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
Intended status: Standards Track
Expires: July 13, 2017

R. Bellis
ISC

January 09, 2017

EDNS X-Proxied-For
draft-bellis-dnsop-xpf-01

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 an option within the EDNS(0) Extension Mechanism for DNS that allows a DNS server to receive the original client source IP address 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 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 July 13, 2017.

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 address (e.g. ACLs, DNS Response Rate Limiting, etc).

This document defines an option within the EDNS(0) Extension Mechanism for DNS [RFC6891] that allows a DNS server to receive the original client source IP address when supplied by trusted proxies.

Whilst in some circumstances it would be possible to re-use the Client Subnet EDNS Option [RFC7871] to carry this information, a new option is defined to allow both this option and the Client Subnet option to co-exist in the same packet.

2. Terminology

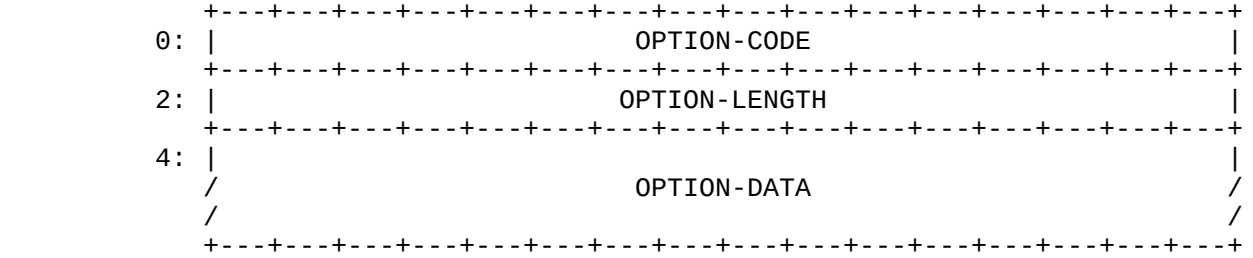
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

The word "proxy" 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. This is to distinguish these from a "forwarder" since that term is usually understood to describe a network component that sits on the oubound query path of a client.

3. Description

3.1. EDNS Option Format

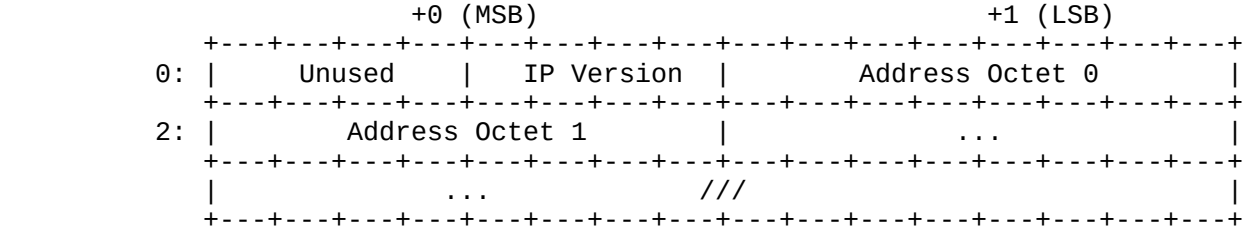
The overall format of an EDNS option is shown for reference below, per [RFC6891], followed by the option specific data:



OPTION-CODE: TBD, with mnemonic "XPF".

OPTION-LENGTH: Size (in octets) of OPTION-DATA.

OPTION-DATA: Option specific, as below:



Unused: Currently reserved. These 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).

Address: The source IP address of the client.

3.2. Proxy Processing

Proxies MUST append this option to each request packet received before sending it to the intended DNS server.

If this option is already present in an incoming request it MUST be stripped from the request unless the request was received from an upstream proxy that is itself white-listed by the receiving proxy (i.e. if the proxies are configured in a multi-tier architecture), in which case the original value of the option MUST be preserved.

If the proxy has to create a new OPT RR (because none was present in the original request) it MUST strip any OPT RR subsequently seen in the response for conformance with Section 7 of [RFC6891].

3.3. Server Processing

When this option is received from a white-listed client the DNS server MUST (SHOULD?) use the address from the option contained therein in preference to the client's source IP address for any data processing logic that would otherwise depend on the latter.

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

If the IP version is not understood by the server it MUST return a REFUSED response.

If the length of the client IP address contained in the OPTION-DATA is not consistent with that expected for the given IP version then the server MUST return a FORMERR response.

Servers MUST NOT send this option in DNS responses.

3.4. Secret Key Transaction Authentication for DNS (TSIG)

The considerations for TSIG [RFC2845] from Section 4.5 of "DNS Proxy Implementation Guidelines" [RFC5625] apply here.

A TSIG-signed request MUST either:

1. be forwarded according to RFC 5625 without addition of this option, or

2. be verified using a secret shared between client and proxy, updated with this option, and then re-signed with a (potentially different) shared secret before sending to the server.

In the case of option 1, the server might still be able to uniquely identify and authenticate the client through its shared key, but not by its IP address.

If option 2 is used, there is an operational trade-off to be considered as to whether the two secrets (between client and proxy, and between proxy and server) are actually the same secret. A potential advantage of three-way sharing of the secret is that if the server response requires no modifications it may be returned directly to the client without any further TSIG operations.

Author's note: A third alternative exists, which is to append an additional TSIG signature to the packet based on a secret shared only between the proxy and server. If end-to-end TSIG validation is required alongside TSIG validation between proxy and server, the server would have to 1) validate that second signature, 2) strip it, and then 3) perform further validation on the original signature. Feedback is sought on whether this is worth pursuing.

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. Privacy Considerations

Used incorrectly, this option 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.

6. IANA Considerations

IANA are directed to assign the value TBD for the XPF option in the DNS EDNS0 Option Codes Registry.

7. Acknowledgements

8. References

8.1. Normative References

[IANA-IP] IANA, "IANA IP Version Registry", November 2016, <http://www.iana.org/assignments/version-numbers/>.

[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>.

[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>.

[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, <http://www.rfc-editor.org/info/rfc2845>.

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

[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>.

8.2. Informative References

[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>.

Author’s Address

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

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

dnsop
Internet-Draft
Intended status: Best Current Practice
Expires: September 30, 2017

D. Crocker
Brandenburg InternetWorking

March 29, 2017

DNS Scoped Data Through Global '_Underscore' Naming of Attribute Leaves
draft-ietf-dnsop-attrleaf-02

Abstract

Formally, any DNS "RR" may occur for any domain name. However some services have defined an operational convention that applies to DNS leaf nodes that have a reserved node name, beginning with an underscore. The underscore construct is used to define a semantic scope for DNS records that are associated with the parent domain. This specification explores the nature of this DNS usage and defines the "DNS Global Underscore Scoped Entry Registry" registry with IANA.

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 30, 2017.

Copyright Notice

Table of Contents

1. Introduction

The core DNS technical specifications assign no semantics to domain names or their parts, and no constraints upon which resource records (RRs) are permitted to be associated with particular names. 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. Data semantics have been limited to the specification of particular resource records, on the expectation that new ones would be added as needed.

As an alternative to defining new RRs, some DNS service enhancements reuse an existing resource record, but have specified a restricted scope for its occurrence. That scope is a leaf node, within which the uses of specific resource records can be formally defined and constrained. The leaf has a distinguished naming convention: It uses a reserved DNS node name that begins 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 [RFC1035]. Effectively, this convention for leaf node naming creates a space for attributes that are associated with the parent domain, one level up.

One example is the "SRV" record [RFC2782] which generalizes concepts long-used for email routing by the "MX" record [RFC0974][RFC5321]. An equivalent usage to "SRV" is the "URI" "RR" [RFC7553]. Relying on

special DNS names has significant benefits and detriments. Some of these are explored in [RFC5507].

[Comment]: The terms "resolution context" and "scoping rules" have been suggested, in place of "semantic scope". In order to avoid concern for matters of semantics, this specification uses the term "scoping rules", to create a focus on the mechanics being defined, rather than nuances of interpretation for the mechanism.

The scoping feature is particularly useful when generalized resource records are used -- notably "TXT", "SRV" and "URI". It provides efficient separation of one use of them from others. Absent this separation, an undifferentiated mass of these "RR"s 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 the records do not adequately self-identify. With underscore-based scoping, only the relevant "RR"s are returned.

This specification discusses the underscore "attribute" enhancement, provides an explicit definition of it, and establishes an IANA registry for the highest-level reserved names that begin with _underscore; underscore-based names that are farther down the hierarchy is handled within the scope of the highest-level _underscore name. It updates the many existing specifications that have defined underscore names, in order to aggregate the references to a single IANA table.

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

2. Scaling Benefits and TXT and SRV Resource Records

Some resource records are generic and support a variety of uses. Each additional use defines its own rules and, possibly, its own internal syntax and node-naming conventions to distinguish among particular types. The "TXT" and "SRV" records are notable examples. Used freely, some of these approaches scale poorly, particularly when the same "RR" can be present in the same leaf node, but with different uses. An increasingly-popular approach, with excellent scaling properties, uses an underscore-based name, at a defined place in the DNS tree, so as to constrain to particular uses for particular "RR"s farther down the branch using that name. This means that a direct lookup produces only the desired records, at no greater cost than a typical DNS lookup.

In the case of "TXT" records, different uses have developed largely without coordination. One side-effect is that there is no

consistently distinguishable internal syntax for the record; even the inefficiencies of internal inspection might not provide a reliable means of distinguishing among the different uses. Underscore-based names therefore define an administrative way of separating "TXT" records that might have different uses, but otherwise would have no syntactic markers for distinguishing among them.

In the case of the "SRV" "RR" and "URI" "RR", distinguishing among different types of use was part of the design [RFC2782], [RFC7553]. The "SRV" and "URI" specifications serve as templates, defining "RR"s that might only be used for specific applications when there is an additional specification. The template definition includes reference to two levels of tables of names from which underscore-names should be drawn. The lower-level (local scope) set of <"_service"> names is defined in terms of other IANA tables, namely any table with symbolic names. The upper-level (global scope) "SRV" naming field is <"_proto">, although its pool of names is not explicitly defined.

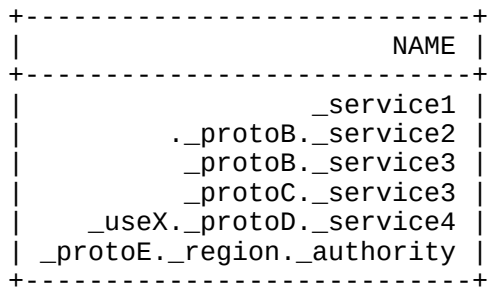
The current definition of a global underscore registry attends only to the "upper-level" names used for these RRs, that is the "_proto" names.

3. DNS Global Underscore Scoped Entry Registry Function

This specification creates a registry for DNS nodes names that begin with an underscore and are used to define scope of use for specific resource records. A given name defines a specific, constrained context for the use of such records. Within this scope, use of other resource records that are not specified is permitted. The purpose of the Underscore registry is to avoid collisions resulting from the use of the same underscore-based name, for different applications.

Structurally, the registry is defined as a single, flat table of names that begin with underscore. In some cases, such as for "SRV", an underscore 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 an underscore name in different underscore context; a subordinate name is meaningful only within the scope of the first (parent) underscore name. As such, they can be ignored by this DNS Global Underscore Scoped Entry Registry. That is, the registry is for the definition of highest-level underscore node name used.

+---+



Example of Underscore Names

Only the right-most names are registered in the IANA Underscore table. Definition and registration of the subordinate names is the responsibility of the specification that creates the highest-level (right-most) registry entry.

4. DNS Global Underscore Scoped Entry Registry Definition

A registry entry contains:

Name: Specifies a textual name for a scoped portion of the DNS. The name will usually be taken from the specification cited in the "Purpose" column and is intended for use in discussions about the entry.

DNS Label: Specifies a single _underscore name that defines a name reservation; this name is the "global" entry name for the scoped resource records that are associated with that name.

Constraints: Specifies any restrictions on use of the name.

RR(s): Lists the RRs that are defined for use within this scope.

References Lists specifications that define the records and their use under this Name.

Purpose: Specifies the particular purpose/use for specific "RR"(s), defined for use within the scope of the registered underscore name.

5. IANA Considerations

Per [RFC5226], IANA is requested to establish a DNS Global Underscore Scoped Entry Registry, for DNS node names that begin with the underscore character (_) and have been specified in any published RFC, or are documented by a specification published by another standards organization. The contents of each entry are defined in Section 4.

Initial entries in the registry are:

> { Enhancement of this table to include all underscore name reservations in effect at the time this document is published is left as an exercise to the readers... /d }

NAME	LABEL	RR	REFERENCE	PURPOSE
"SRV"	_srv	"SRV"	[RFC2782]	"SRV" template -- pro forma entry, not directly usable
"SRV" TCP	_tcp	"SRV"	[RFC2782]	Use of "SRV" for a TCP-based service
"SRV" UDP	_udp	"SRV"	[RFC2782]	Use of "SRV" for a UDP-based service
LDAP	_ldap	"SRV"	[RFC2782]	LDAP server
SIP	_sip	NAPTR	[RFC3263] [RFC6011]	Locating SIP Servers and UA configuration
SPF	_spf	"TXT"	[RFC7372]	Authorized IP addresses for sending mail
DKIM	_domainkey	"TXT"	[RFC6376]	Public key for verifying DKIM signature.
PKI LDAP	_PKIXREP	"SRV"	[RFC4386]	PKI Repository
VBR	_vouch	"TXT"	[RFC5518]	Vouch-by-refererence domain assertion
DDDS	--???!--	"SRV"	[RFC3404]	Mapping DDDS query to DNS records
SOAP BEEP	_soap-beep	"SRV"	[RFC4227]	SOAP over BEEP lookup, when no port specified

XMLRPC BEEP	_xmlrpc-beep	"SRV"	[RFC3529]	Resolve url for XML-RPC using BEEP
Diameter	_diameter	"SRV"	[RFC6733]	Diameter rendezvous
Tunnel	_tunnel	"SRV"	[RFC3620]	Finding the appropriate address for tunneling into a particular domain
SLP	_slpda	"SRV"	[RFC3832]	Discovering desired services in given DNS domains
Msg Track	_mtqp	"SRV"	[RFC3887]	Assist in determining the path that a particular message has taken through a messaging system
XMPP Client	_xmpp-client	"SRV"	[RFC6120]	XMPP client lookup of server
XMPP Server	_xmpp-server	"SRV"	[RFC6120]	XMPP server-server lookup
DDDS "SRV"	_???	"SRV" (and NAPTR ?)	[RFC3958]	Map domain name, application service name, and application protocol dynamically to target server and port
Kerberos	_kerberos	"SRV"	[RFC4120]	purpose
PKI	_pkixrep	"SRV"	[RFC4386]	Enables certificate-using systems to locate PKI repositories
Certificates	_certificates	"SRV"	[RFC4387]	Obtain certificates and certificate revocation lists (CRLs) from PKI repositories
PGP Key Store	_pgpkeys	"SRV"	[RFC4387]	Obtain certificates and certificate revocation lists

				(CRLs) from PKI repositories
MSRP Relay Locator	_msrp	"SRV"	[RFC4976]	purpose
Mobile IPv6 Bootstrap	_mip6	"SRV"	[RFC5026] [RFC5555]	Bootstrap Mobile IPv6 Home Agent information from non-topological information
Digital Video Broadcasting	_dvbservdsc	"SRV"	[RFC5328]	Discover non-default DVB entry points addresses
CAPWAP AC	_capwap-control	rrs	[RFC5415]	Discover the CAPWAP AC address(es)
IEEE 802.21 Mobility	_mihis	NAPTR, "SRV"	[RFC5679]	Discovering servers that provide IEEE 802.21-defined Mobility Services
STUN Client/Server	_stun	"SRV"	[RFC5389]	Find a STUN server
TURN	_turn	"SRV"	[RFC5766] [RFC5928]	Control the operation of a relay to bypass NAT
STUN NAT Behavior Discovery	_stun-behavior	"SRV"	[RFC5780]	Discover the presence and current behavior of NATs and firewalls between the STUN client and the STUN server
Sieve Management	_sieve	"SRV"	[RFC5804]	Manage Sieve scripts on a remote server
AFS VLDB	_afs3-vlserver	"SRV"	[RFC5864]	Locate services for the AFS distributed file system
AFS PTS	_afs3-prserver	"SRV"	[RFC5864]	Locate services for the AFS distributed file system
Mail MSA Submission	_submission	"SRV"	[RFC6186]	Locate email services
IMAP	_imap	"SRV"	[RFC6186]	Locate email

				services
POP	_pop3	"SRV"	[RFC6186]	Locate email services
POP TLS	_pop3s	"SRV"	[RFC6186]	Locate email services

Table 1: DNS Global Underscore Scoped Entry Registry (with initial values)

6. Related and Updated Registries

This section needs to contained details specification of the updates to existing underscore "registries", in order to have those specifications point to this new registry.

Numerous specifications have defined their own, independent registries for use of underscore names. It is likely that adoption of the proposed, integrated registry should render these piecemeal registries obsolete

Registries that are candidates for replacement include:

Instant Messaging "SRV" Protocol Label Registry

Public Key Infrastructure using X.509 (PKIX) Parameters

Presence "SRV" Protocol Label Registry

7. Security Considerations

This memo raises no security issues.

8. References

8.1. Normative References

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, May 2008.

8.2. References -- Informative

[RFC0974] Partridge, C., "Mail routing and the domain system", RFC 974, January 1986.

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

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

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[RFC3404] MMealling, M., "Dynamic Delegation Discovery System (DDDS) Part Four: The Uniform Resource Identifiers (URI) Resolution Application", RFC 3404, October 2002.

[RFC3529] Harold, W., "Using Extensible Markup Language-Remote Procedure Calling (XML-RPC) in Blocks Extensible Exchange Protocol (BEEP)", RFC 3529, April 2003.

[RFC3620] New, D., "The TUNNEL Profile", RFC 3620, October 2003.

[RFC3832] Columbia University, Columbia University, Sun Microsystems, IBM, and IBM, "Remote Service Discovery in the Service Location Protocol (SLP) via DNS SRV", RFC 3832, July 2004.

[RFC3887] "Message Tracking Query Protocol", RFC 3887, September 2007.

[RFC3958] Daigle, L. and A. Newton, "Domain-Based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)", RFC 3958, January 2005.

[RFC4120] USC-ISI, MIT, MIT, and MIT, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[RFC4227] O'Tuathail, E. and M. Rose, "Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP)", RFC 4227, January 2006.

[RFC4386] Boeyen, S. and P. Hallam-Baker, "Internet X.509 Public Key Infrastructure: Repository Locator Service", RFC 4386, February 2006.

[RFC4387] Gutmann, P., Ed., "Internet X.509 Public Key Infrastructure Operational Protocols: Certificate Store Access via HTTP", RFC 4387, February 2006.

[RFC4976] Jennings, C., Mahy, R., and Roach, "Relay Extensions for the Message Session Relay Protocol (MSRP)", RFC 4976, September 2007.

[RFC5026] Giaretta, G., Ed., Kempf, J., and V. Devarapalli, Ed., "Mobile IPv6 Bootstrapping in Split Scenario", RFC 5026, October 2007.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, Oct 2008.

[RFC5328] Adolf, A. and P. MacAvock, "A Uniform Resource Name (URN) Namespace for the Digital Video Broadcasting Project (DVB)", RFC 5328, September 2008.

[RFC5389] Rosenberg, , Mahy, , Matthews, , and Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5415] Calhoun, P., Ed., Montemurro, M., Ed., and D. Stanley, Ed., "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification", RFC 5415, March 2009.

[RFC5507] Faltstrom, P., Ed. and R. Austein, Ed., "Design Choices When Expanding the DNS", RFC 5507, April 2009.

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

[RFC5555] Soliman, H., Ed., "Mobile IPv6 Support for Dual Stack Hosts and Routers", RFC 5555, June 2009.

[RFC5679] Bajko, G., "Locating IEEE 802.21 Mobility Services Using DNS", RFC 5679, December 2009.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[RFC5780] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN)", RFC 5780, May 2010.

[RFC5804] Melnikov, A., Ed. and T. Martin, "A Protocol for Remotely Managing Sieve Scripts", RFC 5804, July 2010.

[RFC5864] Allbery, R., "NS SRV Resource Records for AFS", RFC 5864, April 2010.

[RFC5928] Petit-Huguenin, M., "Traversal Using Relays around NAT (TURN) Resolution Mechanism", RFC 5928, August 2010.

[RFC6011] Lawrence, S., Ed. and J. Elwell, "Session Initiation Protocol (SIP) User Agent Configuration", RFC 6011, October 2010.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6186] Daboo, C., "Use of SRV Records for Locating Email Submission/Access Services", RFC 6186, March 2011.

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

[RFC6733] Fajardo, V., Arkko, J., Loughney, J., and G. Zorn, "Diameter Base Protocol", RFC 6733, October 2012.

[RFC7372] Kitterman, S., "Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1", RFC 7372, April 2014.

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

8.3. URIs

[1] mailto:dnsop@ietf.org

Appendix A. Acknowledgements

Thanks go to Bill Fenner, Tony Hansen, Peter Koch, Olaf Kolkman, and Andrew Sullivan for diligent review of the (much) earlier drafts. For the later enhancements, thanks to: Tim Wicinski, John Levine, Bob Harold, Joel Jaeggli, Ondřej Sury and Paul Wouters. Special thanks to Ray Bellis for more than 10 years of 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: October 20, 2017

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

April 18, 2017

C-DNS: A DNS Packet Capture Format
draft-ietf-dnsop-dns-capture-format-02

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 October 20, 2017.

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
- Minimizing 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 the some common use cases in which such DNS data is collected Section 3
- A discussion of the major design considerations in developing an efficient data representation for collections of DNS messages Section 4
- A conceptual overview of the C-DNS format Section 5
- A description of why CBOR [RFC7049] was chosen for this format Section 6
- The definition of the C-DNS format for the collection of DNS messages Section 7.
- Notes on converting C-DNS data to PCAP format Section 9
- Some high level implementation considerations for applications designed to produce C-DNS Section 10

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 individual IPv4 or IPv6 packets. Typically these are UDP, but may be constructed from a TCP packet. "Message", unless otherwise qualified, refers to a DNS payload extracted from a UDP or TCP data stream.

The parts of DNS messages are named as they are in [RFC1035]. In specific, 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
  - 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 was 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 was also essential that interoperability with less restricted infrastructure was 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 clients 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.

2. Each Q/R data item will comprise a default Q/R data description and a set of optional sections. Inclusion of optional sections shall be configurable.

   * 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, however this will limit the ability to reconstruct packet captures. For example, omitting the resource records from a Response will reduce the files size, and 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.

   * [TODO: Further discussion of commonality between DNS messages e.g. common query signatures, a finite set of valid responses from authoritatives]

4. Metadata about other packets received can optionally be included in each block. For example, counts of malformed DNS packets and

non-DNS packets (e.g. ICMP, TCP resets) sent to the server may be of interest.

5. The wire format content of malformed DNS packets can 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 packets (see Section 8). Only those packets that can be transformed reasonably into the structured format can be represented by the format. 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 packets available directly in the C-DNS file.

5. 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 Q/R data item and Block tables format (PNG) [3]

Figure showing the Q/R data item and Block tables format (SVG) [4]

6. 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 ([RFC7159]) for debugging and other human inspection requirements.

- CBOR data schemas can be described using CDDL [I-D.greevenbosch-appsawg-cbor-cddl].

7. The C-DNS format

7.1. CDDL definition

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

7.2. Format overview

A C-DNS file begins with a file header containing a file type identifier and a preamble. The preamble contains information on the collection settings.

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

A block consists of a block header, containing various tables of common data, and some statistics for the traffic received over the block. The block header is then followed by a list of the Q/R data items detailing the queries and responses received during processing of the block input. The list of Q/R data items is in turn followed by a list of per-client counts of particular IP events that occurred during collection of the block data.

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.

If no field type is specified, then the field is unsigned.

In all quantities that contain bit flags, bit 0 indicates the least significant bit. An item described as an index is the index of the Q/R data item in the referenced table. Indexes are 1-based. An index value of 0 is reserved to mean "not present".

7.3. File header contents

The file header contains the following:

Field	Type	Description
file-type-id	Text string	String "C-DNS" identifying the file type.
file-preamble	Map of items	Collection information for the whole file.
file-blocks	Array of Blocks	The data blocks.

7.4. File preamble contents

The file preamble contains the following:

Field	Type	Description
major-format-version	Unsigned	Unsigned integer '1'. The major version of format used in file.
minor-format-version	Unsigned	Unsigned integer '0'. The minor version of format used in file.
private-version	Unsigned	Version indicator available for private use by applications. Optional.
configuration	Map of items	The collection configuration. Optional.
generator-id	Text string	String identifying the collection program. Optional.
host-id	Text string	String identifying the collecting host. Empty if converting an existing packet capture file. Optional.

7.5. Configuration contents

The collection configuration contains the following items. All are optional.

Field	Type	Description
query-timeout	Unsigned	To be matched with a query, a response must arrive within this number of seconds.
skew-timeout	Unsigned	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	Unsigned	Collect up to this many bytes per packet.
promisc	Unsigned	1 if promiscuous mode was enabled on the interface, 0 otherwise.
interfaces	Array of text strings	Identifiers of the interfaces used for collection.
server-addresses	Array of byte strings	Server collection IP addresses. Hint for downstream analysers; does not affect collection.
vlan-ids	Array of unsigned	Identifiers of VLANs selected for collection.
filter	Text string	'tcpdump' [pcap] style filter for input.
query-options	Unsigned	Bit flags indicating sections in Query messages to be collected. Bit 0. Collect second and subsequent Questions in the Question section. Bit 1. Collect Answer sections. Bit 2. Collect Authority sections. Bit 3. Collection Additional

			sections.
	response-options	Unsigned	Bit flags indicating sections in Response messages to be collected. Bit 0. Collect second and subsequent Questions in the Question section. Bit 1. Collect Answer sections. Bit 2. Collect Authority sections. Bit 3. Collection Additional sections.
	accept-rr-types	Array of text strings	A set of RR type names [rrtypes]. If not empty, only the nominated RR types are collected.
	ignore-rr-types	Array of text strings	A set of RR type names [rrtypes]. If not empty, all RR types are collected except those listed. If present, this item must be empty if a non-empty list of Accept RR types is present.
	max-block-qr-items	Unsigned	Maximum number of Q/R data items in a block.
	collect-malformed	Unsigned	1 if malformed packet contents are collected, 0 otherwise.

7.6. Block contents

Each block contains the following:

Field	Type	Description
preamble	Map of items	Overall information for the block.
statistics	Map of statistics	Statistics about the block. Optional.
tables	Map of tables	The tables containing data referenced by individual Q/R data items.
queries	Array of Q/R data items	Details of individual Q/R data items.
address-event-counts	Array of Address Event counts	Per client counts of ICMP messages and TCP resets. Optional.
malformed-packet-data	Array of malformed packets	Wire contents of malformed packets. Optional.

7.7. Block preamble map

The block preamble map contains overall information for the block.

Field	Type	Description
earliest-time	Array of unsigned	A timestamp for the earliest record in the block. The timestamp is specified as a CBOR array with two or three elements. The first two elements are as in Posix struct timeval. The first element is an unsigned integer time_t and the second is an unsigned integer number of microseconds. The third, if present, is an unsigned integer number of picoseconds. The microsecond and picosecond items always have a value between 0 and 999,999.

7.8. Block statistics

The block statistics section contains some basic statistical information about the block. All are optional.

Field	Type	Description
total-packets	Unsigned	Total number of packets processed from the input traffic stream during collection of the block data.
total-pairs	Unsigned	Total number of Q/R data items in the block.
unmatched-queries	Unsigned	Number of unmatched queries in the block.
unmatched-responses	Unsigned	Number of unmatched responses in the block.
malformed-packets	Unsigned	Number of malformed packets found in input for the block.

Implementations may choose to add additional implementation-specific fields to the statistics.

7.9. Block table map

The block table map contains the block tables. Each element, or table, is an array. The following tables detail the contents of each block table.

The Present column in the following tables indicates the circumstances when an optional field will be present. A Q/R data item may be:

- A Query plus a Response.
- A Query without a Response.
- A Response without a Query.

Also:

- A Query and/or a Response may contain an OPT section.
- A Question may or may not be present. If the Query is available, the Question section of the Query is used. If no Query is available, the Question section of the Response is used. Unless

otherwise noted, a Question refers to the first Question in the Question section.

So, for example, a field listed with a Present value of QUERY is present whenever the Q/R data item contains a Query. If the pair contains a Response only, the field will not be present.

7.10. IP address table

The table "ip-address" holds all client and server IP addresses in the block. Each item in the table is a single IP address.

Field	Type	Description
ip-address	Byte string	The IP address, in network byte order. The string is 4 bytes long for an IPv4 address, 16 bytes long for an IPv6 address.

7.11. Class/Type table

The table "classtype" holds pairs of RR CLASS and TYPE values. Each item in the table is a CBOR map.

Field	Description
type	TYPE value.
class	CLASS value.

7.12. Name/RDATA table

The table "name-rdata" holds the contents of all NAME or RDATA items in the block. Each item in the table is the content of a single NAME or RDATA.

Field	Type	Description
name-rdata	Byte string	The NAME or RDATA contents. NAMEs, and labels within RDATA contents, are in uncompressed label format.

7.13. Query Signature table

The table "query-sig" holds elements of the Q/R data item that are often common between multiple individual Q/R data items. Each item in the table is a CBOR map. Each item in the map has an unsigned value and an unsigned key.

The following abbreviations are used in the Present (P) column

- Q = QUERY
- A = Always
- QT = QUESTION
- QO = QUERY, OPT
- QR = QUERY & RESPONSE
- R = RESPONSE

Field	P	Description
server-address-index	A	The index in the IP address table of the server IP address.
server-port	A	The server port.
transport-flags	A	Bit flags describing the transport used to service the query. Bit 0 is the least significant bit. Bit 0. Transport type. 0 = UDP, 1 = TCP. Bit 1. IP type. 0 = IPv4, 1 = IPv6. Bit 2. Trailing bytes in query payload. The DNS query message in the UDP payload was followed by some additional bytes, which were discarded.
qr-sig-flags	A	Bit flags indicating information present in this Q/R data item. Bit 0 is the least significant bit. Bit 0. 1 if a Query is present. Bit 1. 1 if a Response is present. Bit 2. 1 if one or more Question is present.

		Bit 3. 1 if a Query is present and it has an OPT Resource Record. Bit 4. 1 if a Response is present and it has an OPT Resource Record. Bit 5. 1 if a Response is present but has no Question.
query-opcode	Q	Query OPCODE. Optional.
qr-dns-flags	A	Bit flags with values from the Query and Response DNS flags. Bit 0 is the least significant bit. 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	Q	Query RCODE. If the Query contains OPT, this value incorporates any EXTENDED_RCODE_VALUE. Optional.
query-classtype-index	QT	The index in the Class/Type table of the CLASS and TYPE of the first Question. Optional.
query-qd-count	QT	The QDCOUNT in the Query, or Response if no Query present.

			Optional.
	query-an-count	Q	Query ANCOUNT. Optional.
	query-ar-count	Q	Query ARCOUNT. Optional.
	query-ns-count	Q	Query NSCOUNT. Optional.
	edns-version	QO	The Query EDNS version. Optional.
	udp-buf-size	QO	The Query EDNS sender's UDP payload size. Optional.
	opt-rdata-index	QO	The index in the NAME/RDATA table of the OPT RDATA. Optional.
	response-rcode	R	Response RCODE. If the Response contains OPT, this value incorporates any EXTENDED_RCODE_VALUE. Optional.

7.14. Question table

The table "qrr" holds details on individual Questions in a Question section. Each item in the table is a CBOR map containing a single Question. Each item in the map has an unsigned value and an unsigned key. This data is optionally collected.

Field	Description
name-index	The index in the NAME/RDATA table of the QNAME.
classtype-index	The index in the Class/Type table of the CLASS and TYPE of the Question.

7.15. Resource Record (RR) table

The table "rr" holds details on individual Resource Records in RR sections. Each item in the table is a CBOR map containing a single Resource Record. This data is optionally collected.

Field	Description
name-index	The index in the NAME/RDATA table of the NAME.
classtype-index	The index in the Class/Type table of the CLASS and TYPE of the RR.
ttl	The RR Time to Live.
rdata-index	The index in the NAME/RDATA table of the RR RDATA.

7.16. Question list table

The table "qlist" holds a list of second and subsequent individual Questions in a Question section. Each item in the table is a CBOR unsigned integer. This data is optionally collected.

Field	Description
question	The index in the Question table of the individual Question.

7.17. Resource Record list table

The table "rrlist" holds a list of individual Resource Records in a Answer, Authority or Additional section. Each item in the table is a CBOR unsigned integer. This data is optionally collected.

Field	Description
rr	The index in the Resource Record table of the individual Resource Record.

7.18. Query/Response data

The block Q/R data is a CBOR array of individual Q/R data items. Each item in the array is a CBOR map containing details on the individual Q/R data item.

Note that there is no requirement that the elements of the Q/R array are presented in strict chronological order.

The following abbreviations are used in the Present (P) column

- Q = QUERY
- A = Always
- QT = QUESTION
- QO = QUERY, OPT
- QR = QUERY & RESPONSE
- R = RESPONSE

Each item in the map has an unsigned value (with the exception of those listed below) and an unsigned key.

- query-extended and response-extended which are of type Extended Information.
- delay-useconds and delay-pseconds which are integers (The delay can be negative if the network stack/capture library returns them out of order.)

Field	P	Description
time-useconds	A	Q/R timestamp as an offset in microseconds from the Block preamble Timestamp. The timestamp is the timestamp of the Query, or the Response if there is no Query.
time-pseconds	A	Picosecond component of the timestamp. Optional.
client-address-index	A	The index in the IP address table of the client IP address.
client-port	A	The client port.
transaction-id	A	DNS transaction identifier.
query-signature-index	A	The index of the Query Signature table record for this data item.
client-hoplimit	Q	The IPv4 TTL or IPv6 Hoplimit from the Query packet. Optional.

delay-useconds	QR	The time difference between Query and Response, in microseconds. Only present if there is a query and a response.
delay-pseconds	QR	Picosecond component of the time different between Query and Response. If delay-useconds is non-zero then delay-pseconds (if present) MUST be of the same sign as delay-useconds, or be 0. Optional.
query-name-index	QT	The index in the NAME/RDATA table of the QNAME for the first Question. Optional.
query-size	R	DNS query message size (see below). Optional.
response-size	R	DNS query message size (see below). Optional.
query-extended	Q	Extended Query information. This item is only present if collection of extra Query information is configured. Optional.
response-extended	R	Extended Response information. This item is only present if collection of extra Response information is configured. Optional.

An implementation must always collect basic Q/R information. It may be configured to collect details on Question, Answer, Authority and Additional sections of the Query, the Response or both. Note that only the second and subsequent Questions of any Question section are collected (the details of the first are in the basic information), and that OPT Records are not collected in the Additional section.

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 and will therefore include any trailing bytes if present. Trailing bytes with queries are routinely observed in traffic to authoritative servers and this value allows a calculation of how many trailing bytes were present. For TCP it is the size of the DNS message as specified in the two-byte message length header.

The Extended information is a CBOR map as follows. Each item in the map is present only if collection of the relevant details is configured. Each item in the map has an unsigned value and an unsigned key.

Field	Description
question-index	The index in the Questions list table of the entry listing any second and subsequent Questions in the Question section for the Query or Response.
answer-index	The index in the RR list table of the entry listing the Answer Resource Record sections for the Query or Response.
authority-index	The index in the RR list table of the entry listing the Authority Resource Record sections for the Query or Response.
additional-index	The index in the RR list table of the entry listing the Additional Resource Record sections for the Query or Response.

7.19. Address Event counts

This table holds counts of various IP related events relating to traffic with individual client addresses.

Field	Type	Description
ae-type	Unsigned	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	Unsigned	A code relating to the event. Optional.
ae-address-index	Unsigned	The index in the IP address table of the client address.
ae-count	Unsigned	The number of occurrences of this event during the block collection period.

7.20. Malformed packet records

This optional table records the original wire format content of malformed packets (see Section 8).

Field	Type	Description
time-useconds	A	Packet timestamp as an offset in microseconds from the Block preamble Timestamp.
time-pseconds	A	Picosecond component of the timestamp. Optional.
packet-content	Byte string	The packet content in wire format.

8. Malformed Packets

In the context of generating a C-DNS file it is assumed that only those packets which can be parsed to produce a well-formed DNS message are stored in the C-DNS format. This means as a minimum:

- The packet has a well-formed 12 bytes DNS Header
- The section counts are consistent with the section contents
- All of the resource records can be parsed

In principle, packets that do not meet these criteria could be classified into two categories:

- Partially malformed: those packets which can be decoded sufficiently to extract
  - a DNS header (and therefore a DNS transaction ID)
  - a QDCOUNT
  - the first Question in the Question section if QDCOUNT is greater than 0

  but suffer other issues while parsing. This is the minimum information required to attempt Query/Response matching as described in Section 10.1
- Completely malformed: those packets that cannot be decoded to this extent.

An open question is whether there is value in attempting to process partially malformed packets in an analogous manner to well formed packets in terms of attempting to match them with the corresponding query or response. This could be done by creating 'placeholder' records during Query/Response matching with just the information extracted as above. If the packet were then matched the resulting C-DNS Q/R data item would include a flag to indicate a malformed record (in addition to capturing the wire format of the packet).

An advantage of this would be that it would result in more meaningful statistics about matched packets because, for example, some partially malformed queries could be matched to responses. However it would only apply to those queries where the first Question is well formed. It could also simplify the downstream analysis of C-DNS files and the reconstruction of packet streams from C-DNS.

A disadvantage is that this adds complexity to the Query/Response matching and data representation, could potentially lead to false matches and some additional statistics would be required (e.g. counts for matched-partially-malformed, unmatched-partially-malformed, completely-malformed).

9. 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 9.1 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 packets 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.

9.1. 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.

10. Data Collection

This section describes a non-normative proposed algorithm for the processing of a captured stream of DNS queries and responses and 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 10.2.1)

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 application. It should be noted that packet capture libraries do not necessary provide packets in strict chronological order.

TODO: Discuss the corner cases resulting from this in more detail.

10.1. 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.

10.2. Message identifiers

10.2.1. 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

10.2.2. 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.

10.3. Algorithm Parameters

1. Query timeout

2. Skew timeout

10.4. 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.

10.5. 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.

10.6. Workspace

A FIFO structure is used to hold the Q/R data items during processing.

10.7. 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.

10.8. Post Processing

When ending capture, all remaining entries in the Q/R data item FIFO should be treated as timed out queries.

11. IANA Considerations

None

12. Security Considerations

Any control interface MUST perform authentication and encryption.

Any data upload MUST be authenticated and encrypted.

13. 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 and Jerry Lundstroem for review.

Also, Miek Gieben for mmark [7]

14. Changelog

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 packet 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 packets 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

15. References

15.1. Normative References

[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>.

[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>.

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

15.2. 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.greevenbosch-appsawg-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-10 (work in progress), March 2017.

[I-D.hoffman-dns-in-json] Hoffman, P., "Representing DNS Messages in JSON", draft-hoffman-dns-in-json-11 (work in progress), March 2017.

[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>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[rrtypes] IANA, "RR types", 2016, <http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-4>.

15.3. URIs

[1] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-02/cdns_format.png

[2] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-02/cdns_format.svg

[3] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-02/qr_data_format.png

[4] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-02/qr_data_format.svg

[5] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-02/packet_matching.png

[6] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-02/packet_matching.svg

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

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

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

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

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

[12] http://cbor.io

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

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

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

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

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

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

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

[20] 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.

File = [
    file-type-id  : tstr, ; = "C-DNS"
    file-preamble : FilePreamble,
    file-blocks   : [* Block],
]

FilePreamble = {
    major-format-version => uint,  ; = 1
    minor-format-version => uint,  ; = 0
    ? private-version    => uint,
    ? configuration      => Configuration,
    ? generator-id       => tstr,
    ? host-id            => tstr,
}

major-format-version = 0
minor-format-version = 1
private-version      = 2
configuration        = 3
generator-id         = 4
host-id              = 5

Configuration = {
    ? query-timeout      => uint,
    ? skew-timeout       => uint,
    ? snaplen            => uint,
    ? promisc            => uint,
    ? interfaces         => [* tstr],
```

```
      ? server-addresses   => [* IPAddress], ; Hint for later analysis
      ? vlan-ids           => [* uint],
      ? filter             => tstr,
      ? query-options      => QRCollectionSections,
      ? response-options   => QRCollectionSections,
      ? accept-rr-types    => [* uint],
      ? ignore-rr-types    => [* uint],
      ? max-block-qr-items => uint,
      ? collect-malformed  => uint,
  }

  QRCollectionSectionValues = &(
      question  : 0, ; Second & subsequent questions
      answer    : 1,
      authority : 2,
      additional: 3,
  )
  QRCollectionSections = uint .bits QRCollectionSectionValues

  query-timeout      = 0
  skew-timeout       = 1
  snaplen            = 2
  promisc            = 3
  interfaces         = 4
  vlan-ids           = 5
  filter             = 6
  query-options      = 7
  response-options   = 8
  accept-rr-types    = 9
  ignore-rr-types    = 10
  server-addresses   = 11
  max-block-qr-items = 12
  collect-malformed  = 13

  Block = {
      preamble               => BlockPreamble,
      ? statistics           => BlockStatistics,
      tables                 => BlockTables,
      queries                => [* QueryResponse],
      ? address-event-counts  => [* AddressEventCount],
      ? malformed-packet-data => [* MalformedPacket],
  }

  preamble             = 0
  statistics           = 1
  tables               = 2
  queries              = 3
  address-event-counts  = 4
```

```
  malformed-packet-data = 5

  BlockPreamble = {
      earliest-time => Timeval
  }

  earliest-time = 1

  Timeval = [
      seconds      : uint,
      microseconds  : uint,
      ? picoseconds : uint,
  ]

  BlockStatistics = {
      ? total-packets               => uint,
      ? total-pairs                 => uint,
      ? unmatched-queries           => uint,
      ? unmatched-responses         => uint,
      ? malformed-packets           => uint,
  }

  total-packets               = 0
  total-pairs                 = 1
  unmatched-queries           = 2
  unmatched-responses         = 3
  malformed-packets           = 4

  BlockTables = {
      ip-address => [* IPAddress],
      classtype  => [* ClassType],
      name-rdata => [* bstr], ; Holds both Name RDATA and RDATA
      query-sig  => [* QuerySignature]
      ? qlist    => [* QuestionList],
      ? qrr      => [* Question],
      ? rrlist   => [* RRList],
      ? rr       => [* RR],
  }

  ip-address = 0
  classtype  = 1
  name-rdata = 2
  query-sig  = 3
  qlist      = 4
  qrr        = 5
  rrlist     = 6
  rr         = 7
```

```
  QueryResponse = {
      time-useconds         => uint, ; Time offset from start of block
      ? time-pseconds       => uint, ; in microseconds and picoseconds
      client-address-index  => uint,
      client-port           => uint,
      transaction-id        => uint,
      query-signature-index => uint,
      ? client-hoplimit     => uint,
      ? delay-useconds      => int,
      ? delay-pseconds      => int, ; Has same sign as delay-useconds
      ? query-name-index    => uint,
      ? query-size          => uint, ; DNS size of query
      ? response-size       => uint, ; DNS size of response
      ? query-extended      => QueryResponseExtended,
      ? response-extended   => QueryResponseExtended,
  }

  time-useconds         = 0
  time-pseconds         = 1
  client-address-index  = 2
  client-port           = 3
  transaction-id        = 4
  query-signature-index = 5
  client-hoplimit       = 6
  delay-useconds        = 7
  delay-pseconds        = 8
  query-name-index      = 9
  query-size            = 10
  response-size         = 11
  query-extended        = 12
  response-extended     = 13

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

  type  = 0
  class = 1

  DNSFlagValues = &(
      query-cd   : 0,
      query-ad   : 1,
      query-z    : 2,
      query-ra   : 3,
      query-rd   : 4,
      query-tc   : 5,
      query-aa   : 6,
```

```
      query-d0   : 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

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

  TransportFlagValues = &(
      tcp               : 0,
      ipv6              : 1,
      query-trailingdata: 2,
  )
  TransportFlags = uint .bits TransportFlagValues

  QuerySignature = {
      server-address-index    => uint,
      server-port             => uint,
      transport-flags         => TransportFlags,
      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-ar-count        => uint,
      ? query-ns-count        => uint,
      ? edns-version          => uint,
      ? udp-buf-size          => uint,
      ? opt-rdata-index       => uint,
      ? response-rcode        => uint,
  }

  server-address-index  = 0
```

```
  server-port           = 1
  transport-flags       = 2
  qr-sig-flags          = 3
  query-opcode          = 4
  qr-dns-flags          = 5
  query-rcode           = 6
  query-classtype-index = 7
  query-qd-count        = 8
  query-an-count        = 9
  query-ar-count        = 10
  query-ns-count        = 11
  edns-version          = 12
  udp-buf-size          = 13
  opt-rdata-index       = 14
  response-rcode        = 15

  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

  RRList = [
      * uint, ; Index of RR
  ]

  RR = {
      name-index      => uint, ; Index to a name in the name-rdata table
      classtype-index => uint,
      ttl             => uint,
      rdata-index     => uint, ; Index to RDATA in the name-rdata table
  }

  ttl         = 2
  rdata-index = 3

  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

  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,
  )

  MalformedPacket = {
      time-useconds   => uint, ; Time offset from start of block
      ? time-pseconds => uint, ; in microseconds and picoseconds
      packet-content  => bstr, ; Raw packet contents
  }

  time-useconds    = 0
  time-pseconds    = 1
  packet-content   = 2

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

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 [8] 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 [9] 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 [10]. 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 [11]. 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 [12]. 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.greevenbosch-appsawg-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 [13]. A command line compression tool based on the Google Snappy [14] library.
- lz4 [15]. The command line compression tool from the reference C LZ4 implementation.
- gzip [16]. The ubiquitous GNU zip tool.
- zstd [17]. Compression using the Zstandard algorithm.
- xz [18]. 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 6.

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) [19]

Figure showing effect of block size on file size (SVG) [20]

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

Email: jad@sinodun.com

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

Email: jim@sinodun.com

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

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)
Intended status: Best Current Practice
Expires: September 14, 2017

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

March 13, 2017

DNS Terminology
draft-ietf-dnsop-terminology-bis-05

Abstract

The 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.

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.

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. (See Section 2 for a fuller definition.) 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://specs.webplatform.org/url/webspecs/develop/.

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 do not not have a clear definitions 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 zero 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 the definition in [RFC1034] 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 an acyclic directed graph. A domain name can be relative to other parts of the tree, or it can be fully qualified (in which case, it ends 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 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.

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 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, encoded as ASCII, with the root label last, and a "." character between each label. In presentation format, a fully-qualified domain name includes the root label and the associated separator dot. In presentation format, a fully-qualified domain name with two additional 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. In common display format, a fully-qualified domain name with two additional labels is usually shown as "example.tld" instead of "example.tld.". Names in the common display format are normally written such that the first label in the ordered list is in the last position from the point of view of the directionality of the writing system (so, in both English and C the first label is the right-most label; but in Arabic it may be the left-most label, depending on local conventions).

Administration of names -- Administration is specified by delegation (see the definition of to "delegation" in Section 6). 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 4). 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 defintion of "empty non-terminals" in Section 6).

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 7.

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.

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 tha 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 final, 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 some of the right-most names are left off and 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". 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, 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 [RFC6672]. 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)

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, and on which HTTP cookies ([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. At the time this document is published, the IETF DBOUND Working Group [DBOUND] is dealing with issues concerning 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 the case of the "uk" TLD around the time this document is published.

3. DNS Header and Response Codes

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 definitions are 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.)

[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."

Some of response codes that are defined in [RFC1035] have acquired their own shorthand names. Some common response code names that appear without reference to the numeric value are "FORMERR", "SERVFAIL", and "NXDOMAIN" (the latter of which is also referred to as "Name Error"). All of the RCODEs are listed at http://www.iana.org/assignments/dns-parameters, although that site uses mixed-case capitalization, while most documents use all-caps.

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.

Referrals: Data from the authority section of a non-authoritative answer. [RFC1035] Section 2.1 defines "authoritative" data. However, referrals at zone cuts (defined in Section 6) are not authoritative. Referrals may be zone cut NS resource records and their glue records. NS records on the parent side of a zone cut are an authoritative delegation, but are normally not treated as authoritative data. In general, a referral is a way for a server to send an answer saying that the server does not know the answer, but knows where the query should be directed in order to get an answer. Historically, many authoritative servers answered with a referral to the root zone when queried for a name for which they were not authoritative, but this practice has declined.

4. 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". (This definition is definitely not the same as "the response one gets to a query for QTYPE=ANY", which is an unfortunate misunderstanding.)

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.)

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. Also, if a value is flushed from the cache when its value is still positive, the value effectively becomes zero. 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.

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 classes other than IN (class 1, the Internet).

5. DNS Servers and Clients

This section defines the terms used for the systems that act as DNS clients, DNS servers, or both.

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.

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".

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". 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.

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).

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.

Priming: The mechanism used by a resolver to determine where to send queries before there is anything in the resolver’s cache. 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])

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 6 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). [RFC2182] describes secondary servers in 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". Secondary servers are also discussed in [RFC1034].

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]). Although early DNS RFCs such as [RFC1996] referred to this as a "master", the current common usage has shifted to "primary". Primary servers are also discussed in [RFC1034].

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.

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 master 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 be either a secondary or a primary master.

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 need 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.

View: A configuration for a DNS server that allows it to provide different answers depending on attributes of the query. 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 large amounts of DNS data by storing DNS responses from servers. Some of these systems also collect the DNS queries associated with the responses; this can raise privacy issues. Passive DNS databases can be used to answer historical questions about DNS zones such as which records were available for them at what times in the past. Passive DNS databases allow searching of the stored records on keys other than just the name, 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)

Split DNS: "Where a corporate network serves up partly or completely different DNS inside and outside its firewall. There are many possible variants on this; the basic point is that the correspondence between a given FQDN (fully qualified domain name) and a given IPv4 address is no longer universal and stable over long periods." (Quoted from [RFC2775], Section 3.8)

6. 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" as '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.

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.

In-bailiwick:

(a) An adjective to describe a name server whose name is either subordinate to or (rarely) the same as the zone origin. In-bailiwick name servers require glue records in their parent zone (using the first of the definitions of "glue records" in the definition above).

(b) Data for which the server is either authoritative, or else authoritative for an ancestor of the owner name. This sense of the term normally is used when discussing the relevancy of glue records in a response. For example, the server for the parent zone "example.com" might reply with glue records for "ns.child.example.com". Because the "child.example.com" zone is a

descendant of the "example.com" zone, the glue records are in-bailiwick.

Out-of-bailiwick: The antonym of in-bailiwick.

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) It is noted that this definition might inadvertently also include any NS records that appear in the zone, 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.

Root zone: The zone 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.

Wildcard: [RFC1034] defined "wildcard", but in a way that turned out to be confusing to implementers. 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) For an extended discussion of wildcards, including clearer definitions, see [RFC4592].

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)

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) It is often 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.)

Infrastructure domain: A domain whose "role is to support the operating infrastructure of the Internet". (Quoted from [RFC3172], Section 2.)

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.)

7. 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.

8. 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.)

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)

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 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 RR 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".

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) Note that the roles KSK and ZSK are not mutually exclusive: a single key can be both KSK and ZSK at the same time. 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 defintion 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." (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 RRSIG records at periodic intervals.

Signing software: Authoritative DNS servers that supports DNSSEC often contains 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.

9. 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.

10. Security Considerations

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

11. IANA Considerations

None.

12. References

12.1. 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, <http://www.rfc-editor.org/info/rfc882>.

[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>.

[RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, DOI 10.17487/RFC1996, August 1996, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc2182>.

[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>.

[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>.

[RFC4592] Lewis, E., "The Role of Wildcards in the Domain Name System", RFC 4592, DOI 10.17487/RFC4592, July 2006, <http://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, <http://www.rfc-editor.org/info/rfc5155>.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc6561>.

[RFC6672] Rose, S. and W. Wijngaards, "DNAME Redirection in the DNS", RFC 6672, DOI 10.17487/RFC6672, June 2012, <http://www.rfc-editor.org/info/rfc6672>.

[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>.

[RFC6840] Weiler, S., Ed. and D. Blacka, Ed., "Clarifications and Implementation Notes for DNS Security (DNSSEC)", RFC 6840, DOI 10.17487/RFC6840, February 2013, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7344>.

[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>.

12.2. Informative References

[DBOUND] IETF, "Domain Boundaries (dbound) Working Group", 2016, <https://datatracker.ietf.org/wg/dbound/charter/>.

[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, <http://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, <http://www.rfc-editor.org/info/rfc952>.

[RFC1995] Ohta, M., "Incremental Zone Transfer in DNS", RFC 1995, DOI 10.17487/RFC1995, August 1996, <http://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, <http://www.rfc-editor.org/info/rfc2133>.

[RFC2775] Carpenter, B., "Internet Transparency", RFC 2775, DOI 10.17487/RFC2775, February 2000, <http://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, <http://www.rfc-editor.org/info/rfc3172>.

[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, <http://www.rfc-editor.org/info/rfc3757>.

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

[RFC4641] Kolkman, O. and R. Gieben, "DNSSEC Operational Practices", RFC 4641, DOI 10.17487/RFC4641, September 2006, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc4956>.

[RFC5625] Bellis, R., "DNS Proxy Implementation Guidelines", BCP 152, RFC 5625, DOI 10.17487/RFC5625, August 2009, <http://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, <http://www.rfc-editor.org/info/rfc5890>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <http://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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc6055>.

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

[RFC6303] Andrews, M., "Locally Served DNS Zones", BCP 163, RFC 6303, DOI 10.17487/RFC6303, July 2011, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc6365>.

[RFC7129] Gieben, R. and W. Mekking, "Authenticated Denial of Existence in the DNS", RFC 7129, DOI 10.17487/RFC7129, February 2014, <http://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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7483>.

[RFC7484] Blanchet, M., "Finding the Authoritative Registration Data (RDAP) Service", RFC 7484, DOI 10.17487/RFC7484, March 2015, <http://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, <http://www.rfc-editor.org/info/rfc7485>.

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]

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: John Dickinson, Bob Harold, [[MORE NAMES WILL APPEAR HERE AS FOLKS CONTRIBUTE]].

Authors' Addresses

Paul Hoffman
ICANN

Email: paul.hoffman@icann.org

Andrew Sullivan
Dyn
150 Dow Street, Tower 2
Manchester, NH 03101
United States

Email: asullivan@dyn.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.

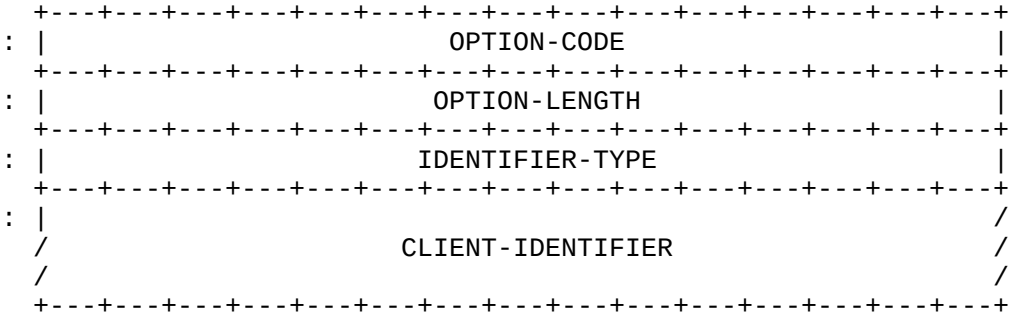
Nameserver A DNS server capable of resolving a DNS query and formulating a response.

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

DNSOP Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

D. Lawrence
Akamai Technologies
W. Kumari
Google

March 13, 2017

Serving Stale Data to Improve DNS Resiliency
draft-tale-dnsop-serve-stale-00

Abstract

This draft defines a method for recursive resolvers to use stale DNS data to avoid outages when authoritative nameservers cannot be reached to refresh expired data.

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/serve-stale>. The most recent version of the document, open issues, etc should all be available here. The authors gratefully accept pull requests.

Table of Contents

1. Introduction

Traditionally the Time To Live (TTL) of a DNS resource record has been understood to represent the maximum number of seconds that a record can be used before it must be discarded, based on its description and usage in [RFC1035] and clarifications in [RFC2181]. Specifically, [RFC1035] Section 3.2.1 says that it "specifies the time interval that the resource record may be cached before the source of the information should again be consulted".

Notably, the original DNS specification does not say that data past its expiration cannot be used. This document proposes a method for how recursive resolvers should handle stale DNS data to balance the competing needs of resiliency and freshness. It is predicated on the observation that authoritative server unavailability can cause outages even when the underlying data those servers would return is typically unchanged.

There are a number of reasons why an authoritative server may become unreachable, including Denial of Service (DoS) attacks, network issues, and so on. This document suggests that, if the recursive

server is unable to contact the authoritative server but still has data for the query name, it essentially extends the TTL of the existing data on the assumption that "stale bread is better than no bread".

Several major recursive resolver operations currently use stale data for answers in some way, including Akamai, OpenDNS, and Xerocole.

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].

For a comprehensive treatment of DNS terms, please see [RFC7719].

3. Description

Three notable timers drive considerations for the use of stale data, as follows:

- A client response timer, which is the maximum amount of time a recursive resolver should allow between the receipt of a resolution request and sending its response.

- A query resolution timer, which caps the total amount of time a recursive resolver spends processing the query.

- A maximum stale timer, which caps the amount of time that records will be kept past their expiration.

Recursive resolvers already have the second timer; the first and third timers are new concepts for this mechanism.

When a request is received by the recursive resolver, it SHOULD start the client response timer. This timer is used to avoid client timeouts. It SHOULD be configurable, with a recommended value of 1.8 seconds.

The resolver then checks its cache for an unexpired answer. If it finds none and the Recursion Desired flag is not set in the request, it SHOULD immediately return the response without consulting the cache for expired records.

If iterative lookups will be done, it SHOULD start the query resolution timer. This timer bounds the work done by the resolver, and is commonly around 10 to 30 seconds. [BIND 9 used to use a hard-coded constant of 30 seconds and has more recently added a

configuration parameter that defaults to 10 seconds and is capped at 30. A rigorous exploration of other implementations has not yet been done.]

If the answer has not been completely determined by the time the client response timer has elapsed, the resolver SHOULD then check its cache to see whether there is expired data that would satisfy the request. If so, it adds that data to the response message and SHOULD set the TTL of each expired record in the message to 1 second. [This 1 second TTL is ripe for discussion.] The response is then sent to the client while the resolver continues its attempt to refresh the data.

The maximum stale timer is used for cache management and is independent of the query resolution process. This timer is conceptually different from the maximum cache TTL that exists in many resolvers, the latter being a clamp on the value of TTLs as received from authoritative servers. The maximum stale timer SHOULD be configurable, and defines the length of time after a record expires that it SHOULD be retained in the cache. The suggested value is 7 days, which gives time to notice the resolution problem and for human intervention for fixing it.

This same basic technique MAY be used to handle stale data associated with delegations. If authoritative server addresses are not able to be refreshed, resolution can possibly still be successful if the authoritative servers themselves are still up.

4. Implementation Caveats

Answers from authoritative servers that have a DNS Response Code of either 0 (NOERROR) or 3 (NXDOMAIN) MUST be considered to have refreshed the data at the resolver. In particular, this means that this method is not meant to protect against operator error at the authoritative server that turns a name that is intended to be valid into one that is non-existent, because there is no way for a resolver to know intent.

Resolution is given a chance to succeed before stale data is used to adhere to the original intent of the design of the DNS. This mechanism is only intended to add robustness to failures, and to be enabled all the time. If stale data were used immediately and then a cache refresh attempted after the client response has been sent, the resolver would frequently be sending data that it would have had no trouble refreshing.

It is important to continue the resolution attempt after the stale response has been sent, until the query resolution timeout, because

some pathological resolutions can take many seconds to succeed as they cope with unavailable servers, bad networks, and other problems. Stopping the resolution attempt when the response with expired data has been sent would mean that answers in these pathological cases would never be refreshed.

Canonical Name (CNAME) records mingled in the expired cache with other records at the same owner name can cause surprising results. This was observed with an initial implementation in BIND, where a hostname changed from having a CNAME record to an IPv4 Address (A) record. BIND does not evict CNAMEs in the cache when other types are received, which in normal operations is not an issue. However, after both records expired and the authorities became unavailable, the fallback to stale answers returned the older CNAME instead of the newer A.

[This probably applies to other occluding types, so more thought should be given to the overall issue. It should probably also be rewritten to not suggest that this only a quirk of BIND.]

Keeping records around after their normal expiration will of course cause caches to grow larger than if records were removed at their TTL. Specific guidance on managing cache sizes is outside the scope of this document. Some areas for consideration include whether to track the popularity of names in client requests versus evicting by maximum age, and whether to provide a feature for manually flushing only stale records.

5. Implementation Status

[RFC Editor: per RFC 6982 this section should be removed prior to publication.]

The algorithm described in this draft was originally implemented as a patch to BIND 9.7.0. It has been in production on Akamai’s production network since 2011, and effectively smoothed over transient failures and longer outages that would have resulted in major incidents. The patch has been contributed to the Internet Systems Consortium in anticipation that it will be incorporated to their main BIND distribution.

6. Security Considerations

The most obvious security issue is the increased likelihood of DNSSEC validation failures when using stale data because signatures could be returned outside their validity period. This would only be an issue if the authoritative servers are unreachable, the only time the

techniques in this document are used, and thus does not introduce a new failure in place of what would have otherwise been success.

Additionally, bad actors have been known to use DNS caches to keep records alive even after their authorities have gone away. This makes that easier.

7. Privacy Considerations

This document does not add any practical new privacy issues.

8. NAT Considerations

The method described here is not affected by the use of NAT devices.

9. IANA Considerations

This document contains no actions for IANA. This section will be removed during conversion into an RFC by the RFC editor.

10. Acknowledgements

The authors wish to thank Matti Klock, Mukund Sivaraman, Jean Roy, and Jason Moreau for initial review.

11. References

11.1. Normative References

[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>.

[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>.

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

11.2. Informative References

[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>.

Authors' Addresses

David C Lawrence
Akamai Technologies
150 Broadway
Cambridge MA 02142-1054
USA

Email: tale@akamai.com

Warren Kumari
Google
1600 Amphitheatre Parkway
Mountain View CA 94043
USA

Email: warren@kumari.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 08, 2017

J. Vcelak
CZ.NIC
S. Goldberg
Boston University
D. Papadopoulos
University of Maryland
S. Huque
Salesforce
D. Lawrence
Akamai Technologies

March 07, 2017

NSEC5, DNSSEC Authenticated Denial of Existence
draft-vcelak-nsec5-04

Abstract

The Domain Name System Security Extensions (DNSSEC) introduced the NSEC resource record (RR) for authenticated denial of existence and the NSEC3 RR for hashed authenticated denial of existence. 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.

Status of This Memo

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) [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 FIPS 186-3 P-256 elliptic curve and on the Ed25519 elliptic curve. A formal cryptographic proof of security for elliptic curve (EC) 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 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, verify an NSEC5 proof, and process an NSEC5 proof to obtain its NSEC5 hash.

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 the 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 proof-to-hash 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. The server then responds with the 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 proof-to-hash 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 input for the NSEC5 proof computation is an RR owner name in [RFC4034] canonical wire format followed by a private NSEC5 key. The output is an octet string.

The input for the NSEC5 hash computation is the corresponding NSEC5 proof; the output is an octet string.

This document defines EC-P256-SHA256 NSEC5 algorithm as follows:

- The NSEC5 proof is computed using an Elliptic Curve VRF with FIPS 186-3 P-256 curve. The proof computation and verification, and the proof-to-hash function are formally specified in Appendix A. The curve parameters are specified in [FIPS-186-3] (Section D.1.2.3) and [RFC5114] (Section 2.6).

- The NSEC5 hash is the x-coordinate of the group element gamma from the NSEC5 proof (specified in Appendix A), encoded as a 32-octet unsigned integer in network byte order. In practice, the hash is a substring of the proof ranging from 2nd through 33th octet of the proof, inclusive.

- 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.

This document defines EC-ED25519-SHA256 NSEC5 algorithm as follows:

- The NSEC5 proof and NSEC5 hash are the same as with EC-P256-SHA256 but using Ed25519 elliptic curve with parameters defined in [RFC7748] Section 4.1.

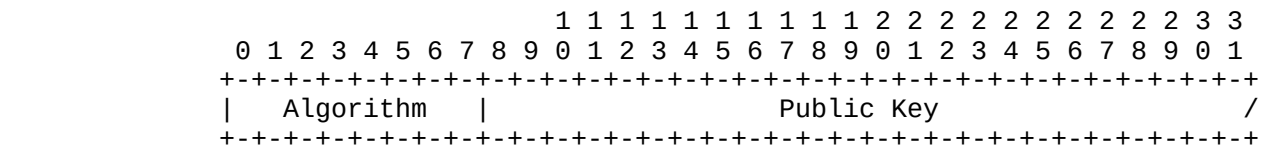
- The public key format to be used in the NSEC5KEY RR is defined in Section 3 of [RFC8080] and thus is the same as the format used to store Ed25519 public keys in DNSKEY RRs.

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 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.

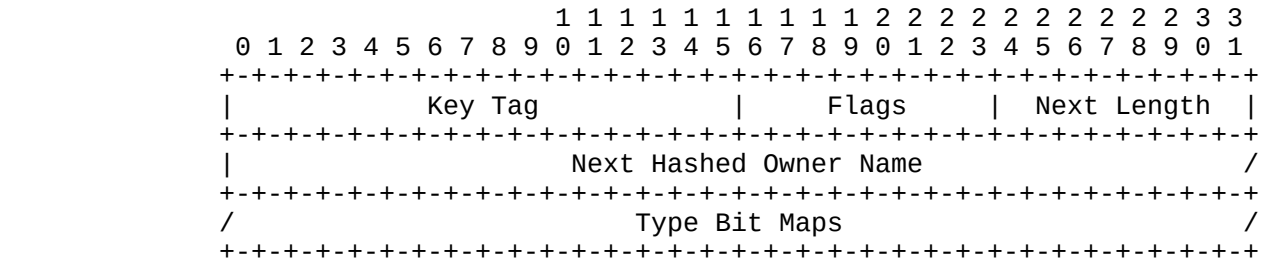
The Public Key field is represented in Base64 encoding. Whitespace is allowed within the Base64 text.

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 covered until the next owner name hashed in the RDATA.

6.1. NSEC5 RDATA Wire Format

The RDATA for 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, which is used to compute key tag values for DNSKEY RRs. The 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.

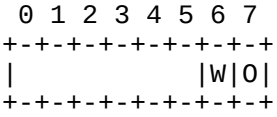
The Next Length field is an unsigned single octet specifying the length of the Next Hashed Owner Name field in octets.

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, 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.

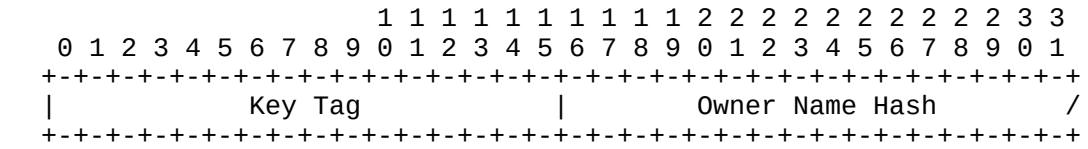
The Type Bit Maps representation is equivalent to the representation used in NSEC3 RR, described in [RFC5155].

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 NSEC5PROOF 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 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:

No RRset matching the QNAME exists.

No RRset matching the QNAME via wildcard expansion exists.

The QNAME does not fall into a delegation.

The QNAME does not fall into a DNAME redirection.

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.

Closest encloser has the Wildcard flag cleared.

Closest encloser does not have NS without SOA in the Type Bit Map.

Closest encloser does not have DNAME in the Type Bit Maps.

Next closer name is derived correctly.

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:

An RRset matching the QNAME exists.

No QTYPE RRset matching the QNAME exists.

No CNAME RRset matching the QNAME exists.

Authoritative server proofs:

NSEC5PROOF for the QNAME and matching NSEC5 RR.

Validator checks:

The QNAME is in the zone.

The QNAME does not have QTYPE in the Type Bit Map.

The QNAME does not have CNAME in the 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.

Closest provable encloser has the Opt-Out flag set.

8.3. Wildcard Responses

Facts to prove:

No RRset matching the QNAME exists.

No wildcard closer to the QNAME exists.

Authoritative server proofs:

NSEC5PROOF for next closer name and covering NSEC5 RR.

Validator checks:

Next closer name is derived correctly.

Next closer name is not in the zone.

8.4. Wildcard No Data Responses

Facts to prove:

No RRset matching the QNAME exists.

No QTYPE RRset exists at the wildcard matching the QNAME.

No CNAME RRset exists at the wildcard matching the QNAME.

No wildcard closer to the QNAME exists.

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:

Source of synthesis matches the QNAME.

Source of synthesis does not have QTYPE in the Type Bit Map.

Source of synthesis does not have CNAME in the Type Bit Map.

Next closer name is derived correctly.

Next closer name is not in the zone.

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. Generate the public and private NSEC5 keys.
2. Add a 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

The worst-case scenario when answering a negative query with NSEC5 requires authoritative server to respond with two NSEC5PROOF RRs and two NSEC5 RRs. Per Section 8, 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 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 unknown NSEC5 algorithm or 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 implementation did not introduce additional library dependencies; all cryptographic primitives are already present in OpenSSL v1.0.2j, which is used by both implementations. The implementation supports the full spectrum of negative responses, (i.e., NXDOMAIN, NODATA, Wildcard, Wildcard NODATA, and unsigned delegation). The implementation supports 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 attempt 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

[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, November 1987.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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>.

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

[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>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://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, <http://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, March 2008.

[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, <http://www.rfc-editor.org/info/rfc6234>.

[RFC6605] Hoffman, P. and W. Wijngaards, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", RFC 6605, April 2012.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <http://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,
<http://www.rfc-editor.org/info/rfc8080>.

[FIPS-186-3]
National Institute for Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-3, June 2009.

[SECG1] Standards for Efficient Cryptography Group (SECG), "SEC 1: Elliptic Curve Cryptography", Version 2.0, May 2009, <http://www.secg.org/sec1-v2.pdf>.

18.2. Informative References

[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>.

[nsec3gpu]
Wander, M., Schwittmann, L., Boelmann, C., and T. Weis, "GPU-Based NSEC3 Hash Breaking", in IEEE Symp. Network Computing and Applications (NCA), 2014.

[nsec3walker]
Bernstein, D., "Nsec3 walker", 2011, <http://dnscurve.org/nsec3walker.html>.

[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>.

[nsec3map]
anonion0, ., "nsec3map with John the Ripper plugin", 2015, <https://github.com/anonion0/nsec3map.>.

[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.

[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>.

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

[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>.

[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.

Appendix A. Elliptic Curve VRF

The Elliptic Curve Verifiable Random Function (EC-VRF) operates in a cyclic group G of prime order with generator g. The cyclic group G MAY be over the NIST-P256 elliptic curve, with curve parameters as specified in [FIPS-186-3] (Section D.1.2.3) and [RFC5114] (Section 2.6). The group G MAY alternatively be over the Ed25519 elliptic curve with parameters defined in [RFC7748] (Section 4.1). The security of this VRF follows from the decisional Diffie-Hellman (DDH) assumption in the cyclic group G in the random oracle model. Formal security proofs for this VRF are in [nsec5ecc].

Fixed options:

G - elliptic curve (EC) group

Used parameters:

g^x - EC public key

x - EC private key

q - prime order of group G

g - generator of group G

Used primitives:

"" - empty octet string

|| - octet string concatenation

p^k - EC point multiplication

p1*p2 - EC point addition

SHA256 - hash function SHA-256 as specified in [RFC6234]

ECP2OS - EC point to octet string conversion with point compression as specified in Section 2.3.3 of [SECG1]

OS2ECP - octet string to EC point conversion with point compression as specified in Section 2.3.4 of [SECG1]

A.1. EC-VRF Auxiliary Functions

A.1.1. EC-VRF Hash To Curve

ECVRF_hash_to_curve(m)

Input:

m - value to be hashed, an octet string

Output:

h - hashed value, EC point

Steps:

1. c = 0
2. C = I2OSP(c, 4)
3. xc = SHA256(m || C)
4. p = 0x02 || xc
5. If p is not a valid octet string representing encoded compressed point in G:

   a. c = c + 1

b. Go to step 2.

6. h = OS2ECP(p)

7. Output h

A.1.2. EC-VRF Hash Points

ECVRF_hash_points(p_1, p_2, ..., p_n)

Input:

p_x - EC point in G

Output:

h - hash value, integer between 0 and 2^128-1

Steps:

1. P = ""

2. for p in [p_1, p_2, ... p_n]:
P = P || ECP2OS(p)

3. h’ = SHA256(P)

4. h = OS2IP(first 16 octets of h’)

5. Output h

A.1.3. EC-VRF Decode Proof

ECVRF_decode_proof(pi)

Input:

pi - VRF proof, octet string (81 octets)

Output:

gamma - EC point

c - integer between 0 and 2^128-1

s - integer between 0 and 2^256-1

Steps:

1. let gamma', c', s' be pi split after 33-rd and 49-th octet
2. gamma = OS2ECP(gamma')
3. c = OS2IP(c')
4. s = OS2IP(s')
5. Output gamma, c, and s

A.2. EC-VRF Proving

ECVRF_PROVE(g^x, x, alpha)

Input:

g^x - EC public key

x - EC private key

alpha - message to be signed, octet string

Output:

pi - VRF proof, octet string (81 octets)

beta - VRF hash, octet string (32 octets)

Steps:

1. h = ECVRF_hash_to_curve(alpha)
2. gamma = h^x
3. choose a nonce k from [0, q-1]
4. c = ECVRF_hash_points(g, h, g^x, h^x, g^k, h^k)
5. s = k - c*q mod q
6. pi = ECP2OS(gamma) || I2OSP(c, 16) || I2OSP(s, 32)
7. beta = h2(gamma)
8. Output pi and beta

A.3. EC-VRF Proof To Hash

ECVRF_PROOF2HASH(gamma)

Input:

gamma - VRF proof, EC point in G with coordinates (x, y)

Output:

beta - VRF hash, octet string (32 octets)

Steps:

1. beta = I2OSP(x, 32)

2. Output beta

Note: Because of the format of the compressed form of an elliptic curve, the hash can be retrieved from an encoded gamma simply by omitting the first octet of the gamma.

A.4. EC-VRF Verifying

ECVRF_VERIFY(g^x, pi, alpha)

Input:

g^x - EC public key

pi - VRF proof, octet string

alpha - message to verify, octet string

Output:

"valid signature" or "invalid signature"

beta - VRF hash, octet string (32 octets)

Steps:

1. gamma, c, s = ECVRF_decode_proof(pi)

2. u = (g^x)^c * g^s

3. h = ECVRF_hash_to_curve(alpha)

4. v = gamma^c * h^s

5. c’ = ECVRF_hash_points(g, h, g^x, gamma, u, v)

6. beta = ECVRF_PROOF2HASH(gamma)

7. If c and c’ are the same, output "valid signature"; else output "invalid signature". Output beta.

[[TODO: check validity of gamma before hashing]]

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.

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
University of Maryland
8223 Paint Branch Dr
College Park, MD 20740
USA

EMail: dipapado@umd.edu

Shumon Huque
Salesforce
2550 Wasser Terrace
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: August 31, 2017

W. Kumari
Google
E. Hunt
ISC
R. Arends
Nominet
W. Hardaker
USC/ISI

February 27, 2017

Extended DNS Errors
draft-wkumari-dnsop-extended-error-00

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.

[Note: I always have a hard time with EDNS terminology - I'm saying that Extended DNS Errors are carried as EDNS Options, but is this correct? They are optional TLVs in "options" in RDATA in OPTion RR , but that's not readable.]

[Open question: The document currently defines a registry for errors. It has also been suggested that the option also carry human readable (text) messages, so 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.]

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 August 31, 2017.

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), their machine receives 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 resulver 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 either 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 whether to retry or not, or by technical users attempting to debug issues.

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 a resolver to make a decision whether or no to retry, or by administrators and technical users attempting to debug issues.

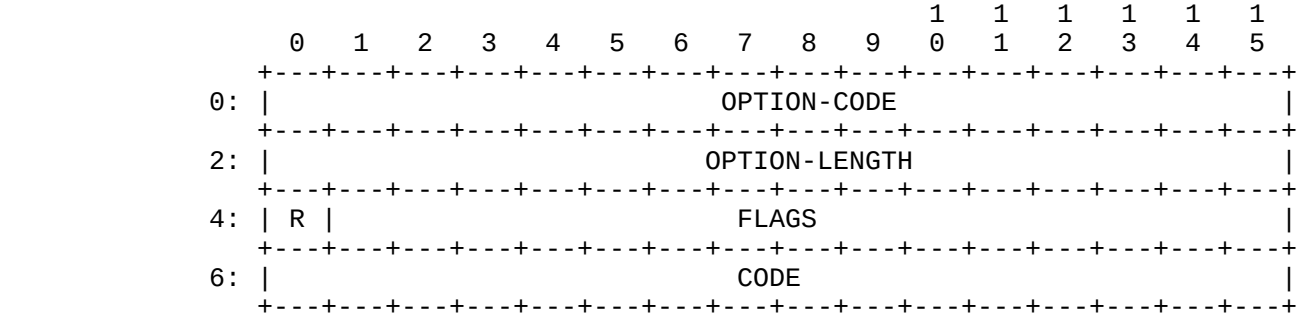
Here is a reference to an "external" (non-RFC / draft) thing: ([IANA.AS_Numbers]). And this is a link to an ID:[I-D.ietf-sidr-iana-objects].

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:



- o OPTION-CODE, 2 octets (Defined in [RFC6891]), for ExtError is TBD.
- o OPTION-LENGTH, 2 octets ((Defined in [RFC6891]) contains the length of the payload (everything after OPTION-LENGTH) in octets and should be 4.
- o FLAGS, 2 octets.
- o 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]

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?

7. Security Considerations

DNSSEC is being deployed - unfortunately a significant number of clients (TODO: Link to Geoff H stats), 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"

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

[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 -00 to -01;

- Placeholder to remind me to include changelog.

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

dnsop
Internet-Draft
Updates: 2308, 4033, 4034, 4035
(if approved)
Intended status: Standards Track
Expires: August 15, 2017

J. Woodworth
D. Ballew
S. Bindinganaveli Raghavan
CenturyLink, Inc.

February 15, 2017

BULK DNS Resource Records
draft-woodworth-bulk-rr-05

Abstract

The BULK DNS resource record type defines a method of pattern based creation of DNS resource records to be used in place of NXDOMAIN errors which would normally be returned. These records are currently restricted to registered DNS resource record types A, AAAA, PTR and CNAME. The key benefit of the BULK resource record type is the simplification of maintaining "generic" record assignments which would otherwise be too many to manage or require scripts or proprietary methods as bind's $GENERATE.

This document updates RFCs 2308, 4033, 4034 and 4035.

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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

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."

Copyright Notice

Table of Contents:

1. Introduction

The BULK DNS Resource Record (BULK) defines a maskable pattern based method for real-time on-the-fly resource record generation. Specifically, it allows one to manage large blocks of DNS records based entirely on record owner data in the RR query and patterns (or templates) designed by knowledgeable zone administrators. Existing DNS resource records covered by this document are Address (A), IPv6 Address (AAAA), Pointer (PTR) and Canonical Name (CNAME). Although other RR types are not explicitly forbidden from use with BULK logic they fall outside of scope and will not be discussed in this document. This document defines the purpose of this new resource record (BULK), its RDATA format, its presentation format (ASCII representation) as well as generated responses to matched DNS queries.

Two Key benefits of this record type are; a) the ability to transfer BULK RR intentions from primary to secondary nameservers with minimal bandwidth and memory requirements; and b) the ability to manage large volumes of pattern based records such as an IPv6 /64 CIDR or larger in a single entry.

Support options for DNSSEC related complications resulting from dynamically generated records are also provided in this document. One such option is in the form of the Numeric Pattern Normalization (NPN) resource record type also described in this document. NPN resource records provide a way of generating pattern based DNSSEC signatures and securely performing DNSSEC validation on such

signatures.

1.1. Background and Related Documents

This document assumes the reader is familiar with the basic DNS concepts described in [RFC1034], [RFC1035], and the subsequent documents that update them, particularly [RFC2181] and [RFC2308].

The reader is also assumed to be familiar with DNSSEC basics as described in [RFC4033], [RFC4034] and [RFC4035] as well as the DNS cryptographic signature generation process described in [RFC4033], [RFC4034], [RFC4035], [RFC2536], [RFC2931] and [RFC3110].

1.2. 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 BULK Resource Record

The BULK resource record consists of details which enable a DNS nameserver to generate RRs of other types based upon query received and patterns provided. Unless otherwise stated the letters used in hexadecimal numbers (a-f) MUST be case insensitive and are assumed to be lowercase. All examples in this document using hexadecimal are provided in lowercase.

The Type value for the BULK RR type is XX.

The BULK RR is class independent.

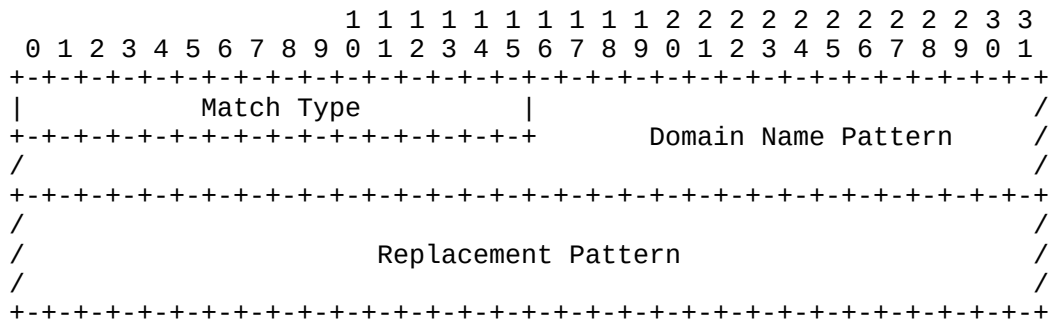
The BULK RR has no special TTL requirements but some security guidelines are offered in a later section.

2.1. BULK OPTIONAL Hidden Wildcards

The BULK RR extends current wildcard substitution logic as defined in [RFC1034] by allowing a single hyphen "-" in the leftmost label to represent the intent of leveraging a modified wildcard matching mechanism. If this condition exists wildcard logic SHALL be used for generated replacement records but not for the BULK resource records themselves. This will become clearer in the "BULK Replacement" section of this document. If an asterisk "*" (the standard wildcard character) is used default wildcard behavior MUST be used.

2.2. BULK RDATA Wire Format

The RDATA for a BULK RR consists of a 2 octet Match Type Field, a Domain Name Pattern Field and a Replacement Pattern Field.



2.2.1. The Match Type Field

The Match Type field identifies the type of the RRset identified by this BULK record. This field consists of two octets corresponding to an RR TYPE code as specified in [RFC1035], Section 3.2.1.

2.2.2. The Domain Name Pattern Field

The Domain Name Pattern Field consists of a text string which may be evaluated by the sections below. The character encoding for this field is [US-ASCII] and may not contain whitespace unless enclosed within double-quote characters. The value of a single hyphen "-" has special implications and will be discussed in greater detail below.

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234].

```
DIGIT  = <as defined in RFC 5234 Appendix B.1>
HEXDIG = <as defined in RFC 5234 Appendix B.1>
DQUOTE = <as defined in RFC 5234 Appendix B.1>

pattern             =   "-" / 1*part / DQUOTE 1*part DQUOTE

part                =   "[" range "]" / string

range               =   number [ "-" number ]

number              =   1*DIGIT / 1*HEXDIG

string              =   1*( %x01-5A / %x5C / %x5E-7F )
                            ; Any [US-ASCII] character excluding
                            ; NUL and square bracket characters
                            ; "[" or "]"
```

Although allowed by [RFC2181]; the square bracket characters, "[" and "]", are reserved to enclose a range specification and MUST NOT appear anywhere outside of a range specification.

2.2.2.1. Single hyphen

If the domain name pattern field consists of a single hyphen it is not necessary to evaluate for numeric ranges or strings. Implementors SHOULD simply set a flag indicating all ranges matching the query's label are true and backreferences (described in further detail in the "BULK Replacement" section) will be automatically set.

2.2.2.2. Numeric ranges

Numeric ranges include decimal or hexadecimal ranges depending on which record type was used in the query. This logic will be described in further detail in the "Replacement Logic" section.

The numeric range pattern will be a range of allowed numbers lower and upper values separated by a single hyphen "-". If upper and lower values are identical a single numeric value (without hyphen) will suffice. To easily distinguish numeric range patterns from string values they MUST be enclosed within square brackets "[" and "]".

2.2.2.3. String values

All values found before or after Numeric ranges (excluding single-hyphen rule) are considered to be string values. These values will be taken literally when evaluating for pattern matches in the "BULK Replacement" section below.

2.2.3. The Replacement Pattern Field

The Replacement Pattern field describes how the answer RRset SHOULD be generated for the matching query. It can either be a single hyphen "-" or a string containing backreferences (described in further detail in the "BULK Replacement" section). This field MUST be evaluated for proper syntax for resource records of its Match Type defined above. A "read" evaluation MAY be performed when a zone is first committed to memory either while converting from Text to Wire format (from stored zone files) or when a RR transfer is received (raw Wire format). Stage two "write" evaluations MUST be performed prior to returning generated replacement answers. Since logic to perform a stage two evaluation is already a requirement for DNS nameservers it may be easier for implementors to perform just stage two evaluations. Stage-two-only evaluation may be also preferred for performance purposes and is acceptable behavior. Any stage two evaluation errors MUST be processed as if the record did not exist and if all BULK generated records for a query answer-set evaluate to errors the original condition of an NXDOMAIN error state MUST be restored.

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234].

```
DIGIT  = <as defined in RFC 5234 Appendix B.1>
HEXDIG = <as defined in RFC 5234 Appendix B.1>
DQUOTE = <as defined in RFC 5234 Appendix B.1>

pattern             =   "-" / 1*part / DQUOTE 1*part DQUOTE

part                =   backreference / string

backreference       =   "$" "{" substitution "}"

substitution        =   range 0*( "," range ) [ options ]

substitution        =/  "*" [ options ]

options             =   delimiter [ interval [ padding ] ]

delimiter           =   "|" 0*1( %x01-23 / %x25-7A / %7E-7F )
                           ; Any single [US-ASCII] character
                           ; excluding NUL, dollar sign "$",
                           ; pipe "|" and curly brace characters
                           ; "{" or "}"

interval            =   "|" *2DIGIT

padding             =   "|" *2DIGIT

range               =   number [ "-" number ]

number              =   1*DIGIT / 1*HEXDIG

string              =   1*( %x01-23 / %x25-7A / %x7C / %7E-7F )
                           ; Any [US-ASCII] character excluding
                           ; NUL, dollar sign "$" and curly brace
                           ; characters "{" or "}"
```

The dollar sign, "$", and curly brace characters, "{" and "}", are reserved to enclose regular-expression-esque backreferences and MUST NOT appear anywhere outside of such a backreference specification. This rigidity is necessary to simplify implementation of this document and may relax once adoption reaches an acceptable level and demand for such an exception exists. The authors feel this limitation is a reasonable limitation for the flexibility offered by this document.

2.3. The BULK RR Presentation Format

The Match Type field is represented as an RR type mnemonic. When the

mnemonic is not known, the TYPE representation as described in [RFC3597], Section 5, MUST be used.

The Domain Name Pattern and Replacement Pattern fields MUST be presented as the TXT RR type described in [RFC1035], Section 3.3.14.

2.4. BULK RR Examples

EXAMPLE 1
The following BULK RR stores a block of A RRs for example.com.

```
*.example.com. 86400 IN BULK A (
                                  pool-A-[0-255]-[0-255].example.com.
                                  10.55.${1}.${2}
                                )
```

The first four fields specify the owner name, TTL, Class, and RR type (BULK). Value "A" indicates that this BULK RR defines the A record type (Address). Value "pool-A-[0-255]-[0-255].example.com." indicates the Domain Name Pattern. Value "10.55.${1}.${2}" indicates the Replacement Pattern. The owner in this example is a wildcard and matches any query ending with the string right of the asterisk.

EXAMPLE 2
The following BULK RR stores the reverse block of PTR records for the first example.

```
*.55.10.in-addr.arpa. 86400 IN BULK PTR (
                                  [0-255].[0-255].55.10.in-addr.arpa.
                                  pool-A-${1}-${2}.example.com.
                                )
```

The first four fields specify the owner name, TTL, Class, and RR type (BULK). Value "PTR" indicates that this BULK RR defines the PTR record type (Pointer). Value "[0-255].[0-255].55.10.in-addr.arpa." indicates the Domain Name Pattern. Value "pool-A-${1}-${2}.example.com." indicates the Replacement Pattern. The owner in this example is a wildcard and matches any query ending with the string right of the asterisk.

Additional examples can be found in the "BULK Replacement" section.

3. BULK Replacement

The BULK Record is designed to enable DNS zone maintainers to manage large blocks of DNS RRs which all conform to a common pattern. The Domain Name Pattern field provides both a tertiary filter (after owner and type) and a definition of all numeric pattern ranges.

When a query is first received by a DNS nameserver it begins its job

of locating an answer-set. In its simplest form this begins by locating the query owner (or wildcard suffix), class and type then returning any matching RR RDATA (or errors).

In the event no matches for the query are found the nameserver of authority will return an error type defined as NXDOMAIN. In the case of a "BULK" enabled authoritative nameserver an additional step MUST be performed. The nameserver MUST query its local RR database for any "BULK" RRs with a matching owner, class and compatible Match Type. If any such RRs are found the query's owner MUST then be matched against the Domain Name Pattern and all matching BULK records MUST be placed into a temporary processing answer-set. This temporary processing answer-set MUST then follow the Replacement Pattern for each matched record and provided no errors are found SHALL then write this new answer-set to the query's complete answer set. Matching replacements will be of the type specified in the Match Type field of the corresponding BULK RR. Additional detail is provided in the following sections.

3.1. Matching BULK "owner" field

The owner field of all BULK records MUST be that of either a wildcard or hidden wildcard as defined in previous sections. While a hidden wildcard will not be searched for BULK records it will be added to the database for use with the corresponding type field of each BULK RR. This allows location of BULK records to be less conspicuous to the public while still leveraging logic already included in the nameserver thus minimizing the complexity of implementation.

A query SHALL pass the first filter stage (owner match) ONLY IF: (1) an NXDOMAIN is set as the query's current answer set AND (2) the query's owner ends with the BULK record's owner field past the leading hyphen "-" or asterisk "*".

3.2. Matching the BULK "Match Type" field

The RR type of the received query must be compatible with that of the Match Type of owners matched in the section above. That is to say a query for an "A" record will only match BULK records with matching owner and Match Types of "A" (or "CNAME"). All other BULK records matching the query's owner are incompatible and MUST be ignored as part of the selected answer set.

3.3. Matching the BULK "Domain Name Pattern" field

Assuming the RR owner and Match Type fields match the next step is to find compatible Domain Name Patterns. The logic for this falls into two categories; automatic and manual which are described in greater detail in the following sections.

3.3.1. Automatic Domain Name Pattern matching

Automatic Domain Name Pattern matching is determined by use of a single hyphen "-" as the value for Domain Name Pattern field. This assumes everything matches and all hexadecimal or decimal fields will be captured for use as backreferences in the Replacement Pattern (described below). Automatic Domain Name Pattern matching is often preferred for large blocks such as the reverse IPv6 address space for the simplicity of record management.

3.3.2. Manual Domain Name Pattern matching

Manual Domain Name Pattern matching, while more complex is designed to be both simple to implement and simple to use. Below is an example implementation for label matching using a combination of parsing by regular expression and matching of numeric ranges.

Domain Name Patterns evaluate to current zone ORIGIN as defined in [RFC1034], Section 3. In short this means all Manual Domain Name Patterns must be terminated with a period "." or are assumed relative to the RR’s origin.

Numeric Ranges are either decimal or hexadecimal as determined by conditions of query.

If query type is "A" ranges are set to decimal.

If query type is "AAAA" ranges are set to hexadecimal.

If query type is PTR or CNAME the RR owner is used to determine decimal or hexadecimal.

If RR owner ends in ".ip6.arpa." ranges are set to hexadecimal.

If RR owner does _not_ end in ".ip6.arpa." ranges are set to decimal.

The square bracket characters, "[" and "]", are reserved to enclose a range specification and MUST NOT appear anywhere outside of a range specification.

3.3.2.1. Manual Domain Name Pattern matching examples

EXAMPLE 1
For this example the query is defined as a PTR record for "10.2.3.4" with an origin of "2.10.in-addr.arpa." and the evaluating BULK RR as:

```
-.2.10.in-addr.arpa. 86400 IN BULK PTR (
                                        [0-255].[0-10]
                                        pool-A-${1}-${2}.example.com.
                                        )
```

STEP 1
Ensure "Domain Name Pattern" is Fully Qualified

[0-255].[0-10] == [0-255].[0-10].2.10.in-addr.arpa.

STEP 2
Determine whether range is decimal or hexadecimal

Query type == "PTR" AND RR owner != "*.ip6.arpa." so range is decimal.

STEP 3
Build regular expression based on fully qualified domain name pattern.

```
[0-255].[0-10].2.10.in-addr.arpa. ==
               /^([0-9]{1,3})\.([0-9]{1,2})\.2\.10\.in-addr\.arpa\.$/
```

The above regular expression simply matches numeric ranges based on decimal or hexadecimal and length. Numeric range validation occurs in the next step.

STEP 4
Compare captured numbers and validate ranges

```
4.3.2.10.in-addr.arpa.
            =~ /^([0-9]{1,3})\.([0-9]{1,2})\.2\.10\.in-addr\.arpa\.$/
```

"4" is captured and within range 0-255 (decimal)
"3" is captured and within range 0-10 (decimal)

EXAMPLE 2
For this example the query is defined as a PTR record for "fc00::55" with an origin of "0.0.c.f.ip6.arpa." and the evaluating BULK RR as:

```
-.0.0.c.f.ip6.arpa. 86400 IN BULK PTR (
                                        -
                                        pool-${1-16|}-${17-
                                        28|}.example.com.
                                        )
```

STEP 1
Ensure "Domain Name Pattern" is Fully Qualified

```
- == [0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].       ~~
     [0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].       ~~
     [0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].       ~~
     [0-f].[0-f].[0-f].[0-f].0.0.c.f.ip6.arpa.
```

NOTE: Data above is shown in multiple lines for clarity.

Since Hyphen invokes "Automatic Domain Name Pattern" matching, all fields are captured for future use as backreferences.

STEP 2
Determine whether range is decimal or hexadecimal

Query type == "PTR" AND RR owner == "*.ip6.arpa." so range is hexadecimal.

STEP 3
Build regular expression based on fully qualified domain name pattern.

```
[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].            ~~
[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].            ~~
[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].[0-f].            ~~
[0-f].[0-f].[0-f].[0-f].0.0.c.f.ip6.arpa. ==
               /^([0-9a-f]{1}\.){28}\.0\.0\.c\.f\.ip6\.arpa\.$/
```

NOTE: Data above is shown in multiple lines for clarity.

The above regular expression simply matches numeric ranges based on decimal or hexadecimal and length. Numeric range validation occurs in the next step.

STEP 4
Compare captured numbers and validate ranges

```
5.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.                            ~~
0.0.0.0.0.0.0.0.0.0.0.0.0.0.c.f.ip6.arpa.
            =~ /^([0-9a-f]{1}\.){28}\.0\.0\.c\.f\.ip6\.arpa\.$/
```

NOTE: Data above is shown in multiple lines for clarity.

"5" is captured and within range 0-f (hexadecimal)
"5" is captured and within range 0-f (hexadecimal)
...
"0" is captured and within range 0-f (hexadecimal)
"0" is captured and within range 0-f (hexadecimal)

EXAMPLE 3
For this example the query is defined as an "AAAA" record for "pool-A-ff-aa.example.com." with an origin of "example.com." and the

evaluating BULK RR as:

```
-.example.com. 86400 IN BULK AAAA (
                                    pool-A-[0-ffff]-[0-ffff]
                                    fc00::${1}:${2}
                                   )
```

STEP 1
Ensure "Domain Name Pattern" is Fully Qualified

pool-A-[0-ffff]-[0-ffff] == pool-A-[0-ffff]-[0-ffff].example.com.

STEP 2
Determine whether range is decimal or hexadecimal

Query type == "AAAA" so range is hexadecimal.

STEP 3
Build regular expression based on fully qualified domain name pattern.

```
pool-A-[0-ffff]-[0-ffff].example.com. ==
     /^pool-A-([0-9a-fA-F]{1,4})-([0-9a-fA-F]{1,4})\.example\.com\.$/
```

The above regular expression simply matches numeric ranges based on decimal or hexadecimal and length. Numeric range validation occurs in the next step.

STEP 4
Compare captured numbers and validate ranges

```
pool-A-ff-aa.example.com.
  =~ /^pool-A-([0-9a-fA-F]{1,4})-([0-9a-fA-F]{1,4})\.example\.com\.$/
```

"ff" is captured and within range 0-ffff (hexadecimal)
"aa" is captured and within range 0-ffff (hexadecimal)

3.4. Record Generation using the BULK "Replacement Pattern" field

Once it has been determined a query meets all criteria for a BULK record generation the below rules are followed to process captured numeric data and Replacement Pattern into RRs to apply to the answer-set.

3.4.1. Replacement Pattern Backreferences

Before a record may be generated data must be captured in the Domain Name Pattern comparison step above. Each provided numeric range is assigned to a temporary buffer to be used in this step. To make the

jobs' of zone administrators easier the order of these buffers will change based on the Match Type and owner so they will default to feel more natural or intuitive. Captured patterns and backreferences are in the same vein as regular expressions and are intended to feel "familiar". This is described in further detail (with examples) in the sections below.

3.4.1.1. Backreference Notation

BULK RRs use a dollar-sign "$" and curly braces "{" and "}" to enclose backreferences within the Replacement Pattern. The following rules are used to determine the final replacement string.

3.4.1.1.1. Simple numeric backreference replacement

The simplest form of backreference notation is its numeric form. In this form only the backreference number falls between the curly braces "{" and "}". An example is "${1}" which would be replaced by the value in the first capture position. Position is described in detail in a later section.

Numeric backreference replacement indices start with one "1" to maintain consistency with regular expression backreferences.

3.4.1.1.2. Star backreference replacement

The next form of backreference notation is its star (or asterisk "*") form. In this form only an asterisk falls between the curly braces "{" and "}". This form "${*}" would be replaced by all captured values in order of ascending position delimited by its default delimiter (described below). Position is described in detail in a later section.

3.4.1.1.3. Numeric range backreference replacement

The next form of backreference notation is the numeric range form. In this form a range of numbers falls between the curly braces "{" and "}". An example of this is "${1-4}" which would be replaced by all captured values within this range (1-4) in order of positions provided delimited its default delimiter (described below). To reverse the order of positions in this example one could simply reverse the upper and lower values to look like "${4-1}". Position is described in detail in a later section.

3.4.1.1.4. Numeric set backreference replacement

The next form of backreference notation is the numeric set form. In this form a set of numbers falls between the curly braces "{" and "}" separated by commas. An example of this is "${1,4}" which would be replaced by the first and fourth captured values in the order of

position provided delimited its default delimiter (described below). Position is described in detail in a later section.

This notation may be combined with the numeric range form allowing specific positions or position ranges to be used. Examples would be "${3,2,1,4-8}" and "${8-12,1-4}".

3.4.1.1.5. Backreference delimiter

The above sections reference a default delimiter. In an effort to provide an intuitive zone management experience the default delimiter will be based on the BULK RR's Match Type. For Match Type "A" the default delimiter SHALL be a period ".", for Match Type "AAAA" the default delimiter SHALL be a colon ":" and for Match Types "PTR" and "CNAME" the default delimiter SHALL be a hyphen "-". In any case the default delimiter MAY be overridden by including it in the backreference braces after the set selectors and a backreference field separator character, the pipe "|". An example would be "${*|-}" which would force a hyphen "-" delimiter. An empty or null delimiter is allowed by not specifying a delimiter character, for example "${*|}", which would simply concatenate all captured values in order of capture position. Position is described in detail in a later section.

3.4.1.1.6. Backreference delimiter interval

The default behavior of a backreference set is to combine each captured value specified with a delimiter between each. To allow captured backreferences to be delimited at another interval a third backreference field is provided. An example would be "${*|-|4}" which would concatenate all captured values but delimiting only every fourth value with hyphens "-". 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 null value MUST be interpreted as "1" or every captured value.

3.4.1.1.7. Backreference padding length

When generating BULK based records a common requirement is to convert from one numeric format to another, padding is among the most common of these. The fourth and final backreference field determines what width to pad to. An example would be "${*|||4}" which would set the width of all captured values to 4 by inserting leading zeros to fill the void. The default is empty or null which MUST be interpreted as NO modification. A width of zero "0" has a special interpretation referred to as "unpad" meaning all leading zeros MUST be removed. If a value is provided captured values longer than this width MUST be truncated to fit the specified width. In the case where a delimiter interval is provided 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 them to a width of 4 characters by inserting leading zeros where necessary.

3.4.1.1.8. Backreference Position

Great effort has gone into providing zone maintainers an intuitive syntax. As part of this effort, the captured values will reverse direction depending on several factors.

As a general rule of thumb, if it makes sense the numeric ranges are in reverse order from query to answer then they will be reversed. Otherwise they will be in the same order.

Take for example a simple reverse DNS lookup, from "10.2.3.4" to "pool-A-3-4.example.com.". Since DNS zones are arranged according to management authority the records appear reversed numerically. In this example "10.2.3.4" becomes "4.3.2.10.in-addr.arpa.". One would intuitively expect this reversal to be reversed so positional indices of captured values would increment toward the right of the Replacement Pattern. This expectation is especially important when using range based replacements.

Formally, the rules for position reversal are as follows:

Match Type RRs for "PTR" are reversed for zone owners ending in either ".in-addr.arpa." or "ip6.arpa.". All other Match Type RRs for "PTR" are forward.

Match Type RRs for "A" (Address), "AAAA" (IPv6 Address) and "CNAME" (Canonical Name) are forward.

3.4.1.1.9. Backreference Position Negation

To allow simple reversal of any backreference notation a single exclamation point character "!" MAY be used as the first character of a backreference set. Examples would be "${!*}" and "${!1-4,7}". In both of the examples the backreference positions SHALL be the exact mirror equivalent as those without the leading exclamation point "!". This can be very important if the BULK generated replacements have values in positions opposite to what is required or expected.

3.4.2. Replacement Pattern examples

This section provides examples of several BULK RR Replacement Patterns. Each example is intended to further understanding for implementors and DNS administrators alike.

EXAMPLE 1
For this example the query is defined as a PTR record for "10.2.3.4"

with an origin of "2.10.in-addr.arpa." and the evaluating BULK RR as:

- 86400 IN BULK PTR - pool-${*}.example.com.

This example contains several of the features described above.

First, the record owner is simply a single hyphen "-" denoting it is a "hidden wildcard" (wildcard for generated records but not for BULK).

Second, the Domain Name Pattern is also a single hyphen "-" denoting all queries matching the owner's wildcard pattern for the "PTR" Match Type are accepted and will be captured for use in the Replacement Pattern.

Third, the Replacement Pattern contains a single "star" backreference denoting all captured numeric (decimal) backreferences will be combined with its default delimiter of hyphen "-" (for PTR) and placed into the backreference's position in the answer-set. Should this generate an invalid hostname the response will be NXDOMAIN unless other BULK records match and are successfully generated without error.

The owner for "10.2.3.4" is "4.3.2.10.in-addr.arpa." and creates matching backreferences for "4", "3", "2" and "10" then reverses their indices so "${1}" resolves to "10", "${2}" to "2", "${3}" to "3" and "${4}" to "4" respectively. When applied to the Replacement Pattern the answer becomes "pool-10-2-3-4.example.com.".

EXAMPLE 2
For this example the query is defined as a PTR record for "10.2.3.4" with an origin of "2.10.in-addr.arpa." and the evaluating BULK RR as:

- 86400 IN BULK PTR - pool-${*|||3}.example.com.

This example expands on EXAMPLE 1 with the differences outlined below.

The only change to the BULK RR is the Replacement Pattern includes additional fields, specifically null values for delimiter and interval and a padding width of 3.

The owner for "10.2.3.4" is "4.3.2.10.in-addr.arpa." and creates matching backreferences for "4", "3", "2" and "10" and reverses their indices so "${1}" resolves to "10", "${2}" to "2", "${3}" to "3" and "${4}" to "4" respectively. When applied to the Replacement Pattern the answer becomes "pool-010002003004.example.com.".

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.
0-3 86400 IN      NS    ns1.sub.example.com.
-   86400 IN BULK CNAME [0-255].[0-3] ${*|.}.0-3
```

For this example, the query would come in for "25.2.2.10.in-addr.arpa.". After matching the owner filter (ending in ".2.10.in-addr.arpa.") and the fully qualified domain name pattern of "[0-255].[0-3].2.10.in-addr.arpa." the answer-set would include a generated RR consisting of captured values "25" and "2" joined by the custom delimiter of period "." then joined by ".0-3" and made fully qualified. The resulting RR would be a "CNAME" with RDATA of "25.2.0-3.2.10.in-addr.arpa.". This record is now one delegated to "ns1.sub.example.com." as its authority and the answer-set is complete.

4. The NPN Resource Record

The NPN resource record provides pre-processing directives for Numeric Pattern Normalization (NPN) based RR signature generation.

The Type value for the NPN RR type is XX.

The NPN RR is class independent.

The NPN RR has no special TTL requirements.

4.1. NPN RDATA Wire Format

The RDATA for a NPN RR consists of a 2 octet Match Type field, a 1 octet Flags field, a 1 octet Owner Ignore field, a 1 octet Left Ignore field and a 1 octet Right Ignore field.

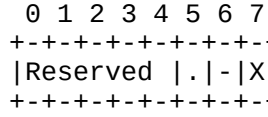
```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Match Type          |     Flags     |  Owner Ignore |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Left Ignore  |  Right Ignore |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.1.1. The Match Type field

The Match Type field identifies the type of the RRset identified by this NPN record.

4.1.2. The Flags field

The Flags field defines additional processing parameters for data normalization. This document defines only the Period-As-Number flag "." (position 5), the Hyphen-As-Number "-" (position 6) and the hexadecimal flag "X" (position 7). All other flags are reserved for future use.



Bits 0-4: Reserved for future
These flags have no default value if set to false (0).

Bit 5: Period As Number (.) Flag
This flag indicates the period (dot) will be processed as a number. This flag has no default value if set to false (0).

Bit 6: Hyphen As Number (-) Flag
This flag indicates the hyphen (dash) will be processed as a number. This flag has no default value if set to false (0).

Bit 7: Hexadecimal (X) Flag
This flag indicates the highest value for Normalization Processing is "f". Normalization Processing will be described in a later section. This flag has a default value of "9" if set to false (0).

4.1.3. The Owner Ignore field

The Owner Ignore field defines the length of characters as counted from the left-hand side of the owner which MUST be ignored by the normalization process. This field offers additional security to pattern based signatures which may not be immediately apparent. By restricting the leftmost characters defined by this value, ultimately the length of the generated portion of the accompanying BULK RR will be confined accordingly. Normalization Processing will be described further in a later section.

4.1.4. The Left Ignore field

The Left Ignore field defines the length of characters as counted from the left-hand side of the generated RDATA which MUST be ignored by the normalization process. Normalization Processing will be described further in a later section.

4.1.5. The Right Ignore field

The Right Ignore field defines the length of characters as counted from the right-hand side of the generated RDATA which MUST be ignored by the normalization process. Normalization Processing will be

described further in a later section.

4.2. The NPN RR Presentation Format

The Match Type field is represented as an RR type mnemonic. When the mnemonic is not known, the TYPE representation as described in [RFC3597], Section 5, MUST be used.

The Flags field MUST be presented as a string of characters representing each flag bit. This document defines only the period ".", hyphen "-" and hexadecimal "X" flags. Flags MAY appear in any order. For example; all three flags could appear as "-9." or ".f-" (without the quotes). If all bits are zero all default values (if defined) would be presented ("9" as currently defined).

All Ignore fields MUST be presented as an unsigned decimal integers and fall within the 0-255 range available to a single octet.

4.3. Normalization Processing of NPN RRs

This document provides a minor yet significant modification to DNSSEC regarding how RRsets will be signed or verified. Specifically the Signature Field of [RFC4034], Section 3.1.8. Prior to processing into canonical form, signed zones may contain associated RRs where; owner, class and type of a non NPN RR directly corresponds with an NPN RR matching owner, class and Match Type. If this condition exists the NPN RR’s RDATA defines details for processing the associated RDATA into a "Normalized" format. Normalized data is based on pre-canonical formatting and zero padded for "A" and "AAAA" RR types for acceptable precision during the process. This concept will become clearer in the NPN pseudocode and examples provided in the sections to follow.

The rules for this transformation are simple:

For RR’s Owner field, characters from the beginning to the index of the Owner Ignore value or the final string of characters belonging to the zone’s ORIGIN MUST NOT be modified by this algorithm. While the Owner Ignore value is not used for BULK records but is included with the expectation other pattern-based resource records may emerge and leverage NPN records for their DNSSEC support requirements.

For RR’s RDATA field, character from beginning to the index of Left Ignore value or characters with index of Right Ignore value to the end MUST NOT be modified by this algorithm.

In the remaining portion of both Owner and RDATA strings of numeric data, defined as character "0" through "f" or "0" through "9" depending on whether or not the Hexadecimal flag is set or

not, MUST be consolidated to a single character and set to the highest value defined by the Hexadecimal flag. Examples may be found in the following section. If period-as-number or hyphen-as-number flags are set whichever are used ("." or "-") would be treated as part of the number and consolidated where appropriate.

Once the normalization has been performed the signature will continue processing into canonical form using the normalized RRs in the place of original ones.

One thing to keep in mind when calculating values for the Ignore fields is the Domain Name Pattern and Replacement Pattern fields are considered relative unless terminated by a period. When processing NPN records the fully-qualified Patterns will be used for determining which characters should be ignored.

NPN RRs MAY be included in the "Additional" section to provide a hint for NPN processing required for verification path.

It is important to note, properly sizing the Ignore fields is critical to minimizing the risk of spoofed signatures. Never intentionally set all Ignore values to zero in order to make validation easier as it places the validity of zone data at risk. Only accompany RRs which are pattern derived (such as BULK) with NPN records as doing so may unnecessarily reduce the confidence level of generated signatures.

4.3.1. Pseudocode for NPN Normalization Processing

This section provides a simple demonstration of process flow for NPN rdata normalization and DNSSEC signatures.

The pseudocode provided below assumes all associated RRs are valid members of a DNSSEC compatible RRset (including BULK generated ones).

```
for rr in rrset
    if (has_NPN<rr.owner, rr.class, rr.type>)
        rr.rdata_normal = NPN_normalize<rr.rdata>
        add_to_sigrrset<NPN.owner, rr.class, rr.type,
            rr.rdata_normal>
        next
    else
        add_to_sigrrset<rr.owner, rr.class, rr.type, rr.rdata>
        next

process_canonical_form<sigrrset>

dnssec_sign<sigrrset>
```

Similar logic MUST be used for determining DNSSEC validity of RRsets

in verification (validation) nameservers for signatures generated based on NPN normalization.

4.3.2. NPN Normalization Processing examples

EXAMPLE 1
For this example the query is defined as a PTR record for "10.2.3.44" with an origin of "2.10.in-addr.arpa." and the evaluating BULK and NPN RR as:

```
-.2.10.in-addr.arpa. 86400 IN BULK PTR (
                                    [0-255].[0-10]
                                    pool-A-${1}-${2}.example.com.
                                    )
*.2.10.in-addr.arpa. 86400 IN NPN  PTR 9 0 7 13
```

As shown previously in BULK RR examples the query would enter the nameserver with an owner of "44.3.2.10.in-addr.arpa." and a "PTR" RR with the RDATA of "pool-A-3-44.example.com." would be generated.

By protecting the "Ignore" characters from the generated RR's RDATA the focus for normalization becomes "3-44" as illustrated below.

```
                          1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
                    v---------
        p o o l - A - 3 - 4 4 . e x a m p l e . c o m .
                     ---------^
    2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1
    6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

Everything to the left of "3-44" will remain intact as will everything to its right. The remaining characters will be processed for numbers between "0" and "9" as indicated by the NPN record's hexadecimal flag "9" and each run replaced by the single character "9". The final Normalized RDATA would therefore become "pool-A-9-9.example.com." and its signature would be based on this "normalized" RDATA field. This new "normalized" string would be used as an RDATA for the wildcard label of "*.2.10.in-addr.arpa." now encompassing all possible permutations of the "pool-A-${1}-${2}.example.com." pattern.

Since the verification (validation) nameserver would use the identical NPN record for processing and comparison, all RRs generated by the BULK record can now be verified with a single wildcard signature.

EXAMPLE 2
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.
0-3 86400 IN      NS    ns1.sub.example.com.
-   86400 IN BULK CNAME [0-255].[0-3] ${*|.}.0-3
*   86400 IN NPN  CNAME 9 0 0 23
```

For this example, a query of "10.2.2.65" would enter the nameserver as "65.2.2.10.in-addr.arpa." and a "CNAME" RR with the RDATA of "65.2.0-3.2.10.in-addr.arpa." would be generated.

By protecting the "Ignore" characters from the generated RR's RDATA the focus for normalization becomes "65.2" as illustrated below.

```
                         1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
       v---------
         6 5 . 2 . 0 - 3 . 2 . 1 0 . i n - a d d r . a r p a .
        ---------^
     2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
     9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

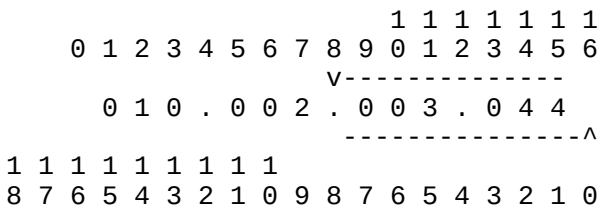
Everything to the left of "65.2" will remain intact as will everything to its right. The remaining characters will be processed for numbers between "0" and "9" as indicated by the NPN record's hexadecimal flag "9" and each run replaced by the single character "9". The final Normalized RDATA would therefore become "9.9.0-3.2.10.in-addr.arpa." and its signature would be based on this "normalized" RDATA field. This new "normalized" string would be used as an RDATA for the wildcard label of "*.2.10.in-addr.arpa." now encompassing all possible permutations of the "${*|.}.0-3.2.10.in-addr.arpa." pattern.

As in example 1, the verification (validation) nameserver would use the same NPN record for comparison.

EXAMPLE 3
This example provides reverse logic for example 1 by providing an IPv4 "A" record for a requested hostname. For this example the query is defined as an "A" record for "pool-A-3-44.example.com." with an origin of "example.com.". RRs for this example are defined as:

```
-.example.com. 86400 IN BULK A (
                                    pool-A-[0-10]-[0-255]
                                    10.2.${*}
                                   )
*.example.com. 86400 IN NPN  A 9 0 8 0
```

By protecting the "Ignore" characters from the generated RR's RDATA the focus for normalization becomes "003.044" as illustrated below.



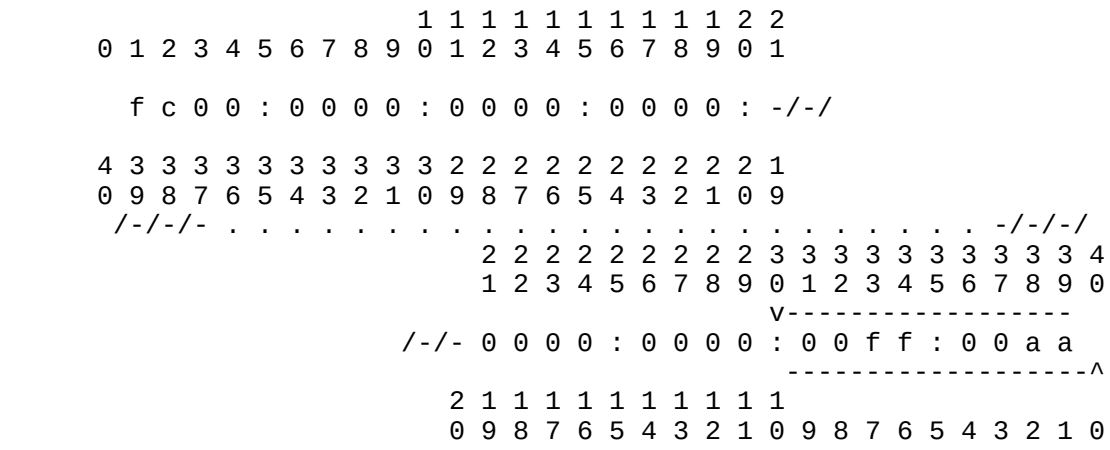
This example illustrates a key point about NPN records; since they are pre-canonical they MUST operate on a strict subset of WIRE formatted data. For "A" and "AAAA" records this means the "Ignore" fields are based on zero padded data. In this example our generated record MUST be converted into "010.002.003.044" (shown above) prior to processing. After processing, wire format would become "0x0A02032C" (shown in hexadecimal). This format would be too imprecise for normalization so padded decimal is used.

Everything to the left of "003.044" will remain intact as will everything to its right. The remaining characters will be processed for numbers between "0" and "9" as indicated by the NPN record's hexadecimal flag "9" and each run replaced by the single character "9". The final Normalized RDATA would therefore become "10.2.9.9" and its signature would be based on this "normalized" RDATA field. This new "normalized" "A" RR would be used as an RDATA for the wildcard label of "*.example.com." now encompassing all possible permutations of the "10.2.${*}" pattern.

EXAMPLE 4
This example provides similar logic for an IPv6 AAAA record. For this example the query is defined as an "AAAA" record for "pool-A-ff-aa.example.com." with an origin of "example.com.". RRs for this example are defined as:

```
-.example.com. 86400 IN BULK AAAA (
                                  pool-A-[0-ffff]-[0-ffff]
                                  fc00::${1}:${2}
                                 )
*.example.com. 86400 IN NPN  AAAA X 0 30 0
```

By protecting the "Ignore" characters from the generated RR's RDATA the focus for normalization becomes "00ff:00aa" as illustrated below.



This example reinforces the point on pre-canonical processing of NPN records; they MUST operate on a strict subset of WIRE formatted data. For "A" and "AAAA" records this means the "Ignore" fields are based on zero padded data. In this example our generated record MUST be converted into "fc00:0000:0000:0000:0000:0000:00ff:00aa" (shown above) prior to processing. After processing, wire format would become "0xFC000000000000000000000000FF00AA" (shown in hexadecimal). This format is slightly misleading as it is truly only 16 bytes of WIRE data and would be too imprecise for normalization so padded hexadecimal is used.

Everything to the left of "00ff:00aa" will remain intact as will everything to its right. The remaining characters will be processed for numbers between "0" and "f" as indicated by the NPN record's hexadecimal flag "X" and each run replaced by the single character "f". The final Normalized RDATA would therefore become "fc00::f:f" and its signature would be based on this "normalized" RDATA field. This new "normalized" "AAAA" RR would be used as an RDATA for the wildcard label of "*.example.com." now encompassing all possible permutations of the "fc00::${1}:${2}" pattern.

5. Positive Side-Effects

This section highlights positive side effects of some architectural decisions regarding the BULK RR design.

5.1. Record Superimposition

The main side-effect of the BULK RR design is superimposition. RRs created by the BULK generation process generally rely on the logic of wildcard assignment. This logic only provides answers where no others exist. This means in the reverse DNS world (network assignment) HUGE blocks of addresses can be assigned a single BULK record and where delegated to another customer or SWIP will be

automatically overridden.

When compared with bind's $GENERATE statement, if a singleton record such as CNAME appears within a $GENERATE range, either the CNAME or $GENERATE becomes invalid. While a BULK record range would automatically notch out the CNAME without user intervention or creating a potential management problem for the future when two $GENERATES create a hole where the CNAME no longer exists. BULK RRs would again automatically reassign the missing record to one of its own.

5.2. Pattern Based DNSSEC support

The NPN resource record can be used to support other dynamic RR types which do not currently exist.

6. Known Limitations

This section defines known limitations of the BULK resource type.

6.1. Increased CPU utilization for NXDOMAIN RRs

Nameserver requirements to support BULK RRs will minimally increase CPU utilization requirements compared to most RR types. However, since the inception of DNSSEC more is expected of DNS servers at a system resource level and it is the authors' belief the benefit outweighs the sacrifice.

A quick comparison of BULK versus bind's $GENERATE expansion reveals much more memory would be sacrificed with $GENERATES to save the CPU cycles required to support BULK records. Additionally, $GENERATES cannot be transferred (i.e. AXFR) without expansion and an IPv6 CIDR even as small as /96 would be simply impossible. BULK on the other hand can easily support IPv6 CIDRs of /64 and much larger with very little effort.

6.2. Pre-Adoption Nameserver Implications

While there is an added demand on authoritative nameservers, there are no new requirements to recursive (caching) resolvers for non-DNSSEC record handling. Even authoritative nameservers are able to transfer to and from supporting nameservers with no requirement (although would be unable to return BULK generated records without support).

Prior to widespread adoption on the authoritative side all generated records would be invisible if served on nameservers lacking support. Since generated records are generally NOT service impacting records this should be understood but not of great concern.

Once adoption has reached an appreciable level on the producer (authoritative) side only DNSSEC requirements remain for the consumer (resolver) side. This behavior is fully expected.

7. Security Considerations

Two known security considerations exist for the BULK resource record, DNSSEC and DDOS attack vectors. Both are addressed in the following sections.

7.1. DNSSEC Signature Strategies

DNSSEC was designed to provide verification (validation) for DNS resource records. In a nutshell this requires each (owner, class, type) tuple 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.

7.1.1. On-the-fly (Live) Signatures

This solution requires authoritative nameservers to sign generated records _as_they_are_generated_. Not all authoritative nameserver implementations offer on-the-fly (realtime) signatures so 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.

No changes to recursive (resolving) nameservers is required to support this solution.

7.1.2. Normalized (NPN Based) Signatures

This solution provides the most flexible solution as nameservers without on-the-fly signing capabilities can still support signatures for BULK records. The down side to this solution is it requires DNSSEC aware recursive (resolving) nameserver support. Unless a recursive nameserver can verify the signature it is _unverifiable_.

It has been pointed out due to this limitation creation of DNSSEC signed BULK RRs requiring NPN support SHOULD be formally discouraged until such time a respectable percentage (>80) of DNSSEC verification (validation) nameservers "in-the-wild" possess NPN processing capabilities. Until that time, on-the-fly signing and unsigned BULK records offer the intended capabilities of this document while requiring zero new features to support RR resolution. The authors would like to encourage opening this door for pattern based

technologies such as NPN records as a solution to BULK RRs as well as other pattern based RRs to come. Given enough time, enough nameservers will be patched and upgraded for unrelated reasons and by means of simple attrition can supply a level of "inertia" and eventually widespread adoption can be assumed.

NPN records are likely to be a topic of great debate as to their own security limitations. It is, however, the authors' belief; while any logic which limits the input of digital signatures, lessens the validity of such signatures, the limitation is minimal and the gain is significant. The main reason for this is as a general rule, RRs used in a generic manner such as conventional $GENERATE RRs or scripted mass pattern generated RRs have a lesser importance than other RRs in managed zones. These therefore inherently pose less risk by means of attack and have a much less reward by defeating security measures.

This being said, care must still be taken to set the Ignore fields appropriately to minimize exposure and only use NPN RRs to secure pattern-based records such as BULK.

7.1.3. 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 (resolving) nameservers.

7.2. DNSSEC Verifier Details

Verification of DNSSEC signed BULK generated RRs may be performed against on-the-fly signatures with zero modification to their behavior. However, verification against NPN records would require changes to the logic to incorporate processing RDATA generated by BULK logic as described above so the results will be compatible.

7.3. 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) in its examples values between 300 to 900 are likely more appropriate and is RECOMMENDED. What is ultimately deemed appropriate may differ from zone to zone and administrator to administrator.

7.4. 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, implementors 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.

8. IANA Considerations

IANA is requested to assign numbers for two DNS resource record types identified in this document; BULK and NPN.

9. 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.)

10. References

10.1. Normative References

[US-ASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, March 1998.

[RFC2317] Eidnes, H., de Groot, G., Vixie, P. "Classless IN-ADDR.ARPA delegation", RFC 2317, March 1998.

[RFC2536] Eastlake 3rd, D., "DSA KEYs and SIGs in the Domain Name System (DNS)", RFC 2536, March 1999.

[RFC2931] Eastlake 3rd, D., "DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, September 2000.

[RFC3110] Eastlake 3rd, D., "RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS)", RFC 3110, May 2001.

[RFC3597] A. Gustafsson, "Handling of Unknown DNS Resource Record (RR) Types", RFC 3597, September 2003.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

[RFC5234] D. Crocker, Ed., P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, January 2008.

10.2. Informative References

Authors' Addresses

John Woodworth
4250 North Fairfax Drive
Arlington, VA 22203
USA

EMail: John.Woodworth@CenturyLink.com

Dean Ballew
2355 Dulles Corner Boulevard Suite 200 300
Herndon, VA 20171
USA

EMail: Dean.Ballew@CenturyLink.com

Shashwath Bindinganaveli Raghavan
2355 Dulles Corner Boulevard Suite 200 300
Herndon, VA 20171
USA

EMail: Shashwath.Bindinganaveliraghavan@CenturyLink.com

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

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