Network Working Group
Internet-Draft
Intended status: Informational
Expires: 16 November 2025

S. Knauer

15 May 2025

Secure Webhook Token (SWT)
draft-knauer-secure-webhook-token-00

Abstract

The Secure Webhook Token (SWT) is a specialized JSON Web Token (JWT) format designed for securely authorizing and verifying webhook requests. It defines a set of claims to ensure that the token is explicitly tied to webhook events and that its integrity can be reliably validated by the recipient. A key feature of SWT is the introduction of a unique claim, webhook, which encapsulates webhook-specific information, such as the event type.

By providing a structured and secure approach to webhook authentication, SWT aims to be a lightweight and flexible solution while maintaining compatibility with typical JWT structures. It is designed with the best practices in mind and may serve as a foundation for future standardization efforts.

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 16 November 2025.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

The increasing use of webhooks requires a secure, standardized approach to authorizing webhook requests and verifying the sender’s authenticity. The SWT specifies a JWT-based [RFC7519] structure, which includes unique claims tailored for webhook events. This specification mandates that all tokens must be transmitted using either HTTP HEAD or HTTP POST requests (See Section 9.3.1 of [RFC9110]), ensuring minimal data exposure and optimal compatibility with typical HTTP implementations.

1.1. Notational Conventions

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 "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119]. The interpretation should only be applied when the terms appear in all capital letters.

2. Secure Webhook Token (SWT) Overview

This specification defines required claims within the JWT [RFC7519] payload, prescribes the transport protocol as HTTP HEAD (RECOMMENDED) or HTTP POST, and suggests a maximum payload size of 6kB for HEAD requests to remain efficient and compatible with the widely used default 8kB HTTP header size limit. These constraints ensure interoperability across diverse systems and help prevent issues on servers with unmodified configurations.

To support larger payloads or non-JSON content, HTTP POST can be used as an alternative transport method, allowing the event data to be sent in the request body.

3. Transport Requirements

SWTs must be transmitted via HTTP HEAD OR POST requests only. The HEAD method MUST be used if the event data size is less than or equal to 6kB and contains valid JSON. Otherwise, the POST method MUST be used.

SWTs MUST be transmitted using either HTTP HEAD or HTTP POST requests.

* The HEAD method SHOULD be used when the event data is valid JSON and the total header size is reasonably small (e.g., <= 6kB). This is based on the default 8kB header size limit enforced by many web servers.

* The POST method MUST be used for larger payloads, non-JSON data, or when the header size exceeds what is supported by the server.

| Note: There is no hard limit on the size of a HEAD request, but practical constraints exist. For example, implementations such as Go's http.Server allow configurable limits (e.g., up to 1MB headers), though such configurations should be applied cautiously due to security considerations.

4. Token Size Limitation

Recommended Maximum Token Size: While there is no strict maximum size for an SWT, a recommended size limit of 6kB is proposed for the event data, so that the total size of the SWT is less than 8KB, which is the default max header size of most existing HTTP servers. This is based on the primary use case for SWTs, which is to securely authorize and trigger events rather than transmit extensive data. The token SHOULD be kept concise, containing only essential claims to support efficient processing and minimize resource usage.

Header Size Considerations: In practice, many web servers impose a default maximum HTTP header size of 8KB. If servers are not configured to allow larger headers, exceeding this size MAY lead to transmission errors or dropped requests. However, maintaining a compact token design aligns with this specification's intent to facilitate secure and lightweight webhook interactions.

5. JWT Structure

An SWT comprises the following:

5.1. Header

* *alg*: Specifies the signing algorithm [RFC7518], typically "HS256" (HMAC SHA-256).
* *typ*: Specifies the token type, which MUST be "JWT".

5.2. Payload

The payload contains the following required claims, carefully chosen to meet security and identification needs:

* *webhook*: Custom claim specific to this specification.
  - *event*: Describes the webhook event type (e.g., "payment.received"). Values should be concise to meet the size limitations.

- *data* (OPTIONAL): Describes the webhook data itself, which is related to the aforementioned *event*.

* *iss* (Issuer): Identifies the token issuer (e.g., a unique name or domain representing the entity). Short, meaningful identifiers are RECOMMENDED.

* *sub* (Subject) (OPTIONAL): While traditionally used to identify the principal subject of a JWT, the sub claim is not required in SWT. The purpose and intent of the token are unambiguously defined by the webhook claim. However, implementers may include sub to identify the system, service, or entity responsible for issuing the token, if such context is useful.

* *exp* (Expiration): Specifies the token’s expiration time as a Unix timestamp. This prevents tokens from being valid indefinitely.

* *nbf* (Not Before): Specifies the earliest time the token is considered valid, using a Unix timestamp.

* *iat* (Issued At): The timestamp when the token was issued, aiding in token age validation.

* *jti* (JWT ID): A unique identifier for each token to prevent replay attacks. A UUID [RFC9562] or similarly unique identifier is RECOMMENDED.

5.2.1. Example Payload Structure

5.2.1.1. SWT payload with an event only:

```
{
  "webhook": {
    "event": "ping"
  },
  "exp": 1733987961,
  "nbf": 1733987661,
  "iat": 1733987661,
  "iss": "swt.example.com",
  "jti": "2020B14D-C365-4BCF-84CD-5D423E0C6687"
}
```

5.2.1.2. SWT payload with event and data:

```
{
  "webhook": {
    "event": "delivery.scheduled",
    "data": {
      "carrier": "UPS",
      "scheduledDeliveryDate": "2024-12-24T12:24Z",
      "recipient": {
        "name": "Lucky Kid",
        "address": {
          "locality": "Somewhere",
          "postalCode": "0815",
          "street": "Happy Place 1"
        },
        "sender": {
          "name": "Santa Claus",
          "address": {
            "locality": "North Pole",
            "postalCode": "88888",
            "street": "123 Elf Road"
          }
        }
      }
    }
  },
  "exp": 1733987961,
  "nbf": 1733987661,
  "iat": 1733987661,
  "iss": "swt.example.com",
  "jti": "2020B14D-C365-4BCF-84CD-5D423E0C6687"
}
```


6. Security Considerations

1. *HTTPS Transport*: It is RECOMMENDED that SWT transmissions occur over HTTPS to ensure the token’s confidentiality.

2. *Replay Protection*: The jti claim SHOULD be checked for uniqueness on the server side to prevent replay attacks. Each jti value SHOULD only be accepted once.

3. *Expiration Enforcement*: Systems MUST validate the exp claim to ensure that tokens cannot be used beyond their intended validity.

4. *Respect Header Limits*: Excessively large headers SHOULD be avoided. HEAD method is ideal for compact, JSON-only payloads.

7. Interoperability Considerations

SWT's RECOMMENDED 6kB size target and support for both HEAD and POST methods ensure broad compatibility. The structure is simple to parse and easy to integrate with existing JWT validation libraries.

8. Signing and Encryption Recommendations

To maintain a balance between security and usability, SWTs are RECOMMENDED to be used as JSON Web Signatures (JWS) [RFC7515], specifically signed JWTs. This provides data integrity and allows the recipient to verify the token's authenticity. The following signing algorithms are suggested for SWTs, as they offer a secure yet practical approach:

* HS256, HS384, and HS512: Symmetric algorithms using HMAC with SHA-256, SHA-384, and SHA-512, respectively. These algorithms are considered secure and performant for most applications, especially when using shared symmetric keys between the issuer and the receiver.

While these symmetric algorithms are RECOMMENDED for ease of use and efficiency, other supported algorithms, including asymmetric methods such as RS256 (RSA with SHA-256) or ES256 (ECDSA with SHA-256), MAY also be used if needed based on security requirements or system constraints.

8.1. Encrypted JWT Support (JWE)

In addition to JWS, SWT allows the use of the JSON Web Encryption (JWE) [RFC7516] standard if encrypted JWTs are required to protect sensitive data within the token. JWE adds encryption layers to JWTs, providing confidentiality as well as integrity protection, which may be appropriate in contexts where tokens contain sensitive information.

9. IANA Considerations

This document updates the IANA "JSON Web Token Claims" registry for JWT Claim Names. Following the format in Section 10.1.1 of [RFC7519], the following should be registered.

* Claim Name: "webhook"
* Claim Description: Webhook
* Change Controller: IESG
* Specification Document(s): Section 9.1

9.1. JWT webhook claim

The webhook claim is a JSON object that describes the event to be triggered and, optionally, the data related to that event.

It consists of the following fields:

* event: A case-sensitive string describing the name of the webhook event. It should be concise and specific (e.g., "order.created").
* data (OPTIONAL): Describes the event data. The handling of this field depends on the transport method used (HEAD or POST):

| Although NOT RECOMMENDED, it may be necessary to send non-JSON
| event data via SWT. The HTTP _Content-Type_ header SHOULD be set
| accordingly if using the POST method.

9.1.1. Case 1: HTTP HEAD Requests

If the event data is valid JSON and small (<= 6kB), it MAY be included inline in the data field of the JWT. In this case:

* Use of data is OPTIONAL.
* Sending "data": null is equivalent to omitting the field.
* This is the RECOMMENDED case, as it minimizes complexity and improves performance.

9.1.2. Case 2: HTTP POST Requests

For larger payloads or non-JSON content types, the data MUST be sent in the POST request body. The data field in the JWT’s webhook claim must contain a descriptor object with the following fields:

* hash: The hash digest of the body payload (base64- or hex-encoded).
* size: The exact byte size of the event data to be sent in the request body.
* hashAlg (RECOMMENDED): The algorithm used to compute the hash. If omitted, the default SHOULD be assumed to be sha3-256.

9.1.2.1. Supported hashAlg values:

The hashAlg value MUST be one of the following:

* sha256
* sha384
* sha512
* sha3-256
* sha3-384

* sha3-512

Implementers MAY restrict the set of supported algorithms based on their security posture.

| If hashAlg is omitted, receivers SHOULD assume the use of sha3-256 by default. Senders and receivers SHOULD agree on supported algorithms in advance.

9.1.3. Example of JWT webhook claim for HEAD Method

```
"webhook": {
  "event": "delivery.scheduled",
  "data": {
    "carrier": "UPS",
    "scheduledDeliveryDate": "2024-12-24T12:24Z",
    "recipient": {
      "name": "Lucky Kid",
      "address": {
        "locality": "Somewhere",
        "postalCode": "0815",
        "street": "Happy Place 1"
      },
      "sender": {
        "name": "Santa Claus",
        "address": {
          "locality": "North Pole",
          "postalCode": "88888",
          "street": "123 Elf Road"
        }
      }
    }
  }
}
```

9.1.4. Example of JWT webhook claim for POST Method

```
"webhook": {
  "event": "issue.opened",
  "data": {
    "hash": "ab6e46d9c488...",
    "hashAlg": "sha3-256",
    "size": 6123
  }
}
```

10. Validation

To validate an SWT, the receiving system MUST distinguish between the HTTP method used (HEAD or POST):

1. HTTP HEAD method. Sending an SWT via HTTP HEAD method, the received SWT is valid only if the JWT itself is valid.
2. HTTP POST method. Sending an SWT via HTTP POST method, the received SWT MUST be a valid JWT and contain the webhook claim with the hash and size values of event data to be sent in the body. If either the size or the hash of the payload is different from the ones specified within the SWT webhook claim then it MUST be rejected.

If validation fails, the server SHOULD respond with a 400 Bad Request or 403 Forbidden, depending on the context.

10.1. Examples:

10.1.1. Case 1: Data size is smaller <= 6kB and is sent via HTTP-HEAD request directly with additional data

Webhook claim:

```
    {
     "webhook": {
       "event": "delivery.scheduled",
       "data": {
         "carrier": "UPS",
         "scheduledDeliveryDate": "2024-12-24T12:24Z",
         "recipient": {
           "name": "Lucky Kid",
           "address": {
             "locality": "Somewhere",
             "postalCode": "0815",
             "street": "Happy Place 1"
           },
           "sender": {
             "name": "Santa Claus",
             "address": {
               "locality": "North Pole",
               "postalCode": "88888",
               "street": "123 Elf Road"
             }
           }
         }
       }
     }
   }
```

HEAD request:

```
HEAD / HTTP/1.1
Content-Type: application/json
Authorization: Bearer eyJ0eXAiOiJKV...
Host: localhost
```

10.1.2. Case 2: Data size is > 6kB send the JWT webhook claim MUST contain only "hash" and "size" values of the data to be sent. The actual data is send in the POST request body.

Webhook claim:

```
{
  "webhook": {
    "event": "issue.opened",
    "data": {
      "hash": "ab6e46d9c488...",
      "size": 6123
    }
  }
}
```

POST request:

```
POST / HTTP/1.1
Content-Length: 6123
Content-Type: application/json
Authorization: Bearer eyJ0eXAiOiJK...
Host: localhost

{
   "action": "opened",
   "issue": {
     "url": "https://api.github.com/repos/octocat/...",
     "number": 1347,
     ...
   },
   "repository" : {
     "id": 1296269,
     "full_name": "octocat/Hello-World",
     "owner": {
       "login": "octocat",
       "id": 1,
       ...
     },
     ...
   },
   "sender": {
     "login": "octocat",
     "id": 1,
     ...
   }
 }
```

11. Conclusion

The Secure Webhook Token (SWT) format provides a secure, interoperable, and efficient solution for webhook authentication. Its focus on signed payloads, minimal overhead, and clear transport guidance makes it suitable for modern web service ecosystems.

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/info/rfc2119>.

[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>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[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>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

12.2. Informative References

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

Author’s Address

Stephan Knauer
Leipzig, SN
Germany
Email: mail@knauer.consulting