Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: August 5, 2019

T. Harrison
G. Michaelson
APNIC
A. Newton
ARIN

February 1, 2019

RDAP Mirroring Protocol (RMP)
draft-harrison-regext-rdap-mirroring-00

Abstract

The Registration Data Access Protocol (RDAP) is used by Regional Internet Registries (RIRs) and Domain Name Registries (DNRs) to provide access to their resource registration information. While most clients can retrieve the information they need on an ad hoc basis from the public services maintained by each registry, there are instances where local copies of those remote data sources need to be maintained, for various reasons (e.g. performance requirements). This document defines a protocol for transferring bulk RDAP response data and for keeping a local copy of that data up to date.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Registration Data Access Protocol (RDAP) [RFC7480] is used by Regional Internet Registries (RIRs) and Domain Name Registries (DNRs) to provide access to their resource registration information. For a client, this typically involves following the bootstrap process [RFC7484] to determine the base URL for the query, constructing an RDAP request, sending it, and then processing the response.

This mode of operation is appropriate for many use cases. However, some clients may need local access to the whole data set:

their performance requirements may be such that the time required for sending/receiving HTTP requests to arbitrary remote servers is not acceptable;

they may be conducting analysis of the data set as a whole; or

they may be providing access to the data set in their own right, as an alternative to redirecting to the authoritative source for the data.

This document defines a protocol that can be used by a client to retrieve a local copy of a remote RDAP data set, as well as to maintain that local copy as further remote updates occur.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. RDAP Mirroring Protocol Implementation

2.1. Overview

A registry that wants to make use of this protocol publishes an Update Notification File to a specified URL. That file in turn links to a Snapshot File and a series of Delta Files. The Snapshot File contains all of the registry's RDAP state as at a given point in time. The Delta Files contain changes that have been made to the registry's RDAP state since the Snapshot File was generated.

As further changes are made to the registry's RDAP state, the registry publishes new Delta Files, amends the Update Notification File to include links to the new Delta Files, and then republishes the Update Notification File. Periodically, the registry regenerates and republishes the Snapshot File, which in turn allows for older Delta Files to be removed from the Update Notification File.

All files in the protocol are signed by the server using JSON Web Signature (JWS) [RFC7515]. The JSON Web Key (JWK) [RFC7517] used by the server to sign the files is distributed out-of-band.

A client that wants to make use of this protocol needs to learn the server's Update Notification File URL and JSON Web Key out-of-band. Once these are known, the client retrieves the Update Notification File, validates its signature, and follows the links in it to retrieve the Snapshot File and Delta Files. It validates the signatures on these files, and then uses them to initialize its local state. It then records the serial number of the most-recently-issued Delta File, or of the Snapshot File if no Delta Files are present.

The client will then periodically retrieve the Update Notification File, determine the Delta Files that have been added since it was last retrieved by the client, retrieve those Delta Files, and update its local state accordingly.

A server may opt not to publish a Snapshot File in the Update Notification File. Such servers will only publish Delta Files in their Update Notification File, and must distribute the Snapshot File out-of-band.

2.2. File Definitions

2.2.1. Update Notification File

Example Update Notification File:

```
{
    "version": 1,
    "refresh": 3600,
    "snapshot": { "uri": "https://example.com/1/snapshot.json",
                  "serial": 1 },
    "deltas": [
        { "uri": "https://example.com/2/delta.json",
          "serial": 2 },
        { "uri": "https://example.com/3/delta.json",
          "serial": 3 },
    ]
}
```

The following validation rules MUST be observed when creating or parsing Update Notification Files:

Update Notification Files MUST be well-formed JSON [RFC8259].

The "version" attribute in the root element MUST be present, with a value of "1".

A "refresh" attribute MAY be present. If it is present, it is an integer representing how long the client should wait (in seconds) after retrieving the Update Notification File before attempting to retrieve it again.

A "snapshot" attribute containing a link to a Snapshot File MAY be present.

A "deltas" attribute containing an array of links to Delta Files MUST be present. If no Delta Files have been published by the server, this array will be empty.

The Delta File entries in the "deltas" attribute MUST be in serial number order, and the serial numbers MUST form a contiguous sequence.

If a Snapshot File is included, its serial number MUST either be equal to that of one of the Delta Files, or one less than the smallest Delta File serial number.

2.2.2. Snapshot File

Example Snapshot File:

```
{
    "version": 1,
    "serial": 3,
    "defaults": { "port43": "whois.example.com",
                  ... },
    "objects": [
        { "id": "https://example.org/I1",
          "object": { "rdapConformance": [ "rdap_level_0" ],
                      "objectClassName": "ip network",
                      ... } },
        { "id": "https://example.org/D2",
          "object": { "rdapConformance": [ "rdap_level_0" ],
                      "objectClassName": "domain",
                      ... } },
        ...
    ]
}
```

The following validation rules MUST be observed when creating or parsing Snapshot Files:

Snapshot Files MUST be well-formed JSON [RFC8259].

The "version" attribute in the root element MUST be present, with a value of "1".

The "serial" attribute in the root element MUST be present, with a value that is an unsigned 32-bit integer.

An "objects" attribute containing an array of RDAP object (see [RFC7483]) and identifier pairs MUST be present. If no RDAP objects have been published by the server, this array will be empty.

A "defaults" attribute MAY be present. For each object received from the server, the client should treat the object as also having each attribute from the "defaults" attribute, except where the object already contains an attribute with that name. This applies to objects received both before and after the Snapshot File is processed.

2.2.3. Delta File

Example Delta File:

```
{
    "version": 1,
    "serial": 4,
    "defaults": { "port43": "whois-2.example.org",
                  ... },
    "removed_objects": [ "1" ],
    "added_or_updated_objects": [
        { "id": "https://example.org/I3",
          "object": { "rdapConformance": [ "rdap_level_0" ],
                      "objectClassName": "ip network",
                      ... } },
        { "id": "https://example.org/D4",
          "object": { "rdapConformance": [ "rdap_level_0" ],
                      "objectClassName": "domain",
                      ... } },
        ...
    ]
}
```

The following validation rules MUST be observed when creating or parsing Delta Files:

Snapshot Files MUST be well-formed JSON [RFC8259].

The "version" attribute in the root element MUST be present, with a value of "1".

The "serial" attribute in the root element MUST be present, with a value that is an unsigned 32-bit integer.

A "defaults" attribute MAY be present. For each object received from the server, the client should treat the object as also having each attribute from the "defaults" attribute, except where the object already contains an attribute with that name. This applies to objects received both before and after the Delta File is processed.

A "removed_objects" attribute containing an array of RDAP object identifiers MUST be present. If no RDAP objects have been removed since the previous Delta File was generated by the server, this array will be empty.

An "added_or_updated_objects" attribute containing an array of RDAP object (see [RFC7483]) and identifier pairs MUST be present. If no RDAP objects have been added or updated since the previous Delta File was generated by the server, this array will be empty.

2.3. RDAP Objects

The base RDAP object definitions from [RFC7483] do not contain any mandatory attributes. For the purposes of this protocol, each RDAP object MUST include an "rdapConformance" attribute, so that a client can determine whether the object is one that it is able to process.

Each RDAP object is paired with an identifier (the "id" attribute in the parent object). The "id" attribute value MUST be a URI ([RFC3986]). Its value uniquely identifies an RDAP object within the data set, so as to support internal linking and for subsequent removal of the object by a Delta File.

In each RDAP object, each link to an RDAP object that is a member of the data set for which the server is providing mirroring MUST have as the value of its "href" attribute the identifier for that object (i.e. the value of the "id" attribute from the target's parent object). This includes self-references (i.e. links with the "self" relation type).

If a server includes a link object with the "self" relation type with each of its RDAP objects, then using the value of the "href" attribute of that link object as the identifier for each RDAP object is RECOMMENDED.

For example, a Delta File containing an entity object, along with an IP network object that links to it:

```
   {
       "version": 1,
       "serial": 5,
       "removed_objects": [],
       "added_or_updated_objects": [
           { "id": "https://example.org/E5",
             "object": { "rdapConformance": [ "rdap_level_0" ],
                         "objectClassName": "entity",
                         "handle": "E5",
                         "links": [
                             { "rel": "self",
                               "href": "https://example.org/E5",
                               ... }
                         ],
                         ... } },
           { "id": "https://example.org/I6",
             "object": { "rdapConformance": [ "rdap_level_0" ],
                         "objectClassName": "ip network",
                         "links": [
                             { "rel": "self",
                               "href": "https://example.org/I6",
                               ... }
                         ],
                         "entities": [
                             { "handle": "E5",
                               "links": [
                                   { "rel": "self",
                                     "href": "https://example.org/E5",
                                     ... }
                               ],
                               ... }
                         ],
                         ... } }
           ...
       ]
   }
```

A server MAY omit from an object data that it returns as part of its corresponding public service response, when that data can be determined by reference to another object in the data set. In such cases, the server MUST include a "links" attribute containing a link object with a "self" relation, so that the target object can be resolved by the client.

2.4. Serial Numbers

Serial numbers in the files defined in this protocol are unsigned 32-bit integers. For the purposes of this protocol, the serial number arithmetic defined in [RFC1982] applies.

2.5. Server Use

2.5.1. Initialization

If a server is publishing a Snapshot File via the Update Notification File, then initialization is like so:

> generate an initial Snapshot File, with a serial number selected by the server;
>
> sign the Snapshot File using JWS, and publish the result using JWS Compact Serialization at a URL that is unique to this serial number;
>
> generate an initial Update Notification File, with a serial number equal to that of the Snapshot File, and containing a link to the Snapshot File; and
>
> sign the Update Notification File using JWS, and publish the result using JWS Compact Serialization.

To avoid doubt, any RDAP object that is part of the data set for which the server is providing mirroring, as well as being the target of a link contained within another RDAP object, MUST be present within the Snapshot File.

If a server is not publishing a Snapshot File via the Update Notification File, then initialization is like so:

> generate an initial Snapshot File, with a serial number selected by the server;
>
> sign the Snapshot File using JWS, and distribute the result to clients out-of-band using JWS Compact Serialization;
>
> generate an initial Update Notification File, containing no Snapshot File link or Delta File links, with a serial number equal to that of the Snapshot File (i.e. the one that will be distributed out-of-band); and
>
> sign the Update Notification File using JWS, and publish the result using JWS Compact Serialization.

2.5.2. Publishing Updates

The server periodically publishes changes that have been made to its RDAP state as Delta Files. The timing and frequency of publication is a local policy matter for the server. The process is like so:

if a Delta File has been generated previously: generate a new Delta File, containing the changes that have been made to the RDAP state since the last Delta File was generated, with a serial number that is one greater than the serial number of the last Delta File;

if no Delta File has been generated previously: generate a new Delta File, containing the changes that have been made to the RDAP state since the last Snapshot File was generated, with a serial number that is one greater than the serial number of the last Snapshot File;

sign the Delta File using JWS, and publish the result using JWS Compact Serialization at a URL that is unique to its serial number;

take the currently-published Update Notification File, increment its serial number, add a link to the new Delta File, and optionally perform the steps described in the "Consolidation" section below; and

sign the Update Notification File using JWS, and publish the result using JWS Compact Serialization.

Delta Files MUST NOT remove an RDAP object that would cause a relative reference link within the client's local state to become unresolvable.

2.5.3. Consolidation

On publishing an update, the server may optionally consolidate the Snapshot File and Delta Files that it is publishing. The process is like so:

if the server is publishing a Snapshot File: generate a new Snapshot File based on the server's current state with a serial number equal to that of the new Delta File, publish the new Snapshot File, and replace the link in the Update Notification File to the previous Snapshot File with a link to the new Snapshot File; and remove Delta Files from the Update Notification File that have become stale.

Whether a given Delta File is ’stale’ is a local policy matter for the server.

2.6. Client Use

2.6.1. Processing the Update Notification File

2.6.1.1. Initial

The client downloads the signed Update Notification File using the URL provided by the server (out-of-band) and validates the signature against the server’s JWK.

The client validates the signature of the Snapshot File against the server’s JWK, and then uses that file to initialize its local state by adding all of the objects from the "objects" attribute. It then records the serial number of the Snapshot File. The signed Snapshot File is either accessible from the Update Notification File, or made available to the client out-of-band.

The client then processes each Delta File from the Update Notification File in order, from the Delta File with a serial number one greater than the client’s recorded serial number, through to the Delta File with the largest serial number, in order to update its local state.

Processing a Delta File involves three steps:

verify the signature against the server’s JWK;

for each entry in the "removed_objects" attribute, remove from the local state any object with an "id" attribute value equal to the entry;

for each entry in the "added_or_updated_objects" attribute: if an object with the given values for the "id" attribute exists in the local state, then replace that object with the new object; otherwise, add the new object to the local state.

Once this is complete, the client records the serial number of the last Delta File that it processed.

2.6.1.2. Subsequent

The client downloads the signed Update Notification File using the URL provided by the server (out-of-band), and validates the signature against the server's JWK. If the frequency with which the client should do this has been suggested by the server via the "refresh" attribute, the client SHOULD honor that suggestion. If the "refresh" attribute is not present, retrieval frequency is a local policy matter for the client.

The client then processes each Delta File from the Update Notification File in order, from the Delta File with a serial number one greater than that which has been recorded, through to the Delta File with the largest serial number. Processing of the Delta Files is otherwise as per the instructions for initial processing.

If the Update Notification File retrieved by the client does not contain a Delta File with a serial number one greater than that which has been recorded, the client MUST delete all of its local state and reinitialize itself. If the Update Notification File contains a Snapshot File, then that Snapshot File can be used for reinitialization. If it does not, then a new Snapshot File must be located out-of-band.

3. Operational Considerations

A server may omit previously-published Delta Files from its Update Notification File as a matter of local policy. If a server is publishing its Snapshot Files out-of-band, then omitting a Delta File that a client needs will result in the client needing to perform an out-of-band action in order to reinitialize its state. Even if a server is linking to its Snapshot Files from the Update Notification File, reinitialization may be an expensive operation for a client. Servers should consider adopting local policy that limits the chance of reinitialization happening: for example, by using the "refresh" attribute value in the Update Notification File.

4. Security Considerations

[RFC7481] describes security requirements and considerations for RDAP generally. Those requirements and considerations also apply to the use of this protocol.

This protocol requires the use of JWS ([RFC7515]) and JWK ([RFC7517]), which in turn refer to JSON Web Algorithms (JWA) ([RFC7518]). Implementations MUST support ES256 as defined in JWA ([RFC7518], section 3.4) for signing and validating files in this protocol. Implementations MAY support other algorithms from the

"JSON Web Signature and Encryption Algorithms" registry created by [RFC7518].

5. Acknowledgements

This protocol is largely a repurposing of the RPKI Repository Delta Protocol (RRDP) [RFC8182] for RDAP. Much of the terminology (e.g. Update Notification File, Snapshot File, Delta File) is taken from that document, and the structure is also quite similar.

Experience with the Near Real Time Mirroring (NRTM) [NRTM] protocol, which serves a similar purpose for databases that are based on the Routing Policy Specification Language (RPSL) [RFC2622], helped to inform this corresponding effort in RDAP.

6. IANA Considerations

TBD

7. References

7.1. Normative References

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, DOI 10.17487/RFC1982, August 1996, <https://www.rfc-editor.org/info/rfc1982>.

[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>.

[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>.

[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>.

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

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

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

7.2. Informative References

[NRTM] "Near Real Time Mirroring", December 2010, <https://www.ripe.net/manage-ips-and-asns/db/support/documentation/mirroring>.

[RFC2622] Alaettinoglu, C., Villamizar, C., Gerich, E., Kessens, D., Meyer, D., Bates, T., Karrenberg, D., and M. Terpstra, "Routing Policy Specification Language (RPSL)", RFC 2622, DOI 10.17487/RFC2622, June 1999, <https://www.rfc-editor.org/info/rfc2622>.

[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>.

[RFC7484] Blanchet, M., "Finding the Authoritative Registration Data (RDAP) Service", RFC 7484, DOI 10.17487/RFC7484, March 2015, <https://www.rfc-editor.org/info/rfc7484>.

[RFC8182] Bruijnzeels, T., Muravskiy, O., Weber, B., and R. Austein, "The RPKI Repository Delta Protocol (RRDP)", RFC 8182, DOI 10.17487/RFC8182, July 2017, <https://www.rfc-editor.org/info/rfc8182>.

Authors' Addresses

Tom Harrison
Asia-Pacific Network Information Centre
6 Cordelia St
South Brisbane, QLD 4101
Australia

Email: tomh@apnic.net

George G. Michaelson
Asia-Pacific Network Information Centre
6 Cordelia St
South Brisbane, QLD 4101
Australia

Email: ggm@apnic.net

Andrew Lee Newton
American Registry for Internet Numbers
PO Box 232290
Centreville, VA 20120
United States of America

Email: andy@arin.net