Workgroup: REGEXT Working Group
Internet-Draft:
draft-ietf-regext-rdap-openid-14
Published: 24 May 2022
Intended Status: Standards Track
Expires: 25 November 2022
Authors: S. Hollenbeck
Verisign Labs

Federated Authentication for the Registration Data Access Protocol (RDAP) using OpenID Connect

Abstract

The Registration Data Access Protocol (RDAP) provides "RESTful" web services to retrieve registration metadata from domain name and regional internet registries. RDAP allows a server to make access control decisions based on client identity, and as such it includes support for client identification features provided by the Hypertext Transfer Protocol (HTTP). Identification methods that require clients to obtain and manage credentials from every RDAP server operator present management challenges for both clients and servers, whereas a federated authentication system would make it easier to operate and use RDAP without the need to maintain server-specific client credentials. This document describes a federated authentication system for RDAP based on OpenID Connect.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 25 November 2022.

Table of Contents

1. Introduction

The Registration Data Access Protocol (RDAP) provides "RESTful" web services to retrieve registration metadata from domain name and regional internet registries. RDAP allows a server to make access control decisions based on client identity, and as such it includes support for client identification features provided by the Hypertext Transfer Protocol (HTTP) [RFC7230].

RDAP is specified in multiple documents, including "HTTP Usage in the Registration Data Access Protocol (RDAP)" [RFC7480], "Security Services for the Registration Data Access Protocol (RDAP)" [RFC7481], "Registration Data Access Protocol Query Format" [RFC9082], and "JSON Responses for the Registration Data Access Protocol (RDAP)" [RFC9083]. RFC 7481 describes client identification and authentication services that can be used with RDAP, but it does not specify how any of these services can (or should) be used with RDAP.

1.1. Problem Statement

The traditional "user name and password" authentication method does not scale well in the RDAP ecosystem. Assuming that all domain name and address registries will eventually provide RDAP service, it is impractical and inefficient for users to secure login credentials from the hundreds of different server operators. Authentication methods based on user names and passwords do not provide information that describes the user in sufficient detail (while protecting the personal privacy of the user) for server operators to make fine-

grained access control decisions based on the user's identity. The authentication system used for RDAP needs to address all of these needs.

1.2. Proposal

A basic level of RDAP service can be provided to users who possess an identifier issued by a recognized provider who is able to authenticate and validate the user. The identifiers issued by social media services, for example, can be used. Users who require higher levels of service (and who are willing to share more information about them self to gain access to that service) can secure identifiers from specialized providers who are or will be able to provide more detailed information about the user. Server operators can then make access control decisions based on the identification information provided by the user.

A federated authentication system in which an RDAP server outsources identification and authentication services to a trusted OpenID Provider would make it easier to operate and use RDAP by re-using existing identifiers to provide a basic level of access. It can also provide the ability to collect additional user identification information, and that information can be shared with the consent of the user. This type of system allows an RDAP server to make access control decisions based on the nature of a query and the identity, authentication, and authorization information that is received from the OpenID Provider. This document describes a federated authentication system for RDAP based on OpenID Connect [OIDC] that meets all of these needs.

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Federated Authentication for RDAP

RDAP itself does not include native security services. Instead, RDAP relies on features that are available in other protocol layers to provide needed security services including access control, authentication, authorization, availability, data confidentiality, data integrity, and identification. A description of each of these security services can be found in "Internet Security Glossary, Version 2" [RFC4949]. This document focuses on a federated authentication system for RDAP that provides services for authentication, authorization, and identification, allowing a server

operator to make access control decisions. Section 3 of RFC 7481 [RFC7481] describes general considerations for RDAP access control, authentication, and authorization.

The traditional client-server authentication model requires clients to maintain distinct credentials for every RDAP server. This situation can become unwieldy as the number of RDAP servers increases. Federated authentication mechanisms allow clients to use one credential to access multiple RDAP servers and reduce client credential management complexity.

3.1. RDAP and OpenID Connect

OpenID Connect 1.0 [OIDCC] is a decentralized, single sign-on (SSO) federated authentication system that allows users to access multiple web resources with one identifier instead of having to create multiple server-specific identifiers. Users acquire identifiers from OpenID Providers, or OPs. Relying Parties, or RPs, are applications (such as RDAP) that outsource their user authentication function to an OP. OpenID Connect is built on top of the authorization framework provided by the OAuth 2.0 [RFC6749] protocol.

The OAuth authorization framework describes a method for users to access protected web resources without having to hand out their credentials. Instead, clients are issued Access Tokens by authorization servers with the permission of the resource owners. Using OpenID Connect and OAuth, multiple RDAP servers can form a federation and clients can access any server in the federation by providing one credential registered with any OP in that federation. The OAuth authorization framework is designed for use with HTTP and thus can be used with RDAP.

3.1.1. Terminology

This document uses the terms "client" and "server" defined by RDAP [RFC7480]. An RDAP client performs the role of an OpenID Connect Core [OIDCC] Entity or End-User. An RDAP server performs the role of an OpenID Connect Core Relying Party (RP). Additional terms from Section 1.2 of the OpenID Connect Core specification are incorporated by reference.

3.1.2. Overview

At a high level, RDAP authentication of a browser-like client using OpenID Connect requires completion of the following steps:

1. An RDAP client sends an RDAP "help" query to an RDAP server to determine the type and capabilities of the OpenID Authorization Servers that are used by the RDAP server. This information is returned in the rdapConformance section of the response. A

value of "roidc1" indicates support for the extension described in this specification. If one or more remote Authorization Servers are supported, the RDAP client SHOULD evaluate the additional information described in Section 4.1.3 in order to discover the capabilties of the RDAP server and optionally obtain the set of supported OPs.

2. An RDAP client (acting as an OpenID End-User) sends an RDAP "login" request to an RDAP server as described in Section 4.2.
3. The RDAP server (acting as an OpenID Relying Party (RP)) prepares an Authentication Request containing the desired request parameters.
4. The RDAP server sends the RDAP client and Authentication Request to an Authorization Server operated by an OpenID Provider (OP) using an HTTP redirect.
5. The Authorization Server authenticates the End-User.
6. The Authorization Server obtains End-User consent/ authorization.
7. The Authorization Server sends the RDAP Client back to the RDAP server with an Authorization Code using an HTTP redirect.
8. The RDAP server requests a response using the Authorization Code at the Token Endpoint.
9. The RDAP server receives a response that contains an ID Token and Access Token in the response body.
10. The RDAP server validates the ID Token and retrieves the claims associated with the End-User's identity.

The RDAP server can then make identification, authorization, and access control decisions based on End-User identity information and local policies. Note that OpenID Connect describes different process flows for other types of clients, such as script-based or command line clients.

3.1.3. RDAP Authentication and Authorization Steps

End-Users MAY present an identifier (an OpenID) issued by an OP to use OpenID Connect with RDAP. If the RDAP server supports a default Authorization Server or End-User identifier discovery is not supported, the End-User identifier MAY be omitted. An OP SHOULD include support for the claims described in Section 3.1.4 to provide additional information needed for RDAP End-User authorization. OpenID Connect requires RPs to register with OPs to use OpenID Connect services for an End-User. The registration process is often completed using out-of-band methods, but it is also possible to use the automated method described by the "OpenID Connect Dynamic Client Registration" protocol [OIDCR]. The parties involved can use any method that is mutually acceptable.

3.1.3.1. Provider Discovery

An RDAP server/RP needs to be able to map an End-User's identifier to an OP. This can be accomplished using the OPTIONAL "OpenID Connect Discovery" protocol [OIDCD], but that protocol is not widely implemented. Out-of-band methods are also possible and can be more dependable. For example, an RP can support a limited number of OPs and maintain internal associations of those identifiers with the OPs that issued them.

Alternatively, if mapping of an End-User's identifier is not possible, or not supported by the RDAP server, the RDAP server SHOULD support explicit specification of a remote OP by the RDAP client in the form of a query parameter as described in Section 4.2.2. An RDAP server SHOULD provide information about its capabilities and supported OPs in the "help" query response in the "openidcConfiguration" data structure described in Section 4.1.3.

An RP can also ask an End-User to identify the OP that issued their identifier as part of an RDAP query workflow. In this case, the RP will need to maintain state for the association between the user's identifier and the OP in order to process later queries that rely on passing the access token and user identifier as authorization parameters. An RDAP server MUST support at least one of these methods of OP discovery.

3.1.3.2. Authentication Request

Once the OP is known, an RP MUST form an Authentication Request and send it to the OP as described in Section 3 of the OpenID Connect Core protocol [OIDCC]. The authentication path followed (authorization, implicit, or hybrid) will depend on the Authentication Request response_type set by the RP. The remainder of the processing steps described here assume that the Authorization Code Flow is being used by setting "response_type=code" in the Authentication Request.

The benefits of using the Authorization Code Flow for authenticating a human user are described in Section 3.1 of the OpenID Connect Core protocol. The Implicit Flow is more commonly used by clients implemented in a web browser using a scripting language; it is described in Section 3.2 of the OpenID Connect Core protocol. The Hybrid Flow (described in Section 3.3 of the OpenID Connect Core protocol) combines elements of the Authorization and Implicit Flows by returning some tokens from the Authorization Endpoint and others from the Token Endpoint.

An Authentication Request can contain several parameters. REQUIRED parameters are specified in Section 3.1.2.1 of the OpenID Connect

Core protocol [OIDCC]. Apart from these parameters, it is RECOMMENDED that the RP include the optional "login_hint" parameter in the request, with the value being that of the "roidc1_id" query parameter of the End-User's RDAP "login" request, if provided. Passing the "login_hint" parameter allows a client to pre-fill login form information, so logging in can be more convenient for users. Other parameters MAY be included.

The OP receives the Authentication Request and attempts to validate it as described in Section 3.1.2.2 of the OpenID Connect Core protocol [OIDCC]. If the request is valid, the OP attempts to authenticate the End-User as described in Section 3.1.2.3 of the OpenID Connect Core protocol [OIDCC]. The OP returns an error response if the request is not valid or if any error is encountered.

3.1.3.3. End-User Authorization

After the End-User is authenticated, the OP MUST obtain authorization information from the End-User before releasing information to the RDAP Server/RP. This process is described in Section 3.1.2.4 of the OpenID Connect Core protocol [OIDCC].

3.1.3.4. Authorization Response and Validation

After the End-User is authenticated, the OP will send a response to the RP that describes the result of the authorization process in the form of an Authorization Grant. The RP MUST validate the response. This process is described in Sections 3.1.2.5 - 3.1.2.7 of the OpenID Connect Core protocol [OIDCC].

3.1.3.5. Token Processing

The RP sends a Token Request using the Authorization Grant to a Token Endpoint to obtain a Token Response containing an Access Token, ID Token, and an OPTIONAL Refresh Token. The RP MUST validate the Token Response. This process is described in Section 3.1.3 of the OpenID Connect Core protocol [OIDCC].

3.1.3.6. Delivery of User Information

The set of claims can be retrieved by sending a request to a UserInfo Endpoint using the Access Token. The claims MAY be returned in the ID Token. The process of retrieving claims from a UserInfo Endpoint is described in Section 5.3 of the OpenID Connect Core protocol [OIDCC].

OpenID Connect specifies a set of standard claims in Section 5.1. Additional claims for RDAP are described in Section 3.1.4.

3.1.4. Specialized Claims for RDAP

OpenID Connect claims are pieces of information used to make assertions about an entity. Section 5 of the OpenID Connect Core protocol [OIDCC] describes a set of standard claims that can be used to identify a person. Section 5.1.2 notes that additional claims MAY be used, and it describes a method to create them. The set of claims that are specific to RDAP are associated with an OAuth scope request parameter value (see Section 3.3 of RFC 6749 ([RFC6749])) of "rdap".

3.1.4.1. Stated Purposes

There are communities of RDAP users and operators who wish to make and validate claims about a user's "need to know" when it comes to requesting access to a resource. For example, a law enforcement agent or a trademark attorney may wish to be able to assert that they have a legal right to access a protected resource, and a server operator will need to be able to receive and validate that claim. These needs can be met by defining and using an additional "rdap_allowed_purposes" claim.

The "rdap_allowed_purposes" claim identifies the purposes for which access to a protected resource can be requested by an End-User. Use of the "rdap_allowed_purposes" claim is OPTIONAL; processing of this claim is subject to server acceptance of the purposes, the trust level assigned to this claim by the server, and successful authentication of the End-User. Unrecognized purpose values MUST be ignored and the associated query MUST be processed as if the unrecognized purpose value was not present at all.

The "rdap_allowed_purposes" claim is represented as an array of case-sensitive StringOrURI values as specified in Section 2 of the JSON Web Token (JWT) specification ([RFC7519]). An example:

"rdap_allowed_purposes": ["domainNameControl","dnsTransparency"]

Individual purpose values are registered with IANA. Each entry in the registry contains the following fields:

Value: the purpose string value being registered. Value strings can contain upper case characters from "A" to "Z", lower case ASCII characters from "a" to "z", and the underscore ("_") character. Value strings contain at least one character and no more than 64 characters.

Description: a one- or two-sentence description of the meaning of the purpose value, how it might be used, and/or how it should be interpreted by clients and servers.

This registry is operated under the "Specification Required" policy defined in RFC 5226 ([RFC5226]). The set of initial values used to populate the registry as described in Section 8.3 are taken from the final report produced by the Expert Working Group on gTLD Directory Services chartered by the Internet Corporation for Assigned Names and Numbers (ICANN).

3.1.4.2. Do Not Track

There are also communities of RDAP users and operators who wish to make and validate claims about a user's wish to not have their queries logged, tracked, or recorded. For example, a law enforcement agent may wish to be able to assert that their queries are part of a criminal investigation and should not be tracked due to a risk of query exposure compromising the investigation, and a server operator will need to be able to receive and validate that claim. These needs can be met by defining and using an additional "do not track" claim.

The "do not track" ("rdap_dnt_allowed") claim can be used to identify an End-User that is authorized to perform queries without the End-User's association with those queries being logged, tracked, or recorded by the server. Client use of the "rdap_dnt_allowed" claim is OPTIONAL. Server operators MUST NOT log, track, or record any association of the query and the End-User's identity if the End-User is successfully identified and authorized, the "rdap_dnt_allowed" claim is present, the value of the claim is "true", and accepting the claim complies with local regulations regarding logging and tracking.

The "rdap_dnt_allowed" value is represented as a JSON boolean literal. An example:

```
rdap_dnt_allowed: true
```

No special query tracking processing is required if this claim is not present or if the value of the claim is "false". Use of this claim MUST be limited to End-Users who are granted "do not track" privileges in accordance with service policies and regulations. Specification of these policies and regulations is beyond the scope of this document.

4. Protocol Parameters

This specification adds the following protocol parameters to RDAP:

1. Data structures to return information that describes an established session, the information needed to establish a session for a UI-constrained device, and the RDAP server's OpenID Connect extension configuration.

2. A query parameter to request authentication for a specific End-User identity.
3. A query parameter to identify the purpose of the query.
4. A query parameter to request that the server not log or otherwise record information about the identity associated with a query.
5. Path segments to start, stop, refresh, and determine the status of an authenticated session for a specific End-User identity.

4.1. Data Structures

This specification describes three new data structures that are used to return information to a client: a "roidc1_session" data structure that contains information that describes an established session, a "roidc1_deviceInfo" data structure that contains information that describes an active attempt to establish a session on a UI-constrained device, and a "roidc1_openidcConfiguration" data structure that describes the OpenID Connect configuration and related extension features supported by the RDAP server.

4.1.1. Session

The "roidc1_session" data structure is an object that contains two sub-objects:

1. A "userClaims" object that contains the set of claims associated with the End-User's identity as used/requested by the RDAP server to make access control decisions. The set of possible values is determined by OP policy.
2. A "sessionInfo" object that contains two members:

    a. "tokenExpiration": an integer value that represents the number of seconds from the current time for which the Access Token remains valid, and
    b. "tokenRefresh": A boolean value that indicates if the OP supports refresh tokens. As described in RFC 6749 [RFC6749], support for refresh tokens is OPTIONAL.

An example of a "roidc1_session" data structure:

```
"roidc1_session": {
  "userClaims": {
    "sub": "103892603076825016132",
    "name": "User Person",
    "given_name": "User",
    "family_name": "Person",
    "picture": "https://lh3.example.com/a-/AOh14=s96-c",
    "email": "user@example.com",
    "email_verified": true,
    "locale": "en",
    "rdap_allowed_purposes": [
      "domainNameControl",
      "personalDataProtection"
    ],
    "rdap_dnt_allowed": false
  },
  "sessionInfo": {
    "tokenExpiration": 3599,
    "tokenRefresh": true
  }
}
```

Figure 1

4.1.2. Device Info

The flow described in Section 3.1.3 requires an End-User to interact with a server using a user interface that can process HTTP. This will not work well in situations where the client is automated or an End-User is using a command line user interface such as curl or wget. This limitation can be addressed using a web browser on a second device. The information that needs to be entered using the web browser is contained in the "roidc1_deviceInfo" data structure.

The "roidc1_deviceInfo" data structure is an object that contains three members:

1. "verification_url": the URL that the End-User needs to visit using the web browser,
2. "user_code": the string value that the End-User needs to enter on the form presented in the web browser, and
3. "expires_in": an integer value that represents the number of seconds after which the opportunity to visit the URL and enter the user_code will expire.

An example of a "roidc1_deviceInfo" data structure:

```
"roidc1_deviceInfo": {
  "verification_url": "https://www.example.com/device",
  "user_code": "NJJQ-GJFC",
  "expires_in": "1800"
}
```

Figure 2

4.1.3. OpenID Connect Configuration

The "roidc1_openidcConfiguration" data structure is an object with the following members:

1. "dntSupported": (MANDATORY) a boolean value that describes RDAP server support for the "roidc1_dnt" query parameter (see Section 4.3.2).
2. "endUserIdentifierDiscoverySupported": (OPTIONAL) a boolean value that describes RDAP server support for discovery of End-User identifiers. The default value is "true".
3. "issuerIdentifierSupported": (OPTIONAL) a boolean value that describes RDAP server support for explicit client specification of an Issuer Identifier. The default value is "true".
4. "implicitTokenRefreshSupported": (OPTIONAL) a boolean value that describes RDAP server support for implicit token refresh. The default value is "false".
5. "openidcProviders": (OPTIONAL) a list of objects with the following members that describes the set of OPs that are supported by the RDAP server. This data is RECOMMENDED if the value of issuerIdentifierSupported is "true":
6. a. "iss": (MANDATORY) a string value equal to Issuer Identifier of the OP as per OpenID Connect Core specification [OIDCC]
   b. "name": (MANDATORY) a string value representing the human-friendly name of the OP.
   c. "default": (OPTIONAL) a boolean value that describes RDAP server support for an OPTIONAL default OP that will be used when a client omits the "roidc1_id" and "roidc1_iss" query parameters from a "roidc1_session/login" request. Only one member of this set can be identified as the default OP by setting a value of "true". The default value is "false".

An example of a "roidc1_openidcConfiguration" data structure:

```
"roidc1_openidcConfiguration": {
  "dntSupported": false,
  "endUserIdentifierDiscoverySupported": true,
  "issuerIdentifierSupported": true,
  "openidcProviders":
    [
      {
        "iss": "https://idp.example.com",
        "name": "Example IDP"
      },
      {
        "iss": "https://accounts.example.net",
        "name": "Login with EXAMPLE"
      },
      {
        "iss": "https://auth.nic.example/auth/realms/rdap",
        "name": "Default OP for the Example RDAP server",
        "default": "true"
      }
    ]
}
```

Figure 3

4.2. Client Login

Client authentication is requested by sending a "roidc1_session/login" request to an RDAP server. If the RDAP server supports only remote Authorization Servers, the "roidc1_session/login" request MUST include at least one of an End-User Identifier or an OP Issuer Identifier.

4.2.1. End-User Identifier

The End-User identifier is delivered using one of two methods: by adding a query component to an RDAP request URI using the syntax described in Section 3.4 of RFC 3986 [RFC3986], or by including an HTTP authorization header for the Basic authentication scheme as described in RFC 7617 [RFC7617]. Clients can use either of these methods to deliver the End-User identifier to a server that supports remote Authorization Servers and End-User identifier discovery. Servers that support remote Authorization Servers and End-User identifier discovery MUST accept both methods. If the RDAP server supports a default Authorization Server or End-User identifier discovery is not supported, the End-User identifier MAY be omitted.

The query used to request client authentication is represented as an OPTIONAL "key=value" pair using a key value of "roidc1_id" and a

value component that contains the client identifier issued by an OP. An example for client identifier "user.idp.example":

https://example.com/rdap/roidc1_session/login?
roidc1_id=user.idp.example

The authorization header for the Basic authentication scheme contains a Base64-encoded representation of the client identifier issued by an OP. No password is provided. An example for client identifier "user.idp.example":

https://example.com/rdap/roidc1_session/login

Authorization: Basic dXNlci5pZHAuZXhhbXBsZQ==

An example for use with a default Authorization Server:

https://example.com/rdap/roidc1_session/login

4.2.2. OP Issuer Identifier

The OP's Issuer Identifier is delivered by adding a query component to an RDAP request URI using the syntax described in Section 3.4 of RFC 3986 [RFC3986]. If the RDAP server supports a default Authorization Server, the Issuer Identifier MAY be omitted.

The query used to request client authentication is represented as an OPTIONAL "key=value" pair using a key value of "roidc1_iss" and a value component that contains the Issuer Identifier associated with an OP. An RDAP server MAY accept Issuer Identifiers not specified in the "roidc1_openidcConfiguration" data structure and MAY also decide to accept specific Issuer Identifiers only from specific clients.

An example for Issuer Identifier "https://idp.example.com":

https://example.com/rdap/roidc1_session/login?
roidc1_iss=https%3A%2F%2Fidp.example.com

4.2.3. Login Response

The response to the login request MUST use the response structures specified in RFC 9083 [RFC9083]. In addition, the response MUST include an indication of the requested operation's success or failure in the "notices" data structure (including the client identifier), and, if successful, a "roidc1_session" data structure.

An example of a successful "roidc1_session/login" response:

```
{
  "rdapConformance": [
    "roidc1"
  ],
  "lang": "en-US",
  "notices": {
    "title": "Login Result",
    "description": [
      "Login succeeded",
      "user.idp.example"
    ],
  },
  "roidc1_session": {
    "userClaims": {
      "sub": "103892603076825016132",
      "name": "User Person",
      "given_name": "User",
      "family_name": "Person",
      "picture": "https://lh3.example.com/a-/AOh14=s96-c",
      "email": "user@example.com",
      "email_verified": true,
      "locale": "en",
      "rdap_allowed_purposes": [
        "domainNameControl",
        "personalDataProtection"
      ],
      "rdap_dnt_allowed": false
    },
    "sessionInfo": {
      "tokenExpiration": 3599,
      "tokenRefresh": true
    }
  }
}
```


Figure 4

An example of a failed "roidc1_session/login" response:

```
{
  "rdapConformance": [
    "roidc1"
  ],
  "lang": "en-US",
  "notices": {
    "title": "Login Result",
    "description": [
      "Login failed",
      "user.idp.example"
      ]
  }
}
```

Figure 5

4.2.4. Clients with Limited User Interfaces

The "OAuth 2.0 Device Authorization Grant" [RFC8628] provides an OPTIONAL method to request user authorization from devices that have an Internet connection, but lack a suitable browser for a more traditional OAuth flow. This method requires an End-User to use a second device (such as a smart telephone) that has access to a web browser for entry of a code sequence that is presented on the UI-constrained device.

4.2.4.1. UI-constrained Client Login

Client authentication is requested by sending a "roidc1_session/device" request to an RDAP server. If the RDAP server supports only remote Authorization Servers, the "roidc1_session/device" request MUST include an End-User identifier that's delivered using one of two methods: by adding a query component to an RDAP request URI using the syntax described in Section 3.4 of RFC 3986 [RFC3986], or by including an HTTP authorization header for the Basic authentication scheme as described in RFC 7617 [RFC7617]. If the RDAP server supports a default Authorization Server, the End-User identifier MAY be omitted. Clients can use either of these methods. Servers MUST support both methods.

The query used to request client authentication is represented as an OPTIONAL "key=value" pair using a key value of "roidc1_id" and a value component that contains the client identifier issued by an OP.

An example using wget for client identifier "user.idp.example":

```
wget -qO- --keep-session-cookies --save-cookies\
https://example.com/rdap/roidc1_session/device?roidc1_id=user.idp.exa
```

Figure 6

The authorization header for the Basic authentication scheme contains a Base64-encoded representation of the client identifier issued by an OP. No password is provided.

An example using curl and an authorization header:

```
curl -H "Authorization: Bearer dXNlci5pZHAuZXhhbXBsZQ==\"\
-c cookies.txt https://example.com/rdap/roidc1_session/device
```

Figure 7

The response to this request MUST use the response structures specified in RFC 9083 [RFC9083]. In addition, the response MUST include an indication of the requested operation's success or failure in the "notices" data structure (including the client identifier), and, if successful, a "roidc1_deviceInfo" data structure.

An example of a "roidc1_session/device" response:

```
{
  "rdapConformance": [
    "roidc1"
  ],
  "lang": "en-US",
  "notices": {
    "title": "Device Login Result",
    "description": [
      "Login succeeded",
      "user.idp.example"
    ]
  },
  "roidc1_deviceInfo": {
    "verification_url": "https://www.example.com/device",
    "user_code": "NJJQ-GJFC",
    "expires_in": 1800
  }
}
```

Figure 8

4.2.4.2. UI-constrained Client Login Polling

After successful processing of the "roidc1_session/device" request, the client MUST send a "roidc1_session/devicepoll" request to the RDAP server to continue the login process. This request performs the polling function described in RFC 8628 [RFC8628], allowing the RDAP server to wait for the End-User to enter the information returned from the "roidc1_session/device" request using the interface on their second device. After the End-User has completed that process,

or if the process fails or times out, the OP will respond to the polling requests with an indication of success or failure.

An example using wget:

```
wget -qO- --load-cookies cookies.txt\
https://example.com/rdap/roidc1_session/devicepoll
```

Figure 9

An example using curl:

```
curl -b cookies.txt https://example.com/rdap/roidc1_session/devicepol
```

Figure 10

The response to this request MUST use the response structures described in Section 4.2. RDAP query processing can continue normally on the UI-constrained device once the "login" process has been completed.

4.3. RDAP Query Parameters

This specification describes two OPTIONAL query parameters for use with RDAP queries that request access to information associated with protected resources:

1. "roidc1_qp": A query parameter to identify the purpose of the query.
2. "roidc1_dnt": A query parameter to request that the server not log or otherwise record information about the identity associated with a query.

One or both of these parameters MAY be added to an RDAP request URI using the syntax described in Section 3.4 of RFC 3986 [RFC3986].

4.3.1. RDAP Query Purpose

This query is represented as a "key=value" pair using a key value of "roidc1_qp" and a value component that contains a single query purpose string from the set of allowed purposes associated with the End-User's identity (see Section 3.1.4.1). If present, the server SHOULD compare the value of the parameter to the "rdap_allowed_purposes" claim values associated with the End-User's identity and ensure that the requested purpose is present in the set of allowed purposes. The RDAP server MAY choose to ignore both requested purpose and the "rdap_allowed_purposes" claim values if they are inconsistent with local server policy. The server MUST return an HTTP 403 (Forbidden) response if the requested purpose is not an allowed purpose. If this parameter is not present, the server

MUST proces the query and make an acces control decision based on any other information known to the server about the End-User and the information they are requesting. For example, a server MAY treat the request as one performed by an unidentified or unauthenticated user and return either an error or an appropriate subset of the available data. An example domain query using the "roidc1_qp" query parameter:

https://example.com/rdap/domain/example.com?roidc1_qp=legalActions

4.3.2. RDAP Do Not Track

This query is represented as a "key=value" pair using a key value of "roidc1_dnt" and a value component that contains a single boolean value. A value of "true" indicates that the End-User is requesting that their query not be tracked or logged in accordance with server policy. A value of "false" indicates that the End-User is accepting that their query can be tracked or logged in accordance with server policy. The server MUST return an HTTP 501 (Not Implemented) response if the server is unable to perform the action requested by this query parameter. An example domain query using the "roidc1_dnt" query parameter:

https://example.com/rdap/domain/example.com?roidc1_dnt=true

4.4. Session Status

Clients MAY send a query to an RDAP server to determine the status of an existing login session using a "roidc1_session/status" path segment. An example "roidc1_session/status" request:

https://example.com/rdap/roidc1_session/status

The response to this query MUST use the response structures specified in RFC 9083 [RFC9083]. In addition, the response MUST include an indication of the requested operation's success or failure in the "notices" data structure (including the client identifier), and, if successful, a "roidc1_session" data structure.

An example of a "roidc1_session/status" response:

```
{
  "rdapConformance": [
    "roidc1"
  ],
  "lang": "en-US",
  "notices": {
    "title": "Session Status Result",
    "description": [
      "Session status succeeded",
      "user.idp.example"
    ]
  },
  "roidc1_session": {
    "userClaims": {
      "sub": "103892603076825016132",
      "name": "User Person",
      "given_name": "User",
      "family_name": "Person",
      "picture": "https://lh3.example.com/a-/AOh14=s96-c",
      "email": "user@example.com",
      "email_verified": true,
      "locale": "en",
      "rdap_allowed_purposes": [
        "domainNameControl",
        "personalDataProtection"
      ],
      "rdap_dnt_allowed": false
    },
    "sessionInfo": {
      "tokenExpiration": 3490,
      "tokenRefresh": true
    }
  }
}
```


Figure 11

4.5. Session Refresh

Clients MAY send a request to an RDAP server to refresh, or extend, an existing login session using a "roidc1_session/refresh" path segment. The RDAP server MAY attempt to refresh the access token associated with the current session as part of extending the session for a period of time determined by the RDAP server. As described in RFC 6749 [RFC6749], OP support for refresh tokens is OPTIONAL. An RDAP server MUST determine if the OP supports token refresh and process the refresh request by either requesting refresh of the access token or by returning a response that indicates that token

refresh is not supported by the OP in the "notices" data structure.
An example "roidc1_session/refresh" request:

https://example.com/rdap/roidc1_session/refresh

The response to this request MUST use the response structures specified in RFC 9083 [RFC9083]. In addition, the response MUST include an indication of the requested operation's success or failure in the "notices" data structure (including the client identifier), and, if successful, a "roidc1_session" data structure.

An example of a "roidc1_session/refresh" response:


```
{
  "rdapConformance": [
    "roidc1"
  ],
  "lang": "en-US",
  "notices": {
    "title": "Session Refresh Result",
    "description": [
      "Session refresh succeeded",
      "user.idp.example",
      "Token refresh succeeded."
    ]
  },
  "roidc1_session": {
    "userClaims": {
      "sub": "103892603076825016132",
      "name": "User Person",
      "given_name": "User",
      "family_name": "Person",
      "picture": "https://lh3.example.com/a-/AOh14=s96-c",
      "email": "user@example.com",
      "email_verified": true,
      "locale": "en",
      "rdap_allowed_purposes": [
        "domainNameControl",
        "personalDataProtection"
      ],
      "rdap_dnt_allowed": false
    },
    "sessionInfo": {
      "tokenExpiration": 3599,
      "tokenRefresh": true
    }
  }
}
```

Figure 12

Alternatively, an RDAP server MAY implicitly attempt to refresh an access token upon receipt of a query if the access token associated with an existing session has expired and the corresponding OP supports token refresh. The default RDAP server behavior is described in the "implicitTokenRefreshSupported" value that's include in the "roidc1_openidcConfiguration" data structure (see Section 4.1.3). If the value of "implicitTokenRefreshSupported" is "true", the client MAY either explicitly attempt to refresh the session using the "roidc1_session/refresh" query, or it MAY depend on the RDAP server to implicitly attempt to refresh the session as necessary when an RDAP query is received by the server. If the value of "implicitTokenRefreshSupported" is "false", the client MUST explicitly attempt to refresh the session using the "roidc1_session/refresh" query to extend an existing session. If a session cannot be extended for any reason, the client MUST establish a new session to continue authenticated query processing by submitting a "roidc1_session/login" query. If the OP does not support token refresh, the client MUST submit a new "roidc1_session/login" request to establish a new session once an access token has expired.

4.6. Client Logout

Clients MAY send a request to an RDAP server to terminate an existing login session. Termination of a session is requested using a "roidc1_session/logout" path segment. Access and refresh tokens can be revoked during the "roidc1_session/logout" process as described in RFC 7009 [RFC7009] if supported by the OP (token revocation endpoint support is OPTIONAL per RFC 8414 [RFC8414]). If supported, this feature SHOULD be used to ensure that the tokens are not mistakenly associated with a future RDAP session. Alternatively, an RDAP server MAY attempt to logout from the OP using the "OpenID Connect RP-Initiated Logout" protocol ([OIDCL]) if that protocol is supported by the OP.

An example "roidc1_session/logout" request:

https://example.com/rdap/roidc1_session/logout

The response to this request MUST use the response structures specified in RFC 9083 [RFC9083]. In addition, the response MUST include an indication of the requested operation's success or failure in the "notices" data structure (including the client identifier). The "notices" data structure MUST also include an indication of the success or failure of any attempt to logout from the OP or to revoke the tokens issued by the OP.

An example of a "roidc1_session/logout" response:

```
{
  "rdapConformance": [
    "roidc1"
  ],
  "lang": "en-US",
  "notices": {
    "title": "Logout Result",
    "description": [
      "Logout succeeded",
      "user.idp.example",
      "Provider logout failed: Not supported by provider.",
      "Token revocation successful."
    ]
  }
}
```

Figure 13

In the absence of a "logout" request, an RDAP session MUST be terminated by the RDAP server after a server-defined period of time. The server should also take appropriate steps to ensure that the tokens associated with the terminated session cannot be reused. This SHOULD include revoking the tokens or logging out from the OP if either operation is supported by the OP.

4.7. Parameter Processing

Unrecognized query parameters MUST be ignored. An RDAP server that processes an authenticated query MUST determine if the End-User identification information is associated with an OP that is recognized and supported by the server. Servers MUST reject queries that include identification information that is not associated with a supported OP by returning an HTTP 501 (Not Implemented) response. An RDAP server that receives a query containing identification information associated with a recognized OP MUST perform the steps required to authenticate the user with the OP, process the query, and return an RDAP response that is appropriate for the End-User's level of authorization and access.

5. Token Exchange

ID tokens include an audience parameter that contains the OAuth 2.0 client_id of the RP as an audience value. In some operational scenarios (such as a client that is providing a proxy service), an RP can receive tokens with an audience value that does not include the RP's client_id. These tokens might not be trusted by the RP, and the RP might refuse to accept the tokens. This situation can be remedied by having the RP exchange these tokens with the OP for a set of trusted tokens that reset the audience parameter. This token

exchange protocol is described in RFC 8693 [RFC8693]. This issue is not visible to the RDAP client and should be managed by the OpenID implementation used by the RDAP server.

6. RDAP Query Processing

Once an RDAP session is active, an RDAP server MUST determine if the End-User is authorized to perform any queries that are received during the duration of the session. This MAY include rejecting queries outright, and it MAY include omitting or otherwise redacting information that the End-User is not authorized to receive. Specific processing requirements are beyond the scope of this document. A client can end a session explicitly by sending a "roidc1_session/logout" request to the RDAP server. A session can also be ended implicitly by the server after a server-defined period of time. The status of a session can be determined at any time by sending a "roidc1_session/status" query to the RDAP server.

An RDAP server MUST maintain session state information for the duration of an active session. This is commonly done using HTTP cookies as described in RFC 6265 [RFC6265]. Doing so allows End-User to submit queries without having to explicitly identify and authenticate themselves for each and every query.

7. RDAP Conformance

RDAP responses that contain values described in this document MUST indicate conformance with this specification by including an rdapConformance ([RFC9083]) value of "roidc1" (RDAP OpenID Connect version 1). The information needed to register this value in the RDAP Extensions Registry is described in Section 8.1.

Example rdapConformance structure with extension specified:

```
"rdapConformance" :
  [
    "rdap_level_0",
    "roidc1"
  ]
```

Figure 14

8. IANA Considerations

8.1. RDAP Extensions Registry

IANA is requested to register the following value in the RDAP Extensions Registry:

Extension identifier: roidc1

Registry operator: Any
Published specification: This document.
Contact: IESG <iesg@ietf.org>
Intended usage: This extension describes version 1 of a federated authentication method for RDAP using OAuth 2.0 and OpenID Connect.

8.2. JSON Web Token Claims Registry

IANA is requested to register the following values in the JSON Web Token Claims Registry:

Claim Name: "rdap_allowed_purposes"
Claim Description: This claim describes the set of RDAP query purposes that are available to an identity that is presented for access to a protected RDAP resource.
Change Controller: IESG
Specification Document(s): Section 3.1.4.1 of this document.
Claim Name: "rdap_dnt_allowed"
Claim Description: This claim contains a JSON boolean literal that describes a "do not track" request for server-side tracking, logging, or recording of an identity that is presented for access to a protected RDAP resource.
Change Controller: IESG
Specification Document(s): Section 3.1.4.2 of this document.

8.3. RDAP Query Purpose Registry

IANA is requested to create a new protocol registry to manage RDAP query purpose values. This registry should be named "Registration Data Access Protocol (RDAP) Query Purpose Values" and should appear under the "Registration Data Access Protocol (RDAP)" section of IANA's protocol registries. The information to be registered and the procedures to be followed in populating the registry are described in Section 3.1.4.1.

Section at http://www.iana.org/protocols: Registration Data Access Protocol (RDAP)

Name of registry: Registration Data Access Protocol (RDAP) Query Purpose Values

Registration Procedure: Specification Required

Reference: This document

Required information: See Section 3.1.4.1.

Review process: "Specification Required" as described in RFC 5226 [RFC5226].

Size, format, and syntax of registry entries: See Section 3.1.4.1.

Initial assignments and reservations:

-----BEGIN FORM-----

Value: domainNameControl

Description: Tasks within the scope of this purpose include creating and managing and monitoring a registrant's own domain name, including creating the domain name, updating information about the domain name, transferring the domain name, renewing the domain name, deleting the domain name, maintaining a domain name portfolio, and detecting fraudulent use of the Registrant's own contact information.

-----END FORM-----

-----BEGIN FORM-----

Value: personalDataProtection

Description: Tasks within the scope of this purpose include identifying the accredited privacy/proxy provider associated with a domain name and reporting abuse, requesting reveal, or otherwise contacting the provider.

-----END FORM-----

-----BEGIN FORM-----

Value: technicalIssueResolution

Description: Tasks within the scope of this purpose include (but are not limited to) working to resolve technical issues, including email delivery issues, DNS resolution failures, and web site functional issues.

-----END FORM-----

-----BEGIN FORM-----

Value: domainNameCertification

Description: Tasks within the scope of this purpose include a Certification Authority (CA) issuing an X.509 certificate to a subject identified by a domain name.

-----END FORM-----

-----BEGIN FORM-----

Value: individualInternetUse

Description: Tasks within the scope of this purpose include identifying the organization using a domain name to instill consumer trust, or contacting that organization to raise a customer complaint to them or file a complaint about them.

-----END FORM-----

-----BEGIN FORM-----

Value: businessDomainNamePurchaseOrSale

Description: Tasks within the scope of this purpose include making purchase queries about a domain name, acquiring a domain name from a registrant, and enabling due diligence research.

-----END FORM-----

-----BEGIN FORM-----

Value: academicPublicInterestDNSRResearch

Description: Tasks within the scope of this purpose include academic public interest research studies about domain names published in the registration data service, including public information about the registrant and designated contacts, the domain name's history and status, and domain names registered by a given registrant (reverse query).

-----END FORM-----

-----BEGIN FORM-----

Value: legalActions

Description: Tasks within the scope of this purpose include investigating possible fraudulent use of a registrant's name or address by other domain names, investigating possible trademark infringement, contacting a registrant/licensee's legal representative prior to taking legal action and then taking a legal action if the concern is not satisfactorily addressed.

-----END FORM-----

-----BEGIN FORM-----

Value: regulatoryAndContractEnforcement

Description: Tasks within the scope of this purpose include tax authority investigation of businesses with online presence, Uniform Dispute Resolution Policy (UDRP) investigation, contractual compliance investigation, and registration data escrow audits.

-----END FORM-----

-----BEGIN FORM-----

Value: criminalInvestigationAndDNSAbuseMitigation

Description: Tasks within the scope of this purpose include reporting abuse to someone who can investigate and address that abuse, or contacting entities associated with a domain name during an offline criminal investigation.

-----END FORM-----

-----BEGIN FORM-----

Value: dnsTransparency

Description: Tasks within the scope of this purpose involve querying the registration data made public by registrants to satisfy a wide variety of use cases around informing the general public.

-----END FORM-----

9. Implementation Status

NOTE: Please remove this section and the reference to RFC 7942 prior to publication as an RFC.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942 [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of

running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

Version -09 of this specification introduced changes that are incompatible with earlier implementations. Implementations that are consistent with this specification will be added as they are identified.

9.1. Editor Implementation

Location: https://procuratus.net/rdap/
Description: This implementation is a functionally-limited RDAP server that supports only the path segments described in this specification. It uses the "jumbojett/OpenID-Connect-PHP" library found on GitHub, which appears to no longer be under active development. The library was modified to add support for the device authorization grant. Session variable management is still a little buggy. Supported OPs include Google (Gmail) and Yahoo.
Level of Maturity: This is a "proof of concept" research implementation.
Coverage: This implementation includes all of the features described in this specification.
Version compatibility: Version -11+ of this specification.
Contact Information: Scott Hollenbeck, shollenbeck@verisign.com

9.2. Verisign Labs

Responsible Organization: Verisign Labs
Location: https://rdap.verisignlabs.com/
Description: This implementation includes support for domain registry RDAP queries using live data from the .cc and .tv country code top-level domains and the .career generic top-level domain. Three access levels are provided based on the authenticated identity of the client:

1. Unauthenticated: Limited information is returned in response to queries from unauthenticated clients.
2. Basic: Clients who authenticate using a publicly available identity provider like Google Gmail or Microsoft Hotmail will receive all of the information available to an unauthenticated client plus additional registration metadata, but no personally identifiable information associated with entities.
3. Advanced: Clients who authenticate using a more restrictive identity provider will receive all of the information available to a Basic client plus whatever information the server operator deems appropriate for a fully authorized

client. Currently supported identity providers include those developed by Verisign Labs (https://testprovider.rdap.verisignlabs.com/) and CZ.NIC (https://www.mojeid.cz/).

Level of Maturity: This is a "proof of concept" research implementation.

Coverage: This implementation includes all of the features described in this specification.

Version compatibility: Version -07 of this specification.

Contact Information: Scott Hollenbeck, shollenbeck@verisign.com

9.3. Viagenie

Responsible Organization: Viagenie

Location: https://auth.viagenie.ca

Description: This implementation is an OpenID identity provider enabling users and registries to connect to the federation. It also includes a barebone RDAP client and RDAP server in order to test the authentication framework. Various level of purposes are available for testing.

Level of Maturity: This is a "proof of concept" research implementation.

Coverage: This implementation includes most features described in this specification as an identity provider.

Version compatibility: Version -07 of this specification.

Contact Information: Marc Blanchet, marc.blanchet@viagenie.ca

10. Security Considerations

Security considerations for RDAP can be found in RFC 7481 [RFC7481]. Security considerations for OpenID Connect Core [OIDCC] and OAuth 2.0 [RFC6749] can be found in their reference specifications. OpenID Connect defines optional mechanisms for robust signing and encryption that can be used to provide data integrity and data confidentiality services as needed.

10.1. Authentication and Access Control

Having completed the client identification, authorization, and validation process, an RDAP server can make access control decisions based on a comparison of client-provided information and local policy. For example, a client who provides an email address (and nothing more) might be entitled to receive a subset of the information that would be available to a client who provides an email address, a full name, and a stated purpose. Development of these access control policies is beyond the scope of this document.

11. Acknowledgments

The author would like to acknowledge the following individuals for their contributions to the development of this document: Marc Blanchet, Tom Harrison, Russ Housley, Jasdip Singh, Rhys Smith, Jaromir Talir, Rick Wilhelm, and Alessandro Vesely. In addition, the Verisign Registry Services Lab development team of Joseph Harvey, Andrew Kaizer, Sai Mogali, Anurag Saxena, Swapneel Sheth, Nitin Singh, and Zhao Zhao provided critical "proof of concept" implementation experience that helped demonstrate the validity of the concepts described in this document.

Pawel Kowalik and Mario Loffredo provided significant text contributions that led to welcome improvements in several sections of this document. Their contributions are greatly appreciated.

12. References

12.1. Normative References

[OIDC] OpenID Foundation, "OpenID Connect", <http://openid.net/connect/>.

[OIDCC] OpenID Foundation, "OpenID Connect Core incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[OIDCD] OpenID Foundation, "OpenID Connect Discovery 1.0 incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-discovery-1_0.html>.

[OIDCL] OpenID Foundation, "OpenID Connect RP-Initiated Logout 1.0 - draft 01", August 2020, <https://openid.net/specs/openid-connect-rpinitiated-1_0.html>.

[OIDCR] OpenID Foundation, "OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-registration-1_0.html>.

[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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[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>.

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <https://www.rfc-editor.org/info/rfc7009>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7480] Newton, A., Ellacott, B., and N. Kong, "HTTP Usage in the Registration Data Access Protocol (RDAP)", STD 95, RFC 7480, DOI 10.17487/RFC7480, March 2015, <https://www.rfc-editor.org/info/rfc7480>.

[RFC7481] Hollenbeck, S. and N. Kong, "Security Services for the Registration Data Access Protocol (RDAP)", STD 95, RFC 7481, DOI 10.17487/RFC7481, March 2015, <https://www.rfc-editor.org/info/rfc7481>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <https://www.rfc-editor.org/info/rfc7617>.

[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>.

[RFC8628] Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Authorization Grant", RFC 8628, DOI 10.17487/RFC8628, August 2019, <https://www.rfc-editor.org/info/rfc8628>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/info/rfc8693>.

[RFC9082] Hollenbeck, S. and A. Newton, "Registration Data Access Protocol (RDAP) Query Format", STD 95, RFC 9082, DOI 10.17487/RFC9082, June 2021, <https://www.rfc-editor.org/info/rfc9082>.

[RFC9083] Hollenbeck, S. and A. Newton, "JSON Responses for the Registration Data Access Protocol (RDAP)", STD 95, RFC 9083, DOI 10.17487/RFC9083, June 2021, <https://www.rfc-editor.org/info/rfc9083>.

12.2. Informative References

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[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>.

Appendix A. Change Log

00: Initial working group version ported from draft-hollenbeck-regext-rdap-openid-10.

01: Modified ID Token delivery approach to note proper use of an HTTP bearer authorization header.

02: Modified token delivery approach (Access Token is the bearer token) to note proper use of an HTTP bearer authorization header, fixing the change made in -01.

03: Updated OAuth 2.0 Device Authorization Grant description and reference due to publication of RFC 8628.

04: Updated OAuth 2.0 token exchange description and reference due to publication of RFC 8693. Corrected the RDAP conformance identifier to be registered with IANA.

05: Keepalive refresh.

06: Keepalive refresh.

07: Added "login_hint" description to Section 3.1.3.2. Added some text to Section 3.1.4.2 to note that "do not track" requires compliance with local regulations.

08: Rework of token management processing in Sections 4 and 5.

09: Updated RDAP specification references. Added text to describe both default and remote Authorization Server processing. Removed text that described passing of ID Tokens as query parameters.

10: Updated Section 3.1.3.1. Replaced token processing queries with "login", "session", and "logout" queries.

11: Replaced queries with "session/*" queries. Added description of "rdap" OAuth scope. Added implementation status information.

12: Updated data structure descriptions. Updated Section 8. Minor formatting changes due to a move to xml2rfc-v3 markup.

13: Added support for OP discovery via OP's Issuer Identifier. Modified the RDAP conformance text to use "roidc1", and added that value to extension path segments, data structures, and query parameters. Changed the "purpose" and "dnt" claims to "rdap_allowed_purposes" (making it an array) and "rdap_dnt_allowed". Added the "roidc1_qp" and "roidc1_dnt" query parameters. Changed the descriptions of "local" OPs to "default" OPs.

14: Fixed a few instances of "id" that were changed to "roidc1_id" and "session" that were changed to "roidc1_session". Added "implicitTokenRefreshSupported".

Author's Address

Scott Hollenbeck
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
United States of America

Email: shollenbeck@verisign.com
URI: http://www.verisignlabs.com/