Workgroup: Network Working Group
Internet-Draft:
draft-demarco-oauth-nonce-endpoint-00
Published: 6 February 2024
Intended Status: Informational
Expires: 9 August 2024
Authors: G. D. Marco Independent
O. Steele Transmute

OAuth 2.0 Nonce Endpoint

Abstract

This document defines the Nonce Endpoint for OAuth 2.0 implementations [RFC6749]. It details how an Authorization Server generates and issues opaque Nonces and how a client can learn about this endpoint to obtain a Nonce generated by the Authorization Server.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://peppelinux.github.io/draft-demarco-nonce-endpoint/draft-demarco-nonce-endpoint.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-demarco-oauth-nonce-endpoint/.

Source for this draft and an issue tracker can be found at https://github.com/peppelinux/draft-demarco-nonce-endpoint.

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 9 August 2024.

Table of Contents

1. Introduction

This specification presents a comprehensive guide to the Nonce endpoint in OAuth 2.0 implementations [RFC6749]. It describes in detail how a client can request and receive a server-generated Nonce, which is a unique, one-time use, opaque string. This document provides in-depth insights into the cryptographic methods used in generating Nonces to protect the confidentiality of the information associated with them. In addition, it is a great resource for developers and system architects who desire to strengthen the scalability, security, and efficiency of their systems while using OAuth 2.0.

2. Conventions and Definitions

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Terminology

Nonce: A random or pseudo-random number that is generated for a specific use, typically for cryptographic communication. The Nonce is used to protect against replay attacks by ensuring that a message or data cannot be reused or retransmitted. The term "Nonce" stands for "number used once" and it **MUST** be unique within some scope.

Nonce Issuer: The entity that generates and provides the Nonce. In the context of OAuth 2.0, the Nonce Issuer would typically be the Authorization Server.

Nonce Endpoint: The HTTP endpoint provided by the Nonce Issuer for the issuance of the Nonces.

4. Requirements

The **Nonce Endpoint MUST** satisfy the following requirements:

*Employ TLS for securitung the Endpoint [RFC5246].

The **Nonce MUST** satisfy the following requirements:

*Pure opacity to recieving Clients.

The **Nonce Issuer MUST** satisfy the following requirements:

*Generate a unique Nonce for each request to ensure the Nonce Issuer never produces identical Nonces, regardless of whether they occur simultaneously or at different times;

*Encrypt the Nonce with an encryption key that:

-**MUST NOT** supplied by the Nonce Issuer to the Client;

-**MUST NOT** disclosed by the Nonce Issuer to any external entity beyond its domain.

The **audiences of the Nonces** satisfies the following requirements:

* The servers, within the Nonce Issuer's domain, **SHOULD** decrypt the Nonce and access its decrypted contents. No other entity might decrypt or know the decrypted contents of the Nonce.

5. Nonce Request

When a Client needs a Nonce, it sends an HTTP GET request to the Nonce Endpoint.

Below is a non normative example of the HTTP Request made by a Client to the Nonce Endpoint.

```
GET /nonce HTTP/1.1
Host: server.example.com
```

6. Nonce Response

The Nonce Endpoint provides a Nonce to the Client, encapsulated within a JSON object [RFC7159]. The response **MUST** use the HTTP Header Content-Type value set to application/json and **MUST** provide in the response message a JSON object with the member nonce.

Below is a non-normative example of the response given by a Nonce Endpoint:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "nonce": "d2JhY2NhbG91cmVqdWFuZGFt"
}
```

7. Nonce Endpoint Discovery

When an Authorization Server requires the use of a Nonce in the request for a specific resource and the Client does not provide it in its request, the Authorization Server **MUST** return an HTTP response with the HTTP status code 400 and an error field with the value set to "nonce_required".

This response **MUST** also contain the Nonce-Endpoint-URI HTTP header, with the value set to the URL corresponding to the Nonce Endpoint, where the Client **SHOULD** request and fetch a new Nonce. Once the Nonce is received, the Client **MAY** renew the request to the Authorization Server, including the obtained Nonce.

Below is a non-normative example of an error response issued by an Authorization Server that requires the Nonce in the Client request,

the response informs the Client about the Nonce Endpoint where the Nonce should be requested:

```
HTTP/1.1 400 Bad Request
Nonce-Endpoint-URI: https://server.example.org/nonce-endpoint

{
  "error": "nonce_required",
  "error_description":
    "Authorization server requires the nonce in the request"
}
```

In cases where, for some reasons, a correctly issued Nonce can no longer be considered valid by the Authorization Server that receives it, the Authorization Server **MUST** return the generic error "nonce_required" reporting the same description as "error_description", as if the Nonce had not been received. The cases when an issued Nonce is considered no longer valid **MAY** be caused by the rotation of the encryption keys, its expiration or other specific conditions internal to an implementation.

8. Non-normative Examples of a Nonce Payload

The decrypted Nonce payload may use different formats and encodings, according to the different implemententative requirements, and contains any kind of implementation-specific claims, such as the issuance time, the time of expiration, the audiences and other where needed.

Below are provided some non-normative examples, describing how a decrypted and JSON serialized Nonce payload may appear:

```
{
  "iss": "https://server.example.org",
  "iat": 1615908701,
  "exp": 1615995101,
  "source_endpoint": "https://server.example.org/nonce-endpoint",
  "aud": [
    "https://service.example.com/endpoint",
    "https://another.example.com/cb"
  ]
}
```

Please note that the values represented in the previous examples may depend on domain specific requirements and **MUST NOT** be intended as normative.

9. Security Considerations

The Nonce Endpoint **MUST** be protected by TLS to prevent eavesdropping and man-in-the-middle attacks, therefore the practices defined in [BCP195] should be followed.

The Nonce Issuer **MUST** securely generate and store the encryption key used to encrypt the Nonce. The robustness of the encryption key plays a crucial role in the security of the Nonce Endpoint. The following considerations **MUST** be taken into account:

1. **Key Strength**: The cryptographic key used to encrypt the Nonce requires sufficient length to withstand brute-force attacks. A key length of 256 bits has been proposed as a common practice to ensure a minimum level of security.

2. **Key Management**: The cryptographic key requires secure management, which includes secure generation, storage, and revocation. Access to the key necessitates strict control, with access granted only to authorized entities.

3. **Key Rotation**: Regular key rotation is a good practice to mitigate the risk of key compromise. The frequency of key rotation depends on the specific requirements and threat model, but a common practice is to rotate keys frequently.

4. **Randomness**: To assure the randomness of the cryptographic key, it requires the usage of a safe random number generator. Attackers can simply guess predictable keys.

5. **Secure Transmission**: If the cryptographic key needs to be transmitted over a network and within the Nonce Issuer domain, it requires the usage of secure protocols such as TLS.

6. **Backup and Recovery**: Cryptographic keys require secure backup and recovery mechanisms. This ensures that the key can be retrieved in the event of its loss while also prohibiting unauthorised access to the backup.

The security of the Nonce Endpoint is only as strong as the security of the encryption key. Therefore, proper key management practices are essential.

10. Considerations about Nonce vs. jti

This section provides some thought about the main differences and scopes of the Nonce in compared to the jti claim defined in [RFC7519].

Both jti and Nonces are used to prevent replay attacks, however Nonces offer more implementation flexibility and are considered best practice. They can be created and managed stateless (e.g., by issuing the hmac over the current time as the Nonce), as this document outlines.

The main differences between the use of the jti and the Nonces can be summarized as follows:

1. **Generation**: Nonces are generated by the server, while jti is generated by the Client.

2. **Storage**: Nonces can be self-authenticating and self-contained and therefore need not be stored. A common way to achieve this is for the Nonce to contain content encrypted to the Authorization Server that creates it. On the other hand, checking jti properly definitely requires a store that is shared across all domains that the associated JWT can be presented in.

3. **Lifetime**: The life span difference between a Nonce and a jti is significant. Nonces are kept just until the Client responds, which happens practically immediately after they are obtained, resulting in a very short lifespan. A jti, on the other hand, must be stored until the expiration window of its associated JWT expires, which is a substantially longer length than that of a Nonce.

4. **Security**: Nonces prevent replay attacks by ensuring that the proof of possession is fresh. On the other hand, jti does not guarantee freshness and using client-generated timestamps has problems, even for non-attacking Clients (e.g. devices with incorrect time-zones or daylight saving settings).

11. IANA Considerations

This document has no IANA actions.

12. References

12.1. 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/rfc/rfc2119>.

[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/rfc/rfc5246>.

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

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/rfc/rfc7159>.

[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/rfc/rfc7519>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

12.2. Informative References

[BCP195] Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, March 2021.
Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, November 2022.
<https://www.rfc-editor.org/info/bcp195>

Acknowledgments

TODO acknowledge.

Authors' Addresses

Giuseppe De Marco
Independent

Email: demarcog83@gmail.com

Orie Steele
Transmute

Email: orie@transmute.industries