Workgroup: GNAP
Internet-Draft:
draft-ietf-gnap-resource-servers-00
Published: 28 April 2021
Intended Status: Standards Track
Expires: 30 October 2021
Authors: J. Richer, Ed. A. Parecki F. Imbault
Bespoke Engineering Okta acert.io

Grant Negotiation and Authorization Protocol Resource Server Connections

Abstract

GNAP defines a mechanism for delegating authorization to a piece of software, and conveying that delegation to the software. This extension defines methods for resource servers (RS) to communicate with authorization servers (AS) in an interoperable fashion.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The core GNAP protocol does not define any one specific mechanism for the resource server (RS) to communicate with the authorization server (AS), allowing the connection between these components to be solved orthogonally to the core protocol's concerns. For example, the RS and AS roles could be fulfilled by the same piece of software with common storage, obviating the need for any connecting protocol. However, it is often desirable to have the RS and AS communicate at runtime for a variety of purposes, including allowing the RS to validate and understand the rights and privileges associated with a grant of access represented by an access token issued by (AS), or negotiating the capabilities of either party. These types of connections are particularly useful for connecting an AS and RS from different vendors, allowing interoperable distributed deployments of GNAP-protected systems.

This specification defines several means for a RS and AS to communicate these aspects with each other, including structured access tokens and RS-facing APIs. This specification also discusses methods for an RS to derive a downstream token for calling another chained RS as well as a client-facing discovery mechanism that can be used to bootstrap the GNAP process when the client instance does not know which AS protects a given RS.

The means of the authorization server issuing the access token to the client instance and the means of the client instance presenting the access token to the resource server are the subject of the GNAP core protocol specification [I-D.ietf-gnap-core-protocol].

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.

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 no assumptions or demands on the format or contents of the access token, but 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.

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

3. Resource-Server-Facing API

To facilitate runtime and dynamic connections, 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 using the URL .well-known/gnap-as-rs.

This endpoint contains a JSON document [RFC8259] consisting of a single JSON object with any combination of the following optional fields:

introspection_endpoint: The URL of the endpoint offering introspection. Section 3.3

token_formats_supported: A list of token formats supported by this AS.

resource_registration_endpoint: The URL of the endpoint offering resource registration. Section 3.4

grant_endpoint: The grant endpoint of the GNAP AS.

3.2. Protecting RS requests to the AS

Unless otherwise specified, the RS protects its calls to the AS using any of the signature methods defined by GNAP. This signing method MUST cover all of the appropriate portions of the HTTP request message, including any body elements, tokens, or headers required for functionality.

The AS MAY require an RS to pre-register its keys or could alternatively allow calls from arbitrary keys, in a trust-on-first-use model. The RS MAY present its keys by reference or by value in the same fashion as a client instance calling the AS in the core protocol of GNAP [I-D.ietf-gnap-core-protocol].

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

* has not expired

* 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. [[See issue #115]]

```
+--------+       +------+       +------+
| Client |--(1)->|  RS  |       |  AS  |
|Instance|       |      |--(2)->|      |
|        |       |      |       |      |
|        |       |      |<-(3)--|      |
|        |       |      |       +------+
|        |<-(4)--|      |
+--------+       +------+
```

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 client instance'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 access token as the body of the request.

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

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

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. The response MAY include any fields defined in an access token response.

```
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...."
        }
    }
}
```


3.4. Registering a Resource Handle

If the RS needs to, it can post a set of resources as described in the Resource Access Rights section of [I-D.ietf-gnap-core-protocol] to the AS's resource registration endpoint.

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
Detached-JWS: ejy0...

{
    "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 handle appropriate to represent the resources list that the RS presented.

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

{
    "resource_handle": "FWWIKYBQ6U56NL1"
}
```

The RS MAY make this handle available as part of a response (Section 5) or as documentation to developers.

[[See issue #117]]

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 token 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]{#structure} 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.

```
+--------+       +-------+       +------+       +-------+
| Client |--(1)->|  RS1  |       |  AS  |       |  RS2  |
|Instance|       |       |--(2)->|      |       |       |
|        |       |       |<-(3)--|      |       |       |
|        |       |       |       +------+       |       |
|        |       |       |                      |       |
|        |       |       |-----------(4)------->|       |
|        |       |       |<----------(5)--------|       |
|        |<-(6)--|       |                      |       |
+--------+       +-------+                      +-------+
```

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 [I-D.ietf-gnap-core-protocol] 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.

[[See issue #116]]

```
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 [I-D.ietf-gnap-core-protocol]. The downstream RS2 could repeat this process as necessary for calling further RS's.

5. Requesting Resources With Insufficient Access

If the client instance calls an RS without an access token, or with an invalid access token, the RS MAY respond to the client instance with an authentication header indicating that GNAP needs to be used to access the resource. The address of the GNAP endpoint MUST be sent in the "as_uri" parameter. The RS MAY additionally return a resource reference that the client instance MAY use in its access token request. This resource reference handle SHOULD be sufficient for at least the action the client instance was attempting to take at the RS. The RS MAY use the dynamic resource handle request (Section 3.4) to register a new resource handle, or use a handle that has been pre-configured to represent what the RS is protecting. The content of this handle is opaque to the RS and the client instance in both cases.

```
WWW-Authenticate: \
  GNAP as_uri=https://server.example/tx,access=FWWIKYBQ6U56NL1
```

The client instance then makes a call to the "as_uri" as described in [I-D.ietf-gnap-core-protocol], with the value of "access" as one of the members of the access array in the access_token portion of the request. The client instance MAY request additional resources and other information, and MAY request multiple access tokens.

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

{
    "access_token": {
        "access": [
            "FWWIKYBQ6U56NL1",
            "dolphin-metadata"
        ]
    },
    "client": "KHRS6X63AJ7C7C4AZ9AO"
}
```

[[See issue #118]]

6. Acknowledgements

(TODO: the ACK section should probably be split between the documents)

7. IANA Considerations

[[TBD: There are a lot of items in the document that are expandable through the use of value registries.]]

8. Security Considerations

[[TBD: There are a lot of security considerations to add.]]

All requests have to be over TLS or equivalent as per [BCP195]. Many handles act as shared secrets, though they can be combined with a requirement to provide proof of a key as well.

9. Privacy Considerations

[[TBD: There are a lot of privacy considerations to add.]]

When introspection is used, the AS is made aware of a particular token being used at a particular AS, and the AS would not otherwise have insight into this.

When the client instance receives information about the protecting AS from an RS, this can be used to derive information about the resources being protected without releasing the resources themselves.

10. Normative References

[BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", May 2015, <https://www.rfc-editor.org/info/bcp195>.

[I-D.ietf-gnap-core-protocol] Richer, J., Parecki, A., and F. Imbault, "Grant Negotiation and Authorization Protocol", Work in Progress, Internet-Draft, draft-ietf-gnap-core-protocol-04, 22 February 2021, <https://www.ietf.org/archive/id/draft-ietf-gnap-core-protocol-04.txt>.

[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>.

[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>.

Appendix A. Document History

* -00

  - Extracted resource server section.

Authors' Addresses

Justin Richer (editor)
Bespoke Engineering

Email: ietf@justin.richer.org
URI: https://bspk.io/

Aaron Parecki
Okta

Email: aaron@parecki.com
URI: https://aaronparecki.com

Fabien Imbault
acert.io

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