Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 3, 2019

O. Levi Hevroni
Soluto by Asurion

August 02, 2018

Seamless OAuth 2.0 Client Assertion Grant
draft-hevroni-oauth-seamless-flow-01

Abstract

This specification defines the use of a One Time Password, encoded as JSON Web Token (JWS) Bearer Token, as a means for requesting an OAuth 2.0 access token as well as for client authentication.

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 February 3, 2019.

Table of Contents

1. Introduction

1.1. Motivation

Authentication is a crucial part of modern application. There are various authentication methods for client side applications, and all those methods requires user interaction (e.g. login). This is due to the fact that there is no secure way to embed credentials in the application code.

While asking the user to login in order to authenticate the app is a strong authentication solution, it has impact on the application behavior. A login is just another step the user has to complete in order to use the apps, which users don’t always like to fulfill.

Also, there are cases for applications without any UI, for example - Internet of Things applications. For those applications, adding a login steps could be a challenge.

In this document, we propose an extension to OAuth 2.0 protocol that provides a new authentication grant dedicated for those cases. This grant will allow an application to use strong authentication solution without user interaction.

This document defines how a One Time Password, encoded in a JWS, can be used to authenticate the client. In order for the client to perform an authentication request, an initial registration step is required. This registration step is not part of this protocol, and should be defined by the authorization server.

1.2. Target Audience

The protocol requires the app to be able to persist state in a secure, sand-boxed, persisted storage. It is possible to use this protocol for web application, although it is not recommended. This protocol is targeted for mobile or IoT devices where it is possible (although not always simple) to achieve such storage. See Security Consideration section for more details.

1.3. Existing Solutions

There are alternatives to this protocol, this section will discuss them. Interactive grants (authorization code, resource owner etc) will not be discussed.

1.3.1. Client Credentials grant

This grant (as defined in [RFC6749]) allows applications to authenticate without user interaction. It is intend to be used by applications running on trusted environment. Mobile applications are not running on trusted environment, and therefor should not use this grant. See the Security section for discussion on the various threat and how this protocol mitigate them. Also refer to section 10.1 in [RFC6749], which strongly advise against using this grant on native applications.

1.3.2. Device grant

This grant is for Browserless and Input Constrained Devices. In this grant the login is performed on a different device, which could handle interactive login. Therefore, it still requires user interaction, which this protocol aims to avoid.

1.3.3. JWT Client Assertion

This grant (as defined in [RFC7523]) could be used by mobile application for seamless authentication. The grant used signed JWT (see [RFC7519]) to authenticate the client. It has two disadvantages when compared with this grant:

- Significant part of the security of the protocol is the expiration date of the JWT. In case a hacker was able to obtain a JWT, she will be able to perform authentication request until the JWT expires. Therefore, it is advised to use as shorter expiration time as possible. Time can be a challenge on mobile devices, which are not always synchronized with the global time. Usage of JWT would require the authorization server to allow very long JWT expiration time.

- Detecting Compromised Signing Key. As discussed on the security section, this protocol allows the authorization server to detect compromised signing key. See the discussion there for reference. This mitigation does not exist in JWT client assertion grant.

1.4. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in [RFC2119].

The term "device" used in this document refer to the physical appliance used by the user, which the application code is running on.

2. Note to Readers

> *Note to the RFC Editor:* Please remove this section prior to publication.

Development of this draft takes place on Github at: https://github.com/Soluto/oauth-seamless-flow [1].

3. HTTP Parameter Bindings for Transporting Assertions

The OAuth Assertion Framework [RFC7521] defines generic HTTP parameters for transporting assertions (a.k.a. security tokens) during interactions with a token endpoint. This section defines specific parameters and treatments of those parameters for use with JWS (as defined in [RFC7515]) Bearer Tokens.

3.1. Using OTP JWS for client authentication

To use a OTP JWS, the client first need to generate the OTP as defined in section "JWS format and request processing". Than, the client need to use the following parameter values and encodings.

The value of the "client_assertion_type" is "urn:ietf:params:oauth:client-assertion-type:JWS-otp".

The value of the "client_assertion" parameter contains a single JWS, as defined in [RFC7515]. It MUST NOT contain more than one JWS.

The following example demonstrates client authentication using a JWS during the presentation of an authorization code grant in an access token request (with extra line breaks for display purposes only):

```
POST /token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=token id_token&&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
client-assertion-type%3AJWS-otp&
client_assertion=eyJhbGciOiJSUzI1NiIsImtpZCI6IjIyIn0.
eyJpc3Mi[...omitted for brevity...].
cC4hiUPo[...omitted for brevity...]
```

4. JWS format and request processing

4.1. One Time Password generation

To generate one time password (OTP) as defined in [RFC2289], the client use its state, created during the registration request, which is not covered in this document. The state consist from 2 numbers: "previous" and "next". Each of those numbers can hold signed int, up to 64 bytes length. In order to generate a new JWS, the client has to roll this payload. The rolling is done by setting the value of "previous" to the value of "current", and setting new crypto random, as defined in [RFC4086], value to "next". For example, assuming this is the current state of the app:

```
previous: 1
next: 2
```

After rolling, this will be the payload:

```
previous: 2
next: 5
```

4.2. Creating the JWS

After rolling the payload, the client can create the JWS. This is the format of the JWS payload:

```
{
    previous: 2
    next: 5
    client-id: 89
}
```

Where "client-id" is the id used when this client first registered. All the fields are required. Any other fields besides those will be ignored. To sign the JWS, the client use its own key, which was generated during the registration of this client.

4.3. Request processing

In order to issue an access token response as described in OAuth 2.0 [RFC6749], the authorization server MUST validate the JWS according to the criteria below. Application of additional restrictions and policy are at the discretion of the authorization server. After decoding the JWS and extracting the "client-id", the server will fetch:

- The key correspond to this client, received on the registration request
- The current state of this client, from the last successful request, or from the registration

The server verifies that the JWS is valid, by using the client’s key. If the signature is valid, the server can validate the payload:

- If the client’s "previous" is equals to the server "new", the request is valid. The server will issue a token, as specified in OAuth 2.0 [RFC6749]
- If the client "previous" equals to the server "previous", and the client "next" equals to the server "next", the server construct an error response as defined in OAuth 2.0 [RFC6749]
- Any other case will be treated by the server as an indication of a malicious attack, and should be reported accordingly. The server construct an error response as defined in OAuth 2.0 [RFC6749]

5. Security Considerations

This protocol was designed for mobile application. The following sections will discuss threats which are relevant for mobile applications and are mitigated by this protocol.

5.1. Replay Attacks

Due to the usage of OTP, a replay attack is not feasible. If an attacker will try to replay authentication request, an error response will return. Also, because of how the OTP is generated, guessing it is almost impossible (see the OTP Generation section). Refer to the Request processing section for more details.

5.2. Compromised Signing key

As the application is running on a mobile device, an attacker can gain physical access to the device. In such a scenario, the attacker will be able to compromise it and retrieve the state and the signing key. This will allows the attacker to impersonate the device and request an access token. The attacker will be able to authenticate as until the first time the device will try to authenticate. When the device will try to authenticate, the request will fail. It will fail because the state on the authorization server will match the attacker’s state, not the one on the device.

The device authentication request will revoke the client (see Request processing section). This will cause both the device and the attacker to not be able to perform authentication request. In such cases, an alternative flow is required in order to allow the device to authenticate. Such a flow is not part of this standard.

In order for this mitigation to be effective, the device must to perform an authentication request on a regular basis. The period between authentication requests should be 24 hours or less, depend on the client.

5.3. Man in the Middle

Performing Man in the Middle (MitM) attack on mobile application is relatively simple. It is highly recommended to use TLS [RFC5246] for all authentication requests. It is also recommended to implement Certificate Pinning for all the requests. For more details, please refer to this guide [2] by OWASP.

5.4. Reverse Engineering

The mobile application code is publicly available, which make reverse engineering a simple task. This attack is irrelevant to this protocol. No sensitive data should be embedded in the application code. All that is required for the authentication request should be generated on the device.

5.5. OTP Generation

The security of the OTP is as strong as the randomness used to generate it. Only strong, secure random implementation (as described in [RFC4086]) should be used. Usage of weak random protocol will allow the attacker to guess the numbers generated by the client, and by that generates the OTP herself. The state ("next" and "new") is not considered a secret. Compromise of state only, without the signing key, will not allows the attacker to perform authentication request. It is still advised to store them securely, and follow the operating system recommendation (iOS [3], Android [4]).

5.6. Signing Key Consideration

5.6.1. Generation and Storage

A fundamental part of the security of the protocol is the key used to sign the JWS. The key should be generated and stored in a secure wat, and if possible to use the tools provided by the OS. On iOS, use Keychain [5] to generate and store the key. On Android, the best option is the Keystore [6], but due to implementation limitations (see this post [7] for example), it is advised to use OpenSSL.

5.6.2. Algorithm

Asymmetric encryption and signing algorithms are preferred over symmetric ones. The main advantages of such protocol is that the private key never leaves the device. Even if an attacker was able to capture the public key (either in transit or by compromising the authorization server), she will not be able to use it to perform authentication request. For any algorithm that is chosen, a strong key should be generated. In case of RSA, 2048 bytes is the minimum key size.

6. IANA Considerations

TODO IANA

7. References

7.1. Normative References

[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>.

[RFC2289] Haller, N., Metz, C., Nesser, P., and M. Straw, "A One-Time Password System", STD 61, RFC 2289, DOI 10.17487/RFC2289, February 1998, <https://www.rfc-editor.org/info/rfc2289>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[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>.

[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>.

[RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <https://www.rfc-editor.org/info/rfc7521>.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <https://www.rfc-editor.org/info/rfc7523>.

7.2. URIs

[1] https://github.com/Soluto/oauth-seamless-flow

[2] https://www.owasp.org/index.php/
Certificate_and_Public_Key_Pinning

[3] https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[4] https://developer.android.com/training/articles/security-
tips.html#UserData

[5] https://developer.apple.com/documentation/security/
keychain_services/keychains

[6] https://developer.android.com/training/articles/keystore.html

[7] https://doridori.github.io/android-security-the-forgetful-
keystore/#sthash.CgPjGF4h.dpbs

Author’s Address

Omer Levi Hevroni
Soluto by Asurion

Email: omerlh@gmail.com