DNSOP
Internet-Draft
Intended status: Standards Track
Expires: September 13, 2017

R. Arends
ICANN
J. Schlyter
Kirei AB
M. Larson
ICANN

March 12, 2017

DNS Security (DNSSEC) DNSKEY Algorithm IANA Registry Updates
draft-arends-dnsop-dnssec-algorithm-update-00

Abstract

The DNS Security Extensions (DNSSEC) require the use of cryptographic algorithm suites for generating digital signatures and cryptographic hashes over DNS data. The algorithms specified for use with DNSSEC are reflected in IANA registries. This document updates some entries in these registries. The main reason for these updates is to retire the use of SHA1.

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 September 13, 2017.

Copyright Notice

Table of Contents

1. Introduction

The Domain Name System (DNS) Security Extensions (DNSSEC, defined by [RFC4033], [RFC4034], [RFC4035], [RFC4509], [RFC5155], and [RFC5702]) use digital signatures over DNS data to provide source authentication and integrity protection. DNSSEC uses IANA registries to list codes for digital signature algorithms and hash functions.

This document updates a set of entries in the IANA registries titled "DNS Security (DNSSEC) Algorithm Numbers" and "Delegation Signer (DS) Resource Record (RR) Type Digest Algorithms".

1.1. Reserved Words

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 [RFC2119].

2. The DNS Security Algorithm Implementation Status Lists

2.1. Status Definitions

Must Implement: The algorithm MUST be implemented to interoperate with other implementations of this specification.

Must Not Implement: The algorithm MUST NOT be implemented. An algorithm with this status has known weaknesses.

Recommended to Implement: The algorithm SHOULD be implemented. Utility and interoperability with other implementations will be improved when an algorithm with this status is implemented, though there might be occasions where it is reasonable not to implement the algorithm. An implementer must understand and weigh the full implications of choosing not to implement this particular algorithm.

Optional: The algorithm MAY be implemented, but all implementations MUST be prepared to interoperate with implementations that do or do not implement this algorithm.

2.2. Algorithm Implementation Status Assignment Rationale

RSASHA1 had an implementation status of "Must Implement", consistent with [RFC4034]. The status of RSASHA1 is set to "Recommended to Implement" consistent with RSASHA1-NSEC3-SHA1. The shift from "Must Implement" to "Recommended to Implement" is due to a perceived weakness in SHA1.

The status of RSASHA256 is set to "Must Implement" as major deployments, such as the root zone [RZDPS], use these algorithms. It is believed that RSA/SHA-256 or RSA/SHA-512 algorithms will replace older algorithms (e.g., RSA/SHA-1) that have a perceived weakness.

All other algorithms used in DNSSEC specified without an implementation status are currently set to "Optional".

2.3. DNSSEC Implementation Status Table

The DNSSEC algorithm implementation status table is listed below. Only the algorithms already specified for use with DNSSEC at the time of writing are listed.

Must Implement	Must Not Implement	Recommended	Optional
RSASHA256	RSAMD5	RSASHA1	Any registered algorithm not listed in this table
		RSASHA1-NSEC3-SHA1	
		RSASHA512	
		ECDSAP256SHA256	
		ECDSAP384SHA384	

This table does not list the Reserved values in the IANA registry table or the values for INDIRECT (252), PRIVATE (253), and PRIVATEOID (254). These values may relate to more than one algorithm and are therefore up to the implementer's discretion. As noted, any algorithm not listed in the table is "Optional".

2.4. Specifying New Algorithms and Updating the Status of Existing Entries

[RFC6014] establishes a parallel procedure for adding a registry entry for a new algorithm other than a standards track document. Because any algorithm not listed in the foregoing table is "Optional", algorithms entered into the registry using the [RFC6014] procedure are automatically "Optional".

It has turned out to be useful for implementations to refer to a single document that specifies the implementation status of every algorithm. Accordingly, when a new algorithm is to be registered with a status other than "Optional", this document shall be made obsolete by a new document that adds the new algorithm to the table in Section 2.3. Similarly, if the status of any algorithm in the table in Section 2.3 changes, a new document shall make this document obsolete; that document shall include a replacement of the table in Section 2.3. This way, the goal of having one authoritative document to specify all the status values is achieved.

This document cannot be updated, only made obsolete and replaced by a successor document.

3. IANA Considerations

This document lists the implementation status of cryptographic algorithms used with DNSSEC. These algorithms are maintained in an IANA registry at <http://www.iana.org/assignments/dns-sec-alg-numbers>. Because this document establishes the implementation status of every algorithm, it has been listed as a reference for the registry itself.

4. Security Considerations

This document lists, and in some cases assigns, the implementation status of cryptographic algorithms used with DNSSEC. It is not meant to be a discussion on algorithm superiority.

Since the status of two algorithms have changed, it is important to consider the long term effect of these changes in implementations.

An implementation may have provided a default algorithm to use when generating a DNSKEY. An implementation may select a default algorithm to sign DNSSEC records. It is recommended that implementations that provide a default algorithm use an algorithm with the status "Must Implement".

5. References

5.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

5.2. Informative References

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <http://www.rfc-editor.org/info/rfc4033>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <http://www.rfc-editor.org/info/rfc4034>.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <http://www.rfc-editor.org/info/rfc4035>.

[RFC4509] Hardaker, W., "Use of SHA-256 in DNSSEC Delegation Signer (DS) Resource Records (RRs)", RFC 4509, DOI 10.17487/RFC4509, May 2006, <http://www.rfc-editor.org/info/rfc4509>.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, DOI 10.17487/RFC5155, March 2008, <http://www.rfc-editor.org/info/rfc5155>.

[RFC5702] Jansen, J., "Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC", RFC 5702, DOI 10.17487/RFC5702, October 2009, <http://www.rfc-editor.org/info/rfc5702>.

[RFC6014] Hoffman, P., "Cryptographic Algorithm Identifier Allocation for DNSSEC", RFC 6014, DOI 10.17487/RFC6014, November 2010, <http://www.rfc-editor.org/info/rfc6014>.

[RZDPS] Ljunggren, F., Okubo, T., Lamb, R., and J. Schlyter, "DNSSEC Practice Statement for the Root Zone KSK Operator", October 2010, <https://www.iana.org/dnssec/icann-dps.txt>.

Authors' Addresses

Roy Arends
ICANN

Email: roy.arends@icann.org

Jakob Schlyter
Kirei AB

Email: jakob@kirei.se

Matt Larson
ICANN

Email: matt.larson@icann.org

DNSOP Working Group
Internet-Draft
Updates: RFC 2845, RFC 2931 (if approved) (if approved)
Intended status: Standards Track
Expires: September 6, 2018

R. Bellis
ISC
P. van Dijk
R. Gacogne
PowerDNS

March 05, 2018

DNS X-Proxied-For
draft-bellis-dnsop-xpf-04

Abstract

It is becoming more commonplace to install front end proxy devices in front of DNS servers to provide (for example) load balancing or to perform transport layer conversions.

This document defines a meta resource record that allows a DNS server to receive information about the client's original transport protocol parameters when supplied by trusted proxies.

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 September 6, 2018.

Copyright Notice

Table of Contents

1. Introduction

It is becoming more commonplace to install front end proxy devices in front of DNS servers [RFC1035] to provide load balancing or to perform transport layer conversions (e.g. to add DNS over TLS [RFC7858] to a DNS server that lacks native support).

This has the unfortunate side effect of hiding the clients' source IP addresses from the server, making it harder to employ server-side technologies that rely on knowing those addresses (e.g. ACLs, DNS Response Rate Limiting, etc).

This document defines the XPF meta resource record (RR) that allows a DNS server to receive information about the client's original transport protocol parameters when supplied by trusted proxies.

Whilst in some circumstances it would be possible to re-use the Client Subnet EDNS Option [RFC7871] to carry a subset of this

information, a new RR is defined to allow both this feature and the Client Subnet Option to co-exist in the same packet.

2. Terminology

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.

The XPF RR is analogous to the HTTP "X-Forwarded-For" header, but in DNS the term "forwarder" is usually understood to describe a network component that sits on the outbound query path of a resolver.

Instead we use the term "proxy", which in this document means a network component that sits on the inbound query path in front of a recursive or authoritative DNS server, receiving DNS queries from clients and dispatching them to local servers.

3. Description

The XPF RR contains the entire 6-tuple (IP version, Layer 4 protocol, source address, destination address, source port and destination port) of the packet received from the client by the proxy.

The presence of the source address supports use of ACLs based on the client’s IP address.

The source port allows for ACLs to support Carrier Grade NAT whereby different end-users might share a single IP address.

The destination address supports scenarios where the server behaviour depends upon the packet destination (e.g. BIND view’s "match-destinations" option)

The protocol and destination port fields allow server behaviour to vary depending on whether DNS over TLS [RFC7858] or DNS over DTLS [RFC8094] are in use.

3.1. Client Handling

Stub resolvers, client-side proxy devices, and recursive resolvers MUST NOT add the XPF RR to DNS requests.

3.2. Request Handling

The rules in this section apply to processing of the XPF RR whether by a proxy device or a DNS server.

If this RR is received from a non-white-listed client the server MUST return a REFUSED response.

If a server finds this RR anywhere other than in the Additional Section of a request it MUST return a REFUSED response.

If the value of the RR's IP version field is not understood by the server it MUST return a REFUSED response.

If the length of the IP addresses contained in the RR are not consistent with that expected for the given IP version then the server MUST return a FORMERR response.

Servers MUST NOT send this RR in DNS responses.

3.3. Proxy Handling

For each request received, proxies MUST generate an XPF RR containing the 6-tuple representing the client's Layer 3 and Layer 4 headers and append it to the Additional Section of the request (updating the ARCOUNT field accordingly) before sending it to the intended DNS server.

If a valid XPF RR is received from a white-listed client the original XPF RR MUST be preserved instead.

3.4. Server Handling

When this RR is received from a white-listed client the DNS server SHOULD use the transport information contained therein in preference to the packet's own transport information for any data processing logic (e.g. ACLs) that would otherwise depend on the latter.

3.5. Wire Format

The XPF RR is formatted like any standard RR, but none of the fields except RDLENGTH and RDATA have any meaning in this specification. All multi-octet fields are transmitted in network order (i.e. big-endian).

The required values of the RR header fields are as follows:

NAME: MUST contain a single 0 octet (i.e. the root domain).

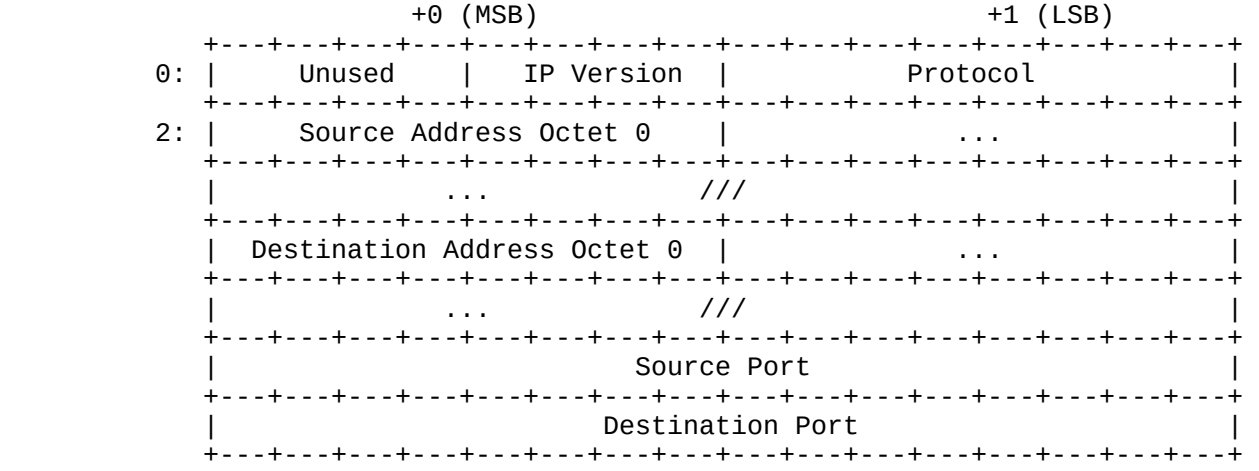
TYPE: MUST contain TBD1 (XPF).

CLASS: MUST contain 1 (IN).

TTL: MUST contain 0 (zero).

RDLENGTH: specifies the length in octets of the RDATA field.

The RDATA of the XPF RR is as follows:



Unused: Currently reserved. These bits MUST be zero unless redefined in a subsequent specification.

IP Version: The IP protocol version number used by the client, as defined in the IANA IP Version Number Registry [IANA-IP]. Implementations MUST support IPv4 (4) and IPv6 (6).

Protocol: The Layer 4 protocol number (e.g. UDP or TCP) as defined in the IANA Protocol Number Registry [IANA-PROTO].

Source Address: The source IP address of the client.

Destination Address: The destination IP address of the request, i.e. the IP address of the proxy on which the request was received.

Source Port: The source port used by the client.

Destination Port: The destination port of the request.

The length of the Source Address and Destination Address fields will be variable depending on the IP Version used by the client.

3.6. Presentation Format

XPF is a meta RR that cannot appear in master format zone files, but a standardised presentation format is defined here for use by debugging utilities that might need to display the contents of an XPF RR.

The Unused bits and the IP Version field are treated as a single octet and presented as an unsigned decimal integer with range 0 .. 255.

The Protocol field is presented as an unsigned decimal integer with range 0 .. 255.

The Source and Destination Address fields are presented either as IPv4 or IPv6 addresses according to the IP Version field. In the case of IPv6 the recommendations from [RFC5952] SHOULD be followed.

The Source and Destination Port fields are presented as unsigned decimal integers with range 0 .. 65535.

3.7. Signed DNS Requests

Any XPF RRs found in a packet MUST be ignored for the purposes of calculating or verifying any signatures used for Secret Key Transaction Authentication for DNS [RFC2845] or DNS Request and Transaction Signatures (SIG(0)) [RFC2931].

Typically it is expected that proxies will append the XPF RR to the packet after any existing TSIG or SIG(0) RRs, and that servers will remove the XPF RR from the packet prior to verification of the original signature, with the ARCOUNT field updated as appropriate.

If either TSIG or SIG(0) are configured between the proxy and server then any XPF RRs MUST be ignored when the proxy calculates the packet signature.

4. Security Considerations

If the white-list of trusted proxies is implemented as a list of IP addresses, the server administrator MUST have the ability to selectively disable this feature for any transport where there is a possibility of the proxy’s source address being spoofed.

This does not mean to imply that use over UDP is impossible - if for example the network architecture keeps all proxy-to-server traffic on a dedicated network and clients have no direct access to the servers then the proxies’ source addresses can be considered unspoofable.

5. Implementation status

[RFC Editor Note: Please remove this entire section 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 [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 [RFC7942], "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".

5.1. dnsdist

Support for adding an XPF RR to proxied packets is provided in the git version of dnsdist. The code point is configurable.

5.2. PowerDNS Recursor

Support for extracting the XPF RR from received packets (when coming from a trusted source) is available in the git version of the PowerDNS Recursor. The code point is configurable.

5.3. Wireshark

Support for dissecting XPF RRs is present in Wireshark 2.5.0, using a temporary code point of 65422.

6. Privacy Considerations

Used incorrectly, this RR could expose internal network information, however it is not intended for use on proxy / forwarder devices that sit on the client-side of a DNS request.

This specification is only intended for use on server-side proxy devices that are under the same administrative control as the DNS servers themselves. As such there is no change in the scope within which any private information might be shared.

Use other than as described above would be contrary to the principles of [RFC6973].

7. IANA Considerations

<< a copy of the RFC 6895 IANA RR TYPE application template will appear here >>

8. Acknowledgements

Mark Andrews, Robert Edmonds, Duane Wessels

9. References

9.1. Normative References

[IANA-IP] IANA, "IANA IP Version Registry", n.d., <http://www.iana.org/assignments/version-numbers/>.

[IANA-PROTO] IANA, "IANA Protocol Number Registry", n.d., <http://www.iana.org/assignments/protocol-numbers/>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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>.

[RFC2845] Vixie, P., Gudmundsson, O., Eastlake 3rd, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, DOI 10.17487/RFC2845, May 2000, <https://www.rfc-editor.org/info/rfc2845>.

[RFC2931] Eastlake 3rd, D., "DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, DOI 10.17487/RFC2931, September 2000, <https://www.rfc-editor.org/info/rfc2931>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[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>.

[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>.

9.2. Informative References

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/info/rfc7858>.

[RFC7871] Contavalli, C., van der Gaast, W., Lawrence, D., and W. Kumari, "Client Subnet in DNS Queries", RFC 7871, DOI 10.17487/RFC7871, May 2016, <https://www.rfc-editor.org/info/rfc7871>.

[RFC8094] Reddy, T., Wing, D., and P. Patil, "DNS over Datagram Transport Layer Security (DTLS)", RFC 8094, DOI 10.17487/RFC8094, February 2017, <https://www.rfc-editor.org/info/rfc8094>.

Authors' Addresses

Ray Bellis
Internet Systems Consortium, Inc.
950 Charter Street
Redwood City CA 94063
USA

Phone: +1 650 423 1200
Email: ray@isc.org

Peter van Dijk
PowerDNS.COM B.V.
Den Haag
The Netherlands

Email: peter.van.dijk@powerdns.com

Remi Gacogne
PowerDNS.COM B.V.
Den Haag
The Netherlands

Email: remi.gacogne@powerdns.com

dnsop
Internet-Draft
Intended status: Best Current Practice
Expires: November 23, 2018

D. Crocker
Brandenburg InternetWorking

May 22, 2018

DNS Scoped Data Through '_Underscore' Naming of Attribute Leaves
draft-ietf-dnsop-attrleaf-09

Abstract

Formally, any DNS resource record may occur for any domain name. However some services have defined an operational convention, which applies to DNS leaf nodes that are under a DNS branch having one or more reserved node names, each beginning with an underscore. The underscore naming construct defines a semantic scope for DNS record types that are associated with the parent domain, above the underscored branch. This specification explores the nature of this DNS usage and defines the "DNS Global Underscore Scoped Entry Registry" with IANA. The purpose of the Underscore registry is to avoid collisions resulting from the use of the same underscore-based name, for different services.

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 23, 2018.

Table of Contents

1. Introduction

The core Domain Name System (DNS) technical specifications assign no semantics to domain names or their parts, and no constraints upon which resource record (RR) types are permitted to be stored under particular names [RFC1035], [RFC2181]. Over time, some leaf node names, such as "www" and "ftp" have come to imply support for particular services, but this is a matter of operational convention, rather than defined protocol semantics. This freedom in the basic technology has permitted a wide range of administrative and semantic policies to be used -- in parallel. DNS data semantics have been limited to the specification of particular resource record types, on the expectation that new ones would be added as needed. Unfortunately, the addition of new resource record types has proven extremely challenging, over the life of the DNS, with significant adoption and use barriers.

1.1. _Underscore Scoping

As an alternative to defining a new RR type, some DNS service enhancements call for using an existing resource record type, but specify a restricted scope for its occurrence. Scope is meant as a

static property, not one dependent on the nature of the query. It is an artifact of the DNS name. That scope is a leaf node, within which the uses of specific resource record sets can be formally defined and constrained. The leaf occurs in a branch having a distinguished naming convention: At the top of the branch -- beneath the parent domain name to which the scope applies -- one or more reserved DNS node names begin with an underscore ("_"). Because the DNS rules for a "host" (host name) are not allowed to use the underscore character, this distinguishes the underscore name from all legal host names [RFC952]. Effectively, this convention for leaf node naming creates a space for the listing of 'attributes' -- in the form of resource record types -- that are associated with the parent domain, above the underscored sub-branch.

The scoping feature is particularly useful when generalized resource record types are used -- notably "TXT", "SRV", and "URI" [RFC1035], [RFC2782], [RFC6335], [RFC7553]. It provides efficient separation of one use of them from others. Absent this separation, an undifferentiated mass of these "RRsets" is returned to the DNS client, which then must parse through the internals of the records in the hope of finding ones that are relevant. Worse, in some cases the results are ambiguous because a record type might not adequately self-identify. With underscore-based scoping, only the relevant "RRsets"s are returned.

A simple example is DKIM [RFC6376] , which uses "_domainkeys" for defining a place to hold a "TXT" record containing signing information for the parent domain.

This specification formally defines how underscore labels are used as "attribute" enhancements for their parent domain names. For example, domain name "_domainkey.example." acts as attribute of parent domain name "example." To avoid collisions resulting from the use of the same underscore-based labels for different applications using the same resource record type, this document establishes DNS Underscore Global Scoped Entry IANA Registry for the highest-level reserved names that begin with _underscore; _underscore-based names that are farther down the hierarchy are handled within the scope of the highest-level _underscore name.

Discussion Venue: Discussion about this draft should be directed to the dnsop@ietf.org [1] mailing list.

NOTE TO RFC EDITOR: Please remove "Discussion Venue" paragraph prior to publication.

1.2. Scaling Benefits

Some resource record types are used in a fashion that can create scaling problems, if an entire RRset associated with a domain name is aggregated in the leaf node for that name. An increasingly-popular approach, with excellent scaling properties, places the RRset under a node having an underscore-based name, at a defined place in the DNS tree under the 'parent' name. This constrains the use of particular "RR" types associated with that parent name. A direct lookup to the subordinate leaf node produces only the desired record types, at no greater cost than a typical DNS lookup.

The definition of a underscore global registry, provided in this specification, primarily attends to the top-most names used for scoping an RR type; that is the _underscore "global" names.

2. DNS Underscore Scoped Entry Registries Function

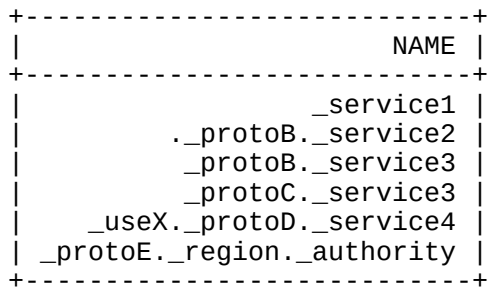
A global registry for DNS nodes names that begin with an _underscore is defined here. The purpose of the Underscore Global Registry is to avoid collisions resulting from the use of the same _underscore-based name, for different applications.

- If a public specification calls for use of an _underscore-prefixed domain node name, the 'global' (right-most) _underscored name MUST be entered into this registry.

The _underscore names define scope of use for specific resource record types, which are associated with the domain name that is the "parent" to the branch defined by the _underscore naming. A given name defines a specific, constrained context for one or more RR types, where use of such record types conforms to the defined constraints.

- Within an _underscore scoped leaf, other RRsets that are not specified as part of the scope MAY be used.

Structurally, the registry is defined as a single, flat table of RR types, under node names beginning with _underscore. In some cases, such as for use of an "SRV" record, the full scoping name might be multi-part, as a sequence of underscore names. Semantically, that sequence represents a hierarchical model and it is theoretically reasonable to allow re-use of a subordinate underscore name in different underscore context; that is, a subordinate name is meaningful only within the scope of the right-most (top-level) underscore name. Therefore they are ignored by this DNS Underscore Global Scoped Entry Registry. This registry is for the definition of highest-level -- ie, global -- underscore node name used.



Example of Underscore Names

Only the right-most _underscore names are registered in the IANA Underscore Global table.

The use of underscored node names is specific to each RRTYPE that is being scoped. Each name defines a place, but does not define the rules for what appears underneath that place, either as additional underscored naming or as a leaf node with resource records. Details for those rules are provided by specifications for individual RRTYPEs. The sections below describe the way that existing underscore labels are used with the RRTYPEs that they name.

Definition and registration of the subordinate underscore node names is the responsibility of the specification that creates the highest-level (right-most) global registry entry.

That is, if a scheme using a global underscore node name also has one or more subordinate levels of underscore node naming, the namespaces from which names for those lower levels is chosen is controlled by the parent underscore node name. Each globally-registered underscore name owns a distinct, subordinate name space.

3. RRset Use Registration Template

This section provides a basic template that can be used to register new entries in the IANA DNS Underscore Global Scoped Entry Registry, if the right-most underscored name above the RRTYPE is not already registered. The text can be added to specifications using RRTYPE/_Node-name combinations that have not already been registered.

"Per {RFC Attrleaf} please add the following entry to the DNS Underscore Global Scoped Entry Registry:"

Note to RFC Editor: Please replace the above "{RFC Attrleaf}" text with a reference to this document's RFC number. /d

RR Type	_NODE NAME	REFERENCE
{RRTYPE}	_{DNS global node name}	{citation for the document making the addition.}

Table 1: Underscore Global Registry Entry

4. IANA Considerations

Per [RFC8126], IANA is requested to establish the:

DNS Underscore Global Scoped Entry Registry

This section describes actions requested of IANA. The guidance in [IANA] is used.

4.1. DNS Underscore Global Scoped Entry Registry

The DNS Global Underscore Scoped Entry Registry is for DNS node names that begin with the underscore character (_) and are the right-most occurrence of any underscored names in a domain name sequence having that form; that is they are the "top" of a DNS branch, under a "parent" domain name.

- This registry is to operate under the IANA rules for "Expert Review" registration; see Section 5.
- The contents of each entry in the Global registry are defined in Section 4.2.
- Each entry in the registry MUST contain values for all of the fields specified in Section 4.2.
- Within the registry, the combination of RR Type and _Node Name MUST be unique.
- The table is to be maintained with entries sorted by the first column (RR Type) and within that the second column (_Node Name).
- The required Reference for an entry MUST have a stable resolution to the organization controlling that registry entry

4.2. DNS Underscore Global Scoped Entry Registry Definition

A registry entry contains:

RR Type: Lists an RR type that is defined for use within this scope.

_Node Name: Specifies a single _underscore name that defines a reserved name; this name is the "global" entry name for the scoped resource record types that are associated with that name

References: Lists specification that defines a record type and its use under this Name. The organization producing the specification retains control over the registry entry for the _Node Name.

Each RR type that is to be used MUST have a separate registry entry.

4.3. Initial entries

Initial entries in the registry are:

RR Type	_NODE NAME	REFERENCE
OPENPGPKEY	_openpgpkey	[RFC7929]
SMIMEA	_smimecert	[RFC8162]
SRV	_dccp	[RFC2782]
SRV	_sctp	[RFC2782]
SRV	_tcp	[RFC2782]
SRV	_udp	[RFC2782]
TLSA	_sctp	[RFC6698]
TLSA	_tcp	[RFC6698]
TLSA	_udp	[RFC6698]
TXT	_mta-sts	[MTA-STS]
TXT	_acme-challenge	[ACME]
TXT	_dmarc	[RFC7489]
TXT	_domainkey	[RFC6376]
TXT	_spf	[RFC7208]
TXT	_vouch	[RFC5518]
URI	_iax	[RFC7553]
URI	_acct	[RFC7553]
URI	_dccp	[RFC7553]
URI	_email	[RFC7553]
URI	_ems	[RFC7553]
URI	_fax	[RFC7553]
URI	_ft	[RFC7553]
URI	_h323	[RFC7553]
URI	_ical-sched	[RFC7553]
URI	_ical-access	[RFC7553]
URI	_ifax	[RFC7553]
URI	_im	[RFC7553]
URI	_mms	[RFC7553]
URI	_pres	[RFC7553]
URI	_pstn	[RFC7553]
URI	_sctp	[RFC7553]
URI	_sip	[RFC7553]
URI	_sms	[RFC7553]
URI	_tcp	[RFC7553]
URI	_udp	[RFC7553]
URI	_unifmsg	[RFC7553]
URI	_vcard	[RFC7553]
URI	_videomsg	[RFC7553]
URI	_voice	[RFC7553]
URI	_voicemsg	[RFC7553]
URI	_vpim	[RFC7553]
URI	_xmp	[RFC7553]

Table 2: Underscore Global Registry (initial entries)

5. Guidance for Expert Review

This section provides guidance for expert review of registration requests in the of DNS Underscore Global Scoped Entry Registry.

> This review is solely to determine adequacy of a requested entry in this Registry, and does not include review of other aspects of the document specifying that entry. For example such a document might also contain a definition of the resource record type that is referenced by the requested entry. Any required review of that definition is separate from the expert review required here.

The review is for the purposes of ensuring that:

- The details for creating the registry entry are sufficiently clear, precise and complete
- The combination of the _underscore name, under which the listed resource record type is used, and the resource record type, is unique in the table

For the purposes of this Expert Review, other matters of the specification’s technical quality, adequacy or the like are outside of scope.

6. Security Considerations

This memo raises no security issues.

7. References

7.1. Normative References

[ACME] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", I-D draft-ietf-acme-acme-11, March 2018.

[IANA] M. Cotton, B. Leiba, and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 8126, June 2017.

[MTA-STS] Margolis, D., Risher, M., Ramakrishnan, B., Brotman, A., and J. Jones, "SMTP MTA Strict Transport Security (MTA-STS)", I-D draft-ietf-uta-mta-sts.

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

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC5518] Hoffman, P., Levine, J., and A. Hathcock, "Vouch By Reference", RFC 5518, April 2009.

[RFC6335] Cotton, M., Eggert, L., Tpuch, J., Westerlund, M., and S. Cheshire, "nternet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", RFC 6335, Aug 2011.

[RFC6376] Crocker, D., Hansen, T., and M. Kucherawy, "DomainKeys Identified Mail (DKIM) Signatures", RFC 6376, Sept 2011.

[RFC6698] Hoffman, J. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, August .

[RFC7208] Kitterman, S., "Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1", RFC 7208, April 2014.

[RFC7489] Kucherawy, M., Ed. and E. Zwicky, Ed., "Domain-based Message Authentication, Reporting, and Conformance (DMARC)", RFC 7489, March 2015.

[RFC7553] Falstrom, P. and O. Kolkman, "The Uniform Resource Identifier (URI) DNS Resource Record", RFC 7553, ISSN 2070-1721, June 2015.

[RFC7929] Wouters, P., , RFC 7929, August 2016.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 8126, June 2017.

[RFC8162] Hoffman, P. and J. Schlyter, "Using Secure DNS to Associate Certificates with Domain Names for S/MIME", RFC 8162, May 2017.

[RFC952] Harrenstien, K., Stahl, M., and E. Feinler, "DOD Internet Host Table Specification", RFC 952, October 1985.

7.2. URIs

[1] mailto:dnsop@ietf.org

Appendix A. Acknowledgements

Thanks go to Bill Fenner, Tony Hansen, Martin Hoffmann, Peter Koch, Olaf Kolkman, and Andrew Sullivan for diligent review of the (much) earlier drafts. For the later enhancements, thanks to: Stephane Bortzmeyer, Bob Harold, Warren Kumari, John Levine, Joel Jaeggli, Petr Špaček, Ondřej Surř, Paul Vixie, Tim Wicinski, and Paul Wouters.

Special thanks to Ray Bellis for his persistent encouragement to continue this effort, as well as the suggestion for an essential simplification to the registration model.

Author’s Address

Dave Crocker
Brandenburg InternetWorking
675 Spruce Dr.
Sunnyvale, CA 94086
USA

Phone: +1.408.246.8253
Email: dcrocker@bbiw.net
URI: http://bbiw.net/

dnsop
Internet-Draft
Intended status: Standards Track
Expires: November 9, 2018

J. Dickinson
J. Hague
S. Dickinson
Sinodun IT
T. Manderson
J. Bond
ICANN

May 8, 2018

C-DNS: A DNS Packet Capture Format
draft-ietf-dnsop-dns-capture-format-07

Abstract

This document describes a data representation for collections of DNS messages. The format is designed for efficient storage and transmission of large packet captures of DNS traffic; it attempts to minimize the size of such packet capture files but retain the full DNS message contents along with the most useful transport metadata. It is intended to assist with the development of DNS traffic monitoring applications.

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 November 9, 2018.

Table of Contents

1. Introduction

There has long been a need to collect DNS queries and responses on authoritative and recursive name servers for monitoring and analysis. This data is used in a number of ways including traffic monitoring, analyzing network attacks and "day in the life" (DITL) [ditl] analysis.

A wide variety of tools already exist that facilitate the collection of DNS traffic data, such as DSC [dsc], packetq [packetq], dnscap [dnscap] and dnstap [dnstap]. However, there is no standard exchange format for large DNS packet captures. The PCAP [pcap] or PCAP-NG [pcapng] formats are typically used in practice for packet captures, but these file formats can contain a great deal of additional information that is not directly pertinent to DNS traffic analysis and thus unnecessarily increases the capture file size.

There has also been work on using text based formats to describe DNS packets such as [I-D.daley-dnsxml], [I-D.hoffman-dns-in-json], but these are largely aimed at producing convenient representations of single messages.

Many DNS operators may receive hundreds of thousands of queries per second on a single name server instance so a mechanism to minimize the storage size (and therefore upload overhead) of the data collected is highly desirable.

The format described in this document, C-DNS (Compacted-DNS), focusses on the problem of capturing and storing large packet capture files of DNS traffic with the following goals in mind:

- Minimize the file size for storage and transmission.
- Minimize the overhead of producing the packet capture file and the cost of any further (general purpose) compression of the file.

This document contains:

- A discussion of some common use cases in which DNS data is collected, see Section 3.
- A discussion of the major design considerations in developing an efficient data representation for collections of DNS messages, see Section 4.
- A description of why CBOR [RFC7049] was chosen for this format, see Section 5.
- A conceptual overview of the C-DNS format, see Section 6.
- The definition of the C-DNS format for the collection of DNS messages, see Section 7.
- Notes on converting C-DNS data to PCAP format, see Section 8.
- Some high level implementation considerations for applications designed to produce C-DNS, see Section 9.

2. Terminology

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 [RFC2119].

"Packet" refers to an individual IPv4 or IPv6 packet. Typically packets are UDP datagrams, but may also be part of a TCP data stream. "Message", unless otherwise qualified, refers to a DNS payload extracted from a UDP datagram or a TCP data stream.

The parts of DNS messages are named as they are in [RFC1035]. Specifically, the DNS message has five sections: Header, Question, Answer, Authority, and Additional.

Pairs of DNS messages are called a Query and a Response.

3. Data collection use cases

In an ideal world, it would be optimal to collect full packet captures of all packets going in or out of a name server. However, there are several design choices or other limitations that are common to many DNS installations and operators.

- DNS servers are hosted in a variety of situations:
  - Self-hosted servers
  - Third party hosting (including multiple third parties)
  - Third party hardware (including multiple third parties)
- Data is collected under different conditions:
  - On well-provisioned servers running in a steady state
  - On heavily loaded servers
  - On virtualized servers
  - On servers that are under DoS attack
  - On servers that are unwitting intermediaries in DoS attacks
- Traffic can be collected via a variety of mechanisms:
  - Within the name server implementation itself
  - On the same hardware as the name server itself
  - Using a network tap on an adjacent host to listen to DNS traffic
  - Using port mirroring to listen from another host

- The capabilities of data collection (and upload) networks vary:
  - Out-of-band networks with the same capacity as the in-band network
  - Out-of-band networks with less capacity than the in-band network
  - Everything being on the in-band network

Thus, there is a wide range of use cases from very limited data collection environments (third party hardware, servers that are under attack, packet capture on the name server itself and no out-of-band network) to "limitless" environments (self hosted, well provisioned servers, using a network tap or port mirroring with an out-of-band networks with the same capacity as the in-band network). In the former, it is infeasible to reliably collect full packet captures, especially if the server is under attack. In the latter case, collection of full packet captures may be reasonable.

As a result of these restrictions, the C-DNS data format is designed with the most limited use case in mind such that:

- data collection will occur on the same hardware as the name server itself
- collected data will be stored on the same hardware as the name server itself, at least temporarily
- collected data being returned to some central analysis system will use the same network interface as the DNS queries and responses
- there can be multiple third party servers involved

Because of these considerations, a major factor in the design of the format is minimal storage size of the capture files.

Another significant consideration for any application that records DNS traffic is that the running of the name server software and the transmission of DNS queries and responses are the most important jobs of a name server; capturing data is not. Any data collection system co-located with the name server needs to be intelligent enough to carefully manage its CPU, disk, memory and network utilization. This leads to designing a format that requires a relatively low overhead to produce and minimizes the requirement for further potentially costly compression.

However, it is also essential that interoperability with less restricted infrastructure is maintained. In particular, it is highly desirable that the collection format should facilitate the re-creation of common formats (such as PCAP) that are as close to the original as is realistic given the restrictions above.

4. Design considerations

This section presents some of the major design considerations used in the development of the C-DNS format.

1. The basic unit of data is a combined DNS Query and the associated Response (a "Q/R data item"). The same structure will be used for unmatched Queries and Responses. Queries without Responses will be captured omitting the response data. Responses without queries will be captured omitting the Query data (but using the Question section from the response, if present, as an identifying QNAME).

    * Rationale: A Query and Response represents the basic level of a client's interaction with the server. Also, combining the Query and Response into one item often reduces storage requirements due to commonality in the data of the two messages.

    In the context of generating a C-DNS file it is assumed that only those DNS payloads which can be parsed to produce a well-formed DNS message are stored in the C-DNS format and that all other messages will be (optionally) recorded as malformed messages. Parsing a well-formed message means as a minimum:

    * The packet has a well-formed 12 byte DNS Header with a recognised OPCODE.

    * The section counts are consistent with the section contents.

    * All of the resource records can be fully parsed.

2. All top level fields in each Q/R data item will be optional.

    * Rationale: Different users will have different requirements for data to be available for analysis. Users with minimal requirements should not have to pay the cost of recording full data, though this will limit the ability to perform certain kinds of data analysis and also to reconstruct packet captures. For example, omitting the resource records from a Response will reduce the C-DNS file size; in principle responses can be synthesized if there is enough context.

3. Multiple Q/R data items will be collected into blocks in the format. Common data in a block will be abstracted and referenced from individual Q/R data items by indexing. The maximum number of Q/R data items in a block will be configurable.

   * Rationale: This blocking and indexing provides a significant reduction in the volume of file data generated. Although this introduces complexity, it provides compression of the data that makes use of knowledge of the DNS message structure.

   * It is anticipated that the files produced can be subject to further compression using general purpose compression tools. Measurements show that blocking significantly reduces the CPU required to perform such strong compression. See Appendix C.2.

   * Examples of commonality between DNS messages are that in most cases the QUESTION RR is the same in the query and response, and that there is a finite set of query signatures (based on a subset of attributes). For many authoritative servers there is very likely to be a finite set of responses that are generated, of which a large number are NXDOMAIN.

4. Traffic metadata can optionally be included in each block. Specifically, counts of some types of non-DNS packets (e.g. ICMP, TCP resets) sent to the server may be of interest.

5. The wire format content of malformed DNS messages may optionally be recorded.

   * Rationale: Any structured capture format that does not capture the DNS payload byte for byte will be limited to some extent in that it cannot represent malformed DNS messages. Only those messages that can be fully parsed and transformed into the structured format can be fully represented. Note, however, this can result in rather misleading statistics. For example, a malformed query which cannot be represented in the C-DNS format will lead to the (well formed) DNS responses with error code FORMERR appearing as 'unmatched'. Therefore it can greatly aid downstream analysis to have the wire format of the malformed DNS messages available directly in the C-DNS file.

5. Choice of CBOR

This document presents a detailed format description using CBOR, the Concise Binary Object Representation defined in [RFC7049].

The choice of CBOR was made taking a number of factors into account.

- CBOR is a binary representation, and thus is economical in storage space.
- Other binary representations were investigated, and whilst all had attractive features, none had a significant advantage over CBOR. See Appendix C for some discussion of this.
- CBOR is an IETF standard and familiar to IETF participants. It is based on the now-common ideas of lists and objects, and thus requires very little familiarization for those in the wider industry.
- CBOR is a simple format, and can easily be implemented from scratch if necessary. More complex formats require library support which may present problems on unusual platforms.
- CBOR can also be easily converted to text formats such as JSON ([RFC8259]) for debugging and other human inspection requirements.
- CBOR data schemas can be described using CDDL [I-D.ietf-cbor-cddl].

6. C-DNS format conceptual overview

The following figures show purely schematic representations of the C-DNS format to convey the high-level structure of the C-DNS format. Section 7 provides a detailed discussion of the CBOR representation and individual elements.

Figure showing the C-DNS format (PNG) [1]

Figure showing the C-DNS format (SVG) [2]

Figure showing the Query/Response data item and Block Tables format (PNG) [3]

Figure showing the Query/Response item and Block Tables format (SVG) [4]

A C-DNS file begins with a file header containing a File Type Identifier and a File Preamble. The File Preamble contains information on the file Format Version and an array of Block Parameters items (the contents of which include Collection and Storage Parameters used for one or more blocks).

The file header is followed by a series of data Blocks.

A Block consists of a Block Preamble item, some Block Statistics for the traffic stored within the Block and then various arrays of common data collectively called the Block Tables. This is then followed by an array of the Query/Response data items detailing the queries and responses stored within the Block. The array of Query/Response data items is in turn followed by the Address/Event Counts data items (an array of per-client counts of particular IP events) and then Malformed Message data items (an array of malformed messages that stored in the Block).

The exact nature of the DNS data will affect what block size is the best fit, however sample data for a root server indicated that block sizes up to 10,000 Q/R data items give good results. See Appendix C.6 for more details.

6.1. Block Parameters

The details of the Block Parameters items are not shown in the diagrams but are discussed here for context.

An array of Block Parameters items is stored in the File Preamble (with a minimum of one item at index 0); a Block Parameters item consists of a collection of Storage and Collection Parameters that applies to any given Block. An array is used in order to support use cases such as wanting to merge C-DNS files from different sources. The Block Preamble item then contains an optional index for the Block Parameters item that applies for that Block; if not present the index defaults to 0. Hence, in effect, a global Block Parameters item is defined which can then be overridden per Block.

6.2. Storage Parameters

The Block Parameters item includes a Storage Parameters item - this contains information about the specific data fields stored in the C-DNS file.

These parameters include:

- The sub-second timing resolution used by the data.
- Information (hints) on which optional data are omitted. See Section 6.2.1.
- Recorded OPCODES and RR types. See Section 6.2.2.
- Flags indicating, for example, whether the data is sampled or anonymised. See Section 6.2.3.

- Client and server IPv4 and IPv6 address prefixes. See Section 6.2.4

6.2.1. Optional data items

To enable implementations to store data to their precise requirements in as space-efficient manner as possible, all fields in the following arrays are optional:

- Query/Response
- Query Signature
- Malformed messages

In other words, an implementation can choose to omit any data item that is not required for its use case. In addition, implementations may be configured to not record all RRs, or only record messages with certain OPCODES.

This does, however, mean that a consumer of a C-DNS file faces two problems:

1. How can it quickly determine if a file definitely does not contain the data items it requires to complete a particular task (e.g. reconstructing query traffic or performing a specific piece of data analysis)?
2. How can it determine if a data item is not present because it was:
    * explicitly not recorded or
    * the data item was not available/present.

For example, capturing C-DNS data from within a nameserver implementation makes it unlikely that the Client Hoplimit can be recorded. Or, if there is no query ARCount recorded and no query OPT RDATA recorded, is that because no query contained an OPT RR, or because that data was not stored?

The Storage Parameters therefore also contains a Storage Hints item which specifies which items the encoder of the file omits from the stored data. An implementation decoding that file can then use these to quickly determine whether the input data is rich enough for its needs.

6.2.2. Optional RRs and OPCODEs

Also included in the Storage Parameters are explicit arrays listing the RR types and the OPCODEs to be recorded. These remove any ambiguity over whether messages containing particular OPCODEs or RR types are not present because they did not occur, or because the implementation is not configured to record them.

In the case of OPCODEs, for a message to be fully parsable, the OPCODE must be known to the collecting implementation. Any message with an OPCODE unknown to the collecting implementation cannot be validated as correctly formed, and so must be treated as malformed. Messages with OPCODES known to the recording application but not listed in the Storage Parameters are discarded (regardless of whether they are malformed or not).

In the case of RR records, each record in a message must be fully parsable, including parsing the record RDATA, as otherwise the message cannot be validated as correctly formed. Any RR record with an RR type not known to the collecting implementation cannot be validated as correctly formed, and so must be treated as malformed.

Once a message is correctly parsed, an implementation is free to record only a subset of the RR records present.

6.2.3. Storage flags

The Storage Parameters contains flags that can be used to indicate if:

- the data is anonymised,
- the data is produced from sample data, or
- names in the data have been normalised (converted to uniform case).

The Storage Parameters also contains optional fields holding details of the sampling method used and the anonymisation method used. It is RECOMMENDED these fields contain URIs pointing to resources describing the methods used.

6.2.4. IP Address storage

The format contains fields to indicate if only IP prefixes were stored. If IP address prefixes are given, only the prefix bits of addresses are stored. For example, if a client IPv4 prefix of 16 is

specified, a client address of 192.0.2.1 will be stored as 0xc000 (192.0), reducing address storage space requirements.

7. C-DNS format detailed description

The CDDL definition for the C-DNS format is given in Appendix A.

7.1. Map quantities and indexes

All map keys are integers with values specified in the CDDL. String keys would significantly bloat the file size.

All key values specified are positive integers under 24, so their CBOR representation is a single byte. Positive integer values not currently used as keys in a map are reserved for use in future standard extensions.

Implementations may choose to add additional implementation-specific entries to any map. Negative integer map keys are reserved for these values. Key values from -1 to -24 also have a single byte CBOR representation, so such implementation-specific extensions are not at any space efficiency disadvantage.

An item described as an index is the index of the data item in the referenced array. Indexes are 0-based.

7.2. Tabular representation

The following sections present the C-DNS specification in tabular format with a detailed description of each item.

In all quantities that contain bit flags, bit 0 indicates the least significant bit, i.e. flag "n" in quantity "q" is on if "(q & (1 << n)) != 0".

For the sake of readability, all type and field names defined in the CDDL definition are shown in double quotes. Type names are by convention camel case (e.g. "BlockTable"), field names are lower-case with hyphens (e.g. "block-tables").

For the sake of brevity, the following conventions are used in the tables:

- The column O marks whether items in a map are optional.
  - O - Optional. The item may be omitted.
  - M - Mandatory. The item must be present.

- The column T gives the CBOR data type of the item.
  - U - Unsigned integer
  - I - Signed integer
  - B - Byte string
  - T - Text string
  - M - Map
  - A - Array

In the case of maps and arrays, more information on the type of each value, include the CDDL definition name if applicable, is given in the description.

7.3. "File"

A C-DNS file has an outer structure "File", a map that contains the following:

Field	O	T	Description
file-type-id	M	T	String "C-DNS" identifying the file type.
file-preamble	M	M	Version and parameter information for the whole file. Map of type "FilePreamble", see Section 7.4.
file-blocks	M	A	Array of items of type "Block", see Section 7.5. The array may be empty if the file contains no data.

7.4. "FilePreamble"

Information about data in the file. A map containing the following:

Field	O	T	Description
major-format-version	M	U	Unsigned integer '1'. The major version of format used in file.
minor-format-version	M	U	Unsigned integer '0'. The minor version of format used in file.
private-version	O	U	Version indicator available for private use by implementations.
block-parameters	M	A	Array of items of type "BlockParameters", see Section 7.4.1. The array must contain at least one entry. (The "block-parameters-index" item in each "BlockPreamble" indicates which array entry applies to that "Block".)

7.4.1. "BlockParameters"

Parameters relating to data storage and collection which apply to one or more items of type "Block". A map containing the following:

Field	O	T	Description
storage-parameters	M	M	Parameters relating to data storage in a "Block" item. Map of type "StorageParameters", see Section 7.4.1.1.
collection-parameters	O	M	Parameters relating to collection of the data in a "Block" item. Map of type "CollectionParameters", see Section 7.4.2.

7.4.1.1. "StorageParameters"

Parameters relating to how data is stored in the items of type "Block". A map containing the following:

Field	O	T	Description
ticks-per-second	M	U	Sub-second timing is recorded in ticks. This specifies the number of ticks in a second.
max-block-items	M	U	The maximum number of items stored in any of the arrays in a "Block" item (Q/R items, address event counts or malformed messages). An indication to a decoder of the resources needed to process the file.
storage-hints	M	M	Collection of hints as to which fields are omitted in the arrays that have optional fields. Map of type "StorageHints", see Section 7.4.1.1.1.
opcodes	M	A	Array of OPCODES (unsigned integers) recorded by the collection implementation. See Section 6.2.2.
rr-types	M	A	Array of RR types (unsigned integers) recorded by the collection implementation. See Section 6.2.2.
storage-flags	O	U	Bit flags indicating attributes of stored data. Bit 0. 1 if the data has been anonymised. Bit 1. 1 if the data is sampled data. Bit 2. 1 if the names have been normalised (converted to uniform case).
client-address -prefix-ipv4	O	U	IPv4 client address prefix length. If specified, only the address prefix bits are stored.
client-address -prefix-ipv6	O	U	IPv6 client address prefix length. If specified, only the address prefix bits are stored.
server-address -prefix-ipv4	O	U	IPv4 server address prefix length. If specified, only the address prefix bits are stored.
server-address	O	U	IPv6 server address prefix length. If

-prefix-ipv6			specified, only the address prefix bits are stored.
sampling-method	O	T	Information on the sampling method used. See Section 6.2.3.
anonymisation -method	O	T	Information on the anonymisation method used. See Section 6.2.3.

7.4.1.1.1. "StorageHints"

An indicator of which fields the collecting implementation omits in the arrays with optional fields. A map containing the following:

Field	O	T	Description
query-response -hints	M	U	Hints indicating which "QueryResponse" fields are omitted, see section Section 7.6. If the field is omitted the bit is unset. Bit 0. time-offset Bit 1. client-address-index Bit 2. client-port Bit 3. transaction-id Bit 4. qr-signature-index Bit 5. client-hoplimit Bit 6. response-delay Bit 7. query-name-index Bit 8. query-size Bit 9. response-size Bit 10. response-processing-data Bit 11. query-question-sections Bit 12. query-answer-sections Bit 13. query-authority-sections Bit 14. query-additional-sections Bit 15. response-answer-sections Bit 16. response-authority-sections Bit 17. response-additional-sections
query-response -signature-hints	M	U	Hints indicating which "QueryResponseSignature" fields are omitted, see section Section 7.5.3.2. If the field is omitted the bit is unset. Bit 0. server-address Bit 1. server-port

			Bit 2. qr-transport-flags
			Bit 3. qr-type
			Bit 4. qr-sig-flags
			Bit 5. query-opcode
			Bit 6. dns-flags
			Bit 7. query-rcode
			Bit 8. query-class-type
			Bit 9. query-qdcount
			Bit 10. query-ancount
			Bit 11. query-nscount
			Bit 12. query-arcount
			Bit 13. query-edns-version
			Bit 14. query-udp-size
			Bit 15. query-opt-rdata
			Bit 16. response-rcode
rr-hints	M	U	Hints indicating which optional "RR" fields are omitted, see Section 7.5.3.4. If the field is omitted the bit is unset. Bit 0. ttl Bit 1. rdata-index
other-data-hints	M	U	Hints indicating which other data types are are omitted. If the data type is are omitted the bit is unset. Bit 0. malformed-messages Bit 1. address-event-counts

7.4.2. "CollectionParameters"

Parameters relating to how data in the file was collected.

These parameters have no default. If they do not appear, nothing can be inferred about their value.

A map containing the following items:

Field	O	T	Description
query-timeout	O	U	To be matched with a query, a response must arrive within this number of seconds.
skew-timeout	O	U	The network stack may report a response before the corresponding query. A response is not considered to be missing a query until after this many micro-seconds.
snaplen	O	U	Collect up to this many bytes per packet.
promisc	O	U	1 if promiscuous mode was enabled on the interface, 0 otherwise.
interfaces	O	A	Array of identifiers (of type text string) of the interfaces used for collection.
server-addresses	O	A	Array of server collection IP addresses (of type byte string). Hint for downstream analysers; does not affect collection.
vlan-ids	O	A	Array of identifiers (of type unsigned integer) of VLANs selected for collection.
filter	O	T	"tcpdump" [pcap] style filter for input.
generator-id	O	T	String identifying the collection method.
host-id	O	T	String identifying the collecting host. Empty if converting an existing packet capture file.

7.5. "Block"

Container for data with common collection and and storage parameters. A map containing the following:

Field	O	T	Description
block-preamble	M	M	Overall information for the "Block" item. Map of type "BlockPreamble", see Section 7.5.1.
block-statistics	O	M	Statistics about the "Block" item. Map of type "BlockStatistics", see Section 7.5.2.
block-tables	O	M	The arrays containing data referenced by individual "QueryResponse" or "MalformedMessage" items. Map of type "BlockTables", see Section 7.5.3.
query-responses	O	A	Details of individual DNS Q/R data items. Array of items of type "QueryResponse", see Section 7.6. If present, the array must not be empty.
address-event -counts	O	A	Per client counts of ICMP messages and TCP resets. Array of items of type "AddressEventCount", see Section 7.7. If present, the array must not be empty.
malformed-messages	O	A	Details of malformed DNS messages. Array of items of type "MalformedMessage", see Section 7.8. If present, the array must not be empty.

7.5.1. "BlockPreamble"

Overall information for a "Block" item. A map containing the following:

Field	O	T	Description
earliest-time	O	A	A timestamp (2 unsigned integers, "Timestamp") for the earliest record in the "Block" item. The first integer is the number of seconds since the Posix epoch ("time_t"). The second integer is the number of ticks since the start of the second. This timestamp can only be omitted if all block items containing a time offset from the start of the block also omit that time offset.
block-parameters-index	O	U	The index of the item in the "block-parameters" array (in the "file-premable" item) applicable to this block. If not present, index 0 is used. See Section 7.4.1.

7.5.2. "BlockStatistics"

Basic statistical information about a "Block" item. A map containing the following:

Field	O	T	Description
processed-messages	O	U	Total number of DNS messages processed from the input traffic stream during collection of data in this "Block" item.
qr-data-items	O	U	Total number of Q/R data items in this "Block" item.
unmatched-queries	O	U	Number of unmatched queries in this "Block" item.
unmatched-responses	O	U	Number of unmatched responses in this "Block" item.
discarded-opcode	O	U	Number of DNS messages processed from the input traffic stream during collection of data in this "Block" item but not recorded because their OPCODE is not in the list to be collected.
malformed-items	O	U	Number of malformed messages found in input for this "Block" item.

7.5.3. "BlockTables"

Arrays containing data referenced by individual "QueryResponse" or "MalformedMessage" items in this "Block". Each element is an array which, if present, must not be empty.

An item in the "qlist" array contains indexes to values in the "qrr" array. Therefore, if "qlist" is present, "qrr" must also be present. Similarly, if "rrlist" is present, "rr" must also be present.

The map contains the following items:

Field	O	T	Description
ip-address	O	A	Array of IP addresses, in network byte order (of type byte string). If client or server address prefixes are set, only the address prefix bits are stored. Each string is therefore up

			to 4 bytes long for an IPv4 address, or up to 16 bytes long for an IPv6 address. See Section 7.4.1.1.
classtype	O	A	Array of RR class and type information. Type is "ClassType", see Section 7.5.3.1.
name-rdata	O	A	Array where each entry is the contents of a single NAME or RDATA (of type byte string). Note that NAMEs, and labels within RDATA contents, are full domain names or labels; no DNS style name compression is used on the individual names/labels within the format.
qr-sig	O	A	Array Q/R data item signatures. Type is "QueryResponseSignature", see Section 7.5.3.2.
qlist	O	A	Array of type "QuestionList". A "QuestionList" is an array of unsigned integers, indexes to "Question" items in the "qrr" array.
qrr	O	A	Array of type "Question". Each entry is the contents of a single question, where a question is the second or subsequent question in a query. See Section 7.5.3.3.
rrlist	O	A	Array of type "RRList". An "RRList" is an array of unsigned integers, indexes to "RR" items in the "rr" array.
rr	O	A	Array of type "RR". Each entry is the contents of a single RR. See Section 7.5.3.4.
malformed-message-data	O	A	Array of the contents of malformed messages. Array of type "MalformedMessageData", see Section 7.5.3.5.

7.5.3.1. "ClassType"

RR class and type information. A map containing the following:

Field	O	T	Description
type	M	U	TYPE value.
class	M	U	CLASS value.

7.5.3.2. "QueryResponseSignature"

Elements of a Q/R data item that are often common between multiple individual Q/R data items. A map containing the following:

Field	O	T	Description
server-address-index	O	U	The index in the item in the "ip-address" array of the server IP address. See Section 7.5.3.
server-port	O	U	The server port.
qr-transport-flags	O	U	Bit flags describing the transport used to service the query. Bit 0. IP version. 0 if IPv4, 1 if IPv6 Bit 1-4. Transport. 4 bit unsigned value where 0 = UDP, 1 = TCP, 2 = TLS, 3 = DTLS. Values 4-15 are reserved for future use. Bit 5. 1 if trailing bytes in query packet. See Section 10.2.
qr-type	O	U	Type of Query/Response transaction. 0 = Stub. A query from a stub resolver. 1 = Client. An incoming query to a recursive resolver. 2 = Resolver. A query sent from a recursive resolver to an authorative resolver. 3 = Authorative. A query to an authorative resolver. 4 = Forwarder. A query sent from a

			recursive resolver to an upstream recursive resolver. 5 = Tool. A query sent to a server by a server tool.
qr-sig-flags	O	U	Bit flags explicitly indicating attributes of the message pair represented by this Q/R data item (not all attributes may be recorded or deducible). Bit 0. 1 if a Query was present. Bit 1. 1 if a Response was present. Bit 2. 1 if a Query was present and it had an OPT Resource Record. Bit 3. 1 if a Response was present and it had an OPT Resource Record. Bit 4. 1 if a Query was present but had no Question. Bit 5. 1 if a Response was present but had no Question (only one query-name-index is stored per Q/R item).
query-opcode	O	U	Query OPCODE.
qr-dns-flags	O	U	Bit flags with values from the Query and Response DNS flags. Flag values are 0 if the Query or Response is not present. Bit 0. Query Checking Disabled (CD). Bit 1. Query Authenticated Data (AD). Bit 2. Query reserved (Z). Bit 3. Query Recursion Available (RA). Bit 4. Query Recursion Desired (RD). Bit 5. Query TrunCation (TC). Bit 6. Query Authoritative Answer (AA). Bit 7. Query DNSSEC answer OK (DO). Bit 8. Response Checking Disabled (CD). Bit 9. Response Authenticated Data (AD). Bit 10. Response reserved (Z). Bit 11. Response Recursion Available (RA). Bit 12. Response Recursion Desired (RD).

			Bit 13. Response TrunCation (TC). Bit 14. Response Authoritative Answer (AA).
query-rcode	O	U	Query RCODE. If the Query contains OPT, this value incorporates any EXTENDED_RCODE_VALUE.
query-classtype-index	O	U	The index to the item in the the "classtype" array of the CLASS and TYPE of the first Question. See Section 7.5.3.
query-qd-count	O	U	The QDCOUNT in the Query, or Response if no Query present.
query-an-count	O	U	Query ANCOUNT.
query-ns-count	O	U	Query NSCOUNT.
query-ar-count	O	U	Query ARCOUNT.
edns-version	O	U	The Query EDNS version.
udp-buf-size	O	U	The Query EDNS sender’s UDP payload size.
opt-rdata-index	O	U	The index in the "name-rdata" array of the OPT RDATA. See Section 7.5.3.
response-rcode	O	U	Response RCODE. If the Response contains OPT, this value incorporates any EXTENDED_RCODE_VALUE.

7.5.3.3. "Question"

Details on individual Questions in a Question section. A map containing the following:

Field	O	T	Description
name-index	M	U	The index in the "name-rdata" array of the QNAME. See Section 7.5.3.
classtype-index	M	U	The index in the "classtype" array of the CLASS and TYPE of the Question. See Section 7.5.3.

7.5.3.4. "RR"

Details on individual Resource Records in RR sections. A map containing the following:

Field	O	T	Description
name-index	M	U	The index in the "name-rdata" array of the NAME. See Section 7.5.3.
classtype-index	M	U	The index in the "classtype" array of the CLASS and TYPE of the RR. See Section 7.5.3.
ttl	O	U	The RR Time to Live.
rdata-index	O	U	The index in the "name-rdata" array of the RR RDATA. See Section 7.5.3.

7.5.3.5. "MalformedMessageData"

Details on malformed message items in this "Block" item. A map containing the following:

Field	O	T	Description
server-address-index	O	U	The index in the "ip-address" array of the server IP address. See Section 7.5.3.
server-port	O	U	The server port.
mm-transport-flags	O	U	Bit flags describing the transport used to service the query. Bit 0 is the least significant bit. Bit 0. IP version. 0 if IPv4, 1 if IPv6 Bit 1-4. Transport. 4 bit unsigned value where 0 = UDP, 1 = TCP, 2 = TLS, 3 = DTLS. Values 4-15 are reserved for future use.
mm-payload	O	B	The payload (raw bytes) of the DNS message.

7.6. "QueryResponse"

Details on individual Q/R data items.

Note that there is no requirement that the elements of the "query-responses" array are presented in strict chronological order.

A map containing the following items:

Field	O	T	Description
time-offset	O	U	Q/R timestamp as an offset in ticks from "earliest-time". The timestamp is the timestamp of the Query, or the Response if there is no Query.
client-address-index	O	U	The index in the "ip-address" array of the client IP address. See Section 7.5.3.
client-port	O	U	The client port.
transaction-id	O	U	DNS transaction identifier.

qr-signature-index	O	U	The index in the "qr-sig" array of the "QueryResponseSignature" item. See Section 7.5.3.
client-hoplimit	O	U	The IPv4 TTL or IPv6 Hoplimit from the Query packet.
response-delay	O	I	The time difference between Query and Response, in ticks. Only present if there is a query and a response. The delay can be negative if the network stack/capture library returns packets out of order.
query-name-index	O	U	The index in the "name-rdata" array of the item containing the QNAME for the first Question. See Section 7.5.3.
query-size	O	U	DNS query message size (see below).
response-size	O	U	DNS query message size (see below).
response-processing-data	O	M	Data on response processing. Map of type "ResponseProcessingData", see Section 7.6.1.
query-extended	O	M	Extended Query data. Map of type "QueryResponseExtended", see Section 7.6.2.
response-extended	O	M	Extended Response data. Map of type "QueryResponseExtended", see Section 7.6.2.

The "query-size" and "response-size" fields hold the DNS message size. For UDP this is the size of the UDP payload that contained the DNS message. For TCP it is the size of the DNS message as specified in the two-byte message length header. Trailing bytes in UDP queries are routinely observed in traffic to authoritative servers and this value allows a calculation of how many trailing bytes were present.

7.6.1. "ResponseProcessingData"

Information on the server processing that produced the response. A map containing the following:

Field	O	T	Description
bailiwick-index	O	U	The index in the "name-rdata" array of the owner name for the response bailiwick. See Section 7.5.3.
processing-flags	O	U	Flags relating to response processing. Bit 0. 1 if the response came from cache.

7.6.2. "QueryResponseExtended"

Extended data on the Q/R data item.

Each item in the map is present only if collection of the relevant details is configured.

A map containing the following items:

Field	O	T	Description
question-index	O	U	The index in the "qlist" array of the entry listing any second and subsequent Questions in the Question section for the Query or Response. See Section 7.5.3.
answer-index	O	U	The index in the "rrlist" array of the entry listing the Answer Resource Record sections for the Query or Response. See Section 7.5.3.
authority-index	O	U	The index in the "rrlist" array of the entry listing the Authority Resource Record sections for the Query or Response. See Section 7.5.3.
additional-index	O	U	The index in the "rrlist" array of the entry listing the Additional Resource Record sections for the Query or Response. See Section 7.5.3. Note that Query OPT RR data can be optionally stored in the QuerySignature.

7.7. "AddressEventCount"

Counts of various IP related events relating to traffic with individual client addresses. A map containing the following:

Field	O	T	Description
ae-type	M	U	The type of event. The following events types are currently defined: 0. TCP reset. 1. ICMP time exceeded. 2. ICMP destination unreachable. 3. ICMPv6 time exceeded. 4. ICMPv6 destination unreachable. 5. ICMPv6 packet too big.
ae-code	O	U	A code relating to the event.
ae-address-index	M	U	The index in the "ip-address" array of the client address. See Section 7.5.3.
ae-count	M	U	The number of occurrences of this event during the block collection period.

7.8. "MalformedMessage"

Details of malformed messages. A map containing the following:

Field	O	T	Description
time-offset	O	U	Message timestamp as an offset in ticks from "earliest-time".
client-address-index	O	U	The index in the "ip-address" array of the client IP address. See Section 7.5.3.
client-port	O	U	The client port.
message-data-index	O	U	The index in the "malformed-message-data" array of the message data for this message. See Section 7.5.3.

8. C-DNS to PCAP

It is possible to re-construct PCAP files from the C-DNS format in a lossy fashion. Some of the issues with reconstructing both the DNS payload and the full packet stream are outlined here.

The reconstruction depends on whether or not all the optional sections of both the query and response were captured in the C-DNS file. Clearly, if they were not all captured, the reconstruction will be imperfect.

Even if all sections of the response were captured, one cannot reconstruct the DNS response payload exactly due to the fact that some DNS names in the message on the wire may have been compressed. Section 8.1 discusses this is more detail.

Some transport information is not captured in the C-DNS format. For example, the following aspects of the original packet stream cannot be re-constructed from the C-DNS format:

- IP fragmentation
- TCP stream information:
  - Multiple DNS messages may have been sent in a single TCP segment
  - A DNS payload may have be split across multiple TCP segments
  - Multiple DNS messages may have be sent on a single TCP session
- Malformed DNS messages if the wire format is not recorded
- Any Non-DNS messages that were in the original packet stream e.g. ICMP

Simple assumptions can be made on the reconstruction: fragmented and DNS-over-TCP messages can be reconstructed into single packets and a single TCP session can be constructed for each TCP packet.

Additionally, if malformed messages and Non-DNS packets are captured separately, they can be merged with packet captures reconstructed from C-DNS to produce a more complete packet stream.

8.1. Name compression

All the names stored in the C-DNS format are full domain names; no DNS style name compression is used on the individual names within the format. Therefore when reconstructing a packet, name compression must be used in order to reproduce the on the wire representation of the packet.

[RFC1035] name compression works by substituting trailing sections of a name with a reference back to the occurrence of those sections earlier in the message. Not all name server software uses the same algorithm when compressing domain names within the responses. Some attempt maximum recompression at the expense of runtime resources, others use heuristics to balance compression and speed and others use different rules for what is a valid compression target.

This means that responses to the same question from different name server software which match in terms of DNS payload content (header, counts, RRs with name compression removed) do not necessarily match byte-for-byte on the wire.

Therefore, it is not possible to ensure that the DNS response payload is reconstructed byte-for-byte from C-DNS data. However, it can at least, in principle, be reconstructed to have the correct payload length (since the original response length is captured) if there is enough knowledge of the commonly implemented name compression algorithms. For example, a simplistic approach would be to try each algorithm in turn to see if it reproduces the original length, stopping at the first match. This would not guarantee the correct algorithm has been used as it is possible to match the length whilst still not matching the on the wire bytes but, without further information added to the C-DNS data, this is the best that can be achieved.

Appendix B presents an example of two different compression algorithms used by well-known name server software.

9. Data collection

This section describes a non-normative proposed algorithm for the processing of a captured stream of DNS queries and responses and production of a stream of query/response items, matching queries/responses where possible.

For the purposes of this discussion, it is assumed that the input has been pre-processed such that:

1. All IP fragmentation reassembly, TCP stream reassembly, and so on, has already been performed.

2. Each message is associated with transport metadata required to generate the Primary ID (see Section 9.2.1).

3. Each message has a well-formed DNS header of 12 bytes and (if present) the first Question in the Question section can be parsed to generate the Secondary ID (see below). As noted earlier, this requirement can result in a malformed query being removed in the pre-processing stage, but the correctly formed response with RCODE of FORMERR being present.

DNS messages are processed in the order they are delivered to the implementation.

It should be noted that packet capture libraries do not necessarily provide packets in strict chronological order. This can, for example, arise on multi-core platforms where packets arriving at a network device are processed by different cores. On systems where this behaviour has been observed, the timestamps associated with each packet are consistent; queries always have a timestamp prior to the response timestamp. However, the order in which these packets appear in the packet capture stream is not necessarily strictly choronological; a response can appear in the capture stream before the query that provoked the response. For this discussion, this non-chronological delivery is termed "skew".

In the presence of skew, a response packets can arrive for matching before the corresponding query. To avoid generating false instances of responses without a matching query, and queries without a matching response, the matching algorithm must take account of the possibility of skew.

9.1. Matching algorithm

A schematic representation of the algorithm for matching Q/R data items is shown in the following diagram:

Figure showing the Query/Response matching algorithm format (PNG) [5]

Figure showing the Query/Response matching algorithm format (SVG) [6]

Further details of the algorithm are given in the following sections.

9.2. Message identifiers

9.2.1. Primary ID (required)

A Primary ID is constructed for each message. It is composed of the following data:

1. Source IP Address
2. Destination IP Address
3. Source Port
4. Destination Port
5. Transport
6. DNS Message ID

9.2.2. Secondary ID (optional)

If present, the first Question in the Question section is used as a secondary ID for each message. Note that there may be well formed DNS queries that have a QDCOUNT of 0, and some responses may have a QDCOUNT of 0 (for example, responses with RCODE=FORMERR or NOTIMP). In this case the secondary ID is not used in matching.

9.3. Algorithm parameters

1. Query timeout, QT. A query arrives with timestamp t1. If no response matching that query has arrived before other input arrives timestamped later than (t1 + QT), a query/response item containing only a query item is recorded. The query timeout value is typically of the order of 5 seconds.

2. Skew timeout, ST. A response arrives with timestamp t2. If a response has not been matched by a query before input arrives timestamped later than (t2 + ST), a query/response item containing only a response is recorded. The skew timeout value is typically a few microseconds.

9.4. Algorithm requirements

The algorithm is designed to handle the following input data:

1. Multiple queries with the same Primary ID (but different Secondary ID) arriving before any responses for these queries are seen.

2. Multiple queries with the same Primary and Secondary ID arriving before any responses for these queries are seen.

3. Queries for which no later response can be found within the specified timeout.

4. Responses for which no previous query can be found within the specified timeout.

9.5. Algorithm limitations

For cases 1 and 2 listed in the above requirements, it is not possible to unambiguously match queries with responses. This algorithm chooses to match to the earliest query with the correct Primary and Secondary ID.

9.6. Workspace

A FIFO structure is used to hold the Q/R data items during processing. A secondary responses FIFO holds responses awaiting matching queries.

9.7. Output

The output is a list of Q/R data items. Both the Query and Response elements are optional in these items, therefore Q/R data items have one of three types of content:

1. A matched pair of query and response messages

2. A query message with no response

3. A response message with no query

The timestamp of a list item is that of the query for cases 1 and 2 and that of the response for case 3.

9.8. Post processing

When ending capture, all items in the responses FIFO are timed out immediately, generating response-only entries to the Q/R data item FIFO. These and all other remaining entries in the Q/R data item FIFO should be treated as timed out queries.

10. Implementation guidance

Whilst this document makes no specific recommendations with respect to Canonical CBOR (see Section 3.9 of [RFC7049]) the following guidance may be of use to implementors.

Adherence to the first two rules given in Section 3.9 of [RFC7049] will minimise file sizes.

Adherence to the last two rules given in Section 3.9 of [RFC7049] for all maps and arrays would unacceptably constrain implementations, for example, in the use case of real-time data collection in constrained environments.

10.1. Optional data

When decoding C-DNS data some of the items required for a particular function that the consumer wishes to perform may be missing. Consumers should consider providing configurable default values to be used in place of the missing values in their output.

10.2. Trailing bytes

A DNS query message in a UDP or TCP payload can be followed by some additional (spurious) bytes, which are not stored in C-DNS.

When DNS traffic is sent over TCP, each message is prefixed with a two byte length field which gives the message length, excluding the two byte length field. In this context, trailing bytes can occur in two circumstances with different results:

1. The number of bytes consumed by fully parsing the message is less than the number of bytes given in the length field (i.e. the length field is incorrect and too large). In this case, the surplus bytes are considered trailing bytes in an analogous manner to UDP and recorded as such. If only this case occurs it is possible to process a packet containing multiple DNS messages where one or more has trailing bytes.

2. There are surplus bytes between the end of a well-formed message and the start of the length field for the next message. In this case the first of the surplus bytes will be processed as the first byte of the next length field, and parsing will proceed from there, almost certainly leading to the next and any subsequent messages in the packet being considered malformed. This will not generate a trailing bytes record for the processed well-formed message.

10.3. Limiting collection of RDATA

Implementations should consider providing a configurable maximum RDATA size for capture, for example, to avoid memory issues when confronted with large XFR records.

11. Implementation status

[Note to RFC Editor: please remove this section and reference to [RFC7942] prior to publication.]

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 [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 [RFC7942], "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".

11.1. DNS-STATS Compactor

ICANN/Sinodun IT have developed an open source implementation called DNS-STATS Compactor. The Compactor is a suite of tools which can capture DNS traffic (from either a network interface or a PCAP file) and store it in the Compacted-DNS (C-DNS) file format. PCAP files for the captured traffic can also be reconstructed. See Compactor [7].

This implementation:

- covers the whole of the specification described in the -03 draft with the exception of support for malformed messages and pico second time resolution. (Note: this implementation does allow malformed messages to be recorded separately in a PCAP file).

- is released under the Mozilla Public License Version 2.0.

- has a users mailing list available, see dns-stats-users [8].

There is also some discussion of issues encountered during development available at Compressing Pcap Files [9] and Packet Capture [10].

This information was last updated on 3rd of May 2018.

12. IANA considerations

None

13. Security considerations

Any control interface MUST perform authentication and encryption.

Any data upload MUST be authenticated and encrypted.

14. Acknowledgements

The authors wish to thank CZ.NIC, in particular Tomas Gavenciak, for many useful discussions on binary formats, compression and packet matching. Also Jan Vcelak and Wouter Wijngaards for discussions on name compression and Paul Hoffman for a detailed review of the document and the C-DNS CDDL.

Thanks also to Robert Edmonds, Jerry Lundstroem, Richard Gibson, Stephane Bortzmeyer and many other members of DNSOP for review.

Also, Miek Gieben for mmark [11]

15. Changelog

draft-ietf-dnsop-dns-capture-format-07

- Resolve outstanding questions and TODOs
- Make RR RDATA optional
- Update matching diagram and explain skew
- Add count of discarded messages to block statistics
- Editorial clarifications and improvements

draft-ietf-dnsop-dns-capture-format-06

- Correct BlockParameters type to map

- Make RR ttl optional
- Add storage flag indicating name normalisation
- Add storage parameter fields for sampling and anonymisation methods
- Editorial clarifications and improvements

draft-ietf-dnsop-dns-capture-format-05

- Make all data items in Q/R, QuerySignature and Malformed Message arrays optional
- Re-structure the FilePreamble and ConfigurationParameters into BlockParameters
- BlockParameters has separate Storage and Collection Parameters
- Storage Parameters includes information on what optional fields are present, and flags specifying anonymisation or sampling
- Addresses can now be stored as prefixes.
- Switch to using a variable sub-second timing granularity
- Add response bailiwick and query response type
- Add specifics of how to record malformed messages
- Add implementation guidance
- Improve terminology and naming consistency

draft-ietf-dnsop-dns-capture-format-04

- Correct query-d0 to query-do in CDDL
- Clarify that map keys are unsigned integers
- Add Type to Class/Type table
- Clarify storage format in section 7.12

draft-ietf-dnsop-dns-capture-format-03

- Added an Implementation Status section

draft-ietf-dnsop-dns-capture-format-02

- Update qr_data_format.png to match CDDL
- Editorial clarifications and improvements

draft-ietf-dnsop-dns-capture-format-01

- Many editorial improvements by Paul Hoffman
- Included discussion of malformed message handling
- Improved Appendix C on Comparison of Binary Formats
- Now using C-DNS field names in the tables in section 8
- A handful of new fields included (CDDL updated)
- Timestamps now include optional picoseconds
- Added details of block statistics

draft-ietf-dnsop-dns-capture-format-00

- Changed dnstap.io to dnstap.info
- qr_data_format.png was cut off at the bottom
- Update authors address
- Improve wording in Abstract
- Changed DNS-STAT to C-DNS in CDDL
- Set the format version in the CDDL
- Added a TODO: Add block statistics
- Added a TODO: Add extend to support pico/nano. Also do this for Time offset and Response delay
- Added a TODO: Need to develop optional representation of malformed messages within C-DNS and what this means for packet matching. This may influence which fields are optional in the rest of the representation.
- Added section on design goals to Introduction

- Added a TODO: Can Class be optimised? Should a class of IN be inferred if not present?

draft-dickinson-dnsop-dns-capture-format-00

- Initial commit

16. References

16.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

16.2. Informative References

[ditl] DNS-OARC, "DITL", 2016, <https://www.dns-oarc.net/oarc/data/ditl>.

[dnscap] DNS-OARC, "DNSCAP", 2016, <https://www.dns-oarc.net/tools/dnscap>.

[dnstap] dnstap.info, "dnstap", 2016, <http://dnstap.info/>.

[dsc] Wessels, D. and J. Lundstrom, "DSC", 2016, <https://www.dns-oarc.net/tools/dsc>.

[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.hoffman-dns-in-json] Hoffman, P., "Representing DNS Messages in JSON", draft-hoffman-dns-in-json-14 (work in progress), April 2018.

[I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.

[packetq] .SE - The Internet Infrastructure Foundation, "PacketQ", 2014, <https://github.com/dotse/PacketQ>.

[pcap] tcpdump.org, "PCAP", 2016, <http://www.tcpdump.org/>.

[pcapng] Tuexen, M., Risso, F., Bongertz, J., Combs, G., and G. Harris, "pcap-ng", 2016, <https://github.com/pcapng/pcapng>.

[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>.

[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>.

16.3. URIs

[1] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-07/cdns_format.png

[2] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-07/cdns_format.svg

[3] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-07/qr_data_format.png

[4] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-07/qr_data_format.svg

[5] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-07/packet_matching.png

[6] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-07/packet_matching.svg

[7] https://github.com/dns-stats/compactor/wiki

[8] https://mm.dns-stats.org/mailman/listinfo/dns-stats-users

[9] https://www.sinodun.com/2017/06/compressing-pcap-files/

[10] https://www.sinodun.com/2017/06/more-on-debian-jessieubuntu-trusty-packet-capture-woes/

[11] https://github.com/miekg/mmark

[12] https://www.nlnetlabs.nl/projects/nsd/

[13] https://www.knot-dns.cz/

[14] https://avro.apache.org/

[15] https://developers.google.com/protocol-buffers/

[16] http://cbor.io

[17] https://github.com/kubo/snzip

[18] http://google.github.io/snappy/

[19] http://lz4.github.io/lz4/

[20] http://www.gzip.org/

[21] http://facebook.github.io/zstd/

[22] http://tukaani.org/xz/

[23] https://github.com/dns-stats/draft-dns-capture-format/blob/master/file-size-versus-block-size.png

[24] https://github.com/dns-stats/draft-dns-capture-format/blob/master/file-size-versus-block-size.svg

Appendix A. CDDL

```
; CDDL specification of the file format for C-DNS,
; which describes a collection of DNS messages and
; traffic meta-data.

;
; The overall structure of a file.
;
File = [
    file-type-id  : tstr .regexp "C-DNS",
    file-preamble : FilePreamble,
    file-blocks   : [* Block],
```

```
]

;
; The file preamble.
;
FilePreamble = {
    major-format-version => uint .eq 1,
    minor-format-version => uint .eq 0,
    ? private-version    => uint,
    block-parameters     => [+ BlockParameters],
}
major-format-version = 0
minor-format-version = 1
private-version      = 2
block-parameters     = 3

BlockParameters = {
    storage-parameters      => StorageParameters,
    ? collection-parameters => CollectionParameters,
}
storage-parameters    = 0
collection-parameters = 1

  StorageParameters = {
      ticks-per-second            => uint,
      max-block-items             => uint,
      storage-hints               => StorageHints,
      opcodes                     => [+ uint],
      rr-types                    => [+ uint],
      ? storage-flags             => StorageFlags,
      ? client-address-prefix-ipv4 => uint,
      ? client-address-prefix-ipv6 => uint,
      ? server-address-prefix-ipv4 => uint,
      ? server-address-prefix-ipv6 => uint,
      ? sampling-method           => tstr,
      ? anonymisation-method      => tstr,
  }
  ticks-per-second           = 0
  max-block-items            = 1
  storage-hints              = 2
  opcodes                    = 3
  rr-types                   = 4
  storage-flags              = 5
  client-address-prefix-ipv4 = 6
  client-address-prefix-ipv6 = 7
  server-address-prefix-ipv4 = 8
  server-address-prefix-ipv6 = 9
  sampling-method            = 10
```

```
  anonymisation-method       = 11

    ; A hint indicates if the collection method will output the
    ; item or will ignore the item if present.
    StorageHints = {
        query-response-hints           => QueryResponseHints,
        query-response-signature-hints => QueryResponseSignatureHints,
        rr-hints                       => RRHints,
        other-data-hints               => OtherDataHints,
    }
    query-response-hints           = 0
    query-response-signature-hints = 1
    rr-hints                       = 2
    other-data-hints               = 3

      QueryResponseHintValues = &(
          time-offset                 : 0,
          client-address-index        : 1,
          client-port                 : 2,
          transaction-id              : 3,
          qr-signature-index          : 4,
          client-hoplimit             : 5,
          response-delay              : 6,
          query-name-index            : 7,
          query-size                  : 8,
          response-size               : 9,
          response-processing-data    : 10,
          query-question-sections     : 11,    ; Second & subsequent questions
          query-answer-sections       : 12,
          query-authority-sections    : 13,
          query-additional-sections   : 14,
          response-answer-sections    : 15,
          response-authority-sections : 16,
          response-additional-sections : 17,
      )
      QueryResponseHints = uint .bits QueryResponseHintValues

      QueryResponseSignatureHintValues = &(
          server-address     : 0,
          server-port        : 1,
          qr-transport-flags : 2,
          qr-type            : 3,
          qr-sig-flags       : 4,
          query-opcode       : 5,
          dns-flags          : 6,
          query-rcode        : 7,
          query-class-type   : 8,
          query-qdcount      : 9,
```

```
          query-ancount      : 10,
          query-arcount      : 11,
          query-nscount      : 12,
          query-edns-version : 13,
          query-udp-size     : 14,
          query-opt-rdata    : 15,
          response-rcode     : 16,
      )
      QueryResponseSignatureHints = uint .bits QueryResponseSignatureHintValues

      RRHintValues = &(
          ttl         : 0,
          rdata-index : 1,
      )
      RRHints = uint .bits RRHintValues

      OtherDataHintValues = &(
          malformed-messages   : 0,
          address-event-counts : 1,
      )
      OtherDataHints = uint .bits OtherDataHintValues

    StorageFlagValues = &(
        anonymised-data      : 0,
        sampled-data         : 1,
        normalised-names     : 2,
    )
    StorageFlags = uint .bits StorageFlagValues

  CollectionParameters = {
      ? query-timeout      => uint,
      ? skew-timeout       => uint,
      ? snaplen            => uint,
      ? promisc            => uint,
      ? interfaces         => [+ tstr],
      ? server-addresses   => [+ IPAddress], ; Hint for later analysis
      ? vlan-ids           => [+ uint],
      ? filter             => tstr,
      ? generator-id       => tstr,
      ? host-id            => tstr,
  }
  query-timeout      = 0
  skew-timeout       = 1
  snaplen            = 2
  promisc            = 3
  interfaces         = 4
  server-addresses   = 5
  vlan-ids           = 6
```

```
  filter             = 7
  generator-id       = 8
  host-id            = 9

;
; Data in the file is stored in Blocks.
;
Block = {
    block-preamble          => BlockPreamble,
    ? block-statistics      => BlockStatistics, ; Much of this could be derived
    ? block-tables          => BlockTables,
    ? query-responses       => [+ QueryResponse],
    ? address-event-counts  => [+ AddressEventCount],
    ? malformed-messages    => [+ MalformedMessage],
}
block-preamble        = 0
block-statistics      = 1
block-tables          = 2
query-responses       = 3
address-event-counts  = 4
malformed-messages    = 5

;
; The (mandatory) preamble to a block.
;
BlockPreamble = {
    ? earliest-time          => Timestamp,
    ? block-parameters-index => uint .default 0,
}
earliest-time          = 0
block-parameters-index = 1

; Ticks are subsecond intervals. The number of ticks in a second is file/block
; metadata. Signed and unsigned tick types are defined.
ticks = int
uticks = uint

Timestamp = [
    timestamp-secs   : uint,
    timestamp-uticks : uticks,
]

;
; Statistics about the block contents.
;
BlockStatistics = {
    ? processed-messages  => uint,
    ? qr-data-items       => uint,
```

```
    ? unmatched-queries   => uint,
    ? unmatched-responses => uint,
    ? discarded-opcode    => uint,
    ? malformed-items     => uint,
}
processed-messages  = 0
qr-data-items       = 1
unmatched-queries   = 2
unmatched-responses = 3
discarded-opcode    = 4
malformed-items     = 5

;
; Tables of common data referenced from records in a block.
;
BlockTables = {
    ? ip-address            => [+ IPAddress],
    ? classtype             => [+ ClassType],
    ? name-rdata            => [+ bstr],    ; Holds both Name RDATA and RDATA
    ? qr-sig                => [+ QueryResponseSignature],
    ? QuestionTables,
    ? RRTables,
    ? malformed-message-data => [+ MalformedMessageData],
}
ip-address            = 0
classtype             = 1
name-rdata            = 2
qr-sig                = 3
qlist                 = 4
qrr                   = 5
rrlist                = 6
rr                    = 7
malformed-message-data = 8

IPv4Address = bstr .size 4
IPv6Address = bstr .size 16
IPAddress = IPv4Address / IPv6Address

ClassType = {
    type  => uint,
    class => uint,
}
type  = 0
class = 1

QueryResponseSignature = {
    ? server-address-index  => uint,
    ? server-port           => uint,
```

```
    ? qr-transport-flags    => QueryResponseTransportFlags,
    ? qr-type               => QueryResponseType,
    ? qr-sig-flags          => QueryResponseFlags,
    ? query-opcode          => uint,
    ? qr-dns-flags          => DNSFlags,
    ? query-rcode           => uint,
    ? query-classtype-index => uint,
    ? query-qd-count        => uint,
    ? query-an-count        => uint,
    ? query-ns-count        => uint,
    ? query-ar-count        => uint,
    ? edns-version          => uint,
    ? udp-buf-size          => uint,
    ? opt-rdata-index       => uint,
    ? response-rcode        => uint,
}
server-address-index  = 0
server-port           = 1
qr-transport-flags    = 2
qr-type               = 3
qr-sig-flags          = 4
query-opcode          = 5
qr-dns-flags          = 6
query-rcode           = 7
query-classtype-index = 8
query-qd-count        = 9
query-an-count        = 10
query-ns-count        = 12
query-ar-count        = 12
edns-version          = 13
udp-buf-size          = 14
opt-rdata-index       = 15
response-rcode        = 16

  Transport = &(
      udp               : 0,
      tcp               : 1,
      tls               : 2,
      dtls              : 3,
  )

  TransportFlagValues = &(
      ip-version         : 0,     ; 0=IPv4, 1=IPv6
      ; Transport value bits 1-4
  ) / (1..4)
  TransportFlags = uint .bits TransportFlagValues

  QueryResponseTransportFlagValues = &(
```

```
      query-trailingdata : 5,
  ) / TransportFlagValues
  QueryResponseTransportFlags = uint .bits QueryResponseTransportFlagValues

  QueryResponseType = &(
      stub      : 0,
      client    : 1,
      resolver  : 2,
      auth      : 3,
      forwarder : 4,
      tool      : 5,
  )

  QueryResponseFlagValues = &(
      has-query               : 0,
      has-reponse             : 1,
      query-has-opt           : 2,
      response-has-opt        : 3,
      query-has-no-question   : 4,
      response-has-no-question: 5,
  )
  QueryResponseFlags = uint .bits QueryResponseFlagValues

  DNSFlagValues = &(
      query-cd   : 0,
      query-ad   : 1,
      query-z    : 2,
      query-ra   : 3,
      query-rd   : 4,
      query-tc   : 5,
      query-aa   : 6,
      query-do   : 7,
      response-cd: 8,
      response-ad: 9,
      response-z : 10,
      response-ra: 11,
      response-rd: 12,
      response-tc: 13,
      response-aa: 14,
  )
  DNSFlags = uint .bits DNSFlagValues

QuestionTables = (
    qlist => [+ QuestionList],
    qrr   => [+ Question]
)

  QuestionList = [+ uint]           ; Index of Question
```

```
  Question = {                      ; Second and subsequent questions
      name-index      => uint,      ; Index to a name in the name-rdata table
      classtype-index => uint,
  }
  name-index      = 0
  classtype-index = 1

RRTables = (
    rrlist => [+ RRList],
    rr     => [+ RR]
)

  RRList = [+ uint]                     ; Index of RR

  RR = {
      name-index      => uint,          ; Index to a name in the name-rdata tabl
e
      classtype-index => uint,
      ? ttl           => uint,
      ? rdata-index   => uint,          ; Index to RDATA in the name-rdata table
  }
  ; Other map key values already defined above.
  ttl         = 2
  rdata-index = 3

MalformedMessageData = {
    ? server-address-index   => uint,
    ? server-port            => uint,
    ? mm-transport-flags     => TransportFlags,
    ? mm-payload             => bstr,
}
; Other map key values already defined above.
mm-transport-flags      = 2
mm-payload              = 3

;
; A single query/response pair.
;
QueryResponse = {
    ? time-offset              => uticks,     ; Time offset from start of block
    ? client-address-index     => uint,
    ? client-port              => uint,
    ? transaction-id           => uint,
    ? qr-signature-index       => uint,
    ? client-hoplimit          => uint,
    ? response-delay           => ticks,
    ? query-name-index         => uint,
    ? query-size               => uint,       ; DNS size of query
    ? response-size            => uint,       ; DNS size of response
```

```
    ? response-processing-data => ResponseProcessingData,
    ? query-extended           => QueryResponseExtended,
    ? response-extended        => QueryResponseExtended,
}
time-offset              = 0
client-address-index     = 1
client-port              = 2
transaction-id           = 3
qr-signature-index       = 4
client-hoplimit          = 5
response-delay           = 6
query-name-index         = 7
query-size               = 8
response-size            = 9
response-processing-data = 10
query-extended           = 11
response-extended        = 12

ResponseProcessingData = {
    ? bailiwick-index  => uint,
    ? processing-flags => ResponseProcessingFlags,
}
bailiwick-index = 0
processing-flags = 1

  ResponseProcessingFlagValues = &(
      from-cache : 0,
  )
  ResponseProcessingFlags = uint .bits ResponseProcessingFlagValues

QueryResponseExtended = {
    ? question-index   => uint,       ; Index of QuestionList
    ? answer-index     => uint,       ; Index of RRList
    ? authority-index  => uint,
    ? additional-index => uint,
}
question-index   = 0
answer-index     = 1
authority-index  = 2
additional-index = 3

;
; Address event data.
;
AddressEventCount = {
    ae-type          => &AddressEventType,
    ? ae-code        => uint,
    ae-address-index => uint,
```

```
    ae-count         => uint,
}
ae-type          = 0
ae-code          = 1
ae-address-index = 2
ae-count         = 3

AddressEventType = (
    tcp-reset              : 0,
    icmp-time-exceeded     : 1,
    icmp-dest-unreachable  : 2,
    icmpv6-time-exceeded   : 3,
    icmpv6-dest-unreachable: 4,
    icmpv6-packet-too-big  : 5,
)

;
; Malformed messages.
;
MalformedMessage = {
    ? time-offset           => uticks,   ; Time offset from start of block
    ? client-address-index  => uint,
    ? client-port           => uint,
    ? message-data-index    => uint,
}
; Other map key values already defined above.
message-data-index = 3
```

Appendix B. DNS Name compression example

The basic algorithm, which follows the guidance in [RFC1035], is simply to collect each name, and the offset in the packet at which it starts, during packet construction. As each name is added, it is offered to each of the collected names in order of collection, starting from the first name. If labels at the end of the name can be replaced with a reference back to part (or all) of the earlier name, and if the uncompressed part of the name is shorter than any compression already found, the earlier name is noted as the compression target for the name.

The following tables illustrate the process. In an example packet, the first name is example.com.

N	Name	Uncompressed	Compression Target
1	example.com		

The next name added is bar.com. This is matched against example.com. The com part of this can be used as a compression target, with the remaining uncompressed part of the name being bar.

N	Name	Uncompressed	Compression Target
1	example.com		
2	bar.com	bar	1 + offset to com

The third name added is www.bar.com. This is first matched against example.com, and as before this is recorded as a compression target, with the remaining uncompressed part of the name being www.bar. It is then matched against the second name, which again can be a compression target. Because the remaining uncompressed part of the name is www, this is an improved compression, and so it is adopted.

N	Name	Uncompressed	Compression Target
1	example.com		
2	bar.com	bar	1 + offset to com
3	www.bar.com	www	2

As an optimization, if a name is already perfectly compressed (in other words, the uncompressed part of the name is empty), then no further names will be considered for compression.

B.1. NSD compression algorithm

Using the above basic algorithm the packet lengths of responses generated by NSD [12] can be matched almost exactly. At the time of writing, a tiny number (<.01%) of the reconstructed packets had incorrect lengths.

B.2. Knot Authoritative compression algorithm

The Knot Authoritative [13] name server uses different compression behavior, which is the result of internal optimization designed to balance runtime speed with compression size gains. In brief, and omitting complications, Knot Authoritative will only consider the QNAME and names in the immediately preceding RR section in an RRSET as compression targets.

A set of smart heuristics as described below can be implemented to mimic this and while not perfect it produces output nearly, but not quite, as good a match as with NSD. The heuristics are:

1. A match is only perfect if the name is completely compressed AND the TYPE of the section in which the name occurs matches the TYPE of the name used as the compression target.

2. If the name occurs in RDATA:

   * If the compression target name is in a query, then only the first RR in an RRSET can use that name as a compression target.

   * The compression target name MUST be in RDATA.

   * The name section TYPE must match the compression target name section TYPE.

   * The compression target name MUST be in the immediately preceding RR in the RRSET.

Using this algorithm less than 0.1% of the reconstructed packets had incorrect lengths.

B.3. Observed differences

In sample traffic collected on a root name server around 2-4% of responses generated by Knot had different packet lengths to those produced by NSD.

Appendix C. Comparison of Binary Formats

Several binary serialisation formats were considered, and for completeness were also compared to JSON.

- Apache Avro [14]. Data is stored according to a pre-defined schema. The schema itself is always included in the data file. Data can therefore be stored untagged, for a smaller serialisation size, and be written and read by an Avro library.

  * At the time of writing, Avro libraries are available for C, C++, C#, Java, Python, Ruby and PHP. Optionally tools are available for C++, Java and C# to generate code for encoding and decoding.

- Google Protocol Buffers [15]. Data is stored according to a pre-defined schema. The schema is used by a generator to generate

code for encoding and decoding the data. Data can therefore be stored untagged, for a smaller serialisation size. The schema is not stored with the data, so unlike Avro cannot be read with a generic library.

* Code must be generated for a particular data schema to to read and write data using that schema. At the time of writing, the Google code generator can currently generate code for encoding and decoding a schema for C++, Go, Java, Python, Ruby, C#, Objective-C, Javascript and PHP.

- CBOR [16]. Defined in [RFC7049], this serialisation format is comparable to JSON but with a binary representation. It does not use a pre-defined schema, so data is always stored tagged. However, CBOR data schemas can be described using CDDL [I-D.ietf-cbor-cddl] and tools exist to verify data files conform to the schema.

  * CBOR is a simple format, and simple to implement. At the time of writing, the CBOR website lists implementations for 16 languages.

Avro and Protocol Buffers both allow storage of untagged data, but because they rely on the data schema for this, their implementation is considerably more complex than CBOR. Using Avro or Protocol Buffers in an unsupported environment would require notably greater development effort compared to CBOR.

A test program was written which reads input from a PCAP file and writes output using one of two basic structures; either a simple structure, where each query/response pair is represented in a single record entry, or the C-DNS block structure.

The resulting output files were then compressed using a variety of common general-purpose lossless compression tools to explore the compressibility of the formats. The compression tools employed were:

- snzip [17]. A command line compression tool based on the Google Snappy [18] library.

- lz4 [19]. The command line compression tool from the reference C LZ4 implementation.

- gzip [20]. The ubiquitous GNU zip tool.

- zstd [21]. Compression using the Zstandard algorithm.

- xz [22]. A popular compression tool noted for high compression.

In all cases the compression tools were run using their default settings.

Note that this draft does not mandate the use of compression, nor any particular compression scheme, but it anticipates that in practice output data will be subject to general-purpose compression, and so this should be taken into consideration.

"test.pcap", a 662Mb capture of sample data from a root instance was used for the comparison. The following table shows the formatted size and size after compression (abbreviated to Comp. in the table headers), together with the task resident set size (RSS) and the user time taken by the compression. File sizes are in Mb, RSS in kb and user time in seconds.

Format	File size	Comp.	Comp. size	RSS	User time
PCAP	661.87	snzip	212.48	2696	1.26
		lz4	181.58	6336	1.35
		gzip	153.46	1428	18.20
		zstd	87.07	3544	4.27
		xz	49.09	97416	160.79
JSON simple	4113.92	snzip	603.78	2656	5.72
		lz4	386.42	5636	5.25
		gzip	271.11	1492	73.00
		zstd	133.43	3284	8.68
		xz	51.98	97412	600.74
Avro simple	640.45	snzip	148.98	2656	0.90
		lz4	111.92	5828	0.99
		gzip	103.07	1540	11.52
		zstd	49.08	3524	2.50
		xz	22.87	97308	90.34
CBOR simple	764.82	snzip	164.57	2664	1.11
		lz4	120.98	5892	1.13
		gzip	110.61	1428	12.88
		zstd	54.14	3224	2.77
		xz	23.43	97276	111.48
PBuf simple	749.51	snzip	167.16	2660	1.08
		lz4	123.09	5824	1.14
		gzip	112.05	1424	12.75
		zstd	53.39	3388	2.76
		xz	23.99	97348	106.47

JSON block	519.77	snzip	106.12	2812	0.93
		lz4	104.34	6080	0.97
		gzip	57.97	1604	12.70
		zstd	61.51	3396	3.45
		xz	27.67	97524	169.10
Avro block	60.45	snzip	48.38	2688	0.20
		lz4	48.78	8540	0.22
		gzip	39.62	1576	2.92
		zstd	29.63	3612	1.25
		xz	18.28	97564	25.81
CBOR block	75.25	snzip	53.27	2684	0.24
		lz4	51.88	8008	0.28
		gzip	41.17	1548	4.36
		zstd	30.61	3476	1.48
		xz	18.15	97556	38.78
PBuf block	67.98	snzip	51.10	2636	0.24
		lz4	52.39	8304	0.24
		gzip	40.19	1520	3.63
		zstd	31.61	3576	1.40
		xz	17.94	97440	33.99

The above results are discussed in the following sections.

C.1. Comparison with full PCAP files

An important first consideration is whether moving away from PCAP offers significant benefits.

The simple binary formats are typically larger than PCAP, even though they omit some information such as Ethernet MAC addresses. But not only do they require less CPU to compress than PCAP, the resulting compressed files are smaller than compressed PCAP.

C.2. Simple versus block coding

The intention of the block coding is to perform data de-duplication on query/response records within the block. The simple and block formats above store exactly the same information for each query/response record. This information is parsed from the DNS traffic in the input PCAP file, and in all cases each field has an identifier and the field data is typed.

The data de-duplication on the block formats show an order of magnitude reduction in the size of the format file size against the

simple formats. As would be expected, the compression tools are able to find and exploit a lot of this duplication, but as the de-duplication process uses knowledge of DNS traffic, it is able to retain a size advantage. This advantage reduces as stronger compression is applied, as again would be expected, but even with the strongest compression applied the block formatted data remains around 75% of the size of the simple format and its compression requires roughly a third of the CPU time.

C.3. Binary versus text formats

Text data formats offer many advantages over binary formats, particularly in the areas of ad-hoc data inspection and extraction. It was therefore felt worthwhile to carry out a direct comparison, implementing JSON versions of the simple and block formats.

Concentrating on JSON block format, the format files produced are a significant fraction of an order of magnitude larger than binary formats. The impact on file size after compression is as might be expected from that starting point; the stronger compression produces files that are 150% of the size of similarly compressed binary format, and require over 4x more CPU to compress.

C.4. Performance

Concentrating again on the block formats, all three produce format files that are close to an order of magnitude smaller that the original "test.pcap" file. CBOR produces the largest files and Avro the smallest, 20% smaller than CBOR.

However, once compression is taken into account, the size difference narrows. At medium compression (with gzip), the size difference is 4%. Using strong compression (with xz) the difference reduces to 2%, with Avro the largest and Protocol Buffers the smallest, although CBOR and Protocol Buffers require slightly more compression CPU.

The measurements presented above do not include data on the CPU required to generate the format files. Measurements indicate that writing Avro requires 10% more CPU than CBOR or Protocol Buffers. It appears, therefore, that Avro’s advantage in compression CPU usage is probably offset by a larger CPU requirement in writing Avro.

C.5. Conclusions

The above assessments lead us to the choice of a binary format file using blocking.

As noted previously, this draft anticipates that output data will be subject to compression. There is no compelling case for one particular binary serialisation format in terms of either final file size or machine resources consumed, so the choice must be largely based on other factors. CBOR was therefore chosen as the binary serialisation format for the reasons listed in Section 5.

C.6. Block size choice

Given the choice of a CBOR format using blocking, the question arises of what an appropriate default value for the maximum number of query/response pairs in a block should be. This has two components; what is the impact on performance of using different block sizes in the format file, and what is the impact on the size of the format file before and after compression.

The following table addresses the performance question, showing the impact on the performance of a C++ program converting "test.pcap" to C-DNS. File size is in Mb, resident set size (RSS) in kb.

Block size	File size	RSS	User time
1000	133.46	612.27	15.25
5000	89.85	676.82	14.99
10000	76.87	752.40	14.53
20000	67.86	750.75	14.49
40000	61.88	736.30	14.29
80000	58.08	694.16	14.28
160000	55.94	733.84	14.44
320000	54.41	799.20	13.97

Increasing block size, therefore, tends to increase maximum RSS a little, with no significant effect (if anything a small reduction) on CPU consumption.

The following figure plots the effect of increasing block size on output file size for different compressions.

Figure showing effect of block size on file size (PNG) [23]

Figure showing effect of block size on file size (SVG) [24]

From the above, there is obviously scope for tuning the default block size to the compression being employed, traffic characteristics, frequency of output file rollover etc. Using a strong compression,

block sizes over 10,000 query/response pairs would seem to offer limited improvements.

Authors' Addresses

John Dickinson
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom

Email: jad@sinodun.com

Jim Hague
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom

Email: jim@sinodun.com

Sara Dickinson
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom

Email: sara@sinodun.com

Terry Manderson
ICANN
12025 Waterfront Drive
Suite 300
Los Angeles CA 90094-2536

Email: terry.manderson@icann.org

John Bond
ICANN
12025 Waterfront Drive
Suite 300
Los Angeles CA 90094-2536

Email: john.bond@icann.org

Network Working Group
Internet-Draft
Obsoletes: 7719 (if approved)
Updates: 2308 (if approved)
Intended status: Best Current Practice
Expires: October 29, 2018

P. Hoffman
ICANN
A. Sullivan
Oracle
K. Fujiwara
JPRS

April 27, 2018

DNS Terminology
draft-ietf-dnsop-terminology-bis-10

Abstract

The domain name system (DNS) is defined in literally dozens of different RFCs. The terminology used by implementers and developers of DNS protocols, and by operators of DNS systems, has sometimes changed in the decades since the DNS was first defined. This document gives current definitions for many of the terms used in the DNS in a single document.

This document will be the successor to RFC 7719, and thus will obsolete RFC 7719. It will also update RFC 2308.

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 October 29, 2018.

Table of Contents

1. Introduction

The Domain Name System (DNS) is a simple query-response protocol whose messages in both directions have the same format. (Section 2 gives a definition of "public DNS", which is often what people mean when they say "the DNS".) The protocol and message format are defined in [RFC1034] and [RFC1035]. These RFCs defined some terms, but later documents defined others. Some of the terms from [RFC1034] and [RFC1035] now have somewhat different meanings than they did in 1987.

This document collects a wide variety of DNS-related terms. Some of them have been precisely defined in earlier RFCs, some have been loosely defined in earlier RFCs, and some are not defined in any earlier RFC at all.

Most of the definitions here are the consensus definition of the DNS community -- both protocol developers and operators. Some of the definitions differ from earlier RFCs, and those differences are noted. In this document, where the consensus definition is the same as the one in an RFC, that RFC is quoted. Where the consensus definition has changed somewhat, the RFC is mentioned but the new stand-alone definition is given. See Appendix A for a list of the definitions that this document updates.

It is important to note that, during the development of this document, it became clear that some DNS-related terms are interpreted quite differently by different DNS experts. Further, some terms that are defined in early DNS RFCs now have definitions that are generally agreed to, but that are different from the original definitions. Therefore, this document is a substantial revision to [RFC7719].

The terms are organized loosely by topic. Some definitions are for new terms for things that are commonly talked about in the DNS community but that never had terms defined for them.

Other organizations sometimes define DNS-related terms their own way. For example, the W3C defines "domain" at <https://url.spec.whatwg.org/>. The Root Server System Advisory Committee (RSSAC) has a good lexicon [RSSAC026].

Note that there is no single consistent definition of "the DNS". It can be considered to be some combination of the following: a commonly used naming scheme for objects on the Internet; a distributed database representing the names and certain properties of these objects; an architecture providing distributed maintenance, resilience, and loose coherency for this database; and a simple query-response protocol (as mentioned below) implementing this architecture. Section 2 defines "global DNS" and "private DNS" as a way to deal with these differing definitions.

Capitalization in DNS terms is often inconsistent among RFCs and various DNS practitioners. The capitalization used in this document is a best guess at current practices, and is not meant to indicate that other capitalization styles are wrong or archaic. In some cases, multiple styles of capitalization are used for the same term due to quoting from different RFCs.

2. Names

Naming system: A naming system associates names with data. Naming systems have many significant facets that help differentiate them. Some commonly-identified facets include:

* Composition of names
* Format of names
* Administration of names
* Types of data that can be associated with names
* Types of metadata for names
* Protocol for getting data from a name
* Context for resolving a name

Note that this list is a small subset of facets that people have identified over time for naming systems, and the IETF has yet to agree on a good set of facets that can be used to compare naming systems. For example, other facets might include "protocol to update data in a name", "privacy of names", and "privacy of data associated with names", but those are not as well-defined as the ones listed above. The list here is chosen because it helps describe the DNS and naming systems similar to the DNS.

Domain name: An ordered list of one or more labels.

Note that this is a definition independent of the DNS RFCs, and the definition here also applies to systems other than the DNS. [RFC1034] defines the "domain name space" using mathematical trees and their nodes in graph theory, and this definition has the same practical result as the definition here. Using graph theory, a domain name is a list of labels identifying a portion along one edge of a directed acyclic graph. A domain name can be relative to parts of the tree, or it can be fully qualified (in which case, it begins at the common root of the graph).

Also note that different IETF and non-IETF documents have used the term "domain name" in many different ways. It is common for earlier documents to use "domain name" to mean "names that match the syntax in [RFC1035]", but possibly with additional rules such as "and are, or will be, resolvable in the global DNS" or "but only using the presentation format".

Label: An ordered list of zero or more octets and which makes up a portion of a domain name. Using graph theory, a label identifies one node in a portion of the graph of all possible domain names.

Global DNS: Using the short set of facets listed in "Naming system", the global DNS can be defined as follows. Most of the rules here

come from [RFC1034] and [RFC1035], although the term "global DNS" has not been defined before now.

Composition of names -- A name in the global DNS has one or more labels. The length of each label is between 0 and 63 octets inclusive. In a fully-qualified domain name, the first label in the ordered list is 0 octets long; it is the only label whose length may be 0 octets, and it is called the "root" or "root label". A domain name in the global DNS has a maximum total length of 255 octets in the wire format; the root represents one octet for this calculation. (Multicast DNS [RFC6762] allows names up to 255 bytes plus a terminating zero byte based on a different interpretation of RFC 1035 and what is included in the 255 octets.)

Format of names -- Names in the global DNS are domain names. There are three formats: wire format, presentation format, and common display.

The basic wire format for names in the global DNS is a list of labels ordered by decreasing distance from the root, with the root label last. Each label is preceded by a length octet. [RFC1035] also defines a compression scheme that modifies this format.

The presentation format for names in the global DNS is a list of labels ordered by decreasing distance from the root, encoded as ASCII, with a "." character between each label. In presentation format, a fully-qualified domain name includes the root label and the associated separator dot. For example, in presentation format, a fully-qualified domain name with two non-root labels is always shown as "example.tld." instead of "example.tld". [RFC1035] defines a method for showing octets that do not display in ASCII.

The common display format is used in applications and free text. It is the same as the presentation format, but showing the root label and the "." before it is optional and is rarely done. For example, in common display format, a fully-qualified domain name with two non-root labels is usually shown as "example.tld" instead of "example.tld.". Names in the common display format are normally written such that the directionality of the writing system presents labels by decreasing distance from the root (so, in both English and C the root or TLD label in the ordered list is right-most; but in Arabic it may be left-most, depending on local conventions).

Administration of names -- Administration is specified by delegation (see the definition of "delegation" in Section 7).

Policies for administration of the root zone in the global DNS are determined by the names operational community, which convenes itself in the Internet Corporation for Assigned Names and Numbers (ICANN). The names operational community selects the IANA Functions Operator for the global DNS root zone. At the time this document is published, that operator is Public Technical Identifiers (PTI). The name servers that serve the root zone are provided by independent root operators. Other zones in the global DNS have their own policies for administration.

Types of data that can be associated with names -- A name can have zero or more resource records associated with it. There are numerous types of resource records with unique data structures defined in many different RFCs and in the IANA registry at [IANA_Resource_Registry].

Types of metadata for names -- Any name that is published in the DNS appears as a set of resource records (see the definition of "RRset" in Section 5). Some names do not themselves have data associated with them in the DNS, but "appear" in the DNS anyway because they form part of a longer name that does have data associated with it (see the definition of "empty non-terminals" in Section 7).

Protocol for getting data from a name -- The protocol described in [RFC1035].

Context for resolving a name -- The global DNS root zone distributed by PTI.

Private DNS: Names that use the protocol described in [RFC1035] but that do not rely on the global DNS root zone, or names that are otherwise not generally available on the Internet but are using the protocol described in [RFC1035]. A system can use both the global DNS and one or more private DNS systems; for example, see "Split DNS" in Section 6.

Note that domain names that do not appear in the DNS, and that are intended never to be looked up using the DNS protocol, are not part of the global DNS or a private DNS even though they are domain names.

Multicast DNS: "Multicast DNS (mDNS) provides the ability to perform DNS-like operations on the local link in the absence of any conventional Unicast DNS server. In addition, Multicast DNS designates a portion of the DNS namespace to be free for local use, without the need to pay any annual fee, and without the need to set up delegations or otherwise configure a conventional DNS

server to answer for those names." Quoted from [RFC6762], Abstract) Although it uses a compatible wire format, mDNS is strictly speaking a different protocol than DNS. Also, where the above quote says "a portion of the DNS namespace", it would be clearer to say "a portion of the domain name space" The names in mDNS are not intended to be looked up in the DNS.

Locally served DNS zone: A locally served DNS zone is a special case of private DNS. Names are resolved using the DNS protocol in a local context. [RFC6303] defines subdomains of IN-ADDR.ARPA that are locally served zones. Resolution of names through locally served zones may result in ambiguous results. For example, the same name may resolve to different results in different locally served DNS zone contexts. The context through which a locally served zone may be explicit, for example, as defined in [RFC6303], or implicit, as defined by local DNS administration and not known to the resolution client.

Fully qualified domain name (FQDN): This is often just a clear way of saying the same thing as "domain name of a node", as outlined above. However, the term is ambiguous. Strictly speaking, a fully qualified domain name would include every label, including the zero-length label of the root: such a name would be written "www.example.net." (note the terminating dot). But because every name eventually shares the common root, names are often written relative to the root (such as "www.example.net") and are still called "fully qualified". This term first appeared in [RFC0819]. In this document, names are often written relative to the root.

The need for the term "fully qualified domain name" comes from the existence of partially qualified domain names, which are names where one or more of the earliest labels in the ordered list are omitted (for example, a name of "www" derived from "www.example.net"). Such relative names are understood only by context.

Host name: This term and its equivalent, "hostname", have been widely used but are not defined in [RFC1034], [RFC1035], [RFC1123], or [RFC2181]. The DNS was originally deployed into the Host Tables environment as outlined in [RFC0952], and it is likely that the term followed informally from the definition there. Over time, the definition seems to have shifted. "Host name" is often meant to be a domain name that follows the rules in Section 3.5 of [RFC1034], the "preferred name syntax" (that is, every character in each label are a letter, a digit, or a hyphen). Note that any label in a domain name can contain any octet value; hostnames are generally considered to be domain names where every label follows the rules in the "preferred name syntax", with the amendment that

labels can start with ASCII digits (this amendment comes from Section 2.1 of [RFC1123]).

People also sometimes use the term hostname to refer to just the first label of an FQDN, such as "printer" in "printer.admin.example.com". (Sometimes this is formalized in configuration in operating systems.) In addition, people sometimes use this term to describe any name that refers to a machine, and those might include labels that do not conform to the "preferred name syntax".

TLD: A Top-Level Domain, meaning a zone that is one layer below the root, such as "com" or "jp". There is nothing special, from the point of view of the DNS, about TLDs. Most of them are also delegation-centric zones (defined in Section 7, and there are significant policy issues around their operation. TLDs are often divided into sub-groups such as Country Code Top-Level Domains (ccTLDs), Generic Top-Level Domains (gTLDs), and others; the division is a matter of policy, and beyond the scope of this document.

IDN: The common abbreviation for "Internationalized Domain Name". The IDNA protocol is the standard mechanism for handling domain names with non-ASCII characters in applications in the DNS. The current standard, normally called "IDNA2008", is defined in [RFC5890], [RFC5891], [RFC5892], [RFC5893], and [RFC5894]. These documents define many IDN-specific terms such as "LDH label", "A-label", and "U-label". [RFC6365] defines more terms that relate to internationalization (some of which relate to IDNs), and [RFC6055] has a much more extensive discussion of IDNs, including some new terminology.

Subdomain: "A domain is a subdomain of another domain if it is contained within that domain. This relationship can be tested by seeing if the subdomain's name ends with the containing domain's name." (Quoted from [RFC1034], Section 3.1). For example, in the host name "nnn.mmm.example.com", both "mmm.example.com" and "nnn.mmm.example.com" are subdomains of "example.com".

Alias: The owner of a CNAME resource record, or a subdomain of the owner of a DNAME resource record. See also "canonical name".

Canonical name: A CNAME resource record "identifies its owner name as an alias, and specifies the corresponding canonical name in the RDATA section of the RR." (Quoted from [RFC1034], Section 3.6.2) This usage of the word "canonical" is related to the mathematical concept of "canonical form".

CNAME: "It is traditional to refer to the owner of a CNAME record as 'a CNAME'. This is unfortunate, as 'CNAME' is an abbreviation of 'canonical name', and the owner of a CNAME record is an alias, not a canonical name." (Quoted from [RFC2181], Section 10.1.1)

3. DNS Response Codes

Some of response codes that are defined in [RFC1035] have acquired their own shorthand names. All of the RCODEs are listed at [IANA_Resource_Registry], although that site uses mixed-case capitalization, while most documents use all-caps. Some of the common names are described here, but the official list is in the IANA registry.

NOERROR: "No error condition" (Quoted from [RFC1035], Section 4.1.1.)

FORMERR: "Format error - The name server was unable to interpret the query." (Quoted from [RFC1035], Section 4.1.1.)

SERVFAIL: "Server failure - The name server was unable to process this query due to a problem with the name server." (Quoted from [RFC1035], Section 4.1.1.)

NXDOMAIN: "Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist." (Quoted from [RFC1035], Section 4.1.1.) [RFC2308] established NXDOMAIN as a synonym for Name Error.

NOTIMP: "Not Implemented - The name server does not support the requested kind of query." (Quoted from [RFC1035], Section 4.1.1.)

REFUSED: "Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g., zone transfer) for particular data." (Quoted from [RFC1035], Section 4.1.1.)

NODATA: "A pseudo RCODE which indicates that the name is valid for the given class, but there are no records of the given type. A NODATA response has to be inferred from the answer." (Quoted from [RFC2308], Section 1.) "NODATA is indicated by an answer with the RCODE set to NOERROR and no relevant answers in the answer section. The authority section will contain an SOA record, or there will be no NS records there." (Quoted from [RFC2308],

Section 2.2.) Note that referrals have a similar format to NODATA replies; [RFC2308] explains how to distinguish them.

The term "NXRRSET" is sometimes used as a synonym for NODATA. However, this is a mistake, given that NXRRSET is a specific error code defined in [RFC2136].

Negative response: A response that indicates that a particular RRset does not exist, or whose RCODE indicates the nameserver cannot answer. Sections 2 and 7 of [RFC2308] describe the types of negative responses in detail.

4. DNS Transactions

The header of a DNS message is its first 12 octets. Many of the fields and flags in the header diagram in Sections 4.1.1 through 4.1.3 of [RFC1035] are referred to by their names in that diagram. For example, the response codes are called "RCODEs", the data for a record is called the "RDATA", and the authoritative answer bit is often called "the AA flag" or "the AA bit".

QNAME: The most commonly-used rough definition is that the QNAME is a field in the Question section of a query. "A standard query specifies a target domain name (QNAME), query type (QTYPE), and query class (QCLASS) and asks for RRs which match." (Quoted from [RFC1034], Section 3.7.1.). Strictly speaking, the definition comes from [RFC1035], Section 4.1.2, where the QNAME is defined in respect of the Question Section. This definition appears to be applied consistently: the discussion of inverse queries in section 6.4 refers to the "owner name of the query RR and its TTL", because inverse queries populate the Answer Section and leave the Question Section empty. (Inverse queries are deprecated in [RFC3425], and so relevant definitions do not appear in this document.)

[RFC2308], however, has an alternate definition that puts the QNAME in the answer (or series of answers) instead of the query. It defines QNAME as: "...the name in the query section of an answer, or where this resolves to a CNAME, or CNAME chain, the data field of the last CNAME. The last CNAME in this sense is that which contains a value which does not resolve to another CNAME." This definition has a certain internal logic, because of the way CNAME substitution works and the definition of CNAME. If a name server does not find an RRset that matches a query, but it finds the same name in the same class with a CNAME record, then the name server "includes the CNAME record in the response and restarts the query at the domain name specified in the data field of the CNAME record." ([RFC1034] Section 3.6.2). This is made

explicit in the resolution algorithm outlined in Section 4.3.2 of [RFC1034], which says to "change QNAME to the canonical name in the CNAME RR, and go back to step 1" in the case of a CNAME RR. Since a CNAME record explicitly declares that the owner name is canonically named what is in the RDATA, then there is a way to view the new name (i.e. the name that was in the RDATA of the CNAME RR) as also being the QNAME.

This creates a kind of confusion, however, because the answer to a query that results in CNAME processing contains in the echoed Question Section one QNAME (the name in the original query), and a second QNAME that is in the data field of the last CNAME. The confusion comes from the iterative/recursive mode of resolution, which finally returns an answer that need not actually have the same owner name as the QNAME contained in the original query.

To address this potential confusion, it is helpful to distinguish between three meanings:

* QNAME (original): The name actually sent in the Question Section in the original query, which is always echoed in the (final) reply in the Question Section when the QR bit is set to 1.

* QNAME (effective): A name actually resolved, which is either the name originally queried, or a name received in a CNAME chain response.

* QNAME (final): The name actually resolved, which is either the name actually queried or else the last name in a CNAME chain response.

Note that, because the definition in [RFC2308] is actually for a different concept than what was in [RFC1034], it would have be better if [RFC2308] had used a different name for that concept. In general use today, QNAME almost always means what is defined above as "QNAME (original)".

Referrals: A type of response in which a server, signaling that it is not (completely) authoritative for an answer, provides the querying resolver with an alternative place to send its query. Referrals can be partial.

A referral arises when a server is not performing recursive service while answering a query. It appears in step 3(b) of the algorithm in [RFC1034], Section 4.3.2.

There are two types of referral response. The first is a downward referral (sometimes described as "delegation response"), where the server is authoritative for some portion of the QNAME. The authority section RRset's RDATA contains the name servers specified at the referred-to zone cut. In normal DNS operation, this kind of response is required in order to find names beneath a delegation. The bare use of "referral" means this kind of referral, and many people believe that this is the only legitimate kind of referral in the DNS.

The second is an upward referral (sometimes described as "root referral"), where the server is not authoritative for any portion of the QNAME. When this happens, the referred-to zone in the authority section is usually the root zone (.). In normal DNS operation, this kind of response is not required for resolution or for correctly answering any query. There is no requirement that any server send upward referrals. Some people regard upward referrals as a sign of a misconfiguration or error. Upward referrals always need some sort of qualifier (such as "upward" or "root"), and are never identified by the bare word "referral".

A response that has only a referral contains an empty answer section. It contains the NS RRset for the referred-to zone in the authority section. It may contain RRs that provide addresses in the additional section. The AA bit is clear.

In the case where the query matches an alias, and the server is not authoritative for the target of the alias but it is authoritative for some name above the target of the alias, the resolution algorithm will produce a response that contains both the authoritative answer for the alias, and also a referral. Such a partial answer and referral response has data in the answer section. It has the NS RRset for the referred-to zone in the authority section. It may contain RRs that provide addresses in the additional section. The AA bit is set, because the first name in the answer section matches the QNAME and the server is authoritative for that answer (see [RFC1035], Section 4.1.1).

5. Resource Records

RR: An acronym for resource record. ([RFC1034], Section 3.6.)

RRset: A set of resource records with the same label, class and type, but with different data. (Definition from [RFC2181]) Also spelled RRSet in some documents. As a clarification, "same label" in this definition means "same owner name". In addition, [RFC2181] states that "the TTLs of all RRs in an RRSet must be the same".

Note that RRSIG resource records do not match this definition. [RFC4035] says: "An RRset MAY have multiple RRSIG RRs associated with it. Note that as RRSIG RRs are closely tied to the RRsets whose signatures they contain, RRSIG RRs, unlike all other DNS RR types, do not form RRsets. In particular, the TTL values among RRSIG RRs with a common owner name do not follow the RRset rules described in [RFC2181]."

Master file: "Master files are text files that contain RRs in text form. Since the contents of a zone can be expressed in the form of a list of RRs a master file is most often used to define a zone, though it can be used to list a cache's contents." ([RFC1035], Section 5.) Master files are sometimes called "zone files".

Presentation format: The text format used in master files. This format is shown but not formally defined in [RFC1034] and [RFC1035]. The term "presentation format" first appears in [RFC4034].

EDNS: The extension mechanisms for DNS, defined in [RFC6891]. Sometimes called "EDNS0" or "EDNS(0)" to indicate the version number. EDNS allows DNS clients and servers to specify message sizes larger than the original 512 octet limit, to expand the response code space, and potentially to carry additional options that affect the handling of a DNS query.

OPT: A pseudo-RR (sometimes called a "meta-RR") that is used only to contain control information pertaining to the question-and-answer sequence of a specific transaction. (Definition from [RFC6891], Section 6.1.1) It is used by EDNS.

Owner: The domain name where a RR is found ([RFC1034], Section 3.6). Often appears in the term "owner name".

SOA field names: DNS documents, including the definitions here, often refer to the fields in the RDATA of an SOA resource record by field name. Those fields are defined in Section 3.3.13 of [RFC1035]. The names (in the order they appear in the SOA RDATA) are MNAME, RNAME, SERIAL, REFRESH, RETRY, EXPIRE, and MINIMUM. Note that the meaning of MINIMUM field is updated in Section 4 of [RFC2308]; the new definition is that the MINIMUM field is only "the TTL to be used for negative responses". This document tends to use field names instead of terms that describe the fields.

TTL: The maximum "time to live" of a resource record. "A TTL value is an unsigned number, with a minimum value of 0, and a maximum value of 2147483647. That is, a maximum of 2^31 - 1. When

transmitted, the TTL is encoded in the less significant 31 bits of the 32 bit TTL field, with the most significant, or sign, bit set to zero." (Quoted from [RFC2181], Section 8) (Note that [RFC1035] erroneously stated that this is a signed integer; that was fixed by [RFC2181].)

The TTL "specifies the time interval that the resource record may be cached before the source of the information should again be consulted". (Quoted from [RFC1035], Section 3.2.1) Also: "the time interval (in seconds) that the resource record may be cached before it should be discarded". (Quoted from [RFC1035], Section 4.1.3). Despite being defined for a resource record, the TTL of every resource record in an RRset is required to be the same ([RFC2181], Section 5.2).

The reason that the TTL is the maximum time to live is that a cache operator might decide to shorten the time to live for operational purposes, such as if there is a policy to disallow TTL values over a certain number. Some servers are known to ignore the TTL on some RRsets (such as when the authoritative data has a very short TTL) even though this is against the advice in RFC 1035. An RRset can be flushed from the cache before the end of the TTL interval, at which point the value of the TTL becomes unknown because the RRset with which it was associated no longer exists.

There is also the concept of a "default TTL" for a zone, which can be a configuration parameter in the server software. This is often expressed by a default for the entire server, and a default for a zone using the $TTL directive in a zone file. The $TTL directive was added to the master file format by [RFC2308].

Class independent: A resource record type whose syntax and semantics are the same for every DNS class. A resource record type that is not class independent has different meanings depending on the DNS class of the record, or the meaning is undefined for some class. Most resorece record types are defined for class 1 (IN, the Internet), but many are undefined for other classes.

Address records: Records whose type is A or AAAA. [RFC2181] informally defines these as "(A, AAAA, etc)". Note that new types of address records could be defined in the future.

6. DNS Servers and Clients

This section defines the terms used for the systems that act as DNS clients, DNS servers, or both. In the RFCs, DNS servers are sometimes called "name servers", "nameservers", or just "servers".

There is no formal definition of DNS server, but the RFCs generally assume that it is an Internet server that listens for queries and sends responses using the DNS protocol defined in [RFC1035] and its successors.

It is important to note that the terms "DNS server" and "name server" require context in order to understand the services being provided. Both authoritative servers and recursive resolvers are often called "DNS servers" and "name servers" even though they serve different roles (and may be part of the same software package).

For terminology specific to the public DNS root server system, see [RSSAC026]. That document defines terms such as "root server", "root server operator", and terms that are specific to the way that the root zone of the public DNS is served.

Resolver: A program "that extract[s] information from name servers in response to client requests." (Quoted from [RFC1034], Section 2.4) "The resolver is located on the same machine as the program that requests the resolver's services, but it may need to consult name servers on other hosts." (Quoted from [RFC1034], Section 5.1) A resolver performs queries for a name, type, and class, and receives answers. The logical function is called "resolution". In practice, the term is usually referring to some specific type of resolver (some of which are defined below), and understanding the use of the term depends on understanding the context.

A related term is "resolve", which is not formally defined in [RFC1034] or [RFC1035]. An imputed definition might be "asking a question that consists of a domain name, class, and type, and receiving some sort of answer". Similarly, an imputed definition of "resolution" might be "the answer received from resolving".

Stub resolver: A resolver that cannot perform all resolution itself. Stub resolvers generally depend on a recursive resolver to undertake the actual resolution function. Stub resolvers are discussed but never fully defined in Section 5.3.1 of [RFC1034]. They are fully defined in Section 6.1.3.1 of [RFC1123].

Iterative mode: A resolution mode of a server that receives DNS queries and responds with a referral to another server. Section 2.3 of [RFC1034] describes this as "The server refers the client to another server and lets the client pursue the query". A resolver that works in iterative mode is sometimes called an "iterative resolver". See also "iterative resolution" later in this section.

Recursive mode: A resolution mode of a server that receives DNS queries and either responds to those queries from a local cache or sends queries to other servers in order to get the final answers to the original queries. Section 2.3 of [RFC1034] describes this as "The first server pursues the query for the client at another server". Section 4.3.1 of of [RFC1034] says "in \[recursive\] mode the name server acts in the role of a resolver and returns either an error or the answer, but never referrals." That same section also says "The recursive mode occurs when a query with RD set arrives at a server which is willing to provide recursive service; the client can verify that recursive mode was used by checking that both RA and RD are set in the reply."

A server operating in recursive mode may be thought of as having a name server side (which is what answers the query) and a resolver side (which performs the resolution function). Systems operating in this mode are commonly called "recursive servers". Sometimes they are called "recursive resolvers". While strictly the difference between these is that one of them sends queries to another recursive server and the other does not, in practice it is not possible to know in advance whether the server that one is querying will also perform recursion; both terms can be observed in use interchangeably.

Recursive resolver: A resolver that acts in recursive mode. In general, a recursive resolver is expected to cache the answers it receives (which would make it a full-service resolver), but some recursive resolvers might not cache.

[RFC4697] tried to differentiate between a recursive resolver and an iterative resolver.

Recursive query: A query with the Recursion Desired (RD) bit set to 1 in the header. (See Section 4.1.1 of [RFC1035].) If recursive service is available and is requested by the RD bit in the query, the server uses its resolver to answer the query. (See Section 4.3.2 of [RFC1035].)

Non-recursive query: A query with the Recursion Desired (RD) bit set to 0 in the header. A server can answer non-recursive queries using only local information: the response contains either an error, the answer, or a referral to some other server "closer" to the answer. (See Section 4.3.1 of [RFC1035].)

Iterative resolution: A name server may be presented with a query that can only be answered by some other server. The two general approaches to dealing with this problem are "recursive", in which the first server pursues the query on behalf of the client at

another server, and "iterative", in which the server refers the client to another server and lets the client pursue the query there. (See Section 2.3 of [RFC1034].)

In iterative resolution, the client repeatedly makes non-recursive queries and follows referrals and/or aliases. The iterative resolution algorithm is described in Section 5.3.3 of [RFC1034].

Full resolver: This term is used in [RFC1035], but it is not defined there. RFC 1123 defines a "full-service resolver" that may or may not be what was intended by "full resolver" in [RFC1035]. This term is not properly defined in any RFC.

Full-service resolver: Section 6.1.3.1 of [RFC1123] defines this term to mean a resolver that acts in recursive mode with a cache (and meets other requirements).

Priming: "The act of finding the list of root servers from a configuration that lists some or all of the purported IP addresses of some or all of those root servers." (Quoted from [RFC8109], Section 2.) In order to operate in recursive mode, a resolver needs to know the address of at least one root server. Priming is most often done from a configuration setting that contains a list of authoritative servers for the root zone.

Root hints: "Operators who manage a DNS recursive resolver typically need to configure a 'root hints file'. This file contains the names and IP addresses of the authoritative name servers for the root zone, so the software can bootstrap the DNS resolution process. For many pieces of software, this list comes built into the software." (Quoted from [IANA_RootFiles]) This file is often used in priming.

Negative caching: "The storage of knowledge that something does not exist, cannot give an answer, or does not give an answer." (Quoted from [RFC2308], Section 1)

Authoritative server: "A server that knows the content of a DNS zone from local knowledge, and thus can answer queries about that zone without needing to query other servers." (Quoted from [RFC2182], Section 2.) It is a system that responds to DNS queries with information about zones for which it has been configured to answer with the AA flag in the response header set to 1. It is a server that has authority over one or more DNS zones. Note that it is possible for an authoritative server to respond to a query without the parent zone delegating authority to that server. Authoritative servers also provide "referrals", usually to child zones delegated from them; these referrals have the AA bit set to

0 and come with referral data in the Authority and (if needed) the Additional sections.

Authoritative-only server: A name server that only serves authoritative data and ignores requests for recursion. It will "not normally generate any queries of its own. Instead, it answers non-recursive queries from iterative resolvers looking for information in zones it serves." (Quoted from [RFC4697], Section 2.4)

Zone transfer: The act of a client requesting a copy of a zone and an authoritative server sending the needed information. (See Section 7 for a description of zones.) There are two common standard ways to do zone transfers: the AXFR ("Authoritative Transfer") mechanism to copy the full zone (described in [RFC5936], and the IXFR ("Incremental Transfer") mechanism to copy only parts of the zone that have changed (described in [RFC1995]). Many systems use non-standard methods for zone transfer outside the DNS protocol.

Secondary server: "An authoritative server which uses zone transfer to retrieve the zone" (Quoted from [RFC1996], Section 2.1). Secondary servers are also discussed in [RFC1034]. [RFC2182] describes secondary servers in more detail. Although early DNS RFCs such as [RFC1996] referred to this as a "slave", the current common usage has shifted to calling it a "secondary".

Slave server: See secondary server.

Primary server: "Any authoritative server configured to be the source of zone transfer for one or more [secondary] servers" (Quoted from [RFC1996], Section 2.1) or, more specifically, "an authoritative server configured to be the source of AXFR or IXFR data for one or more [secondary] servers" (Quoted from [RFC2136]). Primary servers are also discussed in [RFC1034]. Although early DNS RFCs such as [RFC1996] referred to this as a "master", the current common usage has shifted to "primary".

Master server: See primary server.

Primary master: "The primary master is named in the zone’s SOA MNAME field and optionally by an NS RR". (Quoted from [RFC1996], Section 2.1). [RFC2136] defines "primary master" as "Master server at the root of the AXFR/IXFR dependency graph. The primary master is named in the zone’s SOA MNAME field and optionally by an NS RR. There is by definition only one primary master server per zone." The idea of a primary master is only used by [RFC2136], and is considered archaic in other parts of the DNS.

The idea of a primary master is only used in [RFC1996] and [RFC2136]. A modern interpretation of the term "primary master" is a server that is both authoritative for a zone and that gets its updates to the zone from configuration (such as a master file) or from UPDATE transactions.

Stealth server: This is "like a slave server except not listed in an NS RR for the zone." (Quoted from [RFC1996], Section 2.1)

Hidden master: A stealth server that is a primary server for zone transfers. "In this arrangement, the master name server that processes the updates is unavailable to general hosts on the Internet; it is not listed in the NS RRset." (Quoted from [RFC6781], Section 3.4.3). An earlier RFC, [RFC4641], said that the hidden master's name "appears in the SOA RRs MNAME field", although in some setups, the name does not appear at all in the public DNS. A hidden master can also be a secondary server for the zone itself.

Forwarding: The process of one server sending a DNS query with the RD bit set to 1 to another server to resolve that query. Forwarding is a function of a DNS resolver; it is different than simply blindly relaying queries.

[RFC5625] does not give a specific definition for forwarding, but describes in detail what features a system that forwards needs to support. Systems that forward are sometimes called "DNS proxies", but that term has not yet been defined (even in [RFC5625]).

Forwarder: Section 1 of [RFC2308] describes a forwarder as "a nameserver used to resolve queries instead of directly using the authoritative nameserver chain". [RFC2308] further says "The forwarder typically either has better access to the internet, or maintains a bigger cache which may be shared amongst many resolvers." That definition appears to suggest that forwarders normally only query authoritative servers. In current use, however, forwarders often stand between stub resolvers and recursive servers. [RFC2308] is silent on whether a forwarder is iterative-only or can be a full-service resolver.

Policy-implementing resolver: A resolver acting in recursive mode that changes some of the answers that it returns based on policy criteria, such as to prevent access to malware sites or objectionable content. In general, a stub resolver has no idea whether upstream resolvers implement such policy or, if they do, the exact policy about what changes will be made. In some cases, the user of the stub resolver has selected the policy-implementing resolver with the explicit intention of using it to implement the

policies. In other cases, policies are imposed without the user of the stub resolver being informed.

Open resolver: A full-service resolver that accepts and processes queries from any (or nearly any) stub resolver. This is sometimes also called a "public resolver", although the term "public resolver" is used more with open resolvers that are meant to be open, as compared to the vast majority of open resolvers that are probably misconfigured to be open. Open resolvers are discussed in [RFC5358]

Split DNS: The terms "split DNS" and "split-horizon DNS" have long been used in the DNS community without formal definition. In general, they refer to situations in which DNS servers that are authoritative for a particular set of domains provide partly or completely different answers in those domains depending on the source of the query. The effect of this is that a domain name that is notionally globally unique nevertheless has different meanings for different network users. This can sometimes be the result of a "view" configuration, described below.

[RFC2775], Section 3.8 gives a related definition that is too specific to be generally useful.

View: A configuration for a DNS server that allows it to provide different answers depending on attributes of the query, such as for "split DNS". Typically, views differ by the source IP address of a query, but can also be based on the destination IP address, the type of query (such as AXFR), whether it is recursive, and so on. Views are often used to provide more names or different addresses to queries from "inside" a protected network than to those "outside" that network. Views are not a standardized part of the DNS, but they are widely implemented in server software.

Passive DNS: A mechanism to collect DNS data by storing DNS responses from name servers. Some of these systems also collect the DNS queries associated with the responses, although doing so raises some privacy concerns. Passive DNS databases can be used to answer historical questions about DNS zones such as which values were present at a given time in the past, or when a name was spotted first. Passive DNS databases allow searching of the stored records on keys other than just the name and type, such as "find all names which have A records of a particular value".

Anycast: "The practice of making a particular service address available in multiple, discrete, autonomous locations, such that datagrams sent are routed to one of several available locations."

(Quoted from [RFC4786], Section 2) See [RFC4786] for more detail on Anycast and other terms that are specific to its use.

Instance: "When anycast routing is used to allow more than one server to have the same IP address, each one of those servers is commonly referred to as an 'instance'." "An instance of a server, such as a root server, is often referred to as an 'Anycast instance'." (Quoted from [RSSAC026])

Privacy-enabling DNS server: "A DNS server that implements DNS over TLS [RFC7858] and may optionally implement DNS over DTLS [RFC8094]." (Quoted from [RFC8310], Section 2)

7. Zones

This section defines terms that are used when discussing zones that are being served or retrieved.

Zone: "Authoritative information is organized into units called 'zones', and these zones can be automatically distributed to the name servers which provide redundant service for the data in a zone." (Quoted from [RFC1034], Section 2.4)

Child: "The entity on record that has the delegation of the domain from the Parent." (Quoted from [RFC7344], Section 1.1)

Parent: "The domain in which the Child is registered." (Quoted from [RFC7344], Section 1.1) Earlier, "parent name server" was defined in [RFC0882] as "the name server that has authority over the place in the domain name space that will hold the new domain". (Note that [RFC0882] was obsoleted by [RFC1034] and [RFC1035].) [RFC0819] also has some description of the relationship between parents and children.

Origin:

(a) "The domain name that appears at the top of a zone (just below the cut that separates the zone from its parent). The name of the zone is the same as the name of the domain at the zone's origin." (Quoted from [RFC2181], Section 6.) These days, this sense of "origin" and "apex" (defined below) are often used interchangeably.

(b) The domain name within which a given relative domain name appears in zone files. Generally seen in the context of "$ORIGIN", which is a control entry defined in [RFC1035], Section 5.1, as part of the master file format. For example, if

the $ORIGIN is set to "example.org.", then a master file line for "www" is in fact an entry for "www.example.org.".

Apex: The point in the tree at an owner of an SOA and corresponding authoritative NS RRset. This is also called the "zone apex". [RFC4033] defines it as "the name at the child's side of a zone cut". The "apex" can usefully be thought of as a data-theoretic description of a tree structure, and "origin" is the name of the same concept when it is implemented in zone files. The distinction is not always maintained in use, however, and one can find uses that conflict subtly with this definition. [RFC1034] uses the term "top node of the zone" as a synonym of "apex", but that term is not widely used. These days, the first sense of "origin" (above) and "apex" are often used interchangeably.

Zone cut: The delimitation point between two zones where the origin of one of the zones is the child of the other zone.

"Zones are delimited by 'zone cuts'. Each zone cut separates a 'child' zone (below the cut) from a 'parent' zone (above the cut)." (Quoted from [RFC2181], Section 6; note that this is barely an ostensive definition.) Section 4.2 of [RFC1034] uses "cuts" instead of "zone cut".

Delegation: The process by which a separate zone is created in the name space beneath the apex of a given domain. Delegation happens when an NS RRset is added in the parent zone for the child origin. Delegation inherently happens at a zone cut. The term is also commonly a noun: the new zone that is created by the act of delegating.

Authoritative data: "All of the RRs attached to all of the nodes from the top node of the zone down to leaf nodes or nodes above cuts around the bottom edge of the zone." (Quoted from [RFC1034], Section 4.2.1) Note that this definition might inadvertently also cause any NS records that appear in the zone to be included, even those that might not truly be authoritative because there are identical NS RRs below the zone cut. This reveals the ambiguity in the notion of authoritative data, because the parent-side NS records authoritatively indicate the delegation, even though they are not themselves authoritative data.

Lame delegation: "A lame delegations exists when a nameserver is delegated responsibility for providing nameservice for a zone (via NS records) but is not performing nameservice for that zone (usually because it is not set up as a primary or secondary for the zone)." (Definition from [RFC1912], Section 2.8)

Glue records: "[Resource records] which are not part of the authoritative data [of the zone], and are address resource records for the [name servers in subzones]. These RRs are only necessary if the name server's name is 'below' the cut, and are only used as part of a referral response." Without glue "we could be faced with the situation where the NS RRs tell us that in order to learn a name server's address, we should contact the server using the address we wish to learn." (Definition from [RFC1034], Section 4.2.1)

A later definition is that glue "includes any record in a zone file that is not properly part of that zone, including nameserver records of delegated sub-zones (NS records), address records that accompany those NS records (A, AAAA, etc), and any other stray data that might appear" ([RFC2181], Section 5.4.1). Although glue is sometimes used today with this wider definition in mind, the context surrounding the [RFC2181] definition suggests it is intended to apply to the use of glue within the document itself and not necessarily beyond.

Bailiwick: "In-bailiwick" is an adjective to describe a name server whose name is either a subdomain of or (rarely) the same as the origin of the zone that contains the delegation to the name server. In-bailiwick name servers may have glue records in their parent zone (using the first of the definitions of "glue records" in the definition above). (The term "bailiwick" means the district or territory where a bailiff or policeman has jurisdiction.)

"In-bailiwick" names are divided into two type of name server names: "in-domain" names and "sibling domain" names.

* In-domain: an adjective to describe a name server whose name is either subordinate to or (rarely) the same as the owner name of the NS resource records. An in-domain name server name MUST have glue records or name resolution fails. For example, a delegation for "child.example.com" may have "in-domain" name server name "ns.child.example.com".

* Sibling domain: a name server's name that is either subordinate to or (rarely) the same as the zone origin and not subordinate to or the same as the owner name of the NS resource records. Glue records for sibling domains are allowed, but not necessary. For example, a delegation for "child.example.com" in "example.com" zone may have "sibling" name server name "ns.another.example.com".

"Out-of-bailiwick" is the antonym of in-bailiwick. An adjective to describe a name server whose name is not subordinate to or the same as the zone origin. Glue records for out-of-bailiwick name servers are useless. Following table shows examples of delegation types.

Delegation	Parent	Name Server Name	Type
com	.	a.gtld-servers.net	in-bailiwick / sibling domain
net	.	a.gtld-servers.net	in-bailiwick / in-domain
example.org	org	ns.example.org	in-bailiwick / in-domain
example.org	org	ns.ietf.org	in-bailiwick / sibling domain
example.org	org	ns.example.com	out-of-bailiwick
example.jp	jp	ns.example.jp	in-bailiwick / in-domain
example.jp	jp	ns.example.ne.jp	in-bailiwick / sibling domain
example.jp	jp	ns.example.com	out-of-bailiwick

Root zone: The zone of a DNS-based tree whose apex is the zero-length label. Also sometimes called "the DNS root".

Empty non-terminals (ENT): "Domain names that own no resource records but have subdomains that do." (Quoted from [RFC4592], Section 2.2.2.) A typical example is in SRV records: in the name "_sip._tcp.example.com", it is likely that "_tcp.example.com" has no RRsets, but that "_sip._tcp.example.com" has (at least) an SRV RRset.

Delegation-centric zone: A zone that consists mostly of delegations to child zones. This term is used in contrast to a zone that might have some delegations to child zones, but also has many data resource records for the zone itself and/or for child zones. The term is used in [RFC4956] and [RFC5155], but is not defined there.

Occluded name: "The addition of a delegation point via dynamic update will render all subordinate domain names to be in a limbo, still part of the zone, but not available to the lookup process. The addition of a DNAME resource record has the same impact. The subordinate names are said to be ’occluded’." (Quoted from [RFC5936], Section 3.5)

Fast flux DNS: This "occurs when a domain is found in DNS using A records to multiple IP addresses, each of which has a very short Time-to-Live (TTL) value associated with it. This means that the domain resolves to varying IP addresses over a short period of time." (Quoted from [RFC6561], Section 1.1.5, with typo corrected) In addition to having legitimate uses, fast flux DNS can used to deliver malware. Because the addresses change so rapidly, it is difficult to ascertain all the hosts. It should be

noted that the technique also works with AAAA records, but such use is not frequently observed on the Internet as of this writing.

Reverse DNS, reverse lookup: "The process of mapping an address to a name is generally known as a 'reverse lookup', and the IN-ADDR.ARPA and IP6.ARPA zones are said to support the 'reverse DNS'." (Quoted from [RFC5855], Section 1)

Forward lookup: "Hostname-to-address translation". (Quoted from [RFC2133], Section 6)

arpa: Address and Routing Parameter Area Domain: "The 'arpa' domain was originally established as part of the initial deployment of the DNS, to provide a transition mechanism from the Host Tables that were common in the ARPANET, as well as a home for the IPv4 reverse mapping domain. During 2000, the abbreviation was redesignated to 'Address and Routing Parameter Area' in the hope of reducing confusion with the earlier network name." (Quoted from [RFC3172], Section 2.) .arpa is an "infrastructure domain", a domain whose "role is to support the operating infrastructure of the Internet". (Quoted from [RFC3172], Section 2.) See [RFC3172] for more history of this name.

Service name: "Service names are the unique key in the Service Name and Transport Protocol Port Number registry. This unique symbolic name for a service may also be used for other purposes, such as in DNS SRV records." (Quoted from [RFC6335], Section 5.)

8. Wildcards

Wildcard: [RFC1034] defined "wildcard", but in a way that turned out to be confusing to implementers. For an extended discussion of wildcards, including clearer definitions, see [RFC4592]. Special treatment is given to RRs with owner names starting with the label "*". "Such RRs are called 'wildcards'. Wildcard RRs can be thought of as instructions for synthesizing RRs." (Quoted from [RFC1034], Section 4.3.3)

Asterisk label: "The first octet is the normal label type and length for a 1-octet-long label, and the second octet is the ASCII representation for the '*' character. A descriptive name of a label equaling that value is an 'asterisk label'." (Quoted from [RFC4592], Section 2.1.1)

Wildcard domain name: "A 'wildcard domain name' is defined by having its initial (i.e., leftmost or least significant) label be asterisk label." (Quoted from [RFC4592], Section 2.1.1)

Closest encloser: "The longest existing ancestor of a name." (Quoted from [RFC5155], Section 1.3) An earlier definition is "The node in the zone's tree of existing domain names that has the most labels matching the query name (consecutively, counting from the root label downward). Each match is a 'label match' and the order of the labels is the same." (Quoted from [RFC4592], Section 3.3.1)

Closest provable encloser: "The longest ancestor of a name that can be proven to exist. Note that this is only different from the closest encloser in an Opt-Out zone." (Quoted from [RFC5155], Section 1.3)

Next closer name: "The name one label longer than the closest provable encloser of a name." (Quoted from [RFC5155], Section 1.3)

Source of Synthesis: "The source of synthesis is defined in the context of a query process as that wildcard domain name immediately descending from the closest encloser, provided that this wildcard domain name exists. 'Immediately descending' means that the source of synthesis has a name of the form: <asterisk label>.<closest encloser>." (Quoted from [RFC4592], Section 3.3.1)

9. Registration Model

Registry: The administrative operation of a zone that allows registration of names within that zone. People often use this term to refer only to those organizations that perform registration in large delegation-centric zones (such as TLDs); but formally, whoever decides what data goes into a zone is the registry for that zone. This definition of "registry" is from a DNS point of view; for some zones, the policies that determine what can go in the zone are decided by superior zones and not the registry operator.

Registrant: An individual or organization on whose behalf a name in a zone is registered by the registry. In many zones, the registry and the registrant may be the same entity, but in TLDs they often are not.

Registrar: A service provider that acts as a go-between for registrants and registries. Not all registrations require a registrar, though it is common to have registrars involved in registrations in TLDs.

EPP: The Extensible Provisioning Protocol (EPP), which is commonly used for communication of registration information between registries and registrars. EPP is defined in [RFC5730].

WHOIS: A protocol specified in [RFC3912], often used for querying registry databases. WHOIS data is frequently used to associate registration data (such as zone management contacts) with domain names. The term "WHOIS data" is often used as a synonym for the registry database, even though that database may be served by different protocols, particularly RDAP. The WHOIS protocol is also used with IP address registry data.

RDAP: The Registration Data Access Protocol, defined in [RFC7480], [RFC7481], [RFC7482], [RFC7483], [RFC7484], and [RFC7485]. The RDAP protocol and data format are meant as a replacement for WHOIS.

DNS operator: An entity responsible for running DNS servers. For a zone's authoritative servers, the registrant may act as their own DNS operator, or their registrar may do it on their behalf, or they may use a third-party operator. For some zones, the registry function is performed by the DNS operator plus other entities who decide about the allowed contents of the zone.

Public suffix: "A domain that is controlled by a public registry." (Quoted from [RFC6265], Section 5.3) A common definition for this term is a domain under which subdomains can be registered by third parties, and on which HTTP cookies (which are described in detail in [RFC6265]) should not be set. There is no indication in a domain name whether it is a public suffix; that can only be determined by outside means. In fact, both a domain and a subdomain of that domain can be public suffixes.

There is nothing inherent in a domain name to indicate whether it is a public suffix. One resource for identifying public suffixes is the Public Suffix List (PSL) maintained by Mozilla (http://publicsuffix.org/).

For example, at the time this document is published, the "com.au" domain is listed as a public suffix in the PSL. (Note that this example might change in the future.)

Note that the term "public suffix" is controversial in the DNS community for many reasons, and may be significantly changed in the future. One example of the difficulty of calling a domain a public suffix is that designation can change over time as the registration policy for the zone changes, such as was the case with the "uk" TLD in 2014.

Subordinate and Superordinate: These terms are introduced in [RFC3731] for use in the registration model, but not defined there. Instead, they are given in examples. "For example, domain name 'example.com' has a superordinate relationship to host name ns1.example.com'." "For example, host ns1.example1.com is a subordinate host of domain example1.com, but it is a not a subordinate host of domain example2.com." (Quoted from [RFC3731], Section 1.1.) These terms are strictly ways of referring to the relationship standing of two domains where one is a subdomain of the other.

10. General DNSSEC

Most DNSSEC terms are defined in [RFC4033], [RFC4034], [RFC4035], and [RFC5155]. The terms that have caused confusion in the DNS community are highlighted here.

DNSSEC-aware and DNSSEC-unaware: These two terms, which are used in some RFCs, have not been formally defined. However, Section 2 of [RFC4033] defines many types of resolvers and validators, including "non-validating security-aware stub resolver", "non-validating stub resolver", "security-aware name server", "security-aware recursive name server", "security-aware resolver", "security-aware stub resolver", and "security-oblivious 'anything'". (Note that the term "validating resolver", which is used in some places in DNSSEC-related documents, is also not defined in those RFCs, but is defined below.)

Signed zone: "A zone whose RRsets are signed and that contains properly constructed DNSKEY, Resource Record Signature (RRSIG), Next Secure (NSEC), and (optionally) DS records." (Quoted from [RFC4033], Section 2.) It has been noted in other contexts that the zone itself is not really signed, but all the relevant RRsets in the zone are signed. Nevertheless, if a zone that should be signed contains any RRsets that are not signed (or opted out), those RRsets will be treated as bogus, so the whole zone needs to be handled in some way.

It should also be noted that, since the publication of [RFC6840], NSEC records are no longer required for signed zones: a signed zone might include NSEC3 records instead. [RFC7129] provides additional background commentary and some context for the NSEC and NSEC3 mechanisms used by DNSSEC to provide authenticated denial-of-existence responses. NSEC and NSEC3 are described below.

Unsigned zone: Section 2 of [RFC4033] defines this as "a zone that is not signed". Section 2 of [RFC4035] defines this as "A zone that does not include these records [properly constructed DNSKEY,

Resource Record Signature (RRSIG), Next Secure (NSEC), and (optionally) DS records] according to the rules in this section". There is an important note at the end of Section 5.2 of [RFC4035] that defines an additional situation in which a zone is considered unsigned: "If the resolver does not support any of the algorithms listed in an authenticated DS RRset, then the resolver will not be able to verify the authentication path to the child zone. In this case, the resolver SHOULD treat the child zone as if it were unsigned."

NSEC: "The NSEC record allows a security-aware resolver to authenticate a negative reply for either name or type non-existence with the same mechanisms used to authenticate other DNS replies." (Quoted from [RFC4033], Section 3.2.) In short, an NSEC record provides authenticated denial of existence.

"The NSEC resource record lists two separate things: the next owner name (in the canonical ordering of the zone) that contains authoritative data or a delegation point NS RRset, and the set of RR types present at the NSEC RR's owner name." (Quoted from Section 4 of RFC 4034)

NSEC3: Like the NSEC record, the NSEC3 record also provides authenticated denial of existence; however, NSEC3 records mitigate against zone enumeration and support Opt-Out. NSEC resource records require associated NSEC3PARAM resource records. NSEC3 and NSEC3PARAM resource records are defined in [RFC5155].

Note that [RFC6840] says that [RFC5155] "is now considered part of the DNS Security Document Family as described by Section 10 of [RFC4033]". This means that some of the definitions from earlier RFCs that only talk about NSEC records should probably be considered to be talking about both NSEC and NSEC3.

Opt-out: "The Opt-Out Flag indicates whether this NSEC3 RR may cover unsigned delegations." (Quoted from [RFC5155], Section 3.1.2.1.) Opt-out tackles the high costs of securing a delegation to an insecure zone. When using Opt-Out, names that are an insecure delegation (and empty non-terminals that are only derived from insecure delegations) don't require an NSEC3 record or its corresponding RRSIG records. Opt-Out NSEC3 records are not able to prove or deny the existence of the insecure delegations. (Adapted from [RFC7129], Section 5.1)

Insecure delegation: "A signed name containing a delegation (NS RRset), but lacking a DS RRset, signifying a delegation to an unsigned subzone." (Quoted from [RFC4956], Section 2.)

Zone enumeration: "The practice of discovering the full content of a zone via successive queries." (Quoted from [RFC5155], Section 1.3.) This is also sometimes called "zone walking". Zone enumeration is different from zone content guessing where the guesser uses a large dictionary of possible labels and sends successive queries for them, or matches the contents of NSEC3 records against such a dictionary.

Validation: Validation, in the context of DNSSEC, refers to one of the following:

* Checking the validity of DNSSEC signatures

* Checking the validity of DNS responses, such as those including authenticated denial of existence

* Building an authentication chain from a trust anchor to a DNS response or individual DNS RRsets in a response

The first two definitions above consider only the validity of individual DNSSEC components such as the RRSIG validity or NSEC proof validity. The third definition considers the components of the entire DNSSEC authentication chain, and thus requires "configured knowledge of at least one authenticated DNSKEY or DS RR" (as described in [RFC4035], Section 5).

[RFC4033], Section 2, says that a "Validating Security-Aware Stub Resolver... performs signature validation" and uses a trust anchor "as a starting point for building the authentication chain to a signed DNS response", and thus uses the first and third definitions above. The process of validating an RRSIG resource record is described in [RFC4035], Section 5.3.

[RFC5155] refers to validating responses throughout the document, in the context of hashed authenticated denial of existence; this uses the second definition above.

The term "authentication" is used interchangeably with "validation", in the sense of the third definition above. [RFC4033], Section 2, describes the chain linking trust anchor to DNS data as the "authentication chain". A response is considered to be authentic if "all RRsets in the Answer and Authority sections of the response [are considered] to be authentic" ([RFC4035]). DNS data or responses deemed to be authentic or validated have a security status of "secure" ([RFC4035], Section 4.3; [RFC4033], Section 5). "Authenticating both DNS keys and data is a matter of local policy, which may extend or even

override the [DNSSEC] protocol extensions" ([RFC4033], Section 3.1).

The term "verification", when used, is usually synonym for "validation".

Validating resolver: A security-aware recursive name server, security-aware resolver, or security-aware stub resolver that is applying at least one of the definitions of validation (above), as appropriate to the resolution context. For the same reason that the generic term "resolver" is sometimes ambiguous and needs to be evaluated in context (see Section 6), "validating resolver" is a context-sensitive term.

Key signing key (KSK): DNSSEC keys that "only sign the apex DNSKEY RRset in a zone."(Quoted from [RFC6781], Section 3.1)

Zone signing key (ZSK): "DNSSEC keys that can be used to sign all the RRsets in a zone that require signatures, other than the apex DNSKEY RRset." (Quoted from [RFC6781], Section 3.1) Also note that a ZSK is sometimes used to sign the apex DNSKEY RRset.

Combined signing key (CSK): "In cases where the differentiation between the KSK and ZSK is not made, i.e., where keys have the role of both KSK and ZSK, we talk about a Single-Type Signing Scheme." (Quoted from [RFC6781], Section 3.1) This is sometimes called a "combined signing key" or CSK. It is operational practice, not protocol, that determines whether a particular key is a ZSK, a KSK, or a CSK.

Secure Entry Point (SEP): A flag in the DNSKEY RDATA that "can be used to distinguish between keys that are intended to be used as the secure entry point into the zone when building chains of trust, i.e., they are (to be) pointed to by parental DS RRs or configured as a trust anchor. Therefore, it is suggested that the SEP flag be set on keys that are used as KSKs and not on keys that are used as ZSKs, while in those cases where a distinction between a KSK and ZSK is not made (i.e., for a Single-Type Signing Scheme), it is suggested that the SEP flag be set on all keys." (Quoted from [RFC6781], Section 3.2.3.) Note that the SEP flag is only a hint, and its presence or absence may not be used to disqualify a given DNSKEY RR from use as a KSK or ZSK during validation.

The original definition of SEPs was in [RFC3757]. That definition clearly indicated that the SEP was a key, not just a bit in the key. The abstract of [RFC3757] says: "With the Delegation Signer (DS) resource record (RR), the concept of a public key acting as a

secure entry point (SEP) has been introduced. During exchanges of public keys with the parent there is a need to differentiate SEP keys from other public keys in the Domain Name System KEY (DNSKEY) resource record set. A flag bit in the DNSKEY RR is defined to indicate that DNSKEY is to be used as a SEP." That definition of the SEP as a key was made obsolete by [RFC4034], and the definition from [RFC6781] is consistent with [RFC4034].

Trust anchor: "A configured DNSKEY RR or DS RR hash of a DNSKEY RR. A validating security-aware resolver uses this public key or hash as a starting point for building the authentication chain to a signed DNS response. In general, a validating resolver will have to obtain the initial values of its trust anchors via some secure or trusted means outside the DNS protocol." (Quoted from [RFC4033], Section 2)

DNSSEC Policy (DP): A statement that "sets forth the security requirements and standards to be implemented for a DNSSEC-signed zone." (Quoted from [RFC6841], Section 2)

DNSSEC Practice Statement (DPS): "A practices disclosure document that may support and be a supplemental document to the DNSSEC Policy (if such exists), and it states how the management of a given zone implements procedures and controls at a high level." (Quoted from [RFC6841], Section 2)

Hardware security module (HSM): A specialized piece of hardware that is used to create keys for signatures and to sign messages. In DNSSEC, HSMs are often used to hold the private keys for KSKs and ZSKs and to create the signatures used in RRSIG records at periodic intervals.

Signing software: Authoritative DNS servers that support DNSSEC often contain software that facilitates the creation and maintenance of DNSSEC signatures in zones. There is also stand-alone software that can be used to sign a zone regardless of whether the authoritative server itself supports signing. Sometimes signing software can support particular HSMs as part of the signing process.

11. DNSSEC States

A validating resolver can determine that a response is in one of four states: secure, insecure, bogus, or indeterminate. These states are defined in [RFC4033] and [RFC4035], although the two definitions differ a bit. This document makes no effort to reconcile the two definitions, and takes no position as to whether they need to be reconciled.

Section 5 of [RFC4033] says:

> A validating resolver can determine the following 4 states:
>
> Secure: The validating resolver has a trust anchor, has a chain of trust, and is able to verify all the signatures in the response.
>
> Insecure: The validating resolver has a trust anchor, a chain of trust, and, at some delegation point, signed proof of the non-existence of a DS record. This indicates that subsequent branches in the tree are provably insecure. A validating resolver may have a local policy to mark parts of the domain space as insecure.
>
> Bogus: The validating resolver has a trust anchor and a secure delegation indicating that subsidiary data is signed, but the response fails to validate for some reason: missing signatures, expired signatures, signatures with unsupported algorithms, data missing that the relevant NSEC RR says should be present, and so forth.
>
> Indeterminate: There is no trust anchor that would indicate that a specific portion of the tree is secure. This is the default operation mode.

Section 4.3 of [RFC4035] says:

A security-aware resolver must be able to distinguish between four cases:

Secure: An RRset for which the resolver is able to build a chain of signed DNSKEY and DS RRs from a trusted security anchor to the RRset. In this case, the RRset should be signed and is subject to signature validation, as described above.

Insecure: An RRset for which the resolver knows that it has no chain of signed DNSKEY and DS RRs from any trusted starting point to the RRset. This can occur when the target RRset lies in an unsigned zone or in a descendent [sic] of an unsigned zone. In this case, the RRset may or may not be signed, but the resolver will not be able to verify the signature.

Bogus: An RRset for which the resolver believes that it ought to be able to establish a chain of trust but for which it is unable to do so, either due to signatures that for some reason fail to validate or due to missing data that the relevant DNSSEC RRs indicate should be present. This case may indicate an attack but may also indicate a configuration error or some form of data corruption.

Indeterminate: An RRset for which the resolver is not able to determine whether the RRset should be signed, as the resolver is not able to obtain the necessary DNSSEC RRs. This can occur when the security-aware resolver is not able to contact security-aware name servers for the relevant zones.

12. Security Considerations

These definitions do not change any security considerations for the DNS.

13. IANA Considerations

None.

14. References

14.1. Normative References

[IANA_RootFiles]
Internet Assigned Numbers Authority, "IANA Root Files", 2016, <http://www.iana.org/domains/root/files>.

[RFC0882] Mockapetris, P., "Domain names: Concepts and facilities", RFC 882, DOI 10.17487/RFC0882, November 1983, <https://www.rfc-editor.org/info/rfc882>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[RFC1912] Barr, D., "Common DNS Operational and Configuration Errors", RFC 1912, DOI 10.17487/RFC1912, February 1996, <https://www.rfc-editor.org/info/rfc1912>.

[RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, DOI 10.17487/RFC1996, August 1996, <https://www.rfc-editor.org/info/rfc1996>.

[RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <https://www.rfc-editor.org/info/rfc2136>.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <https://www.rfc-editor.org/info/rfc2181>.

[RFC2182] Elz, R., Bush, R., Bradner, S., and M. Patton, "Selection and Operation of Secondary DNS Servers", BCP 16, RFC 2182, DOI 10.17487/RFC2182, July 1997, <https://www.rfc-editor.org/info/rfc2182>.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <https://www.rfc-editor.org/info/rfc2308>.

[RFC3731] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Domain Name Mapping", RFC 3731, DOI 10.17487/RFC3731, March 2004, <https://www.rfc-editor.org/info/rfc3731>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <https://www.rfc-editor.org/info/rfc4034>.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <https://www.rfc-editor.org/info/rfc4035>.

[RFC4592] Lewis, E., "The Role of Wildcards in the Domain Name System", RFC 4592, DOI 10.17487/RFC4592, July 2006, <https://www.rfc-editor.org/info/rfc4592>.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, DOI 10.17487/RFC5155, March 2008, <https://www.rfc-editor.org/info/rfc5155>.

[RFC5358] Damas, J. and F. Neves, "Preventing Use of Recursive Nameservers in Reflector Attacks", BCP 140, RFC 5358, DOI 10.17487/RFC5358, October 2008, <https://www.rfc-editor.org/info/rfc5358>.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <https://www.rfc-editor.org/info/rfc5730>.

[RFC5855] Abley, J. and T. Manderson, "Nameservers for IPv4 and IPv6 Reverse Zones", BCP 155, RFC 5855, DOI 10.17487/RFC5855, May 2010, <https://www.rfc-editor.org/info/rfc5855>.

[RFC5936] Lewis, E. and A. Hoenes, Ed., "DNS Zone Transfer Protocol (AXFR)", RFC 5936, DOI 10.17487/RFC5936, June 2010, <https://www.rfc-editor.org/info/rfc5936>.

[RFC6561] Livingood, J., Mody, N., and M. O'Reirdan, "Recommendations for the Remediation of Bots in ISP Networks", RFC 6561, DOI 10.17487/RFC6561, March 2012, <https://www.rfc-editor.org/info/rfc6561>.

[RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <https://www.rfc-editor.org/info/rfc6781>.

[RFC6840] Weiler, S., Ed. and D. Blacka, Ed., "Clarifications and Implementation Notes for DNS Security (DNSSEC)", RFC 6840, DOI 10.17487/RFC6840, February 2013, <https://www.rfc-editor.org/info/rfc6840>.

[RFC6841] Ljunggren, F., Eklund Lowinder, AM., and T. Okubo, "A Framework for DNSSEC Policies and DNSSEC Practice Statements", RFC 6841, DOI 10.17487/RFC6841, January 2013, <https://www.rfc-editor.org/info/rfc6841>.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <https://www.rfc-editor.org/info/rfc6891>.

[RFC7344] Kumari, W., Gudmundsson, O., and G. Barwood, "Automating DNSSEC Delegation Trust Maintenance", RFC 7344, DOI 10.17487/RFC7344, September 2014, <https://www.rfc-editor.org/info/rfc7344>.

[RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <https://www.rfc-editor.org/info/rfc7719>.

[RFC8310] Dickinson, S., Gillmor, D., and T. Reddy, "Usage Profiles for DNS over TLS and DNS over DTLS", RFC 8310, DOI 10.17487/RFC8310, March 2018, <https://www.rfc-editor.org/info/rfc8310>.

14.2. Informative References

[IANA_Resource_Registry] Internet Assigned Numbers Authority, "Resource Record (RR) TYPEs", 2017, <http://www.iana.org/assignments/dns-parameters/>.

[RFC0819] Su, Z. and J. Postel, "The Domain Naming Convention for Internet User Applications", RFC 819, DOI 10.17487/RFC0819, August 1982, <https://www.rfc-editor.org/info/rfc819>.

[RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, DOI 10.17487/RFC0952, October 1985, <https://www.rfc-editor.org/info/rfc952>.

[RFC1995] Ohta, M., "Incremental Zone Transfer in DNS", RFC 1995, DOI 10.17487/RFC1995, August 1996, <https://www.rfc-editor.org/info/rfc1995>.

[RFC2133] Gilligan, R., Thomson, S., Bound, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 2133, DOI 10.17487/RFC2133, April 1997, <https://www.rfc-editor.org/info/rfc2133>.

[RFC2775] Carpenter, B., "Internet Transparency", RFC 2775, DOI 10.17487/RFC2775, February 2000, <https://www.rfc-editor.org/info/rfc2775>.

[RFC3172] Huston, G., Ed., "Management Guidelines & Operational Requirements for the Address and Routing Parameter Area Domain ("arpa")", BCP 52, RFC 3172, DOI 10.17487/RFC3172, September 2001, <https://www.rfc-editor.org/info/rfc3172>.

[RFC3425] Lawrence, D., "Obsoleting IQUERY", RFC 3425, DOI 10.17487/RFC3425, November 2002, <https://www.rfc-editor.org/info/rfc3425>.

[RFC3757] Kolkman, O., Schlyter, J., and E. Lewis, "Domain Name System KEY (DNSKEY) Resource Record (RR) Secure Entry Point (SEP) Flag", RFC 3757, DOI 10.17487/RFC3757, April 2004, <https://www.rfc-editor.org/info/rfc3757>.

[RFC3912] Daigle, L., "WHOIS Protocol Specification", RFC 3912, DOI 10.17487/RFC3912, September 2004, <https://www.rfc-editor.org/info/rfc3912>.

[RFC4641] Kolkman, O. and R. Gieben, "DNSSEC Operational Practices", RFC 4641, DOI 10.17487/RFC4641, September 2006, <https://www.rfc-editor.org/info/rfc4641>.

[RFC4697] Larson, M. and P. Barber, "Observed DNS Resolution Misbehavior", BCP 123, RFC 4697, DOI 10.17487/RFC4697, October 2006, <https://www.rfc-editor.org/info/rfc4697>.

[RFC4786] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, DOI 10.17487/RFC4786, December 2006, <https://www.rfc-editor.org/info/rfc4786>.

[RFC4956] Arends, R., Kosters, M., and D. Blacka, "DNS Security (DNSSEC) Opt-In", RFC 4956, DOI 10.17487/RFC4956, July 2007, <https://www.rfc-editor.org/info/rfc4956>.

[RFC5625] Bellis, R., "DNS Proxy Implementation Guidelines", BCP 152, RFC 5625, DOI 10.17487/RFC5625, August 2009, <https://www.rfc-editor.org/info/rfc5625>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <https://www.rfc-editor.org/info/rfc5891>.

[RFC5892] Faltstrom, P., Ed., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, DOI 10.17487/RFC5892, August 2010, <https://www.rfc-editor.org/info/rfc5892>.

[RFC5893] Alvestrand, H., Ed. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, DOI 10.17487/RFC5893, August 2010, <https://www.rfc-editor.org/info/rfc5893>.

[RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, DOI 10.17487/RFC5894, August 2010, <https://www.rfc-editor.org/info/rfc5894>.

[RFC6055] Thaler, D., Klensin, J., and S. Cheshire, "IAB Thoughts on Encodings for Internationalized Domain Names", RFC 6055, DOI 10.17487/RFC6055, February 2011, <https://www.rfc-editor.org/info/rfc6055>.

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

[RFC6303] Andrews, M., "Locally Served DNS Zones", BCP 163, RFC 6303, DOI 10.17487/RFC6303, July 2011, <https://www.rfc-editor.org/info/rfc6303>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <https://www.rfc-editor.org/info/rfc6365>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/info/rfc6762>.

[RFC7129] Gieben, R. and W. Mekking, "Authenticated Denial of Existence in the DNS", RFC 7129, DOI 10.17487/RFC7129, February 2014, <https://www.rfc-editor.org/info/rfc7129>.

[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>.

[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>.

[RFC7485] Zhou, L., Kong, N., Shen, S., Sheng, S., and A. Servin, "Inventory and Analysis of WHOIS Registration Objects", RFC 7485, DOI 10.17487/RFC7485, March 2015, <https://www.rfc-editor.org/info/rfc7485>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/info/rfc7858>.

[RFC8094] Reddy, T., Wing, D., and P. Patil, "DNS over Datagram Transport Layer Security (DTLS)", RFC 8094, DOI 10.17487/RFC8094, February 2017, <https://www.rfc-editor.org/info/rfc8094>.

[RFC8109] Koch, P., Larson, M., and P. Hoffman, "Initializing a DNS Resolver with Priming Queries", BCP 209, RFC 8109, DOI 10.17487/RFC8109, March 2017, <https://www.rfc-editor.org/info/rfc8109>.

[RSSAC026] Root Server System Advisory Committee (RSSAC), "RSSAC Lexicon", 2017, <https://www.icann.org/en/system/files/files/rssac-026-14mar17-en.pdf>.

Appendix A. Definitions Updated by this Document

The following definitions from RFCs are updated by this document:

- Forwarder in [RFC2308]
- Secure Entry Point (SEP) in [RFC3757]; note, however, that this RFC is already obsolete

Appendix B. Definitions First Defined in this Document

The following definitions are first defined in this document:

- "Alias" in Section 2
- "Apex" in Section 7
- "arpa" in Section 7
- "Class independent" in Section 5
- "Delegation-centric zone" in Section 7
- "Delegation" in Section 7
- "DNS operator" in Section 9

- "DNSSEC-aware" in Section 10
- "DNSSEC-unaware" in Section 10
- "Forwarding" in Section 6
- "Full resolver" in Section 6
- "Fully qualified domain name" in Section 2
- "Global DNS" in Section 2
- "Hardware Security Module (HSM)" in Section 10
- "Host name" in Section 2
- "IDN" in Section 2
- "In-bailiwick" in Section 7
- "Label" in Section 2
- "Locally served DNS zone" in Section 2
- "Naming system" in Section 2
- "Negative response" in Section 3
- "Open resolver" in Section 6
- "Out-of-bailiwick" in Section 7
- "Passive DNS" in Section 6
- "Policy-implementing resolver" in Section 6
- "Presentation format" in Section 5
- "Priming" in Section 6
- "Private DNS" in Section 2
- "Recursive resolver" in Section 6
- "Referrals" in Section 4
- "Registrant" in Section 9

- "Registrar" in Section 9
- "Registry" in Section 9
- "Root zone" in Section 7
- "Secure Entry Point (SEP)" in Section 10
- "Signing software" in Section 10
- "Stub resolver" in Section 6
- "TLD" in Section 2
- "Validating resolver" in Section 10
- "Validation" in Section 10
- "View" in Section 6
- "Zone transfer" in Section 6

Index

Zone signing key (ZSK) 31
Zone transfer 18

Acknowledgements

The following is the Acknowledgements for RFC 7719. Additional acknowledgements may be added as this draft is worked on.

The authors gratefully acknowledge all of the authors of DNS-related RFCs that proceed this one. Comments from Tony Finch, Stephane Bortzmeyer, Niall O'Reilly, Colm MacCarthaigh, Ray Bellis, John Kristoff, Robert Edmonds, Paul Wouters, Shumon Huque, Paul Ebersman, David Lawrence, Matthijs Mekking, Casey Deccio, Bob Harold, Ed Lewis, John Klensin, David Black, and many others in the DNSOP Working Group helped shape RFC 7719.

Additional people contributed to this document, including: Bob Harold, Dick Franks, Evan Hunt, John Dickinson, Mark Andrews, Martin Hoffmann, Paul Vixie, Peter Koch, Duane Wessels [[MORE NAMES WILL APPEAR HERE AS FOLKS CONTRIBUTE]].

Authors' Addresses

Paul Hoffman
ICANN

Email: paul.hoffman@icann.org

Andrew Sullivan
Oracle
100 Milverton Drive
Mississauga, ON L5R 4H1
Canada

Email: andrew.s.sullivan@oracle.com

Kazunori Fujiwara
Japan Registry Services Co., Ltd.
Chiyoda First Bldg. East 13F, 3-8-1 Nishi-Kanda
Chiyoda-ku, Tokyo 101-0065
Japan

Phone: +81 3 5215 8451
Email: fujiwara@jprs.co.jp

Domain Name System Operations
Internet-Draft
Intended status: Best Current Practice
Expires: September 14, 2017

J. Kristoff
DePaul University

March 13, 2017

DNS Transport over TCP - Operational Requirements
draft-kristoff-dnsop-dns-tcp-requirements-02

Abstract

This document encourages the practice of permitting DNS messages to be carried over TCP on the Internet. It also describes some of the consequences of this behavior and the potential operational issues that can arise when this best common practice is not upheld.

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 September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction

DNS messages may be delivered using UDP or TCP communications. While most DNS transactions are carried over UDP, some operators have been led to believe that any DNS over TCP traffic is unwanted or unnecessary for general DNS operation. As usage and features have evolved, TCP transport has become increasingly important for correct and safe operation of the Internet DNS. Reflecting modern usage, the DNS standards were recently updated to declare support for TCP is now a required part of the DNS implementation specifications in [RFC7766]. This document is the formal requirements equivalent for the operational community, encouraging operators to ensure DNS over TCP communications support is on par with DNS over UDP communications.

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. Background

2.1. Uneven Transport Usage and Preference

In the original suite of DNS specifications, [RFC1034] and [RFC1035] clearly specified that DNS messages could be carried in either UDP or TCP, but they also made clear a preference for UDP as the transport for queries in the general case. As stated in [RFC1035]:

> "While virtual circuits can be used for any DNS activity, datagrams are preferred for queries due to their lower overhead and better performance."

Another early, important, and influential document, [RFC1123], detailed the preference for UDP more explicitly:

> "DNS resolvers and recursive servers MUST support UDP, and SHOULD support TCP, for sending (non-zone-transfer) queries."

and further stipulated:

> A name server MAY limit the resources it devotes to TCP queries, but it SHOULD NOT refuse to service a TCP query just because it would have succeeded with UDP.

Culminating in [RFC1536], DNS over TCP came to be associated primarily with the zone transfer mechanism, while most DNS queries and responses were seen as the dominion of UDP.

2.2. Waiting for Large Messages and Reliability

As stipulated in the original specifications, DNS messages over UDP were restricted to a 512-byte message size. However, even while [RFC1123] made a clear preference for UDP, it foresaw DNS over TCP becoming more popular in the future:

> "[...] it is also clear that some new DNS record types defined in the future will contain information exceeding the 512 byte limit that applies to UDP, and hence will require TCP.

At least two new, widely anticipated developments were set to elevate the need for DNS over TCP transactions. The first was dynamic

updates defined in [RFC2136] and the second was the set of extensions collectively known as DNSSEC originally specified in [RFC2541]. The former suggested "requestors who require an accurate response code must use TCP", while the later warned "[...] larger keys increase the size of KEY and SIG RRs. This increases the chance of DNS UDP packet overflow and the possible necessity for using higher overhead TCP in responses."

Yet defying some expectations, DNS over TCP remained little used in real traffic across the Internet. Dynamic updates saw little deployment between autonomous networks. Around the time DNSSEC was first defined, another new feature affecting DNS over UDP helped solidify its dominance for message transactions.

2.3. EDNS0

In 1999 the IETF published the Extension Mechanisms for DNS (EDNS0) in [RFC2671]. This document standardized a way for communicating DNS nodes to perform rudimentary capabilities negotiation. One such capability written into the base specification and present in every ENDS0 compatible message is the value of the maximum UDP payload size the sender can support. This unsigned 16-bit field specifies in bytes the maximum DNS MTU. In practice, typical values are a subset of the 512 to 4096 byte range. EDNS0 was rapidly and widely deployed over the next several years and numerous surveys have shown many systems currently support larger UDP MTUs [CASTRO2010], [NETALYZR] with EDNS0.

The natural effect of EDNS0 deployment meant large DNS messages would be less reliant on TCP than they might otherwise have been. While a nonneglible population of DNS systems lack EDNS0 or may still fall back to TCP for some transactions, DNS over TCP transactions remain a very small fraction of overall DNS traffic [VERISIGN]. Nevertheless, some average increase in DNS message size, the continued development of new DNS features and a denial of service mitigation technique (see Section 9) have suggested that DNS over TCP transactions are as important to the correct and safe operation of the Internet DNS as ever, if not more so. Furthermore, there has been serious research that has suggested connection-oriented DNS transactions may provide security and privacy advantages over UDP transport [TDNS]. In fact, [RFC7858], a Standards Track document is just this sort of specification. Therefore, it might be desirable for network operators to avoid artificially inhibiting the potential utility and advances in the DNS such as these.

2.4. "Only Zone Tranfers Use TCP"

Even while many in the DNS community expect DNS over TCP transactions to occur without interference, in practice there has been a long held belief by some operators, particularly for security-related reasons, to the contrary [CHES94], [DJBDNS]. A popular meme has also held the imagination of some that DNS over TCP is only ever used for zone transfers and is generally unnecessary otherwise, with filtering all DNS over TCP traffic even described as a best practice. Arguably any exposed Internet service poses some risk, but this reasoning is often invalid.

3. DNS over TCP Requirements

Section 6.1.3.2 in [RFC1123] is updated: All general-purpose DNS servers MUST be able to service both UDP and TCP queries.

- Authoritative servers MUST service TCP queries so that they do not limit the size of responses to what fits in a single UDP packet.

- Recursive servers (or forwarders) MUST service TCP queries so that they do not prevent large responses from a TCP-capable server from reaching its TCP-capable clients.

Regarding the choice of limiting the resources a server devotes to queries, Section 6.1.3.2 in [RFC1123] also says:

> A name server MAY limit the resources it devotes to TCP queries, but it SHOULD NOT refuse to service a TCP query just because it would have succeeded with UDP.

This requirement is hereby updated: A name server MAY limit the the resources it devotes to queries, but it MUST NOT refuse to service a query just because it would have succeeded with another transport protocol.

DNS over TCP filtering is considered harmful in the general case. DNS resolver and server operators MUST provide DNS service over both UDP and TCP transports. Likewise, network operators MUST allow DNS service over both UDP and TCP transports. It must be acknowledged that DNS over TCP service can pose operational challenges that are not present when running DNS over UDP alone. However, it is the aim of this document to argue that the potential damage incurred by prohibiting DNS over TCP service is more detrimental to the continued utility and success of the DNS than when its usage is allowed.

4. Network and System Considerations

TODO: refer to IETF RFC 7766 connection handling discussion, various TCP hardening documents, network operator protocol and traffic best practices, etc.

5. DNS over TCP Filtering Risks

Networks that filter DNS over TCP may inadvertently cause problems for third party resolvers as experienced by [TOYAMA]. If for instance a resolver receives a truncated answer from a server, but if when the resolver resends the query using TCP and the TCP response never arrives, the resolver will incur the full extent of TCP retransmissions and time outs.

Networks that filter DNS over TCP risk losing access to significant or important pieces of the DNS name space. For a variety of reasons a DNS answer may require a DNS over TCP query. This may include large message sizes, lack of EDNS0 support, DDoS mitigation techniques, or perhaps some future capability that is as yet unforeseen will also demand TCP transport.

Even if any or all particular answers have consistently been returned successfully with UDP in the past, this continued behavior cannot be guaranteed when DNS messages are exchanged between autonomous systems. Therefore, filtering of DNS over TCP is considered harmful and contrary to the safe and successful operation of the Internet.

6. Standards Related to DNS Transport over TCP

This section enumerates all known IETF RFC documents that are currently of status standard, informational, best common practice or experimental and either implicitly or explicitly make assumptions or statements about the use of TCP as a transport for the DNS germane to this document.

6.1. TODO - additional, relevant RFCs

6.2. IETF RFC 7477 - Child-to-Parent Synchronization in DNS

This standards track document [RFC7477] specifies a RRType and protocol to signal and synchronize NS, A, and AAAA resource record changes from a child to parent zone. Since this protocol may require multiple requests and responses, it recommends utilizing DNS over TCP to ensure the conversation takes place between a consistent pair of end nodes.

6.3. IETF RFC 7766 - DNS Transport over TCP - Implementation Requirements

The standards track document [RFC7766] is might be considered the direct ancestor of this operational requirements document. The implementation requirements document codifies mandatory support for DNS over TCP in compliant DNS software.

6.4. IETF RFC 7828 - The edns-tcp-keepalive EDNS0 Option

This standards track document [RFC7828] defines an EDNS0 option to negotiate an idle timeout value for long-lived DNS over TCP connections. Consequently, this document is only applicable and relevant to DNS over TCP sessions and between implementations that support this option.

6.5. IETF RFC 7873 - Domain Name System (DNS) Cookies

This standards track document [RFC7873] describes an EDNS0 option to provide additional protection against query and answer forgery. This specification mentions DNS over TCP as a reasonable fallback mechanism when DNS Cookies are not available. The specification does make mention of DNS over TCP processing in two specific situations. In one, when a server receives only a client cookie in a request, the server should consider whether the request arrived over TCP and if so, it should consider accept TCP as sufficient to authenticate the request and respond accordingly. In another, when a client receives a BADCOOKIE reply using a fresh server cookie, the client should retry using TCP as the transport.

6.6. IETF RFC 7901 - CHAIN Query Requests in DNS

This experimental specification [RFC7901] describes an EDNS0 option that can be used by a security-aware validating resolver to request and obtain a complete DNSSEC validation path for any single query. This document requires the use of DNS over TCP or a source IP address verified transport mechanism such as EDNS-COOKIE.[RFC7873]

6.7. IETF RFC 8027 - DNSSEC Roadblock Avoidance

This document [RFC8027] details observed problems with DNSSEC deployment and mitigation techniques. Network traffic blocking and restrictions, including DNS over TCP messages, are highlighted as one reason for DNSSEC deployment issues. While this document suggests these sorts of problems are due to "non-compliant infrastructure" and is of type BCP, the scope of the document is limited to detection and mitigation techniques to avoid so-called DNSSEC roadblocks.

7. Acknowledgements

This document was initially motivated by feedback from students who pointed out that they were hearing contradictory information about filtering DNS over TCP messages. Thanks in particular to my teaching colleague, JPL, who perhaps unknowingly encouraged the initial research into the differences of what the IETF community has historically said and did. Thanks to all the NANOG 63 attendees who provided feedback to an early talk on this subject.

The following individuals provided an array of feedback to help improve this document: Bob Harold, Paul Hoffman, and Sara Dickinson. The author is indebted to their contributions. Any remaining errors or imperfections are the sole responsbility of the document author.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

Ironically, returning truncated DNS over UDP answers in order to induce a client query to switch to DNS over TCP has become a common response to source address spoofed, DNS denial-of-service attacks [RRL]. Historically, operators have been wary of TCP-based attacks, but in recent years, UDP-based flooding attacks have proven to be the most common protocol attack on the DNS. Nevertheless, a high rate of short-lived DNS transactions over TCP may pose challenges. While many operators have provided DNS over TCP service for many years without duress, past experience is no guarantee of future success.

DNS over TCP is not unlike many other Internet TCP services. TCP threats and many mitigation strategies have been well documented in a series of documents such as [RFC4953], [RFC4987], [RFC5927], and [RFC5961].

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

10.2. Informative References

[CASTRO2010]
Castro, S., Zhang, M., John, W., Wessels, D., and k. claffy, "Understanding and preparing for DNS evolution", 2010.

[CHES94] Cheswick, W. and S. Bellovin, "Firewalls and Internet Security: Repelling the Wily Hacker", 1994.

[DJBDNS] D.J. Bernstein, "When are TCP queries sent?", 2002, <https://cr.yp.to/djbdns/tcp.html#why>.

[NETALYZR]
Kreibich, C., Weaver, N., Nechaev, B., and V. Paxson, "Netalyzr: Illuminating The Edge Network", 2010.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <http://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <http://www.rfc-editor.org/info/rfc1035>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <http://www.rfc-editor.org/info/rfc1123>.

[RFC1536] Kumar, A., Postel, J., Neuman, C., Danzig, P., and S. Miller, "Common DNS Implementation Errors and Suggested Fixes", RFC 1536, DOI 10.17487/RFC1536, October 1993, <http://www.rfc-editor.org/info/rfc1536>.

[RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <http://www.rfc-editor.org/info/rfc2136>.

[RFC2541] Eastlake 3rd, D., "DNS Security Operational Considerations", RFC 2541, DOI 10.17487/RFC2541, March 1999, <http://www.rfc-editor.org/info/rfc2541>.

[RFC2671] Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC 2671, DOI 10.17487/RFC2671, August 1999, <http://www.rfc-editor.org/info/rfc2671>.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <http://www.rfc-editor.org/info/rfc4953>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <http://www.rfc-editor.org/info/rfc4987>.

[RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, DOI 10.17487/RFC5927, July 2010, <http://www.rfc-editor.org/info/rfc5927>.

[RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <http://www.rfc-editor.org/info/rfc5961>.

[RFC7477] Hardaker, W., "Child-to-Parent Synchronization in DNS", RFC 7477, DOI 10.17487/RFC7477, March 2015, <http://www.rfc-editor.org/info/rfc7477>.

[RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <http://www.rfc-editor.org/info/rfc7766>.

[RFC7828] Wouters, P., Abley, J., Dickinson, S., and R. Bellis, "The edns-tcp-keepalive EDNS0 Option", RFC 7828, DOI 10.17487/RFC7828, April 2016, <http://www.rfc-editor.org/info/rfc7828>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <http://www.rfc-editor.org/info/rfc7858>.

[RFC7873] Eastlake 3rd, D. and M. Andrews, "Domain Name System (DNS) Cookies", RFC 7873, DOI 10.17487/RFC7873, May 2016, <http://www.rfc-editor.org/info/rfc7873>.

[RFC7901] Wouters, P., "CHAIN Query Requests in DNS", RFC 7901, DOI 10.17487/RFC7901, June 2016, <http://www.rfc-editor.org/info/rfc7901>.

[RFC8027] Hardaker, W., Gudmundsson, O., and S. Krishnaswamy, "DNSSEC Roadblock Avoidance", BCP 207, RFC 8027, DOI 10.17487/RFC8027, November 2016, <http://www.rfc-editor.org/info/rfc8027>.

[RRL] Vixie, P. and V. Schryver, "DNS Response Rate Limiting (DNS RRL)", ISC-TN 2012-1 Draft1, April 2012.

[TDNS] Zhu, L., Heidemann, J., Wessels, D., Mankin, A., and N. Somaiya, "Connection-oriented DNS to Improve Privacy and Security", 2015.

[TOYAMA] Toyama, K., Ishibashi, K., Ishino, M., Yoshimura, C., and K. Fujiwara, "DNS Anomalies and Their Impacts on DNS Cache Servers", NANOG 32 Reston, VA USA, 2004.

[VERISIGN] Thomas, M. and D. Wessels, "An Analysis of TCP Traffic in Root Server DITL Data", DNS-OARC 2014 Fall Workshop Los Angeles, 2014.

Author’s Address

John Kristoff
DePaul University
Chicago, IL
US

Phone: +1 312 493 0305
Email: jtk@depaul.edu

DNSOP Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

R. Licht
Charter Communications
D. Lawrence, Ed.
Akamai Technologies

March 13, 2017

Client ID in Forwarded DNS Queries
draft-tale-dnsop-edns0-clientid-01

Abstract

This draft defines a DNS EDNS option to carry a client-specific identifier in DNS queries, with guidance for privacy protection of such information.

Ed note

Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication. This document is being collaborated on in GitHub at <https://github.com/vttale/edns0-clientid>. The most recent version of the document, open issues, etc should all be available here. The authors gratefully accept pull requests.

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 September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction

Some DNS operators generate, or wish to generate, customized DNS responses based on the originator of a DNS query. For example, [RFC7871], "Client Subnet in DNS Queries", defines a method to convey partial IP network address information about the device that originated a DNS request, so that a response could be targeted to be topographically near the source.

Some specialized services, however, need more precise client identity information to function adequately. For example, a parental control service that restricts access to particular domains from particular devices needs to have a device-specific identifier.

This document defines an EDNS [RFC6891] option to convey client identification information that is relevant to the DNS query. It is added by software on the client's local area network, for transmission to the upstream DNS provider.

A similar EDNS option is already being used on the public Internet in two different implementations. One is between the [dnsmasq] resolver on the client side and Nominum's [Vantio_CacheServe] upstream. It uses EDNS option code 65073 from the "Reserved for Local/Experimental Use" range to pass the client's Media Access Control (MAC) address. The other implementation is for Cisco's [Umbrella], aka OpenDNS, which encodes the client's MAC address and complete IP address. It uses option codes 26946 and 20292, respectively, from the middle of the "Unassigned" range.

This document codifies a more flexible format that can accommodate the needs of both implementations, as well as other more opaque identifiers. It is intended to supersede those non-standard options.

This option is intended only for constrained environments where its use can be carefully controlled. It is completely optional and should be ignored by most DNS software.

2. Privacy Considerations

The IETF is actively working on enhancing DNS privacy [DPRIVE_Working_Group], and the re-injection of personally identifiable information has been identified as a problematic design pattern [I-D.hardie-privsec-metadata-insertion].

Because this option transmits information that is meant to identify specific clients, to be considered compliant with this draft implementations MUST NOT add the option without explicit opt-in by an administrator on the local area network. For example, agreeing to the terms of service for a device-specific DNS filtering product would allow the option to be enabled, and only for communication to the product's DNS server(s).

Implementers need to be aware of the various laws and regulations governing handling personal data, but they are out of scope of this document.

No explicit provision is made in the protocol to opt-out. For more discussion on this, see Section 9, "Security Considerations".

3. Terminology

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 [RFC2119]

For a comprehensive treatment of DNS terms, please see [RFC7719]. This document uses the following additional terms:

ECID EDNS Client Identification.

Client The user or device that originates a DNS lookup.

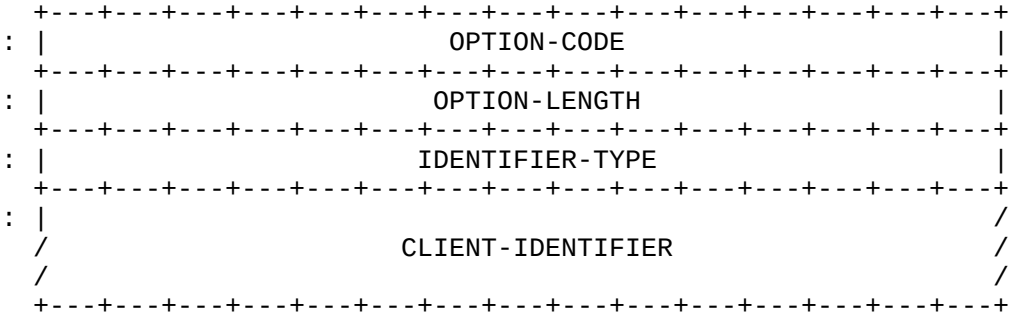
Nameserver A DNS server capable of resolving a DNS query and formulating a response.

Forwarding Resolver A nameserver that does not do iterative resolution itself, but instead passes that responsibility to another resolver, called a "Forwarder" in [RFC2308] section 1.

Tailored Response A response from a nameserver that is customized based on a policy defined for the client requesting the query.

4. Option Format

This protocol uses an EDNS [RFC6891] option to include client identification information in DNS messages. The option is structured as follows:



OPTION-CODE: 2 octets per [RFC6891]. For ECID the code is TBD by IANA.

OPTION-LENGTH: 2 octets per [RFC6891]. Contains the length of the payload following OPTION-LENGTH, in octets.

IDENTIFIER-TYPE: 2 octets per [Address_Family_Numbers], describing the format of CLIENT-IDENTIFIER as elaborated below. [Is it better to call this ADDRESS-FAMILY?]

CLIENT-IDENTIFIER: A variable number of octets, depending on IDENTIFIER-TYPE.

All fields are in network byte order ("big-endian", per [RFC1700], Data Notation).

This draft only specifies behaviour for the following IDENTIFIER-TYPE values and the corresponding CLIENT-IDENTIFIER lengths:

- 16389 (0x4005, 48-bit MAC): 6 octets, fixed.
- 1 (0x0001, IP version 4): 4 octets, fixed.
- 2 (0x0002, IP version 6): 16 octets, fixed.
- 16 (0x0010, Domain Name System): Variable-length domain name in uncompressed wire format followed by a variable-length custom token.

For DNS servers that implement ECID, it is RECOMMENDED that they recognize at least the 48-bit MAC CLIENT-IDENTIFIER.

The use of Domain Name System as an address family is to facilitate custom tokens that are not well-conceptualized as addresses, as described in Section 6.

Other types of identifying addresses, such as a 64-bit MAC [RFC7042] or a DHCP Unique Identifier [RFC3315] and [RFC6355] could be accommodated as devices and needs change, without needing to define new EDNS option codes to cover them. [Why not just bless those obvious candidates now?]

Multiple ECID options MAY appear in the OPT record. However, the same IDENTIFIER-TYPE SHOULD not appear more than once, and each ECID option MUST only carry one IDENTIFIER-TYPE and CLIENT-IDENTIFIER pair.

5. Protocol Description

5.1. DNS Query

Any client that originates a DNS query message MAY include the ECID option in the DNS Query message. It is normally expected that the client itself would not do this, but rather that it will be added by the local forwarding resolver.

When a DNS forwarding resolver, provided as part of a router for example, receives a DNS query message from the originating client it adds any IDENTIFIER-TYPE / CLIENT-IDENTIFIER pairs that it supports but which are not present in the existing client request. It then sends the request to the upstream full-service resolver.

Because the option contains personally identifiable information, it should be protected by either only being used within Autonomous

Systems [RFC1930] controlled by the same provider, by going over an opaque channel such as DNS over TLS [RFC7858], or by being securely encoded and varying per request. It MUST NOT be sent in clear-text across the Internet.

5.2. DNS Response

The logic used by a full-service resolver to customize a response based on ECID is out of scope for this document. The resolver MUST NOT include the ECID option in any queries that it makes to external authoritative DNS servers.

For possible caching purposes, the forwarding resolver needs to know whether filtering affected the response. If the name resolution involved any names for which customization was possible, even if such filtering resulted in delivering the original data, the response SHOULD include an ECID option which contains the FAMILY-ADDRESS and CLIENT-IDENTIFIER pairs that were considered for filtering.

For example, if a filter is set such that only names in the example.com domain are possibly restricted to some devices, then a request for foo.example.com would have the ECID in the response even when the request came from a device which was not restricted. Requests for any other names would not include ECID in the response.

So that the caching forwarding resolver does not need to have any knowledge about what filters are in place, it is the full-service resolver’s responsibility to adjust any TTLs in the response as might be dictated by the filter policy it has configured. That is, if some name is filtered only between the hours of 09:00 and 17:00 and a request is received for that name at 16:59:59, the TTL on a positive response or the SOA ncache field on a negative response should be set to just one second and the ECID option included as described above.

If the request contains a malformed ECID option, such as CLIENT-IDENTIFIER not correctly matching the length of described by OPTION-LENGTH and IDENTIFIER-TYPE, the resolver SHOULD reply with DNS rcode FORMERR.

If the resolver by policy does not respond to requests that are lacking ECID of the appropriate IDENTIFIER-TYPE, it SHOULD reply with DNS rcode REFUSED.

6. Using the DNS Address Family

When IDENTIFIER-TYPE 16 is used, the uncompressed wire format of the domain name is followed by a token that is otherwise opaque to this specification. The length of that token is defined by OPTION-LENGTH

less the two octets used for IDENTIFIER-TYPE and the length of the domain name.

The name used SHOULD be in a namespace that is controlled by the service provider that is using the option, but need not be resolvable in the DNS. We RECOMMEND that providers use short domain names to minimize DNS packet length.

The domain name provides protection against conflicts with other users of the option without the burden of creating yet another IANA Registry to manage yet another two-octet code. Co-operating forwarder/resolver pairs are the only users of the data who need to be concerned with its format.

7. Implementation Status

[RFC Editor: per RFC 6982 this section should be removed prior to publication.]

The protocol proposed here is not currently used anywhere exactly as described, though the Nominum and Umbrella implementations are substantially similar.

The authors know of at least two providers who wish to have it properly standardized and would implement the standard in preference to either of the existing non-standard methods.

8. NAT Considerations

Devices that perform Network Address Translation (NAT) SHOULD NOT give special consideration for ECID. NAT translates between a layer 3 private IP address assigned to a client device on the Local Area Network and a layer 3 public IP address for use within the Wide Area Network. If ECID is being used to pass an IPv4 or IPv6 address, it SHOULD use the internal address without NAT translation, because transforming it to the public address of the NAT device would coalesce all internal devices to just one address.

Other ECID options identify a client device by a different means, e.g. its layer 2 address. This sort of device’s identifier is not impacted by NAT. Therefore, DNS queries may be passed without modification of any ECID information.

9. Security Considerations

The identifier of the client that initiated the request will be visible to all servers that are passed the ECID option, and the

various devices on the path between the local network and the full-service resolver being used by the forwarding resolver.

DNS filtering products are easy circumvented and should not be considered real security measures. With commonly available tools it is trivial to discover the non-filtered DNS responses and use them in place of the filtered responses.

Along those lines, opting out of this specific protocol is as simple as using a different resolver, such as a full-service resolver on the device itself or one of the well-known public resolvers. Of course, other devices on the local network will still be able to see unencrypted DNS requests from the device, and the only way to really protect against such monitoring is to use an opaque tunnel to a trusted resolver.

10. IANA Considerations

IANA is requested to assign a new value in the DNS EDNS Option Codes registry for the Device ID option.

11. Acknowledgements

The authors wish to thank the Barry Greene, Martin Deen, Benjamin Petrin, and Robert Fleischman for their feedback and review during the initial development of this document.

12. References

12.1. Normative References

[Address_Family_Numbers]
IANA, ., "Address Family Numbers", n.d., <http://www.iana.org/assignments/address-family-numbers/>.

[RFC1700] Reynolds, J. and J. Postel, "Assigned Numbers", RFC 1700, DOI 10.17487/RFC1700, October 1994, <http://www.rfc-editor.org/info/rfc1700>.

[RFC1930] Hawkinson, J. and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)", BCP 6, RFC 1930, DOI 10.17487/RFC1930, March 1996, <http://www.rfc-editor.org/info/rfc1930>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <http://www.rfc-editor.org/info/rfc2308>.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <http://www.rfc-editor.org/info/rfc6891>.

[RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <http://www.rfc-editor.org/info/rfc7719>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <http://www.rfc-editor.org/info/rfc7858>.

[RFC7871] Contavalli, C., van der Gaast, W., Lawrence, D., and W. Kumari, "Client Subnet in DNS Queries", RFC 7871, DOI 10.17487/RFC7871, May 2016, <http://www.rfc-editor.org/info/rfc7871>.

12.2. Informative References

[dnsmasq] Kelley, S., "dnsmasq", n.d., <http://www.thekelleys.org.uk/dnsmasq/doc.html>.

[DPRIVE_Working_Group] Kumari, W. and T. Wicinski, "DPRIVE Working Group", n.d., <https://datatracker.ietf.org/wg/dprive/charter/>.

[I-D.hardie-privsec-metadata-insertion] Hardie, T., "Design considerations for Metadata Insertion", draft-hardie-privsec-metadata-insertion-07 (work in progress), March 2017.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <http://www.rfc-editor.org/info/rfc3315>.

[RFC6355] Narten, T. and J. Johnson, "Definition of the UUID-Based DHCPv6 Unique Identifier (DUID-UUID)", RFC 6355, DOI 10.17487/RFC6355, August 2011, <http://www.rfc-editor.org/info/rfc6355>.

[RFC7042] Eastlake 3rd, D. and J. Abley, "IANA Considerations and IETF Protocol and Documentation Usage for IEEE 802 Parameters", BCP 141, RFC 7042, DOI 10.17487/RFC7042, October 2013, <http://www.rfc-editor.org/info/rfc7042>.

[Umbrella]
Cisco Systems, Inc., "Umbrella", n.d., <https://docs.umbrella.com/developer/networkdevices-api/identifying-dns-traffic2>.

[Vantio_CacheServe]
Nominum, Inc., "Vantio CacheServe", n.d., <http://www.nominum.com/product/caching-dns/>.

Authors' Addresses

Robert Licht
Charter Communications
13820 Sunrise Valley Dr
Herndon VA 20171
USA

Email: robert.licht@charter.com

David C Lawrence (editor)
Akamai Technologies
150 Broadway
Cambridge MA 02142-1054
USA

Email: tale@akamai.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 6, 2018

J. Vcelak
CZ.NIC
S. Goldberg
Boston University
D. Papadopoulos
HKUST
S. Huque
Salesforce
D. Lawrence
Akamai Technologies
January 2, 2018

NSEC5, DNSSEC Authenticated Denial of Existence
draft-vcelak-nsec5-06

Abstract

The Domain Name System Security Extensions (DNSSEC) introduced two resource records (RR) for authenticated denial of existence: the NSEC RR and the NSEC3 RR. This document introduces NSEC5 as an alternative mechanism for DNSSEC authenticated denial of existence. NSEC5 uses verifiable random functions (VRFs) to prevent offline enumeration of zone contents. NSEC5 also protects the integrity of the zone contents even if an adversary compromises one of the authoritative servers for the zone. Integrity is preserved because NSEC5 does not require private zone-signing keys to be present on all authoritative servers for the zone, in contrast to DNSSEC online signing schemes like NSEC3 White Lies.

Ed note

Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication. This document is being collaborated on in GitHub at <https://github.com/fcelda/nsec5-draft>. The most recent version of the document, open issues, etc should all be available here. The authors gratefully accept pull requests.

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 July 6, 2018.

Table of Contents

1. Introduction

1.1. Rationale

NSEC5 provides an alternative mechanism for authenticated denial of existence for the DNS Security Extensions (DNSSEC). NSEC5 has two key security properties. First, NSEC5 protects the integrity of the

zone contents even if an adversary compromises one of the authoritative servers for the zone. Second, NSEC5 prevents offline zone enumeration, where an adversary makes a small number of online DNS queries and then processes them offline in order to learn all of the names in a zone. Zone enumeration can be used to identify routers, servers or other "things" that could then be targeted in more complex attacks. An enumerated zone can also be a source of probable email addresses for spam, or as a "key for multiple WHOIS queries to reveal registrant data that many registries may have legal obligations to protect" [RFC5155].

All other DNSSEC mechanisms for authenticated denial of existence either fail to preserve integrity against a compromised server, or fail to prevent offline zone enumeration.

When offline signing with NSEC is used [RFC4034], an NSEC chain of all existing domain names in the zone is constructed and signed offline. The chain is made of resource records (RRs), where each RR represents two consecutive domain names in canonical order present in the zone. The authoritative server proves the non-existence of a name by presenting a signed NSEC RR which covers the name. Because the authoritative server does not need not to know the private zone-signing key, the integrity of the zone is protected even if the server is compromised. However, the NSEC chain allows for easy zone enumeration: N queries to the server suffice to learn all N names in the zone (see e.g., [nmap-nsec-enum], [nsec3map], and [ldns-walk]).

When offline signing with NSEC3 is used [RFC5155], the original names in the NSEC chain are replaced by their cryptographic hashes. Offline signing ensures that NSEC3 preserves integrity even if an authoritative server is compromised. However, offline zone enumeration is still possible with NSEC3 (see e.g., [nsec3walker], [nsec3gpu]), and is part of standard network reconnaissance tools (e.g., [nmap-nsec3-enum], [nsec3map]).

When online signing is used, the authoritative server holds the private zone-signing key and uses this key to synthesize NSEC or NSEC3 responses on the fly (e.g. NSEC3 White Lies (NSEC3-WL) or Minimally-Covering NSEC, both described in [RFC7129]). Because the synthesized response only contains information about the queried name (but not about any other name in the zone), offline zone enumeration is not possible. However, because the authoritative server holds the private zone-signing key, integrity is lost if the authoritative server is compromised.

Scheme	Integrity vs network attacks?	Integrity vs compromised auth. server?	Prevents offline zone enumeration?	Online crypto?
Unsigned	NO	NO	YES	NO
NSEC	YES	YES	NO	NO
NSEC3	YES	YES	NO	NO
NSEC3-WL	YES	NO	YES	YES
NSEC5	YES	YES	YES	YES

NSEC5 prevents offline zone enumeration and also protects integrity even if a zone's authoritative server is compromised. To do this, NSEC5 replaces the unkeyed cryptographic hash function used in NSEC3 with a verifiable random function (VRF) [I-D.goldbe-vrf] [MRV99]. A VRF is the public-key version of a keyed cryptographic hash. Only the holder of the private VRF key can compute the hash, but anyone with public VRF key can verify the correctness of the hash.

The public VRF key is distributed in an NSEC5KEY RR, similar to a DNSKEY RR, and is used to verify NSEC5 hash values. The private VRF key is present on all authoritative servers for the zone, and is used to compute hash values. For every query that elicits a negative response, the authoritative server hashes the query on the fly using the private VRF key, and also returns the corresponding precomputed NSEC5 record(s). In contrast to the online signing approach [RFC7129], the private key that is present on all authoritative servers for NSEC5 cannot be used to modify the zone contents.

Like online signing approaches, NSEC5 requires the authoritative server to perform online public key cryptographic operations for every query eliciting a denying response. This is necessary; [nsec5] proved that online cryptography is required to prevent offline zone enumeration while still protecting the integrity of zone contents against network attacks.

NSEC5 is not intended to replace NSEC or NSEC3. It is an alternative mechanism for authenticated denial of existence. This document specifies NSEC5 based on the VRFs in [I-D.goldbe-vrf] over the FIPS 186-3 P-256 elliptic curve and over the the Ed25519 elliptic curve. A formal cryptographic proof of security for NSEC5 is in [nsec5ecc].

1.2. Requirements

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 [RFC2119].

1.3. Terminology

The reader is assumed to be familiar with the basic DNS and DNSSEC concepts described in [RFC1034], [RFC1035], [RFC4033], [RFC4034], and [RFC4035]; subsequent RFCs that update them in [RFC2136], [RFC2181], [RFC2308], [RFC5155], and [RFC7129]; and DNS terms in [RFC7719].

The reader should also be familiar with verifiable random functions (VRFs) as defined in [I-D.goldbe-vrf].

The following terminology is used through this document:

Base32hex: The "Base 32 Encoding with Extended Hex Alphabet" as specified in [RFC4648]. The padding characters ("=") are not used in the NSEC5 specification.

Base64: The "Base 64 Encoding" as specified in [RFC4648].

QNAME: The domain name being queried (query name).

Private NSEC5 key: The private key for the verifiable random function (VRF).

Public NSEC5 key: The public key for the VRF.

NSEC5 proof: A VRF proof. The holder of the private NSEC5 key (e.g., authoritative server) can compute the NSEC5 proof for an input domain name. Anyone who knows the public VRF key can verify that the NSEC5 proof corresponds to the input domain name.

NSEC5 hash: A cryptographic digest of an NSEC5 proof. If the NSEC5 proof is known, anyone can compute its corresponding NSEC5 hash.

NSEC5 algorithm: A triple of VRF algorithms that compute an NSEC5 proof (VRF_prove), verify an NSEC5 proof (VRF_verify), and process an NSEC5 proof to obtain its NSEC5 hash (VRF_proof2hash).

2. Backward Compatibility

The specification describes a protocol change that is not backward compatible with [RFC4035] and [RFC5155]. An NSEC5-unaware resolver will fail to validate responses introduced by this document.

To prevent NSEC5-unaware resolvers from attempting to validate the responses, new DNSSEC algorithms identifiers are introduced in Section 16 which alias existing algorithm numbers. The zones signed according to this specification MUST use only these algorithm

identifiers, thus NSEC5-unaware resolvers will treat the zone as insecure.

3. How NSEC5 Works

With NSEC5, the original domain name is hashed using a VRF [I-D.goldbe-vrf] using the following steps:

1. The domain name is processed using a VRF keyed with the private NSEC5 key to obtain the NSEC5 proof. Anyone who knows the public NSEC5 key, normally acquired via an NSEC5KEY RR, can verify that a given NSEC5 proof corresponds to a given domain name.

2. The NSEC5 proof is then processed using a publicly-computable VRF proof2hash function to obtain the NSEC5 hash. The NSEC5 hash can be computed by anyone who knows the input NSEC5 proof.

The NSEC5 hash determines the position of a domain name in an NSEC5 chain.

To sign a zone, the private NSEC5 key is used to compute the NSEC5 hashes for each name in the zone. These NSEC5 hashes are sorted in canonical order [RFC4034], and each consecutive pair forms an NSEC5 RR. Each NSEC5 RR is signed offline using the private zone-signing key. The resulting signed chain of NSEC5 RRs is provided to all authoritative servers for the zone, along with the private NSEC5 key.

To prove non-existence of a particular domain name in response to a query, the server uses the private NSEC5 key to compute the NSEC5 proof and NSEC5 hash corresponding to the queried name. The server then identifies the NSEC5 RR that covers the NSEC5 hash, and responds with this NSEC5 RR and its corresponding RRSIG signature RRset, as well as a synthesized NSEC5PROOF RR that contains the NSEC5 proof corresponding to the queried name.

To validate the response, the client verifies the following items:

- The client uses the public NSEC5 key, normally acquired from the NSEC5KEY RR, to verify that the NSEC5 proof in the NSEC5PROOF RR corresponds to the queried name.

- The client uses the VRF proof2hash function to compute the NSEC5 hash from the NSEC5 proof in the NSEC5PROOF RR. The client verifies that the NSEC5 hash is covered by the NSEC5 RR.

- The client verifies that the NSEC5 RR is validly signed by the RRSIG RRset.

4. NSEC5 Algorithms

The algorithms used for NSEC5 authenticated denial are independent of the algorithms used for DNSSEC signing. An NSEC5 algorithm defines how the NSEC5 proof and the NSEC5 hash are computed and validated.

The NSEC5 proof corresponding to a name is computed using ECVRF_prove(), as specified in [I-D.goldbe-vrf]. The input to ECVRF_prove() is a public NSEC5 key followed by a private NSEC5 key followed by an RR owner name in [RFC4034] canonical wire format. The output NSEC5 proof is an octet string.

An NSEC5 hash corresponding to a name is computed from its NSEC5 proof using ECVRF_proof2hash(), as specified in [I-D.goldbe-vrf]. The input to VRF_proof2hash() is an NSEC5 proof as an octet string. The output NSEC5 hash is either an octet string, or INVALID.

An NSEC5 proof for a name is verified using ECVRF_verify(), as specified in [I-D.goldbe-vrf]. The input is the NSEC5 public key, followed by an NSEC5 proof as an octet string, followed by an RR owner name in [RFC4034] canonical wire format. The output is either VALID or INVALID.

This document defines the EC-P256-SHA256 NSEC5 algorithm as follows:

- The VRF is the EC-VRF algorithm using the EC-VRF-P256-SHA256 ciphersuite specified in [I-D.goldbe-vrf].

- The public key format to be used in the NSEC5KEY RR is defined in Section 4 of [RFC6605] and thus is the same as the format used to store ECDSA public keys in DNSKEY RRs.
  [NOTE: This specification does not compress the elliptic curve point used for the public key, but we do compress curve points in every other place we use them. The NSEC5KEY record can be shrunk by 31 additional octets by encoding the public key with point compression.]

This document defines the EC-ED25519-SHA256 NSEC5 algorithm as follows:

- The VRF is the EC-VRF algorithm using the EC-VRF-ED25519-SHA256 ciphersuite specified in [I-D.goldbe-vrf].

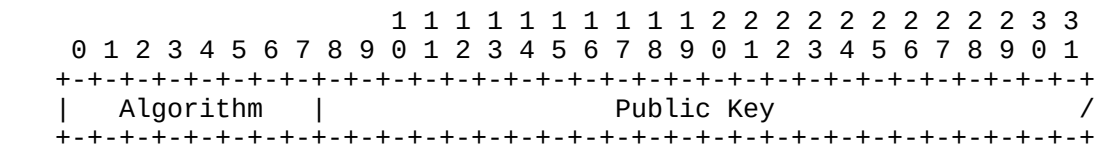
- The public key format to be used in the NSEC5KEY RR is defined in Section 3 of [RFC8080] and thus is the same as the format used to store Ed25519 public keys in DNSKEY RRs.

5. The NSEC5KEY Resource Record

The NSEC5KEY RR stores a public NSEC5 key. The key allows clients to validate an NSEC5 proof sent by a server.

5.1. NSEC5KEY RDATA Wire Format

The RDATA for the NSEC5KEY RR is as shown below:



Algorithm is a single octet identifying the NSEC5 algorithm.

Public Key is a variable-sized field holding public key material for NSEC5 proof verification.

5.2. NSEC5KEY RDATA Presentation Format

The presentation format of the NSEC5KEY RDATA is as follows:

The Algorithm field is represented as an unsigned decimal integer.

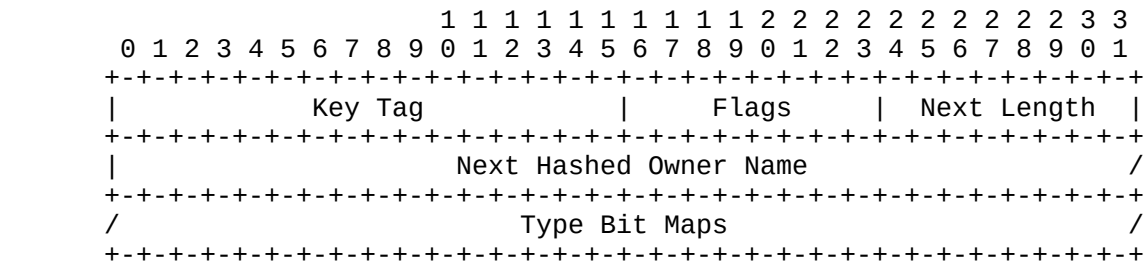
The Public Key field is represented in Base64 encoding. Whitespace is allowed within the Base64 text.

6. The NSEC5 Resource Record

The NSEC5 RR provides authenticated denial of existence for an RRset or domain name. One NSEC5 RR represents one piece of an NSEC5 chain, proving existence of the owner name and non-existence of other domain names in the part of the hashed domain space that is covered until the next owner name hashed in the RDATA.

6.1. NSEC5 RDATA Wire Format

The RDATA for the NSEC5 RR is as shown below:



The Key Tag field contains the key tag value of the NSEC5KEY RR that validates the NSEC5 RR, in network byte order. The value is computed from the NSEC5KEY RDATA using the same algorithm used to compute key tag values for DNSKEY RRs. This algorithm is defined in [RFC4034].

The Flags field is a single octet. The meaning of individual bits of the field is defined in Section 6.2.

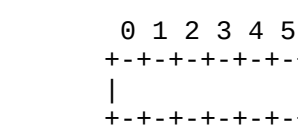
The Next Length field is an unsigned single octet specifying the length of the Next Hashed Owner Name field in octets.

The Next Hashed Owner Name field is a sequence of binary octets. It contains an NSEC5 hash of the next domain name in the NSEC5 chain.

Type Bit Maps is a variable-sized field encoding RR types present at the original owner name matching the NSEC5 RR. The format of the field is equivalent to the format used in the NSEC3 RR, described in [RFC5155].

6.2. NSEC5 Flags Field

The following one-bit NSEC5 flags are defined:



O - Opt-Out flag

W - Wildcard flag

All the other flags are reserved for future use and MUST be zero.

The Opt-Out flag has the same semantics as in NSEC3. The definition and considerations in [RFC5155] are valid, except that NSEC3 is replaced by NSEC5.

The Wildcard flag indicates that a wildcard synthesis is possible at the original domain name level (i.e., there is a wildcard node immediately descending from the immediate ancestor of the original domain name). The purpose of the Wildcard flag is to reduce the maximum number of RRs required for an authenticated denial of existence proof from (at most) three to (at most) two, as originally described in [I-D.gieben-nsec4] Section 7.2.1.

6.3. NSEC5 RDATA Presentation Format

The presentation format of the NSEC5 RDATA is as follows:

The Key Tag field is represented as an unsigned decimal integer.

The Flags field is represented as an unsigned decimal integer.

The Next Length field is not represented.

The Next Hashed Owner Name field is represented as a sequence of case-insensitive Base32hex digits without any whitespace and without padding.

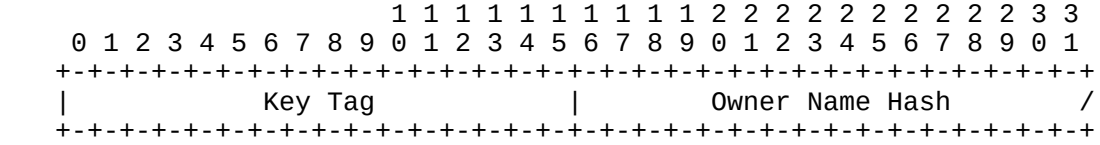
The Type Bit Maps representation is equivalent to the representation used in NSEC3 RR, described in [RFC5155].

7. The NSEC5PROOF Resource Record

The NSEC5PROOF record is not to be included in the zone file. The NSEC5PROOF record contains the NSEC5 proof, proving the position of the owner name in an NSEC5 chain.

7.1. NSEC5PROOF RDATA Wire Format

The RDATA for the NSEC5PROOF RR is shown below:



Key Tag field contains the key tag value of the NSEC5KEY RR that validates the NSEC5PROOF RR, in network byte order.

Owner Name Hash is a variable-sized sequence of binary octets encoding the NSEC5 proof of the owner name of the RR.

7.2. NSEC5PROOF RDATA Presentation Format

The presentation format of the NSEC5PROOF RDATA is as follows:

The Key Tag field is represented as an unsigned decimal integer.

The Owner Name Hash is represented in Base64 encoding. Whitespace is allowed within the Base64 text.

8. Types of Authenticated Denial of Existence with NSEC5

This section summarizes all possible types of authenticated denial of existence. For each type the following lists are included:

1. Facts to prove: the minimum amount of information that an authoritative server must provide to a client to assure the client that the response content is valid.

2. Authoritative server proofs: the names for which the NSEC5PROOF RRs are synthesized and added into the response along with the NSEC5 RRs matching or covering each such name. These records together prove the listed facts.

3. Validator checks: the individual checks that a validating server is required to perform on a response. The response content is considered valid only if all of the checks pass.

If NSEC5 is said to match a domain name, the owner name of the NSEC5 RR has to be equivalent to an NSEC5 hash of that domain name. If an NSEC5 RR is said to cover a domain name, the NSEC5 hash of the domain name must sort in canonical order between that NSEC5 RR’s Owner Name and Next Hashed Owner Name.

8.1. Name Error Responses

Facts to prove:

Non-existence of the domain name that explictly matches the QNAME.

Non-existence of the wildcard that matches the QNAME.

Authoritative server proofs:

NSEC5PROOF for closest encloser and matching NSEC5 RR.

NSEC5PROOF for next closer name and covering NSEC5 RR.

Validator checks:

Closest encloser is in the zone.

The NSEC5 RR matching the closest encloser has its Wildcard flag cleared.

The NSEC5 RR matching the closest encloser does not have NS without SOA in the Type Bit Map.

The NSEC5 RR matching the closest encloser does not have DNAME in the Type Bit Map.

Next closer name is not in the zone.

8.2. No Data Responses

The processing of a No Data response for DS QTYPE differs if the Opt-Out is in effect. For DS QTYPE queries, the validator has two possible checking paths. The correct path can be simply decided by inspecting if the NSEC5 RR in the response matches the QNAME.

Note that the Opt-Out is valid only for DS QTYPE queries.

8.2.1. No Data Response, Opt-Out Not In Effect

Facts to prove:

Existence of an RRset explicitly matching the QNAME.

Non-existence of QTYPE RRset matching the QNAME.

Non-existence of CNAME RRset matching the QNAME.

Authoritative server proofs:

NSEC5PROOF for the QNAME and matching NSEC5 RR.

Validator checks:

QNAME is in the zone.

NSEC5 RR matching the QNAME does not have QTYPE in Type Bit Map.

NSEC5 RR matching the QNAME does not have CNAME in Type Bit Map.

8.2.2. No Data Response, Opt-Out In Effect

Facts to prove:

The delegation is not covered by the NSEC5 chain.

Authoritative server proofs:

NSEC5PROOF for closest provable encloser and matching NSEC5 RR.

Validator checks:

Closest provable encloser is in zone.

Closest provable encloser covers (not matches) the QNAME.

NSEC5 RR matching the closest provable encloser has Opt-Out flag set.

8.3. Wildcard Responses

Facts to prove:

A signed positive response to the QNAME demonstrating the existence of the wildcard (label count in RRSIG is less than in QNAME), and also providing closest encloser name.

Non-existence of the domain name matching the QNAME.

Authoritative server proofs:

A signed positive response for the wildcard expansion of the QNAME.

NSEC5PROOF for next closer name and covering NSEC5 RR.

Validator checks:

Next closer name is not in the zone.

8.4. Wildcard No Data Responses

Facts to prove:

The existence of the wildcard at the closest encloser to the QNAME.

Non-existence of both the QTYPE and of the CNAME type that matches QNAME via wildcard expansion.

Authoritative server proofs:

NSEC5PROOF for source of synthesis (i.e., wildcard at closest encloser) and matching NSEC5 RR.

NSEC5PROOF for next closer name and covering NSEC5 RR.

Validator checks:

Closest encloser to the QNAME exists.

NSEC5 RR matching the wildcard label prepended to the closest encloser, and which does not have the bits corresponding to the QTYPE and CNAME types set it the type bitmap.

9. Authoritative Server Considerations

9.1. Zone Signing

Zones using NSEC5 MUST satisfy the same properties as described in Section 7.1 of [RFC5155], with NSEC3 replaced by NSEC5. In addition, the following conditions MUST be satisfied as well:

- If the original owner name has a wildcard label immediately descending from the original owner name, the corresponding NSEC5 RR MUST have the Wildcard flag set in the Flags field. Otherwise, the flag MUST be cleared.
- The zone apex MUST include an NSEC5KEY RRset containing a NSEC5 public key allowing verification of the current NSEC5 chain.

The following steps describe one possible method to properly add required NSEC5 related records into a zone. This is not the only such existing method.

1. Select an algorithm for NSEC5 and generate the public and private NSEC5 keys.
2. Add an NSEC5KEY RR into the zone apex containing the public NSEC5 key.
3. For each unique original domain name in the zone and each empty non-terminal, add an NSEC5 RR. If Opt-Out is used, owner names of unsigned delegations MAY be excluded.

A. The owner name of the NSEC5 RR is the NSEC5 hash of the original owner name encoded in Base32hex without padding, prepended as a single label to the zone name.

B. Set the Key Tag field to be the key tag corresponding to the public NSEC5 key.

C. Clear the Flags field. If Opt-Out is being used, set the Opt-Out flag. If there is a wildcard label directly descending from the original domain name, set the Wildcard flag. Note that the wildcard can be an empty non-terminal (i.e., the wildcard synthesis does not take effect and therefore the flag is not to be set).

D. Set the Next Length field to a value determined by the used NSEC5 algorithm. Leave the Next Hashed Owner Name field blank.

E. Set the Type Bit Maps field based on the RRsets present at the original owner name.

4. Sort the set of NSEC5 RRs into canonical order.

5. For each NSEC5 RR, set the Next Hashed Owner Name field by using the owner name of the next NSEC5 RR in the canonical order. If the updated NSEC5 is the last NSEC5 RR in the chain, the owner name of the first NSEC5 RR in the chain is used instead.

The NSEC5KEY and NSEC5 RRs MUST have the same class as the zone SOA RR. Also the NSEC5 RRs SHOULD have the same TTL value as the SOA minimum TTL field.

Notice that a use of Opt-Out is not indicated in the zone. This does not affect the ability of a server to prove insecure delegations. The Opt-Out MAY be part of the zone-signing tool configuration.

9.1.1. Precomputing Closest Provable Encloser Proofs

Per Section 8, the worst-case scenario when answering a negative query with NSEC5 requires the authoritative server to respond with two NSEC5PROOF RRs and two NSEC5 RRs. One pair of NSEC5PROOF and NSEC5 RRs corresponds to the closest provable encloser, and the other pair corresponds to the next closer name. The NSEC5PROOF corresponding to the next closer name MUST be computed on the fly by the authoritative server when responding to the query. However, the NSEC5PROOF corresponding to the closest provable encloser MAY be precomputed and stored as part of zone signing.

Precomputing NSEC5PROOF RRs can halve the number of online cryptographic computations required when responding to a negative query. NSEC5PROOF RRs MAY be precomputed as part of zone signing as follows: For each NSEC5 RR, compute an NSEC5PROOF RR corresponding to the original owner name of the NSEC5 RR. The content of the precomputed NSEC5PROOF record MUST be the same as if the record was computed on the fly when serving the zone. NSEC5PROOF records are not part of the zone and SHOULD be stored separately from the zone file.

9.2. Zone Serving

This specification modifies DNSSEC-enabled DNS responses generated by authoritative servers. In particular, it replaces use of NSEC or NSEC3 RRs in such responses with NSEC5 RRs and adds NSEC5PROOF RRs.

According to the type of a response, an authoritative server MUST include NSEC5 RRs in the response, as defined in Section 8. For each NSEC5 RR in the response, a corresponding RRSIG RRset and an NSEC5PROOF MUST be added as well. The NSEC5PROOF RR has its owner name set to the domain name required according to the description in Section 8. The class and TTL of the NSEC5PROOF RR MUST be the same as the class and TTL value of the corresponding NSEC5 RR. The RDATA payload of the NSEC5PROOF is set according to the description in Section 7.1.

Notice that the NSEC5PROOF owner name can be a wildcard (e.g., source of synthesis proof in wildcard No Data responses). The name also always matches the domain name required for the proof while the NSEC5 RR may only cover (not match) the name in the proof (e.g., closest encloser in Name Error responses).

If NSEC5 is used, an answering server MUST use exactly one NSEC5 chain for one signed zone.

NSEC5 MUST NOT be used in parallel with NSEC, NSEC3, or any other authenticated denial of existence mechanism that allows for enumeration of zone contents, as this would defeat a principal security goal of NSEC5.

Similarly to NSEC3, the owner names of NSEC5 RRs are not represented in the NSEC5 chain and therefore NSEC5 records deny their own existence. The desired behavior caused by this paradox is the same as described in Section 7.2.8 of [RFC5155].

9.3. NSEC5KEY Rollover Mechanism

Replacement of the NSEC5 key implies generating a new NSEC5 chain. The NSEC5KEY rollover mechanism is similar to "Pre-Publish Zone Signing Key Rollover" as specified in [RFC6781]. The NSEC5KEY rollover MUST be performed as a sequence of the following steps:

1. A new public NSEC5 key is added into the NSEC5KEY RRset in the zone apex.

2. The old NSEC5 chain is replaced by a new NSEC5 chain constructed using the new key. This replacement MUST happen as a single atomic operation; the server MUST NOT be responding with RRs from both the new and old chain at the same time.

3. The old public key is removed from the NSEC5KEY RRset in the zone apex.

The minimum delay between steps 1 and 2 MUST be the time it takes for the data to propagate to the authoritative servers, plus the TTL value of the old NSEC5KEY RRset.

The minimum delay between steps 2 and 3 MUST be the time it takes for the data to propagate to the authoritative servers, plus the maximum zone TTL value of any of the data in the previous version of the zone.

9.4. Secondary Servers

This document does not define mechanism to distribute private NSEC5 keys. See Section 15.2 for security considerations for private NSEC5 keys.

9.5. Zones Using Unknown NSEC5 Algorithms

Zones that are signed with an unknown NSEC5 algorithm or with an unavailable private NSEC5 key cannot be effectively served. Such zones SHOULD be rejected when loading and servers SHOULD respond with RCODE=2 (Server failure) when handling queries that would fall under such zones.

9.6. Dynamic Updates

A zone signed using NSEC5 MAY accept dynamic updates [RFC2136]. The changes to the zone MUST be performed in a way that ensures that the zone satisfies the properties specified in Section 9.1 at any time. The process described in [RFC5155] Section 7.5 describes how to

handle the issues surrounding the handling of empty non-terminals as well as Opt-Out.

It is RECOMMENDED that the server rejects all updates containing changes to the NSEC5 chain and its related RRSIG RRs, and performs itself any required alternations of the NSEC5 chain induced by the update. Alternatively, the server MUST verify that all the properties are satisfied prior to performing the update atomically.

10. Resolver Considerations

The same considerations as described in Section 9 of [RFC5155] for NSEC3 apply to NSEC5. In addition, as NSEC5 RRs can be validated only with appropriate NSEC5PROOF RRs, the NSEC5PROOF RRs MUST be all together cached and included in responses with NSEC5 RRs.

11. Validator Considerations

11.1. Validating Responses

The validator MUST ignore NSEC5 RRs with Flags field values other than the ones defined in Section 6.2.

The validator MAY treat responses as bogus if the response contains NSEC5 RRs that refer to a different NSEC5KEY.

According to a type of a response, the validator MUST verify all conditions defined in Section 8. Prior to making decision based on the content of NSEC5 RRs in a response, the NSEC5 RRs MUST be validated.

To validate a denial of existence, public NSEC5 keys for the zone are required in addition to DNSSEC public keys. Similarly to DNSKEY RRs, the NSEC5KEY RRs are present at the zone apex.

The NSEC5 RR is validated as follows:

1. Select a correct public NSEC5 key to validate the NSEC5 proof. The Key Tag value of the NSEC5PROOF RR must match with the key tag value computed from the NSEC5KEY RDATA.

2. Validate the NSEC5 proof present in the NSEC5PROOF Owner Name Hash field using the public NSEC5 key. If there are multiple NSEC5KEY RRs matching the key tag, at least one of the keys must validate the NSEC5 proof.

3. Compute the NSEC5 hash value from the NSEC5 proof and check if the response contains NSEC5 RR matching or covering the computed

NSEC5 hash. The TTL values of the NSEC5 and NSEC5PROOF RRs must be the same.

4. Validate the signature on the NSEC5 RR.

If the NSEC5 RR fails to validate, it MUST be ignored. If some of the conditions required for an NSEC5 proof are not satisfied, the response MUST be treated as bogus.

Notice that determining the closest encloser and next closer name in NSEC5 is easier than in NSEC3. NSEC5 and NSEC5PROOF RRs are always present in pairs in responses and the original owner name of the NSEC5 RR matches the owner name of the NSEC5PROOF RR.

11.2. Validating Referrals to Unsigned Subzones

The same considerations as defined in Section 8.9 of [RFC5155] for NSEC3 apply to NSEC5.

11.3. Responses With Unknown NSEC5 Algorithms

A validator MUST ignore NSEC5KEY RRs with unknown NSEC5 algorithms. The practical result of this is that zones signed with unknown algorithms will be considered bogus.

12. Special Considerations

12.1. Transition Mechanism

[TODO: The following information will be covered.]

- Transition to NSEC5 from NSEC/NSEC3
- Transition from NSEC5 to NSEC/NSEC3
- Transition to new NSEC5 algorithms

12.2. Private NSEC5 keys

This document does not define a format to store private NSEC5 keys. Use of a standardized and adopted format is RECOMMENDED.

The private NSEC5 key MAY be shared between multiple zones, however a separate key is RECOMMENDED for each zone.

12.3. Domain Name Length Restrictions

NSEC5 creates additional restrictions on domain name lengths. In particular, zones with names that, when converted into hashed owner names, exceed the 255 octet length limit imposed by [RFC1035] cannot use this specification.

The actual maximum length of a domain name depends on the length of the zone name and the NSEC5 algorithm used.

All NSEC5 algorithms defined in this document use 256-bit NSEC5 hash values. Such a value can be encoded in 52 characters in Base32hex without padding. When constructing the NSEC5 RR owner name, the encoded hash is prepended to the name of the zone as a single label which includes the length field of a single octet. The maximum length of the zone name in wire format using the 256-bit hash is therefore 202 octets (255 - 53).

13. Implementation Status

NSEC5 has been implemented for the Knot DNS authoritative server (version 1.6.4) and the Unbound recursive server (version 1.5.9). The implementations did not introduce additional library dependencies; all cryptographic primitives are already present in OpenSSL v1.0.2j, which is used by both implementations. The implementations support the full spectrum of negative responses, (i.e., NXDOMAIN, NODATA, Wildcard, Wildcard NODATA, and unsigned delegation) using the EC-P256-SHA256 algorithm. The code is deliberately modular, so that the EC-ED25519-SHA256 algorithm could be implemented by using the Ed25519 elliptic curve [RFC8080] as a drop-in replacement for the P256 elliptic curve. The authoritative server implements the optimization from Section 9.1.1 to precompute the NSEC5PROOF RRs matching each NSEC5 record.

14. Performance Considerations

The performance of NSEC5 has been evaluated in [nsec5ecc].

15. Security Considerations

15.1. Zone Enumeration Attacks

NSEC5 is robust to zone enumeration via offline dictionary attacks by any attacker that does not know the private NSEC5 key. Without the private NSEC5 key, that attacker cannot compute the NSEC5 proof that corresponds to a given domain name. The only way it can learn the NSEC5 proof value for a domain name is by querying the authoritative server for that name. Without the NSEC5 proof value, the attacker

cannot learn the NSEC5 hash value. Thus, even an attacker that collects the entire chain of NSEC5 RR for a zone cannot use offline attacks to "reverse" that NSEC5 hash values in these NSEC5 RR and thus learn which names are present in the zone. A formal cryptographic proof of this property is in [nsec5] and [nsec5ecc].

15.2. Compromise of the Private NSEC5 Key

NSEC5 requires authoritative servers to hold the private NSEC5 key, but not the private zone-signing keys or the private key-signing keys for the zone.

The private NSEC5 key cannot be used to modify zone contents, because zone contents are signed using the private zone-signing key. As such, a compromise of the private NSEC5 key does not compromise the integrity of the zone. An adversary that learns the private NSEC5 key can, however, perform offline zone-enumeration attacks. For this reason, the private NSEC5 key need only be as secure as the DNSSEC records whose privacy (against zone enumeration) is being protected by NSEC5. A formal cryptographic proof of this property is in [nsec5] and [nsec5ecc].

To preserve this property of NSEC5, the private NSEC5 key MUST be different from the private zone-signing keys or key-signing keys for the zone.

15.3. Key Length Considerations

The NSEC5 key must be long enough to withstand attacks for as long as the privacy of the zone contents is important. Even if the NSEC5 key is rolled frequently, its length cannot be too short, because zone privacy may be important for a period of time longer than the lifetime of the key. For example, an attacker might collect the entire chain of NSEC5 RR for the zone over one short period, and then, later (even after the NSEC5 key expires) perform an offline dictionary attack that attempts to reverse the NSEC5 hash values present in the NSEC5 RRs. This is in contrast to zone-signing and key-signing keys used in DNSSEC; these keys, which ensure the authenticity and integrity of the zone contents, need to remain secure only during their lifetime.

15.4. NSEC5 Hash Collisions

If the NSEC5 hash of a QNAME collides with the NSEC5 hash of the owner name of an NSEC5 RR, it will be impossible to prove the non-existence of the colliding QNAME. However, the NSEC5 VRFs ensure that obtaining such a collision is as difficult as obtaining a collision in the SHA-256 hash function, requiring approximately 2^128

effort. Note that DNSSEC already relies on the assumption that a cryptographic hash function is collision-resistant, since these hash functions are used for generating and validating signatures and DS RRs. See also the discussion on key lengths in [nsec5].

16. IANA Considerations

This document updates the IANA registry "Domain Name System (DNS) Parameters" in subregistry "Resource Record (RR) TYPEs", by defining the following new RR types:

NSEC5KEY value TBD.

NSEC5 value TBD.

NSEC5PROOF value TBD.

This document creates a new IANA registry for NSEC5 algorithms. This registry is named "DNSSEC NSEC5 Algorithms". The initial content of the registry is:

0 is Reserved.

1 is EC-P256-SHA256.

2 is EC-ED25519-SHA256.

3-255 is Available for assignment.

This document updates the IANA registry "DNS Security Algorithm Numbers" by defining following aliases:

TBD is NSEC5-ECDSAP256SHA256 alias for ECDSAP256SHA256 (13).

TBD is NSEC5-ED25519, alias for ED25519 (15).

17. Contributors

This document would not be possible without help of Moni Naor (Weizmann Institute), Sachin Vasant (Cisco Systems), Leonid Reyzin (Boston University), and Asaf Ziv (Weizmann Institute) who contributed to the design of NSEC5. Ondrej Sury (CZ.NIC Labs), and Duane Wessels (Verisign Labs) provided advice on the implementation of NSEC5, and assisted with evaluating its performance.

18. References

18.1. Normative References

[FIPS-186-3]
National Institute for Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-3, June 2009.

[I-D.goldbe-vrf]
Goldberg, S., Papadopoulos, D., and J. Vcelak, "Verifiable Random Functions (VRFs)", draft-goldbe-vrf-01 (work in progress), June 2017.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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>.

[RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <https://www.rfc-editor.org/info/rfc2136>.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <https://www.rfc-editor.org/info/rfc2181>.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <https://www.rfc-editor.org/info/rfc2308>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <https://www.rfc-editor.org/info/rfc4034>.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <https://www.rfc-editor.org/info/rfc4035>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5114] Lepinski, M. and S. Kent, "Additional Diffie-Hellman Groups for Use with IETF Standards", RFC 5114, DOI 10.17487/RFC5114, January 2008, <https://www.rfc-editor.org/info/rfc5114>.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, DOI 10.17487/RFC5155, March 2008, <https://www.rfc-editor.org/info/rfc5155>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC6605] Hoffman, P. and W. Wijngaards, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", RFC 6605, DOI 10.17487/RFC6605, April 2012, <https://www.rfc-editor.org/info/rfc6605>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8080] Sury, O. and R. Edmonds, "Edwards-Curve Digital Security Algorithm (EdDSA) for DNSSEC", RFC 8080, DOI 10.17487/RFC8080, February 2017, <https://www.rfc-editor.org/info/rfc8080>.

18.2. Informative References

[I-D.gieben-nsec4] Gieben, R. and M. Mekking, "DNS Security (DNSSEC) Authenticated Denial of Existence", draft-gieben-nsec4-01 (work in progress), July 2012.

[ldns-walk] NLNetLabs, "ldns", 2015, <http://git.nlnetlabs.nl/ldns/tree/examples/ldns-walk.c>.

[MRV99] Michali, S., Rabin, M., and S. Vadhan, "Verifiable Random Functions", in FOCS, 1999.

[nmap-nsec-enum]
Bond, J., "nmap: dns-nsec-enum", 2011, <https://nmap.org/nsedoc/scripts/dns-nsec-enum.html>.

[nmap-nsec3-enum]
Nikolic, A. and J. Bond, "nmap: dns-nsec3-enum", 2011, <https://nmap.org/nsedoc/scripts/dns-nsec3-enum.html>.

[nsec3gpu]
Wander, M., Schwittmann, L., Boelmann, C., and T. Weis, "GPU-Based NSEC3 Hash Breaking", in IEEE Symp. Network Computing and Applications (NCA), 2014.

[nsec3map]
anonion0, "nsec3map with John the Ripper plugin", 2015, <https://github.com/anonion0/nsec3map.>.

[nsec3walker]
Bernstein, D., "Nsec3 walker", 2011, <http://dnscurve.org/nsec3walker.html>.

[nsec5] Goldberg, S., Naor, M., Papadopoulos, D., Reyzin, L., Vasant, S., and A. Ziv, "NSEC5: Provably Preventing DNSSEC Zone Enumeration", in NDSS'15, July 2014, <https://eprint.iacr.org/2014/582.pdf>.

[nsec5ecc]
Papadopoulos, D., Wessels, D., Huque, S., Vcelak, J., Naor, M., Reyzin, L., and S. Goldberg, "Can NSEC5 be Practical for DNSSEC Deployments?", in ePrint Cryptology Archive 2017/099, February 2017, <https://eprint.iacr.org/2017/099.pdf>.

[RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <https://www.rfc-editor.org/info/rfc6781>.

[RFC7129] Gieben, R. and W. Mekking, "Authenticated Denial of Existence in the DNS", RFC 7129, DOI 10.17487/RFC7129, February 2014, <https://www.rfc-editor.org/info/rfc7129>.

[RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <https://www.rfc-editor.org/info/rfc7719>.

Appendix A. Examples

We use a small DNS zone to illustrate how negative responses are handled with NSEC5. For brevity, the class is not shown (defaults to IN) and the SOA record is shortened, resulting in the following zone file:

```
example.org.         SOA ( ... )
example.org.         NS  a.example.org

a.example.org.       A 192.0.2.1

c.example.org.       A 192.0.2.2
c.example.org.       TXT "c record"

d.example.org.       NS ns1.d.example.org

ns1.d.example.org.   A 192.0.2.4

g.example.org.       A 192.0.2.1
g.example.org.       TXT "g record"

*.a.example.org.     TXT "wildcard record"
```

Notice the delegation to an unsigned zone d.example.org served by ns1.d.example.org. (Note: if the d.example.org zone was signed, then the example.org zone have a DS record for d.example.org.)

Next we present example responses. All cryptographic values are shortened as indicated by "..." and ADDITIONAL sections have been removed.

A.1. Name Error Example

Consider a query for a type A record for a.b.c.example.org.

The server must prove the following facts:

- Existence of closest encloser c.example.org.
- Non-existence of wildcard at closest encloser *.c.example.org.
- Non-existence of next closer b.c.example.org.

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NXDOMAIN; id: 5937

;; QUESTION SECTION:
;; a.b.c.example.org.           IN      A

;; AUTHORITY SECTION:
example.org.         3600 IN SOA a.example.org. hostmaster.example.org. (
            2010111214 21600 3600 604800 86400 )

example.org.         3600 IN RRSIG  SOA 16 2 3600 (
            20170412024301 20170313024301 5137 example.org. rT231b1rH... )
```

This is an NSEC5PROOF RR for c.example.com. It's RDATA is the NSEC5 proof corresponding to c.example.com. (NSEC5 proofs are randomized values, because NSEC5 proof values are computed uses the EC-VRF from [I-D.goldbe-vrf].) Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
c.example.org.      86400 IN NSEC5PROOF 48566 Amgn22zUiZ9JVyaT...
```

This is a signed NSEC5 RR "matching" c.example.org, which proves the existence of closest encloser c.example.org. The NSEC5 RR has its owner name equal to the NSEC5 hash of c.example.org, which is O4K89V. (NSEC5 hash values are deterministic given the public NSEC5 key.) The NSEC5 RR also has its Wildcard flag cleared (see the "0" after the key ID 48566). This proves the non-existence of the wildcard at the closest encloser *.c.example.com. NSEC5 RRs are precomputed.

```
o4k89v.example.org. 86400 IN NSEC5   48566 0 0O49PI A TXT RRSIG
o4k89v.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. zDNTSMQNlz... )
```

This is an NSEC5PROOF RR for b.c.example.org. It's RDATA is the NSEC5 proof corresponding to b.c.example.com. This NSEC5PROOF RR must be computed on the fly.

```
b.c.example.org.    86400 IN NSEC5PROOF 48566 AuvvJqbUcEs8sCpY...
```

This is a signed NSEC5 RR "covering" b.c.example.org, which proves the non-existence of the next closer name b.c.example.org The NSEC5 hash of b.c.example.org, which is AO5OF, sorts in canonical order between the "covering" NSEC5 RR's Owner Name (which is 0O49PI) and Next Hashed Owner Name (which is BAPROH).

```
0o49pi.example.org. 86400 IN NSEC5      48566 0 BAPROH (
            NS SOA RRSIG DNSKEY NSEC5KEY )

0o49pi.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. 4HT1uj1YlMzO)
```

[TODO: Add discussion of CNAME and DNAME to the example?]

A.2. No Data Example

Consider a query for a type MX record for c.example.org.

The server must prove the following facts:

- Existence of c.example.org. for any type other than MX or CNAME

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 38781

;; QUESTION SECTION:
;; c.example.org.    IN MX

;; AUTHORITY SECTION:
example.org.    3600 IN SOA     a.example.org. hostmaster.example.org. (
            2010111214 21600 3600 604800 86400 )

example.org.    3600 IN RRSIG   SOA 16 2 3600 20170412024301 20170313024301 5137
 example.org. /rT231b1rH/p
```

This is an NSEC5PROOF RR for c.example.com. Its RDATA corresponds to the NSEC5 proof for c.example.com. which is a randomized value. Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
c.example.org. 86400 IN NSEC5PROOF 48566 Amgn22zUiZ9JVyaT
```

This is a signed NSEC5 RR "matching" c.example.org. with CNAME and MX Type Bits cleared and its TXT Type Bit set. This NSEC5 RR has its owner name equal to the NSEC5 hash of c.example.org. This proves the existence of c.example.org. for a type other than MX and CNAME. NSEC5 RR are precomputed.

```
o4k89v.example.org. 86400 IN NSEC5   48566 0 0O49PI A TXT RRSIG

o4k89v.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. zDNTSMQNlz/J)
```

A.3. Delegation to an Unsigned Zone in an Opt-Out span Example

Consider a query for a type A record for foo.d.example.org.

Here, d.example.org is a delegation to an unsigned zone, which lies within an Opt-Out span.

The server must prove the following facts:

- Non-existence of signature on next closer name d.example.org.
- Opt-out bit is set in NSEC5 record covering next closer name d.example.org.
- Existence of closest provable encloser example.org

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 45866

;; QUESTION SECTION:
;; foo.d.example.org.         IN A

;; AUTHORITY SECTION:
d.example.org.       3600  IN NS      ns1.d.example.org.
```

This is an NSEC5PROOF RR for d.example.org. Its RDATA is the NSEC5 proof corresponding to d.example.org. This NSEC5PROOF RR is computed on the fly.

```
d.example.org.      86400   IN      NSEC5PROOF      48566 A9FpmeH79q7g6VNW
```

This is a signed NSEC5 RR "covering" d.example.org with its Opt-out bit set (see the "1" after the key ID 48566). The NSEC5 hash of d.example.org (which is BLE8LR) sorts in canonical order between the "covering" NSEC5 RR's Owner Name (BAPROH) and Next Hashed Owner Name (JQBMG4). This proves that no signed RR exists for d.example.org, but that the zone might contain a unsigned RR for a name whose NSEC5 hash sorts in canonical order between BAPROH and JQBMG4.

```
baproh.example.org. 86400 IN NSEC5   48566 1 JQBMG4 A TXT RRSIG

baproh.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. fjTcoRKgdML1)
```

This is an NSEC5PROOF RR for example.com. It's RDATA is the NSEC5 proof corresponding to example.com. Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
example.org.        86400 IN NSEC5PROOF      48566 AjwsPCJZ8zH/D0Tr
```

This is a signed NSEC5 RR "matching" example.org which proves the existence of a signed RRs for example.org. This NSEC5 RR has its owner name equal to the NSEC5 hash of example.org which is 0O49PI. NSEC5 RR are precomputed.

```
0o49pi.example.org. 86400 IN NSEC5   48566 0 BAPROH (
            NS SOA RRSIG DNSKEY NSEC5KEY)

0o49pi.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412034216 20170313034216 5137 example.org. 4HT1uj1YlMzO)
```

A.4. Wildcard Example

Consider a query for a type TXT record for foo.a.example.org.

The server must prove the following facts:

- Existence of the TXT record for the wildcard *.a.example.org
- Non-existence of the next closer name foo.a.example.org.

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 53731

;; QUESTION SECTION:
;; foo.a.example.org.        IN TXT
```

This is a signed TXT record for the wildcard at a.example.org (number of labels is set to 3 in the RRSIG record).

```
;; ANSWER SECTION:
foo.a.example.org.      3600 IN TXT     "wildcard record"

foo.a.example.org.      3600 IN RRSIG   TXT 16 3 3600 (
            20170412024301 20170313024301 5137 example.org. aeaLgZ8sk+98)

;; AUTHORITY SECTION:
example.org.            3600 IN NS      a.example.org.

example.org.            3600 IN RRSIG   NS 16 2 3600 (
            20170412024301 20170313024301 5137 example.org. 8zuN0h2x5WyF)
```

This is an NSEC5PROOF RR for foo.a.example.org. This NSEC5PROOF RR must be computed on-the-fly.

```
  foo.a.example.org.     86400 IN NSEC5PROOF      48566 AjqF5FGGVso40Lda
```

This is a signed NSEC5 RR "covering" foo.a.example.org. The NSEC5 hash of foo.a.example.org is FORDMO and sorts in canonical order between the NSEC5 RR's Owner Name (which is BAPROH) and Next Hashed Owner Name (which is JQBMG4). This proves the non-existence of the next closer name foo.a.example.com. NSEC5 RRs are precomputed.

```
      baproh.example.org. 86400 IN NSEC5   48566 1 JQBMG4 A TXT RRSIG
      baproh.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
          20170412024301 20170313024301 5137 example.org. fjTcoRKgdML1
```

A.5. Wildcard No Data Example

Consider a query for a type MX record for foo.a.example.org.

The server must prove the following facts:

- Existence of wildcard at closest encloser *.a.example.org. for any type other than MX or CNAME.
- Non-existence of the next closer name foo.a.example.org.

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 17332

;; QUESTION SECTION:
;; foo.a.example.org.           IN      MX

;; AUTHORITY SECTION:
example.org.       3600 IN SOA     a.example.org. hostmaster.example.org. (
            2010111214 21600 3600 604800 86400 )

example.org.       3600 IN RRSIG   SOA 16 2 3600 (
            20170412024301 20170313024301 5137 example.org. /rT231b1rH/p )
```

This is an NSEC5PROOF RR for *.a.example.com, with RDATA equal to the NSEC5 proof for *.a.example.com. Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
*.a.example.org.  86400 IN NSEC5PROOF      48566 Aq38RWWPhbs/vtih
```

This is a signed NSEC5 RR "matching" *.a.example.org with its CNAME and MX Type Bits cleared and its TXT Type Bit set. This NSEC5 RR has its owner name equal to the NSEC5 hash of *.a.example.org. NSEC5 RRs are precomputed.

```
mpu6c4.example.org. 86400 IN NSEC5   48566 0 O4K89V TXT RRSIG

mpu6c4.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. m3I75ttcWwVC )
```

This is an NSEC5PROOF RR for foo.a.example.com. This NSEC5PROOF RR must be computed on-the-fly.

```
foo.a.example.org.  86400 IN NSEC5PROOF      48566 AjqF5FGGVso40Lda
```

This is a signed NSEC5 RR "covering" foo.a.example.org. The NSEC5 hash of foo.a.example.org is FORDMO, and sorts in canonical order between this covering NSEC5 RR's Owner Name (which is BAPROH) and Next Hashed Owner Name (which is JQBMG4). This proves the existence of the wildcard at closest encloser *.a.example.org. for any type other than MX or CNAME. NSEC5 RRs are precomputed.

```
baproh.example.org. 86400 IN NSEC5   48566 1 JQBMG4 A TXT RRSIG

baproh.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. fjTcoRKgdML1 )
```

Appendix B. Change Log

Note to RFC Editor: if this document does not obsolete an existing RFC, please remove this appendix before publication as an RFC.

pre 00 - initial version of the document submitted to mailing list only

00 - fix NSEC5KEY rollover mechanism, clarify NSEC5PROOF RDATA, clarify inputs and outputs for NSEC5 proof and NSEC5 hash computation.

01 - Add Performance Considerations section.

02 - Add elliptic curve based VRF. Add measurement of response sizes based on empirical data.

03 - Mention precomputed NSEC5PROOF Values in Performance Considerations section.

04 - Edit Rationale, How NSEC5 Works, and Security Consideration sections for clarity. Edit Zone Signing section, adding precomputation of NSEC5PROOFs. Remove RSA-based NSEC5 specification. Rewrite Performance Considerations and Implementation Status sections.

05 - Remove appendix specifying VRFs and add reference to [I-D.goldbe-vrf]. Add Appendix A.

06 - Editorial changes. Minor updates to Section 8.1.

Authors' Addresses

Jan Vcelak
CZ.NIC
Milesovska 1136/5
Praha 130 00
CZ

EMail: jan.vcelak@nic.cz

Sharon Goldberg
Boston University
111 Cummington St, MCS135
Boston, MA 02215
USA

EMail: goldbe@cs.bu.edu

Dimitrios Papadopoulos
HKUST
Clearwater Bay
Hong Kong

EMail: dipapado@ust.hk

Shumon Huque
Salesforce
2550 Wasser Terr
Herndon, VA 20171
USA

EMail: shuque@gmail.com

David C Lawrence
Akamai Technologies
150 Broadway
Boston, MA 02142-1054
USA

EMail: tale@akamai.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 18, 2018

W. Kumari
Google
E. Hunt
ISC
R. Arends
Nominet
W. Hardaker
USC/ISI
D. Lawrence
Akamai Technologies
July 17, 2017

Extended DNS Errors
draft-wkumari-dnsop-extended-error-02

Abstract

This document defines an extensible method to return additional information about the cause of DNS errors. The primary use case is to extend SERVFAIL to provide additional information about the cause of DNS and DNSSEC failures.

[Open question: The document currently defines a registry for errors. It has also been suggested that the option also carry human readable (text) messages, to allow the server admin to provide additional debugging information (e.g: "example.com pointed their NS at us. No idea why...", "We don't provide recursive DNS to 192.0.2.0. Please stop asking...", "Have you tried Acme Anvil and DNS? We do DNS right..." (!). Please let us know if you think text is needed, or if a 16bit FCFS registry is expressive enough.]

[Open question: This document discusses extended *errors*, but it has been suggested that this could be used to also annotate *non-error* messages. The authors do not think that this is a good idea, but could be persuaded otherwise.]

Copyright Notice

Table of Contents

1. Introduction and background

There are many reasons that a DNS query may fail, some of them transient, some permanent; some can be resolved by querying another server, some are likely best handled by stopping resolution. Unfortunately, the error signals that a DNS server can return are very limited, and are not very expressive. This means that applications and resolvers often have to "guess" at what the issue is - e.g the answer was marked REFUSED because of a lame delegation, or because there is a lame delegation or because the nameserver is still starting up and loading zones? Is a SERVFAIL a DNSSEC validation issue, or is the nameserver experiencing a bad hair day?

A good example of issues that would benefit by additional error information is an error caused by a DNSSEC validation issue. When a stub resolver queries a DNSSEC bogus name (using a validating resolver), the stub resolver receives only a SERVFAIL in response. Unfortunately, SERVFAIL is used to signal many sorts of DNS errors, and so the stub resolver simply asks the next configured DNS resolver. The result of trying the next resolver is one of two outcomes: either the next resolver also validates, a SERVFAIL is returned again, and the user gets an (largely) incomprehensible error message; or the next resolver is not a validating resolver, and the user is returned a potentially harmful result.

This document specifies a mechanism to extend (or annotate) DNS errors to provide additional information about the cause of the error. This information can be used by the resolver to make a decision regarding whether or not to retry, or by technical users attempting to debug issues.

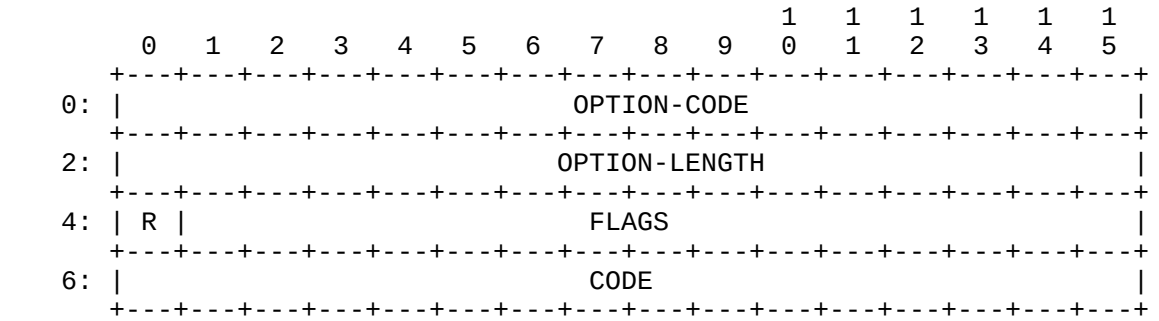
Here is a reference to an "external" (non-RFC / draft) thing: ([IANA.AS_Numbers]). And this is a link to an ID:[I-D.ietf-sidr-iana-objects].

1.1. Requirements notation

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 [RFC2119].

2. Extended Error EDNS0 option format

This draft uses an EDNS0 ([RFC2671]) option to include extended error (ExtError) information in DNS messages. The option is structured as follows:



- OPTION-CODE, 2 octets (defined in [RFC6891]), for ExtError is TBD.
- OPTION-LENGTH, 2 octets ((defined in [RFC6891]) contains the length of the payload (everything after OPTION-LENGTH) in octets and should be 4.
- FLAGS, 2 octets.
- CODE, 2 octets.

Currently the only defined flag is the R flag.

R - Retry The R (or Retry) flag provides a hint to the receiver if it should retry the query, possibly by querying another server. If the R bit is set (1), the sender believes that retrying the query may provide a successful answer next time; if the R bit is clear (0), the sender believes that it should not ask another server.

The remaining bits in the flags field MUST be set to 0 by the sender and SHOULD be ignored by the receiver.

Code: A code point into the IANA "Extended DNS Errors" registry.

3. Use of the Extended DNS Error option

The Extended DNS Error (EDE) is an EDNS option. It can be included in any error response (SERVFAIL, NXDOMAIN, REFUSED, etc) to a query that includes an EDNS option. This document includes a set of initial codepoints (and requests to the IANA to add them to the registry), but is extensible via the IANA registry to allow additional error codes to be defined in the future.

The R (Retry) flag provides a hint (or suggestion) as to what the receiver may want to do with this annotated error. The mechanism is specifically designed to be extensible, and so implementations may

receive EDE codes that it does not understand. The R flag allows implementations to make a decision as to what to do if it receives a response with an unknown code - retry or drop the query. Note that this flag is only a suggestion or hint. Receivers can choose to ignore this hint.

4. Defined Extended DNS Errors

This document defines some initial EDE codes. The mechanism is intended to be extensible, and additional codepoints will be registered in the "Extended DNS Errors" registry. This document provides suggestions for the R flag, but the originating server may ignore these recommendations if it knows better.

4.1. Extended DNS Error Code 1 - DNSSEC Bogus

The resolver attempted to perform DNSSEC validation, but validation ended in the Bogus state. The R flag should be set.

4.2. Extended DNS Error Code 2 - DNSSEC Indeterminite

The resolver attempted to perform DNSSEC validation, but validation ended in the Indeterminate state.

Usually attached to SERVFAIL messages. The R flag should be set.

4.3. Extended DNS Error Code 3 - Lame

An authoritative resolver that receives a query (with the RD bit clear) for a domain for which it is not authoritative SHOULD include this EDE code in the REFUSED response.

Implementations should not set the R flag in this case (another nameserver might not be lame).

4.4. Extended DNS Error Code 4 - Prohibited

An authoritative or recursive resolver that receives a query from an "unauthorized" client can annotate its REFUSED message with this code. Examples of "unauthorized" clients are recursive queries from IP addresses outside the network, blacklisted IP addresses, etc.

Implementations SHOULD allow operators to define what to set the R flag to in this case.

4.5. Extended DNS Error Code 5 - TooBusy

[Ed: This might be a bad idea. It is intended to allow servers under a DoS (for example a random subdomain attack) to signal to recursive clients that they are being abusive and should back off. This may be a bad idea -- it may "complete the attack", it may be spoofable (by anyone who could also do a MITM style attack), etc.]

A nameserver which is under excessive load (for example, because it is experiencing a DoS) may annotate any answer with this code.

It is RECOMMENDED that implementations set the R flag in this case, but may allow operators to define what to set the R flag to.

[agreed: bad idea -wjh]

5. IANA Considerations

[This section under construction, beware.]

This document defines a new EDNS(0) option, entitled "Extended DNS Error", assigned a value of TBD1 from the "DNS EDNS0 Option Codes (OPT)" registry [to be removed upon publication: [http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-11]

Value	Name	Status	Reference
TBD	Extended DNS Error	TBD	[This document]

Data Tag Name Length Meaning ---- ---- ------ ------- TBD1 FooBar N FooBar server

The IANA is requested to create and maintain the "Extended DNS Error codes" registry. The codepoint space is broken into 3 ranges:

- 1 - 16384: Specification required.
- 16385 - 65000: First Come First Served
- 65000 - 65534: Experimental / Private use

The codepoints 0, 65535 are reserved.

6. Open questions

1 Can this be included in *any* response or only responses to requests that included an EDNS option? Resolvers are supposed to ignore additional. EDNS capable ones are supposed to simply ignore unknown options. I know the spec says you can only include EDNS0 in a response if in a request -- it is time to reevaluate this?

2 Can this be applied to *any* response, or only error responses?

3 Should textual information be allowed as well? What if the only thing allowed is a domain name, e.g to point at where validation began failing?

7. Security Considerations

DNSSEC is being deployed - unfortunately a significant number of clients (~11% according to [GeoffValidation]), when receiving a SERVFAIL from a validating resolver because of a DNSSEC validaion issue simply ask the next (non-validating) resolver in their list, and do don't get any of the protections which DNSSEC should provide. This is very similar to a kid asking his mother if he can have another cookie. When the mother says "No, it will ruin your dinner!", going off and asking his (more permissive) father and getting a "Yes, sure, cookie!".

8. Acknowledgements

The authors wish to thank Geoff Huston. They also vaguely remember discussing this with a number of people over the years, but have forgotten who all they were -- if you were one of them, and are not listed, please let us know and we'll acknowledge you.

I also want to thank the band "Infected Mushroom" for providing a good background soundtrack (and to see if I can get away with this!) Another author would like to thank the band "Mushroom Infectors". This was funny at the time we wrote it, but I cannot remember why...

9. References

9.1. Normative References

[IANA.AS_Numbers]
IANA, "Autonomous System (AS) Numbers",
<http://www.iana.org/assignments/as-numbers>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

9.2. Informative References

[GeoffValidation]
IANA, "A quick review of DNSSEC Validation in today's Internet", June 2016, <http://www.potaroo.net/presentations/2016-06-27-dnssec.pdf>.

[I-D.ietf-sidr-iana-objects]
Manderson, T., Vegoda, L., and S. Kent, "RPKI Objects issued by IANA", draft-ietf-sidr-iana-objects-03 (work in progress), May 2011.

Appendix A. Changes / Author Notes.

[RFC Editor: Please remove this section before publication]

From -02 to -03:

- Added David Lawrence -- I somehow missed that in last version.

From -00 to -01;

- Fixed up some of the text, minor clarifications.

Authors' Addresses

Warren Kumari
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
US

Email: warren@kumari.net

Evan Hunt
ISC
950 Charter St
Redwood City, CA 94063
US

Email: each@isc.org

Roy Arends
Nominet
UK

Email: TBD

Wes Hardaker
USC/ISI
P.O. Box 382
Davis, VA 95617
US

David C Lawrence
Akamai Technologies
150 Broadway
Cambridge, MA 02142-1054
US

Email: tale@akamai.com

DNSOP Working Group
Internet-Draft
Updates: 2308, 4033, 4034, 4035 (if approved)
Intended status: Standards Track
Expires: May 3, 2018

J. Woodworth
D. Ballew
CenturyLink, Inc.
S. Bindinganaveli Raghavan
Hughes Network Systems
D. Lawrence
Akamai Technologies

October 30, 2017

BULK DNS Resource Records
draft-woodworth-bulk-rr-07

Abstract

The BULK DNS resource record type defines a method of pattern-based creation of DNS resource records based on numeric substrings of query names. The intent of BULK is to simplify generic assignments in a memory-efficient way that can be easily shared between the primary and secondary nameservers for a zone.

Ed note

Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication. This document is being collaborated on in GitHub at <https://github.com/vttale/bulk-rr>. The most recent version of the document, open issues, etc should all be available here. The authors gratefully accept pull requests.

Status of This Memo

Table of Contents

1. Introduction

The BULK DNS resource record defines a pattern-based method for on-the-fly resource record generation. It is essentially an enhanced wildcard mechanism, constraining generated resource record owner names to those that match a pattern of variable numeric substrings. It is also akin to the $GENERATE master file directive [bind-arm] without being limited to numeric values and without creating all possible records in the zone data.

For example, consider the following record:

```
example.com. 86400 IN BULK A (
                        pool-A-[0-255]-[0-255].example.com.
                        10.55.${1}.${2}
                    )
```

It will answer requests for pool-A-0-0.example.com through pool-A-255-255.example.com with the IPv4 addresses 10.55.0.0 through 10.55.255.255.

Much larger record sets can be defined while minimizing the associated requirements for server memory and zone transfer network bandwidth.

This record addresses a number of real-world operational problems that authoritative DNS service providers experience. For example, operators who host many large reverse lookup zones, even for only IPv4 space in in-addr.arpa, would benefit from the disk space, memory size, and zone transfer efficiencies that are gained by encapsulating a simple record-generating algorithm versus enumerating all of the individual records to cover the same space.

Production zones of tens of thousands of pattern-generated records currently exist, that could be reduced to just one BULK RR. These zones can look deceptively small on the primary nameserver and balloon to 100MB or more when expanded,

BULK also allows administrators to more easily deal with singletons, records in the pattern space that are an exception to the normal data generation rules. Whereas a mechanism like $GENERATE may need to be adjusted to account for these individual records, the processing rules for BULK have explicit records more naturally override the dynamically generated ones. This collision problem is not just a theoretical concern, but a real source of support calls for providers.

Pattern-generated records are also not only for the reverse DNS space. Forward zones also occasionally have entries that follow patterns that would be well-addressed by the BULK RR.

1.1. Background and Terminology

The reader is assumed to be familiar with the basic DNS and DNSSEC concepts described in [RFC1034], [RFC1035], [RFC4033], [RFC4034], and [RFC4035]; subsequent RFCs that update them in [RFC2181] and [RFC2308]; and DNS terms in [RFC7719].

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 [RFC2119] when, and only when, they appear in all capitals, as shown here.

2. The BULK Resource Record

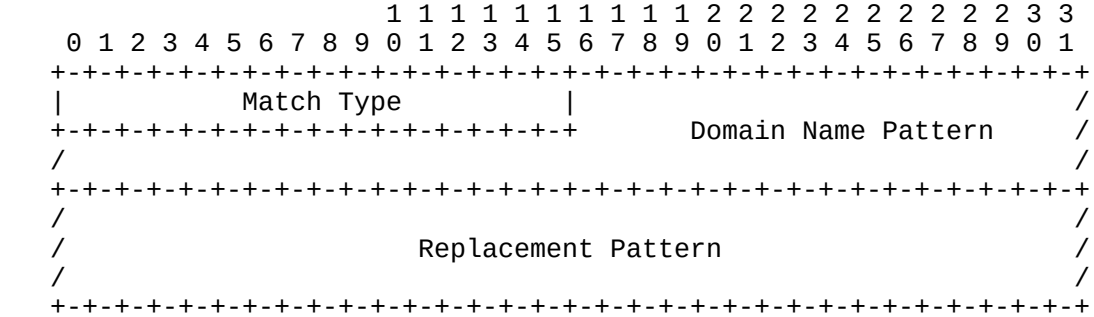
The BULK resource record enables an authoritative nameserver to generate RRs for other types based upon the query received.

The Type value for the BULK RR type is TBD.

The BULK RR is class-independent.

2.1. BULK RDATA Wire Format

The RDATA for a BULK RR is as follows:



Match Type identifies the type of the RRset to be generated by this BULK record. It is two octets corresponding to an RR TYPE code as specified in [RFC1035], Section 3.2.1.

Domain Name Pattern consists of a pattern encoded as a wire-format fully qualified domain name. The full name is used so that numeric

substrings above the zone cut can be captured in addition to those in the zone. It needs no length indicator for the entire field because the root label marks its end.

Special characters are interpreted as per the following Augmented Backus-Naur Form (ABNF) notation from [RFC5234].

```
match         =  1*(range / string)

range         =  "[" [decnum "-" decnum] "]" /
                  "<" [hexnum "-" hexnum] ">"
                       ; create references for substitution
                       ; limit of 32 references
                       ; [] is syntactic sugar for 0-255
                       ; <> is syntactic sugar for 00-ff

string        =  1*(ctext / quoted-char)

decnum        =  1*decdigit
                       ; constrained to 65535 maximum.

hexnum        =  1*hexdigit
                       ; constrained to ffff maximum.

octet         =  %x00-FF

decdigit      =  %x30-39
                       ; 0-9
hexdigit      =  decdigit / 0x41-0x46 / 0x61-66
                       ; 0-9, A-F, a-f

ctext         =  <any octet excepting "\">

quoted-char   = "\" octet
                        ; to allow special characters as literals
```

Interpretation of the Domain Name Pattern is described in detail in the "BULK Replacement" section. Note that quoted-char must be stored in the wire format to preserve its semantics when the BULK RR is interpreted by nameservers.

The limit of 32 references is meant to simplify implementation details. It is largely but not entirely arbitrary, as it could capture every individual character of the text representation of a full IPv6 address.

Replacement Pattern describes how the answer RRset MUST be generated for the matching query. It needs no length indicator because its end

can be derived from the RDATA length minus Match Type and Domain Name Pattern lengths. It uses the following additional ABNF elements:

```
replace      =   1*(reference / string)

reference    =   "$" "{" (positions / "*") [options] "}"

positions    =   (position / posrange) 0*("," (position / posrange))

posrange     =   position "-" position

position     =   1*decnum

options      =   delimiter [interval [padding]]

delimiter    =   "|" 0*(ctext | quoted-char)
                       ; "\|" to use "|" as delimiter
                       ; "\\" to use "\" as delimiter

interval     =   "|" *2decdigit

padding      =   "|" *2decdigit
```

[Is the formatting complexity beyond simple ${1}, ${2}, etc, really worth it? I definitely see how it could make for shorter replacement patterns, but does it enhance their clarity and usability, adding a feature someone really wants?]

The Replacement Pattern MUST end in the root label if it is intended to represent a fully qualified domain name.

2.2. The BULK RR Presentation Format

Match Type is represented as an RR type mnemonic or with [RFC3597]’s generic TYPE mechanism.

Domain Name Pattern is represented as a fully qualified domain name as per [RFC1035] Section 5.1 rules for encoding whitespace and other special characters.

Replacement Pattern is represented by the standard <character-string> text rules for master files as per [RFC1035] section 5.1.

It is suggested that lines longer than 80 characters be wrapped with parenthetical line continuation, per [RFC1035] Section 5.1, starting after Match Type and ending after Replacement Pattern.

3. BULK Replacement

When a BULK-aware authoritative nameserver receives a query for which it does not have a matching name or a covering wildcard, it MUST then look for BULK RRs at the zone apex, selecting all BULK RRs with a Match Type that matches the query type and a Domain Name Pattern that matches the query name. Note that query type ANY will select all Match Types, and all query types match a CNAME or DNAME Match Type. One or more answer RRs will be generated per the replacement rules below. Examples are provided in an appendix.

By only triggering the BULK algorithm when the query name does not exist, administrators are given the flexibility to explicitly override the behaviour of specific names that would otherwise match the BULK record's Domain Name Pattern. This is unlike BIND's $GENERATE directive, which adds the generated RRs to any existing names.

3.1. Matching the Domain Name Pattern

A query name matches the Domain Name Pattern if the characters that appear outside the numeric ranges match exactly and those within numeric ranges have values that fall within the range. Numeric matches MUST be of the appropriate decimal or hexadecimal type as specified by the delimiters in the pattern. For example, if a range is given as [0-255], then FF does not match even though its value as a hexadecimal number is within the range. Leading zeros in the numeric part(s) of the qname MUST be ignored; for example, 001.example.com, 01.example.com and 1.example.com would all match [].example.com.

When a query name matches a Domain Name Pattern, the value in each numeric range is stored for use by the Replacement Pattern, with reference numbers starting at 1 and counting from the left. For example, matching the query name host-24-156 against host-[0-255]-[0-255] assigns 24 to ${1} and 156 to ${2}.

3.2. Record Generation using Replacement Pattern

The Replacement Pattern generates the record data by replacing the ${...} references with data captured from the query name, and copying all other characters literally.

The simplest form of reference uses only the reference number between the braces, "{" and "}". The value of the reference is simply copied directly from the matching position of the query name.

The next form of reference notation uses the asterisk, "*". With ${*}, all captured values in order of ascending position, delimited by its default delimiter (described below), are placed in the answer.

Numeric range references, such as ${1-4}, replaces all values captured by those references, in order, delimited by the default delimiter described below. To reverse the order in which they are copied, reverse the upper and lower values, such as ${4-1}. This is useful for generating PTR records from query names in which the address is encoded in network order.

Similar to range references, separating positions by commas creates sets for replacement. For example, ${1,4} would be replaced by the first and fourth captured values, delimited its default delimiter. This notation may be combined with the numeric range form, such as ${3,2,1,8-4}.

3.2.1. Delimiters

A reference can specify a delimiter to use by following a vertical bar, "|", with zero or more characters. Zero characters, such as in ${1-3|}, means no delimiter is used, while other characters up to an unescaped vertical bar or closing brace are copied between position values in the replacement. The default delimiter is the hyphen, "-".

3.2.2. Delimiter intervals

A second vertical bar in the reference options introduces a delimiter interval. The default behavior of a multi-position reference is to combine each captured value specified with a delimiter between each. With a delimiter interval the delimiters are only added between every Nth value. For example, ${*|-|4} adds a hyphen between every group of four captured positions. This can be a handy feature in the IPv6 reverse namespace where every nibble is captured as a separate value and generated hostnames include sets of 4 nibbles. An empty or 0 value for the delimiter interval MUST be interpreted as the default value of 1.

3.2.3. Padding length

The fourth and final reference option determines the field width of the copied value. Shorter values MUST be padded with leading zeroes ("0") and longer values MUST be truncated to the width.

The default behavior, and that of an explicit empty padding length, is that the captured query name substring is copied exactly. A width of zero "0" is a signal to "unpad", and any leading zeros MUST be removed. [Unnecessary complexity?]

If a delimiter interval greater than 1 is used, captured values between the intervals will be concatenated and the padding or unpadding applied as a unit and not individually. An example of this would be ${*||4|4} which would combine each range of 4 captured values and pad or truncate them to a width of 4 characters.

[If this is kept, the element/feature should probably be renamed from "padding" since it is just as likely to truncate.]

3.2.4. Final processing

The string that results from all replacements is converted to the appropriate RDATA format for the record type. If the conversion fails, the SERVFAIL rcode MUST be set on the response, representing a misconfiguration that the server was unable to perform. [The EDNS extended-error code would be useful here.]

The TTL of each RR generated by a BULK RR is the TTL of the corresponding BULK record itself. [BULK should probably have its own TTL field because using that of the record itself feels like bad design. On the other hand, if BULK is never meant to be queried for directly and only appears in authoritative data, its own TTL is pretty useless normally.]

The class for the RRSet is the class of the BULK RR.

If the generated record type is one that uses domain names in its resource record data, such as CNAME, a relative domain names MUST be fully qualified with the origin domain of the BULK RR.

4. Known Limitations

This section defines known limitations of the BULK resource type.

4.1. Unsupported Nameservers

Authoritative nameservers that do not understand the semantics of the new record type will not be able to deliver the intended answers even when the type appears in their zone data This significantly affects the interoperability of primary versus secondary authorities that are not all running the same software. Adding new RRs which affect handling by authoritative servers, or being unable to add them, is an issue that needs to be explored more thoroughly within dnsop.

5. Security Considerations

Two known security considerations exist for the BULK resource record, DNSSEC and DDOS attack vectors.

5.1. DNSSEC Signature Strategies

DNSSEC was designed to provide validation for DNS resource records, requiring each tuple of owner, class, and type to have its own signature. This essentially defeats the purpose of providing large generated blocks of RRs in a single RR as each generated RR would require its own legitimate RRSIG record.

In the following sections several options are discussed to address this issue. Of the options, on-the-fly provides the most secure solution and NPN provides the most flexible.

5.1.1. On-the-fly Signatures

A significant design goal of DNSSEC was to be able to do offline cryptographic signing of zone contents, keeping the key material more secure.

On-the-fly processing requires authoritative nameservers to sign generated records as they are created. Not all authoritative nameserver implementations offer on-the-fly signatures, and even with those that do not all operators will want to keep signing keys online. This solution would either require all implementations to support on-the-fly signing or be ignored by implementations which can not or will not comply.

One possibly mitigation for addressing the risk of keeping the zone signing key online would be to continue to keep the key for signing positive answers offline and introduce a second key for online signing of negative answers.

No changes to validating resolvers is required to support this solution.

5.1.2. Alternative Signature Scheme

Previous versions of this draft proposed a new signature scheme using a Numeric Pattern Normalization (NPN) RR. It was a method to support offline signatures for BULK records, with the drawback that is required updates to DNSSEC-aware resolvers.

That mechanism is not specific to BULK and has been removed from the current draft. If there is further interest in pursuing it, it can be reopened as a separate draft.

5.1.3. Non-DNSSEC Zone Support Only

As a final option zones which wish to remain entirely without DNSSEC support may serve such zones without either of the above solutions and records generated based on BULK RRs will require zero support from recursive resolvers.

5.2. DDOS Attack Vectors and Mitigation

As an additional defense against Distributed Denial Of Service (DDOS) attacks against recursive (resolving) nameservers it is highly recommended shorter TTLs be used for BULK RRs than others. While disabling caching with a zero TTL is not recommended, as this would only result in a shift of the attack target, a balance will need to be found. While this document uses 24 hours (86400 seconds) in its examples, values between 300 to 900 seconds are likely more appropriate and is RECOMMENDED. What is ultimately deemed appropriate may differ from zone to zone and administrator to administrator.

[I am unclear how this helps DDOS mitigation against anyone at all, and suspect this section should be removed..]

5.3. Implications of Large-Scale DNS Records

The production of such large-scale records in the wild may have some unintended side-effects. These side-effects could be of concern or add unexpected complications to DNS based security offerings or forensic and anti-spam measures. While outside the scope of this document, implementers of technology relying on DNS resource records for critical decision making must take into consideration how the existence of such a volume of records might impact their technology.

Solutions to the magnitude problem for BULK generated RRs are expected be similar if not identical to that of existing wildcard records, the core difference being the resultant RDATA will be unique for each requested Domain Name within its scope.

The authors of this document are confident that by careful consideration, negative_side-effects produced by implementing the features described in this document can be eliminated from any such service or product.

6. Privacy Considerations

The BULK record does not introduce any new privacy concerns to DNS data.

7. IANA Considerations

IANA is requested to assign numbers for the BULK RR.

8. Acknowledgments

This document was created as an extension to the DNS infrastructure. As such, many people over the years have contributed to its creation and the authors are appreciative to each of them even if not thanked or identified individually.

A special thanks is extended for the kindness, wisdom and technical advice of Robert Whelton (CenturyLink, Inc.) and Gary O'Brien (Secure64 Software Corp).

9. References

9.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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>.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <https://www.rfc-editor.org/info/rfc2181>.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <https://www.rfc-editor.org/info/rfc2308>.

[RFC2317] Eidnes, H., de Groot, G., and P. Vixie, "Classless IN-ADDR.ARPA delegation", BCP 20, RFC 2317, DOI 10.17487/RFC2317, March 1998, <https://www.rfc-editor.org/info/rfc2317>.

[RFC3597] Gustafsson, A., "Handling of Unknown DNS Resource Record (RR) Types", RFC 3597, DOI 10.17487/RFC3597, September 2003, <https://www.rfc-editor.org/info/rfc3597>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <https://www.rfc-editor.org/info/rfc4034>.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <https://www.rfc-editor.org/info/rfc4035>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

9.2. Informative References

[bind-arm] Internet Systems Consortium, "BIND 9 Configuration Reference", 2016, <https://ftp.isc.org/isc/bind9/cur/9.9/doc/arm/Bv9ARM.html>.

[RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <https://www.rfc-editor.org/info/rfc7719>.

Appendix A. BULK Examples

A.1. Example 1

```
$ORIGIN 2.10.in-addr.arpa.
@ 86400 IN BULK PTR (
          [0-255].[0-255].[0-255].[0-255].in-addr.arpa.
          pool-${4-1}.example.com.
        )
```

A query received for the PTR of 4.3.2.10.in-addr.arpa will create the references ${1} through ${4} with the first four labels of the query name. The ${4-1} reference in the replacement pattern will then substitute them in reverse with the default delimiter of hyphen between every character and no special field width modifications. The TTL of the BULK RR is used for the generated record, making the response:

```
4.3.2.10.in-addr.arpa 86400 IN PTR pool-10-2-3-4.example.com.
```

A.2. Example 2

```
$ORIGIN 2.10.in-addr.arpa.
@ 86400 IN BULK PTR (
          [0-255].[0-255].[0-255].[0-255].in-addr.arpa.
          pool-${2,1|||3}.example.com.
        )
```

Example 2 is similar to Example 1, except that it modifies the replacement pattern. The empty option after the first vertical bar causes no delimiters to be inserted, while the second empty option that would keep the delimiter interval as 1. The latter is relevant because the final value, padding of 3, is applied over each delimiter interval even when no delimiter is used. Not all captures from the substring are required to be used in the response.

The result is that a query for the PTR of 4.3.2.10.in-addr.arpa generates this response:

```
4.3.2.10.in-addr.arpa 86400 IN PTR pool-003004.example.com.
```

[Admittedly you can’t do this very effectively without the field width complexity. Is this sort of name common? Does it need support? Admittedly $GENERATE had the feature, but is that reason enough?]

[Change this to a hex matching example?]

A.3. Example 3

This example contains a classless IPv4 delegation on the /22 CIDR boundary as defined by [RFC2317]. The network for this example is "10.2.0/22" delegated to a nameserver "ns1.sub.example.com.". RRs for this example are defined as:

```
$ORIGIN 2.10.in-addr.arpa.
@    7200 IN BULK CNAME [0-255].[0-3] ${*|.}.0-3
0-3 86400 IN NS ns1.sub.example.com.
```

A query for the PTR of 25.2.2.10.in-addr.arpa is received and the BULK record with the CNAME Match Type matches all query types. 25 and 2 are captured as references, and joined in the answer by the period (".") character as a delimiter, with ".0-3" then appended literally and fully qualified by the origin domain. The final synthesized record is:

```
25.2.2.10.in-addr.arpa 7200 IN CNAME 25.2.0-3.2.10.in-addr.arpa.
```

[Without $* and options complexity, the pattern to get the same result is just ${1}.{$2}.0-3 which is not really significantly onerous to enter, and slightly less arcane looking to comprehend.]

Authors' Addresses

John Woodworth
CenturyLink, Inc.
4250 N Fairfax Dr
Arlington VA 22203
USA

Email: John.Woodworth@CenturyLink.com

Dean Ballew
CenturyLink, Inc.
2355 Dulles Corner Blvd, Ste 200 300
Herndon VA 20171
USA

Email: Dean.Ballew@CenturyLink.com

Shashwath Bindinganaveli Raghavan
Hughes Network Systems
11717 Exploration Lane
Germantown MD 20876
USA

Email: shashwath.bindinganaveliraghavan@hughes.com

David C Lawrence
Akamai Technologies
150 Broadway
Cambridge MA 02142-1054
USA

Email: tale@akamai.com

dnsop
Internet-Draft
Obsoletes: 6944 (if approved)
Intended status: Standards Track
Expires: April 14, 2017

P. Wouters
Red Hat
O. Sury
CZ.NIC

October 11, 2016

Algorithm Implementation Requirements and Usage Guidance for DNSSEC
draft-wouters-sury-dnsop-algorithm-update-02

Abstract

The DNSSEC protocol makes use of various cryptographic algorithms in order to provide authentication of DNS data and proof of non-existence. To ensure interoperability between DNS resolvers and DNS authoritative servers, it is necessary to specify a set of algorithm implementation requirements and usage guidance to ensure that there is at least one algorithm that all implementations support. This document defines the current algorithm implementation requirements and usage guidance for DNSSEC. This document obsoletes RFC-6944.

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 April 14, 2017.

Copyright Notice

Table of Contents

1. Introduction

The DNSSEC signing algorithms are defined by various RFCs, including [RFC4034], [RFC5155], [RFC5702], [RFC5933], [RFC6605], [I-D.ietf-curdle-dnskey-eddsa]. DNSSEC is used to provide authentication of data. To ensure interoperability, a set of "mandatory-to-implement" DNSKEY algorithms are defined. This document obsoletes [RFC6944].

1.1. Updating Algorithm Implementation Requirements and Usage Guidance

The field of cryptography evolves continuously. New stronger algorithms appear and existing algorithms are found to be less secure then originally thought. Therefore, algorithm implementation requirements and usage guidance need to be updated from time to time to reflect the new reality. The choices for algorithms must be conservative to minimize the risk of algorithm compromise.

1.2. Updating Algorithm Requirement Levels

The mandatory-to-implement algorithm of tomorrow should already be available in most implementations of DNSSEC by the time it is made mandatory. This document attempts to identify and introduce those

algorithms for future mandatory-to-implement status. There is no guarantee that the algorithms in use today may become mandatory in the future. Published algorithms are continuously subjected to cryptographic attack and may become too weak or could become completely broken before this document is updated.

This document only provides recommendations for the mandatory-to-implement algorithms or algorithms too weak that are recommended not to be implemented. As a result, any algorithm listed at the [DNSKEY-IANA] and [DS-IANA] registries not mentioned in this document MAY be implemented. For clarification and consistency, an algorithm will be set to MAY only when it has been downgraded.

Although this document updates the algorithms to keep the DNSSEC authentication secure over time, it also aims at providing recommendations so that DNSSEC implementations remain interoperable. DNSSEC interoperability is addressed by an incremental introduction or deprecation of algorithms.

It is expected that deprecation of an algorithm is performed gradually. This provides time for various implementations to update their implemented algorithms while remaining interoperable. Unless there are strong security reasons, an algorithm is expected to be downgraded from MUST to MUST- or SHOULD, instead of MUST NOT. Similarly, an algorithm that has not been mentioned as mandatory-to-implement is expected to be introduced with a SHOULD instead of a MUST.

Since the effects of using an unknown DNSKEY algorithm is for the zone to be treated as insecure, it is recommended that algorithms downgraded to SHOULD- or below are no longer used by authoritative nameservers and DNSSEC signers to create new DNSKEY’s. This will allow for algorithms to slowly become more unused over time. Once deployment has reached a sufficiently low point these algorithms can finally be marked as MUST NOT so that recursive nameservers can remove support for these algorithms.

Recursive nameservers are encouraged to keep support for all algorithms not marked as MUST NOT.

1.3. Document Audience

The recommendations of this document mostly target DNSSEC implementers as implementations need to meet both high security expectations as well as high interoperability between various vendors and with different versions. Interoperability requires a smooth move to more secure algorithms. This may differ from a user point of view that may deploy and configure DNSSEC with only the safest algorithm.

On the other hand, comments and recommendations from this document are also expected to be useful for such users.

2. Conventions Used in This Document

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 [RFC2119].

We define some additional terms here:

SHOULD+ This term means the same as SHOULD. However, it is likely that an algorithm marked as SHOULD+ will be promoted at some future time to be a MUST.

SHOULD- This term means the same as SHOULD. However, an algorithm marked as SHOULD- may be deprecated to a MAY in a future version of this document.

MUST- This term means the same as MUST. However, it is expected at some point in the near future that this algorithm will no longer be a MUST in a future document. Although its status will be determined at a later time, it is reasonable to expect that if a future revision of a document alters the status of a MUST- algorithm, it will remain at least a SHOULD or a SHOULD-.

3. Algorithm Selection

3.1. DNSKEY Algorithms

Recommendations for DNSKEY algorithms [DNSKEY-IANA]

Number	Mnemonics	DNSSEC Signing	DNSSEC Validation
1	RSAMD5	MUST NOT	MUST NOT
3	DSA	MUST NOT	MUST NOT
5	RSASHA1	MUST-	MUST-
6	DSA-NSEC3-SHA1	MUST NOT	MUST NOT
7	RSASHA1-NSEC3-SHA1	MUST-	MUST-
8	RSASHA256	MUST	MUST
10	RSASHA512	SHOULD-	MUST
12	ECC-GOST	SHOULD NOT	SHOULD-
13	ECDSAP256SHA256	SHOULD-	MUST-
14	ECDSAP384SHA384	SHOULD NOT	SHOULD-
TBD	ED25519	SHOULD+	SHOULD+
TBD	ED448	SHOULD+	SHOULD+

RSAMD5 is not widely deployed and there is an industry-wide trend to deprecate MD5 usage.

RSASHA1 and RSASHA1-NSEC3-SHA1 are widely deployed, although zones deploying it are recommended to switch to RSASHA256 as there is an industry-wide trend to deprecate SHA1 usage. RSASHA1 does not support NSEC3. RSASHA1-NSEC3-SHA1 can be used with or without NSEC3.

DSA and DSA-NSEC3-SHA1 are not widely deployed and vulnerable to private key compromise when generating signatures using a weak or compromised random number generator.

RSASHA512 is at the SHOULD level for DNSSEC Signing because it has not seen wide deployment, but there are some deployments hence DNSSEC Validation MUST implement RSASHA512 to ensure interoperability.

ECC-GOST is at the SHOULD NOT level because it has not seen wide deployment and the algorithm has not seen wide scrutiny in the crypto community.

ECDSAP256SHA256 and ECDSAP384SHA384 provide more strength for signature size than RSASHA256 and RSASHA512 variants. ECDSAP256SHA256 has seen increased deployment and has been raised to MUST- level for resolving and SHOULD- for signing. It is seen as a temporary improvement over RSA until the [I-D.ietf-curdle-dnskey-eddsa] algorithms are published, implemented and deployed. ECDSAP384SHA384 offers little over ECDSAP256SHA256 and has not seen wide deployment, so the use is discouraged, especially for signing.

ED25519 and ED448 uses Edwards-curve Digital Security Algorithm (EdDSA). There are three main advantages of the EdDSA algorithm: It does not require the use of a unique random number for each signature, there are no padding or truncation issues as with ECDSA, and it is more resilient to side-channel attacks. Hence it is expected that these algorithms will be raised to SHOULD for signing and MUST for resolving once it has seen more implementations and deployement.

3.2. DS and CDS Algorithms

Recommendations for Delegation Signer Digest Algorithms [DNSKEY-IANA] These also apply to the CDS RRTYPE as specified in [RFC7344]

Number	Mnemonics	DNSSEC Delegation	DNSSEC Validation
0	NULL (CDS only)	MUST NOT [*]	MUST NOT [*]
1	SHA-1	SHOULD NOT	MUST-
2	SHA-256	MUST	MUST
3	GOST R 34.11-94	MAY	SHOULD
4	SHA-384	MAY	SHOULD+

[*] - This is a special type of CDS record signaling removal of DS at the parent in [I-D.ietf-dnsop-maintain-ds]

NULL is a special case, see [I-D.ietf-dnsop-maintain-ds]

SHA-1 is in wide use for DS records, but its use is discouraged as it is an aging algorithm. Users of SHA-1 SHOULD upgrade to SHA-256.

SHA-256 is in wide use and considered strong.

GOST R 34.11-94 is not in wide use. It is still recommended to be supported in validators so that adoption can increase.

SHA-384 is not in wide use. It is still recommended to be supported in validators so that adoption can increase.

4. Security Considerations

The security of cryptographic-based systems depends on both the strength of the cryptographic algorithms chosen and the strength of the keys used with those algorithms. The security also depends on the engineering of the protocol used by the system to ensure that there are no non-cryptographic ways to bypass the security of the overall system.

This document concerns itself with the selection of cryptographic algorithms for the use of DNSSEC, specifically with the selection of "mandatory-to-implement" algorithms. The algorithms identified in this document as "MUST implement" or "SHOULD implement" are not known to be broken at the current time, and cryptographic research so far leads us to believe that they will likely remain secure into the foreseeable future. However, this isn't necessarily forever and it is expected that new revisions of this document will be issued from time to time to reflect the current best practice in this area.

Retiring an algorithm too soon would result in a signed zone with such an algorithm to be downgraded to the equivalent of an unsigned

zone. Therefore, algorithm deprecation must be done very slowly and only after careful consideration and measurements of its use.

5. Operational Considerations

DNSKEY algorithm rollover in a live zone is a complex process. See [RFC6781] and [RFC7583] for guidelines on how to perform algorithm rollovers.

6. IANA Considerations

This document makes no requests of IANA.

7. Acknowledgements

This document borrows text from RFC 4307 by Jeffrey I. Schiller of the Massachusetts Institute of Technology (MIT) and the 4307bis document by Yoav Nir, Tero Kivinen, Paul Wouters and Daniel Migault. Much of the original text has been copied verbatim.

We wish to thank Olafur Gudmundsson and Paul Hoffman for their imminent feedback.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

8.2. Informative References

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <http://www.rfc-editor.org/info/rfc4034>.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, DOI 10.17487/RFC5155, March 2008, <http://www.rfc-editor.org/info/rfc5155>.

[RFC5702] Jansen, J., "Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC", RFC 5702, DOI 10.17487/RFC5702, October 2009, <http://www.rfc-editor.org/info/rfc5702>.

[RFC5933] Dolmatov, V., Ed., Chuprina, A., and I. Ustinov, "Use of GOST Signature Algorithms in DNSKEY and RRSIG Resource Records for DNSSEC", RFC 5933, DOI 10.17487/RFC5933, July 2010, <http://www.rfc-editor.org/info/rfc5933>.

[RFC6605] Hoffman, P. and W. Wijngaards, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", RFC 6605, DOI 10.17487/RFC6605, April 2012, <http://www.rfc-editor.org/info/rfc6605>.

[RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <http://www.rfc-editor.org/info/rfc6781>.

[RFC6944] Rose, S., "Applicability Statement: DNS Security (DNSSEC) DNSKEY Algorithm Implementation Status", RFC 6944, DOI 10.17487/RFC6944, April 2013, <http://www.rfc-editor.org/info/rfc6944>.

[RFC7344] Kumari, W., Gudmundsson, O., and G. Barwood, "Automating DNSSEC Delegation Trust Maintenance", RFC 7344, DOI 10.17487/RFC7344, September 2014, <http://www.rfc-editor.org/info/rfc7344>.

[RFC7583] Morris, S., Ihren, J., Dickinson, J., and W. Mekking, "DNSSEC Key Rollover Timing Considerations", RFC 7583, DOI 10.17487/RFC7583, October 2015, <http://www.rfc-editor.org/info/rfc7583>.

[I-D.ietf-curdle-dnskey-eddsa] Sury, O. and R. Edmonds, "EdDSA for DNSSEC", draft-ietf-curdle-dnskey-eddsa-01 (work in progress), October 2016.

[I-D.ietf-dnsop-maintain-ds] Gudmundsson, O. and P. Wouters, "Managing DS records from parent via CDS/CDNSKEY", draft-ietf-dnsop-maintain-ds-03 (work in progress), June 2016.

[DNSKEY-IANA] "DNSKEY Algorithms", <http://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml>.

[DS-IANA] "Delegation Signer Digest Algorithms", <http://www.iana.org/assignments/ds-rr-types/ds-rr-types.xhtml>.

Authors' Addresses

Paul Wouters
Red Hat

EMail: pwouters@redhat.com

Ondrej Sury
CZ.NIC

EMail: ondrej.sury@nic.cz