Network Working Group
Internet-Draft
Intended status: Experimental
Expires: March 5, 2015

P. Hoffman
VPN Consortium

September 01, 2014

Representing DNS Messages in JSON
draft-hoffman-dns-in-json-01

Abstract

Some applications use DNS messages, or parts of DNS messages, as data. For example, a system that captures DNS queries and responses might want to be able to easily search those without having to decode the messages each time. Another example is a system that puts together DNS queries and responses from message parts. This document fully describes a standardized format for DNS message data in JSON.

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 http://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 March 5, 2015.

Table of Contents

1. Introduction

The DNS message format is defined in [RFC1035]. DNS queries and DNS responses have exactly the same structure. Many of the field names and data type names given in RFC 1035 are commonly used in discussions of DNS. For example, it is common to hear things like "the query had a QNAME of 'example.com'" or "the RDATA has a simple structure".

There are hundreds of data interchange formats for serializing structured data. Currently, JSON [RFC7159] is quite popular for many types of data, particularly data that has named sub-fields and optional parts.

This document uses JSON to describe DNS messages. It also defines how to describe a stream of DNS queries or a stream of DNS responses, and how to describe a paired DNS query and response; these are useful for logging on a DNS server.

1.1. Design of the Format

There are many ways to design a data format. This document uses a specific design methodology based on the DNS format.

- The format is based on JSON objects in order to allow a writer to include or exclude parts of the format at will. No object members are ever required.

- All values that are eight bits or shorter (even booleans) are represented by JSON integers.

- Many values that can have non-ASCII data in them have names that end in "*" and are stored in base16 encoding (hex with uppercase letters) defined in [RFC4648].

- Some values will be very long and implementations might care about the size of the records on the wire. Values that are expected to be long (the entire record, some resource records such as those with cryptographic keys, and so on) have names that end in "!" and are stored in base64url encoding defined in [RFC4648].

- Domain names apprear in many places, and there is a desire to show fields with domain names in a mostly-human-readable form. Names that end in "&" are domain names that have escaping for some characters, as described in Section 2.5.

- All field names used in RFC 1035 are used in this format as-is. Names not defined in RFC 1035 use "camel case" with the first letter lowercase.

- Because domain names that follow the host name rules are so common, there are additional members for domain names that are also host names and can be shown as unencoded strings. See [RFC1123] for the host name rules.

- The same data may be represented in multiple object members multiple times. For example, there is a member for the octets of the DNS message header, and there are members for each named part of the header. A message object can thus inadvertently have inconsistent data, such as a header member whose value does not match the value of the first bits in the entire message member.

- The design explicitly allows for the description of malformed DNS messages. This is important for systems that are logging messages seen on the wire, particularly messages that might be used as part of an attack. For example, an RR might have an RDLENGTH of 4 but an RDATA whose length is longer than 4 (if it is the last RR in a

message); a DNS message whose QDCOUNT is 0; a DNS message whose length is less than 12 octets, meaning it doesn't even have a full header; and so on.

- An object in this format can have zero or more of the members defined here; that is, no members are required by the format itself. Instead, profiles that use this format might have requirements for mandatory members, optional members, and prohibited members from the format. Also, this format does not prohibit members that are not defined in this format; profiles of the format are free to add new members in the profile.

1.2. [[Maybe More Than Messages?]]

After some people requested that the RDATA be able to be defined as objects with defined members for all known RTYPEs, this format is just _this close_ to also being able to define zone file contents. My use for the format is for DNS messages, but others might want to be able to describe zone files in JSON. If so, I can extend this document a little and rename it.

2. JSON Format for DNS Messages

The following gives all of the members defined for a DNS message. It is organized approximately by levels of the DNS message.

2.1. Message Object Members

- ID - Integer whose value is 0 to 65535
- QR - Integer whose value is 0 or 1
- Opcode - Integer whose value is 0 to 15
- AA - Integer whose value is 0 or 1
- TC - Integer whose value is 0 or 1
- RD - Integer whose value is 0 or 1
- RA - Integer whose value is 0 or 1
- AD - Integer whose value is 0 or 1
- CD - Integer whose value is 0 or 1
- RCODE - Integer whose value is 0 to 15

- QDCOUNT - Integer whose value is 0 to 65535
- ANCOUNT - Integer whose value is 0 to 65535
- NSCOUNT - Integer whose value is 0 to 65535
- ARCOUNT - Integer whose value is 0 to 65535
- QNAME* - The octets of the QNAME of the first Question section of the message
- QNAME& - The octets of the QNAME of the first Question section of the message
- hostQNAME - The host name of the domain name in the QNAME, or an empty string if the QNAME is not a host name
- QTYPE - Integer whose value is 0 to 65535, of the QTYPE of the first Question section of the message
- QCLASS - Integer whose value is 0 to 65535, of the QCLASS of the first Question section of the message
- questionRRs - Array of zero or more resource records in the Question section
- answerRRs - Array of zero or more resource records in the Answer section
- authorityRRs - Array of zero or more resource records in the Authority section
- additionalRRs - Array of zero or more resource records in the Additional section

2.2. Resource Record Object Members

A resource record is represented as an object with the following members.

- NAME* - The octets of the NAME field of the resource record
- NAME& - The octets of the NAME field of the resource record
- hostNAME - The host name of the domain name in the NAME, or an empty string if the NAME is not a host name

- expandedNAME* - The octets of the NAME field after RFC 1035 removing name compression
- TYPE - Integer whose value is 0 to 65535
- CLASS - Integer whose value is 0 to 65535
- TTL - Integer whose value is 0 to 4294967295
- RDLENGTH - Integer whose value is 0 to 65535
- RDATA - An object that contains members representing the type of data for the class, described below
- RDATA* - The octets of the RDATA field of the resource record
- RDATA! - The octets of the RDATA field of the resource record

The values of the NAME* and hostNAME and expandedNAME* members might all be the same, or could be different.

A Question section can be expressed as a resource record. When doing so, the TTL, RDLENGTH, and RDATA* members make no sense.

2.2.1. RDATA Object Members

The rdata object in a resource record has different members depending on the type. These types will be registered in an IANA registry so that programs might be able to automatically handle new resource record types. The types defined here are listed below.

[[Many of the items below represent domain names. They will be changed to use the same name form as is chosen for QNAME and NAME above. The are marked with a $ below, but the listing might be incomplete; I need to do another pass over the obscure types.]]

A (1)
   ipv4Address (string)

NS (2)
   nsdname (string) $

MD (3)
   madname (string) $

MF (4)
   madname (string) $

```
   CNAME (5)
      cname (string) $

   SOA (6)
      mname (string) $
      rname (string) $
      serial (number)
      refresh (number)
      refresh (number)
      retry (number)
      expire (number)

   MB (7)
      madname (string) $

   MG (8)
      mgmname (string) $

   MR (9)
      newname (string) $

   NULL (10)
      anything (string)

   WKS (11)
      address (string) $
      protocol (number)
      bitmap (string)

   PTR (12)
      ptrdname (string) $

   HINFO (13)
      cpu (string)
      os (string)

   MINFO (14)
      rmailbx (string)
      emailbx (string)

   MX (15)
      preference (number)
      exchange (string) $

   TXT (16)
      txtStrings (array) which contains zero or more strings

   RP (17)
```

```
      mboxDname (string)
      txtDname (string)

   AFSDB (18)
      subtype (number)
      hostname (string) $

   X25 (19)
      psdnAddress (string)

   ISDN (20)
      isdnAddress (string)
      sa (string)

   RT (21)
      preference (number)
      intermediateHost (string) $

   NSAP (22)
      nsap (string)

   SIG (24)
      sigNOWOBSOLETE (string)

   KEY (25)
      keyNOWOBSOLETE (string)

   PX (26)
      preference (number)
      map822 (string)
      mapx400 (string)

   GPOS (27)
      longitude (string)
      latitude (string)
      altitude (string)

   AAAA (28)
      ipv6Address (string)

   LOC (29)
      version (number)
      size (number)
      horizPre (number)
      vertPre (number)
      latitude (number)
      longitude (number)
      altitude (number)
```

```
   NXT (30)
      nxtNOWOBSOLETE (string)

   EID (31)
      eid (string)

   NIMLOC (32)
      nimloc (string)

   SRV (33)
      priority (number)
      weight (number)
      port (number)
      target (string) $

   ATMA (34)
      format (number)
      address (string)

   NAPTR (35)
      order (number)
      preference (number)
      flags (string)
      service (string)
      regexp (string)
      replacement (string) $

   KX (36)
      preference (number)
      exchanger (string) $

   CERT (37)
      type (number)
      keyTag (number)
      algorithm (number)
      certificateOrCRL (string)

   A6 (38)
      a6NOWOBSOLETE (string)

   DNAME (39)
      target (string) $

   SINK (40)
      sink (string)

   OPT (41)
      options (array). Each element of the array is two-element array,
```

```
      where each sub-array is an option code (number) followed by an
      option data (string).

   APL (42)
      addressFamily (number)
      prefix (number)
      n (number)
      afdpart (string)

   DS (43)
      keyTag (number)
      algorithm (number)
      digestType (number)
      digest (string)

   SSHFP (44)
      algorithm (number)
      fpType (number)
      fingerprint (string)

   IPSECKEY (45)
      algorithm (number)
      gatewayType (number)
      precedence (number)
      gateway (string) $
      publicKey (string)

   RRSIG (46)
      typeCovered (number)
      algorithm (number)
      labels (number)
      originalTTL (number)
      signatureExpiration (number)
      signatureInception (number)
      keyTag (number)
      signersName (string)
      signature (string)

   NSEC (47)
      nextDomainName (string) $
      typeBitMaps (string)

   DNSKEY (48)
      flags (number)
      protocol (number)
      algorithm (number)
      publicKey (string)
```

```
   DHCID (49)
      dhcidOpaque (string)

   NSEC3 (50)
      hashAlgorithm (number)
      flags (number)
      iterations (number)
      saltLength (number)
      salt (string)
      hashLength (number)
      nextHashedOwnerName (string) $
      typeBitMaps (string)

   NSEC3PARAM (51)
      hashAlgorithm (number)
      flags (number)
      iterations (number)
      saltLength (number)
      salt (string)

   TLSA (52)
      certificateUsage (number)
      selector (number)
      matchingType (number)
      certificateAssociationData (string).

   HIP (55)
      hitLength (number)
      pkAlgorithm (number)
      pkLength (number)
      hit (string)
      publicKey (string)
      rendezvousServers (string)

   NINFO (56)
      ninfo (string)

   RKEY (57)
      rkey (string)

   TALINK (58)
      talink (string)

   CDS (59)
      cds (string)

   SPF (99)
      text (string)
```

```
   UINFO (100)
      uinfo (string)

   UID (101)
      uid (string)

   GID (102)
      gid (string)

   UNSPEC (103)
      unspec (string)

   NID (104)
      preference (number)
      nodeID (string)

   L32 (105)
      preference (number)
      locator32MSBS (string)
      locator64LSBS (string)

   L64 (106)
      preference (number)
      locator64 (string)

   LP (107)
      preference (number)
      fqdn (string)

   TKEY (249)
      algorithm (string)
      inception (number)
      expiration (number)
      mode (number)
      error (number)
      keyData (string)
      otherData (string)

   TSIG (250)
      algorithm (string)
      timeSigned (string)
      fudge (number)
      mac (string)
      originalID (number)
      error (number)
      otherData (string)

   IXFR (251)
```

```
      zones (string)

   AXFR (252)
      zones (string)

   MAILB (253)
      mailb-unknown (string)

   MAILA (254)
      maila-unknown (string)

   URI (256)
      priority (number)
      weight (number)
      target (string) $

   CAA (257)
      flags (number)
      tag (string)
      value (string)

   TA (32768)
      ta (string)

   DLV (32769)
      Identical to DS (43)
```

2.3. The Message and Its Parts as Octets

The following can be members of a message object.

- messageOctets* - The octets of the message
- messageOctets! - The octets of the message
- headerOctets* - The first 12 octets of the message (or fewer, if the message is truncated)
- questionOctets* - The octets of the Question section
- answerOctets* - The octets of the Answer section
- answerOctets! - The octets of the Answer section
- authorityOctets* - The octets of the Authority section
- additionalOctets* - The octets of the Additional section

The following can be a member of a resource record object.

- rrOctets* - The octets of a particular resource record
- rrOctets! - The octets of a particular resource record

2.4. Additional Message Object Members

The following are members that might appear in a message object:

- dateString - The date that the message was sent or received, given as a string in the standard format described in [RFC3339], as refined by Section 3.3 of [RFC4287]
- dateSeconds - The date that the message was sent or received, given as the number of seconds since 1970-01-01T00:00Z in UTC time; this number can be fractional
- comment - An unstructured comment as a string.

2.5. Representing Domain Names

The QNAME member (and the NAME member of RRs below, as well as some of the members in the RDATA fields) represent a domain name, not just a host name. There are many different ways to represent domain names, and different profiles of this format will want to represent them in different ways. The most important thing to remember when considering which way a profile wants to represent the name is that names are actually octets, not strings.

There is a desire to show doamin names that have just a few octets that are not in the hostname repertoire in a friendly fashion.

This becomes a difficult task for JSON because JSON strings are Unicode characters, some of which are escaped. One cannot just say "let my JSON encoder handle whatever I give it" for at least two reasons. A label that is the three octets 0x41 0x7F 0x42 would be represented as an "A" followed by a DEL character followed by a "B". That is, DEL characters are not escaped in JSON. Further, JSON encoders assume that inputs to strings are encoded Unicode strings, usually encoded with UTF-8. In such an encoder, a label that has two octets 0xC2 0xB5 would be interpreted as the "MICRO SIGN" character (U+00B5) because those two octets are the UTF-8 serialization of that character. Another complication is that 0x2E is represented as a ".", so a single-label domain name that consists of 0x41 0x2E 0x2E would display as "A..".

Given this, there are many possible choices for how to represent domain names in JSON in a human-readable fashion. In this list, exampes are given for the label that has six octets 0x4341743A7FB52E under the TLD "example". Some examples are:

- Show the character for all octets that are ASCII displayable characters (0x21 to 0x7E, but not 0x2E), and the RFC 1035 "\DDD" escaping for all other octets: "CAt:\D127\D181\D046.example."

- Show the character for all octets that are ASCII displayable characters (0x21 to 0x7E, but not 0x2E), and the normal JSON escaping for all other octets: "CAt:&u007f&u00b5&u002e.example."

Making a decision on this topic is important because many RRtypes have domain names for their RDATA. While having three or four choices of format for QNAME and NAME may make sense, having three or four for all those RDATA types seems ungainly.

[[Possibly ugly but usable design thought: all names default to one of the format types, but the outer record object can have a "nameFormat" member whose value is "hex", "1035escape", "uplus", and so on.]]

3. JSON Format for Streams of DNS Messages

A strem of DNS messages is represented as an array of zero or more message objects.

4. JSON Format for a Paired DNS Query and Response

A paired DNS query and response is represented as an object. Two optional members of this object are names "queryRecord" and "responseRecord", and each has a value that is an message object. This design was chosen (as compared to the more obvious array of two values) so that a paired DNS query and response could be differentiated from a stream of DNS messages whose length happens to be two.

5. Examples

5.1. Example of the Format of a DNS Query

The following is an example of a query for the A record of example.com.

```
{ "ID": 19678, "QR": 0, "Opcode": 0,
  "AA": 0, "TC": 0, "RD": 0, "RA": 0, "AD": 0, "CD": 0, "RCODE": 0,
  "QDCOUNT": 1, "ANCOUNT": 0, "NSCOUNT": 0, "ARCOUNT": 0,
  "QNAME*": "076578616D706C6503636F6D00", "QTYPE": 1, "QCLASS": 1
}
```

As stated earlier, all members of an object are optional. This example object could have one or more of the following members as well:

```
"answerRRs": []
"authorityOctets*": ""
"comment": "Something pithy goes here"
"dateSeconds": 1408504748.657783
"headerOctets*": "4CDE00000001000000000000"
"QNAME&": "example.com."
"hostQNAME": "example.com."
"messageOctets*":
     "4CDE00000001000000000000076578616D706C6503636F6D0000010001"
"messageOctets!": "TN4AAAABAAAAAAAAB2V4YW1wbGUDY29tAAABAAE="
"questionOctets*": "076578616D706C6503636F6D0000010001"
"questionRRs": [ { "NAME*": "076578616D706C6503636F6D00", "TYPE": 1,
                   "CLASS": 1, "hostNAME" : "example.com." } ]
"questionRRs": [ { "NAME&": "example.com.", "TYPE": 1,
                   "CLASS": 1, } ]
```

5.2. Example of the Format of a Paired DNS Query and Response

The following is a paired DNS query and response for a query for the A record of example.com.

```
{
  "queryRecord": { "ID": 32784, "QR": 0, "Opcode": 0, "AA": 0,
                   "TC": 0, "RD": 0, "RA": 0, "AD": 0, "CD": 0,
                   "RCODE": 0, "QDCOUNT": 1, "ANCOUNT": 0,
                   "NSCOUNT": 0, "ARCOUNT": 0,
                   "QNAME&": "example.com.",
                   "QTYPE": 1, "QCLASS": 1 },
  "responseRecord": { "ID": 32784, "QR": 1, "AA": 1, "RCODE": 0,
                      "QDCOUNT": 1, "ANCOUNT": 1, "NSCOUNT": 1,
                      "ARCOUNT": 0,
    "answerRRs": [ { "NAME&": "example.com.", "TYPE": 1, "CLASS": 1,
                     "TTL": 3600, "RDLENGTH": 4,
                     "RDATA*": "16212C37" } ],
    "authorityRRs": [ { "NAME&": "ns.example.com.", "TYPE": 1,
                        "CLASS": 1, "TTL": 28800, "RDLENGTH": 4,
                        "RDATA*": "A5E3F903" } ] }
}
```

6. Local Format Policy

Systems using this format in this document will likely have policy about what must be in the objects. Those policies are outside the scope of this document.

For example, private DNS systems such as those described in [I-D.dulaunoy-kaplan-passive-dns-cof] covers just DNS responses. Such a system might have a policy that makes QNAME*, QTYPE, and answerRRs mandatory. That document also describes two mandatory times that are not in this format, so the policy would possibly also define those members and make them mandatory. The policy could also define additional members that might appear in a record.

As another example, a program that uses this format for configuring what a test client sends on the wire might have a policy of "each record object can have as few members as it wants; all unstated members are filled in from previous records".

7. IANA Considerations

This document has no effect on IANA registries.

8. Security Considerations

As described in Section 1.1, a message object can have inconsistent data, such as a message with an ANCOUNT of 1 but that has either an empty answerRRs array or an answerRRs array that has 2 or more RRs. Other examples of inconsistent data would be resource records whose RDLENGTH does not match the length of the decoded value in the RDATA* member, or a record whose various header fields do not match the value in headerOctets*, and so on. A reader of this format must never assume that all of the data in an object are all consistent with each other.

Numbers in JSON do not have any bounds checking. Thus, integer values in a record might have invalid values, such as an ID value that is negative, or greater than or equal to 2^16, or has a fractional part.

9. Acknowledgements

Some of the ideas in this document were inspired by [I-D.dulaunoy-kaplan-passive-dns-cof]. The document was also inspired by early ideas from Stephane Bortzmeyer. There was earlier, abandoned work on encoding DNS messages in XML ([I-D.daley-dnsxml] and [I-D.mohan-dns-query-xml], to name just two).

10. References

10.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

10.2. Informative References

[I-D.daley-dnsxml]
Daley, J., Morris, S., and J. Dickinson, "dnsxml - A standard XML representation of DNS data", draft-daley-dnsxml-00 (work in progress), July 2013.

[I-D.dulaunoy-kaplan-passive-dns-cof]
Dulaunoy, A., Kaplan, A., Vixie, P., and H. Stern, "Passive DNS - Common Output Format", draft-dulaunoy-kaplan-passive-dns-cof-02 (work in progress), March 2014.

[I-D.mohan-dns-query-xml]
Parthasarathy, M. and P. Vixie, "Representing DNS messages using XML", draft-mohan-dns-query-xml-00 (work in progress), September 2011.

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

Author's Address

Paul Hoffman
VPN Consortium

Email: paul.hoffman@vpnc.org