Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 1 February 2026

Y. Zehavi
Raiffeisen Bank International

31 July 2025

OAuth 2.0 App2App Browser-less Flow
draft-zehavi-oauth-app2app-browserless-05

Abstract

This document describes a protocol allowing a _Client App_ to obtain an OAuth grant from a _Native App_ using the [App2App] pattern, providing *native* app navigation user-experience (no web browser required), despite both apps residing on 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/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 1 February 2026.

Table of Contents

1. 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.

1.1. 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] and [RFC6750] as well as [OpenID], both in their *authorization code flow*.

"PKCE": Proof Key for Code Exchange (PKCE) [RFC7636], a mechanism to prevent various attacks on OAuth authorization codes.

"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 acting as client of _Initial Authorization Server_. In accordance with "OAuth 2.0 for Native Apps" [RFC8252], Client's redirect_uri is claimed by the app.

"Initial Authorization Server": Authorization Server of _Client App_. As an _OAuth Broker_ it is a client of _Downstream Authorization Servers_, to which it federates requests.

"Downstream Authorization Server": An Authorization Server

downstream of _Initial 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 redirect), 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.

"Native Callback uri": _Client App's_ redirect_uri, claimed as a deep link. This deep link is invoked by _User-Interacting App_ to natively return to _Client App_.

2. Introduction

This document, _OAuth 2.0 App2App Browser-less Flow_, describes a protocol enabling native (*Browser-less*) app navigation of an [App2App] OAuth grant across _different Trust Domains_.

When Clients and Authorization Servers are located on _different Trust Domains_, authorization requests are routedusing 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.

However in [App2App] scenarios the web browser must serve as user-agent, because federating 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:

* A *Native App2App Profile* _Authorization Servers_ SHOULD follow to support browser-less App2App flows.

* A new Authorization Server metadata property: *native_authorization_endpoint*, indicating an _Authorization Server_ supports the *Native App2App Profile*.

* A new [RFC9396] Authorization Details Type: *https://scheme.example.org/native_callback_uri*.

* A new error code value: *native_app2app_unsupported*

2.1. App2App across trust domains requires a web browser

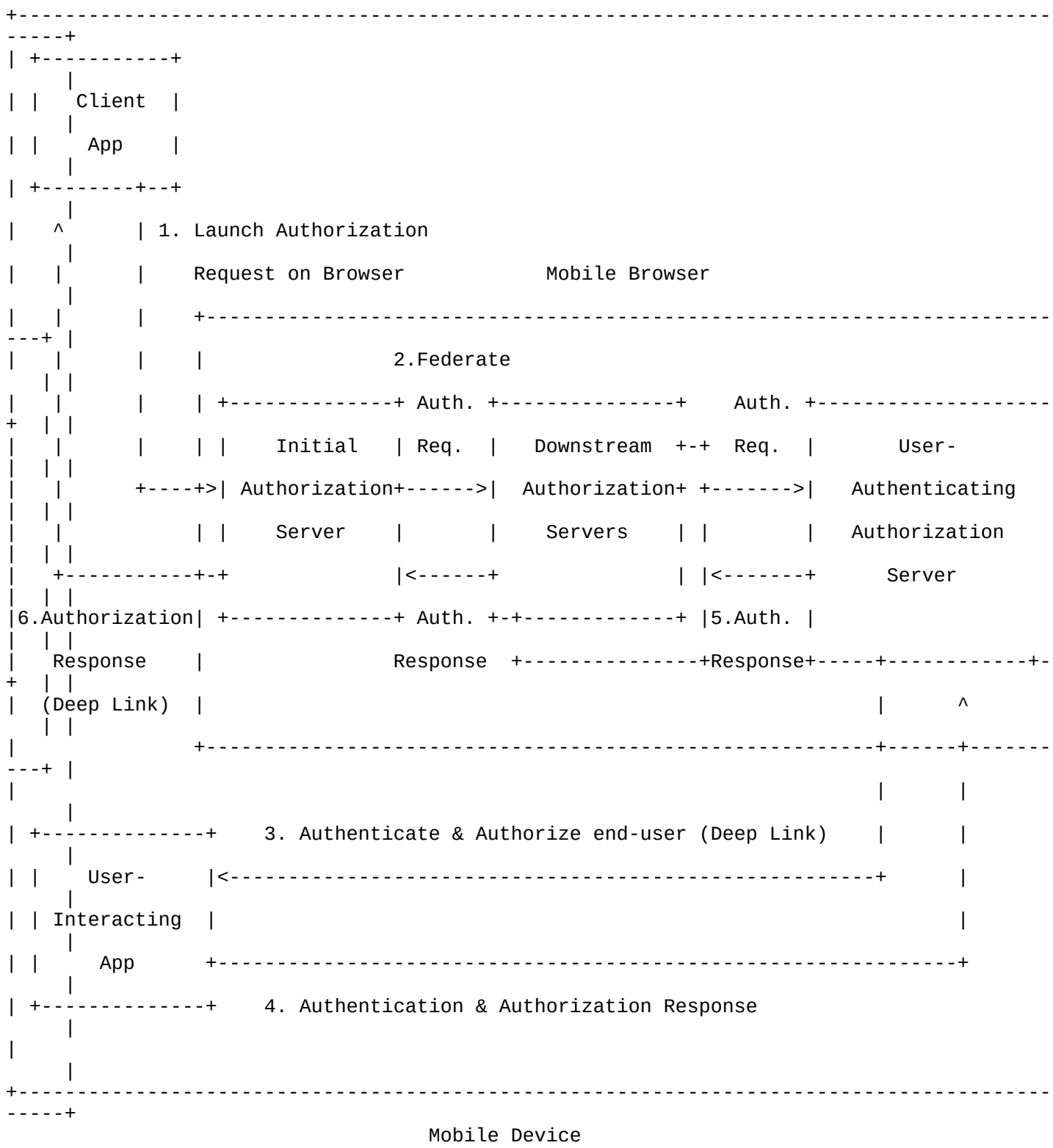

Figure 1: App2App across trust domains using browser

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.

2.2. 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".

2.3. 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.

2.4. 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.

3. Protocol Overview

3.1. Flow Diagram

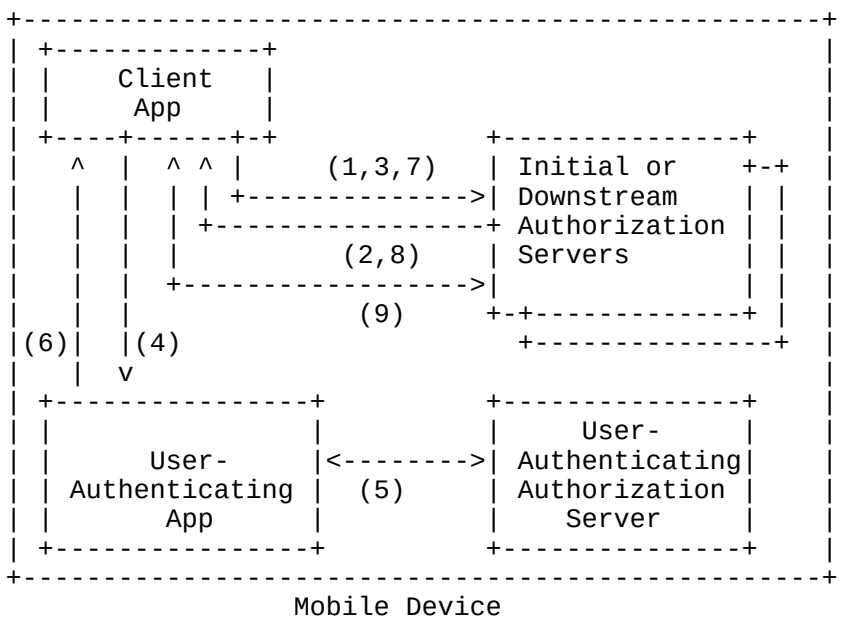

Figure 2: Browser-less App2App across trust domains

* (1) _Client App_ presents an authorization request to _Authorization Server's_ *native_authorization_endpoint*, including the _native_callback_uri_ _Authorization Details Type_.

* (2) _Authorization Server_ returns either a _native authorization request url_ for Downstream Authorization Server which includes the original *native_callback_uri* Authorization Details, or a *Routing Instructions Response*.

* (3) _Client App_ handles obtained _Routing Instructions Response_ by prompting end-user and providing their response to _Authorization Server_, which then responds with a _native authorization request url_. _Client App_ handles obtained _native authorization request urls_ by seeking an app on the device claiming the url. If not found, _Client App_ loops through invocations of obtained _native authorization request urls_, until a claimed url is reached.

* (4) Once a claimed url is reached _Client App_ natively invokes _User-Interacting App_.

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

* (6) _User-Interacting App_ natively invokes *native_callback_uri*, providing as a parameter a url-encoded _redirect_uri_ with its response parameters.

* (7) _Client App_ invokes the obtained _redirect_uri_.

* (8) _Client App_ calls any subsequent uri obtained as 30x redirect directive, until it reaches a location header to its own redirect_uri.

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

3.2. Usage and Applicability

This specification MUST NOT be used when _Client App_ detects _Initial Authorization Server's_ url is claimed by an app on the device.

In such case _Client App_ SHOULD natively invoke the authorization request url.

3.3. Authorization Server Metadata

This document introduces the following authorization server metadata [RFC8414] parameter to indicate it supports the _Native App2App Profile_.

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

3.4. [RFC9396] Authorization Details Type _native_callback_uri_

The protocol described in this document requires *User-Interacting App* to natively navigate end-user back to _Client App_, for which it requires _Client App's_ *native_callback_uri*.

To this end this document defines a new Authorization Details Type:

```
{
   "type": "https://scheme.example.org/native_callback_uri",
   "locations": [
      "https://app.example.com/native_callback_uri"
   ]
}
```

locations array MUST include exactly one instance.

3.5. Native App2App Profile

Authorization servers providing a *native_authorization_endpoint* MUST follow the *Native App2App Profile's* requirements:

* Accept the [RFC9396] Authorization Details Type: *https://scheme.example.org/native_callback_uri*.
* Forward the Authorization Details Type to _Downstream Authorization Servers_.
* Ensure _Downstream Authorization Servers_ it federates to, support the _Native App2App profile_, otherwise return error=native_app2app_unsupported.
* Redirect using HTTP 30x (avoid redirecting using HTTP Form Post or scripts embedded in HTML).
* Avoid challenging end-user with bot-detection such as CAPTCHAs when invoked without cookies.
* MAY provide _Routing Instructions Response_.

3.6. Routing Instructions Response

Authorization servers supporting the _Native App2App profile_, but requiring end-user input to guide request routing, MAY provide a _Routing Instructions Response_.

Example prompting end-user for multiple-choice:

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

{
    "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 prompting end-user for input entry:

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

{
    "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": "Lore 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 supporting _Routing Instructions Response_ identifies the response as such using its Content-Type, then prompts end-user for their input: _logo_ is OPTIONAL and used to brand the interaction and represent the Authorization Server.

userPrompt MUST specify at least _options_ or _inputs_ and MAY specify both.

options specifies 1..n multiple-choice prompts.

inputs specifies free-form input.

Client App provides end-user’s input using _response_ which specifies HTTP GET or POST urls. If provided, _Client App_ includes "id" as interaction identifier.

Example _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 _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
```

3.7. Protocol Flow

3.7.1. Client App calls Authorization Server

Client App uses HTTP to call _Initial Authorization Server's_ _native_authorization_endpoint_ with an authorization request including the _native_callback_uri_ Authorization Details [RFC9396] RAR Type.

3.7.2. Authorization Server

Authorization Server evaluates the native authorization request. It MAY return:

* Error _native_app2app_unsupported_ in case the intended _Downstream Authorization Server_ does not support the _Native App2App Profile_.

* HTTP 200 with a _Routing Instructions Response_, in case it requires user input to guide choosing a _Downstream Authorization Server_.

* HTTP 30x in case the _Downstream Authorization Server_ is known and eligible, with a _native authorization request url_ towards _Downstream Authorization Server_ in the Location header.

3.7.3. Client App processes the response

Client App SHALL terminate the protocol flow if it obtains:

* An error response.

* Or an HTTP 2xx response other than a _Routing Instructions Response_.

* A _Routing Instructions Response_, in case _Client App_ does not support it.

If a _Routing Instructions Response_ was obtained and is supported, _Client App_ interacts with end-user and provides their response to _Authorization Server_.

If an HTTP 30x redirect response was obtained, _Client App_ SHALL use OS SDK's to locate an app claiming the url in the Location header, and if found SHALL natively invoke it to process the _native authorization request_.

If a suitable app is not found, _Client App_ SHALL use HTTP to call the authorization request url and process the response as described herein.

Client App repeats these actions until a native app is reached, or an error occurs.

As the _Client App_ performs HTTP calls, it SHALL maintain a list of all the DNS domains it interacts with, serving as an Allowlist for later invocations as part of the response handling.

As Authorization Servers MAY use Cookies to bind security elements (state, nonce, PKCE) to the user agent, causing the flow to break if required cookies are missing from subsequent HTTP requests, _Client App_ MUST handle cookies:

* Store Cookies it obtains in HTTP responses.

* Send Cookies in subsequent HTTP requests.

3.7.4. Processing by User-Interacting Authorization Server's App:

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

* Validates the native authorization request as an OAuth authorization code request.

* Establishes trust in _native_callback_uri_, otherwise terminates the flow.

* 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*.

3.7.5. Client App traverses Authorization Servers in reverse order

Client App is natively invoked by _User-Interacting Authorization Server App_, with the request's redirect_uri.

Client App MUST validate this url, and any url subsequently obtained via a 30x redirect instruction, against the Allowlist it previously generated, and MUST fail if any url is not included in the Allowlist.

Client App SHALL invoke urls it received using HTTP GET:

* If the response is a redirect instruction (HTTP Code 30x + Location header), _Client App_ SHALL proceed to call obtained urls until reaching its own redirect_uri.

* SHALL handle any other HTTP code (2xx / 4xx / 5xx) as a failure and terminate the flow.

3.7.6. Client App obtains response

Once _Client App's_ own redirect_uri is reached, the traversal of _Authorization Servers_ is complete and _Client App_ proceeds to process the response:

* Exchange code for tokens.

* Or handle errors obtained.

3.7.7. Recovery from failed native App2App flows

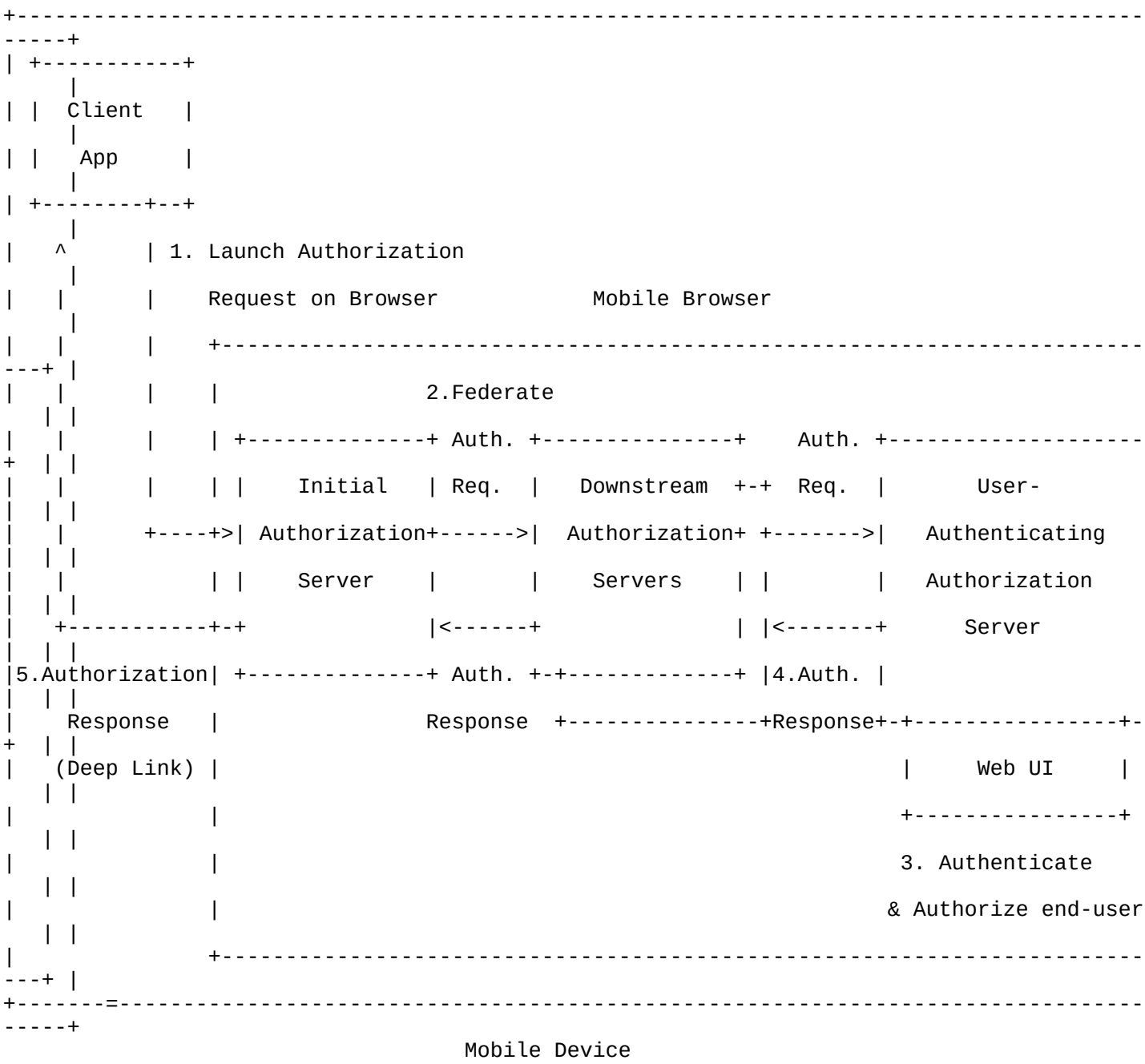

Figure 3: App2Web using the browser

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

* An error response is obtained.

* _Client App_ doesn’t support an obtained _Routing Instructions Response_.

* An HTTP 2xx response other than a _Routing Instructions Response_ is obtained.

In case of such failures, _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 an HTTP 2xx response, other than a _Routing Instructions Response_ was obtained, suggests the _User-Interacting App_ is not installed on end-user’s device. _Client App_ MAY choose in future to retry the _Native App2App_ flow, to benefit if in the meantime the missing app has been installed.

4. Detecting Presence of Native Apps claiming Urls

Native Apps on iOS and Android MAY use OS SDK's to detect if an app owns a url. The general method is the same - App calls an SDK to open the url as deep link and handles an exception thrown if no matching app is found.

4.1. Android

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

4.2. 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.

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.

This risk does not apply to this draft as _Client App_ acts as User Agent only for the purpose of flow redirection, and does not interact with end-user's credentials in any way.

The mechanism for providing routing instructions MUST NOT be used to request end-user to provide any authentication credentials.

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 sepcific trust establishment mechanisms are outside the scope of this document. For example purposes, one way to validate could leverage [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. OAuth request forgery and manipulation

It is RECOMMENDED that _Client App_ acts as a confidential OAuth client.

5.4. Secure Native application communication

If _Client App_ uses a Backend it is RECOMMENDED to communicate with it securely:

* Use TLS in up to date versions and ciphers.

* Use DNSSEC.

* Perform certificate pinning.

5.5. 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.

5.6. 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.

5.7. 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] Sakimura, N., Bradley, J., Jones, M. B., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[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>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/rfc/rfc6750>.

[RFC7636] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <https://www.rfc-editor.org/rfc/rfc7636>.

[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. 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 B. Document History

[[To be removed from the final specification]]

-latest * 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