Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 22, 2020

A. Backman, Ed.
Amazon

November 19, 2019

Proof-of-Possession Tokens for OAuth Using JWS HTTP Signatures
draft-richanna-oauth-http-signature-pop-00

Abstract

This document describes a method of generating and validating proof-of-possession tokens for use with OAuth 2.0. The required proof is provided via a JSON Web Signature (JWS) representing a signature of a minimal subset of elements from the HTTP request.

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 May 22, 2020.

Copyright Notice

Table of Contents

1. Introduction

In order to prove possession of an access token and its associated key, an OAuth 2.0 client needs to compute some cryptographic function and present the results to the protected resource as a signature. The protected resource then needs to verify the signature and compare that to the expected keys associated with the access token. This is in addition to the normal token protections provided by a bearer token [RFC6750] and transport layer security (TLS).

Furthermore, it is desirable to bind the signature to the HTTP request. Ideally, this should be done without replicating the information already present in the HTTP request more than required. However, many HTTP application frameworks insert extra headers, query parameters, and otherwise manipulate the HTTP request on its way from the web server into the application code itself. It is the goal of this draft to have a signature protection mechanism that is sufficiently robust against such deployment constraints while still providing sufficient security benefits.

The key required for this signature calculation is distributed via mechanisms described in companion documents (see [I-D.ietf-oauth-pop-key-distribution] and [I-D.ietf-oauth-pop-architecture]). The JSON Web Signature (JWS)

specification [RFC7515] is used for computing a digital signature (which uses asymmetric cryptography) or a keyed message digest (in case of symmetric cryptography).

The mechanism described in this document assumes that a client is in possession of an access token and asociated key. That client then creates a signature of elements of the HTTP request as a JWS, as described in [[draft-richanna-http-request-signing-jws-00]], including the access token as an additional top-level member of the signature's payload JSON object, and issues a request to a resource server for access to a protected resource using the signature as its authorization. The protected resource validates the signature and parses the JSON object to obtain token information.

2. 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 RFC 2119 [RFC2119].

Other terms such as "client", "authorization server", "access token", and "protected resource" are inherited from OAuth 2.0 [RFC6749].

We use the term 'sign' (or 'signature') to denote both a keyed message digest and a digital signature operation.

3. Generating the Signature

This client creates a JSON Web Signature (JWS) over elements of the HTTP request, as described in [[draft-richanna-http-jwt-signature-00]]. The client includes the following additional top-level members within the JSON object payload of the JWS:

at REQUIRED. The access token value. This string is assumed to have no particular format or structure and remains opaque to the client.

The JWS is signed using the algorithm appropriate to the associated access token key, usually communicated as part of key distribution

[I-D.ietf-oauth-pop-key-distribution].

4. Sending the signed object

In order to send the signed object to the protected resource, the client includes it in one of the following three places.

4.1. HTTP Authorization header

The client SHOULD send the signed object to the protected resource in the _Authorization_ header. The value of the signed object in JWS compact form is appended to the _Authorization_ header as a PoP

value. This is the preferred method. Note that if this method is used, the _Authorization_ header MUST NOT be included in the protected elements of the signed object.

```
GET /resource/foo
Authorization: PoP eyJ....omitted for brevity...
```

4.2. HTTP Form body

If the client is sending the request as a form-encoded HTTP message with parameters in the body, the client MAY send the signed object as part of that form body. The value of the signed object in JWS compact form is sent as the form parameter _pop_access_token_. Note that if this method is used, the body hash cannot be included in the protected elements of the signed object.

```
POST /resource
Content-type: application/www-form-encoded

pop_access_token=eyJ....omitted for brevity...
```

4.3. HTTP Query parameter

If neither the _Authorization_ header nor the form-encoded body parameter are available to the client, the client MAY send the signed object as a query parameter. The value of the signed object in JWS compact form is sent as the query parameter _pop_access_token_. Note that if this method is used, the _pop_access_token_ parameter MUST NOT be included in the protected elements of the signed object.

```
GET /resource?pop_access_token=eyJ....
```

5. Validating the request

Just like with a bearer token [RFC6750], while the access token value included in the signed object is opaque to the client, it MUST be understood by the protected resource in order to fulfill the request. Also like a bearer token, the protected resource traditionally has several methods at its disposal for understanding the access token. It can look up the token locally (such as in a database), it can parse a structured token (such as JWT [RFC7519]), or it can use a service to look up token information (such as introspection [RFC7662]). Whatever method is used to look up token information, the protected resource MUST have access to the key associated with the access token, as this key is required to validate the signature of the incoming request. Validation of the signature is done according the rules defined in [[draft-richanna-http-jwt-signature-00]].

6. IANA Considerations

6.1. The 'pop' OAuth Access Token Type

Section 11.1 of [RFC6749] defines the OAuth Access Token Type Registry and this document adds another token type to this registry.

Type name: pop

Additional Token Endpoint Response Parameters: (none)

HTTP Authentication Scheme(s): Proof-of-possession access token for use with OAuth 2.0

Change controller: IETF

Specification document(s): [[this document]]

7. Security Considerations

7.1. Denial of Service

This specification includes a number of features which may make resource exhaustion attacks against resource servers possible. For example, a resource server may need to process the incoming request, verify the access token, perform signature verification, and might (in certain circumstances) have to consult back-end databases or the authorization server before granting access to the protected resource. Many of these actions are shared with bearer tokens, but the additional cryptographic overhead of validating the signed request needs to be taken into consideration with deployment of this specification.

An attacker may exploit this to perform a denial of service attack by sending a large number of invalid requests to the server. The computational overhead of verifying the keyed message digest alone is not likely sufficient to mount a denial of service attack. To help combat this, it is RECOMMENDED that the protected resource validate the access token (contained in the "at" member of the signed structure) before performing any cryptographic verification calculations.

8. Privacy Considerations

This specification addresses machine to machine communications and raises no privacy considerations beyond existing OAuth transactions.

9. Acknowledgements

The authors thank the OAuth Working Group for input into this work.

In particular, the authors thank Justin Richer for his work on [I-D.ietf-oauth-signed-http-request], on which this specification is based.

10. Normative References

[I-D.ietf-oauth-pop-architecture]
Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-08 (work in progress), July 2016.

[I-D.ietf-oauth-pop-key-distribution]
Bradley, J., Hunt, P., Jones, M., Tschofenig, H., and M. Meszaros, "OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution", draft-ietf-oauth-pop-key-distribution-07 (work in progress), March 2019.

[I-D.ietf-oauth-signed-http-request]
Richer, J., Bradley, J., and H. Tschofenig, "A Method for Signing HTTP Requests for OAuth", draft-ietf-oauth-signed-http-request-03 (work in progress), August 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[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/info/rfc7519>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/info/rfc7662>.

Author's Address

Annabelle Backman (editor)
Amazon

Email: richanna@amazon.com