Network Working Group H. Tschofenig
Internet-Draft Nokia Siemens Networks
Intended status: Standards Track July 9, 2012
Expires: January 10, 2013

The OAuth 2.0 Authorization Framework: Holder-of-the-Key Token Usage
draft-tschofenig-oauth-hotk-00.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 in order to get access. The security model is based on proof-of-possession of the access token: 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 to offer increased security, particularly against token leakage. This document specifies an OAuth security framework using ephemeral asymmetric credentials that are bound to the access token. A client can create these key pairs dynamically and use them, after they are bound to an access token by the authorization server, in communication interactions with resource servers.

This document is discussed at https://www.ietf.org/mailman/listinfo/oauth. This initial version of the specification shall serve as a discussion starter.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

At the time of writing the OAuth 2.0 [1] and accompanying protocols offer one main security mechanism to access protected resources, namely the bearer token. In [8] 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 (proof-of-possession).

The bearer token provides sufficient security properties for a number of use cases OAuth had been designed for, if certain conditions are met (which are documented in [8]). Some usage scenarios, however, require stronger security guarantees and ask for active participation of the client software in form of cryptographic computations when presenting an access token.

In addition to the bearer token a MAC token has been specified, see [9]. The design of the MAC token was inspired by features in the OAuth 1.0 [10]. Unfortunately, the MAC token has not received a lot of deployment attention.

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

1. A mechanism for on-the-fly provisioning of ephemeral asymmetric credentials using the JSON Web Key (JWK) format [2].

2. The ability to access a protected resource using this ephemeral asymmetric credentials for client authentication using a transport layer extension that allows out-of-band key validation [3].

3. A data structure to bind the emphemeral asymmetric credential to an access token. The structure uses the JSON Web Token (JWT) [4].

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 [5].

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)

```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

Figure 1: Abstract OAuth 2.0 Protocol Flow

3.1. Binding a Public 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 [1], 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 [1], is used with every authorization grant except for the implicit grant type since an access token is issued directly. The request contains information about the public key the client would like to bind to the

access token in the JSON Web Key format. This parameter also provides an indication to the authorization servers about the support by the client for this specification. Since the client makes a request to the token endpoint by adding a new of parameters using the "application/x-www-form-urlencoded" format in the HTTP request entity-body the public key information must be encoded into a new parameter, called 'pk-info'. A new token type, called 'hotk', is also defined by this specification.

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
&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 [1].

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.

An example successful response:

```
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",
  "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

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 [3].

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 [12]. 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 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 [6] 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) [7]. In addition, this specificatio requires a TLS extension for usage with

out-of-band validation [3] to be used that allows clients to present raw public keys.

With the usage of the holder-of-the-key concept it is not possible for any party other than the legitimate client to use a token and to re-use it without knowing the corresponding asymmetric key pair. This mechanism prevents against token disclosure. 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. Even in such deployments, token leakage is not a problem.

Client implementations must be carefully implemented to avoid leaking the ephemeral asymmetric key pair.

Token replay is also not possible since an eavesdropper will also have to obtain the corresponding asymmetric key pair that corresponds to the access token. Nevertheless, it is good practice to limit the lifetime of the and therefore the lifetime of the ephemeral asymmetric key associated with it.

4.3. Summary of Recommendations

The following three items represent the main recommendations:

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

Switch keying material regularly: Clients can at any time create a new ephemeral key pair and request the public key to be associated with the 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 'pk-info' parameter in the OAuth Parameters Registry established by [1].

Parameter name: pk-info

Parameter usage location: token request

Change controller: IETF

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

Related information: None

5.2. The 'hotk' JSON Web Token Claims

[4] 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 [1] 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]]

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] Hammer-Lahav, E., Recordon, D., and D. Hardt, "The OAuth 2.0 Authorization Framework", draft-ietf-oauth-v2-28 (work in progress), June 2012.

[2] Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-03 (work in progress), July 2012.

[3] Wouters, P., Gilmore, J., Weiler, S., Kivinen, T., and H. Tschofenig, "TLS Out-of-Band Public Key Validation", draft-ietf-tls-oob-pubkey-03 (work in progress), April 2012.

[4] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-01 (work in progress), July 2012.

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

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

[7] 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.

7.2. Informative References

[8] Jones, M., Hardt, D., and D. Recordon, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", draft-ietf-oauth-v2-bearer-21 (work in progress), June 2012.

[9] Hammer-Lahav, E., "HTTP Authentication: MAC Access Authentication", draft-ietf-oauth-v2-http-mac-01 (work in progress), February 2012.

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

[11] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0", draft-ietf-oauth-assertions-04 (work in progress), July 2012.

[12] Burr, W., Dodson, D., Perlner, R., Polk, T., Gupta, S., and E.

Nabbus, "NIST Special Publication 800-63-1, INFORMATION
SECURITY", December 2008.

Author's Address

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at