Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 9 May 2026

K. Meyer zu Selhausen
Hackmanit
L. Jannett
Ruhr University Bochum
C. Mainka
Hackmanit

5 November 2025

OAuth 2.0 Web Message Response Mode for Popup- and Iframe-based Authorization Flows

draft-meyerzuselha-oauth-web-message-response-mode-01

Abstract

This specification defines the web message response mode that authorization servers use for transmitting authorization response parameters via the user-agent's postMessage API to the client. This mode is intended for authorization flows that use secondary windows, which are well-suited for browser-based applications.

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 9 May 2026.

Copyright Notice

Table of Contents

1. Introduction

OAuth [RFC6749] uses HTTP redirects to transfer authorization response parameters from the authorization server via the user-agent to the client's redirection endpoint. In this case, the authorization response parameters are encoded in the query string (response_mode=query) or in the fragment (response_mode=fragment) of the redirect_uri [oauth.encoding] (Section 2.1). [RFC6749] (Section 1.7) allows other mechanisms available via the user-agent to accomplish this redirection, such as response_mode=form_post [oauth.post].

The standardized query, fragment, and form post response modes are designed for single-window authorization but not for multi-window authorization flows. A common example is a popup-based authorization flow, where the client's primary window opens the authorization server in a secondary window. The secondary window cannot use HTTP redirects to transfer response parameters back to the client in the primary window.

This specification defines the web message response mode that uses the user-agent's postMessage API [whatwg.postmessage] to exchange messages between two different browser windows. Clients inform the authorization server to use this response mode for returning the authorization response by setting the response_mode parameter in the authorization request to web_message. This response mode facilitates popup-based and iframe-based authorization flows, in which the authorization server is called in a secondary window or embedded in a frame on the client.

These flows that span the authorization process across two windows are especially useful for browser-based applications, where the main application window should remain uninterrupted to preserve the user experience. Executing authorization flows in invisible iframes can also enable seamless session resumption and renewal without disrupting the user.

1.1. Conventions and Terminology

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.

This specification uses the terms "client", "user-agent", "authorization server", "authorization endpoint", "redirection endpoint", "redirection URI", "authorization request", and "authorization response" defined by the OAuth 2.0 Authorization Framework [RFC6749]. It further uses the term "response mode" defined by the OAuth 2.0 Multiple Response Type Encoding Practices [oauth.encoding].

This specification defines the following additional terminology.

primary window This is the top-level browsing window that initially holds the client's website. This window has no parent windows and it was not opened by any other window.

secondary window This is the window in which the client in the primary window opens the authorization server.

2. Web Message Response Mode

This specification defines the web message response mode, which is described with the following response_mode parameter value in the authorization request [oauth.encoding].

web_message In the web message response mode, the authorization server in the secondary window encodes the authorization response parameters in a JSON dictionary and transmits it via the user-agent's postMessage API to the client in the primary window.

If the authorization request includes the value web_message for the response_mode parameter, the authorization server:

* MUST dynamically discover the client's primary window reference as described in Section 2.1.
* MUST use the user-agent's postMessage API to return the authorization response to the client's primary window.
* MUST encode the response paramters as key-value string pairs in a JSON dictionary.
* MUST follow [RFC6749] and [RFC9700] when validating the redirection URI.
* MUST use the full redirection URI (the exact same URI also used in other response modes such as query, fragment, or form post) as the postMessage's receiver origin. The user-agent's postMessage API inherently parses the redirection URI and extracts its origin.
* MUST NOT parse the URI itself to reduce the attack surface concerning parsing issues.

The client's redirection URI MUST serve as the postMessage's receiver origin to protect the authorization response from being leaked to malicious origins.

This example illustrates how an authorization server (identified by the issuer https://as.example) in a secondary window returns the authorization response to the client (whose registered redirection URI is https://client.example/cb) in the primary window using the postMessage API:

```
const primaryWindowRef = window.opener // for popup-based flow
const primaryWindowRef = window.parent // for iframe-based flow
const params = {
  "code": "XXXXXXXX",
  "state":"XXXXXXXX",
  "iss": "https://as.example"
}
primaryWindowRef.postMessage(params, "https://client.example/cb")
```

The client MAY signal its preferred primary window reference by using the response_mode parameter. For popup-based authorization flows, it MAY set the value to web_message.opener, and for iframe-based authorization flows, it MAY set the value to web_message.parent. When either web_message.opener or web_message.parent is specified, the authorization server MUST use the corresponding window reference window.opener or window.parent and MUST NOT attempt to dynamically determine the primary window reference as specified in Section 2.1.

2.1. Dynamic Discovery of Primary Window Reference

If the authorization request specifies web_message as the response_mode, the authorization server running in the secondary window MUST dynamically determine the client's primary window reference. This is done by checking whether the window.opener or window.parent references point to a valid primary window. If both references are available, the window.opener reference MUST take precedence to ensure that popup-based authorization flows are prioritized over iframe-based flows.

The following example illustrates how an authorization server may perform this discovery process:

```
let primaryWindowRef = undefined
if (window.opener !== null) // this is a popup-based flow
  primaryWindowRef = window.opener
else if (window.parent !== self) // this is an iframe-based flow
  primaryWindowRef = window.parent
else // abort flow: could not discover primary window reference
```

3. Popup-Based Authorization Flow Using the Web Message Response Mode

In a popup-based authorization flow, the client opens the authorization endpoint in a secondary window. The authorization server uses the user-agent's postMessage API to return the authorization response parameters from the secondary window back to the client running in the primary window. The flow is depicted in Figure 1 and described in the following.

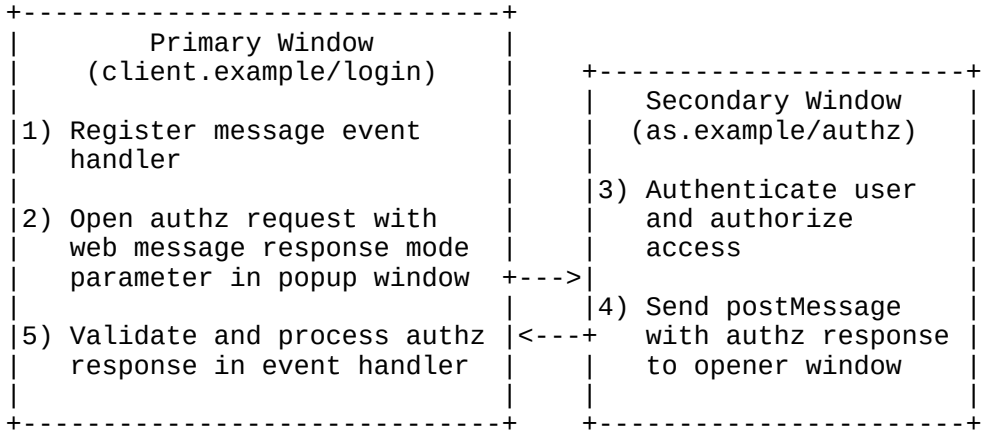

Figure 1: Overview of a popup-based authorization flow using the web message response mode.

1. The client registers a message event handler that receives the authorization response from the authorization server. The client MUST validate the message's origin with an exact string matching the authorization server's origin. As the authorization response is meant for one-time use, the client MUST remove the message event handler after receiving the authorization response to prevent any further authorization attempts.

Example of the registration of a message event handler:

```
const callback = (e) => {
  if (e.origin === "https://as.example") {
    // further validation and processing of the authorization response
    process(e.data)
    window.removeEventListener("message", callback)
  }
}
window.addEventListener("message", callback)
```

2. The client MUST open the authorization request in a secondary window. Therefore, the client MAY use JavaScript and the user-agent's window.open API, MAY use an anchor HTML element with a target attribute, or MAY use any other mechanism suitable for opening secondary windows. The authorization request MUST include the response_mode parameter value web_message or web_message.opener. Additional authorization request parameters and techniques, such as PAR [RFC9126] and RAR [RFC9396], are unaffected by this specification and MAY be used with the web message response mode.

Example of using the window.open API to open the authorization request in a secondary window:

```
window.open(
  "https://as.example/auth?...&response_mode=web_message",
  "secondaryWindowRef",
  "left=100,top=100,width=320,height=320"
)
```

Example of using an anchor tag with a target attribute to open the authorization request in a secondary window:

```
<a
  href="https://as.example/auth?...&response_mode=web_message"
  target="secondaryWindowRef"
>
```

3. The authorization server receives the authorization request, validates its parameters, and proceeds with the end-user authentication and authorization, which is outside of the scope of this specification.

4. If the authorization request includes the response_mode parameter value web_message or web_message.opener, the authorization server MUST follow the web message response mode as described in Section 2 and use the user-agent's postMessage API to return the authorization response parameters from the secondary window to the secondary window's opener window. The receiver window MUST be referenced by the secondary window's opener property.

Example of an authorization server using the postMessage API in a secondary window to return the authorization response to the client in the primary window:

```
const params = {
  "code": "XXXXXXXX",
  "state": "XXXXXXXX",
  "iss": "https://as.example"
}
window.opener.postMessage(params, "https://client.example/cb")
```

5. The client's message event handler receives the postMessage that contains the authorization response parameters. The further processing and validation of all parameters is out of the scope of this specification and MUST be compliant to [RFC6749] and [RFC9700].

4. Iframe-Based Authorization Flow Using the Web Message Response Mode

In addition to secondary windows being popups, iframes can be used as well. Iframes enable a seamless authorization flow where the end-user only sees a single browser tab, without ever leaving the actual client. The client MAY explicitly request the iframe-based authorization flow by setting the response_mode parameter to web_message.parent. The authorization flow proceeds in a manner technically similar to the popup-based flow described in Section 3, with the following modifications:

* In step 2, the client MUST embed the authorization request in an iframe instead of opening a popup window.
* In step 4, the secondary window MUST reference its parent window rather than its opener window.

If user consent has already been granted, iframe-based authorization flows can enable seamless session resumption or renewal by embedding invisible iframes with the prompt parameter set to none [openid.core] (Section 3.1.2.1). In this case, the client receives the authorization response directly, allowing the user session to resume without further interaction. If the prompt parameter is set to none but no prior user authentication or consent exists, the authorization server MUST return an error as specified in [openid.core] (Section 3.1.2.1). In this case, the client can fall back to a popup-based flow or to an iframe-based flow that displays a Clickjacking-protected user interface, as discussed in Section 4.1.

4.1. Clickjacking Mitigations in Iframes

The authorization server MUST implement Clickjacking countermeasures according to [RFC9700] (Section 4.16) and protect its authorization endpoint and consent-page from being embedded in an iframe by origins not deemed trustworthy.

If the user’s browser supports the Intersection Observer v2 API [w3c.observerv2], the authorization server MAY use it to allow the client to embed the authorization endpoint and consent-page. For example, the authorization server’s script running in the embedded consent-page could pre-emptively load itself in a popup window if it detects the Intersection Observer v2 API’s isVisible attribute being set to false, thus avoiding any occlusion of its content.

4.2. User-Session in Iframes

Modern browsers have started to disable the support for third-party cookies. Thereby, iframes do not send authentication cookies along with requests in sub resource requests, such as iframes. Using iframes as secondary windows therefore requires special exceptions to bypass this restriction, such as the Storage Access API [mozilla.storageaccessapi].

This mechanism is unnecessary in first-party scenarios where the authorization server and client share the same origin. In such cases, cookies are treated as first-party and remain included in requests made from iframes.

5. Authorization Server Metadata

Authorization servers MUST announce their support for the web message response mode defined in Section 2 by adding web_message, web_message.opener, web_message.parent, or any combinations of these values to the response_modes_supported list in their authorization server metadata as specified in [RFC8414] (Section 2). Authorization servers MAY declare exclusive support for a specific authorization flow by including either web_message.opener or web_message.parent in the response_modes_supported list. If the response_modes_supported list contains the generic value web_message, the authorization server MUST support both the popup-based and the iframe-based authorization flows.

6. Security Considerations

6.1. Receiver Origin Validation

Authorization servers MUST follow Section 2 and [RFC9700] (Section 4.17.2) when validating the postMessage receiver origin. Otherwise, the authorization response may leak to an attacker, as described in [RFC9700] (Section 4.17.1.1) and [RFC9700] (Section 4.17.1.2).

6.2. Initiator Origin Validation and Cross-Site Request Forgery Protection

In redirect-based authorization flows, there is no inherent mechanism available that enables a client to verify that the trusted authorization server initiates a redirection. Instead, [RFC6749] introduces the state parameter to counter so-called Cross-Site Request Forgery (CSRF) attacks [RFC9700] (Section 4.7) against the client’s redirection endpoint by maintaining a state between the authorization request and response.

The postMessage API provides an inherent mechanism to verify the initiator of a postMessage. The client MUST use this mechanism as described in Section 2 and [RFC9700] (Section 4.17.2) to verify that the trusted authorization server is the initiator of the postMessage. Otherwise, an attacker can inject a maliciously crafted authorization response to the client [RFC9700] (Section 4.17.1.3). This verification prevents some variants of CSRF attacks, where the attacker wants to log in a victim to the attacker's account.

However, attackers can also use CSRF attacks to log in a victim to the victim's own account. This attack variant cannot be mitigated with the checks mentioned above. Instead, a proper CSRF countermeasure, as described in [RFC9700] (Section 4.7.1) MUST be used.

6.3. Data Validation

Even after the initiator origin of the postMessage is validated, the client MUST check that the postMessage has the expected format [whatwg.postmessage] (Section 9.3.2.1). In particular, the postMessage MUST NOT be processed in unsafe JavaScript sinks like eval or innerHTML to prevent cross-site scripting (XSS) flaws and other potentially malicious injections. Otherwise, if the authorization server has been attacked using an XSS flaw, further unchecked processing of the postMessage could result in the attack being propagated into the client.

6.4. Cross-Site Leak Protections on the Authorization Server

The authorization server is opened in a secondary window that needs access to its opener window via its opener property to send postMessages to that referenced window. Thus, the authorization server cannot use cross-site leak protections like the cross-origin opener policy [whatwg.coop] to force the creation of a new top-level browsing context and cross-origin isolate their sites.

However, browsers are working on preventing cross-site leaks without breaking the popup-based authorization flows with the restrict-properties directive being added to the cross-origin opener policy [google.restrictprops]. With this directive, properties that can be used for cross-site leaks are not available but postMessage communication between cross-origin windows is still allowed. Authorization servers MAY set the Cross-Origin-Opener-Policy: restrict-properties header to protect against cross-site leaks.

6.5. Redirection URI vs. Receiver Origin

In redirect-based authorization flows, the confidentiality of the authorization response is scoped to the redirection URI that contains a path. The path separates the authorization response from other, potentially vulnerable paths within the same web application. In flows using the web message response mode, the confidentiality of the authorization response is scoped to the postMessage's receiver origin that does not contain a path. Thus, cross-site scripting (XSS) vulnerabilities on _any_ path within the web application's origin will leak the authorization response to the attacker.

7. IANA Considerations

This draft makes no requests to IANA.

8. Normative References

[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/info/rfc2119>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/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/info/rfc8174>.

[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/info/rfc8414>.

[RFC9700] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "Best Current Practice for OAuth 2.0 Security", BCP 240, RFC 9700, DOI 10.17487/RFC9700, January 2025, <https://www.rfc-editor.org/info/rfc9700>.

[oauth.encoding] de Medeiros, B., Scurtescu, M., Tarjan, P., and M. Jones, "OAuth 2.0 Multiple Response Type Encoding Practices", February 2014, <https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html>.

[whatwg.postmessage]
"HTML Living Standard: Cross-document messaging", <https://html.spec.whatwg.org/multipage/web-messaging.html#web-messaging>.

9. Informative References

[I-D.sakimura-oauth-wmrm]
Yamaguchi, T., Sakimura, N., and N. Matake, "OAuth 2.0 Web Message Response Mode", Work in Progress, Internet-Draft, draft-sakimura-oauth-wmrm-01, 8 November 2023, <https://datatracker.ietf.org/doc/html/draft-sakimura-oauth-wmrm-01>.

[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/info/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/info/rfc9396>.

[google.restrictprops]
and , "Secure popup interactions with restrict-properties", 9 August 2023, <https://developer.chrome.com/blog/coop-restrict-properties/>.

[mozilla.storageaccessapi]
"Storage Access API", 18 October 2023, <https://developer.mozilla.org/en-US/docs/Web/API/Storage_Access_API>.

[oauth.post]
Jones, M. and B. Campbell, "OAuth 2.0 Form Post Response Mode", 27 April 2015, <http://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html>.

[openid.core]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[w3c.observerv2]
and , "Intersection Observer", 4 November 2019,
<https://w3c.github.io/IntersectionObserver/>.

[whatwg.coop]
"HTML Living Standard: Cross-origin opener policies",
<https://html.spec.whatwg.org/multipage/
browsers.html#cross-origin-opener-policies>.

Appendix A. Acknowledgements

We would like to acknowledge the prior work of Toru Yamaguchi, Nat Sakimura, and Nov Matake in [I-D.sakimura-oauth-wmrm] which tried to define a response mode with similarities to this specification. In contrast, this specification is not focused on iframes and does not include the use of the OAuth Implicit Grant.

We would like to thank Vladislav Mladenov, ...

for their valuable feedback on this document.

Appendix B. Document History

[[To be removed from the final specification]]

-01 * Add use cases of popup and iframe flows * Enhance and clarify descriptions and mitigations for iframe flow * Clarify iframe use for first-party context and invisible iframes for reauthentication flows with prompt none * Add dynamic discovery of primary window reference and new values for response_mode: web_message.opener and web_message.parent * Add notes about how this draft differs from Sakimura et al. * Add several example implementations of the web message response mode and popup- or iframe-based flows * Minor improvements of style and language * Update references to newer versions

-00 * Initial draft

Appendix C. Example Implementations

[[To be removed from the final specification]]

Although neither the web_message response mode nor the popup- and iframe-based authorization flows have been officially standardized, they are already widely supported in practice. A study (https://dl.acm.org/doi/10.1145/3548606.3560692) from late 2022 found that 153 of the Tranco Top 1k websites opened the authorization server in a secondary window, compared to only 134 websites that

navigated the primary window to the authorization server. More recently, a scan of all domains in the CrUX dataset (July 2025) revealed 22 authorization servers (https://github.com/Hackmanit/draft-meyerzuselha-oauth-web-message-response-mode/issues/27) that publish the still-undefined web_message value in the response_modes_supported list of their OpenID Provider Metadata (https://openid.net/specs/openid-connect-discovery-1_0.html) at /.well-known/openid-configuration. In addition, several authorization servers support popup- or iframe-based flows without explicitly announcing them in their metadata:

* Apple supports the popup-based flow with response_mode=web_message through its custom JS SDK. The flow is activated if the client enables the SDK's popup mode via a meta tag (https://developer.apple.com/documentation/signinwithapple/configuring-your-webpage-for-sign-in-with-apple). Apple's REST API documentation (https://developer.apple.com/documentation/signinwithapplerestapi/request-an-authorization-to-the-sign-in-with-apple-server) does not mention web_message as a valid response_mode, although the SDK uses it when configured.

* Google supports both the popup-based (https://developers.google.com/identity/gsi/web/guides/integrate#pop-up_versus_redirect) (default) and iframe-based (https://developers.google.com/identity/gsi/web/amp/intermediate-iframe) flows within its Google Identity Services SDKs (https://developers.google.com/identity/gsi/web/guides/overview). However, these flows are not available in its interoperable OAuth and OpenID Connect endpoints (https://developers.google.com/identity/protocols/oauth2/web-server).

* Facebook supports the popup-based flow through its JS SDK (https://developers.facebook.com/docs/facebook-login/web). It always includes the xd_arbiter endpoint as a parameter to the authorization endpoint, e.g., channel_url=https://staticxx.facebook.com/x/connect/xd_arbiter.

* Auth0 officially documents support for the web_message response mode in its API documentation (https://auth0.com/docs/authenticate/protocols/oauth). It references the expired draft [I-D.sakimura-oauth-wmrm] from 2015 and uses it for silent authentication with HTML5 web messaging.

* Nintendo uses the web_message response mode internally, following the expired draft [I-D.sakimura-oauth-wmrm]. It also relies on the additional web_message_uri and web_message_target parameters.

Beyond these providers, other authorization servers also employ web_message or proprietary variants of popup- and iframe-based flows. Apple, Google, and Facebook even make multi-window flows the default within their SDKs, highlighting their practical importance. Yet, the absence of an interoperable, standards-based mechanism for clients to use these flows underscores the need for formal standardization.

Appendix D. Differences to [I-D.sakimura-oauth-wmrm]

[[To be removed from the final specification]]

This draft builds on the earlier work of Toru Yamaguchi, Nat Sakimura, and Nov Matake in [I-D.sakimura-oauth-wmrm], which introduced a web message response mode. While sharing several conceptual similarities, this draft differs in the following ways:

* *Flow illustrations:* [I-D.sakimura-oauth-wmrm] illustrates only the deprecated implicit flow (response_type=token). This draft is flow-agnostic and uses the authorization code flow (response_type=code) in its examples.

* *Primary window reference:* [I-D.sakimura-oauth-wmrm] does not specify how the authorization server should obtain a reference to the client’s primary window. This draft allows the authorization server to either dynamically discover the reference or rely on client-provided hints via the web_message.opener and web_message.parent response modes.

* *Relay mode:* [I-D.sakimura-oauth-wmrm] defines a relay mode in which the authorization response is forwarded through the client to a third window (the "message target window"). This draft does not support tertiary windows and instead limits the exchange to the client’s primary window and the authorization server’s secondary window. This draft omits relay mode due to lack of practical demand.

* *Browser compatibility:* [I-D.sakimura-oauth-wmrm] uses JavaScript constructs that no longer work in modern browsers (e.g., window.opener != window and cross-origin DOM access via evt.source.document.getElementById). These break relay mode entirely, as the secondary window cannot obtain a reference to the message target window.

* *Parameters:* [I-D.sakimura-oauth-wmrm] introduces the web_message_uri (target URI) and web_message_target (DOM id) parameters, both tied to relay mode. This draft omits these parameters and instead relies solely on the established redirect_uri.

* *Message format:* [I-D.sakimura-oauth-wmrm] defines postMessage data as an object with fields type (string) and response (object). This draft simplifies the structure to a single object containing all authorization response parameters, since only one postMessage is used.

* *Iframe restrictions:* [I-D.sakimura-oauth-wmrm] restricts iframe-based flows to already-authenticated sessions with prior consent ("authenticated windows"). This draft permits iframe-based flows in unauthenticated states as well, relying on Clickjacking mitigations (e.g., the Intersection Observer v2 API).

* *Security considerations:* This draft provides extended guidance on secure use of postMessage, Clickjacking mitigations, and implications of browsers phasing out third-party cookies.

Authors' Addresses

Karsten Meyer zu Selhausen
Hackmanit
Email: karsten.meyerzuselhausen@hackmanit.de

Louis Jannett
Ruhr University Bochum
Email: louis.jannett@rub.de

Christian Mainka
Hackmanit
Email: christian.mainka@hackmanit.de