Suggested Changes to OAuth Security BCP

Mar 13, 2025

Summarizing our research, we have identified two main issues in the mix-up attack section of RFC9700 (a.k.a. OAuth Security BCP, https://datatracker.ietf.org/doc/html/rfc9700):

- COAT (Cross-app OAuth Account Takeover) / IdP mix-up attack:

  Lacks the attack scenario, precondition, description and countermeasure tailored for integration platforms;

- CORF (Cross-app OAuth Request Forgery) / Naïve RP session integrity attack:

  Not covered by existing specifications.

For minimal changes to the specification, we prioritize resolving the first issue. As for the second point, which involves different attack goals and requires additional attack descriptions beyond the current mix-up section, we will address it in a future update.

We propose the following concrete changes to RFC9700, with modifications highlighted in red.

Note: The “Major Updates” primarily relate to our new research, while the “Minor Remarks” and “Side Notes” provide additional useful information.

Major Updates

(1) Section 4.4. Mix-Up Attacks

Changes:

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

Rationale:

Extend the mix-up attack scenarios, to reflect the possibility of proactively introducing attacker-controlled authorization servers in open ecosystems like integration platforms.

(Note: I borrowed the more general term “open ecosystems” from the latest OAuth 2.1 changelog on alternative client registration methods, see https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-12#appendix-F)

(2) Section 4.4.1. Attack Description

Changes:

Variants:

- Mix-Up with Interception: ...
- Implicit Grant: ...
- Per-AS Redirect URIs: ...
- OpenID Connect: ...
- Multi-app Integration Ecosystem: In ecosystems such as workflow automation platforms or virtual assistants, a client integrates with multiple pairs of authorization and resource servers that function as connected apps. While several apps may share the same authorization server, each app requires the client to interact with the corresponding resource server in different ways. To handle each app independently, the client needs to treat shared authorization servers as separate servers and obtain authorization codes or access tokens from each individually. In these scenarios, the client typically stores the selected app instead of the selected authorization server in the user’s session. Attackers can mount a mix-up attack by targeting the H-AS of an uncompromised app with the A-AS of a malicious or compromised app. For details on this attack vector, see Section 4.2.1 of [research.cuhk] (“Cross-app OAuth Account Takeover”).

...

7.2. Informative References

...

[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), 13 August 2025, <https://mobitec.ie.cuhk.edu.hk/cross-app-oauth-security/paper.pdf>.

Rationale:

For attack precondition, clarified that the user's choice stored by the client could be not only an authorization server but also an app (e.g., in integration platforms, where multiple apps potentially share one authorization server). Here we did not introduce new terms like "integration platforms", but rather adhered to more general terminology.

Specifically, in integration ecosystems, multiple apps may share the same authorization server, but handle the requests to resource server differently. For example,

1. Sending to *different resource servers* such as an API gateway first;
2. Sending to *different API endpoints of the same resource server*;
3. Sending to the APIs with *different request parameters*;
4. Having *different wrappers around the same APIs* at the OAuth client side.

This motivates the client to differentiate by apps rather than authorization servers.

For attack description, we added a brief attack scenario description and pointed to our paper for further reference. We will update the hyperlink to the (pre-)published version of our paper in the reference when it's available.

(3) Section 4.4.2.2. Mix-Up Defense via Distinct Redirect URIs

Changes:

For this defense, clients MUST use a distinct redirection URI for each issuer they interact with.

Clients MUST check that the authorization response was received from the correct issuer by comparing the distinct redirection URI for the issuer to the URI where the authorization response was received on. If there is a mismatch, the client MUST abort the flow.

While this defense builds upon existing OAuth functionality, it cannot be used in scenarios where clients only register once for the use of many different issuers (as in some open banking schemes) and due to the tight integration with the client registration, it is harder to deploy automatically.

Furthermore, an attacker might be able to circumvent the protection offered by this defense by registering a new client with the "honest" authorization server using the redirect URI that the client assigned to the attacker's authorization server. The attacker

could then run the attack as described above, replacing the client ID with the client ID of their newly created client.

This defense SHOULD therefore only be used if other options are not available.

Note that for the mix-up variant in multi-app integration ecosystem (see Section 4.4.1), where an issuer is not always unique to a client, a variant of this defense is RECOMMENDED: Clients SHOULD use a distinct redirection URI for each app they interact with, and SHOULD check that the authorization response was received from the correct app by comparing the distinct redirection URI for the app to the URI where the authorization response was received on. If there is a mismatch, the client MUST abort the flow.

Rationale:

This defense clearly specifies the use of a *per-app identifier* rather than *per-authorization server (issuer)*, to better reflect the multi-app nature of integration platforms.

To maximize compatibility, it imposes no new dependencies on apps' authorization servers that are already compliant with the original OAuth spec [RFC6749]. This is essential for securing platforms that are integrated with thousands of apps.

We currently mark this variant defense as RECOMMENDED/SHOULD, which is open for discussion.

Important Note:

We did not expand the definition of *issuer* to include app-specific context, keeping it as a static, authorization server-specific identifier. Expanding the definition could cause confusion—for example, in Section 4.4.2.1 (Mix-Up Defense via Issuer Identification), it wouldn't make sense for the authorization server to return an app-specific identifier that varies across different clients as the `iss` parameter value.

In other words, we treat attacks in multi-app integration ecosystems (i.e., integration platforms) as a mix-up *attack variant*, and its corresponding defense as a mix-up *defense variant*. We thus did not change most issuer-related descriptions regarding the *standard* mix-up attack/defense.

Below are two examples (which we have *not* proposed any changes for now):

[Example 1] Section 2.1. Protecting Redirect-Based Flows

When an OAuth client can interact with more than one authorization server, a defense against mix-up attacks (see Section 4.4) is REQUIRED. To this end, clients SHOULD

- use the `iss` parameter as a countermeasure according to [RFC9207], or
- use an alternative countermeasure based on an `iss` value in the authorization response (such as the `iss` claim in the ID Token in [OpenID.Core] or in [OpenID.JARM] responses), processing that value as described in [RFC9207].

In the absence of these options, clients MAY instead use distinct redirection URIs to identify authorization endpoints and token endpoints, as described in Section 4.4.2.

[Example 2] Section 4.4.2. Countermeasures

For both defenses, clients MUST store, for each authorization request, the issuer they sent the authorization request to and bind this information to the user agent. The issuer serves, via the associated metadata, 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 (for example, if neither OAuth Authorization Server Metadata [RFC8414] nor OpenID Connect Discovery [OpenID.Discovery] is used), a different unique identifier for this tuple or the tuple itself can be used instead. For brevity of presentation, such a deployment-specific identifier will be subsumed under the issuer (or issuer identifier) in the following.

Minor Remarks

(4) Section 4.4.1. Attack Description

Changes:

Variants:

...

- Per-AS Redirect URIs: When clients use different redirection URIs for different authorization servers but treat them as the same URI, the attack would still work. An attacker can achieve this by replacing the redirection URI as well as the client ID at A-AS with those at H-AS in the authorization request in Step 3. Alternatively, 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, attackers can mount an attack

called "Cross Social-Network Request Forgery" (refer to [research.jcs_14] for details). These attacks have been observed in practice.

Rationale:

Clarify that even if different redirection URIs are used, mix-up attack could still happen (see Footnote 7 of [arXiv.1601.01229], and COAT_D (Section 4.2.1) of our USENIX Security paper). In our investigation, Microsoft, Google, Amazon, and Samsung's platform all use distinct redirection URIs but still lacked proper mix-up defense (now all fixed).

Note that we may need to further define what it means for "clients ... treat them [different redirection URIs] as the same URI".

Side Notes

(5) Section 4.4.1. Attack Description

RFC9700 Version:

Variants:

...

- 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).
- Per-AS Redirect URIs: 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, 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.

Reflection:

The implicit grant does not involve a token endpoint, landing the issuer concept inapplicable (Section 4.4.2: "The issuer serves, via the associated metadata, as an abstract identifier for the combination of the authorization endpoint and token endpoint that are to be used in the flow"; see also *Appendix A → Impractical Defense → 2. Supporting implicit grant* in our USENIX Security paper).

Moreover, if my understanding is correct, “Cross Social-Network Request Forgery” shares commonalities with other mix-up variants, but the two countermeasures in Section 4.4.2 does not apply. Instead, it requires authorization servers to “check the redirection URIs properly”.

We thus suggest adding a sentence somewhere in Section 4.4.2, highlighting that:

1. The two issuer-based countermeasures (Section 4.4.2) only apply to OAuth authorization code grant and OpenID Connect;
2. To prevent other mix-up variants, other countermeasures are required: For mix-up in implicit grant, OAuth clients simply SHOULD NOT use the implicit grant (see Section 2.1.2). For Cross Social-Network Request Forgery, Authorization servers MUST apply exact string matching against redirection URIs (see Section 4.1.3).

(6) Section 4.4.2.2. Mix-Up Defense via Distinct Redirect URIs

RFC9700 Version:

Furthermore, an attacker might be able to circumvent the protection offered by this defense by registering a new client with the "honest" authorization server using the redirect URI that the client assigned to the attacker's authorization server. The attacker could then run the attack as described above, replacing the client ID with the client ID of their newly created client.

Reflection:

I agree that this is a legitimate concern. However, is this circumvention of protection within the scope of the OAuth attacker model? The ability to register a new client with an honest authorization server seems to be a divergent assumption from that of the mix-up attack (see Attacker A3 in Section 3). With the new client registered, an attacker could launch a phishing attack to directly harvest the victim’s access tokens, making the setup of a malicious or compromised authorization server, and the subsequent mix-up attack, unnecessary. Please correct me if I’m wrong.

On the other hand, I understand the added value compared to a direct phishing attack may lie in its stealthiness, as the victim user still initiates the attack flow on the original client’s website, rather than on an attacker-controlled client’s website. That said, I don’t have a strong opinion on this point and we can keep this paragraph as is.