Stream:	Internet Engineering Task Force (IETF)		
RFC:	9728		
Category:	Standards Track		
Published:	April 2025		
ISSN:	2070-1721		
Authors:	M.B. Jones	P. Hunt	A. Parecki
	Self-Issued Consulting	*Independent Identity, Inc.*	*Okta*

RFC 9728
OAuth 2.0 Protected Resource Metadata

Abstract

This specification defines a metadata format that an OAuth 2.0 client or authorization server can use to obtain the information needed to interact with an OAuth 2.0 protected resource.

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/rfc9728.

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

This specification defines a metadata format enabling OAuth 2.0 clients and authorization servers to obtain information needed to interact with an OAuth 2.0 protected resource. The structure and content of this specification are intentionally as parallel as possible to (1) "OAuth 2.0 Dynamic Client Registration Protocol" [RFC7591], which enables a client to provide metadata about itself to an OAuth 2.0 authorization server and (2) "OAuth 2.0 Authorization Server Metadata" [RFC8414], which enables a client to obtain metadata about an OAuth 2.0 authorization server.

The means by which the client obtains the location of the protected resource is out of scope for this document. In some cases, the location may be manually configured into the client; for example, an email client could provide an interface for a user to enter the URL of their JSON Meta Application Protocol (JMAP) server [RFC8620]. In other cases, it may be dynamically discovered; for example, a user could enter their email address into an email client, the client could perform WebFinger discovery [RFC7033] (in a manner related to the description in Section 2 of [OpenID.Discovery]) to find the resource server, and the client could then fetch the resource server metadata to find the authorization server to use to obtain authorization to access the user's email.

The metadata for a protected resource is retrieved from a well-known location as a JSON [RFC8259] document, which declares information about its capabilities and, optionally, its relationships with other services. This process is described in Section 3.

This metadata can be communicated either in a self-asserted fashion or as a set of signed metadata values represented as claims in a JSON Web Token (JWT) [JWT]. In the JWT case, the issuer is vouching for the validity of the data about the protected resource. This is analogous to the role that the software statement plays in OAuth Dynamic Client Registration [RFC7591].

Each protected resource publishing metadata about itself makes its own metadata document available at a well-known location deterministically derived from the protected resource's URL, even when the resource server implements multiple protected resources. This prevents attackers from publishing metadata that supposedly describes the protected resource but that is not actually authoritative for the protected resource, as described in Section 7.3.

Section 2 defines metadata parameters that a protected resource can publish, which includes things like which scopes are supported, how a client can present an access token, and more. These values, such as the `jwks_uri` (see Section 2), may be used with other specifications; for example, the public keys published in the `jwks_uri` can be used to verify the signed resource responses, as described in [FAPI.MessageSigning].

Section 5 describes the use of `WWW-Authenticate` by protected resources to dynamically inform clients of the URL of their protected resource metadata. This use of `WWW-Authenticate` can indicate that the protected resource metadata may have changed.

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

All applications of JSON Web Signature (JWS) data structures [JWS] and JSON Web Encryption (JWE) data structures [JWE] as discussed in this specification utilize the JWS Compact Serialization or the JWE Compact Serialization; the JWS JSON Serialization and the JWE JSON Serialization are not used. Choosing a single serialization is intended to facilitate interoperability.

1.2. Terminology

This specification uses the terms "access token", "authorization code", "authorization server", "client", "client authentication", "client identifier", "protected resource", and "resource server" defined by OAuth 2.0 [RFC6749], and the terms "Claim Name" and "JSON Web Token (JWT)" defined by "JSON Web Token (JWT)" [JWT].

This specification defines the following term:

Resource Identifier:
: The protected resource's resource identifier, which is a URL that uses the `https` scheme and has no fragment component. As specified in Section 2 of [RFC8707], it also **SHOULD NOT** include a query component, but it is recognized that there are cases that make a query

component a useful and necessary part of a resource identifier. Protected resource metadata is published at a `.well-known` location [RFC8615] derived from this resource identifier, as described in Section 3.

2. Protected Resource Metadata

Protected resources can have metadata describing their configuration. The following protected resource metadata parameters are used by this specification and are registered in the "OAuth Protected Resource Metadata" registry established in Section 8.1:

resource
: **REQUIRED**. The protected resource's resource identifier, as defined in Section 1.2.

authorization_servers
: **OPTIONAL**. JSON array containing a list of OAuth authorization server issuer identifiers, as defined in [RFC8414], for authorization servers that can be used with this protected resource. Protected resources **MAY** choose not to advertise some supported authorization servers even when this parameter is used. In some use cases, the set of authorization servers will not be enumerable, in which case this metadata parameter would not be used.

jwks_uri
: **OPTIONAL**. URL of the protected resource's JSON Web Key (JWK) Set [JWK] document. This contains public keys belonging to the protected resource, such as signing key(s) that the resource server uses to sign resource responses. This URL **MUST** use the `https` scheme. When both signing and encryption keys are made available, a `use` (public key use) parameter value is **REQUIRED** for all keys in the referenced JWK Set to indicate each key's intended usage.

scopes_supported
: **RECOMMENDED**. JSON array containing a list of scope values, as defined in OAuth 2.0 [RFC6749], that are used in authorization requests to request access to this protected resource. Protected resources **MAY** choose not to advertise some scope values supported even when this parameter is used.

bearer_methods_supported
: **OPTIONAL**. JSON array containing a list of the supported methods of sending an OAuth 2.0 bearer token [RFC6750] to the protected resource. Defined values are `["header", "body", "query"]`, corresponding to Sections 2.1, 2.2, and 2.3 of [RFC6750]. The empty array `[]` can be used to indicate that no bearer methods are supported. If this entry is omitted, no default bearer methods supported are implied, nor does its absence indicate that they are not supported.

resource_signing_alg_values_supported
: **OPTIONAL**. JSON array containing a list of the JWS [JWS] signing algorithms (`alg` values) [JWA] supported by the protected resource for signing resource responses, for instance, as described in [FAPI.MessageSigning]. No default algorithms are implied if this entry is omitted. The value `none` **MUST NOT** be used.

resource_name
: Human-readable name of the protected resource intended for display to the end user. It is **RECOMMENDED** that protected resource metadata include this field. The value of this field **MAY** be internationalized, as described in Section 2.1.

resource_documentation
: **OPTIONAL**. URL of a page containing human-readable information that developers might want or need to know when using the protected resource. The value of this field **MAY** be internationalized, as described in Section 2.1.

resource_policy_uri
: **OPTIONAL**. URL of a page containing human-readable information about the protected resource's requirements on how the client can use the data provided by the protected resource. The value of this field **MAY** be internationalized, as described in Section 2.1.

resource_tos_uri
: **OPTIONAL**. URL of a page containing human-readable information about the protected resource's terms of service. The value of this field **MAY** be internationalized, as described in Section 2.1.

tls_client_certificate_bound_access_tokens
: **OPTIONAL**. Boolean value indicating protected resource support for mutual-TLS client certificate-bound access tokens [RFC8705]. If omitted, the default value is false.

authorization_details_types_supported
: **OPTIONAL**. JSON array containing a list of the authorization details `type` values supported by the resource server when the `authorization_details` request parameter [RFC9396] is used.

dpop_signing_alg_values_supported
: **OPTIONAL**. JSON array containing a list of the JWS `alg` values (from the "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE]) supported by the resource server for validating Demonstrating Proof of Possession (DPoP) proof JWTs [RFC9449].

dpop_bound_access_tokens_required
: **OPTIONAL**. Boolean value specifying whether the protected resource always requires the use of DPoP-bound access tokens [RFC9449]. If omitted, the default value is false.

Additional protected resource metadata parameters **MAY** also be used.

2.1. Human-Readable Resource Metadata

Human-readable resource metadata values and resource metadata values that reference human-readable content **MAY** be represented in multiple languages and scripts. For example, the values of fields such as `resource_name`, `resource_documentation`, `resource_tos_uri`, and `resource_policy_uri` might have multiple locale-specific metadata values to facilitate use in different locations.

To specify the languages and scripts, language tags [BCP47] are added to resource metadata parameter names, delimited by a # character. Since member names as discussed in JSON [RFC8259] are case sensitive, it is **RECOMMENDED** that language tag values used in Claim Names be spelled using the character case with which they are registered in the "Language Subtag Registry" [IANA.Language]. In particular, normally, language names are spelled with lowercase characters, region names are spelled with uppercase characters, and languages are spelled with mixed-case characters. However, since language tag values are case insensitive per [BCP47], implementations **SHOULD** interpret the language tag values supplied in a case-insensitive manner. Per the recommendations in [BCP47], language tag values used in metadata parameter names should only be as specific as is necessary. For instance, using `fr` might be sufficient in many contexts, rather than `fr-CA` or `fr-FR`.

For example, a resource could represent its name in English as `"resource_name#en": "My Resource"` and its name in Italian as `"resource_name#it": "La mia bella risorsa"` within its metadata. Any or all of these names **MAY** be displayed to the end user, choosing which names to display based on system configuration, user preferences, or other factors.

If any human-readable field is sent without a language tag, parties using it **MUST NOT** make any assumptions about the language, character set, or script of the string value, and the string value **MUST** be used as is wherever it is presented in a user interface. To facilitate interoperability, it is **RECOMMENDED** that each kind of human-readable metadata provided include an instance of its metadata parameter without any language tags in addition to any language-specific parameters, and it is **RECOMMENDED** that any human-readable fields sent without language tags contain values suitable for display on a wide variety of systems.

2.2. Signed Protected Resource Metadata

In addition to JSON elements, metadata values **MAY** also be provided as a `signed_metadata` value, which is a JSON Web Token (JWT) [JWT] that asserts metadata values about the protected resource as a bundle. A set of metadata parameters that can be used in signed metadata as claims are defined in Section 2. The signed metadata **MUST** be digitally signed or MACed (protected with a Message Authentication Code) using a JSON Web Signature (JWS) [JWS] and **MUST** contain an `iss` (issuer) claim denoting the party attesting to the claims in the signed metadata. Consumers of the metadata **MAY** ignore the signed metadata if they do not support this feature. If the consumer of the metadata supports signed metadata, metadata values conveyed in the signed metadata **MUST** take precedence over the corresponding values conveyed using plain JSON elements.

Signed metadata is included in the protected resource metadata JSON object using this **OPTIONAL** metadata parameter:

signed_metadata
: A JWT containing metadata parameters about the protected resource as claims. This is a string value consisting of the entire signed JWT. A `signed_metadata` parameter **SHOULD NOT** appear as a claim in the JWT; it is **RECOMMENDED** to reject any metadata in which this occurs.

3. Obtaining Protected Resource Metadata

Protected resources supporting metadata **MUST** make a JSON document containing metadata as specified in Section 2 available at a URL formed by inserting a well-known URI string into the protected resource's resource identifier between the host component and the path and/or query components, if any. By default, the well-known URI string used is `/.well-known/oauth-protected-resource`. The syntax and semantics of `.well-known` are defined in [RFC8615]. The well-known URI path suffix used **MUST** be registered in the "Well-Known URIs" registry [IANA.well-known]. Examples of this construction can be found in Section 3.1.

The term "application", as used below (and as used in [RFC8414]), encompasses all the components used to accomplish the task for the use case. That can include OAuth clients, authorization servers, protected resources, and non-OAuth components, inclusive of the code running in each of them. Applications are built to solve particular problems and may utilize many components and services.

Different applications utilizing OAuth protected resources in application-specific ways **MAY** define and register different well-known URI path suffixes for publishing protected resource metadata used by those applications. For instance, if the Example application uses an OAuth protected resource in an Example-specific way and there are Example-specific metadata values that it needs to publish, then it might register and use the `example-protected-resource` URI path suffix and publish the metadata document at the URL formed by inserting `/.well-known/example-protected-resource` between the host and path and/or query components of the protected resource's resource identifier. Alternatively, many such applications will use the default well-known URI string `/.well-known/oauth-protected-resource`, which is the right choice for general-purpose OAuth protected resources, and not register an application-specific one.

An OAuth 2.0 application using this specification **MUST** specify what well-known URI suffix it will use for this purpose. The same protected resource **MAY** choose to publish its metadata at multiple well-known locations derived from its resource identifier -- for example, publishing metadata at both `/.well-known/example-protected-resource` and `/.well-known/oauth-protected-resource`.

3.1. Protected Resource Metadata Request

A protected resource metadata document **MUST** be queried using an HTTP `GET` request at the previously specified URL.

The consumer of the metadata would make the following request when the resource identifier is `https://resource.example.com` and the well-known URI path suffix is `oauth-protected-resource` to obtain the metadata, since the resource identifier contains no path component:

```
GET /.well-known/oauth-protected-resource HTTP/1.1
Host: resource.example.com
```

If the resource identifier value contains a path or query component, any terminating slash (`/`) following the host component **MUST** be removed before inserting `/.well-known/` and the well-known URI path suffix between the host component and the path and/or query components. The consumer of the metadata would make the following request when the resource identifier is `https://resource.example.com/resource1` and the well-known URI path suffix is `oauth-protected-resource` to obtain the metadata, since the resource identifier contains a path component:

```
GET /.well-known/oauth-protected-resource/resource1 HTTP/1.1
Host: resource.example.com
```

Using path components enables supporting multiple resources per host. This is required in some multi-tenant hosting configurations. This use of `.well-known` is for supporting multiple resources per host; unlike its use in [RFC8615], it does not provide general information about the host.

3.2. Protected Resource Metadata Response

The response is a set of metadata parameters about the protected resource's configuration. A successful response **MUST** use the 200 OK HTTP status code and return a JSON object using the `application/json` content type that contains a set of metadata parameters as its members that are a subset of the metadata parameters defined in Section 2. Additional metadata parameters **MAY** be defined and used; any metadata parameters that are not understood **MUST** be ignored.

Parameters with multiple values are represented as JSON arrays. Parameters with zero values **MUST** be omitted from the response.

An error response uses the applicable HTTP status code value.

The following is a non-normative example response:

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

{
 "resource":
   "https://resource.example.com",
 "authorization_servers":
   ["https://as1.example.com",
    "https://as2.example.net"],
 "bearer_methods_supported":
   ["header", "body"],
 "scopes_supported":
   ["profile", "email", "phone"],
 "resource_documentation":
   "https://resource.example.com/resource_documentation.html"
}
```

3.3. Protected Resource Metadata Validation

The `resource` value returned **MUST** be identical to the protected resource's resource identifier value into which the well-known URI path suffix was inserted to create the URL used to retrieve the metadata. If these values are not identical, the data contained in the response **MUST NOT** be used.

If the protected resource metadata was retrieved from a URL returned by the protected resource via the `WWW-Authenticate` `resource_metadata` parameter, then the `resource` value returned **MUST** be identical to the URL that the client used to make the request to the resource server. If these values are not identical, the data contained in the response **MUST NOT** be used.

These validation actions can thwart impersonation attacks, as described in Section 7.3.

The recipient **MUST** validate that any signed metadata was signed by a key belonging to the issuer and that the signature is valid. If the signature does not validate or the issuer is not trusted, the recipient **SHOULD** treat this as an error condition.

4. Authorization Server Metadata

To support use cases in which the set of legitimate protected resources to use with the authorization server is enumerable, this specification defines the authorization server metadata parameter `protected_resources`, which enables the authorization server to explicitly list the protected resources. Note that if the set of legitimate authorization servers to use with a protected resource is also enumerable, lists in the authorization server metadata and protected resource metadata should be cross-checked against one another for consistency when these lists are used by the application profile.

The following authorization server metadata parameter is defined by this specification and is registered in the "OAuth Authorization Server Metadata" registry established in "OAuth 2.0 Authorization Server Metadata" [RFC8414].

protected_resources
: **OPTIONAL**. JSON array containing a list of resource identifiers for OAuth protected resources that can be used with this authorization server. Authorization servers **MAY** choose not to advertise some supported protected resources even when this parameter is used. In some use cases, the set of protected resources will not be enumerable, in which case this metadata parameter will not be present.

5. Use of WWW-Authenticate for Protected Resource Metadata

A protected resource **MAY** use the `WWW-Authenticate` HTTP response header field, as discussed in [RFC9110], to return a URL to its protected resource metadata to the client. The client can then retrieve protected resource metadata as described in Section 3. The client might then, for instance, determine what authorization server to use for the resource based on protected resource metadata retrieved.

A typical end-to-end flow doing so is as follows. Note that while this example uses the OAuth 2.0 authorization code flow, a similar sequence could also be implemented with any other OAuth flow.

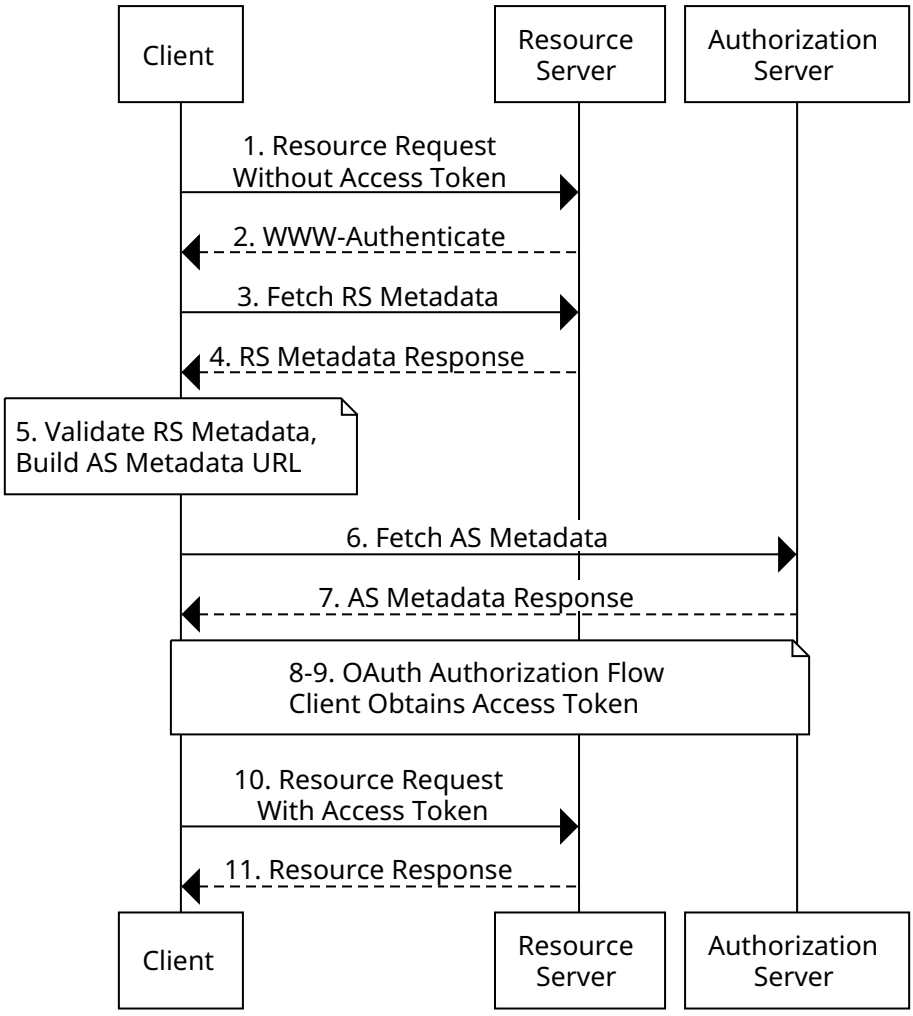

Figure 1: Sequence Diagram

1. The client makes a request to a protected resource without presenting an access token.
2. The resource server responds with a `WWW-Authenticate` header including the URL of the protected resource metadata.
3. The client fetches the protected resource metadata from this URL.
4. The resource server responds with the protected resource metadata according to Section 3.2.
5. The client validates the protected resource metadata, as described in Section 3.3, and builds the authorization server metadata URL from an issuer identifier in the resource metadata according to [RFC8414].
6. The client makes a request to fetch the authorization server metadata.
7. The authorization server responds with the authorization server metadata document according to [RFC8414].

8. The client directs the user agent to the authorization server to begin the authorization flow.
9. The authorization exchange is completed and the authorization server returns an access token to the client.
10. The client repeats the resource request from step 1, presenting the newly obtained access token.
11. The resource server returns the requested protected resource.

5.1. WWW-Authenticate Response

This specification introduces a new parameter in the `WWW-Authenticate` HTTP response header field to indicate the protected resource metadata URL:

resource_metadata:
: The URL of the protected resource metadata.

The response below is an example of a `WWW-Authenticate` header that includes the resource identifier.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer resource_metadata=
  "https://resource.example.com/.well-known/oauth-protected-resource"
```

The HTTP status code in the example response above is defined by [RFC6750].

This parameter **MAY** also be used in `WWW-Authenticate` responses using `authorization` schemes other than `"Bearer"` [RFC6750], such as the `DPoP` scheme defined by [RFC9449].

The `resource_metadata` parameter **MAY** be combined with other parameters defined in other extensions, such as the `max_age` parameter defined by [RFC9470].

5.2. Changes to Resource Metadata

At any point, for any reason determined by the resource server, the protected resource **MAY** respond with a new `WWW-Authenticate` challenge that includes a value for the protected resource metadata URL to indicate that its metadata may have changed. If the client receives such a `WWW-Authenticate` response, it **SHOULD** retrieve the updated protected resource metadata and use the new metadata values obtained, after validating them as described in Section 3.3. Among other things, this enables a resource server to change which authorization servers it uses without any other coordination with clients.

5.3. Client Identifier and Client Authentication

The way in which the client identifier is established at the authorization server is out of scope for this specification.

This specification is intended to be deployed in scenarios where the client has no prior knowledge about the resource server and where the resource server might or might not have prior knowledge about the client.

There are some existing methods by which an unrecognized client can make use of an authorization server, such as using Dynamic Client Registration [RFC7591] to register the client prior to initiating the authorization flow. Future OAuth extensions might define alternatives, such as using URLs to identify clients.

5.4. Compatibility with Other Authentication Methods

Resource servers **MAY** return other `WWW-Authenticate` headers indicating various authentication schemes. This allows the resource server to support clients that may or may not implement this specification and allows clients to choose their preferred authentication scheme.

6. String Operations

Processing some OAuth 2.0 messages requires comparing values in the messages to known values. For example, the member names in the metadata response might be compared to specific member names such as `resource`. Comparing Unicode strings [UNICODE], however, has significant security implications.

Therefore, comparisons between JSON strings and other Unicode strings **MUST** be performed as specified below:

1. Remove any JSON-applied escaping to produce an array of Unicode code points.
2. Unicode Normalization [USA15] **MUST NOT** be applied at any point to either the JSON string or the string it is to be compared against.
3. Comparisons between the two strings **MUST** be performed as a Unicode code-point-to-code-point equality comparison.

Note that this is the same equality comparison procedure as that described in Section 8.3 of [RFC8259].

7. Security Considerations

7.1. TLS Requirements

Implementations **MUST** support TLS. They **MUST** follow the guidance in [BCP195], which provides recommendations and requirements for improving the security of deployed services that use TLS.

The use of TLS at the protected resource metadata URLs protects against information disclosure and tampering.

7.2. Scopes

The `scopes_supported` parameter is the list of scopes the resource server is willing to disclose that it supports. It is not meant to indicate that an OAuth client should request all scopes in the list. The client **SHOULD** still follow OAuth best practices and request tokens with as limited a scope as possible for the given operation, as described in Section 2.3 of "Best Current Practice for OAuth 2.0 Security" [RFC9700].

7.3. Impersonation Attacks

TLS certificate checking **MUST** be performed by the client as described in [RFC9525] when making a protected resource metadata request. Checking that the server certificate is valid for the resource identifier URL prevents adversary-in-the-middle and DNS-based attacks. These attacks could cause a client to be tricked into using an attacker's resource server, which would enable impersonation of the legitimate protected resource. If an attacker can accomplish this, they can access the resources that the affected client has access to, using the protected resource that they are impersonating.

An attacker may also attempt to impersonate a protected resource by publishing a metadata document that contains a `resource` metadata parameter using the resource identifier URL of the protected resource being impersonated but that contains information of the attacker's choosing. This would enable it to impersonate that protected resource, if accepted by the client. To prevent this, the client **MUST** ensure that the resource identifier URL it is using as the prefix for the metadata request exactly matches the value of the `resource` metadata parameter in the protected resource metadata document received by the client, as described in Section 3.3.

7.4. Audience-Restricted Access Tokens

If a client expects to interact with multiple resource servers, the client **SHOULD** request audience-restricted access tokens using [RFC8707], and the authorization server **SHOULD** support audience-restricted access tokens.

Without audience-restricted access tokens, a malicious resource server (RS1) may be able to use the `WWW-Authenticate` header to get a client to request an access token with a scope used by a legitimate resource server (RS2), and after the client sends a request to RS1, then RS1 could reuse the access token at RS2.

While this attack is not explicitly enabled by this specification and is possible in a plain OAuth 2.0 deployment, it is made somewhat more likely by the use of dynamically configured clients. As such, the use of audience-restricted access tokens and Resource Indicators [RFC8707] is **RECOMMENDED** when using the features in this specification.

7.5. Publishing Metadata in a Standard Format

Publishing information about the protected resource in a standard format makes it easier for both legitimate clients and attackers to use the protected resource. Whether a protected resource publishes its metadata in an ad hoc manner or in the standard format defined by this specification, the same defenses against attacks that might be mounted that use this information should be applied.

7.6. Authorization Servers

To support use cases in which the set of legitimate authorization servers to use with the protected resource is enumerable, this specification defines the `authorization_servers` metadata parameter, which enables explicitly listing them. Note that if the set of legitimate protected resources to use with an authorization server is also enumerable, lists in the protected resource metadata and authorization server metadata should be cross-checked against one another for consistency when these lists are used by the application profile.

Secure determination of appropriate authorization servers to use with a protected resource for all use cases is out of scope for this specification. This specification assumes that the client has a means of determining appropriate authorization servers to use with a protected resource and that the client is using the correct metadata for each protected resource. Implementers need to be aware that if an inappropriate authorization server is used by the client, an attacker may be able to act as an adversary-in-the-middle proxy to a valid authorization server without it being detected by the authorization server or the client.

The ways to determine the appropriate authorization servers to use with a protected resource are, in general, application dependent. For instance, some protected resources are used with a fixed authorization server or a set of authorization servers, the locations of which may be known via out-of-band mechanisms. Alternatively, as described in this specification, the locations of the authorization servers could be published by the protected resource as metadata values. In other cases, the set of authorization servers that can be used with a protected resource can be dynamically changed by administrative actions or by changes to the set of authorization servers adhering to a trust framework. Many other means of determining appropriate associations between protected resources and authorization servers are also possible.

7.7. Server-Side Request Forgery (SSRF)

The OAuth client is expected to fetch the authorization server metadata based on the value of the issuer in the resource server metadata. Since this specification enables clients to interoperate with RSs and ASes it has no prior knowledge of, this opens a risk for Server-Side Request Forgery (SSRF) attacks by malicious users or malicious resource servers. Clients **SHOULD** take appropriate precautions against SSRF attacks, such as blocking requests to internal IP address ranges. Further recommendations can be found in the Open Worldwide Application Security Project (OWASP) SSRF Prevention Cheat Sheet [OWASP.SSRF].

7.8. Phishing

This specification may be deployed in a scenario where the desired HTTP resource is identified by a user-selected URL. If this resource is malicious or compromised, it could mislead the user into revealing their account credentials or authorizing unwanted access to OAuth-controlled capabilities. This risk is reduced, but not eliminated, by following best practices for OAuth user interfaces, such as providing clear notice to the user, displaying the authorization server's domain name, supporting origin-bound phishing-resistant authenticators, supporting the use of password managers, and applying heuristic checks such as domain reputation.

7.9. Differences Between Unsigned and Signed Metadata

Unsigned metadata is integrity protected by the use of TLS at the site where it is hosted. This means that its security is dependent upon the Internet Public Key Infrastructure using X.509 (PKIX), as described in [RFC9525]. Signed metadata is additionally integrity protected by the JWS signature applied by the issuer, which is not dependent upon the Internet PKI.

When using unsigned metadata, the party issuing the metadata is the protected resource itself, which is represented by the `resource` value in the metadata, whereas when using signed metadata, the party issuing the metadata is represented by the `iss` (issuer) claim in the signed metadata. When using signed metadata, applications can make trust decisions based on the issuer that performed the signing -- information that is not available when using unsigned metadata. How these trust decisions are made is out of scope for this specification.

7.10. Metadata Caching

Protected resource metadata is retrieved using an HTTP `GET` request, as specified in Section 3.1. Normal HTTP caching behaviors apply, meaning that the `GET` request may retrieve a cached copy of the content, rather than the latest copy. Implementations should utilize HTTP caching directives such as `Cache-Control` with `max-age`, as defined in [RFC9111], to enable caching of retrieved metadata for appropriate time periods.

8. IANA Considerations

Values are registered via Specification Required [RFC8126]. Registration requests should be sent to <oauth-ext-review@ietf.org> to initiate a two-week review period. However, to allow for the allocation of values prior to publication of the final version of a specification, the designated experts may approve registration once they are satisfied that the specification will be completed and published. However, if the specification is not completed and published in a timely manner, as determined by the designated experts, the designated experts may request that IANA withdraw the registration.

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register OAuth Protected Resource Metadata: example").

Within the review period, the designated experts will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful. If the designated experts are not responsive, the registration requesters should contact IANA to escalate the process.

Designated experts should apply the following criteria when reviewing proposed registrations: They must be unique -- that is, they should not duplicate existing functionality; they are likely generally applicable, as opposed to being used for a single application; and they are clear and fit the purpose of the registry.

IANA must only accept registry updates from the designated experts and should direct all requests for registration to the review mailing list.

In order to enable broadly informed review of registration decisions, there should be multiple designated experts to represent the perspectives of different applications using this specification. In cases where registration may be perceived as a conflict of interest for a particular expert, that expert should defer to the judgment of the other experts.

The mailing list is used to enable public review of registration requests, which enables both designated experts and other interested parties to provide feedback on proposed registrations. Designated experts may allocate values prior to publication of the final specification. This allows authors to receive guidance from the designated experts early, so any identified issues can be fixed before the final specification is published.

8.1. OAuth Protected Resource Metadata Registry

This specification establishes the "OAuth Protected Resource Metadata" registry for OAuth 2.0 protected resource metadata names. The registry records the protected resource metadata parameter and a reference to the specification that defines it.

8.1.1. Registration Template

Metadata Name:
: The name requested (e.g., "resource"). This name is case sensitive. Names may not match other registered names in a case-insensitive manner unless the designated experts state that there is a compelling reason to allow an exception.

Metadata Description:
: Brief description of the metadata (e.g., "Resource identifier URL").

Change Controller:
: For IETF Stream RFCs, list "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
: Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

8.1.2. Initial Registry Contents

Metadata Name: `resource`
Metadata Description: Protected resource's resource identifier URL
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `authorization_servers`
Metadata Description: JSON array containing a list of OAuth authorization server issuer identifiers
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `jwks_uri`
Metadata Description: URL of the protected resource's JWK Set document
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `scopes_supported`
Metadata Description: JSON array containing a list of the OAuth 2.0 scope values that are used in authorization requests to request access to this protected resource
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `bearer_methods_supported`
Metadata Description: JSON array containing a list of the OAuth 2.0 bearer token presentation methods that this protected resource supports
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `resource_signing_alg_values_supported`
Metadata Description: JSON array containing a list of the JWS signing algorithms (`alg` values) supported by the protected resource for signed content
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `resource_name`
Metadata Description: Human-readable name of the protected resource

Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `resource_documentation`
Metadata Description: URL of a page containing human-readable information that developers might want or need to know when using the protected resource
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `resource_policy_uri`
Metadata Description: URL of a page containing human-readable information about the protected resource's requirements on how the client can use the data provided by the protected resource
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `resource_tos_uri`
Metadata Description: URL of a page containing human-readable information about the protected resource's terms of service
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `tls_client_certificate_bound_access_tokens`
Metadata Description: Boolean value indicating protected resource support for mutual-TLS client certificate-bound access tokens
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `authorization_details_types_supported`
Metadata Description: JSON array containing a list of the authorization details `type` values supported by the resource server when the `authorization_details` request parameter is used
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `dpop_signing_alg_values_supported`
Metadata Description: JSON array containing a list of the JWS `alg` values supported by the resource server for validating DPoP proof JWTs
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `dpop_bound_access_tokens_required`

Metadata Description: Boolean value specifying whether the protected resource always requires the use of DPoP-bound access tokens
Change Controller: IETF
Specification Document(s): Section 2 of RFC 9728

Metadata Name: `signed_metadata`
Metadata Description: Signed JWT containing metadata parameters about the protected resource as claims
Change Controller: IETF
Specification Document(s): Section 2.2 of RFC 9728

8.2. OAuth Authorization Server Metadata Registry

IANA has registered the following authorization server metadata parameter in the "OAuth Authorization Server Metadata" registry established in "OAuth 2.0 Authorization Server Metadata" [RFC8414].

8.2.1. Registry Contents

Metadata Name: `protected_resources`
Metadata Description: JSON array containing a list of resource identifiers for OAuth protected resources
Change Controller: IETF
Specification Document(s): Section 4 of RFC 9728

8.3. Well-Known URIs Registry

This specification registers the well-known URI defined in Section 3 in the "Well-Known URIs" registry [IANA.well-known].

8.3.1. Registry Contents

URI Suffix: `oauth-protected-resource`
Reference: Section 3 of RFC 9728
Status: permanent
Change Controller: IETF
Related Information: (none)

9. References

9.1. 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>.

[BCP47] Best Current Practice 47, <https://www.rfc-editor.org/info/bcp47>.
At the time of writing, this BCP comprises the following:

Phillips, A., Ed. and M. Davis, Ed., "Matching of Language Tags", BCP 47, RFC 4647, DOI 10.17487/RFC4647, September 2006, <https://www.rfc-editor.org/info/rfc4647>.

Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[IANA.Language] IANA, "Language Subtag Registry", <https://www.iana.org/assignments/language-subtag-registry>.

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

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

[JWK] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[JWS] 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>.

[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>.

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

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/info/rfc7591>.

[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>.

[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>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/info/rfc8414>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/info/rfc8705>.

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, <https://www.rfc-editor.org/info/rfc8707>.

[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>.

[RFC9111] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

[RFC9396] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, DOI 10.17487/RFC9396, May 2023, <https://www.rfc-editor.org/info/rfc9396>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/info/rfc9449>.

[RFC9525] Saint-Andre, P. and R. Salz, "Service Identity in TLS", RFC 9525, DOI 10.17487/RFC9525, November 2023, <https://www.rfc-editor.org/info/rfc9525>.

[UNICODE] The Unicode Consortium, "The Unicode Standard", <https://www.unicode.org/versions/latest/>.

[USA15] Whistler, K., Ed., "Unicode Normalization Forms", Unicode Standard Annex #15, 14 August 2024, <https://www.unicode.org/reports/tr15/>.

9.2. Informative References

[FAPI.MessageSigning] Tonge, D. and D. Fett, "FAPI 2.0 Message Signing (Draft)", 24 March 2023, <https://openid.net/specs/fapi-2_0-message-signing.html>.

[IANA.JOSE] IANA, "JSON Web Signature and Encryption Algorithms", <https://www.iana.org/assignments/jose>.

[IANA.well-known] IANA, "Well-Known URIs", <https://www.iana.org/assignments/well-known-uris>.

[OpenID.Discovery] Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0 incorporating errata set 2", 15 December 2023, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

[OWASP.SSRF] OWASP Foundation, "OWASP Server-Side Request Forgery Prevention Cheat Sheet", <https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html>.

[RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, DOI 10.17487/RFC7033, September 2013, <https://www.rfc-editor.org/info/rfc7033>.

[RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <https://www.rfc-editor.org/info/rfc8620>.

[RFC9470] Bertocci, V. and B. Campbell, "OAuth 2.0 Step Up Authentication Challenge Protocol", RFC 9470, DOI 10.17487/RFC9470, September 2023, <https://www.rfc-editor.org/info/rfc9470>.

[RFC9700] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "Best Current Practice for OAuth 2.0 Security", BCP 240, RFC 9700, DOI 10.17487/RFC9700, January 2025, <https://www.rfc-editor.org/info/rfc9700>.

Acknowledgements

The authors of this specification would like to thank the attendees of the IETF 115 OAuth and HTTP API Working Group meetings and the attendees of subsequent OAuth Working Group meetings for their input on this specification. We would also like to thank Amanda Baber, Mike Bishop, Ralph Bragg, Brian Campbell, Deb Cooley, Gabriel Corona, Roman Danyliw, Vladimir Dzhuvinov, George Fletcher, Arnt Gulbrandsen, Pieter Kasselman, Murray Kucherawy, David

Mandelberg, Tony Nadalin, Francesca Palombini, John Scudder, Rifaat Shekh-Yusef, Filip Skokan, Orie Steele, Atul Tulshibagwale, Éric Vyncke, Paul Wouters, and Bo Wu for their contributions to the specification.

Authors' Addresses

Michael B. Jones
Self-Issued Consulting
Email: michael_b_jones@hotmail.com
URI: https://self-issued.info/

Phil Hunt
Independent Identity, Inc.
Email: phil.hunt@yahoo.com

Aaron Parecki
Okta
Email: aaron@parecki.com
URI: https://aaronparecki.com/