

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: November 11, 2018

S. Hollenbeck  
Verisign Labs  
May 10, 2018

**Federated Authentication for the Registration Data Access Protocol  
(RDAP) using OpenID Connect  
draft-hollenbeck-regext-rdap-openid-08**

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 November 11, 2018.

Copyright Notice

Copyright (c) 2018 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction</a>	<a href="#">3</a>
<a href="#">1.1.</a>	<a href="#">Problem Statement</a>	<a href="#">3</a>
<a href="#">1.2.</a>	<a href="#">Proposal</a>	<a href="#">3</a>
<a href="#">2.</a>	<a href="#">Conventions Used in This Document</a>	<a href="#">4</a>
<a href="#">3.</a>	<a href="#">Federated Authentication for RDAP</a>	<a href="#">4</a>
<a href="#">3.1.</a>	<a href="#">RDAP and OpenID Connect</a>	<a href="#">4</a>
<a href="#">3.1.1.</a>	<a href="#">Terminology</a>	<a href="#">5</a>
<a href="#">3.1.2.</a>	<a href="#">Overview</a>	<a href="#">5</a>
<a href="#">3.1.3.</a>	<a href="#">RDAP Authentication and Authorization Steps</a>	<a href="#">6</a>
<a href="#">3.1.3.1.</a>	<a href="#">Provider Discovery</a>	<a href="#">6</a>
<a href="#">3.1.3.2.</a>	<a href="#">Authentication Request</a>	<a href="#">6</a>
<a href="#">3.1.3.3.</a>	<a href="#">End-User Authorization</a>	<a href="#">7</a>
<a href="#">3.1.3.4.</a>	<a href="#">Authorization Response and Validation</a>	<a href="#">7</a>
<a href="#">3.1.3.5.</a>	<a href="#">Token Processing</a>	<a href="#">7</a>
<a href="#">3.1.3.6.</a>	<a href="#">Delivery of User Information</a>	<a href="#">7</a>
<a href="#">3.1.4.</a>	<a href="#">Specialized Claims for RDAP</a>	<a href="#">8</a>
<a href="#">3.1.4.1.</a>	<a href="#">Stated Purpose</a>	<a href="#">8</a>
<a href="#">3.1.4.2.</a>	<a href="#">Do Not Track</a>	<a href="#">9</a>
<a href="#">4.</a>	<a href="#">Protocol Parameters</a>	<a href="#">9</a>
<a href="#">4.1.</a>	<a href="#">Client Authentication Request and Response</a>	<a href="#">10</a>
<a href="#">4.2.</a>	<a href="#">Token Request and Response</a>	<a href="#">10</a>
<a href="#">4.3.</a>	<a href="#">Token Refresh and Revocation</a>	<a href="#">11</a>
<a href="#">4.4.</a>	<a href="#">Parameter Processing</a>	<a href="#">14</a>
<a href="#">4.5.</a>	<a href="#">RDAP Conformance</a>	<a href="#">15</a>
<a href="#">5.</a>	<a href="#">Clients with Limited User Interfaces</a>	<a href="#">15</a>
<a href="#">5.1.</a>	<a href="#">OAuth 2.0 Device Flow</a>	<a href="#">15</a>
<a href="#">5.2.</a>	<a href="#">Manual Token Management</a>	<a href="#">16</a>
<a href="#">6.</a>	<a href="#">IANA Considerations</a>	<a href="#">17</a>
<a href="#">6.1.</a>	<a href="#">RDAP Extensions Registry</a>	<a href="#">17</a>
<a href="#">6.2.</a>	<a href="#">JSON Web Token Claims Registry</a>	<a href="#">17</a>
<a href="#">6.3.</a>	<a href="#">RDAP Query Purpose Registry</a>	<a href="#">17</a>
<a href="#">7.</a>	<a href="#">Implementation Status</a>	<a href="#">20</a>
<a href="#">7.1.</a>	<a href="#">Verisign Labs</a>	<a href="#">20</a>
<a href="#">7.2.</a>	<a href="#">Viagenie</a>	<a href="#">21</a>
<a href="#">8.</a>	<a href="#">Security Considerations</a>	<a href="#">21</a>
<a href="#">8.1.</a>	<a href="#">Authentication and Access Control</a>	<a href="#">22</a>



<a href="#">9.</a>	<a href="#">Acknowledgements</a>	<a href="#">22</a>
<a href="#">10.</a>	<a href="#">References</a>	<a href="#">22</a>
<a href="#">10.1.</a>	<a href="#">Normative References</a>	<a href="#">22</a>
<a href="#">10.2.</a>	<a href="#">Informative References</a>	<a href="#">24</a>
<a href="#">10.3.</a>	<a href="#">URIs</a>	<a href="#">24</a>
<a href="#">Appendix A.</a>	<a href="#">Change Log</a>	<a href="#">25</a>
	<a href="#">Author's Address</a>	<a href="#">25</a>

## [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" [[RFC7482](#)], and "JSON Responses for the Registration Data Access Protocol (RDAP)" [[RFC7483](#)]. [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 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 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-based client using OpenID Connect requires completion of the following steps:

1. An RDAP client (acting as an OpenID End-User) sends an HTTP (or HTTPS) query containing OAuth 2.0 request parameters to an RDAP server.
2. The RDAP server (acting as an OpenID Relying Party (RP)) prepares an Authentication Request containing the desired request parameters.
3. The RDAP server sends the RDAP client and Authentication Request to an Authorization Server operated by an OpenID Provider (OP) using an HTTP redirect.
4. The Authorization Server authenticates the RDAP Client.
5. The Authorization Server obtains RDAP Client consent/authorization.
6. The Authorization Server sends the RDAP Client back to the RDAP server with an Authorization Code using an HTTP redirect.
7. The RDAP server requests a response using the Authorization Code at the Token Endpoint.
8. The RDAP server receives a response that contains an ID Token and Access Token in the response body.



9. The RDAP server validates the ID Token and retrieves the RDAP client's Subject Identifier.

The RDAP server can then make identification, authorization, and access control decisions based on local policies, the ID Token received from the OP, and the received Claims. 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 MUST possess an identifier (an OpenID) issued by an OP to use OpenID Connect with RDAP. The OpenID Foundation maintains a list of OPs on its web site [[1](#)]. Additional OPs are almost certainly needed to fully realize the potential for federated authentication with RDAP because RDAP has authorization and access control requirements that go beyond the end-user authentication requirements of a typical web site.

OpenID Connect requires RPs to register with OPs to use OpenID Connect services for an End-User. That process is described by the "OpenID Connect Dynamic Client Registration" protocol [[OIDCR](#)].

#### **[3.1.3.1.](#) Provider Discovery**

An RDAP server/RP needs to receive an identifier from an End-User that can be used to discover the End-User's OP. That process is required and is documented in the "OpenID Connect Discovery" protocol [[OIDCD](#)].

#### **[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](#)]. 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 specified 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.

##### **3.1.4.1. Stated Purpose**

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 "purpose" claim.

The "purpose" claim identifies the purpose for which access to a protected resource is being requested. Use of the "purpose" claim is OPTIONAL; processing of this claim is subject to server acceptance of the purpose 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 "purpose" value is a case-sensitive string containing a StringOrURI value as specified in [Section 2](#) of the JSON Web Token (JWT) specification ([[RFC7519](#)]). An example:

```
{"purpose" : "domainNameControl"}
```

Purpose values are themselves 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 6.3](#) are taken from the final report [2] 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" ("dnt") 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 "dnt" 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 "dnt" claim is present, and the value of the claim is "true".

The "dnt" value is a case-sensitive string containing a JSON boolean value. An example:

```
{"dnt" : "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. A query parameter to request authentication for a specific end-user identity.
2. A path segment to request an ID Token and an Access Token for a specific end-user identity.
3. A query parameter to deliver an ID Token and an Access Token for use with an RDAP query.



#### **4.1. Client Authentication Request and Response**

Client authentication is requested by adding a query component to an RDAP request URI using the syntax described in Section 3.4 of [RFC 3986](#) [[RFC3986](#)]. The query used to request client authentication is represented as a "key=value" pair using a key value of "id" and a value component that contains the client identifier issued by an OP. An example:

`https://example.com/rdap/domain/example.com?id=user.idp.example`

The response to an authenticated query MUST use the response structures specified in [RFC 7483](#) [[RFC7483](#)]. Information that the end-user is not authorized to receive MUST be omitted from the response.

#### **4.2. Token Request and Response**

Clients MAY send a request to an RDAP server to authenticate an end-user and return an ID Token and an Access Token from an OP that can be then be passed to the RP/RDAP server to authenticate and process subsequent queries. Identity provider authentication is requested using a "tokens" path segment and a query parameter with key value of "id" and a value component that contains the client identifier issued by an OP. An example:

`https://example.com/rdap/tokens?id=user.idp.example`

In addition to any core RDAP response elements, the response to this query MUST contain four name-value pairs, in any order, representing the returned ID Token and Access Token. The ID Token is represented using a key value of "id\_token". The Access Token is represented using a key value of "access\_token". The access token type is represented using a key value of "token\_type" and a value of "bearer" as described in Sections [4.2.2](#) and [7.1](#) of [RFC 6749](#) [[RFC6749](#)]. The lifetime of the access token is represented using a key value of "expires\_in" and a numerical value that describes the lifetime in seconds of the access token as described in [Section 4.2.2 of RFC 6749](#) [[RFC6749](#)]. The token values returned in the RDAP server response MUST be Base64url encoded as described in RFCs 7515 [[RFC7515](#)] and 7519 [[RFC7519](#)].



An example (the encoded tokens have been abbreviated for clarity):

```
{
  "access_token" : "eyJ0...NiJ9",
  "id_token" : "eyJ0...EjXk",
  "token_type" : "bearer",
  "expires_in" : "3600"
}
```

Figure 1

An RDAP server that processes this type of query MUST determine if the identifier is associated with an OP that is recognized and supported by the server. Servers MUST reject queries that include an identifier associated with an unsupported OP with an HTTP 501 (Not Implemented) response. An RDAP server that receives a query containing an identifier associated with a recognized OP MUST perform the steps required to authenticate the user with the OP using a browser or browser-like client and return encoded tokens to the client. Note that tokens are typically valid for a limited period of time and new tokens will be required when an existing token's validity period has expired.

The tokens can then be passed to the server for use with an RDAP query using a query parameter with key values of "id\_token" and "access\_token" and values that represent the encoded tokens. An example (the encoded tokens have been abbreviated and the URI split across multiple lines for clarity):

```
https://example.com/rdap/domain/example.com
?id_token=eyJ0...EjXk
&access_token=eyJ0...NiJ9
```

The response to an authenticated query MUST use the response structures specified in [RFC 7483](#) [[RFC7483](#)]. Information that the end-user is not authorized to receive MUST be omitted from the response.

#### **[4.3.](#) Token Refresh and Revocation**

An access token can be refreshed as described in [Section 12](#) of the OpenID Connect Core protocol [[OIDCC](#)] and [Section 6](#) of OAuth 2.0 [[RFC6749](#)]. Clients can take advantage of this functionality if it is supported by the OP and accepted by the RDAP server.

A refresh token is requested using a "tokens" path segment and two query parameters. The first query parameter includes a key value of "id" and a value component that contains the client identifier issued



by an OP. The second query parameter includes a key value of "refresh" and a value component of "true". A value component of "false" MUST be processed to return a result that is consistent with not including a "refresh" parameter at all as described in [Section 4.2](#). An example using "refresh=true":

```
https://example.com/rdap/tokens?id=user.idp.example
&refresh=true
```

The response to this query MUST contain all of the response elements described in [Section 4.2](#). In addition, the response MUST contain a name-value pair that represents a refresh token. The name-value pair includes a key value of "refresh\_token" and a Base64url-encoded value that represents the refresh token.

Example refresh token request response (the encoded tokens have been abbreviated for clarity):

```
{
  "access_token" : "eyJ0...NiJ9",
  "id_token" : "eyJ0...EjXk",
  "token_type" : "bearer",
  "expires_in" : "3600",
  "refresh_token" : "eyJ0...c8da"
}
```

Figure 2

Once acquired, a refresh token can be used to refresh an access token. An access token is refreshed using a "tokens" path segment and two query parameters. The first query parameter includes a key value of "id" and a value component that contains the client identifier issued by an OP. The second query parameter includes a key value of "refresh\_token" and a Base64url-encoded value that represents the refresh token. An example:

```
https://example.com/rdap/tokens?id=user.idp.example
&refresh_token=eyJ0...f3jE
```

In addition to any core RDAP response elements, the response to this query MUST contain four name-value pairs, in any order, representing a returned Refresh Token and Access Token. The Refresh Token is represented using a key value of "refresh\_token". The Access Token is represented using a key value of "access\_token". The access token type is represented using a key value of "token\_type" and a value of "bearer" as described in Sections 4.2.2 and 7.1 of [RFC 6749](#) [RFC6749]. The lifetime of the access token is represented using a key value of "expires\_in" and a numerical value that describes the



lifetime in seconds of the access token as described in [Section 4.2.2 of RFC 6749](#) [RFC6749]. The token values returned in the RDAP server response MUST be Base64url encoded as described in RFCs 7515 [RFC7515] and 7519 [RFC7519].

Example access token refresh response (the encoded tokens have been abbreviated for clarity):

```
{
  "access_token" : "0dac...13b0",
  "refresh_token" : "f735...d30c",
  "token_type" : "bearer",
  "expires_in" : "3600"
}
```

Figure 3

Access and refresh tokens can be revoked as described in [RFC 7009](#) [RFC7009] by sending a request to an RDAP server that contains a "tokens/revoke" path segment and two query parameters. The first query parameter includes a key value of "id" and a value component that contains the client identifier issued by an OP. The second query parameter includes a key value of "token" and a Base64url-encoded value that represents either the current refresh token or the associated access token. An example:

```
https://example.com/rdap/tokens/revoke?id=user.idp.example
&token=f735...d30c
```

Note that this command will revoke both access and refresh tokens at the same time. In addition to any core RDAP response elements, the response to this query MUST contain a description of the result of processing the revocation request within the RDAP "notices" data structure.

Example token revocation success:

```
"notices" :
[
  {
    "title" : "Token Revocation Result",
    "description" : "Token revocation succeeded.",
  }
],
"lang" : "en-US"
```

Figure 4



Example token revocation failure:

```
"notices" :  
[  
  {  
    "title" : "Token Revocation Result",  
    "description" : "Token revocation failed.",  
  }  
],  
"errorCode" : 400,  
"lang" : "en-US"
```

Figure 5

#### **4.4. Parameter Processing**

Unrecognized query parameters MUST be ignored. An RDAP request that does not include an "id" query component MUST be processed as an unauthenticated query. An RDAP server that processes an authenticated query MUST determine if the identifier is associated with an OP that is recognized and supported by the server. Servers MUST reject queries that include an identifier associated with an unsupported OP with an HTTP 501 (Not Implemented) response. An RDAP server that receives a query containing an identifier 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.

An RDAP server that receives a query containing tokens associated with a recognized OP and authenticated end user MUST process the query and return an RDAP response that is appropriate for the end user's level of authorization and access. Errors based on processing either the ID Token or the Access Token MUST be signaled with an appropriate HTTP status code as described in [Section 3.1 of RFC 6750 \[RFC6750\]](#).

On receiving a query containing tokens, the RDAP server MUST validate the ID Token. It can do this independently of the OP, because the ID Token is a JWT that contains all the data necessary for validation. The Access Token, however, is an opaque value, and can only be validated by sending a request using it to the UserInfo Endpoint and confirming that a successful response is received. This is different from the OpenID Connect Authorization Code and Implicit flows, where the Access Token can be validated against the at\_hash claim from the ID Token. With a query containing tokens, the Access Token might not validate against the at\_hash claim because the Access Token may have been refreshed since the ID Token was issued.



An RDAP server that processes requests without needing the UserInfo claims does not need to retrieve the claims merely in order to validate the Access Token. Similarly, an RDAP server that has cached the UserInfo claims for an end user, in accordance with the HTTP headers of a previous UserInfo Endpoint response, does not need to retrieve those claims again in order to revalidate the Access Token.

#### **4.5. RDAP Conformance**

RDAP responses that contain values described in this document MUST indicate conformance with this specification by including an rdapConformance ([RFC7483]) value of "rdap\_openidc\_level\_0". The information needed to register this value in the RDAP Extensions Registry is described in [Section 6.1](#).

Example rdapConformance structure with extension specified:

```
"rdapConformance" :  
[  
  "rdap_level_0",  
  "rdap_openidc_level_0"  
]
```

Figure 6

### **5. Clients with Limited User Interfaces**

The flow described in [Section 3.1.3](#) requires a client to interact with a server using a web browser. 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 [3] or wget [4]. There are multiple ways to address this limitation using a web browser on a second device. Two are described here.

#### **5.1. OAuth 2.0 Device Flow**

The "OAuth 2.0 Device Flow for Browserless and Input Constrained Devices" [I-D.ietf-oauth-device-flow] provides one 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 a client 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 constrained device.



## 5.2. Manual Token Management

A second method of requesting user authorization from a constrained device is possible by producing and managing tokens manually as follows:

1. Authenticate with the OP as described in [Section 4.2](#) using a browser or browser-like client.
2. Store the returned ID Token and Access Token locally.
3. Send a request to the content provider/RP along with the ID Token and Access Token received from the OP.

The Access Token MAY be passed to the RP in an HTTP "Authorization" header [[RFC7235](#)] or as a query parameter. The Access Token MUST be specified using the "Bearer" authentication scheme [[RFC6750](#)] if it is passed in an "Authorization" header. The ID Token MUST be passed to the RP as a query parameter.

Here are two examples using the curl and wget utilities. Start by authenticating with the OP:

```
https://example.com/rdap/tokens?id=user.idp.example
```

Save the token information and pass it to the RP along with the URI representing the RDAP query. Using curl (encoded tokens have been abbreviated for clarity:

```
curl -H "Authorization: Bearer eyJ0...NiJ9"\
-k https://example.com/rdap/domain/example.com\
?id_token=eyJ0...EjXk
```

```
curl -k https://example.com/rdap/domain/example.com\
?id_token=eyJ0...EjXk&access_token=eyJ0...NiJ9
```

Using wget:

```
wget --header="Authorization: Bearer eyJ0...NiJ9"\
https://example.com/rdap/domain/example.com\
?id_token=eyJ0...EjXk
```

```
wget https://example.com/rdap/domain/example.com\
?id_token=eyJ0...EjXk&access_token=eyJ0...NiJ9
```

Refresh tokens can be useful to automated or command line clients who wish to continue a session without explicitly re-authenticating an end user. See [Section 4.3](#) for more information.



## **6. IANA Considerations**

### **6.1. RDAP Extensions Registry**

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

Extension identifier: rdap\_openidc  
Registry operator: Any  
Published specification: This document.  
Contact: IESG <iesg@ietf.org>  
Intended usage: This extension includes response information required for federated authentication using OpenID Connect.

### **6.2. JSON Web Token Claims Registry**

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

Claim Name: "purpose"  
Claim Description: This claim describes the stated purpose for submitting a request to access a protected RDAP resource.  
Change Controller: IESG  
Specification Document(s): [Section 3.1.4.1](#) of this document.

Claim Name: "dnt"  
Claim Description: This claim contains a boolean value that describes an End-User's "do not track" preference for identity tracking, logging, or recording when accessing a protected RDAP resource.  
Change Controller: IESG  
Specification Document(s): [Section 3.1.4.2](#) of this document.

### **6.3. RDAP Query Purpose Registry**

IANA is requested to create a new protocol registry to manage RDAP query purpose values. This registry should appear under its own heading on IANA's protocol listings, using the same title as the name of the registry. The information to be registered and the procedures to be followed in populating the registry are described in [Section 3.1.4.1](#).

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

Section at <http://www.iana.org/protocols>:



Registry Title: Registration Data Access Protocol (RDAP) Query Purpose Values

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

Registration Procedure: Specification Required

Reference: This draft

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: academicPublicInterestDNSRRResearch

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-----

## **7. 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".

### **7.1. 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.

Contact Information: Scott Hollenbeck, [shollenbeck@verisign.com](mailto:shollenbeck@verisign.com)

## **7.2. 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.

Contact Information: Marc Blanchet, [marc.blanchet@viagenie.ca](mailto:marc.blanchet@viagenie.ca)

## **8. 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. Security services for ID Tokens and Access



Tokens (with references to the JWT specification) are described in the OpenID Connect Core protocol.

### **8.1. 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.

## **9. Acknowledgements**

The author would like to acknowledge the following individuals for their contributions to the development of this document: Tom Harrison, Russ Housley, Rhys Smith, Jaromir Talir, and Alessandro Vesely. In addition, the Verisign Registry Services Lab development team of 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.

## **10. References**

### **10.1. Normative References**

- [I-D.ietf-oauth-device-flow]  
Denniss, W., Bradley, J., Jones, M., and H. Tschofenig,  
"OAuth 2.0 Device Flow for Browserless and Input  
Constrained Devices", [draft-ietf-oauth-device-flow-09](#)  
(work in progress), April 2018.
- [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](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](http://openid.net/specs/openid-connect-discovery-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](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>>.
- [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>>.
- [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>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", [RFC 7235](#), DOI 10.17487/RFC7235, June 2014, <<https://www.rfc-editor.org/info/rfc7235>>.
- [RFC7480] Newton, A., Ellacott, B., and N. Kong, "HTTP Usage in the Registration Data Access Protocol (RDAP)", [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)", [RFC 7481](#), DOI 10.17487/RFC7481, March 2015, <<https://www.rfc-editor.org/info/rfc7481>>.
- [RFC7482] Newton, A. and S. Hollenbeck, "Registration Data Access Protocol (RDAP) Query Format", [RFC 7482](#), DOI 10.17487/RFC7482, March 2015, <<https://www.rfc-editor.org/info/rfc7482>>.
- [RFC7483] Newton, A. and S. Hollenbeck, "JSON Responses for the Registration Data Access Protocol (RDAP)", [RFC 7483](#), DOI 10.17487/RFC7483, March 2015, <<https://www.rfc-editor.org/info/rfc7483>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", [RFC 7515](#), DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [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>>.
- [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>>.

## **[10.2.](#) 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>>.

## **[10.3.](#) URIs**

- [1] <http://openid.net/get-an-openid/>
- [2] <https://www.icann.org/en/system/files/files/final-report-06jun14-en.pdf>
- [3] <http://curl.haxx.se/>
- [4] <https://www.gnu.org/software/wget/>



## **Appendix A. Change Log**

- 00: Initial version.
- 01: Updated flow description ([Section 3.1.2](#)) and description of the registration process ([Section 3.1.3](#)). Thanks to Jaromir Talir.
- 02: Updated flow description.
- 03: Added description of query parameters and non-browser clients. Updated security considerations to note issues associated with access control.
- 04: Updated references for JSON Web Token, OpenID Connect Core, and OpenID Connect Discovery. Added acknowledgement to the Verisign Labs developers. Changed intended status to Standards Track. Added text to describe protocol parameters and processing. Other minor edits.
- 05: Added examples for curl and wget. Added a reference to [RFC 7235](#).
- 00: Changed WG reference in file name from weirds to regext. Described support for refresh tokens. Editorial updates. Corrected several examples. Added registry of purpose values.
- 01: Added Implementation Status section ([Section 7](#)).
- 02: Updated section [Section 4.4](#) to clarify ID Token validation requirements.
- 03: Keepalive refresh.
- 04: Added rdap\_conformance.
- 05: Added "do not track" claim.
- 06: Added text describing the OAuth device flow and modified other text associated with UI-limited devices. Updated [BCP 14](#) text.
- 07: Removed outdated [BCP 14](#) text.
- 08: Added Viagenie implementation description.

### Author's Address

Scott Hollenbeck  
Verisign Labs  
12061 Bluemont Way  
Reston, VA 20190  
USA

Email: [shollenbeck@verisign.com](mailto:shollenbeck@verisign.com)  
URI: <http://www.verisignlabs.com/>

