Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 18, 2014

J. Bradley
Ping Identity
P. Hunt
Oracle Corporation
T. Nadalin
Microsoft
H. Tschofenig

January 14, 2014

The OAuth 2.0 Authorization Framework: Holder-of-the-Key Token Usage draft-tschofenig-oauth-hotk-03.txt

Abstract

OAuth 2.0 deployments currently rely on bearer tokens for securing access to protected resources. Bearer tokens require Transport Layer Security to be used between an OAuth client and the resource server when presenting the access token. The security model is based on proof-of-possession: access token storage and transfer has to be done with care to prevent leakage.

There are, however, use cases that require a more active involvement of the OAuth client for an increased level of security, particularly to secure against token leakage. This document specifies an OAuth security framework using the holder-of-the-key concept, which requires the OAuth client when presenting an OAuth access token to also demonstrate knowledge of keying material that is bound to the token.

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 http://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 18, 2014.

Table of Contents

1. Introduction

At the time of writing the OAuth 2.0 [3] and accompanying protocols offer one main security mechanism to access protected resources, namely the bearer token. In [12] a bearer token is defined as

> A security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can. Using a bearer token does not require a bearer to prove possession of cryptographic key material.

The bearer token meets the security needs of number of use cases OAuth had been designed for. There are, however, scenarios that require stronger security properties and ask for active participation of the OAuth client software in form of cryptographic computations when presenting an access token to a resource server.

This specification defines a new security mechanism for usage with OAuth that combines various existing specifications to offer enhanced security properties for OAuth. The incredients for this security solution are:

1. A mechanism for dynamic key distribution.

2. Data elements to bind emphemeral keying material to an access token. For the access token we assume a JSON Web Token (JWT) [6] in this specification to specify a complete solution. Future specifications may make this functionality available to other access token formats as well.

3. A mechanism to allow the OAuth client to demonstrate a proof of possession.

The rest of the document describes how these different components work together.

2. Terminology

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this specification are to be interpreted as described in [1].

3. Protocol Specification

To describe the architecture of the proposed security mechanism it is best to start by looking at the main OAuth 2.0 protocol exchange sequence. Figure 1 shows the abstract OAuth 2.0 protocol exchanges graphically. The exchange in this document will focus on two interactions, namely

1. to allow the client to obtain the ephemeral asymmetric credentails in step (D)

2. to use the obtained asymmetric credentials for the interaction with the resource server in step (E)

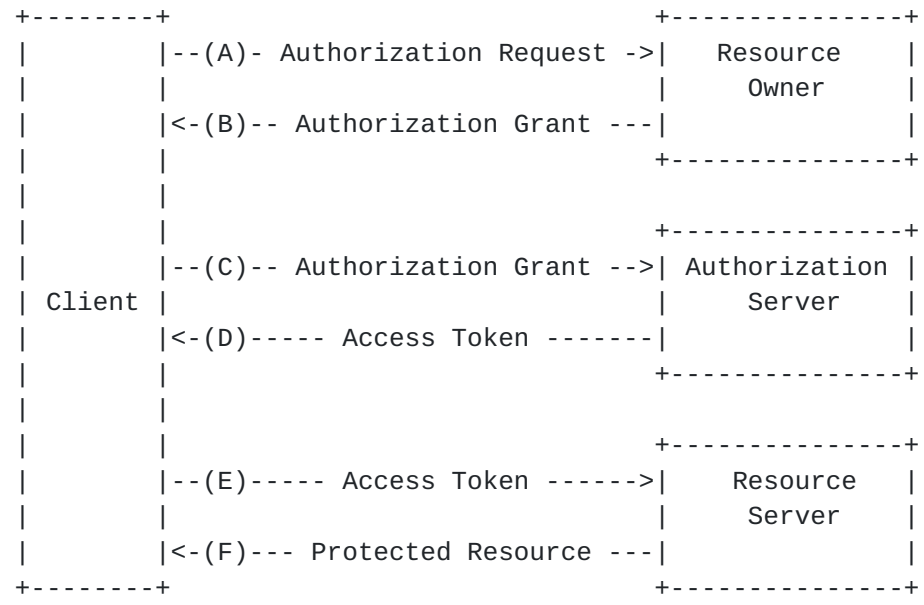

Figure 1: Abstract OAuth 2.0 Protocol Flow

3.1. Binding a Key to an Access Token

OAuth 2.0 offers different ways to obtain an access token, namely using authorization grants and using a refresh token. The core OAuth specification defines four authorization grants, see Section 1.3 of [3], and [11] adds an assertion-based authorization grant to that list.

This document extends the communication with the token endpoint. The token endpoint, which is described in Section 3.2 of [3], is used with every authorization grant except for the implicit grant type. In the implicit grant type the access token is issued directly.

Two types of keying material can be bound to an access token, namely symmetric keys and asymmetric keys, and we explain them in separate sub-sections.

3.1.1. Symmetric Keys

In case a symmetric key shall be bound to an access token then the following procedure is applicable. In the request message from the OAuth client to the authorization server the following parameters MUST be included:

token_type: REQUIRED. For the symmetric holder-of-the-key variant the value MUST be set to "hotk-sk".

profile: REQUIRED. The profile parameter provides information about what mechanisms the client supports to provide proof of possession of the key towards a resource server. The value MUST be taken from the algorithm registry created in Section 5.4. Algorithm names are case-sensitive. If the client supports more than one profile then each individual value MUST be separated by a comma.

For example, the client makes the following HTTP request using TLS (extra line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&token_type=hotk-sk
&profile=jws,mac
```

Example Request to the Authorization Server

If the access token request is valid and authorized, the authorization server issues an access token and optionally a refresh token. If the request client authentication failed or is invalid, the authorization server returns an error response as described in Section 5.2 of [3].

The authorization server MUST include the following parameters in a successful response, if it supports any of the profiles listed by the client.

id: REQUIRED. An ephemeral and unique key identifier. The authorization server MUST NOT select the same key identifier twice within the lifetime of the access token, which is indicated by the 'expires_in' parameter.

key: REQUIRED. A fresh and unique shared symmetric secret with sufficient entrophy.

profile: REQUIRED. The profile parameter provides further information about how the client has to provide proof of

possession of the key with the resource server. The authorization server chooses a value from the list of supported mechanisms supported by the client.

For example:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "access_token":"SlAV.....32hkKG",
  "token_type":"hotk-sk",
  "expires_in":3600,
  "refresh_token":"8xLOxBtZp8",
  "id":"client12345@example.com",
  "key":"adijq39jdlaska9asud",
  "profile":"jws"
}
```

The content of the 'access_token' MUST contain the key identifier value in the 'hotk' element, as shown in the example below.

```
{"typ":"JWT",
 "alg":"HS256"
}
.
{"iss":"authorization-server-id",
 "exp":1300819380,
 "hotk":"client12345@example.com"
}
.
bbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZC
```

DISCUSSION: Should we put the encrypted key into the access token? This would make the mechanism more similar to a Kerberos-based scheme.

The key identifier, the key, and the profile name MUST NOT include characters other than:

```
%x20-21 / %x23-5B / %x5D-7E
; Any printable ASCII character except for <"> and <\>
```

3.1.2. Asymmetric Keys

In case an asymmetric key shall be bound to an access token then the following procedure is applicable. In the request message from the OAuth client to the authorization server the following parameters MUST be included:

token_type: REQUIRED. For the asymmetric holder-of-the-key variant the value MUST be set to "hotk-pk".

pk_info: REQUIRED. This field contains information about the public key the client would like to bind to the access token in the JSON Web Key format. The public key is "application/x-www-form-urlencoded" encoded.

For example, the client makes the following HTTP request using TLS (extra line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&token_type=hotk-pk
&pk_info=eZQQYbYS6WxS...lxlOB
```

whereby the content of the pk_info field represents the following structure:

```
{"keys":
   [
     {"alg":"RSA",
      "mod": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx
 4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMs
 tn64tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2
 QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbI
 SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqb
 w0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
      "exp":"AQAB",
      "kid":"2011-04-29"}
   ]
  }
```

Example Request to the Authorization Server

If the access token request is valid and authorized, the authorization server issues an access token and optionally a refresh token. If the request client authentication failed or is invalid, the authorization server returns an error response as described in Section 5.2 of [3].

The authorization server also places information about the public key used by the client into the access token to create the binding between the two. The new token type, called 'hotk-pk', is placed into the 'token_type' parameter.

An example of a successful response is shown below:

```
 HTTP/1.1 200 OK
 Content-Type: application/json;charset=UTF-8
 Cache-Control: no-store
 Pragma: no-cache

 {
   "access_token":"2YotnFZFE....jr1zCsicMWpAA",
   "token_type":"hotk-pk",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
 }
```

whereby the content of the 'access_token' field, for example, contains an encoded JWT with the following raw structure:

```
{"typ":"JWT",
 "alg":"HS256"}
.
{"iss":"authorization-server-id",
 "exp":1300819380,
 "hotk": {"keys":
 [
   {"alg":"RSA",
    "mod": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx
4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMs
tn64tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2
QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbI
SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqb
w0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
    "exp":"AQAB",
    "kid":"2011-04-29"}
 ]
}
}
.
bbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZC
```

Example Response from the Authorization Server

3.2. Accessing a Protected Resource

Accessing a protected resource depends on the chosen credential type.

3.2.1. Symmetric Keys

When a symmetric key was used as a holder-of-the-key then the client has to demonstrate possession of the key that corresponds to the key identifier found in the access token.

This specification defines three ways for providing this proof of possession, which are indicated as profiles in Section 3.1.1:

jws: When the 'jws' profile is chosen then the client MUST compute the following string by concatenating together, in order, the following HTTP request elements:

1. The HTTP request method in upper case. For example: "HEAD", "GET", "POST", etc.

2. The HTTP request-URI as defined by Section 5.1.2 of [2].

3. The hostname included in the HTTP request using the "Host" request header field in lower case.

4. The port as included in the HTTP request using the "Host" request header field. If the header field does not include a port, the default value for the scheme MUST be used (e.g., 80 for HTTP and 443 for HTTPS).

5. The value of the "ext" "Authorization" request header field attribute if one was included in the request, otherwise, an empty string.

Each element is followed by a new line character (%x0A) including the last element and even when an element value is an empty string. The resulting value MUST be put into the "request" element of a JSON document that is then subject to JWS processing [7]. The resulting JWS structure is put into the body of the HTTP request. A receiving authorization server MUST use the value in the 'kid' structure to identify the shared key and then use that key to verify the keyed message digest. Additionally, the content of the 'request' field needs to be verified against the HTTP header information. If any of these verification steps fail then the request to the protected resource MUST fail with a "401 Unauthorized" error message back to the OAuth client.

The following example shows and the corresponding encoding in a JWS structure:

1) HTTP Request

```
POST /request?b5=%3D%253D&a3=a&c%40=&a2=r%20b&c2&a3=2+q HTTP/1.1
Host: example.com
```

2) JWS Document

```
{"typ":"HOTK-SK",
 "alg":"HS256",
 "kid":"client12345@example.com",
 "timestamp":"2012-07-15T10:20:00.000-05:00" }
.
{"request":"POST/request?b5=%3D%253D&a3=a&c%40=&a2=r%20b&c2&a3=
 2+qexample.com80"}
.
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

JWS Example

mac: When the 'mac' profile is chosen then the client MUST follow the description in [10].

3.2.2. Asymmetric Keys

The client accesses protected resources by presenting the access token to the resource server. It does so via a Transport Layer Security (TLS) secured channel. Since the client had previously bound a public key to an access token it selects this key for usage with TLS as described in [5].

The resource server validates the access token and ensure it has not expired and that its scope covers the requested resource. Additionally, the resource server verifies that the public key presented during the TLS handshake corresponds to the public key that is contained in the access token.

Note that this step confirms that the client is in possession of the private key corresponding to the public key previously bound to the access token. Information about the client authentication may be contained in the token in case the authorization server added this information when it authenticated the client.

4. Security Considerations

4.1. Security Threats

The following list presents several common threats against protocols utilizing some form of tokens. This list of threats is based on NIST Special Publication 800-63 [14]. We exclude a discussion of threats related to any form of registration and authentication.

Token manufacture/modification: An attacker may craft a fake token or modify the token content (such as the authentication or attribute statements), causing a resource server to grant inappropriate access to the attacker. For example, an attacker may modify the token to extend the validity period or the scope to have extended access to information.

Token disclosure: Tokens may contain authentication and attribute statements that include sensitive information.

Token redirect: An attacker uses a token generated for consumption by one resource server to gain access to a different resource server that mistakenly believes the token to be for it.

Token reuse: An attacker attempts to use a token that has already been used with that resource server in the past.

4.2. Threat Mitigation

A large range of threats can be mitigated by protecting the contents of the access token by using a digital signature or a Message Authentication Code (MAC). Consequently, the token integrity protection MUST be sufficient to prevent the token from being modified.

To deal with token redirect, it is important for the authorization server to include the identity of the intended recipients (the audience), typically a single resource server (or a list of resource servers), in the token. Restricting the use of the token to a specific scope is also RECOMMENDED.

The authorization server MUST implement and use TLS. Which version(s) ought to be implemented will vary over time, and depend on the widespread deployment and known security vulnerabilities at the time of implementation. At the time of this writing, TLS version 1.2 [8] is the most recent version. The client MUST validate the TLS certificate chain when making requests to protected resources, including checking the Certificate Revocation List (CRL) [9].

For the interaction between the client and the resource server this specification requires a TLS extension for usage with out-of-band

validation [5] to be used that allows clients to present raw public keys for asymmetric holder-of-the-key usage.

With the usage of the holder-of-the-key concept it is not possible for any party other than the legitimate client to use an access token and to re-use it without knowing the corresponding asymmetric key pair. This mechanism prevents against token disclosure.

With the usage of the asymmetric holder-of-the-key concept the following deployment consideration needs to be taken into consideration. In some deployments, including those utilizing load balancers, the TLS connection to the resource server terminates prior to the actual server that provides the resource. This could leave the token unprotected between the front end server where the TLS connection terminates and the back end server that provides the resource.

Client implementations must be carefully implemented to avoid leaking the ephemeral credentials (either the private key from the asymmetric credential or the shared secret).

Token replay is also not possible since an eavesdropper will also have to obtain the corresponding private key or shared secret that is bound to the access token. Nevertheless, it is good practice to limit the lifetime of the access token and therefore the lifetime of associated key.

4.3. Summary of Recommendations

The following three items represent the main recommendations:

Safeguard the private key/shared secret: Client implementations MUST ensure that the ephemeral private key / shared secret is not leaked to third parties, since those will be able to use the access token together with the keying material to gain access to protected resources.

Switch keying material regularly: Clients can at any time create a new ephemeral credential and associate it with an access token. For example, a client presents a new public key when requesting an access token with the help of a refresh token. Nevertheless, the lifetime of these access token may be longer than the lifetime of bearer tokens.

Issue scoped bearer tokens: Token servers SHOULD issue bearer tokens that contain an audience restriction, scoping their use to the intended relying party or set of relying parties.

5. IANA Considerations

This document requires IANA to take the following actions.

5.1. OAuth Parameters Registration

This specification registers the following parameters in the OAuth Parameters Registry established by [3].

Parameter name: pk_info

Parameter usage location: token request

Change controller: IETF

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

Related information: None

Parameter name: token_type

Parameter usage location: token request, token response, authorization response

Change controller: IETF

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

Related information: None

Parameter name: profile

Parameter usage location: token request, token response, authorization response

Change controller: IETF

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

Related information: None

Parameter name: id

Parameter usage location: token response, authorization response

Change controller: IETF

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

Related information: None

Parameter name: key

Parameter usage location: token response, authorization response

Change controller: IETF

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

Related information: None

5.2. The 'hotk' JSON Web Token Claims

[6] established the IANA JSON Web Token Claims registry for reserved JWT Claim Names and this document adds the 'hotk' name to that registry.

5.3. The 'hotk' OAuth Access Token Type

Section 11.1 of [3] defines the OAuth Access Token Type Registry and this document adds another token type to this registry.

Type name: hotk

Additional Token Endpoint Response Parameters: (none)

HTTP Authentication Scheme(s): Holder of the key confirmation using TLS

Change controller: IETF

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

5.4. Profile Registry

This document asks IANA to create a registry for profiles of symmetric key-based holder-of-the-key mechanisms. The policy for adding new entries to the registry is "Specification Required". IANA is asked to populate the registry with the following values:

- Profile name: jws
- Change controller: IETF
- Specification document(s): [[this document]]
- Profile name: mac

- Change controller: IETF

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

6. Acknowledgements

The author would like to thank the OAuth working group and participants of the Internet Identity Workshop for their discussion input that lead to this document.

7. References

7.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[3] Hardt, D., "The OAuth 2.0 Authorization Framework", draft-ietf-oauth-v2-31 (work in progress), August 2012.

[4] Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-19 (work in progress), December 2013.

[5] Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", draft-ietf-tls-oob-pubkey-10 (work in progress), October 2013.

[6] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-14 (work in progress), December 2013.

[7] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-19 (work in progress), December 2013.

[8] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[9] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[10] Richer, J., Mills, W., Tschofenig, H., and P. Hunt, "OAuth 2.0 Message Authentication Code (MAC) Tokens", draft-ietf-oauth-v2-http-mac-04 (work in progress), July 2013.

7.2. Informative References

[11] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", draft-ietf-oauth-assertions-13 (work in progress), December 2013.

[12] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", draft-ietf-oauth-v2-bearer-23 (work in progress), August 2012.

[13] Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.

[14] Burr, W., Dodson, D., Perlner, R., Polk, T., Gupta, S., and E. Nabbus, "NIST Special Publication 800-63-1, INFORMATION SECURITY", December 2008.

Authors' Addresses

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com

Phil Hunt
Oracle Corporation

Email: phil.hunt@yahoo.com

Tony Nadalin
Microsoft

Email: tonynad@microsoft.com

Hannes Tschofenig

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at