Stream: Internet Engineering Task Force (IETF)
RFC: 9767
Category: Standards Track
Published: April 2025
ISSN: 2070-1721
Authors: J. Richer, Ed. *Bespoke Engineering*
F. Imbault *acert.io*

RFC 9767
Grant Negotiation and Authorization Protocol Resource Server Connections

Abstract

The Grant Negotiation and Authorization Protocol (GNAP) defines a mechanism for delegating authorization to a piece of software (the client) and conveying the results and artifacts of that delegation to the software. This extension defines methods for resource servers (RSs) to connect with authorization servers (ASs) in an interoperable fashion.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9767.

Copyright Notice

Table of Contents

1. Introduction

The core GNAP specification [GNAP] defines distinct roles for the authorization server (AS) and the resource server (RS). However, the core specification does not define how the RS gets answers to important questions, such as whether a given access token is still valid or what set of access rights the access token is approved for.

While it's possible for the AS and RS to be tightly coupled, such as a single deployed server with a shared storage system, GNAP does not presume or require such a tight coupling. It is increasingly common for the AS and RS to be run and managed separately, particularly in cases where a single AS protects multiple RSs simultaneously.

This specification defines a set of RS-facing APIs that an AS can make available for advanced loosely coupled deployments. Additionally, this document defines a general-purpose model for access tokens, which can be used in structured, formatted access tokens or in token introspection responses. This specification also defines a method for an RS to derive a downstream token for calling another chained RS.

The means for the authorization server to issue the access token to the client instance and the means for the client instance to present the access token to the resource server are subjects of the core GNAP specification [GNAP].

1.1. Terminology

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.

This document contains non-normative examples of partial and complete HTTP messages, JSON structures, URLs, query components, keys, and other elements. Some examples use a single trailing backslash `\` to indicate line wrapping for long values, as per [RFC8792]. The `\` character and leading spaces on wrapped lines are not part of the value.

Terminology specific to GNAP is defined in the terminology section of the core specification; see Section 1.1 of [GNAP]. The following protocol roles are defined: authorization server, client, end user, resource owner, and resource server. The following protocol elements are defined: access token, attribute, grant, privilege, protected resource, right, subject, and subject information. The same definitions are used in this document.

2. Access Tokens

Access tokens are used as a mechanism for an AS to provide a client instance limited access to an RS. These access tokens are artifacts representing a particular set of access rights granted to the client instance to act on behalf of the RO. While the format of access tokens varies in different systems (see discussion in Section 2.2), the concept of an access token is consistent across all GNAP systems.

2.1. General-Purpose Access Token Model

The core GNAP specification [GNAP] focuses on the relationship between the client and the AS. Since the access token is opaque to the client, the core specification does not define a token model. However, the AS will need to create tokens, and the RS will need to understand tokens. To facilitate a level of structural interoperability, a common access token model is presented here. Access tokens represent a common set of aspects across different GNAP deployments. This list is not intended to be universal or comprehensive but rather serves as guidance to implementers in developing data structures and associated systems across a GNAP deployment. These data structures are communicated between the AS and RS by using either a structured token or an API-like mechanism such as token introspection (see Section 3.3).

This general-purpose data model does not assume either approach; in fact, both approaches can be used together to convey different pieces of information. Where possible, mappings to the JSON Web Token (JWT) [JWT] standard format are provided for each item in the model.

2.1.1. Value

All access tokens have a *value*, which is the string that is passed on the wire between parties. In order for different access tokens to be differentiated at runtime, the value of a token needs to be unique within a security domain (such as all systems controlled by an AS). Otherwise, two separate tokens would be confused for each other, which would lead to security issues. The AS chooses the value, which can be structured (see Section 2.2) or unstructured. When the token is structured, the token value also has a *format* known to the AS and RS, and the other items in this token model are contained within the token's value in some fashion. When the token is unstructured, the values are usually retrieved by the RS using a service such as token introspection described in Section 3.3.

The access token value is conveyed in the `value` field of an `access_token` response; see Section 3.2 of [GNAP].

The format and content of the access token value is opaque to the client software. While the client software needs to be able to carry and present the access token value, the client software is never expected nor intended to be able to understand the token value itself.

If structured tokens like those in [JWT] are used, the value of the token might not be stored by the AS. Instead, a token identifier can be used along with protection by an AS-generated signature to validate and identify an individual token.

2.1.2. Issuer

The access token is issued by the AS as defined in [GNAP]. The AS will need to identify itself in order to allow an RS to recognize tokens that the AS has issued, particularly in cases where tokens from multiple different ASs could be presented to the same RS.

This information is not usually conveyed directly to the client instance, since the client instance should know this information based on where it receives the token from.

In the payload of a JSON Web Token [JWT] or a token introspection response, this corresponds to the `iss` claim.

2.1.3. Audience

The access token is intended for use at one or more RSs. The AS can list a token's intended RSs to allow each RS to ensure that the RS is not receiving a token intended for someone else. The AS and RS have to agree on the nature of any audience identifiers represented by the token, but the URIs of the RS are a common pattern.

In the payload of a JSON Web Token [JWT] or token introspection response, this corresponds to the `aud` claim.

In cases where more complex access is required, the `location` field of objects in the `access` array can also convey audience information. In such cases, the client instance might need to know the audience information in order to differentiate between possible RSs to present the token to.

2.1.4. Key Binding

Access tokens in GNAP are bound to the client instance's registered or presented key, except in cases where the access token is a bearer token. For all tokens bound to a key, the AS and RS need to be able to identify which key the token is bound to; otherwise, an attacker could substitute their own key during presentation of the token. In the case of an asymmetric algorithm, the AS and RS need to know only the public key, while the client instance will also need to know the private key in order to present the token. In the case of a symmetric algorithm, all parties will need to either know or be able to derive the shared key.

The source of this key information can vary depending on deployment decisions. For example, an AS could decide that all tokens issued to a client instance are always bound to that client instance's current key. When the key needs to be dereferenced, the AS looks up the client instance to which the token was issued and finds the key information there. Alternatively, the AS could bind each token to a specific key that is managed separately from client instance information. In such a case, the AS determines the key information directly. This approach allows the client instance to use a different key for each request or allows the AS to issue a key for the client instance to use with the particular token.

In all cases, the key binding also includes a proofing mechanism, along with any parameters needed for that mechanism such as a signing or digest algorithm. If such information is not included with the proofing key, an attacker could present a token with a seemingly valid key using an insecure and incorrect proofing mechanism.

This value is conveyed to the client instance in the `key` field of the `access_token` response in Section 3.2 of [GNAP]. Since the common case is that the token is bound to the client instance's registered key, this field can be omitted in this case since the client will be aware of its own key.

In the payload of a JSON Web Token [JWT], this corresponds to the `cnf` (confirmation) claim. In a token introspection response, this corresponds to the `key` claim.

In the case of a bearer token, all parties need to know that a token has no key bound to it and will therefore reject any attempts to use the bearer token with a key in an undefined way.

2.1.5. Flags

GNAP access tokens can have multiple associated data flags that indicate special processing or considerations for a token. For example, the data flags can indicate whether a token is a bearer token or should be expected to be durable across grant updates.

The client can request a set of flags using the `flags` field of the `access_token` grant request parameter in Section 2.1.1 of [GNAP].

These flags are conveyed from the AS to the client in the `flags` field of the `access_token` section of the grant response in Section 3.2 of [GNAP].

For token introspection, flags are returned in the `flags` field of the response.

2.1.6. Access Rights

Access tokens are tied to a limited set of access rights. These rights specify in some detail what the token can be used for, how it can be used, and where it can be used. The internal structure of access rights is detailed in Section 8 of [GNAP].

The access rights associated with an access token are calculated from the rights available to the client instance making the request, the rights available to be approved by the RO, the rights actually approved by the RO, and the rights corresponding to the RS in question. The rights for a specific access token are a subset of the overall rights in a grant request.

These rights are requested by the client instance in the `access` field of the `access_token` request; see Section 2.1 of [GNAP].

The rights associated with an issued access token are conveyed to the client instance in the `access` field of the `access_token` response in Section 3.2 of [GNAP].

In token introspection responses, access rights correspond to the `access` claim.

2.1.7. Time Validity Window

The access token can be limited to a certain time window outside of which it is no longer valid for use at an RS. This window can be explicitly bounded by an expiration time and a not-before time, or it could be calculated based on the issuance time of the token. For example, an RS could decide that it will accept tokens for most calls within an hour of a token's issuance, but only within five minutes of the token's issuance for certain high-value calls.

Since access tokens could be revoked at any time for any reason outside of a client instance's control, the client instance often does not know or concern itself with the validity time window of an access token. However, this information can be made available to it by using the `expires_in` field of an access token response; see Section 3.2 of [GNAP].

The issuance time of the token is conveyed in the `iat` claim in the payload of a JSON Web Token [JWT] or a token introspection response.

The expiration time of a token, after which it is to be rejected, is conveyed in the `exp` claim in the payload of a JSON Web Token [JWT] or a token introspection response.

The starting time of a token's validity window, before which it is to be rejected, is conveyed in the `nbf` claim in the payload of a JSON Web Token [JWT] or a token introspection response.

2.1.8. Token Identifier

Individual access tokens often need a unique internal identifier to allow the AS to differentiate between multiple separate tokens. This value of the token can often be used as the identifier, but in some cases, a separate identifier is used.

This separate identifier can be conveyed in the `jti` claim in the payload of a JSON Web Token [JWT] or a token introspection response.

This identifier is not usually exposed to the client instance using the token, because the client instance only needs to use the token by value.

2.1.9. Authorizing Resource Owner

Access tokens are approved on behalf of a resource owner (RO). The identity of this RO can be used by the RS to determine exactly which resource to access or which kinds of access to allow. For example, an access token used to access identity information can hold a user identifier to allow the RS to determine which profile information to return. The nature of this information is subject to agreement by the AS and RS.

This corresponds to the `sub` claim in the payload of a JSON Web Token [JWT] or a token introspection response.

Detailed RO information is not returned to the client instance when an access token is requested alone, and in many cases, returning this information to the client instance would be a privacy violation on the part of the AS. Since the access token represents a specific delegated access, the client instance needs only to use the token at its target RS. Following the profile example, the client instance does not need to know the account identifier to get specific attributes about the account represented by the token.

GNAP does allow for the return of subject information separately from the access token, in the form of identifiers and assertions. These values are returned directly to the client separately from any access tokens that are requested, though it's common that they represent the same party.

2.1.10. End User

The end user is the party operating the client software. The client instance can facilitate the end user interacting with the AS in order to determine the end user's identity, gather authorization, and provide the results of that information back to the client instance.

In many instances, the end user is the same party as the resource owner. However, in some cases, the two roles can be fulfilled by different people, where the RO is consulted asynchronously. The token model should be able to reflect these kinds of situations by representing the end user and RO separately. For example, an end user can request a financial payment, but the RO is the holder of the account that the payment would be withdrawn from. The RO would be consulted for approval by the AS outside of the flow of the GNAP request. A token in such circumstances would need to show both the RO and end user as separate entities.

2.1.11. Client Instance

Access tokens are issued to a specific client instance by the AS. The identity of this instance can be used by the RS to allow specific kinds of access or other attributes about the access token. For example, an AS that binds all access tokens issued to a particular client instance to that client instance's most recent key rotation would need to be able to look up the client instance in order to find the key binding detail.

This corresponds to the `client_id` claim in the payload of a JSON Web Token [JWT] or the `instance_id` field of a token introspection response.

The client is not normally informed of this information separately, since a client instance can usually correctly assume that it is the client instance to which a token that it receives was issued.

2.1.12. Label

When multiple access tokens are requested or a client instance uses token labels, the parties will need to keep track of which labels were applied to each individual token. Since labels can be reused across different grant requests, the token label alone is not sufficient to uniquely identify a given access token in a system. However, within the context of a grant request, these labels are required to be unique.

A client instance can request a specific label using the `label` field of an `access_token` request; see Section 2.1 of [GNAP].

The AS can inform the client instance of a token's label using the `label` field of an `access_token` response; see Section 3.2 of [GNAP].

This corresponds to the `label` field of a token introspection response.

2.1.13. Parent Grant Request

All access tokens are issued in the context of a specific grant request from a client instance. The grant request itself represents a unique tuple of:

- The AS processing the grant request
- The client instance making the grant request
- The RO (or set of ROs) approving the grant request (or needing to approve it)
- The access rights granted by the RO
- The current state of the grant request, as defined in Section 1.5 of [GNAP]

The AS can use this information to tie common information to a specific token. For instance, instead of specifying a client instance for every issued access token for a grant, the AS can store the client information in the grant itself and look it up by reference from the access token.

The AS can also use this information when a grant request is updated. For example, if the client instance asks for a new access token from an existing grant, the AS can use this link to revoke older non-durable access tokens that had been previously issued under the grant.

A client instance will have its own model of an ongoing grant request, especially if that grant request can be continued using the API defined in Section 5 of [GNAP] where several pieces of statefulness need to be kept in hand. The client instance might need to keep an association with the grant request that issued the token in case the access token expires or does not have sufficient access rights, so that the client instance can get a new access token without having to restart the grant request process from scratch.

Since the grant itself does not need to be identified in any of the protocol messages, GNAP does not define a specific grant identifier to be conveyed between any parties in the protocol. Only the AS needs to keep an explicit connection between an issued access token and the parent grant that issued it.

2.1.14. AS-Specific Access Tokens

When an access token is used for the grant continuation API defined in Section 5 of [GNAP] (the continuation access token), the token management API defined in Section 6 of [GNAP] (the token management access token), or the RS-facing API defined in Section 3 (the resource server management access token), the AS **MUST** separate these access tokens from other access tokens used at one or more RSs. The AS can do this through the use of a flag on the access token data structure, by using a special internal access right, or any other means at its disposal. Just like other access tokens in GNAP, the contents of these AS-specific access tokens are opaque to the software presenting the token. Unlike other access tokens, the contents of these AS-specific access tokens are also opaque to the RS.

The client instance is given continuation access tokens only as part of the `continue` field of the grant response in Section 3.1 of [GNAP]. The client instance is given token management access tokens only as part of the `manage` field of the grant response in Section 3.2.1 of [GNAP]. The means by which the RS is given resource server management access tokens is out of scope of this specification, but methods could include preconfiguration of the token value with the RS software or granting the access token through a standard GNAP process.

For continuation access tokens and token management access tokens, a client instance **MUST** take steps to differentiate these special-purpose access tokens from access tokens used at one or more RSs. To facilitate this, a client instance can store AS-specific access tokens separately from other access tokens in order to keep them from being confused with each other and used at the wrong endpoints.

An RS should never see an AS-specific access token presented, so any attempts to process one **MUST** fail. When introspection is used, the AS **MUST NOT** return an `active` value of `true` for AS-specific access tokens to the RS. If an AS implements its protected endpoints in such a way that it uses token introspection internally, the AS **MUST** differentiate these AS-specific access tokens from those issued for use at an external RS.

2.2. Access Token Formats

When the AS issues an access token for use at an RS, the RS needs to have some means of understanding what the access token is for in order to determine how to respond to the request. The core GNAP protocol makes neither assumptions nor demands on the format or contents of the access token, and in fact, the token format and contents are opaque to the client instance. However, such token formats can be the topic of agreements between the AS and RS.

Self-contained structured token formats allow for the conveyance of information between the AS and RS without requiring the RS to call the AS at runtime as described in Section 3.3. Structured tokens can also be used in combination with introspection, allowing the token itself to carry one class of information and the introspection response to carry another.

Some token formats, such as Macaroons [MACAROON] and Biscuits [BISCUIT], allow for the RS to derive sub-tokens without having to call the AS as described in Section 4.

The supported token formats can be communicated dynamically at runtime between the AS and RS in several places:

- The AS can declare its supported token formats as part of RS-facing discovery (Section 3.1).
- The RS can require a specific token format be used to access a registered resource set (Section 3.4).
- The AS can return the token's format in an introspection response (Section 3.3).

In all places where the token format is listed explicitly, it **MUST** be one of the registered values in the "GNAP Token Formats" registry Section 5.3.

3. Resource-Server-Facing API

To facilitate runtime and dynamic connections with an RS, the AS can offer an RS-facing API consisting of one or more of the following optional pieces:

- Discovery
- Introspection
- Token chaining
- Resource reference registration

3.1. RS-Facing AS Discovery

A GNAP AS offering RS-facing services can publish its features on a well-known discovery document at the URL with the same schema and authority as the grant request endpoint URL, at the path `/.well-known/gnap-as-rs`.

The discovery response is a JSON document [RFC8259] consisting of a single JSON object with the following fields:

grant_request_endpoint (string): The location of the AS's grant request endpoint defined by Section 9 of [GNAP]. This URL **MUST** be the same URL used by client instances in support of GNAP requests. The RS can use this to derive downstream access tokens, if supported by the AS. The location **MUST** be a URL [RFC3986] with a scheme component that **MUST** be https, a host component, and (optionally) port, path, and query components and no fragment components. **REQUIRED**. See Section 4.

introspection_endpoint (string): The URL of the endpoint offering introspection. The location **MUST** be a URL [RFC3986] with a scheme component that **MUST** be https, a host component, and (optionally) port, path, and query components and no fragment components. **REQUIRED** if the AS supports introspection. An absent value indicates that the AS does not support introspection. See Section 3.3.

token_formats_supported (array of strings): A list of token formats supported by this AS. The values in this list **MUST** be registered in the "GNAP Token Formats" registry per Section 5.3. **OPTIONAL**.

resource_registration_endpoint (string): The URL of the endpoint offering resource registration. The location **MUST** be a URL [RFC3986] with a scheme component that **MUST** be https, a host component, and (optionally) port, path, and query components and no fragment components. **REQUIRED** if the AS supports dynamic resource registration. An absent value indicates that the AS does not support this feature. See Section 3.4.

key_proofs_supported (array of strings): A list of the AS's supported key proofing mechanisms. The values of this list correspond to possible values of the `proof` field of the key section of the request. Values **MUST** be registered in the "GNAP Key Proofing Methods" registry established by [GNAP]. **OPTIONAL**.

Additional fields are defined in the "GNAP RS-Facing Discovery Document Fields" registry; see Section 5.8.

3.2. Protecting RS Requests to the AS

Unless otherwise specified, the RS **MUST** protect its calls to the AS using any of the signature methods defined in Section 7 of [GNAP].

The RS **MAY** present its keys by reference or by value in a similar fashion to a client instance calling the AS in the core protocol of GNAP, as described in Section 7.1 of [GNAP]. In the protocols defined here, this takes the form of the resource server identifying itself by using a `key` field or by passing an instance identifier directly.

```
POST /continue HTTP/1.1
Host: server.example.com
Authorization: GNAP 80UPRY5NM33OMUKMKSKU
Signature-Input: sig1=...
Signature: sig1=...
Content-Type: application/json

"resource_server": {
    "key": {
        "proof": "httpsig",
        "jwk": {
            "kty": "EC",
            "crv": "secp256k1",
            "kid": "2021-07-06T20:22:03Z",
            "x": "-J9OJIZj4nmopZbQN7T8xv3sbeS5-f_vBNSy_EHnBZc",
            "y": "sjrS51pLtu3P4LUTVvyAIxRfDV_be2RYpI5_f-Yjivw"
        }
    }
}
```

or by reference:

```
POST /continue HTTP/1.1
Host: server.example.com
Signature-Input: sig1=...
Signature: sig1=...
Content-Type: application/json

{
    "resource_server": "7C7C4AZ9KHRS6X63AJAO"
}
```

The means by which an RS's keys are made known to the AS are out of the scope of this specification. The AS **MAY** require an RS to preregister its keys, or it could allow calls from arbitrary keys in a trust-on-first-use model.

The AS **MAY** issue access tokens, called "resource server management access tokens", to the RS to protect the RS-facing API endpoints. If such tokens are issued, the RS **MUST** present them to the RS-facing API endpoints along with the RS authentication.

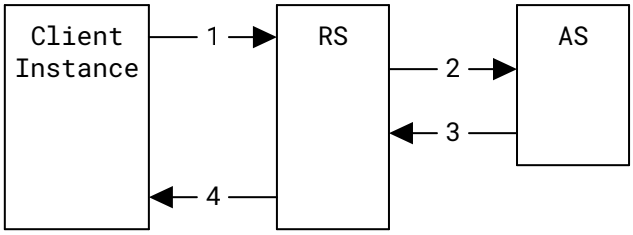
```
POST /continue HTTP/1.1
Host: server.example.com
Authorization: GNAP 80UPRY5NM33OMUKMKSKU
Signature-Input: sig1=...
Signature: sig1=...
Content-Type: application/json

{
    "resource_server": "7C7C4AZ9KHRS6X63AJAO"
}
```

3.3. Token Introspection

The AS issues access tokens representing a set of delegated access rights to be used at one or more RSs. The AS can offer an introspection service to allow an RS to validate that a given access token:

- has been issued by the AS
- is valid at the current time
- has not been revoked
- is appropriate for the RS identified in the call

When the RS receives an access token, it can call the introspection endpoint at the AS to get token information.



1. The client instance calls the RS with its access token.
2. The RS introspects the access token value at the AS. The RS signs the request with its own key (not the client instance's key or the token's key).
3. The AS validates the access token value and the RS's request and returns the introspection response for the token.
4. The RS fulfills the request from the client instance.

The RS signs the request with its own key and sends the value of the access token in the body of the request as a JSON object with the following members:

access_token (string): The access token value presented to the RS by the client instance. **REQUIRED**.

proof (string): The proofing method used by the client instance to bind the token to the RS request. The value **MUST** be registered in the "GNAP Key Proofing Methods" registry. **RECOMMENDED**.

resource_server (object/string): The identification used to authenticate the resource server making this call, either by value or by reference as described in Section 3.2. **REQUIRED**.

access (array of strings/objects): The minimum access rights required to fulfill the request. This **MUST** be in the format described in Section 8 of [GNAP]. **OPTIONAL**.

Additional fields are defined in the "GNAP Token Introspection Request" registry (Section 5.4).

```
POST /introspect HTTP/1.1
Host: server.example.com
Content-Type: application/json
Signature-Input: sig1=...
Signature: sig1=...
Digest: sha256=...

{
    "access_token": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
    "proof": "httpsig",
    "resource_server": "7C7C4AZ9KHRS6X63AJAO"
}
```

The AS **MUST** validate the access token value and determine if the token is active. The parameters of the request provide a context for the AS to evaluate the access token, and the AS **MUST** take all provided parameters into account when evaluating if the token is active. If the AS is unable to process part of the request, such as not understanding part of the `access` field presented, the AS **MUST NOT** indicate the token as active.

An active access token is defined as a token that is all of the following:

- was issued by the processing AS,
- has not been revoked,
- has not expired,
- is bound using the `proof` method indicated,
- is appropriate for presentation at the identified RS, and
- is appropriate for the `access` indicated (if present).

The AS responds with a data structure describing the token's current state and any information the RS would need to validate the token's presentation, such as its intended proofing mechanism and key material.

active (boolean): If `true`, the access token presented is active, as defined above. If any of the criteria for an active token are not true, or if the AS is unable to make a determination (such as the token is not found), the value is set to `false` and other fields are omitted. **REQUIRED**.

If the access token is active, additional fields from the single access token response structure defined in Section 3.2.1 of [GNAP] are included. In particular, these include the following:

access (array of strings/objects): The access rights associated with this access token. This **MUST** be in the format described in Section 8 of [GNAP]. This array **MAY** be filtered or otherwise limited for consumption by the identified RS, including being an empty array, which indicates that the token has no explicit access rights that can be disclosed to the RS. **REQUIRED**.

key (object/string): if the token is bound. The key bound to the access token, to allow the RS to validate the signature of the request from the client instance. If the access token is a bearer token, this **MUST NOT** be included. **REQUIRED**

flags (array of strings): The set of flags associated with the access token. **OPTIONAL**.

exp (integer): The timestamp after which this token is no longer valid. Expressed as integer seconds from UNIX Epoch. **OPTIONAL**.

iat (integer): The timestamp at which this token was issued by the AS. Expressed as integer seconds from UNIX Epoch. **OPTIONAL**.

nbf (integer): The timestamp before which this token is not valid. Expressed as integer seconds from UNIX Epoch. **OPTIONAL**.

aud (string or array of strings): Identifiers for the resource servers this token can be accepted at. **OPTIONAL**.

sub (string): Identifier of the resource owner who authorized this token. **OPTIONAL**.

iss (string): Grant endpoint URL of the AS that issued this token. **REQUIRED**.

instance_id (string): The instance identifier of the client instance that the token was issued to. **OPTIONAL**.

Additional fields are defined in the "GNAP Token Introspection Response" registry (Section 5.5).

The response **MAY** include any additional fields defined in an access token response and **MUST NOT** include the access token `value` itself.

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

{
    "active": true,
    "access": [
        "dolphin-metadata", "some other thing"
    ],
    "key": {
        "proof": "httpsig",
        "jwk": {
                "kty": "RSA",
                "e": "AQAB",
                "kid": "xyz-1",
                "alg": "RS256",
                "n": "kOB5rR4Jv0GMeL...."
        }
    }
}
```

When processing the results of the introspection response, the RS **MUST** determine the appropriate course of action. For instance, if the RS determines that the access token's access rights are not sufficient for the request to which the token was attached, the RS can return an error or a public resource, as appropriate for the RS. In all cases, the final determination of the response is at the discretion of the RS.

3.4. Registering a Resource Set

If the RS needs to, it can post a set of resources, as described in Section 8 ("Resource Access Rights") of [GNAP], to the AS's resource registration endpoint along with information about what the RS will need to validate the request.

access (array of objects/strings): The list of access rights associated with the request in the format described in Section 8 ("Resource Access Rights") of [GNAP]. **REQUIRED**.

resource_server (object/string): The identification used to authenticate the resource server making this call, either by value or by reference as described in Section 3.2. **REQUIRED**.

token_formats_supported (array of strings): The list of token formats that the RS is able to process. The values in this array **MUST** be registered in the "GNAP Token Formats" registry per Section 5.3. If the field is omitted, the token format is at the discretion of the AS. If the AS does not support any of the requested token formats, the AS **MUST** return an error to the RS. **OPTIONAL**.

token_introspection_required (boolean): If present and set to `true`, the RS expects to make a token introspection request as described in Section 3.3. If absent or set to `false`, the RS does not anticipate needing to make an introspection request for tokens relating to this resource set. If the AS does not support token introspection for this RS, the AS **MUST** return an error to the RS. **OPTIONAL**.

Additional fields are defined in the "GNAP Resource Set Registration Request Parameters" registry (Section 5.6).

The RS **MUST** identify itself with its own key and sign the request.

```
POST /resource HTTP/1.1
Host: server.example.com
Content-Type: application/json
Signature-Input: sig1=...
Signature: sig1=...
Digest: ...

{
    "access": [
        {
            "actions": [
                "read",
                "write",
                "dolphin"
            ],
            "locations": [
                "https://server.example.net/",
                "https://resource.local/other"
            ],
            "datatypes": [
                "metadata",
                "images"
            ]
        },
        "dolphin-metadata"
    ],
    "resource_server": "7C7C4AZ9KHRS6X63AJAO"

}
```

The AS responds with a reference appropriate to represent the resources list that the RS presented in its request as well as any additional information the RS might need in future requests.

resource_reference (string): A single string representing the list of resources registered in the request. The RS **MAY** make this handle available to a client instance as part of a discovery response as described in Section 9.1 of [GNAP] or as documentation to client software developers. **REQUIRED**.

instance_id (string): An instance identifier that the RS can use to refer to itself in future calls to the AS, in lieu of sending its key by value. See Section 3.2. **OPTIONAL**.

introspection_endpoint (string): The introspection endpoint of this AS that is used to allow the RS to perform token introspection. See Section 3.3. **OPTIONAL**.

Additional fields are defined in the "GNAP Resource Set Registration Response Parameters" registry (Section 5.7).

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

{
    "resource_reference": "FWWIKYBQ6U56NL1"
}
```

If a resource was previously registered, the AS **MAY** return the same resource reference value as in previous responses.

If the registration fails, the AS returns HTTP status code 400 (Bad Request) to the RS, indicating that the registration was not successful.

The client instance can then use the `resource_reference` value as a string-type access reference as defined in Section 8.1 of [GNAP]. This value **MAY** be combined with any other additional access rights requested by the client instance.

```
{
    "access_token": {
        "access": [
            "FWWIKYBQ6U56NL1",
            {
                "type": "photo-api",
                "actions": [
                    "read",
                    "write",
                    "dolphin"
                ],
                "locations": [
                    "https://server.example.net/",
                    "https://resource.local/other"
                ],
                "datatypes": [
                    "metadata",
                    "images"
                ]
            },
            "dolphin-metadata"
        ]
    },
    "client": "client-12351.bdxqf"
}
```

3.5. Error Responses

In the case of an error from the RS-facing API, the AS responds to the RS with HTTP status code 400 (Bad Request) and a JSON object consisting of a single `error` field, which is either an object or a string.

When returned as a string, the error value is the error code:

```
{
    error: "invalid_access"
}
```

When returned as an object, the error object contains the following fields:

`code` (string): A single ASCII error code defining the error. **REQUIRED**.

`description` (string): A human-readable string description of the error intended for the developer of the client. **OPTIONAL**.

```
{
  "error": {
    "code": "invalid_access",
    "description": "Access to 'foo' is not permitted for this RS."
  }
}
```

This specification defines the following error code values:

`"invalid_request"`: The request is missing a required parameter, includes an invalid parameter value, or is otherwise malformed.

`"invalid_resource_server"`: The request was made from an RS that was not recognized or allowed by the AS, or the RS's signature validation failed.

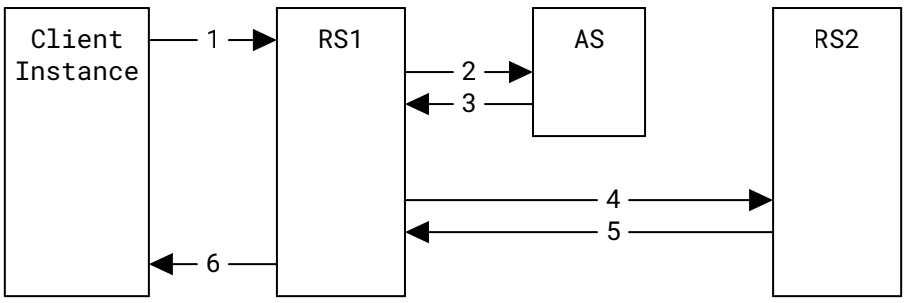
`"invalid_access"` The RS is not permitted to register or introspect for the requested "access" value.

Additional error codes can be defined in the "GNAP RS-Facing Error Codes" registry (Section 5.9).

4. Deriving a Downstream Token

Some architectures require an RS to act as a client instance and use a derived access token for a secondary RS. Since the RS is not the same entity that made the initial grant request, the RS is not capable of referencing or modifying the existing grant. As such, the RS needs to request or generate a new access token for its use at the secondary RS. This internal secondary token is issued in the context of the incoming access token.

While it is possible to use a token format (Section 2) that allows for the RS to generate its own secondary token, the AS can allow the RS to request this secondary access token using the same process used by the original client instance to request the primary access token. Since the RS is acting as its own client instance from the perspective of GNAP, this process uses the same grant endpoint, request structure, and response structure as a client instance's request.



1. The client instance calls RS1 with an access token.

2. RS1 presents that token to the AS to get a derived token for use at RS2. RS1 indicates that it has no ability to interact with the RO. Note that RS1 signs its request with its own key, not the token's key or the client instance's key.
3. The AS returns a derived token to RS1 for use at RS2.
4. RS1 calls RS2 with the token from (3).
5. RS2 fulfills the call from RS1.
6. RS1 fulfills the call from the original client instance.

If the RS needs to derive a token from one presented to it, it can request one from the AS by making a token request as described in [GNAP] and presenting the existing access token's value in the "existing_access_token" field.

Since the RS is acting as a client instance, the RS **MUST** identify itself with its own key in the `client` field and sign the request just as any client instance would, as described in Section 3.2. The AS **MUST** determine that the token being presented is appropriate for use at the RS making the token chaining request.

```
POST /tx HTTP/1.1
Host: server.example.com
Content-Type: application/json
Detached-JWS: ejy0...

{
  "access_token": {
      "access": [
          {
              "actions": [
                  "read",
                  "write",
                  "dolphin"
              ],
              "locations": [
                  "https://server.example.net/",
                  "https://resource.local/other"
              ],
              "datatypes": [
                  "metadata",
                  "images"
              ]
          },
          "dolphin-metadata"
      ]
  },
  "client": "7C7C4AZ9KHRS6X63AJAO",
  "existing_access_token": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0"
}
```

The AS responds with a token for the downstream RS2 as described in [GNAP]. The downstream RS2 could repeat this process as necessary for calling further RSs.

5. IANA Considerations

IANA has added values to existing registries and created five registries under the "Grant Negotiation and Authorization Protocol (GNAP)" registry group.

5.1. Well-Known URIs

The "gnap-as-rs" URI suffix is registered in the "Well-Known URIs" registry to support RS-facing discovery of the AS.

URI Suffix: gnap-as-rs
Change Controller: IETF
Specification Document: Section 3.1 of RFC 9767
Status: Permanent

5.2. GNAP Grant Request Parameters

The following parameter is registered in the "GNAP Grant Request Parameters" registry:

Name: `existing_access_token`
Type: string
Reference: Section 4 of RFC 9767

5.3. GNAP Token Formats

This document defines a GNAP token format, for which IANA has created and maintains a new registry titled "GNAP Token Formats". Initial values for this registry are given in Section 5.3.2. Future assignments and modifications to existing assignment are to be made through the Specification Required registration policy [RFC8126].

The designated expert (DE) is expected to ensure that:

- all registrations follow the template presented in Section 5.3.1.
- the format's definition is sufficiently unique from other formats provided by existing parameters.
- the format's definition specifies the format of the access token in sufficient detail to allow for the AS and RS to be able to communicate the token information.

5.3.1. Registry Template

Name: The name of the format.
Status: Whether or not the format is in active use. Possible values are Active and Deprecated.
Description: The human-readable description of the access token format.
Reference: The specification that defines the token format.

5.3.2. Initial Registry Contents

Name	Status	Description	Reference
`jwt-signed`	Active	JSON Web Token, signed with JWS	[JWT]
`jwt-encrypted`	Active	JSON Web Token, encrypted with JWE	[JWT]
`macaroon`	Active	Macaroon	[MACAROON]
`biscuit`	Active	Biscuit	[BISCUIT]
`zcap`	Active	ZCAP	[ZCAPLD]

Table 1: Initial Contents of the GNAP Token Formats Registry

5.4. GNAP Token Introspection Request

This document defines GNAP token introspection, for which IANA has created and maintains a new registry titled "GNAP Token Introspection Request". Initial values for this registry are given in Section 5.4.2. Future assignments and modifications to existing assignment are to be made through the Specification Required registration policy [RFC8126].

The DE is expected to ensure that:

- all registrations follow the template presented in Section 5.4.1.
- the claim's definition is sufficiently orthogonal to other claims defined in the registry so as avoid overlapping functionality.
- the claim's definition specifies the syntax and semantics of the claim in sufficient detail to allow for the AS and RS to be able to communicate the token values.

5.4.1. Registry Template

Name: The name of the claim.
Type: The JSON data type of the claim value.
Reference: The specification that defines the claim.

5.4.2. Initial Registry Contents

The table below contains the initial contents of the "GNAP Token Introspection Request" registry.

Name	Type	Reference
access_token	string	Section 3.3 of RFC 9767
proof	string	Section 3.3 of RFC 9767
resource_server	object/string	Section 3.3 of RFC 9767

Name	Type	Reference
access	array of strings/objects	Section 3.3 of RFC 9767

Table 2: Initial Contents of the GNAP Token Introspection Request Registry

5.5. GNAP Token Introspection Response

This document defines GNAP token introspection, for which IANA has created and maintains a new registry titled "GNAP Token Introspection Response". Initial values for this registry are given in Section 5.5.2. Future assignments and modifications to existing assignment are to be made through the Specification Required registration policy [RFC8126].

The DE is expected to ensure that:

- all registrations follow the template presented in Section 5.5.1.
- the claim's definition is sufficiently orthogonal to other claims defined in the registry so as avoid overlapping functionality.
- the claim's definition specifies the syntax and semantics of the claim in sufficient detail to allow for the AS and RS to be able to communicate the token values.

5.5.1. Registry Template

Name: The name of the claim.
Type: The JSON data type of the claim value.
Reference: The specification that defines the claim.

5.5.2. Initial Registry Contents

The table below contains the initial contents of the "GNAP Token Introspection Response" registry.

Name	Type	Reference
active	boolean	Section 3.3 of RFC 9767
access	array of strings/objects	Section 3.3 of RFC 9767
key	object/string	Section 3.3 of RFC 9767
flags	array of strings	Section 3.3 of RFC 9767
exp	integer	Section 3.3 of RFC 9767
iat	integer	Section 3.3 of RFC 9767
nbf	integer	Section 3.3 of RFC 9767

Name	Type	Reference
aud	string or array of strings	Section 3.3 of RFC 9767
sub	string	Section 3.3 of RFC 9767
iss	string	Section 3.3 of RFC 9767
instance_id	string	Section 3.3 of RFC 9767

Table 3: Initial Contents of the GNAP Token Introspection Response Registry

5.6. GNAP Resource Set Registration Request Parameters

This document defines a means to register a resource set for a GNAP AS, for which IANA has created and maintains a new registry titled "GNAP Resource Set Registration Request Parameters". Initial values for this registry are given in Section 5.6.2. Future assignments and modifications to existing assignment are to be made through the Expert Review registration policy [RFC8126].

The DE is expected to ensure that:

- all registrations follow the template presented in Section 5.6.1.
- the parameter's definition is sufficiently orthogonal to other parameters defined in the registry so as avoid overlapping functionality.
- the parameter's definition specifies the syntax and semantics of the parameter in sufficient detail to allow for the AS and RS to be able to communicate the resource set.

5.6.1. Registry Template

Name: The name of the parameter.
Type: The JSON data type of the parameter value.
Reference: The specification that defines the token.

5.6.2. Initial Registry Contents

The table below contains the initial contents of the "GNAP Resource Set Registration Request Parameters" registry.

Name	Type	Reference
access	array of strings/objects	Section 3.4 of RFC 9767
resource_server	object/string	Section 3.4 of RFC 9767
token_formats_supported	array of strings	Section 3.4 of RFC 9767

Name	Type	Reference
token_introspection_required	boolean	Section 3.4 of RFC 9767

Table 4: Initial Contents of the GNAP Resource Set Registration Request Parameters Registry

5.7. GNAP Resource Set Registration Response Parameters

This document defines a means to register a resource set for a GNAP AS, for which IANA has created and maintains a new registry titled "GNAP Resource Set Registration Response Parameters". Initial values for this registry are given in Section 5.7.2. Future assignments and modifications to existing assignment are to be made through the Expert Review registration policy [RFC8126].

The DE is expected to ensure that:

- all registrations follow the template presented in Section 5.7.1.
- the parameter's definition is sufficiently orthogonal to other claims defined in the registry so as avoid overlapping functionality.
- the parameter's definition specifies the syntax and semantics of the claim in sufficient detail to allow for the AS and RS to be able to communicate the resource set.

5.7.1. Registry Template

Name: The name of the parameter.
Type: The JSON data type of the parameter value.
Reference: The specification that defines the parameter.

5.7.2. Initial Registry Contents

The table below contains the initial contents of the "GNAP Resource Set Registration Response Parameters" registry.

Name	Type	Reference
resource_reference	string	Section 3.4 of RFC 9767
instance_id	string	Section 3.4 of RFC 9767
introspection_endpoint	string	Section 3.4 of RFC 9767

Table 5: Initial Contents of the GNAP Resource Set Registration Response Parameters Registry

5.8. GNAP RS-Facing Discovery Document Fields

This document defines a means to for a GNAP AS to be discovered by a GNAP RS, for which IANA has created and maintains a new registry titled "GNAP RS-Facing Discovery Document Fields". Initial values for this registry are given in Section 5.8.2. Future assignments and modifications to existing assignment are to be made through the Expert Review registration policy [RFC8126].

The DE is expected to ensure that:

- all registrations follow the template presented in Section 5.8.1.
- the field's definition is sufficiently orthogonal to other fields defined in the registry so as avoid overlapping functionality.
- the field's definition specifies the syntax and semantics of the fields in sufficient detail to allow for the RS to be able to communicate with the AS.

5.8.1. Registry Template

Name: The name of the field.
Type: The JSON data type of the field value.
Reference: The specification that defines the field.

5.8.2. Initial Registry Contents

The table below contains the initial contents of the "GNAP RS-Facing Discovery Document Fields" registry.

Name	Type	Reference
introspection_endpoint	string	Section 3.1 of RFC 9767
token_formats_supported	array of strings	Section 3.1 of RFC 9767
resource_registration_endpoint	string	Section 3.1 of RFC 9767
grant_request_endpoint	string	Section 3.1 of RFC 9767
key_proofs_supported	array of strings	Section 3.1 of RFC 9767

Table 6: Initial Contents of the GNAP RS-Facing Discovery Document Fields Registry

5.9. GNAP RS-Facing Error Codes

This document defines a set of errors that the AS can return to the RS, for which IANA has created and maintains a new registry titled "GNAP RS-Facing Error Codes". Initial values for this registry are given in Section 5.9.2. Future assignments and modifications to existing assignments are to be made through the Specification Required registration policy [RFC8126].

The DE is expected to ensure that:

- all registrations follow the template presented in Section 5.9.1.
- the error response is sufficiently unique from other errors to provide actionable information to the client instance.
- the definition of the error response specifies all conditions in which the error response is returned and what the client instance's expected action is.

5.9.1. Registration Template

Error: A unique string code for the error.
Reference: Reference to the document(s) that specifies the value, preferably including a URI that can be used to retrieve a copy of the document(s). An indication of the relevant sections may also be included but is not required.

5.9.2. Initial Contents

Error	Reference
invalid_request	Section 3.5 of RFC 9767
invalid_resource_server	Section 3.5 of RFC 9767
invalid_access	Section 3.5 of RFC 9767

Table 7: Initial Contents of the GNAP RS-Facing Error Codes Registry

6. Security Considerations

In addition to the normative requirements in this document and in [GNAP], implementers are strongly encouraged to consider the following additional security considerations in implementations and deployments of GNAP.

6.1. TLS Protection in Transit

All requests in GNAP made over untrusted network connections have to be made over TLS as outlined in [BCP195] to protect the contents of the request and response from manipulation and interception by an attacker. This includes all requests from a client instance to the RS and all requests from the RS to an AS.

6.2. Token Validation

The RS has a responsibility to validate the incoming access token in a manner consistent with its deployment. For self-contained stateless tokens such as those described in Section 2.2, this consists of actions such as validating the token's signature and ensuring the relevant fields and

results are appropriate for the request being made. For reference-style tokens or tokens that are otherwise opaque to the RS, the token introspection RS-facing API can be used to provide updated information about the state of the token, as described in Section 3.3.

The RS needs to validate that a token:

- is intended for this RS (audience restriction)
- is presented using the appropriate key for the token (see also Section 6.4)
- identifies an appropriate subject to access the resource (usually this is the resource owner who authorized the token's issuance)
- is issued by a trusted AS for this resource

Even though key proofing mechanisms have to cover the value of the token, validating the key proofing alone is not sufficient to protect a request to an RS. If an RS validates only the presentation method as described in Section 6.4 without validating the token itself, an attacker could use a compromised key or a confused deputy to make arbitrary calls to the RS beyond what has been authorized by the RO.

6.3. Caching Token Validation Result

Since token validation can be an expensive process, requiring either cryptographic operations or network calls to an introspection service as described in Section 3.3, an RS could cache the results of token validation for a particular token. The trade-off for using a cached validation for a token presents an important decision space for implementers: relying on a cached validation result increases performance and lowers processing overhead, but it comes at the expense of the liveness and accuracy of the information in the cache. While a cached value is in use at the RS, an attacker could present a revoked token and have it accepted by the RS.

As with any cache, the consistency of this cache can be managed in a variety of ways. One of the most simple methods is managing the lifetime of the cache in order to balance the performance and security properties. If the cache is too long, an attacker has a larger window in which to use a revoked token. If the window is too short, the benefits of using the cache are diminished. It is also possible that an AS could send a proactive signal to the RS to invalidate revoked access tokens, though such a mechanism is outside the scope of this specification.

6.4. Key Proof Validation

For key-bound access tokens, the proofing method needs to be validated alongside the value of the token itself, as described in Section 6.2. The process of validation is defined by the key proofing method, as described in Section 7.3 of [GNAP].

If the proofing method is not validated, an attacker could use a compromised token without access to the token's bound key.

The RS also needs to ensure that the proofing method is appropriate for the key associated with the token, including any choice of algorithm or identifiers.

The proofing should be validated independently on each request to the RS, particularly as aspects of the call could vary. As such, the RS should never cache the results of a proof validation from one message and apply it to a subsequent message.

6.5. Token Exfiltration

Since the RS sees the token value, it is possible for a compromised RS to leak that value to an attacker. As such, the RS needs to protect token values as sensitive information and protect them from exfiltration.

This is especially problematic with bearer tokens and tokens bound to a shared key, since an RS has access to all information necessary to create a new, valid request using the token in question.

6.6. Token Reuse by an RS

If the access token is a bearer token, or the RS has access to the key material needed to present the token, the RS could be tricked into reusing an access token presented to it by a client. While it is possible to build a system that makes use of this artifact as a feature, it is safer to exchange the incoming access token for another contextual token for use by the RS, as described in Section 4. This access token can be bound to the RS's own keys and limited to access needed by the RS, instead of the full set of rights associated with the token issued to the client instance.

6.7. Token Format Considerations

With formatted tokens, the format of the token is likely to have its own considerations, and the RS needs to follow any such considerations during the token validation process. The application and scope of these considerations is specific to the format and outside the scope of this specification.

6.8. Oversharing Token Contents

The contents of the access token model divulge information about the access token's context and rights to the RS. This is true whether the contents are parsed from the token itself or sent in an introspection response.

It's likely that every RS does not need to know all details of the token model, especially in systems where a single access token is usable across multiple RSs. An attacker could use this to gain information about the larger system by compromising only one RS. By limiting the information available to only that which is relevant to a specific RS, such as using a limited introspection reply as defined in Section 3.3, a system can follow the principle of least disclosure to each RS.

6.9. Resource References

Resource references, as returned by the protocol in Section 3.4, are intended to be opaque to both the RS and the client. However, since they are under the control of the AS, the AS can put whatever content it wants into the reference value. This value could unintentionally disclose system structure or other internal details if it was processed by an unintended party.

Furthermore, such patterns could lead to the client software and RS depending on certain structures being present in the reference value, which diminishes the separation of concerns of the different roles in a GNAP system.

To mitigate this, the AS should only use fully random or encrypted values for resource references.

6.10. Token Reissuance from an Untrusted AS

It is possible for an attacker's client instance to issue its own tokens to another client instance, acting as an AS that the second client instance has chosen to trust. If the token is a bearer token or the reissuance is bound using an AS-provided key, the target client instance will not be able to tell that the token was originally issued by the valid AS. This process allows an attacker to insert their own session and rights into an unsuspecting client instance in the guise of a valid token for the attacker that appears to have been issued to the target client instance on behalf of its own RO.

This attack is predicated on a misconfiguration with the targeted client, as it has been configured to get tokens from the attacker's AS and use those tokens with the target RS, which has no association with the attacker's AS. However, since the token is ultimately coming from the trusted AS and is being presented with a valid key, the RS has no way of telling that the token was passed through an intermediary.

To mitigate this, the RS can publish its association with the trusted AS through either discovery or documentation. Therefore, a client properly following this association would only go directly to the trusted RS for access tokens for the RS.

Furthermore, limiting the use of bearer tokens and AS-provided keys to only highly trusted ASs in certain circumstances prevents the attacker from being able to willingly exfiltrate their token to an unsuspecting client instance.

6.11. Introspection of Token Keys

The introspection response defined in Section 3.3 provides a means for the AS to tell the RS what key material is needed to validate the key proof of the request. Capture of the introspection response can expose these security keys to an attacker. In the case of asymmetric cryptography, only the public key is exposed, and the token cannot be reused by the attacker based on this result alone. This could potentially divulge information about the client instance that was unknown otherwise.

If an access token is bound to a symmetric key, the RS will need access to the full key value in order to validate the key proof of the request, as described in Section 6.4. However, divulging the key material to the RS also gives the RS the ability to create a new request with the token. In this circumstance, the RS is under similar risk of token exfiltration and reuse as a bearer token, as described in Section 6.6. Consequently, symmetric keys should only be used in systems where the RS can be fully trusted to not create a new request with tokens presented to it.

6.12. RS Registration and Management

Most functions of the RS-facing API in Section 3 are protected by requiring the RS to present proof of a signing key along with the request, in order to identify the RS making the call, potentially coupled with an AS-specific access token. This practice allows the AS to differentially respond to API calls to different RSs, such as answering introspection calls with only the access rights relevant to a given RS instead of all access rights an access token could be good for.

While the means by which an RS and its keys become known to the AS is out of scope for this specification, it is anticipated that common practice will be to statically register an RS, allowing it to protect specific resources or certain classes of resources. Fundamentally, the RS can only offer the resources that it serves. However, a rogue AS could attempt to register a set of resources that mimics a different RS in order to solicit an access token that is usable at the target RS. If the access token is a bearer token or is bound to a symmetric key that is known to the RS, then the attacker's RS gains the ability and knowledge needed to use that token elsewhere.

In some ecosystems, dynamic registration of an RS and its associated resources is feasible. In such systems, the identity of the RS could be conveyed by a URI passed in the `location` field of an access rights request, thereby allowing the AS to limit the view the RS has into the larger system.

7. Privacy Considerations

7.1. Token Contents

The contents of the access token could potentially contain personal information about the end user, RO, or other parties. This is true whether the contents are parsed from the token itself or sent in an introspection response.

While an RS will sometimes need this information for processing, it's often the case that an RS is exposed to these details only in passing, and not intentionally. For example, consider a client that has been issued an access token that is usable for both medical and non-medical APIs. If this access token contains a medical record number to facilitate the RS serving the medical API, then any RS for a non-medical API would also learn the user's medical record number in the process, even though that API has no need to make such a correlation.

To mitigate this, a formatted token could contain separate sections targeted to different RSs to segregate data. Alternatively, token introspection can be used to limit the data returned to each RS, as defined in Section 3.3.

7.2. Token Use Disclosure through Introspection

When introspection is used by an RS, the AS is made aware of a particular token being used at a particular RS. When the RS is a separate system, the AS would not otherwise have insight into this action. This can potentially lead to the AS learning about patterns and actions of particular end users by watching which RSs are accessed and when.

7.3. Mapping a User to an AS

When the client instance receives information about the protecting AS from an RS, it can be used to derive information about the resources being protected without releasing the resources themselves. For example, if a medical record is protected by a personal AS, an untrusted client could call an RS to discover the location of the AS protecting the record. Since the AS is tied strongly to a single RO, the untrusted and unauthorized client software can gain information about the resource being protected without accessing the record itself.

8. References

8.1. Normative References

[BCP195] Best Current Practice 195, <https://www.rfc-editor.org/info/bcp195>.
At the time of writing, this BCP comprises the following:

Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, DOI 10.17487/RFC8996, March 2021, <https://www.rfc-editor.org/info/rfc8996>.

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, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/info/rfc9325>.

[GNAP] Richer, J., Ed. and F. Imbault, "Grant Negotiation and Authorization Protocol (GNAP)", RFC 9635, DOI 10.17487/RFC9635, October 2024, <https://www.rfc-editor.org/info/rfc9635>.

[JWT] 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>.

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[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/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

8.2. Informative References

[BISCUIT] Biscuit, "Biscuit Authorization", <https://www.biscuitsec.org/>.

[MACAROON] Birgisson, A., Politz, J. G., Erlingsson, U., Taly, A., Vrable, M., and M. Lentczner, "Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud", NDSS Symposium 2014, DOI 10.14722/ndss.2014.23212, February 2014, <https://www.ndss-symposium.org/ndss2014/ ndss-2014-programme/ macaroons-cookies-contextual-caveats- decentralized-authorization-cloud/>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[ZCAPLD] Lemmer-Webber, C., Ed. and M. Sporny, Ed., "Authorization Capabilities for Linked Data v0.3", W3C Draft Community Group Report, January 2023, <https://w3c-ccg.github.io/zcap-spec/>.

Acknowledgements

The editors would like to thank the following individuals for their reviews, feedback, implementations, and contributions: Aaron Parecki, Adrian Gropper, Andrii Deinega, Annabelle Backman, Dmitry Barinov, Fabien Imbault, Florian Helmschmidt, George Fletcher, Justin Richer, Kathleen Moriarty, Leif Johansson, Mike Varley, Nat Sakimura, Takahiko Kawasaki, and Yaron Sheffer.

Additionally, the editors want to acknowledge the immense contributions of Aaron Parecki to the content of this document. We thank him for his insight, input, and hard work, without which GNAP would not have grown to what it is.

Authors' Addresses

Justin Richer (EDITOR)
Bespoke Engineering
Email: ietf@justin.richer.org
URI: https://bspk.io/

Fabien Imbault
acert.io
Email: fabien.imbault@acert.io
URI: https://acert.io/