Web Authorization Protocol   T. W端rtele
Internet-Draft   P. Hosseyni
Updates: 6749, 6750, 7521, 7522, 7523, 9700   University of Stuttgart
         (if approved)   K. Luo
Intended status: Best Current PracticeThe Chinese University of Hong Kong
Expires: 19 December 2025   A. Fung
                                                  Samsung Research America
                                                        17 June 2025

Updates to OAuth 2.0 Security Best Current Practice
draft-wuertele-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-wuertele-oauth-security-topics-update.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-wuertele-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 3
  1.1. Structure 3
  1.2. Conventions and Terminology 3
2. Attacks and Mitigations 3
  2.1. Audience Injection Attacks 4
    2.1.1. Attack Description 4
    2.1.2. Countermeasures 6
  2.2. Updates to Mix-Up Attacks 8
    2.2.1. Updates to "Per-AS Redirect URIs" Mix-Up Variant . . 8
    2.2.2. Clarifications on Countermeasures for Mix-Up Variants 10
  2.3. Attacks in Open Ecosystems 11
    2.3.1. OAuth in Open Ecosystems 11
    2.3.2. Attack Scenario 12
    2.3.3. Mix-Up Attacks Reloaded 13
    2.3.4. Client Configuration Confusion Attack 16
3. Security Considerations 20
4. IANA Considerations 20
5. References 20
  5.1. Normative References 20
  5.2. Informative References 21
Acknowledgments 23
Document History 24
Authors' Addresses 24

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. Generally, the attacks in this section assume a scenario where a client can interact with multiple authorization servers.

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 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. Updates to Mix-Up Attacks

Mix-up attacks can occur in scenarios where an OAuth client interacts with two or more authorization servers and at least one authorization server is under the control of the attacker.

This section extends the mix-up attack analysis in Section 4.4 of [RFC9700]. It introduces additional attack variants that can occur when clients use distinct redirection URIs for each authorization server and clarifies the applicability of countermeasures to certain attack variants.

2.2.1. Updates to "Per-AS Redirect URIs" Mix-Up Variant

The basic mix-up attack described in the initial paragraphs of Section 4.4.1 of [RFC9700] specifies, as one of its preconditions, that "the client stores the authorization server chosen by the user in a session bound to the user’s browser and uses the same redirection URI for each authorization server".

If the client instead uses a distinct redirection URI for each authorization server (i.e., Per-AS Redirect URIs), three subvariants of mix-up attacks are still possible. This section updates the description of the "Per-AS Redirect URIs" mix-up variant in Section 4.4.1 of [RFC9700], with subvariants 1 and 2 newly introduced in this document:

Per-AS Redirect URIs: There are three subvariants of mix-up attacks when the client uses distinct redirection URIs for each authorization server.

Subvariant 1: Advanced Mix-up Attack. If the client uses different redirection URIs for different authorization servers but treats them as the same URI, a slight variant of the basic mix-up attack remains possible (see Footnote 7 of [arXiv.1601.01229]). In this attack, the attacker not only replaces the client ID as in the basic mix-up attack, but also replaces the redirection URI when redirecting the browser to the authorization endpoint of the honest authorization server.

This advanced mix-up attack is described in detail below, with the Preconditions as well as Step 2 and Step 3 updated compared to the basic mix-up attack.

Preconditions: For this variant of the attack to work, it is assumed that

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

* the client stores the authorization server chosen by the user in a session bound to the user's browser and uses distinct redirection URIs for different authorization servers, but treats them as the same URI (i.e., it does not use the redirection URI to differentiate authorization servers at the redirection endpoint).

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/9XpLmK2qR/cb for H-AS and https://client.com/4FvBn8TzY/cb for A-AS. 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. The user selects to start the grant using A-AS (e.g., by clicking on a button on the client's website).

2. The client stores in the user's session that the user selected "A-AS" 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&redirect_uri=https%3A%2F%2Fclient.com%2F4FvBn8TzY%2Fcb.

3. When the user's browser navigates to the attacker's authorization endpoint, the attacker immediately redirects the browser to the authorization endpoint of H-AS. In the authorization request, the attacker replaces the client ID of the client at A-AS with the client's ID at H-AS, and replaces the redirection URI of A-AS with the redirection URI of H-AS. Therefore, the browser receives a redirection (303 See Other) with a Location header pointing to https://honest.as.example/authorize?response_type=code&client_id=7ZGZldHQ&redirect_uri=https%3A%2F%2Fclient.com%2F9XpLmK2qR%2Fcb.

4. The user authorizes the client to access their resources at H-AS. H-AS issues a code and sends it (via the browser) back to the client.

5. Since the client still assumes that the code was issued by A-AS, 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].

Subvariant 2: Na誰ve RP Session Integrity Attack. If clients use different redirection URIs for different authorization servers, and clients do not store the selected authorization server in the user's session, attackers can mount an attack called "Na誰ve RP Session Integrity Attack". Note that unlike other mix-up variants, the goal of this attack 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. See Section 3.4 of [arXiv.1601.01229] for details.

Subvariant 3: Cross Social-Network Request Forgery. If clients use different redirection URIs for different authorization servers, clients do not store the selected authorization server in the user's session, and authorization servers do not check the redirection URIs properly (see Section 4.1 of [RFC9700]), attackers can mount an attack called "Cross Social-Network Request Forgery". These attacks have been observed in practice. Refer to [research.jcs_14] for details.

2.2.2. Clarifications on Countermeasures for Mix-Up Variants

Section 4.4.2 of [RFC9700] specifies two countermeasures, Mix-Up Defense via Issuer Identification and via Distinct Redirect URIs (referred to hereafter as "existing mix-up defenses"). Both defenses require the client to store and compare the issuer of the authorization server. Specifically, the issuer is defined in the second paragraph of Section 4.4.2 of [RFC9700] as follows:

> The issuer serves ... as an abstract identifier for the combination of the authorization endpoint and token endpoint that are to be used in the flow. If an issuer identifier is not available ..., a different unique identifier for this tuple or the tuple itself can be used instead.

Existing mix-up defenses are sufficient to protect against most attack variants, with a few specific cases requiring clarification:

* For the mix-up attack variant in "Implicit Grant", since the flow does not involve a token endpoint, a unique identifier for the authorization endpoint URL MAY be used as the equivalent of issuer if an issuer identifier is not available. Then, clients MUST apply existing mix-up defenses.

* For all three subvariants of the "Per-AS Redirect URIs" attack variant (Section 2.2.1), clients MUST follow existing mix-up defenses to defend against mix-up attacks. Clients MAY choose to reuse the per-AS redirection URI already configured in their deployments to satisfy the "distinct redirection URI for each issuer" requirement when implementing the "Mix-Up Defense via Distinct Redirect URIs" defense (Section 4.4.2.2 of [RFC9700]). For subvariant 3 ("Cross Social-Network Request Forgery"), the existing mix-up defenses alone are not sufficient. In addition, authorization servers MUST enforce exact redirection URI matching, as specified in Section 4.1.3 of [RFC9700].

Note that when the issuer is not unique to a client (i.e., when a client can interact with multiple configurations of the same authorization server), the security considerations go beyond the scope of mix-up attacks and are discussed further in Section 2.3.

2.3. Attacks in Open Ecosystems

This section describes the OAuth use case and associated attacks in open ecosystems, along with practical countermeasures tailored to such environments.

2.3.1. OAuth in Open Ecosystems

In traditional OAuth deployments, a client registers with an authorization server to access protected resources hosted by a resource server. The means through which the client registers with an authorization server typically involve the client developer manually registering the client at the authorization server’s website, or by using Dynamic Client Registration [RFC7591]. The client may access different resources from distinct resource servers, thereby requiring registration with multiple authorization servers. The choice of what resources to access (and thus which authorization and resource servers to integrate) is at the discretion of the client (or client developer).

This section discusses OAuth in "open ecosystems", where the protected resources accessible to a client are configured beyond the discretion of the client or client developers. This can be the case, for example, if the client offers an open marketplace or registry where external developers can publish information about resource servers and associated authorization servers at development time (e.g., in integration platforms, where external services register themselves as authorization and resource servers with a platform that acts as a client, see [research.cuhk]), or if an end-user can specify any compliant configurations to interact with at runtime (e.g., in the Model Context Protocol [MCP-Spec]).

This document defines a "client configuration" as a set of configuration elements that enable a client to access an OAuth 2.0 protected resource, including details about the authorization server, client registration, and the resource server. In open ecosystems, these configuration elements are typically managed as follows:

* Authorization Server configuration: Clients may employ a manual approach, where external developers or end-users enter the necessary fields (i.e., endpoint URLs of the authorization server) at the client’s website, or support fetching information about the authorization server via Authorization Server Metadata [RFC8414].

* Client Registration configuration at the authorization server: Clients may employ a manual approach, where external developers or end-users enter the necessary fields (i.e., client identifier and client credentials) at the client’s website, or support fetching information about the client registration via Dynamic Client Registration [RFC7591].

* Protected Resource configuration: Clients may employ a manual approach, where external developers or end-users enter the necessary fields (i.e., endpoint URLs of the resource server) at the client’s website, or support fetching information about the protected resource via Protected Resource Metadata [RFC9728]. External developers or end-users may further specify when and how the protected resources shall be fetched and processed by the clients.

The integration pattern in open ecosystems expands the use of OAuth in dynamic scenarios, creating new challenges with respect to functionality and security beyond the scope of [RFC9700].

2.3.2. Attack Scenario

With the new integration model, OAuth in open ecosystems introduces two notable implications:

* Increased risk of malicious integration: Since the responsibility of integrating client configurations is shifted to external developers or end-users, there is an increased risk that malicious client configurations may be introduced, including attacker-controlled authorization servers or resource servers.

* New requirements for handling shared issuers: Clients shall support client configurations with potentially shared issuers (explained below) to fulfill functional needs, which in turn introduces new security gaps.

In traditional OAuth deployments, it is implicitly assumed that there is exactly one client configuration for each client-authorization server pair. Under this assumption, the issuer (as defined in Section 4.4.2 of [RFC9700]) serves as a unique identifier for the client. This has led to the common practice of clients tracking "the authorization server chosen by the user" during OAuth flows, as well as the adoption of existing mix-up defenses (Section 4.4.2 of [RFC9700]), all of which are based on the issuer concept that uniquely identifies each authorization server from the client’s point of view.

In open ecosystems, however, the new integration pattern does not, and fundamentally cannot, restrict the use of the same authorization server across multiple client configurations. For instance, clients may legitimately integrate various functionalities that access different resources or process the same resources differently, while relying on the same authorization server (i.e., having shared issuers, which implies that the issuer no longer uniquely identifies a client configuration).

As a result, the client may integrate with an authorization server under the attacker’s control (Section 2.3.3), or an attacker-controlled resource server paired with an honest authorization server -- even if the authorization server is already integrated at the client under a different client configuration (Section 2.3.4).

2.3.3. Mix-Up Attacks Reloaded

This section provides a tailored attack description and alternative countermeasure for mix-up attacks in open ecosystems. The descriptions here follow [research.cuhk], where additional details are laid out.

2.3.3.1. Attack Description

Section 4.4 of [RFC9700] exemplifies scenarios in which an attacker-controlled authorization server may be introduced:

> This can be the case, for example, if the attacker uses dynamic registration to register the client at their own authorization server, or if an authorization server becomes compromised.

OAuth deployments in open ecosystems extend the above scenarios: an external developer or end-user can introduce an attacker-controlled authorization server by integrating a new client configuration at the client.

Furthermore, multiple client configurations may use the same authorization server (i.e., share the same issuer). To manage such scenarios, the client should treat each client configuration independently, typically by keeping track of the client configuration chosen by the user during the OAuth flow. For example, the client may store a unique identifier for each client configuration (rather than for each authorization server) in the user's session, or assign a distinct redirection URI to each client configuration. This enables the client to distinguish between client configurations, retrieve the correct client registration information for access token requests, and access the intended protected resources.

Attackers can exploit this setup to mount mix-up attacks, using a malicious authorization server from an attacker-controlled client configuration to target the honest authorization server from an honest client configuration. For details on this attack vector, see "Cross-app OAuth Account Takeover" (COAT) and "Cross-app OAuth Request Forgery" (CORF) in Section 4.2 of [research.cuhk].

As an example, the attack analogous to the Advanced Mix-up Attack described in Section 2.2.1 is outlined below.

Preconditions: For this variant of the attack to work, it is assumed that

* the implicit or authorization code grant is used with multiple client configurations of which one is considered "honest" (H-Config) and one is operated by the attacker (A-Config), and

* the client stores the client configuration chosen by the user in a session bound to the user's browser and uses distinct redirection URIs for different client configurations, but treats them as the same URI (i.e., it does not use the redirection URI to differentiate client configurations at the redirection endpoint).

In the following, it is further assumed that the client is registered with an honest authorization server H-AS (URI: https://honest.as.example, client ID: 7ZGZldHQ) under H-Config and with an attacker-controlled authorization server A-AS (URI: https://attacker.example, client ID: 666RVZJTA) under A-Config. Assume that the client issues the redirection URI https://client.com/9XpLmK2qR/cb for H-Config and https://client.com/4FvBn8TzY/cb for A-Config. 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. The user selects to start the grant using A-Config (e.g., by clicking on a button on the client’s website).

2. The client stores in the user’s session that the user selected "A-Config" 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&redirect_uri=https%3A%2F%2Fclient.com%2F4FvBn8TzY%2Fcb.

3. When the user’s browser navigates to the attacker’s authorization endpoint, the attacker immediately redirects the browser to the authorization endpoint of H-AS. In the authorization request, the attacker replaces the client ID of the client at A-Config with the client’s ID at H-Config, and replaces the redirection URI of A-Config with the redirection URI of H-Config. Therefore, the browser receives a redirection (303 See Other) with a Location header pointing to https://honest.as.example/authorize?response_type=code&client_id=7ZGZldHQ&redirect_uri=https%3A%2F%2Fclient.com%2F9XpLmK2qR%2Fcb.

4. The user authorizes the client to access their resources at H-AS. H-AS issues a code and sends it (via the browser) back to the client.

5. Since the client assumes that the code was issued in the interaction with A-Config, 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].

2.3.3.2. Countermeasures

At its core, a client in open ecosystems may integrate multiple configurations of the same authorization server, and therefore the issuer may not be unique to the client. While the existing mix-up defenses in Section 4.4.2 of [RFC9700] are sufficient, a variant of the "Mix-Up Defense via Distinct Redirect URIs" defense described in Section 4.4.2.2 of [RFC9700] MAY be deployed instead for practical reasons:

> To apply this defense, clients MUST use a distinct redirection URI for each client configuration they interact with. Clients MUST check that the authorization response was received from the correct client configuration by comparing the distinct redirection

URI for the client configuration to the URI where the authorization response was received on. If there is a mismatch, the client MUST abort the flow.

To maximize compatibility, this countermeasure imposes no new requirements on authorization servers compliant with the original OAuth 2.0 specification [RFC6749]. This is essential for securing open ecosystems where clients may be integrated with numerous client configurations, and many authorization servers may not support the "Mix-Up Defense via Issuer Identification" defense described in Section 4.4.2.1 of [RFC9700] (e.g., returning the issuer identifier via an iss parameter in the authorization response [RFC9207]).

To ease the development burden, compared to the "Mix-Up Defense via Distinct Redirect URIs" defense outlined in Section 4.4.2.2 of [RFC9700], this countermeasure does not require clients to manage issuers exclusively for mix-up defense. Instead, it relies on existing isolation boundaries that already serve the functional need of differentiating client configurations.

Note that this countermeasure is not intended to redefine the concept of an issuer (or issuer identifier) from an authorization server-specific identifier to one tied to client configurations. Nor does it invalidate the existing countermeasures described in Section 4.4.2 of [RFC9700] and clarified in Section 2.2.2, which remain sufficient to mitigate mix-up attacks in open ecosystems. Rather, this countermeasure MAY serve as an alternative defense.

2.3.4. Client Configuration Confusion Attack

When client authentication is not required such as in the implicit grant or for a public client, or when signature-based client authentication methods such as private_key_jwt (as defined in [OpenID.Core]) or signed JWTs (as defined in [RFC7521] and [RFC7523]) are used, a malicious resource server may be able to obtain an access token issued by an honest authorization server. This is achieved by registering an honest authorization server at the client under a malicious client configuration, pairing it with an attacker-controlled resource server, thus tricking the client into sending its access tokens to the attacker instead of using them at the honest resource server.

Unlike mix-up attacks or audience injection attacks, client configuration confusion attacks do not involve a malicious authorization server, but rather an honest one used in a malicious client configuration.

2.3.4.1. Attack Description

Client configuration confusion attacks are feasible if a client satisfies the following preconditions:

1. The client has at least two client configurations, one of which is configured by an attacker. The client allows for the two configurations to use the same authorization server (i.e., issuer-sharing client configurations) but different resource servers.

2. The client stores the client configuration chosen by the user in a session bound to the user’s browser and assigns the same redirection URI for at least the issuer-sharing, if not all, client configurations.

3. The client uses the same client ID across the issuer-sharing client configurations.

4. Regarding client authentication, one of the following applies:

* The client authenticates to the authorization server in both client configurations with a signature-based authentication method (e.g., the jwt-bearer client authentication from [RFC7523]) using the same key pair.

* The client interacts with the authorization server in both client configurations without requiring client authentication (i.e., using the implicit grant, or as a public client).

For brevity of presentation, in the following, let H-AS, H-RS, and H-Config denote an honest authorization server, resource server, and client configuration, respectively. Let A-AS, A-RS, and A-Config denote an attacker-controlled authorization server, resource server, and client configuration, respectively.

2.3.4.1.1. Core Attack Steps

In the following, it is further assumed that the client is registered with H-AS for both client configurations. The client is configured to use A-RS for A-Config and H-RS for H-Config.

Attack on the authorization code grant:

1. The user selects to start the grant using A-Config (e.g., by clicking on a button on the client’s website).

2. The client stores in the user’s session that the user selected "A-Config" and redirects the user to H-AS’s authorization endpoint.

3. The user authorizes the client to access their resources at H-AS. Having validated that the client ID and redirection URI match the ones assigned for H-Config, H-AS issues a code and sends it (via the browser) back to the client.

4. The client redeems the code issued by H-AS at H-AS’s token endpoint. To authenticate with H-AS, the client creates a client assertion signed by itself. Recall that both A-Config and H-Config registered the client with H-AS under the same client ID, and that the client uses the same key pair for authentication at H-AS. Hence, this client assertion, if required, would be a valid authentication credential for the client at H-AS. Note that generating and validating client assertions in this step is irrelevant for public clients.

5. Since the client stores "A-Config" in the user’s session, it sends the access token to A-RS. The attacker therefore obtains the user’s access token issued by H-AS.

Variant in Implicit Grant: In the implicit grant, H-AS issues an access token instead of the code in Step 3. Step 4 is omitted. The rest of the attack proceeds as above.

2.3.4.1.2. Discussion: on Sharing Client IDs

The scope of client configuration confusion attack is limited to the reuse of registered clients, and considers phishing threats involving the open registration of new clients as out of scope, because such threats can be mitigated by existing mechanisms such as vetting during manual registration, or via initial access tokens as defined in Section 1.2 of [RFC7591].

For the attack to work, A-Config and H-Config need to share the same client ID in client registration (precondition 3 in Section 2.3.4.1). This can be the case, for example, if an external developer or end-user can control the client ID used in A-Config during manual registration, or if the client uses dynamic registration to get assigned the same client ID for A-Config as in H-Config, as detailed below.

When the client is designed to perform dynamic client registration once per client configuration, A-Config and H-Config could feasibly share the same client ID. Unlike the situation in Section 2.1, since A-Config uses H-AS instead of A-AS, the attacker cannot directly

control which client ID the authorization server assigns to A-Config. However, according to the description of client ID returned in Client Information Response (Section 3.2.1 of [RFC7591]):

client_id OAuth 2.0 client identifier string. It SHOULD NOT be currently valid for any other registered client, though an authorization server MAY issue the same client identifier to multiple instances of a registered client at its discretion.

The second half of the last sentence explicitly allows a client to obtain a client ID of the same value across multiple client registrations, as long as the client is considered to be instances of the same registered client by the authorization server.

This behavior is intended for registering different "client instances", i.e., different deployed instances of the same piece of client software (see Section 1.2 of [RFC7591]). However, the authorization server cannot distinguish between this case and a client registering multiple times for different client configurations.

Specifically, when the client initiates dynamic registration at H-AS based on H-Config and A-Config respectively, the authorization server, recognizing the two client registration requests as originating from the same client (e.g., indicated by the identical software identifier or software statement provided by the client, see Section 2 of [RFC7591] for their definitions), is likely to return the same client ID according to Appendix A.4.2 of [RFC7591]:

> Particular authorization servers might choose, for instance, to maintain a mapping between software statement values and client identifier values, and return the same client identifier value for all registration requests for a particular piece of software.

Therefore, a client interacting with H-Config and A-Config could feasibly share the same client ID when dynamic client registration is used.

2.3.4.2. Countermeasures

At its core, client configuration confusion attacks exploit the fact that, an attacker-controlled client configuration can reuse the registered client at the honest authorization server under an honest client configuration.

Clients in open ecosystems that interact with more than one client configuration and support authorization servers that either use signature-based client authentication methods or do not require

client authentication MUST employ the following countermeasure, unless client configuration confusion attacks are mitigated by other means, such as using fresh key material for each authorization server employing signature-based client authentication and disallowing any authorization server that does not require client authentication.

Clients MUST issue a distinct redirection URI for each client configuration they interact with, both during client registration and in OAuth flows. This ensures that the redirection URI for the attacker-controlled client configuration will fail the exact redirection URI match (required by Section 4.1.3 of [RFC9700]) at the honest authorization server, since the redirection URI at the honest client configuration is the only redirection URI registered for the client identifier at the honest authorization server.

When dynamic client registration is supported, clients SHOULD also specify a different software identifier (software_id) in client registration requests for each client configuration. This prevents client registration requests from being rejected by the authorization server when different redirection URIs are used, if the authorization server follows the excerpt from Section 5 of [RFC7591] below:

> An authorization server could also refuse registration requests from a known software identifier that is requesting different redirection URIs or a different client URI.

This countermeasure can be considered an actionable approach to mitigating the "Counterfeit Resource Server" threat (see "Access Token Phishing by Counterfeit Resource Server" in Section 4.9.1 of [RFC9700]) within the context of open ecosystems. It is complementary to general defenses for access token misuses, such as sender-constrained and audience-restricted access tokens as specified in Section 4.10 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>.

[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.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/>.

[MCP-Spec] Anthropic, "Model Context Protocol (MCP) Specification", March 2025, <https://modelcontextprotocol.io/specification/2025-03-26>.

[OAUTH-7523bis] Jones, M. B., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", Work in Progress, Internet-

Draft, draft-ietf-oauth-rfc7523bis-00, 21 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-rfc7523bis-00>.

[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/conference/usenixsecurity25/sec24winter-prepub-332-luo.pdf>.

[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>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/rfc/rfc7591>.

[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>.

[RFC9728] Jones, M.B., Hunt, P., and A. Parecki, "OAuth 2.0 Protected Resource Metadata", RFC 9728, DOI 10.17487/RFC9728, April 2025, <https://www.rfc-editor.org/rfc/rfc9728>.

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

* Updated temporary title

* Added introductory paragraphs, replaced placeholders

* Clarified issuer does not uniquely identify client config

* Cleaned up acknowledgement list

-00

* Initial version

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
Email: kaixuan@ie.cuhk.edu.hk

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