Integration of OAuth into JavaScript Applications

OAuth Security Workshop 2017

Jacob Ideskog – Curity.io

@jacobideskog

Bio

- Stockholm, Sweden
- Work with OAuth 2.0 and OpenID Connect projects exclusively
- Implemented OAuth 2.0 and OpenID connect servers
- Architected and deployed many OAuth and OpenID Connect based infrastructures
- Organizer of Nordic APIs and Stockholm Java Meetup

The front-end struggle

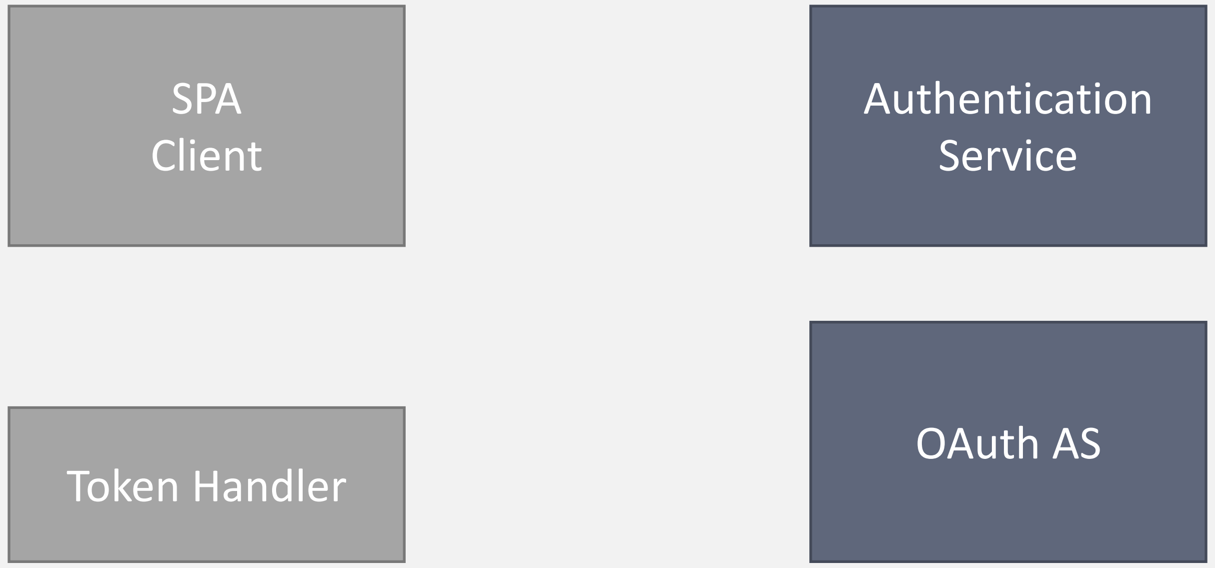
- Developers still struggle with OAuth 2.0
  - What endpoints to use
  - What parameters to send
  - Understanding the purpose of each token
    - Clients introspecting Access tokens
    - Clients sending refresh tokens to the wrong party
    - If using OIDC, sending ID Tokens

Why are we trying to solve this

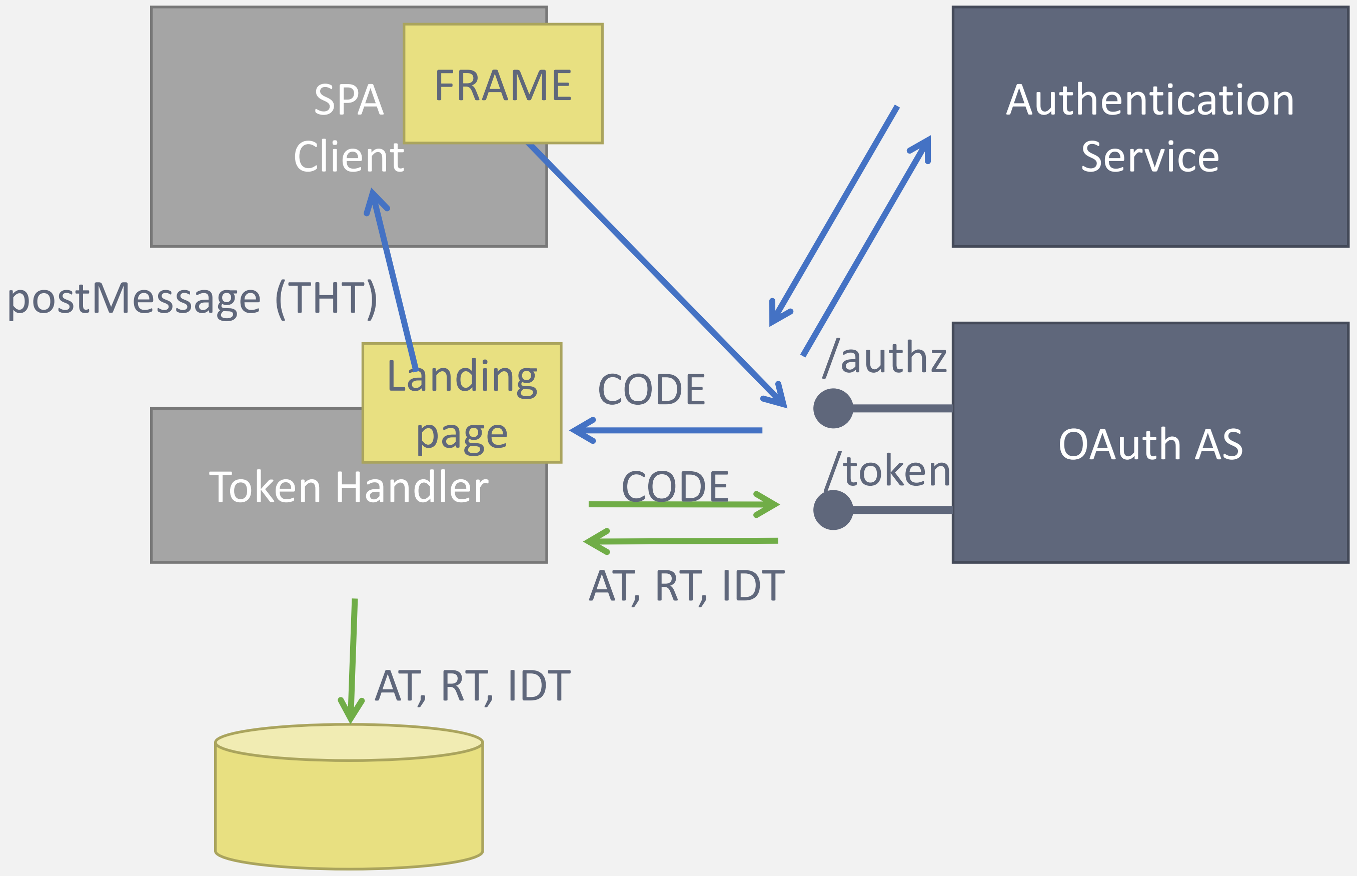
- Beyond the obvious reasons
  - Implicit flow is not enough
    - Loss of state
    - All complexity moved over to the client
    - No session management defined by the spec
    - Only Access Token is not always sufficient
- Token Handler
  - Pattern we've had to use many times for different customers

History: Token Handler (meta client)

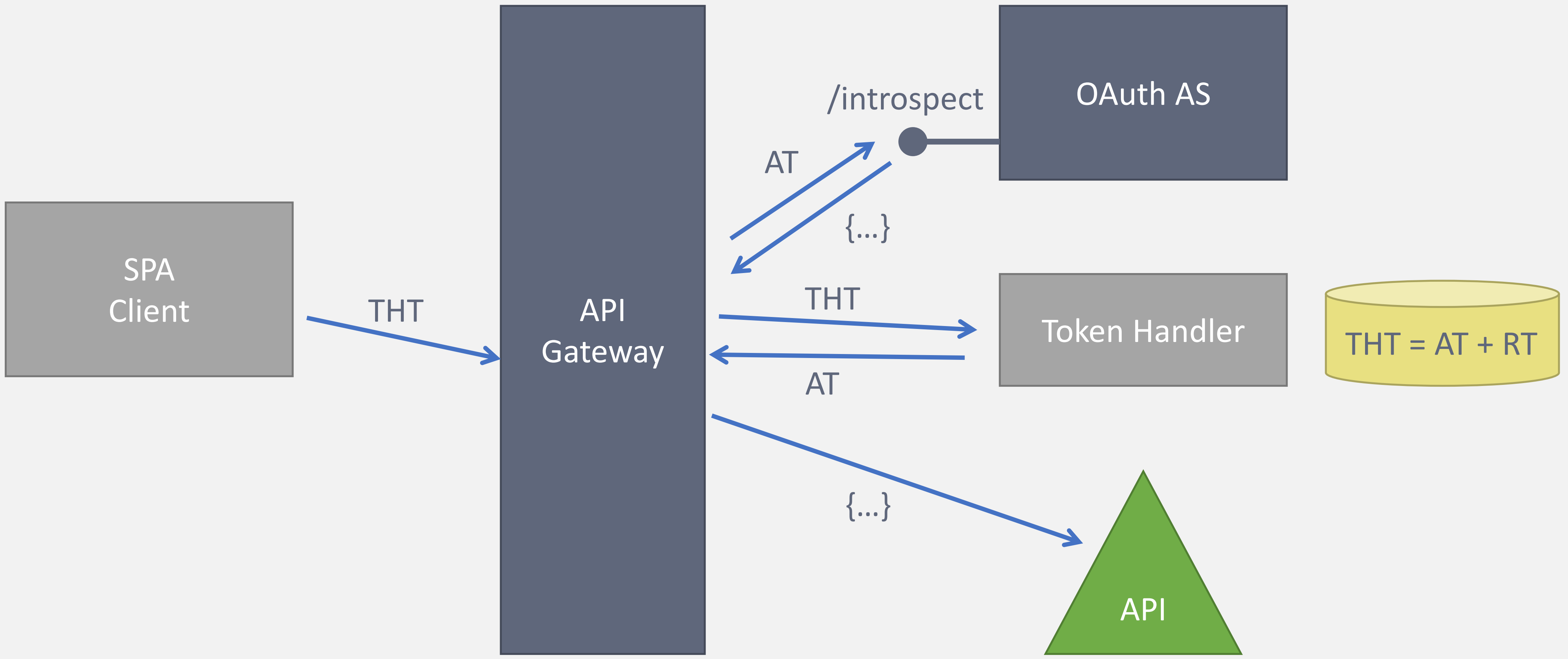
- Frameable (no need to leave the page)
  - Keep authentication decoupled
- Longer lived tokens



Token Handler



Using the THT (Token Handler Token)



Token Handler

Summary

Pros:

- Works around the standard
  - Keeping the solution compliant-ish
- Can use of-the-shelf OAuth servers

Cons:

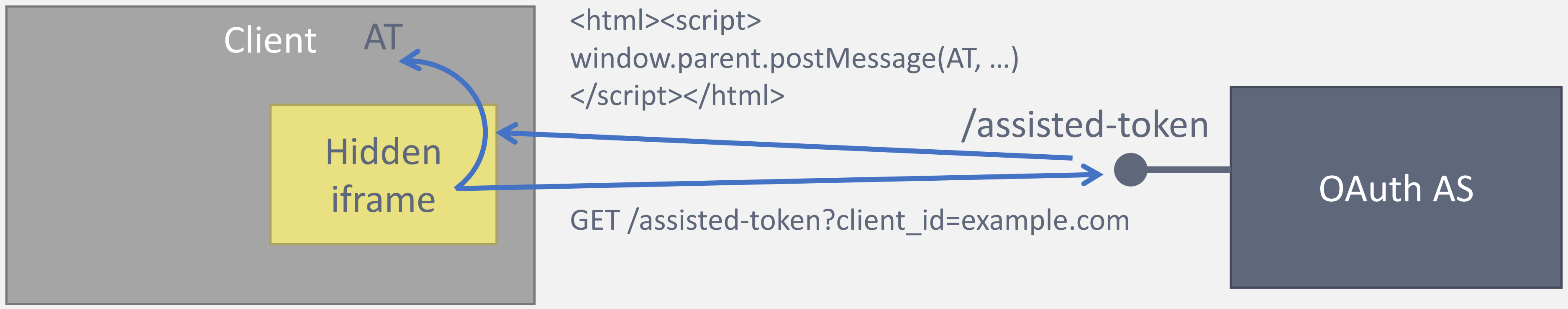
- Complicated
- Security concerns
- Aborting Authentication?
- Revoking tokens?
- Logout

Lessons learned

- Front-end developers need dirt-simple integration
  - Reduce number of parameters required
  - Provide ready APIs (libs)
  - Avoid duplication of tokens
- The SPA problem must be solved inside the OAuth server
- Possibility to strengthen client identity needed
  - What the Token Handler might know, the AS should know

The Assisted Token Flow

Example: User is already authenticated



```
<html><script>
window.parent.postMessage(AT, …)
</script></html>
```

```
GET /assisted-token?client_id=example.com
```

1. Open hidden iframe and do GET on the assisted token endpoint with client_id
2. Assisted token endpoint serves page that performs postMessage to parent frame
3. Parent page receive event with AT and closes iframe.

Success postMessage

- The success postMessage should at least contain:

```
{
   status: "success",
   access_token: “ABCDEFGH”,
   expires_in: 1499,
   scope: ”read write”
}
```

Add ID token if scope includes openid

- Together with target domain:

```
channel.postMessage(data, ”https://origin_of_client");
```

Authenticating

- If the user doesn’t have a session

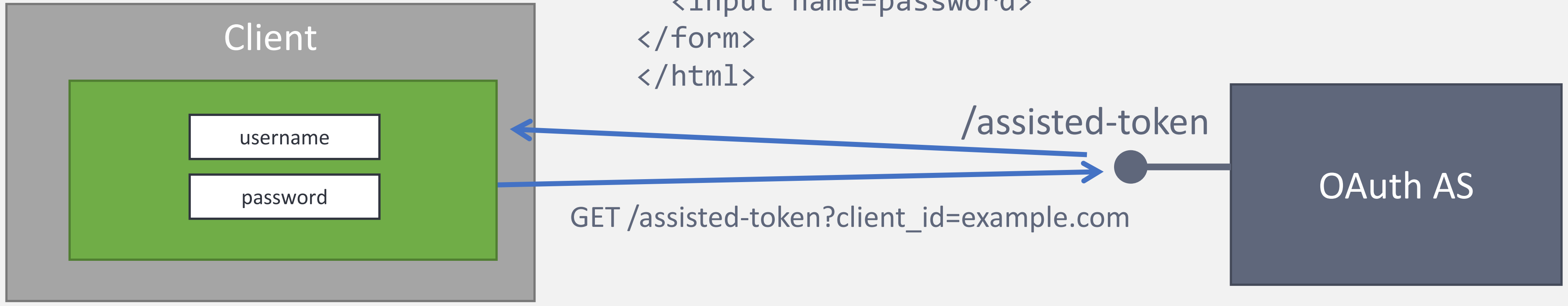
    1. The Client must be told, must be able to take action
    2. Must work without the need for OIDC
    3. Client cannot inspect child frame to find location, browser prohibits.

- Options
    1. Let the client TIMEOUT
    2. prompt=none (respond with error if no session)
    3. postMessage(“authenticating”,…

The Assisted Token Flow

Example: Authenticating user

```
<html>
<script>
  window.parent.postMessage(“authenticating”)
</script>
<form>
  <input name=username>
  <input name=password>
</form>
</html>
```



1. Open hidden iframe and do GET on the assisted token endpoint with client_id
2. Assisted token endpoint serves page that performs postMessage to parent frame informing about the authentication that needs to take place
3. The Client library can then close the hidden iframe, and restart the flow with a visible frame.
4. RO interacts with the content of the visible frame

Storing the token (Curity implementation)

- The Assisted Token endpoint stores the token in a cookie
- Criteria
  - HTTP only, meaning that client-side scripts will not be able to access it
  - Secure, Only transmitted over HTTPS
  - Use the domain of the AS and the path of the Assisted Token Endpoint
  - Have an expiration time that is equal to that of the token

- Benefit:
  - Remove errors possible by front-end developers when storing token

The Assisted Token Endpoint

- Why a new endpoint
  - No overloading = fewer required parameters
  - Clear separation from existing protocol
- Equivalent to new grant type
- Takes 1 parameter mandatory:
  - `client_id`
- Optional parameters:
  - `scope`
  - `for_origin (see table)`
  - `reuse`
  - *`forceAuthN, freshness`* `(OIDC overlay)`

scope	Issue all scopes configured for client if empty
for_origin	Required if more than one origin is configured on client
reuse	Reuse existing session, default true
forceAuthN, freshness	Require new authentication to take place (Open ID overlay)

for_origin

- Not the same as redirect_uri
    - The framer isn’t necessary the one to be callbacked during implicit flow.
- A client is REQUIRED to be configured with a for_origin.
    - The domain from which the client is served.
    - If more than one is configured, the client needs to send for_origin together with client_id.
- The AS can then ensure secure framing
    - Using X-FRAME-OPTIONS (can only handle one origin)
    - Also, server SHOULD respond with CSP headers (supports multiple origins)

scope

- If no scope is requested:
- **ALL configured scopes are returned**
  - Different from regular authorization + token endpoints
  - Reason not to overload
- Needed for client simplicity

Affects

- Dynamic Client Registration spec
  - Should be possible to request dynamically
- Metadata spec
  - Assisted endpoint should be published
  - grant_types_supported should include “assisted-token”
- No known effect on
  - Introspection
  - Revocation

Client side code should be non-normative

- Specification should not define JavaScript API
  - Leave open for implementation preferences
  - Authentication dealt with depending on server
  - Sessions not specified in OAuth so must leave open

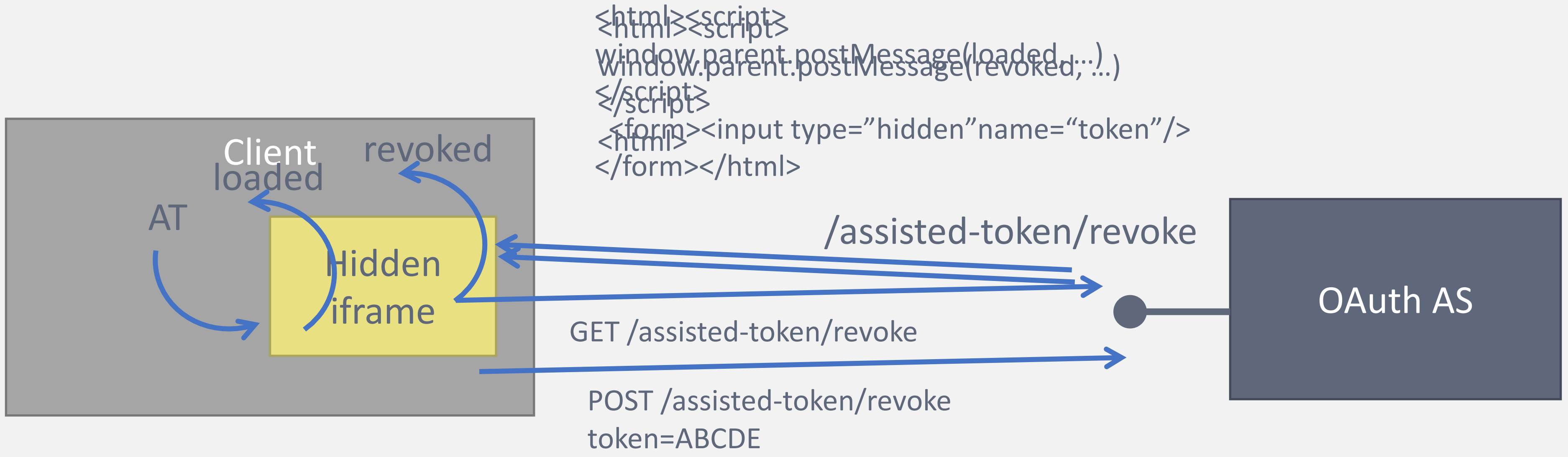
- Psudo code examples should be provided

Revoke and Logout

- The Assisted endpoint overloads revoke
  - To avoid CORS on regular revoke endpoint
- Perform regular FORM POST in iframe with token to revoke
- postMessage response

- Can be combined with session logout for OIDC

Revoke



<html><script>
window.parent.postMessage(loaded,...)
</script>
<html>
<html><script>
window.parent.postMessage(revoked, ...)
</script>
<form><input type="hidden"name="token"/>
</form></html>
Client
loaded
revoked
AT
Hidden iframe
/assisted-token/revoke
GET /assisted-token/revoke
POST /assisted-token/revoke
token=ABCDE
OAuth AS

1. Open hidden iframe and do GET on the assisted token revoke endpoint
2. When loaded, postMessage token to iframe
3. iframe performs POST with token to AS
4. AS responds with new postMEssage
5. Parent page receive event and closes iframe.

Security considerations

- postMessage – protected by the browser to not send to the wrong origin. **(REQUIRED)**
  - AS MUST postMessage to for_origin configured for Client
  - Client MUST check that origin of event matches OAuth server's origin
- X-FRAME-OPTIONS + CSP headers for framing **(REQUIRED)**
  - Provides stronger client assertion than implicit flow
- iframe breakout JavaScript (last resort)
- Simplicity for developer (RECOMMENDATIONS)
  - Libraries should assist with CORS setup
  - Make sure jQuery (XHR) and others send token to whitelisted APIs

Conclusion

- The front-end developers need all the help they can get
- Even 2-4 parameters in the implicit flow is hard (yes)
- A new flow that assists with OAuth is needed
- Should be possible to overlay with OpenID Connect
- Define wire protocol

- We propose to author a spec together with OAuth WG and contribute IP

DEMO & Questions

Assisted Token in Curity

CURITY

info@curity.io

@jacobideskog

jacob.ideskog@curity.io

CURITY