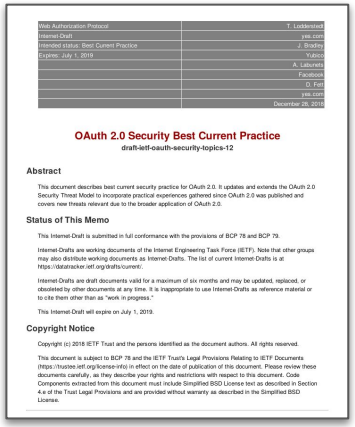
OAuth 2.0 Security Best Current Practice

Torsten Lodderstedt, John Bradley, Andrey Labunets, Daniel Fett

draft-ietf-oauth-security-topics-15

- Refines and enhances security guidance for OAuth 2.0 implementers
- Updates, but does not replace:
  - OAuth 2.0 Threat Model and Security Considerations (RFC 6819)
  - OAuth 2.0 Security Considerations (RFC 6749 & 6750)



Web Authorization Protocol	T. Lodderstedt
Internet-Draft	yes.com
Intended status: Best Current Practice	J. Bradley
Expires: July 1, 2019	Yubico
	A. Labunets
	Facebook
	D. Fett
	yes.com
	December 28, 2018

OAuth 2.0 Security Best Current Practice

draft-ietf-oauth-security-topics-12

Abstract

This document describes best current security practice for OAuth 2.0. It updates and extends the OAuth 2.0 Security Threat Model to incorporate practical experiences gathered since OAuth 2.0 was published and covers new threats relevant due to the broader application of OAuth 2.0.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on July 1, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

- Updated, more comprehensive Threat Model
- Description of Attacks and Mitigations
- Simple and actionable recommendations

Working Group Last Call

- On Version -13
- Several comments, from editorial to content

Changes After WGLC (-13)

Editorial Changes

- Attacker Model moved to Section 2, Recommendations is Section 1
- Various small improvements, e.g.,
  - clarifications,
  - definitions (CSRF and open redirector),
  - expanded some attack details and examples,
  - restructured discussions on mitigations and removed some less-relevant discussions.
- And, yes, we fixed the reference to `RFC 8418~~3~~4`

Normative Changes

- Open redirectors:
  - Clients “SHOULD NOT avoid forwarding…”, … otherwise “are advised to implement countermeasures against open redirection”
    → “Clients MUST NOT expose [open redirectors]”

```
Clients SHOULD avoid forwarding the user's browser to a URI obtained
from a query parameter since such a function could be utilized to
exfiltrate authorization codes and access tokens.  If there is a
strong need for this kind of redirects, clients are advised to
implement appropriate countermeasures against open redirection, e.g.,
as described by OWASP [owasp].
```

```
Clients MUST NOT expose URLs that forward the user's browser to
arbitrary URIs obtained from a query parameter ("open redirector").
Open redirectors can enable exfiltration of authorization codes and
access tokens, see Section 4.9.1.
```

Normative Changes

- PKCE:
  - AS “SHOULD provide a way to detect their support for PKCE” (implementation-specific or metadata)
    → “MUST”

```
AS SHOULD provide a way to detect their support for PKCE.  To this
end, they SHOULD either (a) publish, in their AS metadata
([!@RFC8418]), the element "code_challenge_methods_supported"
containing the supported PKCE challenge methods (which can be used by
the client to detect PKCE support) or (b) provide a deployment-
specific way to ensure or determine PKCE support by the AS.
```

```
Authorization servers MUST provide a way to detect their support for
PKCE.  To this end, they MUST either (a) publish the element
"code_challenge_methods_supported" in their AS metadata ([RFC8418])
containing the supported PKCE challenge methods (which can be used by
the client to detect PKCE support) or (b) provide a deployment-
specific way to ensure or determine PKCE support by the AS.
```

Implicit Grant

Before:

Clients SHOULD NOT use Implicit unless ATs are sender-constrained and AT injection is prevented.

Clients SHOULD use code, which also allows for sender-constraining.

Clients SHOULD use sender-constraining.

After:

Clients SHOULD NOT use Implicit unless AT injection is prevented and token leakage vectors are mitigated.

Clients SHOULD use code.

Clients SHOULD use sender-constraining.

In order to avoid these issues, clients SHOULD NOT use the implicit grant (response type "token") or any other response type issuing access tokens in the authorization response, such as "token id_token" and "code token id token", unless the issued access tokens are sender-constrained and access token injection in the authorization response is prevented.

A sender-constrained access token scopes the applicability of an access token to a certain sender. This sender is obliged to demonstrate knowledge of a certain secret as prerequisite for the acceptance of that token at the recipient (e.g., a resource server).

Clients SHOULD instead use the response type "code" (aka authorization code grant type) as specified in Section 3.1.1 or any other response type that causes the authorization server to issue access tokens in the token response. This allows the authorization server to detect replay attempts and generally reduces the attack surface since access tokens are not exposed in URLs. It also allows the authorization server to sender-constrain the issued tokens.

3.2. Token Replay Prevention

Authorization servers SHOULD use TLS-based methods for sender-constrained access tokens as described in Section 4.8.1.2, such as token binding [I-D.ietf-oauth-token-binding] or Mutual TLS for OAuth 2.0 [I-D.ietf-oauth-mtls] in order to prevent token replay. Refresh tokens MUST be sender-constrained or use refresh token rotation as described in Section 4.12.

In order to avoid these issues, clients SHOULD NOT use the implicit grant (response type "token") or other response types issuing access tokens in the authorization response, unless access token injection in the authorization response is prevented and the aforementioned token leakage vectors are mitigated.

Clients SHOULD instead use the response type "code" (aka authorization code grant type) as specified in Section 2.1.1 or any other response type that causes the authorization server to issue access tokens in the token response, such as the "code id token" response type. This allows the authorization server to detect replay attempts by attackers and generally reduces the attack surface since access tokens are not exposed in URLs. It also allows the authorization server to sender-constrain the issued tokens (see next section).

2.2. Token Replay Prevention

A sender-constrained access token scopes the applicability of an access token to a certain sender. This sender is obliged to demonstrate knowledge of a certain secret as prerequisite for the acceptance of that token at the recipient (e.g., a resource server).

Authorization and resource servers SHOULD use mechanisms for sender-constrained access tokens to prevent token replay as described in Section 4.8.1.1.2. The use of Mutual TLS for OAuth 2.0 [RFC8705] is RECOMMENDED. Refresh tokens MUST be sender-constrained or use refresh token rotation as described in Section 4.12.

Ready for Publication?

Q & A