Web Authorization Protocol
Internet-Draft
Updates: 6749, 6750, 7521, 7522, 7523, 9700 (if approved)
Intended status: Best Current Practice
Expires: 3 September 2026

T. W端rtele
P. Hosseyni
University of Stuttgart
K. Luo
The Chinese University of Hong Kong
A. Fung
Samsung Research America

2 March 2026

Updates to OAuth 2.0 Security Best Current Practice
draft-ietf-oauth-security-topics-update-01

Abstract

This document updates the set of best current security practices for OAuth 2.0 by extending the security advice given in RFC 6749, RFC 6750, and RFC 9700, to cover new threats that have been discovered since the former documents have been published.

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://SECtim.github.io/draft-wuertele-oauth-security-topics-update/draft-ietf-oauth-security-topics-update.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-oauth-security-topics-update/.

Discussion of this document takes place on the Web Authorization Protocol 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/SECtim/draft-wuertele-oauth-security-topics-update.

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

Since the publication of the first OAuth 2.0 Security Best Practices document [RFC9700], new threats to OAuth 2.0 ecosystems have been identified. This document therefore serves as an extension of the original [RFC9700] and is to be read in conjunction with it.

Like [RFC9700] before, this document provides important security recommendations and it is RECOMMENDED that implementers upgrade their implementations and ecosystems as soon as feasible.

1.1. Structure

The remainder of this document is organized as follows: Section 2 is a detailed analysis of the threats and implementation issues that can be found in the wild (at the time of writing) along with a discussion of potential countermeasures.

1.2. 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 "access token", "authorization endpoint", "authorization grant", "authorization server", "client", "client identifier" (client ID), "protected resource", "refresh token", "resource owner", "resource server", and "token endpoint" defined by OAuth 2.0 [RFC6749].

// Make sure to update this list once the technical sections below
// are completed.
//
// -- Tim W.

2. Attacks and Mitigations

This section gives a detailed description of new attacks on OAuth implementations, along with potential countermeasures. Attacks and mitigations already covered in [RFC9700] are not listed here, except where clarifications or new recommendations are made.

2.1. Audience Injection Attacks

When using signature-based client authentication methods such as private_key_jwt as defined in [OpenID.Core] or signed JWTs as defined in [RFC7521] and [RFC7523], a malicious authorization server may be able to obtain and use a client's authentication credential, enabling them to impersonate a client towards another honest authorization server.

2.1.1. Attack Description

The descriptions here follow [research.ust], where additional details of the attack are laid out. Audience injection attacks require a client to interact with at least two authorization servers, one of which is malicious, and to authenticate to both with a signature-based authentication method using the same key pair. The following description uses the jwt-bearer client authentication from [RFC7523], see Section 2.1.1.3 for other affected client authentication methods. Furthermore, the client needs to be willing to authenticate at an endpoint other than the token endpoint at the attacker authorization server (see Section 2.1.1.2).

2.1.1.1. Core Attack Steps

In the following, let H-AS be an honest authorization server and let A-AS be an attacker-controlled authorization server.

Assume that the authorization servers publish the following URIs for their token endpoints, for example via mechanisms such as authorization server metadata [RFC8414] or OpenID Discovery [OpenID.Discovery]. The exact publication mechanism is not relevant, as audience injection attacks are also possible on clients with manually configured authorization server metadata.

Excerpt from H-AS' metadata:

```
"issuer": "https://honest.com",
"token_endpoint": "https://honest.com/token",
...
```

Excerpt from A-AS' metadata:

```
"issuer": "https://attacker.com",
"token_endpoint": "https://honest.com/token",
...
```

Therefore, the attacker authorization server claims to use the honest authorization server's token endpoint. Note that the attacker authorization server does not control this endpoint. The attack then commences as follows:

1. Client registers at H-AS, and gets assigned a client ID cid.

2. Client registers at A-AS, and gets assigned the same client ID cid. Note that the client ID is not a secret (Section 2.2 of [RFC6749]).

Now, whenever the client creates a client assertion for authentication to A-AS, the assertion consists of a JSON Web Token (JWT) that is signed by the client and contains, among others, the following claims:

```
"iss": "cid",
"sub": "cid",
"aud": "https://honest.com/token"
```

Due to the malicious use of H-AS' token endpoint in A-AS' authorization server metadata, the aud claim contains H-AS' token endpoint. Recall that both A-AS and H-AS registered the client with client ID cid, and that the client uses the same key pair for authentication at both authorization servers. Hence, this client assertion is a valid authentication credential for the client at H-AS.

The use of the token endpoint to identify the authorization server as a client assertion's audience even for client assertions that are not sent to the token endpoint is encouraged, or at least allowed by many standards, including [RFC7521], [RFC7522], [RFC7523], [RFC9126], [OpenID.Core], [OpenID.CIBA], and all standards referencing the IANA registry for OAuth Token Endpoint Authentication Methods for available client authentication methods.

As described in [research.ust], the attacker can then utilize the obtained client authentication assertion to impersonate the client and, for example, obtain access tokens.

2.1.1.2. Endpoints Requiring Client Authentication

As mentioned above, the attack is only possible if the client authenticates to an endpoint other than the token endpoint at A-AS. This is because if the client sends a token request to A-AS, it will use A-AS' token endpoint as published by A-AS and hence, send the token request to H-AS, i.e., the attacker cannot obtain the client assertion.

As detailed in [research.ust], the attack is confirmed to be possible if the client authenticates with such client assertions at the following endpoints of A-AS:

* Pushed Authorization Endpoint (see [RFC9126])

* Token Revocation Endpoint (see [RFC7009])

* CIBA Backchannel Authentication Endpoint (see [OpenID.CIBA])

* Device Authorization Endpoint (see [RFC8628])

Note that this list of examples is not exhaustive. Hence, any client that might authenticate at any endpoint other than the token endpoint SHOULD employ countermeasures as described in Section 2.1.2.

2.1.1.3. Affected Client Authentication Methods

The same attacks are possible for the private_key_jwt client authentication method defined in [OpenID.Core], as well as instantiations of client authentication assertions defined in [RFC7521], including the SAML assertions defined in [RFC7522].

Furthermore, a similar attack is possible for jwt-bearer authorization grants as defined in Section 2.1 of [RFC7523], albeit under additional assumptions (see [research.ust] for details).

2.1.2. Countermeasures

At its core, audience injection attacks exploit the fact that, from the client’s point of view, an authorization server’s token endpoint is a mostly opaque value and does not uniquely identify an authorization server. Therefore, an attacker authorization server may claim any URI as its token endpoint, including, for example, an honest authorization server’s issuer identifier. Hence, as long as a client uses the token endpoint as an audience value when authenticating to the attacker authorization server, audience injection attacks are possible. Therefore, audience injection attacks need to be prevented by the client.

Note that the following countermeasures mandate the use of single audience value (as opposed to multiple audiences in array). This is because Section 4.1.3 of [RFC7519] allows the receiver of an audience-restricted JWT to accept the JWT even if the receiver identifies with only one of the values in such an array.

Clients that interact with more than one authorization server and authenticate with signature-based client authentication methods MUST employ one of the following countermeasures, unless audience injection attacks are mitigated by other means, such as using fresh key material for each authorization server.

Note that the countermeasures described in Section 2.1.2.1 and Section 2.1.2.2 do not imply any normative changes to the authorization server: Section 4.1.3 of [RFC7519] requires the authorization server to only accept a JWT if the authorization server can identify itself with (at least one of the elements in) the JWT's audience value. Authentication JWTs produced by a client implementing one of these countermeasures meet this condition. Of course, an authorization server MAY still decide to only accept its issuer identifier (Section 2.1.2.1) or the endpoint that received the JWT (Section 2.1.2.2) as an audience value, for example, to force its clients to adopt the respective countermeasure.

2.1.2.1. Authorization Server Issuer Identifier

Clients MUST use the authorization server's issuer identifier as defined in [RFC8414]/[OpenID.Discovery] as the sole audience value in client assertions. Clients MUST retrieve and validate this value as described in Section 3.3 of [RFC8414]/Section 4.3 of [OpenID.Discovery].

For jwt-bearer client assertions as defined by [RFC7523], this mechanism is also described in [OAUTH-7523bis].

Note that "issuer identifier" here does not refer to the term "issuer" as defined in Section 4.4 of [RFC9700], but to the issuer identifier defined in [RFC8414] and [OpenID.Discovery]. In particular, the issuer identifier is not just "an abstract identifier for the combination of the authorization endpoint and token endpoint".

2.1.2.2. Exact Target Endpoint URI

Clients MUST use the exact endpoint URI to which a client assertion is sent as that client assertion's sole audience value.

This countermeasure can be used for authorization servers that do not use authorization server metadata [RFC8414] or OpenID Discovery [OpenID.Discovery].

2.2. Cross-toolkit OAuth Account Takeover

It is increasingly common and observed that a single OAuth client supports multiple tools. Each set of tools, known as a toolkit, is mapped to an OAuth provider configuration, which includes at least the authorization server (AS) endpoints and client registration. A successful OAuth connection is established when the OAuth client obtains an access token for a tool based on its corresponding OAuth provider configuration. The tool can then use the access token to access the user's protected resource at a resource server (RS).

Multiple OAuth connections can be linked to some form of user identity based on the following common deployment scenarios:

* Application Integration: The OAuth connections made with different toolkits are linked to an application's user account or session (e.g., represented by an application's user identifier or a short-lived anonymous session). This is common where a user authorizes an application (e.g., a cloud platform or an agentic AI service) to orchestrate multiple tools, some of which together with their OAuth providers can be contributed by the public.

* Multi-tenant OAuth-as-a-Service (also known as Token Vault): In cases where OAuth responsibilities of a client are managed by a multi-tenant OAuth-as-a-Service provider, a successful OAuth connection is linked to a tenant's user identifier in addition to the tenant identifier. This is a generalization of the last deployment scenario, where an application using this OAuth-as-a-Service is becoming a tenant. A tenant can usually choose some off-the-shelf toolkits using (partially-) completed OAuth providers, if not adding their own toolkits with custom OAuth providers to support the tenant's service.

When controlled by an attacker, the open configurations of OAuth providers have posed a new threat to this centralized OAuth client design. If the client fails to properly identify, track, and isolate which proper OAuth connection context (representing a combination of OAuth provider, toolkit, and tenant) is in use during an authorization flow, an attacker can exploit this to mount Cross-toolkit OAuth Account Takeover (COAT) attacks (see [research.cuhk] and [research.cuhk3]). The COAT attacker uses a malicious toolkit to steal a victim's authorization code issued by an honest OAuth provider of an honest toolkit, and applies the authorization code injection (as defined in Section 4.5 of [RFC9700]) against a new OAuth connection with the attacker's identity. This results in a compromised OAuth connection between the attacker's application identity and the victim's toolkit access. The impact is equivalent to an account takeover: the attacker can operate the honest toolkit

with the victim’s account (hijacked either under the same application, or even cross-tenant that shares a vulnerable OAuth-as-a-service).

2.2.1. Attack Description

Preconditions: It is assumed that

* the implicit or authorization code grant is used with multiple OAuth connection contexts, of which one is considered "honest" (H-Toolkit using H-AuthProvider with H-AS) and one is operated by the attacker (A-Toolkit using A-AuthProvider with A-AS), and

* the client stores the connection context chosen by the user in a session bound to the user’s browser, and

* the authorization servers properly check the redirection URI by enforcing exact redirection URI matching (otherwise, see Cross Social-Network Request Forgery in [research.jcs_14] for details).

In the following, it is further assumed that the client is registered with H-AS (URI: https://honest.as.example, client ID: 7ZGZldHQ) and with A-AS (URI: https://attacker.example, client ID: 666RVZJTA). Assume that the client issues the redirection URI https://client.com/honest-cb for the honest toolkit and https://client.com/attack-cb for the attacker-controlled toolkit. URLs shown in the following example are shortened for presentation to include only parameters relevant to the attack.

Attack on the authorization code grant:

1. A victim user selects to start the grant using A-AS of A-Toolkit (e.g., by initiating a tool use on an agentic AI service).

2. The client stores in the user’s session that the user has selected this OAuth connection context and redirects the user to A-AS’s authorization endpoint with a Location header containing the URL https://attacker.example/authorize?response_type=code&client_id=666RVZJTA&state=[state]&redirect_uri=https%3A%2F%2Fclient.com%2Fattack-cb.

3. When the user’s browser navigates to the A-AS, the attacker immediately redirects the browser to the authorization endpoint of H-AS. In the authorization request, the attacker uses the honest authorization URL and replaces the state with the one freshly received. Therefore, the browser receives a redirection with a Location header pointing to https://honest.as.example/authorize?response_type=code&client_id=7ZGZldHQ&state=[state]&redirect_uri=https%3A%2F%2Fclient.com%2Fhonest-cb.

4. Due to implicit or prior approvals, the user might not be prompted for a re-authorization (re-consent). H-AS issues a code and sends it (via the browser) back with the state to the client.

5. Since the client still assumes that the code was issued by A-Toolkit, as stored in the user’s session (with state verified), it will try to redeem the code at A-AS’s token endpoint.

6. The attacker therefore obtains code and can either exchange the code for an access token (for public clients) or perform an authorization code injection attack as described in Section 4.5 of [RFC9700].

This Cross-toolkit OAuth Account Takeover (COAT) attack is a generalization of the Cross-app OAuth Account Takeover as defined in [research.cuhk] and the mix-up attack as defined in Section 4.4 of [RFC9700]. This COAT exploits confusion between the OAuth connection context (i.e., a combination of OAuth provider, toolkit, tenant) of a centralized client rather than being limited to confusion between two distinct authorization servers.

Variants:

* COAT under the OAuth-as-a-Service context: the attack above can be launched with a malicious tenant by adding a custom toolkit with an OAuth provider that targets an honest AS used by another tenant’s toolkit. A variation of this attack involves the malicious tenant simply using a shared off-the-shelf toolkit that comes with a pre-built OAuth provider (with client registration included), if so allowed.

* Implicit Grant: In the implicit grant, the attacker receives an access token instead of the code in Step 4. The attacker’s authorization server receives the access token when the client makes either a request to the A-AS userinfo endpoint (defined in [OpenID.Core]) or a request to the attacker’s resource server (since the client believes it has completed the flow with A-AS).

* Cross-toolkit OAuth Request Forgery (CORF): If clients do not store the selected OAuth connection context in the user's session, but in the redirection URI instead, attackers can mount an attack called Cross-toolkit OAuth Request Forgery (CORF). This results in a compromised OAuth connection between the victim's application identity and the attacker's toolkit access. The goal of this specific attack variant is not to obtain an authorization code or access token, but to force the client to use an attacker's authorization code or access token for H-AS. This Cross-toolkit OAuth Request Forgery attack is a generalization of the Cross-app OAuth Request Forgery as defined in [research.cuhk] and the Na誰ve RP Session Integrity Attack when the OAuth connection context is limited to AS, and is detailed in Section 3.4 of [arXiv.1601.01229].

* OpenID Connect: Some variants can be used to attack OpenID Connect. In these attacks, the attacker misuses features of the OpenID Connect Discovery [OpenID.Discovery] mechanism or replays access tokens or ID Tokens to conduct a mix-up attack. The attacks are described in detail in Appendix A of [arXiv.1704.08539] and Section 6 of [arXiv.1508.04324v2] ("Malicious Endpoints Attacks").

2.2.2. Countermeasures

The client MUST NOT share OAuth providers with completed client registrations across toolkits and tenants belonging to different owners.

The client MUST use all variables in its supported OAuth connection context to form a connection context identifier that uniquely identifies each AS instance configured at the client. This identifier always includes the unique toolkit identifier. Additionally,

* a client allowing each toolkit to use multiple OAuth providers, of which one AS may be compromised as assumed in Section 4.4 of [RFC9700], MUST also include the OAuth provider identifier;

* a multi-tenant client MUST also include the tenant identifier, if the toolkit identifier is not globally unique.

Unless otherwise specified as follows, the client MUST issue per-context distinct redirection URI that incorporates this unique connection context identifier. When initiating an authorization request, the client MUST store this identifier in the user's session. When an authorization response was received on the redirection endpoint, the client MUST also check that the context identifier from the distinct redirection URI matches with the one in the user's session. If there is a mismatch, the client MUST abort the flow.

Existing countermeasures for mix-up attacks (Section 4.4 of [RFC9700]) can be a replacement under the following conditions:

* the client has entirely dropped the support to implicit grant, and

* the OAuth provider specifies an AS not by individually configured AS endpoints but instead by an abstract issuer identifier (as defined in Section 4.4.2 of [RFC9700]) that represents the endpoints, and

* the issuer identifier is used either in place of the connection context identifier in the redirection URI or is separately returned according to [RFC9207], and

* an additional runtime resolution is used to resolve the issuer to retrieve the associated AS endpoints (e.g., with the authorization server metadata [RFC8414] or OpenID Discovery [OpenID.Discovery]). Clients using such resolution solely to pre-populate individual AS endpoint fields, without any coupling with the issuer identifier will remain vulnerable.

2.3. Cross-user OAuth Session Fixation

Based on similar deployment needs as outlined in Section 2.2, multiple OAuth connections can be linked to some form of user's identity (e.g., an application's user identifier). This identity information is supposedly maintained in a session established and already bound to the user agent. In real-world deployments, however, this prerequisite can be broken for various reasons. For instance, in OAuth deployments that cross user agents, where an authenticated native app has its backend acting as a confidential OAuth client, the client opens a tool linking URL in an external user agent (typically a browser, as defined in [RFC8252]) that has no authenticated sessions with the client. As a workaround, the client introduces a session fixation vulnerability: it encodes a session identifier into the URL, which fixates a dedicated authorization session to complete the OAuth flow with the user at the client.

The Cross-user OAuth Session Fixation exploits this session fixation attack vector. The attacker attempts to trick a victim into completing an OAuth flow that the attacker has initiated at the client. As a result, the attacker's session will be used to establish an OAuth connection with the victim's tool resources or identity, hence resulting in the same impact as COAT (Section 2.2). However, this attack exploits confusion over the intended user bound to that connection context during the OAuth flow, contrasting with COAT, which exploits confusion within the OAuth connection context (OAuth provider, toolkit, tenant).

In general, this session fixation vulnerability may be viewed as violating the requirement of "binding the contents of state to the browser (more precisely, the initiating user agent) session" to defend against Cross-Site Request Forgery (CSRF, see Section 4.7 of [RFC9700]). However, while PKCE [RFC7636] can mitigate CSRF, PKCE alone cannot mitigate this new attack: Since the entire OAuth flow including the authorization request and the request to redirection endpoint is completed by the same victim user, the cryptographic binding of authorization request and access token request enforced by PKCE is preserved. The impact of the new attack is also more severe than that of typical CSRF attacks.

Note that this section focuses on the authorization code grant. For similar attacks in cross-device OAuth flows, see Section 4 of [CDFS].

2.3.1. Attack Description

Preconditions: It is assumed that the client has maintained a user's session. But it does not want to or cannot authenticate the user at the redirection endpoint for usability reasons, before completing the OAuth flow.

Example Attack:

1. From a vulnerable client, the attacker initiates OAuth against a tool and obtains an authorization request URL, in which the state parameter has encoded a newly created authorization session of the attacker.

2. The attacker sends this authorization request URL to a victim.

3. The victim visits the URL and (automatically, due to prior or implicit approvals,) authorizes the client to access their resources.

4. Upon receiving the state at the redirection endpoint, the client fixates the attacker’s authorization session and completes the OAuth flow.

5. The attacker’s account at the client now gains access to the victim’s resources.

Variant:

The client may first generate a pre-authorization URL for the purpose of fixating a session before redirecting to the authorization endpoint.

Non-normative example request:

```
GET /oauth?auth_session_id=6064f11c-f73e-425b-b9b9-4a36088cdb2b HTTP/1.1
Host: client.com
```

Non-normative example response:

```
HTTP/1.1 303 See Other
Location: https://as.example/authorize?
          response_type=code&client_id=K9dTpWzqL7&state=b1d8f043
          &redirect_uri=https%3A%2F%2Fclient.com%2Fcb
Set-Cookie: auth_session_id=6064f11c-f73e-425b-b9b9-4a36088cdb2b
```

This variant differs from the above only by obtaining and sending the pre-authorization URL instead, which will first fixate the attacker’s authorization session (rather than in Step 4).

2.3.2. Countermeasures

To defend against the Cross-user OAuth Session Fixation attack, the client MUST ensure that an OAuth flow initiated by one user is completed by the same user.

The most straightforward countermeasure is to identify the initiating user via their existing session at the client, rather than introducing a fixated session, if usability conditions permit. However, eliminating the session fixation vector may not always be feasible due to application needs. For instance, when the OAuth client responsibilities of establishing OAuth connections and the application’s session management are handled by separate entities (e.g., separate services isolated under different origins, accessed from different user agents, or when the OAuth client is outsourced to an OAuth-as-a-Service provider), as observed in practice by [research.cuhk2] and [research.cuhk3].

Hence, the client MUST validate the binding of any _newly fixated authorization session_ (conveyed via state or the pre-authorization URL) to the _existing user session_ (maintained at the user agent) that initiates the OAuth flow, before proceeding with the access token request. Depending on the specific current settings:

* If the user session is accessible at the redirection endpoint, the client can validate this binding directly.

* If the user session is not accessible at the redirection endpoint, for example, because the redirection endpoint is hosted in a different origin or accessed from a different user agent than where the user session is maintained, the countermeasure requires one of the following to make the session accessible prior to validation:

  - an implementation change to co-locate the redirection endpoint under the same origin as the endpoint maintaining the user session, and/or to re-authenticate the user at the redirection endpoint from the external user agent (e.g., a browser), or

  - from the current redirection endpoint, performing a further redirection back to the starting origin and/or user agent where the existing session is available. For native apps, the redirect options specified in Section 7 of [RFC8252] MUST be used. The location of this further redirection MUST NOT be controllable by an attacker, or it will result in Open Redirection (Section 4.11 of [RFC9700]).

3. Security Considerations

Security considerations are described in Section 2.

4. IANA Considerations

This document has no IANA actions.

5. References

5.1. 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/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>.

[RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <https://www.rfc-editor.org/rfc/rfc7521>.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <https://www.rfc-editor.org/rfc/rfc7523>.

[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>.

[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/rfc/rfc9700>.

5.2. Informative References

[arXiv.1508.04324v2] Mladenov, V., Mainka, C., and J. Schwenk, "On the security of modern Single Sign-On Protocols: Second-Order Vulnerabilities in OpenID Connect", arXiv:1508.04324v2, January 2016, <https://arxiv.org/abs/1508.04324v2/>.

[arXiv.1601.01229] Fett, D., K端sters, R., and G. Schmitz, "A Comprehensive Formal Security Analysis of OAuth 2.0", arXiv:1601.01229, January 2016, <https://arxiv.org/abs/1601.01229/>.

[arXiv.1704.08539]
Fett, D., K端sters, R., and G. Schmitz, "The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines", arXiv:1704.08539, April 2017, <https://arxiv.org/abs/1704.08539/>.

[CDFS] Kasselman, P., Fett, D., and F. Skokan, "Cross-Device Flows: Security Best Current Practice", Work in Progress, Internet-Draft, draft-ietf-oauth-cross-device-security-15, 23 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-cross-device-security-15>.

[OAUTH-7523bis]
Jones, M. B., Campbell, B., Mortimore, C., and F. Skokan, "Updates to OAuth 2.0 JSON Web Token (JWT) Client Authentication and Assertion-Based Authorization Grants", Work in Progress, Internet-Draft, draft-ietf-oauth-rfc7523bis-05, 12 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-rfc7523bis-05>.

[OpenID.CIBA]
Fernandez, G., Walter, F., Nennker, A., Tonge, D., and B. Campbell, "OpenID Connect Client-Initiated Backchannel Authentication Flow - Core 1.0", September 2021, <https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-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 2", December 2023, <https://openid.net/specs/openid-connect-core-1_0.html>.

[OpenID.Discovery]
Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0 incorporating errata set 2", December 2023, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

[research.cuhk]
Luo, K., Wang, X., Fung, P. H. A., Lau, W. C., and J. Lecomte, "Universal Cross-app Attacks: Exploiting and Securing OAuth 2.0 in Integration Platforms", 34th USENIX Security Symposium (USENIX Security 25), August 2025, <https://www.usenix.org/system/files/usenixsecurity25-luo-kaixuan.pdf>.

[research.cuhk2]
Luo, K., Wang, X., Fung, A., Lecomte, J., and W. C. Lau, "One Hack to Rule Them All: Pervasive Account Takeovers in Integration Platforms for Workflow Automation, Virtual Voice Assistant, IoT, & LLM Services", Black Hat USA 2024, August 2024, <https://www.blackhat.com/us-24/briefings/schedule/#one-hack-to-rule-them-all-pervasive-account-takeovers-in-integration-platforms-for-workflow-automation-virtual-voice-assistant-iot-38-llm-services-38994>.

[research.cuhk3]
Luo, K., Wang, X., Fung, A., Bi, Y., and W. C. Lau, "Back to the Future: Hacking and Securing Connection-based OAuth Architectures in Agentic AI and Integration Platforms", Black Hat USA 2025, August 2025, <https://www.blackhat.com/us-25/briefings/schedule/index.html#back-to-the-future-hacking-and-securing-connection-based-oauth-architectures-in-agentic-ai-and-integration-platforms-44686>.

[research.jcs_14]
Bansal, C., Bhargavan, K., Delignat-Lavaud, A., and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis", Journal of Computer Security, vol. 22, no. 4, pp. 601-657, April 2014, <https://www.doc.ic.ac.uk/~maffeis/papers/jcs14.pdf>.

[research.ust]
Hosseyni, P., K端sters, R., and T. W端rtele, "Audience Injection Attacks: A New Class of Attacks on Web-Based Authorization and Authentication Standards", April 2025, <https://eprint.iacr.org/2025/629>.

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <https://www.rfc-editor.org/rfc/rfc7009>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

[RFC7522] Campbell, B., Mortimore, C., and M. Jones, "Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7522, DOI 10.17487/RFC7522, May 2015, <https://www.rfc-editor.org/rfc/rfc7522>.

[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>.

[RFC8628] Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Authorization Grant", RFC 8628, DOI 10.17487/RFC8628, August 2019, <https://www.rfc-editor.org/rfc/rfc8628>.

[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>.

[RFC9207] Meyer zu Selhausen, K. and D. Fett, "OAuth 2.0 Authorization Server Issuer Identification", RFC 9207, DOI 10.17487/RFC9207, March 2022, <https://www.rfc-editor.org/rfc/rfc9207>.

Acknowledgments

We would like to thank
// TODO add names, sort by last name.
//
// -- Tim W. Daniel Fett, Wing Cheong Lau, Julien Lecomte, Aaron Parecki, Guido Schmitz, and Xianbo Wang

for their valuable feedback and contributions to this document.

Document History

[[To be removed from the final specification]]

-01

* Clarify that shared redirection URI is not a precondition of COAT
* Clarify that COAT countermeasure uniquely identifies each configured AS instance
* Clarify Session Fixation countermeasures and relationship to CSRF and PKCE
* Use terminology that is less ambiguous and better aligned with standard OAuth language
* Editorial clarifications and fixes, and reference updates

-00

* WG adoption, no changes from previous individual draft

Authors' Addresses

Tim W端rtele
University of Stuttgart
Germany
Email: tim.wuertele@sec.uni-stuttgart.de

Pedram Hosseyni
University of Stuttgart
Germany
Email: pedram.hosseyni@sec.uni-stuttgart.de

Kaixuan Luo
The Chinese University of Hong Kong
Hong Kong,
China
Email: kaixuan@ie.cuhk.edu.hk

Adonis Fung
Samsung Research America
United States of America
Email: adonis.fung@samsung.com