Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 19 February 2026

Y. Zehavi
Raiffeisen Bank International

18 August 2025

OAuth 2.0 App2App Browser-less Flow
draft-zehavi-oauth-app2app-browserless-06

Abstract

This document describes a protocol allowing a _Client App_ to obtain an OAuth grant from an _Authorization Server's Native App_ using the [App2App] pattern, providing *native* app navigation user-experience (no web browser used), despite both apps belonging to different trust domains.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://yaron-zehavi.github.io/oauth-app2app-browserless/draft-zehavi-oauth-app2app-browserless.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-zehavi-oauth-app2app-browserless/.

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (mailto:oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/. Subscribe at https://www.ietf.org/mailman/listinfo/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/yaron-zehavi/oauth-app2app-browserless.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

This document describes a protocol enabling native app navigation of an [App2App] OAuth grant across _different Trust Domains_.

When _Clients_ and _Authorization Servers_ are located on _different Trust Domains_, authorization requests traverse across domains using federation, involving _Authorization Servers_ acting as clients of _Downstream Authorization Servers_.

Such federation setups create trust networks, for example in Academia and in the business world across corporations.

In federated [App2App] scenarios the *web browser* serves as user-agent, as federated Authorization Servers url’s are not claimed by any native app.

The use of web browsers in App2App flows degrades the user experience somewhat.

This document specifies:

native_authorization_endpoint: A new Authorization Server endpoint and corresponding metadata property REQUIRED to support the browser-less App2App flow.

native_callback_uri: A new native authorization request parameter, specifying the deep link of _Client App_.

native_app2app_unsupported: A new error code value.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Terminology

In addition to the terms defined in referenced specifications, this document uses the following terms:

OAuth: In this document, "OAuth" refers to OAuth 2.0, [RFC6749] in the *authorization code flow*.

OAuth Broker: An Authorization Server federating to other trust domains by acting as an OAuth Client of _Downstream Authorization Servers_.

Client App: A Native app OAuth client of _Authorization Server_. In accordance with "OAuth 2.0 for Native Apps" [RFC8252], client's redirect_uri is claimed by the app.

Downstream Authorization Server: An Authorization Server downstream of another _Authorization Server_. It may be an _OAuth Broker_ or the _User-Interacting Authorization Server_.

User-Interacting Authorization Server: An Authorization Server which interacts with end-user. The interaction may be interim navigation (e.g: user input is required to guide where to federate), or performs user authentication and request authorization.

User-Interacting App: Native App of _User-Interacting Authorization Server_.

Deep Link: A url claimed by a native application.

3. Protocol

3.1. Flow Overview

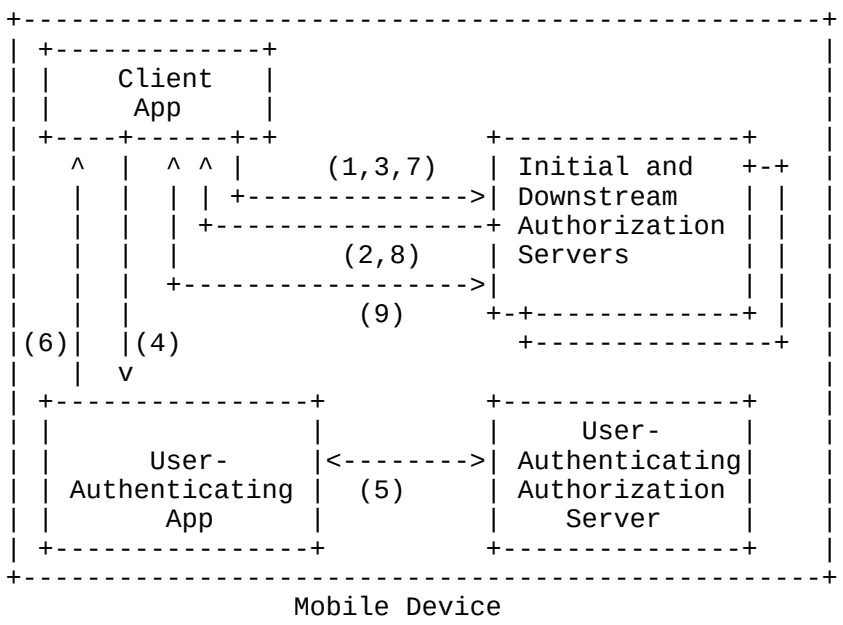


Figure 1: Browser-less App2App across trust domains

* (1) _Client App_ presents an authorization request to _Authorization Server's_ *native_authorization_endpoint*, including a *native_callback_uri*.

* (2) _Authorization Server_ returns either:

  - A _native authorization request url_ for a _Downstream Authorization Server_.

  - A request for end-user input to guide request routing.

  - A _deep link_ url to its _User-Interacting App_.

* (3) _Client App_:

  - Calls _native authorization request urls_ it obtains, so long as such responses are obtained, until a _deep link_ url to _User-Interacting App_ is obtained.

  - Prompts end-user, then provides their input to _Authorization Server_ to guide request routing.

  - Handles _deep links_, by invoking the app claiming the url, if present on the device.

* (4) _Client App_ natively invokes _User-Interacting App_ claiming a _deep link_ it has obtained.

* (5) _User-Interacting App_ authenticates end-user and authorizes the request.

* (6) _User-Interacting App_ returns to _Client App_ by natively invoking *native_callback_uri* and provides the url-encoded _redirect_uri_ with its response parameters.

* (7) _Client App_ invokes the _redirect_uri_ it obtained.

* (8) _Client App_ calls any subsequent uris obtained until its own redirect_uri is obtained.

* (9) _Client App_ exchanges code for tokens and the flow is complete.

3.2. Authorization Server Metadata

This document introduces the following parameter as authorization server metadata [RFC8414], indicating support of _Native App2App_:

native_authorization_endpoint: URL of the authorization server’s native authorization endpoint.

3.3. native_authorization_endpoint

This is an OAuth authorization endpoint, interoperable with other OAuth RFCs.

The following additional requirements apply to native_authorization_endpoint, in line with common REST APIs:

* SHALL NOT use cookies.

* SHALL return Content-Type header with the value "application/json", and a JSON http body.

* SHALL NOT return HTTP 30x redirects.

* SHALL NOT respond with bot-detection challenges such as CAPTCHAs.

3.4. Native Authorization Request

An OAuth authorization request, interoperable with other OAuth RFCs, which also includes the _native_callback_uri_ parameter:

native_callback_uri: REQUIRED. _Client App's_ deep link, to be invoked by _User-Interacting App_. When invoking _native_callback_uri_, it accepts the following parameter:

redirect_uri: REQUIRED. url-encoded redirect_uri from _User-Interacting App_ responding to its OAuth client, including its respective response parameters.

Authorization servers processing a _native authorization request_ MUST also:

* Forward the _native_callback_uri_ in their requests to _Downstream Authorization Servers_.

* Ensure that the _Downstream Authorization Servers_ it federates to, offers a _native_authorization_endpoint_, otherwise return an error response with error code _native_app2app_unsupported_.

3.5. Native Authorization Response

The authorization server responds with _application/json_ and either 200 OK or 4xx/5xx.

3.5.1. Federating response

If the _Authorization Server_ decides to federate to another party such as _Downstream Authorization Server_ or its OAuth client, it responds with 200 OK and the following JSON response body:

action: REQUIRED. A string with the value "call" to indicate that _url_ is to be called with HTTP GET.

url: REQUIRED. A string holding a native authorization request for _Downstream Authorization Server_, or redirect_uri of an OAuth client with a response.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "action": "call",
    "url": "https://next-as.com/auth/native",
}
```

Client App SHALL add all DNS domains of _urls_ it encounters during each flow to an Allowlist, used to validate urls in the response handling phase, after being invoked by the _User-Interacting Authorization Server' App_.

It then MUST make an HTTP GET request to the returned _url_ and process the response as defined in this document.

3.5.2. Deep Link Response

If the _Authorization Server_ wishes to authenticate the user and authorize the request, using its _User-Interacting App_, it responds with 200 OK and the following JSON response body:

action: REQUIRED. A string with the value "deep_link" to indicate that _url_ is to be called with HTTP GET.

url: REQUIRED. A string holding the deep link url claimed by the _User-Interacting App_.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "action": "deep_link",
    "url": "uri of native authorization request handled by *User-Interacting App*",
}
```

Client App MUST use OS mechanisms to invoke the deep link received in _url_ and open the _User-Interacting Authorization Server's App_. If no app claiming the deep link is be found, _Client App_ MUST terminate the flow and MAY attempt a non-native flow. See Section 4.2.

3.5.3. Routing Response

If the _Authorization Server_ requires user input to determine where to federate, it responds with 200 OK and the following JSON body:

id: OPTIONAL. A string holding an interaction identifier used by _Authorization Server_ to link the response to the request.

action: REQUIRED. A string with the value "prompt" to indicate that the client app must prompt the user for input before proceeding.

logo: OPTIONAL. URL or base64-encoded logo of _Authorization

Server_, for branding purposes.

userPrompt: REQUIRED. A JSON object containing the prompt definition. The following parameters MAY be used:

* options: OPTIONAL. A JSON object that defines a dropdown/select input with various options to choose from. Each key is the parameter name to be sent in the response and each value defines the option:
  - title: OPTIONAL. A string holding the input’s title.
  - description: OPTIONAL. A string holding the input’s description.
  - values: REQUIRED. A JSON object where each key is the selection value and each value holds display data for that value:
    o name: REQUIRED. A string holding the display name of the selection value.
    o logo: OPTIONAL. A string holding a URL or base64-encoded image for that selection value.
* inputs: OPTIONAL. A JSON object that defines an input field. Each key is the parameter name to be sent in the response and each value defines the input field:
  - title: OPTIONAL. A string holding the input’s title.
  - hint: OPTIONAL. A string holding the input’s hint that is displayed if the input is empty.
  - description: OPTIONAL. A string holding the input’s description.

response: REQUIRED. A JSON object that holds the URL to which the user input MUST be sent. It only supports two keys, which are mutually exclusive:

* get: The corresponding value is the URL to use for a GET request with user input appended as query parameters.
* post: The corresponding value is the URL to use for a POST request with user input sent in the request body, as application/x-www-form-urlencoded.

Example of prompting end-user for 2 multiple-choice inputs:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "action": "prompt",
    "id": "request-identifier-1",
    "logo": "uri or base64-encoded logo of Authorization Server",
    "userPrompt": {
        "options": {
            "bank": {
                "title": "Bank",
                "description": "Choose your Bank",
                "values": {
                    "bankOfSomething": {
                        "name": "Bank of Something",
                        "logo": "uri or base64-encoded logo"
                    },
                    "firstBankOfCountry": {
                        "name": "First Bank of Country",
                        "logo": "uri or base64-encoded logo"
                    }
                }
            },
            "segment": {
                "title": "Customer Segment",
                "description": "Choose your Customer Segment",
                "values": {
                    "retail": "Retail",
                    "smb": "Small & Medium Businesses",
                    "corporate": "Corporate",
                    "ic": "Institutional Clients"
                }
            }
        }
    },
    "response": {
        "post": "url to POST to using application/x-www-form-urlencoded",
        "get": "url to use for a GET with query params"
    }
}
```

Example of prompting end-user for text input entry:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.oauth.app2app.routing+json

{
    "action": "prompt",
    "id": "request-identifier-2",
    "logo": "uri or base64-encoded logo of Authorization Server",
    "userPrompt": {
        "inputs": {
            "email": {
                "hint": "Enter your email address",
                "title": "E-Mail",
                "description": "Lorem Ipsum"
            }
        }
    },
    "response": {
        "post": "url to POST to using application/x-www-form-urlencoded",
        "get": "url to use for a GET with query params"
    }
}
```

Client App MUST prompt the user according to the response received. It then MUST send the user input to the response endpoint using the requested method including the interaction id, if provided.

Example of _Client App_ response following end-user multiple-choice:

```
POST /native/routing HTTP/1.1
Host: example.as.com
Content-Type: application/x-www-form-urlencoded

id=request-identifier-1
&bank=bankOfSomething
&segment=retail
```

Example of _Client App_ response following end-user input entry:

```
POST /native/routing HTTP/1.1
Host: example.as.com
Content-Type: application/x-www-form-urlencoded

id=request-identifier-2
&email=end_user@example.as.com
```

3.5.4. Error Response

If _Authorization Server_ encounters an error whose audience is its OAuth client, it returns 200 OK with the following JSON body:

action: REQUIRED. A string with the value "call" to indicate that _url_ is to called with HTTP GET.

url: REQUIRED. A string holding the redirect_uri of the OAuth client, including the OAuth error.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "action": "call",
    "url": "https://previous-as.com/auth/redirect?error=...&error_description=...&iss=...
&state=..."
}
```

Client App MUST make an HTTP GET request to the returned _url_ and process the response as defined in this document.

If _Authorization Server_ encounters an error, that it cannot/or must not send to its OAuth client, it responds with 4xx/5xx and the following JSON body:

error: REQUIRED. The error code as defined in [RFC6749] and other OAuth RFCs.

error_description: OPTIONAL. The error description as defined in [RFC6749].

Example:

```
HTTP/1.1 500 OK
Content-Type: application/json

{
    "error": "native_app2app_unsupported",
}
```

Client App SHOULD display an appropriate error message to the user and terminate the flow. In case of _native_app2app_unsupported_, _Client App_ MUST terminate the flow and MAY retry with a non-native flow. See Section 4.2.

3.6. User-Interacting Authorization Server’s App

The _User-Interacting Authorization Server’s_ app handles the native authorization request:

* Validates the native authorization request.

* Establishes trust in _native_callback_uri_ and validates that an app claiming _native_callback_uri_ is on the device. Otherwise terminates the flow.

* Authenticates end-user and authorizes the request.

* MUST use _native_callback_uri_ to invoke _Client App_, providing it the redirect url and its response parameters as the url-encoded query parameter *redirect_uri*.

3.7. Client App response handling

Client App is natively invoked by _User-Interacting Authorization Server App_:

* If invoked with an _error_ parameter, without parameters at all, it MUST terminate the flow.

* It MUST ignore any unknown parameters.

* If invoked with a url-encoded *redirect_uri* as parameter, _Client App_ MUST validate _redirect_uri_, and any url subsequently obtained, using the Allowlist it previously generated, and MUST terminate the flow if any url is not found in the Allowlist.

Client App SHALL invoke _redirect_uri_, and any validated subsequent obtained urls, using HTTP GET.

Authorization Servers processing _Native App2App_ MUST respond to their redirect_uri invocations:

* According to the REST API guidelines specified in Section 3.3.

* Returning a JSON body instructing the next url to call.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "action": "call",
    "url": "redirect_uri of an OAuth Client, including response parameters",
}
```

Client App MUST handle any other response (2xx with an unexpected Content-Type / 3xx / 4xx / 5xx) as a failure and terminate the flow.

3.8. Flow completion

Once _Client App's_ own redirect_uri is obtained, _Client App_ processes the response:

* Exchanges code for tokens.

* Or handles errors obtained.

And the _Native App2App_ flow is complete.

4. Implementation Considerations

4.1. Detecting Presence of Native Apps claiming Urls

Native Apps on iOS and Android MAY use OS SDK's to detect if an app claims a url.

See Appendix A for more details.

4.2. Recovery from failed native App2App flows

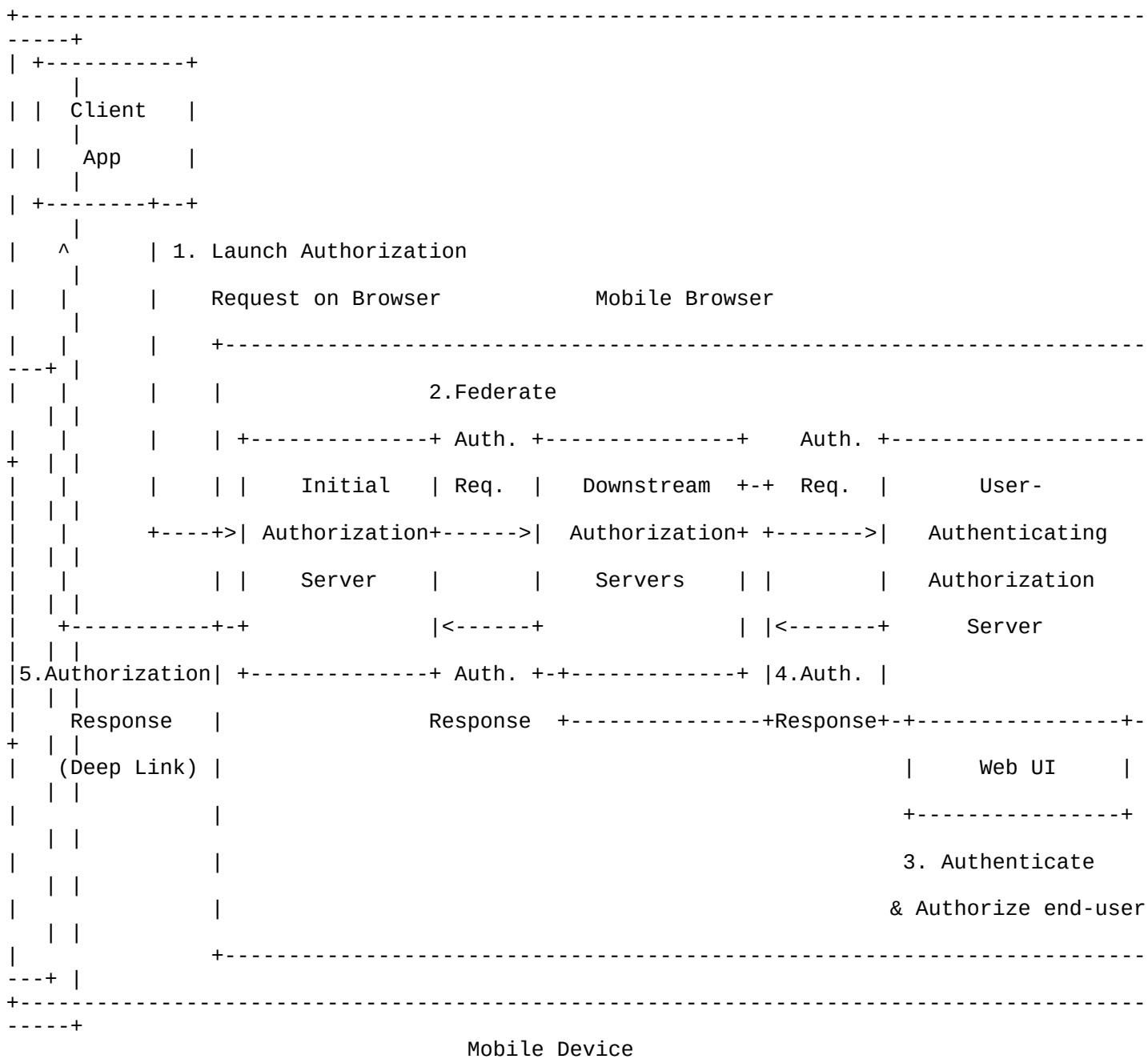


Figure 2: App2Web using the browser

The _Native App2App flow_ described in this document MAY fail when:

* An error response is obtained.

* Required _User-Interacting App_ is not installed on end-user’s device.

Client App MAY recover by launching a new, non-native authorization request on a web browser, in accordance with "OAuth 2.0 for Native Apps" [RFC8252].

Note - Failure because _User-Interacting App_ is not installed on end-user’s device, might succeed in future, if the missing app has been installed. _Client App_ MAY choose if and when to retry the

Native App2App flow after such a failure.

5. Security Considerations

5.1. Embedded User Agents

[RFC8252] Security Considerations advises against using _embedded user agents_. The main concern is preventing theft through keystroke recording of end-user's credentials such as usernames and passwords.

Client App when interacting with end-user to provide routing guiding input MUST NOT be used to request authentication credentials or any other sensitive information.

5.2. Open redirection by Authorization Server's User-Interacting App

To mitigate open redirection attacks, trust establishment in _native_callback_uri_ is RECOMMENDED by _User-Interacting App_. Any federating _Authorization Server_ MAY also wish to establish trust.

The specific trust establishment mechanisms are outside the scope of this document. For example purposes only, one possible way to establish trust is [OpenID.Federation]:

* Strip url path from *native_callback_uri* (retaining the DNS domain).

* Add the url path /.well-known/openid-federation and perform trust chain resolution.

* Inspect Client's metadata for redirect_uri's and validate *native_callback_uri* is included.

5.3. Open redirection by Client App

Client App SHALL construct an Allowlist of DNS domains it traverses while processing the request, used to enforce all urls it later traverses during response processing. This mitigates open redirection attacks as urls not in this Allowlist SHALL be rejected.

In addition _Client App_ MUST ignore any invocation for response processing which is not in the context of a request it initiated. It is RECOMMENDED the Allowlist be managed as a single-use object, destructed after each protocol flow ends.

It is RECOMMENDED _Client App_ allows only one OAuth request processing at a time.

5.4. Deep link hijacking

It is RECOMMENDED that all apps in this specification shall use https-scheme deep links (Android App Links / iOS universal links). Apps SHOULD implement the most specific package identifiers mitigating deep link hijacking by malicious apps.

5.5. Authorization code theft and injection

It is RECOMMENDED that PKCE is used and that the code_verifier is tied to the _Client App_ instance, as mitigation to authorization code theft and injection attacks.

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

[OpenID.Federation] Hedberg, Ed, R., Jones, M. B., Solberg, A. A., Bradley, J., De Marco, G., and V. Dzhuvinov, "OpenID Federation 1.0", March 2025, <https://openid.net/specs/openid-federation-1_0.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <https://www.rfc-editor.org/rfc/rfc8252>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/rfc/rfc8414>.

[RFC9126] Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", RFC 9126, DOI 10.17487/RFC9126, September 2021, <https://www.rfc-editor.org/rfc/rfc9126>.

[RFC9396] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, DOI 10.17487/RFC9396, May 2023, <https://www.rfc-editor.org/rfc/rfc9396>.

7.2. Informative References

[android.method.intent] "Android Intent Method", n.d., <https://developer.android.com/reference/android/content/Intent>.

[App2App] Heenan, J., "Guest Blog: Implementing App-to-App Authorisation in OAuth2/OpenID Connect", October 2019, <https://openid.net/guest-blog-implementing-app-to-app-authorisation-in-oauth2-openid-connect/>.

[iOS.method.openUrl] "iOS open(_:options:completionHandler:) Method", n.d., <https://developer.apple.com/documentation/uikit/uiapplication/open(_:options:completionhandler:)>.

[iOS.option.universalLinksOnly] "iOS method property universalLinksOnly", n.d., <https://developer.apple.com/documentation/uikit/uiapplication/openexternalurloptionskey/universallinksonly>.

[OAuth.First-Party] Parecki, A., Fletcher, G., and P. Kasselman, "OAuth 2.0 for First-Party Applications", November 2022, <https://www.ietf.org/archive/id/draft-ietf-oauth-first-party-apps-01.html>.

[OpenID.Native-SSO] Fletcher, G., "OpenID Connect Native SSO for Mobile Apps", November 2022, <https://openid.net/specs/openid-connect-native-sso-1_0.html>.

Appendix A. Detecting Presence of Native Apps claiming Urls on iOS and Android

A.1. iOS

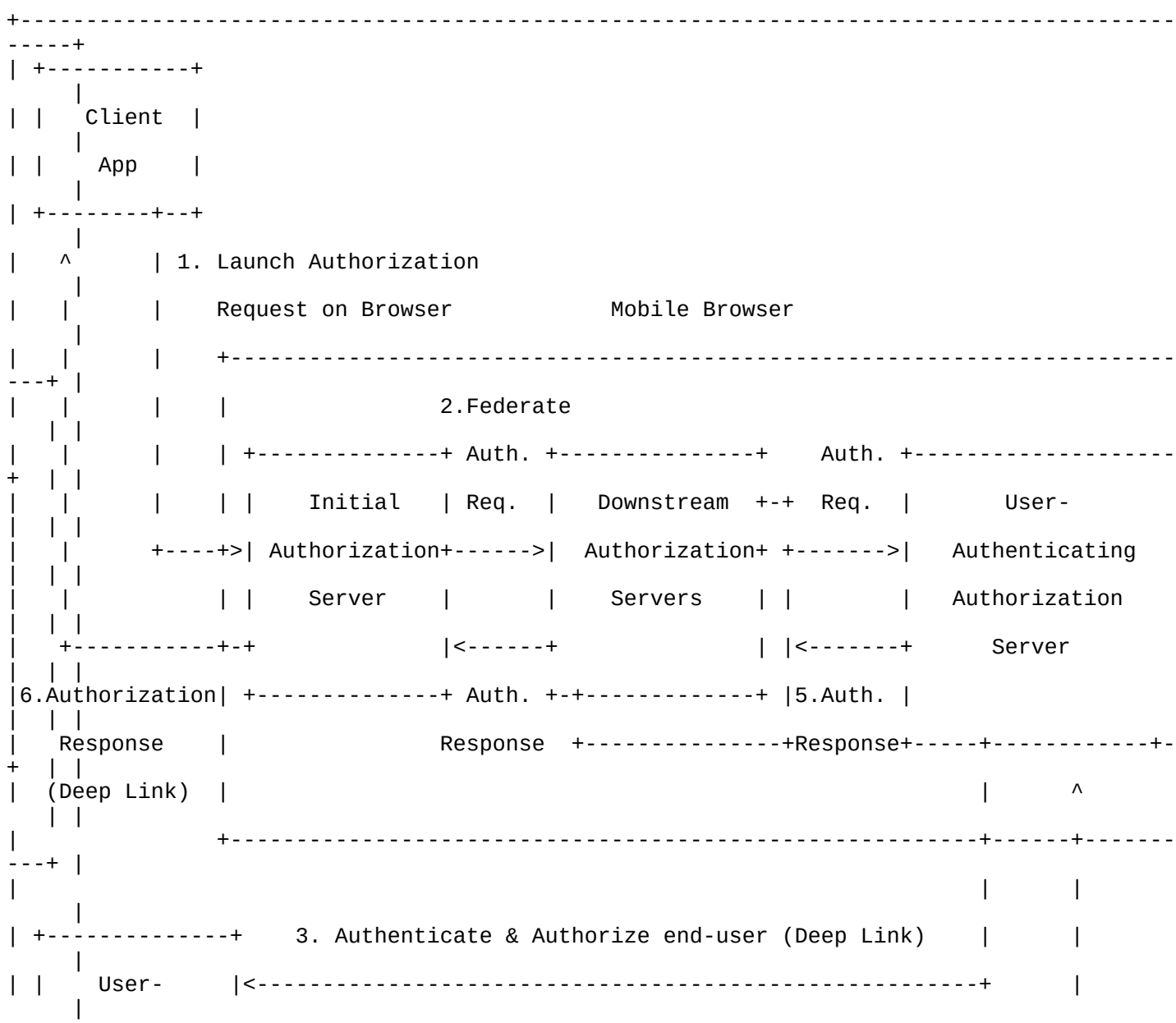
App SHALL invoke iOS [iOS.method.openUrl] method with options [iOS.option.universalLinksOnly] which ensures URLs must be universal links and have an app configured to open them. Otherwise the method returns false in completion.success.

A.2. Android

App SHALL invoke Android [android.method.intent] method with FLAG_ACTIVITY_REQUIRE_NON_BROWSER, which throws ActivityNotFoundException if no matching app is found.

Appendix B. Background and relation to other standards

B.1. App2App across trust domains requires a web browser



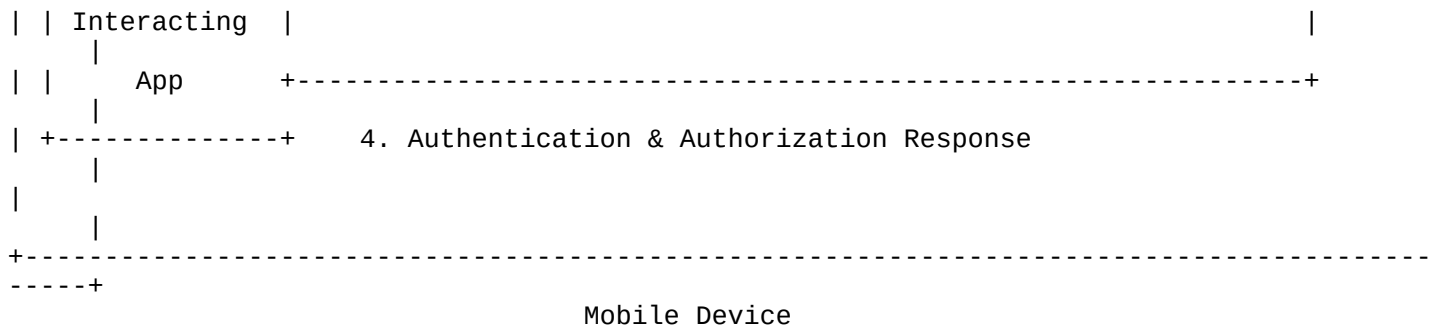

Interacting
App
4. Authentication & Authorization Response
Mobile Device

Figure 3: App2App across trust domains using browser

Since no native app claims the urls of redirecting Authorization Servers (_OAuth Brokers_), mobile Operating Systems default to using the system browser as the User Agent.

B.2. Impact of using a web browser

Using a web browser may degrade the user experience in several ways:

* Some browser’s support for deep links is limited by design, or by the settings used.

* Browsers may prompt end-user for consent before opening apps claiming deep links, introducing additional friction.

* Browsers are noticeable by end-users, rendering the UX less smooth.

* Client app developers don’t control which browser the _User-Interacting App_ uses to provide its response to redirect_uri. Opinionated choices pose a risk that different browsers will use, making necessary cookies used to bind session identifiers to the user agent (nonce, state or PKCE verifier) unavailable, which may break the flow.

* After flow completion, "orphan" browser tabs may remain. They do not directly impact the flow, but can be regarded as unnecessary "clutter".

B.3. Relation to [OpenID.Native-SSO]

[OpenID.Native-SSO] also offers a native SSO flow across apps. However, it is limited to apps:

* Published by the same issuer, therefore can securely share information.

* Using the same Authorization Server.

B.4. Relation to [OAuth.First-Party]

[OAuth.First-Party] also deals with native apps, but it MUST only be used by first-party applications, which is when the authorization server and application are controlled by the same entity, which is not true in the case described in this document.

While this document also discusses a mechanism for _Authorization Servers_ to guide _Client App_ in obtaining user's input to guide routing the request across trust domains, the [OAuth.First-Party] required high degree of trust between the authorization server and the client is not fulfilled.

Appendix C. Acknowledgments

The authors would like to thank the following individuals who contributed ideas, feedback, and wording that shaped and formed the final specification: George Fletcher, Arndt Schwenkschuster, Henrik Kroll, Grese Hyseni. As well as the attendees of the OAuth Security Workshop 2025 session in which this topic was discussed for their ideas and feedback.

Appendix D. Document History

[[To be removed from the final specification]]

-latest

* Replaced Authorization Details Type with a new parameter

* native_authorization_endpoint as REST API - no cookies or HTTP 30x responses

-05

* removed error native_callback_uri_not_claimed

* Added Routing Instructions Response

* Added native_authorization_endpoint and matching AS profile

* Added Authorization Details Type as container for native_callback_uri

-04

* Phrased the challenge in Trust Domain terminology

* Discussed interim Authorization Server interacting the end-user, which is not the User-Authenticating Authorization Server

* Moved Cookies topic to Protocol Flow

* Mentioned that Authorization Servers redirecting not through HTTP 30x force the use of a browser

* Discussed Embedded user agents security consideration

-03

* Defined parameters and values

* Added error native_callback_uri_not_claimed

-02

* Clarified wording

* Improved figures

-01

* Better defined terms

* Explained deep link claiming detection on iOS and android

-00

* initial working group version (previously draft-zehavi-oauth-app2app-browserless)

Author’s Address

Yaron Zehavi
Raiffeisen Bank International
Email: yaron.zehavi@rbinternational.com