Network Working Group
Internet-Draft
Updates: 6891 (if approved)
Intended status: Standards Track
Expires: April 27, 2015

R. Bellis
Nominet UK

October 24, 2014

Connection Close Signalling for DNS
draft-bellis-dnsop-connection-close-00

Abstract

This document updates [RFC6891] by specifying a new single-bit flag in a DNS response that when seen in a packet carried over a connection-orientated transport protocol indicates to the client that it should close the current connection.

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 27, 2015.

Copyright Notice

Table of Contents

1. Introduction

The DNS protocol [RFC1035] supports use of persistent TCP connections, although guidance as to when a connection should be terminated (and by which party) is limited [RFC5966].

This document updates the Extension Mechanisms for DNS (EDNS(0)) [RFC6891] by specifying a new single-bit flag in a DNS response that when seen in a packet carried over a connection-orientated transport protocol indicates to the client that it should close the current connection.

Having the client close the connection reduces the amount of TCP state information that must be stored by the server compared to that resulting from the server initiating a unilateral close itself.

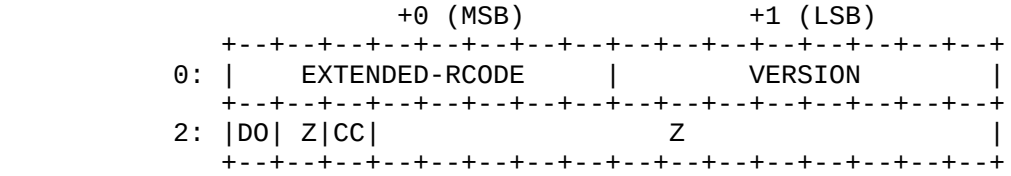
TODO: does it make sense to specify a request side meaning for this flag, indicating that the server may half-close its "read" side of the connection? This would make the semantics even closer to those of the HTTP/1.1 "Connection: close" header (see Section 14.10 of [RFC2616])

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].

3. Specification

The "Connection Close" (CC) bit is held in the third-most signifiant bit of the third byte of the "extended RCODE and flags" portion of an EDNS(0) OPT meta-RR:



Note to RFC editor: replace the first ’Z’ in the figure above with ’TO’ if draft-hzhwm-dprive-start-tls-for-dns is published as an RFC before this specification.

4. Connection Handling

4.1. Servers

Servers MAY set this flag to indicate that further queries received over the current connection should not be sent.

An incompatible client will not understand this flag and may continue sending requests and therefore the server MUST NOT refuse to service subsequent requests. The server MAY unilaterally close idle connections regardless, per [RFC5966] and Section 4.2.2 of [RFC1035]

Since this flag requires EDNS(0) support, note that this flag cannot be set unless the client has indicated support for EDNS(0) by sending an OPT meta-RR itself, per Section 7 of [RFC6891]

TODO: note - the constraint in RFC 6891 appears unnecessarily strict - it appears to mandate that the EDNS(0) support indication is on a per-request basis, but it would be reasonable on a connection-orientated transport to assume that ANY preceding request on that connection with an OPT RR is sufficient to indicate that the client supports EDNS(0).

TODO: if a request-side semantic is defined for this flag, what are the TCP state-maintenance implications if the server performs a ’shutdown(fd, SHUT_RD)’?

4.2. Clients

Clients receiving a packet with this flag set MUST NOT send any further queries over the current connection and MUST initiate closure of that connection.

TODO: what are the TCP state-maintenance implications if the client performs a 'shutdown(fd, SHUT_WR)'?

5. Security Considerations

None identified (yet).

6. IANA Considerations

IANA are requested to update the EDNS Header Flag Registry according to Section 3.

Note to IANA and RFC Editor: The actual bit assigned will depend on whether any other document specifies a used for the above-specificed bit in advance of publication of this document as an RFC.

7. References

7.1. Normative References

[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.

[RFC5966] Bellis, R., "DNS Transport over TCP - Implementation Requirements", RFC 5966, August 2010.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, April 2013.

7.2. Informative References

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

Appendix A. Change Log

Note to RFC editor: remove this section before publication.

draft-bellis-dnsop-connection-close-00

Initial draft

Author's Address

Ray Bellis
Nominet UK
Minerva House
Edmund Halley Road
Oxford Science Park
Oxford OX4 4DQ
United Kingdom

Phone: +44 1865 332211
Email: ray.bellis@nominet.org.uk
URI: http://www.nominet.org.uk/

Network Working Group
Internet-Draft
Updates: 5966 (if approved)
Intended status: Standards Track
Expires: April 29, 2015

J. Dickinson
Sinodun Internet Technologies
R. Bellis
Nominet
A. Mankin
D. Wessels
Verisign Labs
October 26, 2014

DNS Transport over TCP - Implementation Requirements
draft-dickinson-dnsop-5966-bis-00

Abstract

This document updates the requirements for the support of TCP as a transport protocol for DNS implementations.

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 29, 2015.

Copyright Notice

Table of Contents

1. Introduction

Most DNS [RFC1034] transactions take place over UDP [RFC0768]. TCP [RFC0793] is always used for full zone transfers (AXFR) and is often used for messages whose sizes exceed the DNS protocol's original 512-byte limit.

Section 6.1.3.2 of [RFC1123] states:

> DNS resolvers and recursive servers MUST support UDP, and SHOULD support TCP, for sending (non-zone-transfer) queries.

However, some implementors have taken the text quoted above to mean that TCP support is an optional feature of the DNS protocol.

The majority of DNS server operators already support TCP and the default configuration for most software implementations is to support TCP. The primary audience for this document is those implementors whose failure to support TCP restricts interoperability and limits deployment of new DNS features.

This document therefore updates the core DNS protocol specifications such that support for TCP is henceforth a REQUIRED part of a full DNS protocol implementation.

There are several advantages and disadvantages to the increased use of TCP as well as implementation details that need to be considered. This document addresses these issues and updates [RFC5966], with additional considerations and lessons learned from new research and implementations [Connection-Oriented-DNS].

Whilst this document makes no specific requirements for operators of DNS servers to meet, it does offer some suggestions to operators to help ensure that support for TCP on their servers and network is optimal. It should be noted that failure to support TCP (or the blocking of DNS over TCP at the network layer) may result in resolution failure and/or application-level timeouts.

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].

3. Discussion

In the absence of EDNS0 (Extension Mechanisms for DNS 0) (see below), the normal behaviour of any DNS server needing to send a UDP response that would exceed the 512-byte limit is for the server to truncate the response so that it fits within that limit and then set the TC flag in the response header. When the client receives such a response, it takes the TC flag as an indication that it should retry over TCP instead.

RFC 1123 also says:

> ... 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. Thus, resolvers and name servers should implement TCP services as a backup to UDP today, with the knowledge that they will require the TCP service in the future.

Existing deployments of DNS Security (DNSSEC) [RFC4033] have shown that truncation at the 512-byte boundary is now commonplace. For example, a Non-Existent Domain (NXDOMAIN) (RCODE == 3) response from a DNSSEC-signed zone using NextSECure 3 (NSEC3) [RFC5155] is almost invariably larger than 512 bytes.

Since the original core specifications for DNS were written, the Extension Mechanisms for DNS (EDNS0 [RFC6891]) have been introduced. These extensions can be used to indicate that the client is prepared

to receive UDP responses larger than 512 bytes. An EDNS0-compatible server receiving a request from an EDNS0-compatible client may send UDP packets up to that client's announced buffer size without truncation.

However, transport of UDP packets that exceed the size of the path MTU causes IP packet fragmentation, which has been found to be unreliable in some circumstances. Many firewalls routinely block fragmented IP packets, and some do not implement the algorithms necessary to reassemble fragmented packets. Worse still, some network devices deliberately refuse to handle DNS packets containing EDNS0 options. Other issues relating to UDP transport and packet size are discussed in [RFC5625].

The MTU most commonly found in the core of the Internet is around 1500 bytes, and even that limit is routinely exceeded by DNSSEC-signed responses.

The future that was anticipated in RFC 1123 has arrived, and the only standardised UDP-based mechanism that may have resolved the packet size issue has been found inadequate.

4. Transport Protocol Selection

All general-purpose DNS implementations MUST support both UDP and TCP transport.

- Authoritative server implementations MUST support TCP so that they do not limit the size of responses to what fits in a single UDP packet.
- Recursive server (or forwarder) implementations MUST support TCP so that they do not prevent large responses from a TCP-capable server from reaching its TCP-capable clients.
- Stub resolver implementations (e.g., an operating system's DNS resolution library) MUST support TCP since to do otherwise would limit their interoperability with their own clients and with upstream servers.

Regarding the choice of when to use UDP or TCP, Section 6.1.3.2 of RFC 1123 also says:

> ... a DNS resolver or server that is sending a non-zone-transfer query MUST send a UDP query first.

This requirement is hereby relaxed. A resolver MAY elect to send either TCP or UDP queries depending on local operational reasons. If it already has an open TCP connection to the server it SHOULD reuse this connection.

In essence, TCP SHOULD be considered as valid a transport as UDP. It SHOULD NOT be used only for zone transfers and as a fallback.

In addition it is noted that all Recursive and Authoritative servers MUST send responses using the same transport as the query arrived on. In the case of TCP this MUST also be the same connection.

5. Connection Handling

One perceived disadvantage to DNS over TCP is the added connection setup latency, generally equal to one RTT. To amortize connection setup costs, both clients and servers SHOULD support connection reuse by sending multiple queries and responses over a single TCP connection.

DNS currently has no connection signaling mechanism. Clients and servers may close a connection at any time. Clients MUST be prepared to retry failed queries on broken connections.

Section 4.2.2 of [RFC1035] says:

> If the server needs to close a dormant connection to reclaim resources, it should wait until the connection has been idle for a period on the order of two minutes. In particular, the server should allow the SOA and AXFR request sequence (which begins a refresh operation) to be made on a single connection. Since the server would be unable to answer queries anyway, a unilateral close or reset may be used instead of a graceful close.

Other more modern protocols (e.g., HTTP/1.1 [RFC7230]) have support for persistent TCP connections and operational experience has shown that long timeouts can easily cause resource exhaustion and poor response under heavy load. Intentionally opening many connections and leaving them dormant can trivially create a "denial-of-service" attack.

It is therefore RECOMMENDED that the default application-level idle period should be of the order of seconds, but no particular value is specified. In practice, the idle period may vary dynamically, and servers MAY allow dormant connections to remain open for longer periods as resources permit.

To mitigate the risk of unintentional server overload, DNS clients MUST take care to minimize the number of concurrent TCP connections made to any individual server. Similarly, servers MAY impose limits on the number of concurrent TCP connections being handled for any particular client. It is RECOMMENDED that for any given client - server interaction there SHOULD be no more than one connection for regular queries, one for zone transfers and one for each protocol that is being used on top of TCP, for example, if the resolver was using TLS. The server MUST NOT enforce these rules for a particular client because it does not know if the client IP address belongs to a single client or is, for example, multiple clients behind NAT.

6. Query Pipelining

Due to the use of TCP primarily for zone transfer and truncated responses, no existing RFC discusses the idea of pipelining DNS queries over a TCP connection.

In order to achieve performance on par with UDP, it is therefore RECOMMENDED that DNS clients should pipeline their queries. When a DNS client sends multiple queries to a server, it should not wait for an outstanding reply before sending the next query. Clients should treat TCP and UDP equivalently when considering the time at which to send a particular query.

DNS servers (especially recursive) SHOULD expect to receive pipelined queries. The server should process TCP queries in parallel, just as it would for UDP. The handling of responses to pipelined queries is covered in the following section.

When pipelining queries over TCP it is very easy to send more DNS queries than there are DNS Message ID’s. Implementations MUST take care to check their list of outstanding DNS Message ID’s before sending a new query over an existing TCP connection. This is especially important if the server could be performing out-of-order processing. In addition, when sending multiple queries over TCP it is very easy for a name server to overwhelm its own network interface. Implementations MUST take care to manage buffer sizes or to throttle writes to the network interface.

7. Response Reordering

RFC 1035 is ambiguous on the question of whether TCP responses may be reordered -- the only relevant text is in Section 4.2.1, which relates to UDP:

> Queries or their responses may be reordered by the network, or by processing in name servers, so resolvers should not depend on them being returned in order.

For the avoidance of future doubt, this requirement is clarified. Authoritative servers and recursive resolvers are RECOMMENDED to support the sending of responses in parallel and/or out-of-order, regardless of the transport protocol in use. Stub and recursive resolvers MUST be able to process responses that arrive in a different order to that in which the requests were sent, regardless of the transport protocol in use.

In order to achieve performance on par with UDP, recursive resolvers SHOULD process TCP queries in parallel and return individual responses as soon as they are available, possibly out-of-order.

Since responses may arrive out-of-order, clients must take care to match responses to outstanding queries, using the ID field, port number, query name/type/class, and any other relevant protocol features.

8. TCP Fast Open

This section is non-normative.

TCP fastopen [I-D.ietf-tcpm-fastopen] (TFO) allows data to be carried in the SYN packet. It also saves up to one RTT compared to standard TCP.

TFO mitigates the security vulnerabilities inherent in sending data in the SYN, especially on a system like DNS where amplification attacks are possible, by use of a server-supplied cookie. TFO clients request a server cookie in the initial SYN packet at the start of a new connection. The server returns a cookie in its SYN-ACK. The client caches the cookie and reuses it when opening subsequent connections to the same server.

The cookie is stored by the client’s TCP stack (kernel) and persists if either the client or server processes are restarted. TFO also falls back to a regular TCP handshake gracefully.

Adding support for this to existing name server implementations is relatively easy, but does require source code modifications. On the client, the call to connect() is replaced with a TFO aware version of sendmsg() or sendto(). On the server, TFO must be switched into server mode by changing the kernel parameter (net.ipv4.tcp_fastopen on Linux) to enable the server bit (Set the integer value to 2

(server only) or 3 (client and server)) and setting a socket option between the bind() and listen() calls.

DNS services taking advantage of IP anycast [RFC4786] may need to take additional steps when enabling TFO. From [I-D.ietf-tcpm-fastopen]:

> Servers that accept connection requests to the same server IP address should use the same key such that they generate identical Fast Open Cookies for a particular client IP address. Otherwise a client may get different cookies across connections; its Fast Open attempts would fall back to regular 3WHS.

9. Summary of Advantages and Disadvantages to using TCP for DNS

The TCP handshake generally prevents address spoofing and, therefore, the reflection/amplification attacks which plague UDP.

TCP does not suffer from UDP’s issues with fragmentation. Middleboxes are known to block IP fragments, leading to timeouts and forcing client implementations to "hunt" for EDNS0 reply size values supported by the network path. Additionally, fragmentation may lead to cache poisoning [fragmentation-considered-poisonous].

TCP setup costs an additional RTT compared to UDP queries. Setup costs can be amortized by reusing connections, pipelining queries, and enabling TCP Fast Open.

TCP imposes additional state-keeping requirements on clients and servers. The use of TCP Fast Open reduces the cost of closing and re-opening TCP connections.

Long-lived TCP connections to anycast servers may be disrupted due to routing changes. Clients utilizing TCP for DNS must always be prepared to re-establish connections or otherwise retry outstanding queries. It may also possible for TCP Multipath [RFC6824] to allow a server to hand a connection over from the anycast address to a unicast address.

There are many "Middleboxes" in use today that interfere with TCP over port 53 [RFC5625]. This document does not propose any solutions, other than to make it absolutely clear that TCP is a valid transport for DNS and must be supported by all implementations.

10. IANA Considerations

This memo includes no request to IANA.

11. Security Considerations

Some DNS server operators have expressed concern that wider use of DNS over TCP will expose them to a higher risk of denial-of-service (DoS) attacks.

Although there is a higher risk of such attacks against TCP-enabled servers, techniques for the mitigation of DoS attacks at the network level have improved substantially since DNS was first designed.

Readers are advised to familiarise themselves with [CPNI-TCP].

Operators of recursive servers should ensure that they only accept connections from expected clients, and do not accept them from unknown sources. In the case of UDP traffic, this will help protect against reflector attacks [RFC5358] and in the case of TCP traffic it will prevent an unknown client from exhausting the server's limits on the number of concurrent connections.

12. Acknowledgements

The authors would like to thank Liang Zhu, Zi Hu, and John Heidemann for extensive DNS-over-TCP discussions and code; and Lucie Guiraud and Danny McPherson for reviewing early versions of this document. We would also like to thank all those who contributed to RFC 5966.

13. References

13.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[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.

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

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

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

[RFC4786] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, December 2006.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, March 2008.

[RFC5358] Damas, J. and F. Neves, "Preventing Use of Recursive Nameservers in Reflector Attacks", BCP 140, RFC 5358, October 2008.

[RFC5625] Bellis, R., "DNS Proxy Implementation Guidelines", BCP 152, RFC 5625, August 2009.

[RFC5966] Bellis, R., "DNS Transport over TCP - Implementation Requirements", RFC 5966, August 2010.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, April 2013.

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

13.2. Informative References

[CPNI-TCP]
CPNI, "Security Assessment of the Transmission Control Protocol (TCP)", 2009, <http://www.cpni.gov.uk/Docs/tn-03-09-security-assessment-TCP.pdf>.

[Connection-Oriented-DNS]
Zhu, L., Hu, Z., Heidemann, J., Wessels, D., Mankin, A., and N. Somaiya, "T-DNS: Connection-Oriented DNS to Improve Privacy and Security (extended)", <http://www.isi.edu/publications/trpublic/files/tr-693.pdf>.

[I-D.ietf-tcpm-fastopen]
Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", draft-ietf-tcpm-fastopen-09 (work in progress), July 2014.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, January 2013.

[fragmentation-considered-poisonous] Herzberg, A. and H. Shulman, "Fragmentation Considered Poisonous", May 2012, <http://arxiv.org/abs/1205.4011>.

Appendix A. Changes to RFC 5966

This document differs from RFC 5966 in four additions:

1. DNS implementations are recommended not only to support TCP but to support it on an equal footing with UDP

2. DNS implementations are recommended to support reuse of TCP connections

3. DNS implementations are recommended to support pipelining and out of order processing of the query stream

4. A non-normative discussion of use of TCP Fast Open is added

Authors' Addresses

John Dickinson
Sinodun Internet Technologies
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA
UK

Email: jad@sinodun.com
URI: http://sinodun.com

Ray Bellis
Nominet
Edmund Halley Road
Oxford OX4 4DQ
UK

Phone: +44 1865 332211
Email: ray.bellis@nominet.org.uk
URI: http://www.nominet.org.uk/

Allison Mankin
Verisign Labs
12061 Bluemont Way
Reston, VA 20190

Phone: +1 703 948-3200
Email: amankin@verisign.com

Duane Wessels
Verisign Labs
12061 Bluemont Way
Reston, VA 20190

Phone: +1 703 948-3200
Email: dwessels@verisign.com

dnsop
Internet-Draft
Intended status: Standards Track
Expires: April 18, 2015

B. Dickson

October 15, 2014

System to transport DNS over HTTP using JSON
draft-dickson-dnsop-spartacus-system-00

Abstract

This is the SPARTACUS DNS gateway system. It is designed to facilitate the transport of DNS messages opaquely, across problematic sections of the Internet. It uses JSON encoding, and HTTP(S) as the protocol for transport.

The main criteria of SPARTACUS is that it preserve DNS messages verbatim, and that only properly formatted DNS messages are passed.

There are two modes (so far) defined: DNS forwarder (dns clients point to a local gateway, which forwards to a remote gateway for sending to a DNS resolver); and transparent proxy (DNS packets are intercepted, passed to a local gateway, which sends them to the remote gateway, with original destination IP address etc. encoded, and used by the remote gateway as the destination).

DNS messages are NAT-friendly, so changes to IP or UDP headers do not impact them. Thus, SPARTACUS does not interfere with TSIG, SIG(0), or Eastlake Cookies.

This document describes the system, the components, and behavior, with examples.

Author’s Note

Intended Status: Proposed Standard.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

DNS (The Domain Name System) has been deployed since the 1980's [RFC1033][RFC1034][RFC1035]. Since that time, some of the original Resource Record types have been made officially obsolete [RFC3425]. Some elements have been clarified [RFC2181][RFC2308]. New Resource Records have been added [RFC2136][RFC2845][RFC2930][RFC6891]. New definitions of bits in the header have arisen, in part due to DNSSEC's AD and CD bits [RFC4033][RFC4034][RFC4035][RFC5155].

This has resulted in now-outdated implementations of stateful devices (e.g. devices doing either NAT or packet inspection) interfering with end-to-end communication between DNS speakers. Old devices or implementations reject DNS packets that include these newer capabilities, features, or format changes.

At the same time, there has arisen a variety of other devices and systems whose deliberate function is to block, capture, or modify DNS traffic, for profit or for ideological reasons. Examples include hotel wifi systems, ISPs, and state actors.

Owing to the stateless nature of DNS over UDP, it is not possible to distinguish between deliberate and accidental sources of DNS interference.

1.1. Problem Statement

There is a need to provide ways of supporting incremental deployment of new DNS features, in such a way as to prevent deliberate and/or accidental interference in the communication between DNS speakers.

For example, DNS speakers could communicate over protected channels and with data integrity validation via DNSSEC. The foremost limitation is that the communication be over any other port/protocol combination than UDP port 53. Ideally, the choice should be an

encoding that is compatible with whatever port/protocol combination is selected (versus overloading the port/protocol with incompatible payloads).

There is a further need for the communications channel(s) to be standardized, and to not introduce further interoperability problems at the DNS protocol level. Independent implementations need to interoperate completely, to avoid merely pushing the compatibility problem around.

In order to solve these problems (individually and/or collectively), the SPARTACUS system has been developed.

1.2. Rationale

SPARTACUS (Secure, Private Aparatus for Resolution Transported Across Constraining and/or Unmaintained Systems), is a system for encoding and decoding DNS messages (the DNS payload of UDP or TCP packet streams).

The SPARTACUS system consists of bidirectional DNS gateways for transporting DNS over HTTP(S) using a JSON encoding scheme. This is intended to create "bridges" between DNS speakers; perhaps a better analogy would be "ferries", as there is no requirement for a tightly bound relationship between individual Client nodes and Server nodes.

Standardizing the JSON encoding used by SPARTACUS, is intended to ensure a greater likelihood of compatible, interoprable implementations.

The goal is to transport DNS messages from any Client implementation to any Server implementation.

Each gateway must be liberal in what it accepts (any valid DNS message conforming to the relevant RFCs, regardless of DNS implementation) and conservative in what it sends (all packets must parse correctly as DNS messages). In order to ensure forward compatibility, unknown Types and (in the case of OPT) sub-types, MUST be accepted and transported.

DNS messages MUST traverse the encode/decode process unaltered. The round-trip is designed to, and MUST be implemented to, preserve the entire DNS message’s fidelity. This means a 1:1 binary match between input, encoding, decoding, and output. The lengths MUST match, and messages MUST be identical, bit for bit.

A secondary objective of the encoding in JSON is the use of the same names for data elements and structures as in the DNS RFCs. The idea

is to provide human-readable JSON encodings, for easier diagnostics during development, and when investigating operational issues.

1.3. Related Work

A variety of other work exists, and provided inspiration for the SPARTACUS work. This includes web/JSON DNS portals, for providing DNS query responses in JSON format, often with a "looking glass" functionality. FIXME format this list appropriately and decorate with words. END FIXME

- Multi-location DNS Looking Glass - Tool for performing DNS queries via RESTful interface in multiple locations, returning results in JSON format
- DNS Looking Glass - Tool for performing DNS queries via RESTful interface, returning results in JSON format
- DNS JSON - Source code project from circa 2009, partially developed but incomplete/abandoned
- DNSSEC-trigger[trigger] - embedded control function in NLnetlabs' Unbound resolver, for attempting DNS queries over TCP port 80 when DNSSEC problems are encountered
- Various other web-based DNS lookup tools

1.3.1. Comparison

There has been at least one previous effort to develop code for a DNS-JSON encoding, which appears to have been abandoned after one-way encoding was done, circa 2009. The project focused on presenting results to DNS queries in JSON format, with an intention to create a client gateway, which never materialized. The project can be found in two places ([JPF_jsondns] and [jsondns.org]). One major difference is that DNS query response status is converted to HTTP error codes, rather than being embedded in the JSON answer. This makes it unsuitable for bidirectional use. Only a few DNS type codes were implemented.

Another DNS JSON tool [fileformat.info], similarly focuses only on answers, with a limited number of type codes.

Yet another tool for looking up DNS via HTTP with JSON responses is the "dnsrest" [restdns.net]. It too focuses only on answer values, and is similarly not able to fully produce results that can be turned back into DNS answer packets.

The "DNS Looking Glass" [bortzmeyer.org], is primarily designed for returning DNS answer data. As such, it lacks encoding suitable for a bidirectional scheme. It is primarily focused on XML output, with JSON output organized around DNS resolution meta-data, plus answer data in a generic schema. (The schema itself is described in [draft-bortzmeyer-dns-json].)

The "Multilocation DNS Looking Glass" [dns-lg.com], uses a RESTful query mechanism of "node/qname/qtypename" to request the looking glass (LG) to perform a DNS lookup for the qname and qtype, and returns the response in a JSON format. The JSON format is generic, encapsulating all types as string data in presentation format, with a generic label of "rdata". This does not facilitate decoding easily, as the JSON scheme provides no information for parsing the rdata field. The type (qtype for the query, or type for answer/authority/additional) is in string (symbolic) form, and the elements are objects and thus in unordered lists. The JSON scheme is fine for one-way encoding for human readability, but not suitable for two-way conversion back into DNS.

DNSSEC-trigger[trigger] can only be used in environments that use NLnetlabs' Unbound resolver, or where Unbound can be deployed as a replacement for existing recursive resolvers and/or stub resolvers.

A variety of other web lookup tools exist, predominantly producing DNS validation (zone structure and hierarchy), maps, meta-data, or literal output from the 'dig' tool, in formats as varied as the purposes of the tools. Dig output, while being reasonably deterministic, is not sufficiently well-formed as to facilitate "screen scraping" as a parsing method.

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].

3. System Overview

The SPARTACUS system is designed to improve the reliability and security of the DNS system, by providing the means to transport DNS traffic across segments of the Internet. The goal is to bypass problem areas which interfere with DNS communications, regardless of root cause of the interference.

Some familiarity with the DNS protocol is assumed.

3.1. System Elements

The particular system elements used will differ, based on the mode of operation of the Client. Clients may request the use of particular resolvers via additional intra-element signalling.

3.1.1. Node Types

Base node types are the following:

- Standalone SPARTACUS Client forwarder
- Transparent SPARTACUS proxy Client
- Standalone SPARTACUS Server
- Apache module-based SPARTACUS Server
- Stub resolver
- External recursive resolver
- Client-side recursive resolver
- External authority server

Future node types are expected to include:

- Browser-integrated SPARTACUS client and stub resolver
- Mobile-device SPARTACUS client and stub resolver (with exposed getdns API)
- SMTP-integrated SPARTACUS client and stub resolver

3.2. System Modes

The system has two modes of operation:

- DNS Forwarder - an opaque mode of operation, the Client/Server pair act collectively as a single DNS forwarder.
- Transparent Proxy - In this mode, regular DNS traffic is diverted by unspecified means to the SPARTACUS Client.

Additional intra-element signalling facilitates Clients requesting particular resolvers' (recursive or authoritative) use.

3.2.1. Details on DNS Forwarder mode

The Server is configured to use a particular DNS recursive resolver, with the optional ability to support Client-requested resolver(s) via in-band signaling. If present, the Client-requested resolver IP address is passed as an EDNS OPT value. The Server, if it is configured to honor requested resolvers, uses this IP address instead of the default.

Example: Problem caused by firewalls that do not support DNSSEC:

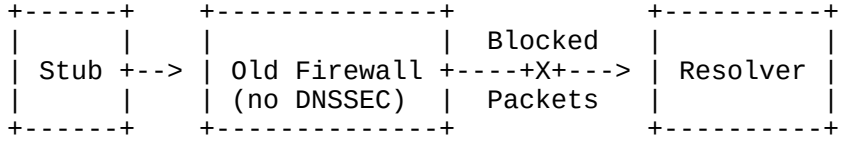


Figure 1

Example: How the stub client sees the SPARTACUS Client/Server pair, in the opaque forwarder configuration:

Figure 2

Example: How the Client/Server pair actually operates:

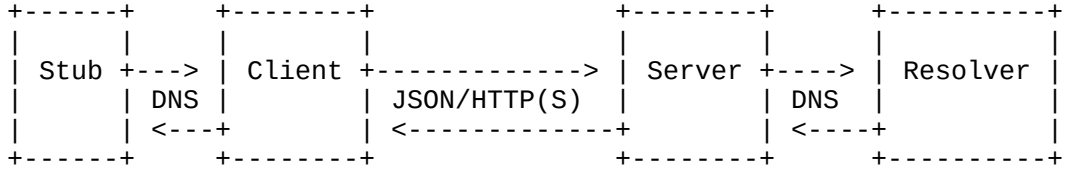


Figure 3

Example: How the Client/Server bypass the old firewall:

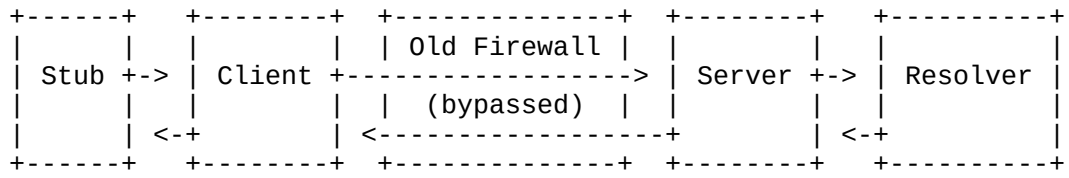


Figure 4

3.2.2. Details on Transparent Proxy mode

Transparent Proxy mode supports transport of stub to recursive traffic (all with the same destination IP address).

Transparent Proxy mode also supports use by a recursive resolver, to handle recursive-to-authoritative traffic (with different destination IP addresses per query).

From the perspective of the DNS client (stub or recursive), it appears that the DNS query packet went to some IP address, and the reply came back directly.

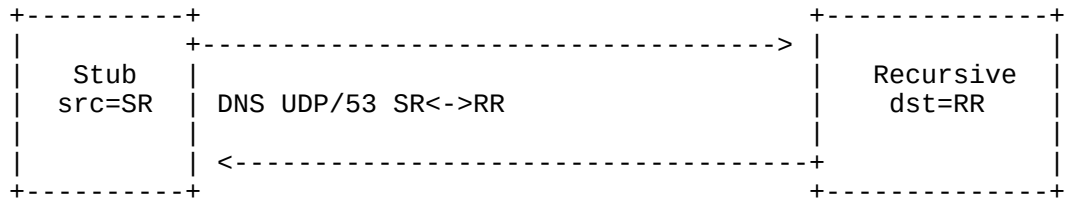


Figure 5

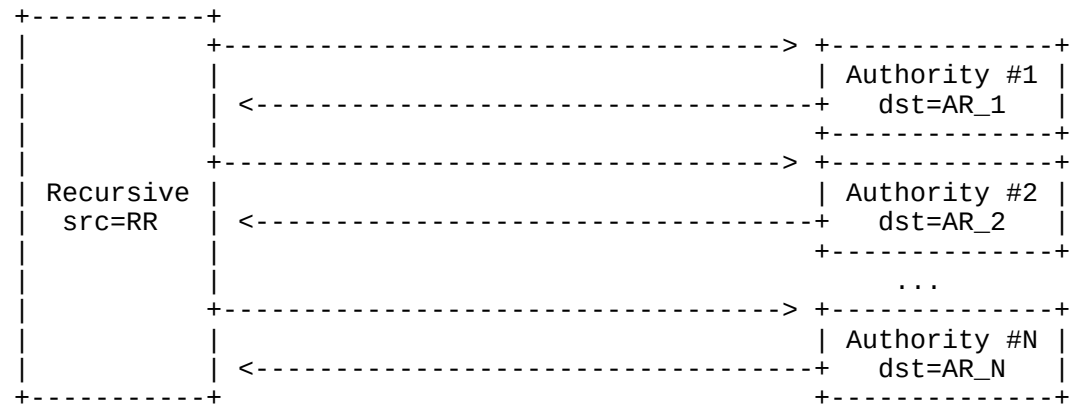


Figure 6

In both use cases, the original IP destination is encoded as an EDNS OPT value, and the DNS message (encoded as JSON) is sent to the SPARTACUS Server. The Server sends the DNS message to the original IP destination, with the SPARTACUS Server's IP address as the source. The resulting answer DNS message is sent to the Client, which changes the reply source IP address to the original destination IP address.

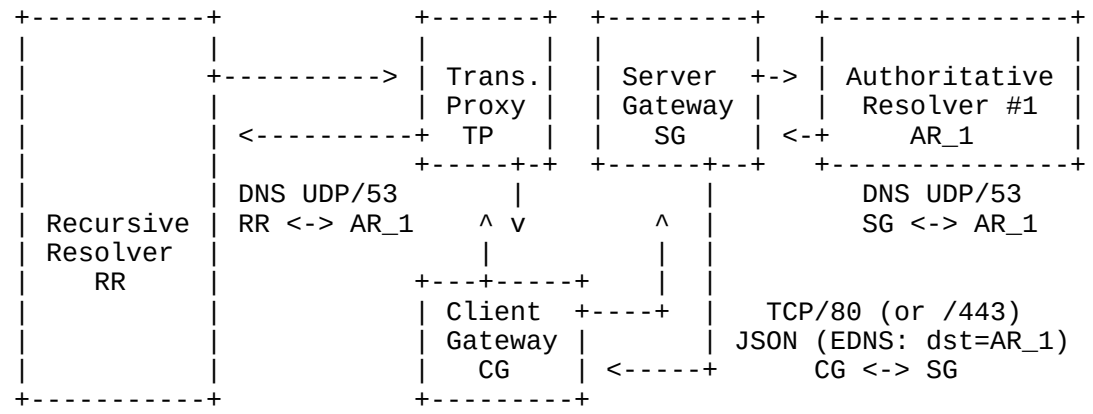


Figure 7

The only practical difference is that some intermediate devices see JSON/HTTP(S) instead of DNS/UDP traffic. For some of those devices, this is in fact the purpose of SPARTACUS - preventing those devices from inspecting the DNS traffic in a problematic manner.

3.3. Interoperability

The purpose of this document is to ensure that independent implementations of Client(s) and Server(s) can interoperate, so long as each is permitted to interoprate with the other.

It is not required that Servers be operated in a completely "open" manner. However, the more open Servers there are, the greater the benefit. Like any web-based service, care should be given towards managing available resources on a Server. In all likelihood, this resource management may be most effectively handled via the web server's own service management system.

3.3.1. In-scope and out-of-scope

The following items are out-of-scope, from an interoperability standpoint.

This means that individual implementations may make independent design decisions, without impacting interoperability.

- Choice(s) of default resolver (on Server)
- Server-side DNS retry and time-out values
- How Client(s) select Servers

The following items are in-scope, from an interoperability standpoint.

- JSON encoding
- How to signal non-support of requested resolver(s)
- How to signal "no response" (timeout) on Server-resolver traffic
- Signalling/encoding of default,requested, and actually-used resolvers
- Stripping of EDNS OPT private values
- Stripping of synthesized EDNS OPT record

The following items are optional, from an interoperability standpoint.

- Whether and how to do edns-client-subnet (on Client)

- Whether to use TLS (HTTPS) on Client-Server traffic
- Whether to honor requested resolvers (on Server)
- Whether to support Transparent Proxy mode (on the Client)
- Whether to do DNSSEC validation
- Whether to do PKI validation of SSL certificates (if HTTPS is used and CA-issued certs used)
- Whether to do DANE validation of SSL certificates (if HTTPS is used and TLSA records exist)
- Whether IPv4 or IPv6 is supported

4. Interactions and Behavior

The Client Gateway needs to make informed decisions about Server Gateways to use. Client Gateways may use pre-configured (static) gateways, or may employ any number of strategies for selection of Server Gateways.

In order to enble Client-controlled Server Selection, each Server Gateway needs to advise the Client about default and actual DNS Servers used. The Client optionally requests DNS Server(s) that the Server should use. If present, the Server includes that in the response.

The SPARTACUS client/server interaction occurs over TCP rather than UDP. As such, other than TCP-based failures (RST aka "reset" for example), every query MUST get a response (owing to the HTTP POST standards).

Since the Server Gateway is performing DNS resolution using UDP transport, it is possible that network packet loss may occur, resulting in unanswered queries.

Also, there are reasons other than network-based packet loss that can result in unanswered queries. DNS resolvers must attempt to infer what causes queries to not be answered.

It is also possible that various other failure modes could occur, which need to be handled on the basis of the nature of the failure.

Each of these is addressed in separate sections below.

4.1. DNS Gateway Encodings

The three DNS Server values (default, requested, actual) are communicated via EDNS OPT type-length-value (TLV) tuples, using three distinct types. Pre-standard experimental values are presently being used. IANA will need to assign permanent OPT Type values for these three type codes.

In order to ensure that the EDNS OPT record is only returned to the original DNS client if it existing in the query, it is necessary to identify cases where the DNS Server value encoding resulted in a "new" OPT record, rather than being added to an existing record. In such cases, an additional OPT TLV type is required, and is added to the OPT record. A fourth OPT Type value needs to be assigned by IANA for this purpose.

The new OPT codes are used to enable the Client and Server to maintain all communication details inside the DNS message itself. This simplifies the design, implementation, and operation of Clients and Servers, and ensures forward/backward compatibility. OPT codes specific to the Client-Server communication MUST be removed prior to forwarding of DNS messages to DNS Clients and Servers. If the EDNS OPT RR is synthesized (added to the DNS message), it MUST be removed.

4.2. UDP Packet Loss

In cases where the Server Gateway did not get a response from the DNS Server, it needs to signal this back to the client. It needs to do this so that the proper Client state is established. This prevents time-out based (undefined) behavior on the Client from being triggered. The Server needs to "pass along" packet loss status to the Client to trigger well-defined Client behavior.

The mechanism is to use a Private EDNS OPT type/length/value (TLV), with the original Question echoed back (to associate with the Query). When receiving this TLV, the Client will treat this as a lost UDP packet, and MUST NOT send back any UDP packet. The UDP client is responsible for handling this lost UDP packet, per the DNS protocol.

4.3. Malformed UDP response

The malformed UDP packet may not be legitimate. To be conservative, this condition is signaled back to the client, and the (actual) received UDP packet is rejected/dropped. This is treated by the DNS client as a lost UDP packet.

4.4. DNSSEC Validation Failure

If DNSSEC validation fails, the presumption needs to be made that the failure is deliberate. The DNSSEC standards call for "SRVFAIL" responses, so that is what a compliant implementation MUST return to the UDP client.

If the Client and/or Server does DNSSEC Validation, it MUST correctly implement Validation signalling via the AD and CD bits.

In other words, it MUST return the answer regardless of Validation if the CD bit is set, and it MUST set the AD bit if Validation succeeds, regardless of the presence and/or state of EDNS bit DO.

5. Client-Server Selection and Topology Examples

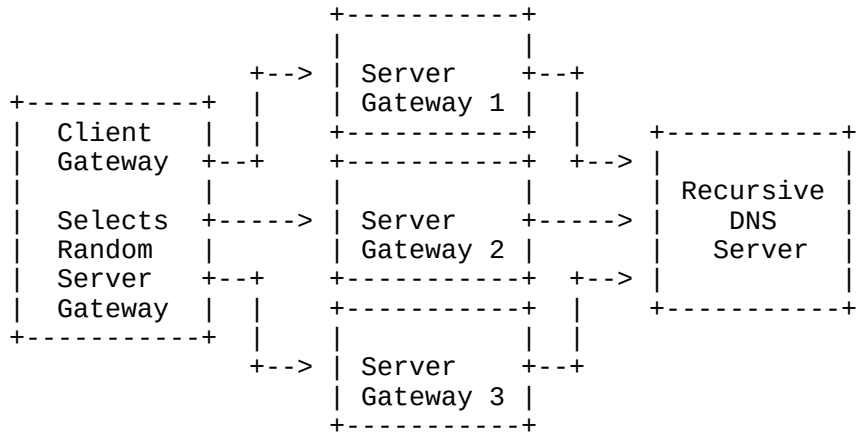


Figure 8

Figure 8 shows the same Recursive DNS Server being used, via multiple Server Gateways. There are several benefits to doing this; they include distributing load among multiple Server Gateways, and reducing the amount of DNS traffic going via any single Server Gateway. This limits the impact of the compromise of any single Server Gateway, or of any single Server Certificate compromise.

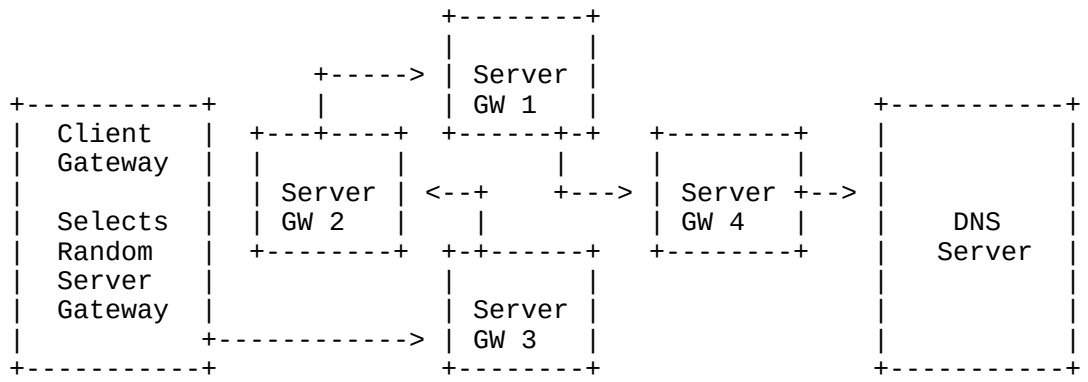


Figure 9

Figure 9 illustrates a path where more than one Server Gateway is traversed during resolution. The objective here is to disassociate the IDENTITY of the client from the CONTENT of the query/answer. The association is only made directly on the first Server Gateway (and only with respect to the Client Gateway). The actual association of the source UDP client is only done on the Client Gateway itself, which may or may not provide further privacy. Since there is more than one Server-Server hop, this significantly reduces the ability to infer associations between Query/Response and Client Gateways.

It should be noted that this looks very much like TOR (The Onion Router), applied to JSON-encoded UDP DNS traffic. There is a proposal for DNS privacy enhancements that applies a similar techique, directly on UDP-based DNS queries/answers. FIXME add xref here to reference to the appropriate Internet Draft. END FIXME

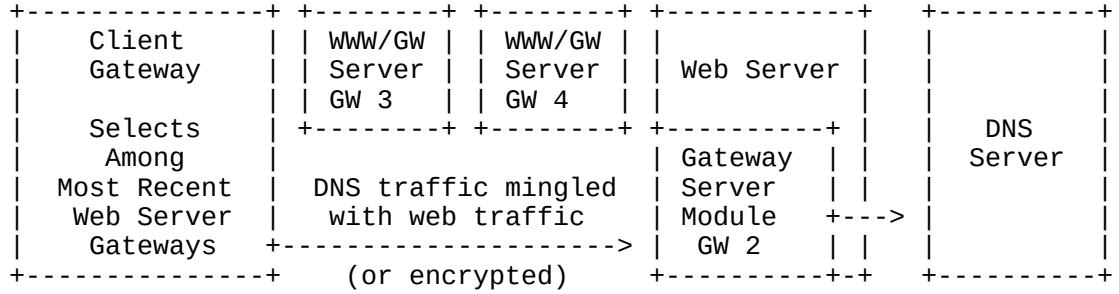


Figure 10

Figure 10 illustrates one query/response when the client is attempting to use something similar to steganography to preserve privacy. In this context, the privacy against passive monitoring is

achieved by using un-blocked web servers which are also Server Gateways. A MitM adversary cannot easily block this traffic without blocking the entire site, or by inspecting every flow to/from the site. A passive observer would similarly need to inspect all flows to find the embedded, encoded DNS traffic. The DNS traffic would be nearly indistinguishable from regular HTTP traffic.

Note that the use of TLS to protect the Client-Server traffic would make it impossible to distinguish the DNS traffic from the other web trafficin this situation. Combining this "tag-along" with TLS provides both strong privacy and strong security.

5.1. Mixed Traffic Walk-Through

Suppose a client were to visit web sites "a" through "j" sequentially, i.e. a,b,c,d,e,f,g,h,i,j. Suppose some of those were also Server Gateways, represented by upper case (vs lower case for web sites without Server Gateway capabilities). Thus the sequence would be A,b,C,D,e,f,g,H,I,j. If the Client Gateway chose a Server Gateway randomly from among the last four web sites visited, the sequence of events after visiting A through D, would look like:

- Select Server from set {A,C,D}, look up "e". Visit "e".
- Select Server from set {C,D}, look up "f". Visit "f".
- Select Server from set {C,D}, look up "g". Visit "g".
- Select Server from set {D}, look up "h". Visit "h".
- Select Server from set {D,H}, look up "i". Visit "i".
- Select Server from set {H,I}, look up "j". Visit "j".

An observer close to the Client would see traffic within a given time window, only to the same set of Web servers. An observer close to any of the Web servers would only see traffic from a given client, for a small interval of time after the first visit.

6. Security Considerations

(None per se.) Need to list considerations etc.

7. IANA Considerations

This document will eventually contain IANA-specific material.

8. Acknowledgements

To be added later.

9. References

9.1. Normative References

[RFC1033] Lottor, M., "Domain administrators operations guide", RFC 1033, November 1987.

[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.

[RFC2136] Vixie, P., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 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.

[RFC2845] Vixie, P., Gudmundsson, O., Eastlake, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000.

[RFC2930] Eastlake, D., "Secret Key Establishment for DNS (TKEY RR)", RFC 2930, September 2000.

[RFC3425] Lawrence, D., "Obsoleting IQUERY", RFC 3425, November 2002.

[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.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, March 2008.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, April 2013.

9.2. Informative References

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

[JPF_jsondns]
"DNS over HTTP", <http://github.com/jpf/jsondns>.

[jsondns.org]
Franusic, J., "Query DNS via REST", <http://jsondns.org/>.

[fileformat.info]
Marcuse, A., "DNS in client-side JavaScript", <http://www.fileformat.info/tool/rest/dns-json.htm>.

[restdns.net]
"REST-DNS", <http://restdns.net/>.

[bortzmeyer.org]
Bortzmeyer, S., "DNS Looking Glass", <http://www.bortzmeyer.org/dns-lg.html>.

[draft-bortzmeyer-dns-json]
Bortzmeyer, S., "DNS in JSON", <http://tools.ietf.org/html/draft-bortzmeyer-dns-json-01>.

[dns-lg.com]
Cambus, F., "Multilocation DNS Looking Glass", <http://www.dns-lg.com/>.

[trigger] NLnet Labs, "Dnssec-Trigger", <http://www.nlnetlabs.nl/projects/dnssec-trigger/>.

Appendix A. DNS Message Encoding Examples

The entire encoding of pairs of DNS messages follows. For each pair,the first is the query, and the second is the response.

A.1. Simple Query/Answer, No EDNS or DNS Server

Query encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 26,
"DICTIONARY" : [
"example",
"com",
""
],
"DSIZE2" : 26,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : false,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : false,
"Z" : false,
"AD" : false,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 0,
"NSCOUNT" : 0,
"ARCOUNT" : 0
],
"Question" : [
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
]
]
```

Response encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 33,
"DICTIONARY" : [
```

```
   "example",
   "com",
   "",
   "@0"
   ],
   "DSIZE2" : 33,
   "Header" : [
   "ID" : 42,
   "HFlags" : [
   "QR" : true,
   "Opcode" : [ "Query" : 0 ],
   "AA" : false,
   "TC" : false,
   "RD" : true,
   "RA" : true,
   "Z" : false,
   "AD" : false,
   "CD" : false,
   "RCODE" : [ "NoError (RFC 1035)" : 0 ]

   ],
   "QDCOUNT" : 1,
   "ANCOUNT" : 1,
   "NSCOUNT" : 0,
   "ARCOUNT" : 0
   ],
   "Question" : [
   "QUESTION (RFC 1035)" : [
   "QNAME" : [ "example.com." : 0 ],
   "QTYPE" : [ "A" : 1 ],
   "QCLASS" : [ "IN" : 1 ]
   ]
   ],
   "Answer" : [
   "RR" : [
   "NAME" : [ "example.com." : 0 ],
   "TYPE" : [ "A" : 1 ],
   "CLASS" : [ "IN" : 1 ],
   "TTL" : 5218,
   "RDLENGTH" : 4,
   "RDATA" : [
   "A" : [
   "Address" : "93.184.216.119"
   ]
   ]
   ]
   ]
   ]
```

A.2. Simple Query/Answer, EDNS, no DNS Server

Query encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 31,
"DICTIONARY" : [
"example",
"com",
"",
""
],
"DSIZE2" : 31,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : false,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : false,
"Z" : false,
"AD" : false,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 0,
"NSCOUNT" : 0,
"ARCOUNT" : 1
],
"Question" : [
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
],
"Additional" : [
"RR" : [
"NAME" : [ "." : 3 ],
"TYPE" : [ "OPT" : 41 ],
"Field3" : [
"UDPSIZEFIELD" : [
"UDPSIZE" : 1500
```

```
]
],
"Field4" : [
"Extended_RCode_Flags" : [
"ERCFlagbits" : [
"RCode" : 0,
"Version" : 0,
"DO" : false,
"Resv" : 0

]
]
],
"RDLENGTH" : 0,
"RDATA" : [
"OPT (RFC 6891)" : [
"TLV_LIST" : [

]
]
]
]
]
]
```

Response encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 38,
"DICTIONARY" : [
"example",
"com",
"",
"@0",
""
],
"DSIZE2" : 38,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : true,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : true,
"Z" : false,
```

```
   "AD" : false,
   "CD" : false,
   "RCODE" : [ "NoError (RFC 1035)" : 0 ]

   ],
   "QDCOUNT" : 1,
   "ANCOUNT" : 1,
   "NSCOUNT" : 0,
   "ARCOUNT" : 1
   ],
   "Question" : [
   "QUESTION (RFC 1035)" : [
   "QNAME" : [ "example.com." : 0 ],
   "QTYPE" : [ "A" : 1 ],
   "QCLASS" : [ "IN" : 1 ]
   ]
   ],
   "Answer" : [
   "RR" : [
   "NAME" : [ "example.com." : 0 ],
   "TYPE" : [ "A" : 1 ],
   "CLASS" : [ "IN" : 1 ],
   "TTL" : 4865,
   "RDLENGTH" : 4,
   "RDATA" : [
   "A" : [
   "Address" : "93.184.216.119"
   ]
   ]
   ]
   ],
   "Additional" : [
   "RR" : [
   "NAME" : [ "." : 4 ],
   "TYPE" : [ "OPT" : 41 ],
   "Field3" : [
   "UDPSIZEFIELD" : [
   "UDPSIZE" : 4000
   ]
   ],
   "Field4" : [
   "Extended_RCode_Flags" : [
   "ERCFlagbits" : [
   "RCode" : 0,
   "Version" : 0,
   "DO" : false,
   "Resv" : 0
```

```
   ]
   ]
   ],
   "RDLENGTH" : 0,
   "RDATA" : [
   "OPT (RFC 6891)" : [
   "TLV_LIST" : [

   ]
   ]
   ]
   ]
   ]
   ]
```

A.3. Simple Query/Answer, no EDNS, with DNS Server

Query encoded in JSON:

```
   "PACKET (RFC 1035)" : [
   "ROLE" : "client",
   "DSIZE" : 31,
   "DICTIONARY" : [
   "example",
   "com",
   "",
   ""
   ],
   "DSIZE2" : 31,
   "Header" : [
   "ID" : 42,
   "HFlags" : [
   "QR" : false,
   "Opcode" : [ "Query" : 0 ],
   "AA" : false,
   "TC" : false,
   "RD" : true,
   "RA" : false,
   "Z" : false,
   "AD" : false,
   "CD" : false,
   "RCODE" : [ "NoError (RFC 1035)" : 0 ]

   ],
   "QDCOUNT" : 1,
   "ANCOUNT" : 0,
   "NSCOUNT" : 0,
   "ARCOUNT" : 1
```

```
   ],
   "Question" : [
   "QUESTION (RFC 1035)" : [
   "QNAME" : [ "example.com." : 0 ],
   "QTYPE" : [ "A" : 1 ],
   "QCLASS" : [ "IN" : 1 ]
   ]
   ],
   "Additional" : [
   "RR" : [
   "NAME" : [ "." : 3 ],
   "TYPE" : [ "OPT" : 41 ],
   "Field3" : [
   "UDPSIZEFIELD" : [
   "UDPSIZE" : 1500
   ]
   ],
   "Field4" : [
   "Extended_RCode_Flags" : [
   "ERCFlagbits" : [
   "RCode" : 0,
   "Version" : 0,
   "DO" : false,
   "Resv" : 0

   ]
   ]
   ],
   "RDLENGTH" : 19,
   "RDATA" : [
   "OPT (RFC 6891)" : [
   "TLV_LIST" : [
   "TLV" : [
   "TYPE" : [ "PrivateType65500" : 65500 ],
   "Len" : 13,
   "Data" : [
   "PrivateType65500" : [
   "GW_NAME" : [ "10:10" , "198.41.1.1" ]

   ]
   ]
   ],
   "TLV" : [
   "TYPE" : [ "PrivateType65510" : 65510 ],
   "Len" : 0,
   "Data" : [
   ]
   ],
```

```
   ]
   ]
   ]
   ]
   ]
   ]
```

Response encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 38,
"DICTIONARY" : [
"example",
"com",
"",
"@0",
""
],
"DSIZE2" : 38,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : true,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : true,
"Z" : false,
"AD" : true,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 1,
"NSCOUNT" : 0,
"ARCOUNT" : 1
],
"Question" : [
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
],
"Answer" : [
```

```
   "RR" : [
   "NAME" : [ "example.com." : 0 ],
   "TYPE" : [ "A" : 1 ],
   "CLASS" : [ "IN" : 1 ],
   "TTL" : 4084,
   "RDLENGTH" : 4,
   "RDATA" : [
   "A" : [
   "Address" : "93.184.216.119"
   ]
   ]
   ]
   ],
   "Additional" : [
   "RR" : [
   "NAME" : [ "." : 4 ],
   "TYPE" : [ "OPT" : 41 ],
   "Field3" : [
   "UDPSIZEFIELD" : [
   "UDPSIZE" : 512
   ]
   ],
   "Field4" : [
   "Extended_RCode_Flags" : [
   "ERCFlagbits" : [
   "RCode" : 0,
   "Version" : 0,
   "DO" : false,
   "Resv" : 0

   ]
   ]
   ],
   "RDLENGTH" : 19,
   "RDATA" : [
   "OPT (RFC 6891)" : [
   "TLV_LIST" : [
   "TLV" : [
   "TYPE" : [ "PrivateType65500" : 65500 ],
   "Len" : 13,
   "Data" : [
   "PrivateType65500" : [
   "GW_NAME" : [ "10:10" , "198.41.1.1" ]

   ]
   ]
   ],
   "TLV" : [
```

```
   "TYPE" : [ "PrivateType65510" : 65510 ],
   "Len" : 0,
   "Data" : [
   ]
   ],

   ]
   ]
   ]
   ]
   ]
   ]
```

A.4. Simple Query/Answer, with EDNS and DNS Server

Query encoded in JSON:

```
   "PACKET (RFC 1035)" : [
   "ROLE" : "client",
   "DSIZE" : 31,
   "DICTIONARY" : [
   "example",
   "com",
   "",
   ""
   ],
   "DSIZE2" : 31,
   "Header" : [
   "ID" : 42,
   "HFlags" : [
   "QR" : false,
   "Opcode" : [ "Query" : 0 ],
   "AA" : false,
   "TC" : false,
   "RD" : true,
   "RA" : false,
   "Z" : false,
   "AD" : false,
   "CD" : false,
   "RCODE" : [ "NoError (RFC 1035)" : 0 ]

   ],
   "QDCOUNT" : 1,
   "ANCOUNT" : 0,
   "NSCOUNT" : 0,
   "ARCOUNT" : 1
   ],
   "Question" : [
```

```
   "QUESTION (RFC 1035)" : [
   "QNAME" : [ "example.com." : 0 ],
   "QTYPE" : [ "A" : 1 ],
   "QCLASS" : [ "IN" : 1 ]
   ]
   ],
   "Additional" : [
   "RR" : [
   "NAME" : [ "." : 3 ],
   "TYPE" : [ "OPT" : 41 ],
   "Field3" : [
   "UDPSIZEFIELD" : [
   "UDPSIZE" : 1500
   ]
   ],
   "Field4" : [
   "Extended_RCode_Flags" : [
   "ERCFlagbits" : [
   "RCode" : 0,
   "Version" : 0,
   "DO" : false,
   "Resv" : 0

   ]
   ]
   ],
   "RDLENGTH" : 15,
   "RDATA" : [
   "OPT (RFC 6891)" : [
   "TLV_LIST" : [
   "TLV" : [
   "TYPE" : [ "PrivateType65500" : 65500 ],
   "Len" : 13,
   "Data" : [
   "PrivateType65500" : [
   "GW_NAME" : [ "10:10" , "198.41.1.1" ]

   ]
   ]
   ],

   ]
   ]
   ]
   ]
   ]
   ]
```

Response encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 38,
"DICTIONARY" : [
"example",
"com",
"",
"@0",
""
],
"DSIZE2" : 38,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : true,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : true,
"Z" : false,
"AD" : true,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 1,
"NSCOUNT" : 0,
"ARCOUNT" : 1
],
"Question" : [
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
],
"Answer" : [
"RR" : [
"NAME" : [ "example.com." : 0 ],
"TYPE" : [ "A" : 1 ],
"CLASS" : [ "IN" : 1 ],
"TTL" : 4084,
"RDLENGTH" : 4,
"RDATA" : [
```

```
   "A" : [
   "Address" : "93.184.216.119"
   ]
   ]
   ]
   ],
   "Additional" : [
   "RR" : [
   "NAME" : [ "." : 4 ],
   "TYPE" : [ "OPT" : 41 ],
   "Field3" : [
   "UDPSIZEFIELD" : [
   "UDPSIZE" : 512
   ]
   ],
   "Field4" : [
   "Extended_RCode_Flags" : [
   "ERCFlagbits" : [
   "RCode" : 0,
   "Version" : 0,
   "DO" : false,
   "Resv" : 0

   ]
   ]
   ],
   "RDLENGTH" : 15,
   "RDATA" : [
   "OPT (RFC 6891)" : [
   "TLV_LIST" : [
   "TLV" : [
   "TYPE" : [ "PrivateType65500" : 65500 ],
   "Len" : 13,
   "Data" : [
   "PrivateType65500" : [
   "GW_NAME" : [ "10:10" , "198.41.1.1" ]

   ]
   ]
   ],

   ]
   ]
   ]
   ]
   ]
   ]
```

Appendix B. Server Gateway HTML code

The entire HTML document needed on the Server for the Client to send/ receive JSON-encoded DNS messages follows:

```
<html>
<body>
<form action="cgi-bin/json-resolver2.pl" method="POST">
<P>
<TEXTAREA name="query" rows="20" cols="80">
</TEXTAREA>
<INPUT type="submit" value="Send"><INPUT type="reset">
</P>
</form>
</body>
</html>
```

The "action" target needs to exist and be executable, and ideally be performance-optimized (e.g. via use of mod_perl).

Appendix C. Server Gateway HTTP POST Handler Pseudo-code

The following pseudo-code illustrates the high-level behavior of the HTML handler for the Server.

The handler is passed the contents of the TEXTAREA, which will be the JSON-encoded DNS message.

```
// initialize parser etc.
// set up socket for UDP query/response to default Resolver
// set up socket for UDP query/response to client-supplied Resolver
// extract JSON-encoded DNS message from HTTP POST variable 'query'
// save original Query-ID, assign new Query-ID (to avoid collisions)
// decode DNS message (into DNS wire format)
// if DNS message has OPT Resource Record
//   if OPT has Client-supplied Resolver option
//     extract Resolver value
//     delete Resolver option from OPT
//   endif
//   if OPT was synthesized by Client
//     delete OPT Resource Record
//   endif
//   send DNS message to Client-specified Resolver
// else
//   send DNS message to default Resolver
// endif
// wait for response or timeout
// if timeout && retry-count < max-retry-count
//   resend DNS message
// elsif timeout && retry-count >= max-retry-count
//   send "retry-count-exceeded" via OPT (synthesized if necessary)
// else
//   set DNS answer's Query-ID value to original Query-ID
//   encode DNS answer
//   send JSON-encoded answer to Client
// endif
```

Appendix D. Client Gateway Pseudo-code

The following pseudo-code illustrates the high-level behavior of the Client.

The Client in this example is pre-configured with a single Server Gateway's address.

```
   // initialize parser etc.
   // set up socket for UDP query/response (Listener)
   // set up HTTP connection to Server Gateway
   // do an HTTP "GET" to the predefined URL of the Server HTML code
   // extract HTML elements needed: handler, variable name
   // loop forever:
   //   listen for DNS query packet
   //   fork (to handle this packet)
   //   if child
   //     save old DNS Query-ID, set new Query-ID
   //     if Use-Supplied-Resolver
   //       if exits OPT
   //         add client-supplied-resolver to OPT
   //       else
   //         synthesize OPT and add client-supplied-resolver
   //       endif
   //     endif
   //     encode DNS message (into JSON)
   //     write HTTP POST onto socket
   //     wait for HTTP response
   //     extract JSON-encoded answer from HTTP
   //     decode DNS answer (from JSON)
   //     if OPT
   //       if OPT.option is error condition
   //         drop answer and continue loop forever:
   //       elsif OPT synthesized
   //         delete OPT
   //       elsif OPT.option SPARTACUS-specific value
   //         delete option
   //       endif
   //     endif
   //     set answer.Query-ID to saved value
   //     send answer to sender
   //   end-of-loop
```

Author’s Address

Brian Dickson
12047B 36th Ave NE
Seattle, wA 98125

Email: brian.peter.dickson@gmail.com

INTERNET-DRAFT
Intended Status: Proposed Standard
Expires: April 10, 2015

Donald Eastlake
Huawei
Mark Andrews
ISC

October 11, 2014

Domain Name System (DNS) Cookies
<draft-eastlake-dnsext-cookies-05.txt>

Abstract

DNS cookies are a lightweight DNS transaction security mechanism that provides limited protection to DNS servers and clients against a variety of increasingly common denial-of-service and amplification / forgery or cache poisoning attacks by off-path attackers. DNS Cookies are tolerant of NAT, NAT-PT, and anycast and can be incrementally deployed.

Status of This Document

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Distribution of this document is unlimited. Comments should be sent to the author or the DNSEXT mailing list <dnsext@ietf.org>.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html. The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Table of Contents

1. Introduction

As with many core Internet protocols, the Domain Name System (DNS) was originally designed at a time when the Internet had only a small pool of trusted users. As the Internet has grown exponentially to a global information utility, the DNS has increasingly been subject to abuse.

This document describes DNS cookies, a lightweight DNS transaction security mechanism specified as an OPT [RFC6891] option. This mechanism provides limited protection to DNS servers and clients against a variety of increasingly common abuses by off-path attackers. It is compatible with and can be used in conjunction with other DNS transaction forgery resistance measures such as those in [RFC5452].

The protection provided by DNS cookies bears some resemblance to that provided by using TCP for DNS transactions. To bypass the weak protection provided by using TCP requires an off-path attacker guessing the 32-bit TCP sequence number in use. To bypass the weak protection provided by DNS Cookies requires such an attacker to guess a 64-bit pseudo-random quantity. Where DNS Cookies are not available but TCP is, a fall back to using TCP is a reasonable strategy.

If only one party to a DNS transaction supports DNS cookies, the mechanism does not provide a benefit or significantly interfere; but, if both support it, the additional security provided is automatically available.

The DNS cookies mechanism is designed to work in the presence of NAT and NAT-PT boxes and guidance is provided herein on supporting the DNS cookies mechanism in anycast servers.

1.1 Contents of This Document

In Section 2, we discuss the threats against which the DNS cookie mechanism provides some protection.

Section 3 describes existing DNS security mechanisms and why they are not adequate substitutes for DNS cookies.

Section 4 describes the COOKIE OPT option.

Section 5 provides a protocol description.

Section 6 discusses some NAT and anycast related DNS Cookies design considerations.

Section 7 discusses incremental deployment considerations.

Sections 8 and 9 describe IANA and Security Considerations.

1.2 Definitions

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].

An "off-path attacker", for a particular DNS client and server, is defined as an attacker who cannot observe the DNS request and response messages between that client and server.

"Soft state" indicates information learned or derived by a host which may be discarded when indicated by the policies of that host but can be later re-instantiated if needed. For example, it could be discarded after a period of time or when storage for caching such data becomes full. If operations requiring that soft state continue after it has been discarded, it will be automatically re-generated, albeit at some cost.

"Silently discarded" indicates that there are no DNS protocol message consequences; however, it is RECOMMENDED that appropriate network management facilities be included in implementations, such as a counter of the occurrences of each such event type.

"IP address" is used herein as a length independent term and includes both IPv4 and IPv6 addresses.

2. Threats Considered

DNS cookies are intended to provide significant but limited protection against certain attacks by off-path attackers as described below. These attacks include denial-of-service, cache poisoning and answer forgery.

2.1 Denial-of-Service Attacks

The typical form of the denial-of-service attacks considered herein is to send DNS requests with forged source IP addresses to a server. The intent can be to attack that server or some other selected host as described below.

2.1.1 DNS Amplification Attacks

A request with a forged IP address generally causes a response to be sent to that forged IP address. Thus the forging of many such requests with a particular source IP address can result in enough traffic being sent to the forged IP address to interfere with service to the host at the IP address. Furthermore, it is generally easy in the DNS to create short requests that produce much longer responses, thus amplifying the attack.

The DNS Cookies mechanism can severely limit the traffic amplification obtained by attackers off path for the server and the attacked host. Enforced DNS cookies would make it hard for an off path attacker to cause any more than rate-limited short error responses to be sent to a forged IP address so the attack would be attenuated rather than amplified. DNS cookies make it more effective to implement a rate limiting scheme for error responses from the server. Such a scheme would further restrict selected host denial-of-service traffic from that server.

2.1.2 DNS Server Denial-of-Service

DNS requests that are accepted cause work on the part of DNS servers. This is particularly true for recursive servers that may issue one or more requests and process the responses thereto, in order to determine their response to the initial request. And the situation can be even worse for recursive servers implementing DNSSEC ([RFC4033] [RFC4034] [RFC4035]) because they may be induced to perform burdensome cryptographic computations in attempts to verify the authenticity of data they retrieve in trying to answer the

request.

The computational or communications burden caused by such requests may not dependent on a forged IP source address, but the use of such addresses makes

+ the source of the requests causing the denial-of-service attack harder to find and
+ restriction of the IP addresses from which such requests should be honored hard or impossible to specify or verify.

Use of DNS cookies should enables a server to reject forged queries from an off path attacker with relative ease and before any recursive queries or public key cryptographic operations are performed.

2.2 Cache Poisoning and Answer Forgery Attacks

The form of the cache poisoning attacks considered is to send forged replies to a resolver. Modern network speeds for well-connected hosts are such that, by forging replies from the IP addresses of a DNS server to a resolver for common names or names that resolver has been induced to resolve, there can be an unacceptably high probability of randomly coming up with a reply that will be accepted and cause false DNS information to be cached by that resolver (the Dan Kaminsky attack). This can be used to facilitate phishing attacks and other diversion of legitimate traffic to a compromised or malicious host such as a web server.

With the use of DNS cookies, a resolver can generally reject such forged replies.

3. Comments on Existing DNS Security

Two forms of security have been added to DNS, data security and message/transaction security.

3.1 Existing DNS Data Security

DNS data security is one part of DNSSEC and is described in [RFC4033], [RFC4034], and [RFC4035] and updates thereto. It provides data origin authentication and authenticated denial of existence. DNSSEC is being deployed and can provide strong protection against forged data; however, it has the unintended effect of making some denial-of-service attacks worse because of the cryptographic computational load it can require and the increased size in DNS response packets that it tends to produce.

3.2 DNS Message/Transaction Security

The second form of security that has been added to DNS provides "transaction" security through TSIG [RFC2845] or SIG(0) [RFC2931]. TSIG could provide strong protection against the attacks for which the DNS Cookies mechanism provide weak protection; however, TSIG is non-trivial to deploy in the general Internet because of the burden it imposes of pre-agreement and key distribution between client-server pairs, the burden of server side key state, and because it requires time synchronization between client and server.

TKEY [RFC2930] can solve the problem of key distribution for TSIG but some modes of TKEY impose a substantial cryptographic computation loads and can be dependent on the deployment of DNS data security (see Section 3.1).

SIG(0) [RFC2931] provides less denial of service protection than TSIG or, in one way, even DNS cookies, because it does not authenticate requests, only complete transactions. In any case, it also depends on the deployment of DNS data security and requires computationally burdensome public key cryptographic operations.

3.3 Conclusions on Existing DNS Security

The existing DNS security mechanisms do not provide the services provided by the DNS Cookies mechanism: lightweight message authentication of DNS requests and responses with no requirement for pre-configuration or per client server side state.

4. The COOKIE OPT Option

COOKIE is an OPT RR [RFC6891] option that can be included in the RDATA portion of an OPT RR in DNS requests and responses. The option length varies depending on the circumstance in which it is being used. There are two cases as described below. Both use the same OPTION-CODE; they are distinguished by their length.

In a request sent by a client to a server when the client does not know the server cookie, its length is 10, consisting of a 2 bytes DNS error code field followed by the 8 byte Client Cookie as shown in Figure 1.

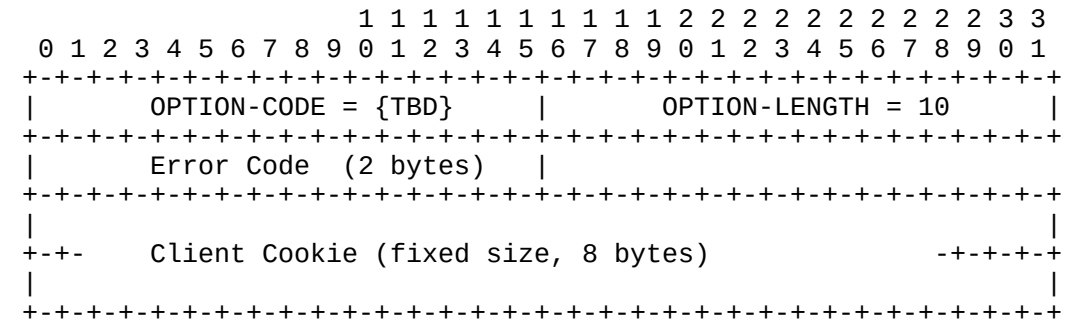


Figure 1. COOKIE Option, Unknown Server Cookie

In a request sent by a client when a server cookie is known and in all responses, the length is variable from 18 to 42 bytes, consisting of a 2 byte DNS error field followed by the 8 bytes Client Cookie and then the variable 8 to 32 bytes Server Cookie as shown in Figure 2. The variability of the option length stems from the variable length Server Cookie. The Server Cookie is an integer number of bytes with a minimum size of 64 bits for security and a maximum size of 256 bits for implementation convenience.

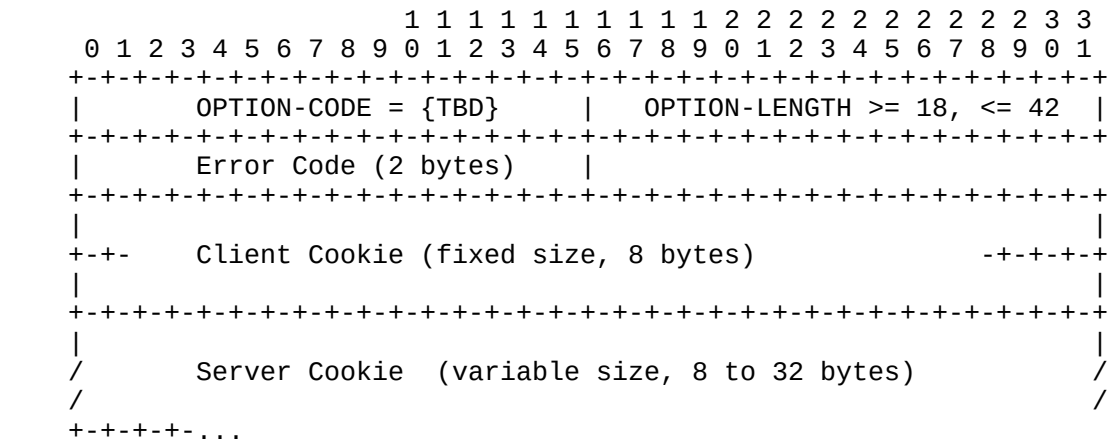


Figure 2. COOKIE Option, Known Server Cookie

4.1 Client Cookie

The Client Cookie SHOULD be a pseudo-random function of the server IP address and a secret quantity known only to the client. This client secret SHOULD have at least 64 bits of entropy [RFC4086] and be changed periodically (see Section 5.4). The selection of the pseudo-random function is a matter private to the client as only the client needs to recognize its own DNS cookies.

For further discussion of the Client Cookie field, see Section 5.1. For example methods of determining a Client Cookie, see Appendix A.

A client MUST NOT use the same Client Cookie value for queries to all servers.

4.2 Server Cookie

The Server Cookie SHOULD consist of or include a 64-bit or larger pseudo-random function of the request source IP address, the request Client Cookie, and a secret quantity known only to the server. (See Section 6 for a discussion of why the Client Cookie is used as input to the Server Cookie but the Server Cookie is not used as an input to the Client Cookie.) This server secret SHOULD have at least 64 bits of entropy [RFC4086] and be changed periodically (see Section 5.4). The selection of the pseudo-random function is a matter private to the server as only the server needs to recognize its own DNS cookies.

For further discussion of the Server Cookie field see Section 5.2. For example methods of determining a Server Cookie, see Appendix B.

A server MUST NOT use the same Server Cookie value for responses to all clients.

4.3 Error Code

In requests, the Error Code field MUST be zero and is ignored on receipt. Replies have a COOKIE OPT with an Error Code equal to one of the following four values: Zero (if the request they respond to had a COOKIE OPT with a correct Server Cookie), NOCOOKIE, MFCOOKIE, or BADCOOKIE.

NOCOOKIE and MFCOOKIE indicate that the server did not receive a Client Cookie, either because there was no COOKIE OPT option in the request (NOCOOKIE) or one was present but the COOKIE OPT option was malformed as not being a valid length (MFCOOKIE). BADCOOKIE indicates that the server did receive a Client Cookie but did not receive the correct Server Cookie either because there was no Server Cookie present or because it was not a valid value.

A server may choose to normally process a request, for example returning the normal answer information for a QUERY, notwithstanding a cookie error condition. For more information on error processing, see Section 5.

5. DNS Cookies Protocol Description

This section discusses using DNS Cookies in the DNS Protocol.

5.1 Originating Requests

A DNS client that implements DNS cookies includes one DNS Cookie option in every DNS request it sends unless DNS cookies are disabled. The COOKIE OPT option in a request always includes a zero Error Code field and a Client Cookie as discussed in Section 4.1.

If the client has no Server Cookie obtained from a previous DNS response and cached under the server's IP address, it uses the shorter form of COOKIE OPT shown in Figure 1. If the client does have such a cached Server Cookie, it uses the form of COOKIE OPT shown in Figure 2 and also includes that cached Server Cookie in the DNS option it sends.

5.2 Responding to Requests

The Server Cookie, when it occurs in a COOKIE OPT option in a request, is intended to weakly assure the server that the request came from a client that is both at the source IP address of the request and using the Client Cookie included in the option. This weak assurance is provided by the Server Cookie that server would send to that client in an earlier response appearing as the Server Cookie field in the request.

At a server where DNS Cookies are not implemented and enabled, the presence of a COOKIE OPT option is ignored and the server responds as before.

When DNS Cookies are implemented and enabled, there are four possibilities: (1) there is no OPT RR at all in the request; (2) there is no valid Client Cookie in the request because the COOKIE OPT option in absent from the request or one is present but not a legal length; (3) there is a valid length cookie option in the request with no Server Cookie or an incorrect Server Cookie; or (4) there is a cookie option in the request with a correct Server Cookie. The four possibilities are discussed in the subsections below.

In the case of multiple COOKIE OPT options in a request, only the first (the one closest to the DNS header) is considered. All others are ignored.

5.2.1 No OPT RR

If there is no OPT RR in the request, the client does not support EDNS since [RFC6891] requires that an OPT RR be included in a request if the requester supports that feature. Under these circumstances, the server cannot expect to ever receive a correct COOKIE OPT option from the client as in Section 5.2.4.

The situation and server options available are the same as those in Section 5.2.2 except that no OPT RR can be included in any response.

5.2.2 No Valid Client Cookie

A request with an OPT RR but no COOKIE OPT option or with a COOKIE that is not a valid length (10 or 18 through 42) could be from a client that does not implement DNS cookies or on which they are disabled or it could be some form of abuse or broken client implementation. A server on which DNS cookies are enabled has the following three choices in responding to such a request:

(1) Silently discard the request.

(2) Not process the request other than returning a minimal length error response. Because of the absence of a validly formatted COOKIE OPT option in the request, it cannot be assumed that the client would understand any new RCODE values. An RCODE of Refused is returned and the Error Field of the returned COOKIE OPT option is set to NOCOOKIE if there was no COOKIE OPT option in the request and set to MFCOOKIE if such an option was present but not a valid length.

(3) Process the request normally and provide a normal response except that a COOKIE OPT option with a non-zero Error Field is included as in point 2 above. The RCODE in the DNS Header is zero unless some non-cookie error occurs in processing the request.

Server policy determines how often the server selects each of the above response choices; however, if the request was received over TCP, the server may wish to take the weak authentication provided by the use of TCP into account, increasing the probability of choice 3 and decreasing the probability of choice 1 perhaps to the extent of never choosing 1. For both response choices 2 and 3, the server should consider setting TC=1 in the response so that future requests from the client are more likely to be received with the weak authentication that can be provided by TCP.

5.2.3 Bad or Absent Server Cookie

If a request is received with the COOKIE OPT option having no Server Cookie value (length 10) or a bad Server Cookie value (length 18 to 42), it could be some attempted abuse or it could just be that the client does not know a currently valid Server Cookie for the server to which the request was sent. For example, the client might have an old, no longer recognized Server Cookie

Servers MUST, at least occasionally, respond to such requests to inform the client of the correct Server Cookie. This is necessary so that such a client can bootstrap to the weakly secure state where requests and responses have recognized Server Cookies and Client Cookies.

In responding to such a request, the server has the following three choices:

(1) Silently discard the request.

(2) Not process the request other than returning a minimal length error response. Because of the correct length COOKIE OPT option in the request, the client can be assume to understand the new error codes assigned in this document. Both the Error Field in the returned COOKIE OPT option and the extended RCODE are set to BADCOOKIE.

(3) Processes the request normally and sends its usual response including a COOKIE OPT option with an Error field of BADCOOKIE and a zero RCODE (unless there was also a non-cookie error in processing the request).

Server policy determines how often the server selects each of the above response choices; however, if the request was received over TCP, the server may wish to take the weak authentication provided by the use of TCP into account, increasing the probability of choice 3 and decreasing the probability of choice 1 perhaps to the extent of never choosing 1.

5.2.4 A Correct Server Cookie

If a server with enabled DNS cookies receives a request where the COOKIE OPT option has a valid length and correct Server Cookie, it processes the request normally and includes a COOKIE OPT option with a zero Error Field in the response. Such a response might have a non-zero RCODE if a non-cookie error occurs in processing the request.

5.3 Processing Responses

The Client Cookie, when it occurs in a COOKIE OPT option in a DNS reply, is intended to weakly assure the client that the reply came from a server at the source IP address use in the response packet because the Client Cookie value is the value that client would send to that server in a request. If there are multiple COOKIE OPT options in a DNS reply, all but the first (the one closest to the DNS Header) are ignored.

A DNS client where DNS cookies are implemented and enabled examines response for DNS cookies and MUST discard the response if it contains an illegal COOKIE OPT option length or an incorrect Client Cookie value. If the COOKIE OPT option Client Cookie is correct, the client caches the Server Cookie provided even if the response is an error response (RCODE non-zero).

If the reply extended RCDOE is BADCOOKIE, it means that the server was unwilling to process the request because it did not have the correct Server Cookie in it. The client should retry the request using the new Server Cookie from the response.

If the RCODE is some value other than BADCOOKIE, including zero, the response is then processed normally.

5.4 Client and Server Secret Rollover

Clients and servers MUST NOT continue to use the same secret in new queries and responses, respectively, for more than 14 days and SHOULD NOT continue to do so for more than 1 day. Many clients rolling over their secret at the same time could briefly increase server traffic and exactly predictable rollover times for clients or servers might facilitate guessing attacks. For example, an attacker might increase the priority of attacking secrets they believe will be in effect for an extended period of time. To avoid rollover synchronization and predictability, it is RECOMMENDED that pseudorandom jitter of at least 30% be applied to the time of a scheduled rollover of a DNS cookie secret.

It is RECOMMENDED that a client keep the Client Cookie it is expecting in a reply associated with the outstanding query to avoid rejection of replies due to a bad Client Cookie right after a change in the Client Secret. It is RECOMMENDED that a server retain its previous secret for a period of time not less than 1 second or more than 3 minutes, after a change in its secret, and consider queries with Server Cookies based on its previous secret to have a correct Server Cookie during that time.

Receiving a sudden increased level of requests with bad Server Cookies or replies with bad Client Cookies would be a good reason to believe a server or client is likely to be under attack and should consider more frequent rollover of its secret.

5.5 Implementation Requirement

DNS clients and servers SHOULD implement DNS cookies to decrease their vulnerability to the threats discussed in Section 2.

6. NAT Considerations and AnyCast Server Considerations

In the Classic Internet, DNS Cookies could simply be a pseudo-random function of the client IP address and a sever secret or the server IP address and a client secret. You would want to compute the Server Cookie that way, so a client could cache its Server Cookie for a particular server for an indefinitely amount of time and the server could easily regenerate and check it. You could consider the Client Cookie to be a weak client signature over the server IP address that the client checks in replies and you could extend this weak signature to cover the request ID, for example, or any other information that is returned unchanged in the reply.

But we have this reality called NAT [RFC3022], Network Address Translation (including, for the purposes of this document, NAT-PT, Network Address and Protocol Translation, which has been declared Historic [RFC4966]). There is no problem with DNS transactions between clients and servers behind a NAT box using local IP addresses. Nor is there a problem with NAT translation of internal addresses to external addresses or translations between IPv4 and IPv6 addresses, as long as the address mapping is relatively stable. Should the external IP address an internal client is being mapped to change occasionally, the disruption is little more than when a client rolls-over its DNS COOKIE secret. And normally external access to a DNS server behind a NAT box is handled by a fixed mapping which forwards externally received DNS requests to a specific host.

However, NAT devices sometimes also map ports. This can cause multiple DNS requests and responses from multiple internal hosts to be mapped to a smaller number of external IP addresses, such as one address. Thus there could be many clients behind a NAT box that appear to come from the same source IP address to a server outside that NAT box. If one of these were an attacker (think Zombie or Botnet), that behind-NAT attacker could get the Server Cookie for some server for the outgoing IP address by just making some random request to that server. It could then include that Server Cookie in the COOKIE OPT of requests to the server with the forged local IP address of some other host and/or client behind the NAT box. (Attacker possession of this Server Cookie will not help in forging responses to cause cache poisoning as such responses are protected by the required Client Cookie.)

To fix this potential defect, it is necessary to distinguish different clients behind a NAT box from the point of view of the server. It is for this reason that the Server Cookie is specified as a pseudo-random function of both the request source IP address and the Client Cookie. From this inclusion of the Client Cookie in the calculation of the Server Cookie, it follows that a stable Client Cookie, for any particular server, is needed. If, for example, the request ID was included in the calculation of the Client Cookie, it

would normally change with each request to a particular server. This would mean that each request would have to be sent twice: first to learn the new Server Cookie based on this new Client Cookie based on the new ID and then again using this new Client Cookie to actually get an answer. Thus the input to the Client Cookie computation must be limited to the server IP address and one or more things that change slowly such as the client secret.

In principle, there could be a similar problem for servers, not due to NAT but due to mechanisms like anycast which may cause queries to a DNS server at an IP address to be delivered to any one of several machines. (External queries to a DNS server behind a NAT box usually occur via port forwarding such that all such queries go to one host.) However, it is impossible to solve this the way the similar problem was solved for NATed clients; if the Server Cookie was included in the calculation of the Client Cookie the same way the Client Cookie is included in the Server Cookie, you would just get an almost infinite series of errors as a request was repeatedly retried.

For servers accessed via anycast to successfully support DNS COOKIES, the server clones must either all use the same server secret or the mechanism that distributes queries to them must cause the queries from a particular client to go to a particular server for a sufficiently long period of time that extra queries due to changes in Server Cookie resulting from accessing different server machines are not unduly burdensome. (When such anycast-accessed servers act as recursive servers or otherwise act as clients they normally use a different unique address to source their queries to avoid confusion in the delivery of responses.)

For simplicity, it is RECOMMENDED that the same server secret be used by each DNS server in a set of anycast servers. If there is limited time skew in updating this secret in different anycast servers, this can be handled by a server accepting requests containing a Server Cookie based on either its old or new secret for the maximum likely time period of such time skew (see also Section 5.4).

7. Deployment

The DNS cookies mechanism is designed for incremental deployment and to complement the orthogonal techniques in [RFC5452]. Either or both techniques can be deployed independently at each DNS server and client.

In particular, a DNS server or client that implements the DNS COOKIE mechanism can interoperate successfully with a DNS client or server that does not implement this mechanism although, of course, in this case it will not get the benefit of the mechanism and the server involved might choose to severely rate limit responses. When such a server or client interoperates with a client or server which also implements the DNS cookies mechanism, they get the weak security benefits of the DNS Cookies mechanism.

8. IANA Considerations

IANA is requested to assign the following four code points:

The OPT option value for COOKIE is <TBD> [10 suggested].

Three new DNS error codes in the range above 16 and below 3,840 as shown below:

RCODE	Name	Description	Reference
TBD1[23]	NOCOOKIE	No client cookie.	[this document]
TBD2[24]	MFCOOKIE	Malformed cookie.	[this document]
TBD3[25]	BADCOOKIE	Bad/missing server cookie.	[this document]

9. Security Considerations

DNS Cookies provide a weak form of authentication of DNS requests and responses. In particular, they provide no protection against "on-path" adversaries; that is, they provide no protection against any adversary that can observe the plain text DNS traffic, such as an on-path router, bridge, or any device on an on-path shared link (unless the DNS traffic in question on that path is encrypted).

For example, if a host is connected via an unsecured IEEE 802.11 link (Wi-Fi), any device in the vicinity that could receive and decode the 802.11 transmissions must be considered "on-path". On the other hand, in a similar situation but one where 802.11 Robust Security (WPAv2) is appropriately deployed on the Wi-Fi network nodes, only the Access Point via which the host is connecting is "on-path" as far as the 802.11 link is concerned.

Despite these limitations, deployment of DNS Cookies on the global Internet is expected to provide a substantial reduction in the available launch points for the traffic amplification and denial of service forgery attacks described in Section 2 above.

Should stronger message/transaction security be desired, it is suggested that TSIG or SIG(0) security be used (see Section 3.2); however, it may be useful to use DNS Cookies in conjunction with these features. In particular, DNS Cookies could screen out many DNS messages before the cryptographic computations of TSIG or SIG(0) are required and, if SIG(0) is in use, DNS Cookies could usefully screen out many requests given that SIG(0) does not screen requests but only authenticates the response of complete transactions.

9.1 Cookie Algorithm Considerations

The cookie computation algorithm for use in DNS Cookies SHOULD be based on a pseudo-random function at least as strong as [FNV] because an excessively weak or trivial algorithm could enable adversaries to guess cookies. However, in light of the weak plain-text token security provided by DNS Cookies, a strong cryptography hash algorithm may not be warranted in many cases, and would cause an increased computational burden. Nevertheless there is nothing wrong with using something stronger, for example, HMAC-SHA256-64 [RFC6234], assuming a DNS processor has adequate computational resources available. DNS processors that feel the need for somewhat stronger security without a significant increase in computational load should consider more frequent changes in their client and/or server secret; however, this does require more frequent generation of a cryptographically strong random number [RFC4086]. See Appendices A and B for specific examples of cookie computation algorithms.

Acknowledgements

The contributions of the following are gratefully acknowledged:

Tim Wicinski

Normative References

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

[RFC4086] - Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC6891] - Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, April 2013.

Informative References

[FNV] - G. Fowler, L. C. Noll, K.-P. Vo, D. Eastlake, "The FNV Non-Cryptographic Hash Algorithm", draft-eastlake-fnv, work in progress.

[RFC2845] - Vixie, P., Gudmundsson, O., Eastlake 3rd, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000.

[RFC2930] - Eastlake 3rd, D., "Secret Key Establishment for DNS (TKEY RR)", RFC 2930, September 2000.

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

[RFC3022] - Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, January 2001.

[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.

[RFC4966] - Aoun, C. and E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status", RFC 4966, July 2007.

[RFC5452] - Hubert, A. and R. van Mook, "Measures for Making DNS More

Resilient against Forged Answers", RFC 5452, January 2009.

[RFC6234] - Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May 2011.

Appendix A: Example Client Cookie Algorithms

A.1 A Simple Algorithm

An simple example method to compute Client Cookies is the FNV-64 [FNV] of the server IP address and the client secret. That is

Client Cookie = FNV-64 (Client Secret | Server IP Address)

where "|" indicates concatenation.

A.2 A More Complex Algorithm

A more complex algorithm to calculate Client Cookies is give below. It uses more computational resources than the simpler algorithm shown in A.1.

Client Cookie = HMAC-SHA256-64 (Client Secret, Server IP Address)

Appendix B: Example Server Cookie Algorithms

B.1 A Simple Algorithm

An example simple method producing a 64-bit Server Cookie is the FNV-64 [FNV] of the request IP address, the Client Cookie, and the server secret. That is

```
Server Cookie =
   FNV-64 ( Server Secret | Request IP Address | Client Cookie )
```

where "|" represents concatenation.

B.2 A More Complex Algorithm

Since the Server Cookie is variable size, the server can store various information in that field as long as it is hard for an adversary to guess the entire quantity used for weak authentication. There should be 64 bits of entropy in the Server Cookie; for example it could have a sub-field of 64-bits computed pseudo-randomly with the server secret as one of the inputs to the pseudo-random function. Types of additional information that could be stored include a time stamp and/or a nonce.

The example below is one variation for the Server Cookie that has been implemented in a beta release of BIND where the Server Cookie is 128 bits composed as follows:

Sub-field	Size
Nonce	32 bits
Time	32 bits
Hash	64 bits

With this algorithm, the server sends a new 128-bit cookie back with every request. The Nonce field assures a low probability that there would be a duplicate.

The Time field gives the server time and makes it easy to reject old cookies.

The Hash part of the Server Cookie is the hard-to-guess part. In the beta release of BIND, its computation can be configured to use AES, HMAC-SHA1, or, as shown below, HMAC-SHA256:

```
      hash =
          HMAC-SHA256-64 ( Server Secret,
              (Client Cookie | nonce | time | client IP Address) )
```

where "|" represents concatenation.

Author's Address

Donald E. Eastlake 3rd
Huawei Technologies
155 Beaver Street
Milford, MA 01757 USA

Telephone: +1-508-333-2270
EMail: d3e3e3@gmail.com

Mark Andrews
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063 USA

Email: marka@isc.org

Copyright, Disclaimer, and Additional IPR Provisions

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: December 25, 2014

L. Howard
Time Warner Cable

June 26, 2014

Reverse DNS in IPv6 for Internet Service Providers
draft-howard-dnsop-ip6rdns-00

Abstract

In IPv4, Internet Service Providers (ISPs) commonly provide IN-ADDR.ARPA. information for their customers by prepopulating the zone with one PTR record for every available address. This practice does not scale in IPv6. This document analyzes different approaches for ISPs to manage the ip6.arpa zone for IPv6 address space assigned to many customers.

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 December 25, 2014.

Table of Contents

1. Introduction

Best practice [RFC1033] is that "Every Internet-reachable host should have a name"[RFC1912]that is recorded with a PTR resource record in the .ARPA zone. Many network services perform a PTR lookup on the source address of incoming packets before performing services.

Individual Internet users in the residential or consumer scale, including small and home businesses, are constantly joining or moving on the Internet. For large Internet service providers who serve residential users, maintenance of individual PTR records is often impractical. Administrators at ISPs should evaluate methods for responding to reverse DNS queries in IPv6.

1.1. Reverse DNS in IPv4

ISPs that provide access to many residential users typically assign one or a few IPv4 addresses to each of those users, and populate an IN-ADDR.ARPA zone with one PTR record for every IPv4 address. Some ISPs also configure forward zones with matching A records, so that lookups match. For instance, if an ISP Example.com aggregated 192.0.2.0/24 at a network hub in Town in the province of AnyWhere, the reverse zone might look like:

```
1.2.0.192.IN-ADDR.ARPA.  IN PTR 1.user.town.AW.example.com.

2.2.0.192.IN-ADDR.ARPA.  IN PTR 2.user.town.AW.example.com.

3.2.0.192.IN-ADDR.ARPA.  IN PTR 3.user.town.AW.example.com.

.

.

.

254.2.0.192.IN-ADDR.ARPA.  IN PTR 254.user.town.AW.example.com.
```

The conscientious Example.com might then also have a zone:

```
1.user.town.AW.example.com.  IN A 192.0.2.1

2.user.town.AW.example.com.  IN A 192.0.2.2

3.user.town.AW.example.com.  IN A 192.0.2.3

.
```

```
.

.

254.user.town.AW.example.com.  IN A 192.0.2.254
```

Many ISPs generate PTR records for all IP addresses used for customers, and many create the matching A record.

1.2. Reverse DNS Considerations in IPv6

The length of individual addresses makes manual zone entries cumbersome. A sample entry for 2001:0db8:0f00:0000:0012:34ff:fe56:789a might be:

```
a.9.8.7.6.5.e.f.f.f.4.3.2.1.0.0.0.0.0.0.0.0.f.0.8.b.d.0.1.0.0.2
.IP6.ARPA.  IN PTR 1.user.town.AW.example.com.
```

Since 2^^80 possible addresses could be configured in the 2001:db8:f00/48 zone alone, it is impractical to write a zone with every possible address entered. If 1000 entries could be written per second, the zone would still not be complete after 38 trillion years.

Furthermore, since the 64 bits in the host portion of the address are frequently assigned using SLAAC [RFC4862] when the host comes online, it is not possible to know which addresses may be in use ahead of time.

[RFC1912]is an informational document that says "PTR records must point back to a valid A record" and further that the administrator should "Make sure your PTR and A records match."[RFC1912] DNS administrators of residential ISPs should consider how to follow this advice for AAAA and PTR RRs in the residential ISP.

2. Alternatives in IPv6

Several options exist for providing reverse DNS in IPv6. All of these options also exist for IPv4, but the scaling problem is much less severe in IPv4. Each option should be evaluated for its scaling ability, its compliance with existing standards and best practices, and its availability in common systems.

2.1. Negative Response

Some ISP DNS administrators may choose to provide only a NXDomain response to PTR queries for subscriber addresses. In some ways, this is the most accurate response, since no name information is known

about the host. Providing a negative response in response to PTR queries does not satisfy the expectation in [RFC1912] for entries to match. Users of services which are dependent on a successful lookup will have a poor experience. For instance, some web services and SSH connections wait for a DNS response, even NXDOMAIN, before responding. For best user experience, then, it is important to return a response, rather than have a lame delegation. On the other hand, external mail servers are likely to reject connections, which might be an advantage in fighting spam. DNS administrators should consider the uses for reverse DNS records and the number of services affecting the number of users when evaluating this option.

2.2. Wildcard match

The use of wildcards in the DNS is described in [RFC4592], and their use in IPv6 reverse DNS is described in [RFC4472].

While recording all possible addresses is not scalable, it may be possible to record a wildcard entry for each prefix assigned to a customer. Consider also that "inclusion of wildcard NS RRSets in a zone is discouraged, but not barred." [RFC4035]

This solution generally scales well. However, since the response will match any address in the wildcard range (/48, /56, /64, etc.), a forward DNS lookup on that response given will not be able to return the same hostname. This method therefore fails the expectation in [RFC1912] for forward and reverse to match. DNSsec [RFC4035] scalability is limited to signing the wildcard zone, which may be satisfactory.

2.3. Dynamic DNS

One way to ensure forward and reverse records match is for hosts to update DNS servers dynamically, once interface configuration (whether SLAAC, DHCPv6, or other means) is complete, as described in [RFC4472]. Hosts would need to provide both AAAA and PTR updates, and would need to know which servers would accept the information.

This option should scale as well or as poorly as IPv4 dynamic DNS does. Dynamic DNS may not scale effectively in large ISP networks which have no single master name server, but a single master server is not best practice. The ISP’s DNS system may provide a point for Denial of Service attacks, including many attempted dDNS updates. Accepting updates only from authenticated sources may mitigate this risk, but only if authentication itself does not require excessive overhead. No authentication of dynamic DNS updates is inherently provided; implementers should consider use of TSIG [RFC2845], or at least ingress filtering so updates are only accepted from customer

address space from internal network interfaces, rate limit the number of updates from a customer per second, and consider impacts on scalability. UDP is allowed per [RFC2136] so transmission control is not assured, though the host should expect an ERROR or NOERROR message from the server [RFC2136]; TCP provides transmission control, but the updating host would need to be configured to use TCP.

Administrators should consider what domain will contain the records, and who will provide the names. If subscribers provide hostnames, they may provide inappropriate strings. Consider "ihate.example.com" or "badword.customer.example.com" or "celebrityname.committed.illegal.acts.example.com."

There is no assurance of uniqueness if multiple hosts try to update with the same name ("mycomputer.familyname.org"). There is no standard way to indicate to a host what server it should send dDNS updates to.

2.3.1. Dynamic DNS from Individual Hosts

In the simplest case, a residential user will have a single host connected to the ISP. Since the typical residential user cannot configure IPv6 addresses and resolving name servers on their hosts, the ISP should provide address information conventionally (i.e., their normal combination of RAs, DHCP, etc.), and should provide a DNS Recursive Name Server and Domain Search List as described in [RFC3646] or [RFC6106]. In determining its Fully Qualified Domain Name, a host will typically use a domain from the Domain Search List. This is an overloading of the parameter; multiple domains could be listed, since hosts may need to search for unqualified names in multiple domains, without necessarily being a member of those domains. Administrators should consider whether the domain search list actually provides an appropriate DNS suffix(es) when considering use of this option. For purposes of dynamic DNS, the host would concatenate its local hostname (e.g., "hostname") plus the domain(s) in the Domain Search List (e.g., "customer.example.com"), as in "hostname.customer.example.com."

Once it learns its address, and has a resolving name server, the host must perform an SOA lookup on the ip6.arpa record to be added, to find the owner, which will lead to the SOA record. Several recursive lookups may be required to find the longest prefix which has been delegated. The DNS administrator must designate the Primary Master Server for the longest match required. Once found, the host sends dynamic AAAA and PTR updates using the concatenation defined above ("hostname.customer.example.com").

In order to use this alternative, hosts must be configured to use dynamic DNS. This is not default behavior for many hosts, which is an inhibitor for the large ISP. This option may be scalable, although registration following an outage may cause signficant load, and hosts using privacy extensions [RFC4941] may update records daily. It is up to the host to provide matching forward and reverse records, and to update them when the address changes.

2.3.2. Dynamic DNS through Residential Gateways

Residential customers may have a gateway, which may provide DHCPv6 service to hosts from a delegated prefix. ISPs should provide a DNS Recursive Name Server and Domain Search List to the gateway, as described above and in[RFC3646] and [RFC6106]. There are two options for how the gateway uses this information. The first option is for the gateway to respond to DHCPv6 requests with the same DNS Recursive Name Server and Domain Search List provided by the ISP. The alternate option is for the gateway to relay dynamic DNS updates from hosts to the servers and domain provided by the ISP. Host behavior is unchanged; they should provide updates to the ISP's servers as described above.

2.3.3. Automatic DNS Delegations

An ISP may delegate authority for a subdomain such as "customer12345.town.AW.customer.example.com" or "customer12345.example.com" to the customer's gateway. Each domain thus delegated must be unique within the DNS. The ISP may also then delegate the ip6.arpa zone for the prefix delegated to the customer, as in (for 2001:db8:f00::/48) "0.0.f.0.8.b.d.0.1.0.0.2.ip6.arpa." Then the customer could provide updates to their own gateway, with forward and reverse. However, individual hosts connected directly to the ISP rarely have the capability to run DNS for themselves; therefore, an ISP can only delegate to customers with gateways capable of being authoritative name servers. If a device requests a DHCPv6 Prefix Delegation, that may be considered a reasonably reliable indicator that it is a gateway, rather than an individual host. It is not necessarily an indicator that the gateway is capable of providing DNS services, and therefore cannot be relied upon as a way to test whether this option is feasible. In fact, this kind of delegation will not work for devices complying with [RFC6092], which includes the requirement, "By DEFAULT, inbound DNS queries received on exterior interfaces MUST NOT be processed by any integrated DNS resolving server."

If the customer's gateway is the name server, it provides its own information to hosts on the network, as often done for enterprise networks, and as described in [RFC2136].

An ISP may elect to provide authoritative responses as a secondary server to the customer's primary server. For instance, the home gateway name server could be the master server, with the ISP providing the only published NS authoritative servers.

To implement this alternative, users' residential gateways must be capable of acting as authoritative name servers capable of dynamic DNS updates. There is no mechanism for an ISP to dynamically communicate to a user's equipment that a zone has been delegated, so user action would be required. Most users have neither the equipment nor the expertise to run DNS servers, so this option is unavailable to the residential ISP.

2.3.4. Generate Dynamic Records

An ISP's name server that receives a dynamic forward or reverse DNS update may create a matching entry. Since a host capable of updating one is generally capable of updating the other, this should not be required, but redundant record creation will ensure a record exists. ISPs implementing this method should check whether a record already exists before accepting or creating updates.

This method is also dependent on hosts being capable of providing dynamic DNS updates, which is not default behavior for many hosts.

2.3.5. Populate from DHCP Server

A ISP's DHCPv6 server may populate the forward and reverse zones when the DHCP request is received, if the request contains enough information. [RFC4704]

However, this method will only work for a single host address (IA_NA); the ISP's DHCP server would not have enough information to update all records for a prefix delegation. If the zone authority is delegated to a home gateway which used this method, the gateway could update records for residential hosts. To implement this alternative, users' residential gateways would have to support the FQDN DHCP option, and would have to either have the zones configured, or send dDNS messages to the ISP's name server.

2.3.6. Populate from RADIUS Server

A user may receive an address or prefix from a RADIUS [RFC2865] server, the details of which may be recorded via RADIUS Accounting [RFC2866] data. The ISP may populate the forward and reverse zones from the accounting data if it contains enough information. This solution allows the ISP to populate data concerning allocated prefixes (as per 2.2 (wildcards)) and CPE endpoints, but as with 2.3.5 does not allow the ISP to populate information concerning individual hosts.

2.4. Delegate DNS

For customers who are able to run their own DNS servers, such as commercial customers, often the best option is to delegate the reverse DNS zone to them, as described in [RFC2317] (for IPv4). However, since most residential users have neither the equipment nor the expertise to run DNS servers, this method is unavailable to residential ISPs.

This is a general case of the specific case described in Section 2.3.3. All of the same considerations still apply.

2.5. Dynamically Generate PTR When Queried ("On the Fly")

Common practice in IPv4 is to provide PTR records for all addresses, regardless of whether a host is actually using the address. In IPv6, ISPs may generate PTR records for all IPv6 addresses as the records are requested. Configuring records "on the fly" may consume more processor resource than other methods, but only on demand. A denial of service is therefore possible, which may be mitigated with rate-limiting and normal countermeasures.

An ISP using this option should generate a PTR record on demand, and cache or prepopulate the forward (AAAA) entry for the duration of the time-to-live of the PTR. Similarly, the ISP would prepopulate the PTR following a AAAA query. Alternatively, if an algorithm is used to generate unique name, it can be employed on the fly in both directions. This option has the advantage of assuring matching forward and reverse entries, while being simpler than dynamic DNS. Administrators should consider whether the lack of user-specified hostnames is a drawback.

This method may not scale well in conjunction with DNSsec [RFC4035], because of the additional load, but since keys may be pregenerated for zones, and not for each record, the risk is moderate. Signing records on the fly may increase load, and may not scale; unsigned records can indicate that these records are less trusted, which might

be acceptable.

Another consideration is that the algorithm used for generating the record must be the same on all servers for a zone. In other words, any server for the zone must produce the same response for a given query. Administrators managing a variety of rules within a zone might find it difficult to keep those rules synchronized on all servers.

3. Recommendations

The best accuracy would be achieved if ISPs delegate authority along with address delegation, but residential users rarely have domain names or authoritative name servers.

Dynamic DNS updates can provide accurate data, but there is no standard way to indicate to residential devices where to send updates, if the hosts support it, and if it scales.

An ISP has no knowledge of its residential users' hostnames, and therefore can either provide a wildcard response, a dynamically generated response, or a negative response. A valid negative response (such as NXDomain) is a valid response; lame delegation should be avoided.

4. Security Considerations

4.1. Using Reverse DNS for Security

Some people think the existence of reverse DNS records, or matching forward and reverse DNS records, provides useful information about the hosts with those records. For example, one might infer that the administrator of a network with properly configured DNS records was better-informed, and by further inference more responsible, than the administrator of a less-thoroughly configured network. For instance, most email providers will not accept incoming connections on port 25 unless forward and reverse DNS entries match. If they match, but information higher in the stack (for instance, mail source) is inconsistent, the packet is questionable. These records may be easily forged though, unless DNSsec or other measures are taken. The string of inferences is questionable, and may become unneeded if other means for evaluating trustworthiness (such as positive reputations) become predominant in IPv6.

Providing location information in PTR records is useful for troubleshooting, law enforcement, and geolocation services, but for the same reasons can be considered sensitive information.

4.2. DNS Security with Dynamic DNS

Security considerations of using dynamic DNS are described in [RFC3007]. DNS Security Extensions are documented in [RFC4033].

Interactions with DNSsec are described throughout this document.

4.3. Considerations for Other Uses of the DNS

Several methods exist for providing encryption keys in the DNS. Any of the options presented here may interfere with these key techniques.

5. Acknowledgements

The author would like to thank Alain Durand, JINMEI Tatuya, David Freedman, Andrew Sullivan, Chris Griffiths, Darryl Tanner, Ed Lewis, John Brzozowski, Chris Donley, Wes George, Jason Weil, John Spence, Ted Lemon, Stephen Lagerholm, Steinar Haug, Mark Andrews, and Chris Roosenraad and many others who discussed and provided suggestions for this document.

6. IANA Considerations

There are no IANA considerations or implications that arise from this document.

7. References

7.1. Normative References

[RFC1033] Lottor, M., "Domain Administrators Operators Guide", November 1987.

[RFC1912] Barr, D., "Common DNS Operational and Configuration Errors", February 1996.

[RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", April 1917.

[RFC2845] "Secret Key Transaction Authentication for DNS (TSIG)".

[RFC2865] "Remote Authentication Dial In User Service (RADIUS)".

[RFC2866] "RADIUS Accounting".

[RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", November 2000.

[RFC3646] Droms, R., Ed., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", December 2003.

[RFC4033] "DNS Security Introduction and Requirements".

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

[RFC4592] Lewis, E., "The Role of Wildcards in the Domain Name System", July 2006.

[RFC4704] Stapp, M., Volz, Y., and Y. Rekhter, "The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option".

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", September 2007.

[RFC4941] "Privacy Extensions for Stateless Address Autoconfiguration in IPv6".

[RFC6106] "IPv6 Router Advertisement Options for DNS Configuration".

7.2. Informative References

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

[RFC2535] Eastlake, D., "Domain Name System Security Extensions", March 1999.

[RFC4472] Durand, A., Ihren, J., and P. Savola, "Operational Considerations and Issues with IPv6 DNS", April 2006.

[RFC6092] Woodyatt, J., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", January 2011.

[inaddr-reqd] Senie, D., "draft-ietf-dnsop-inaddr-required-07",

August 2005.

[rmap-consider]
Senie, D. and A. Sullivan,
"draft-ietf-dnsop-reverse-mapping-considerations-06",
March 2008.

Author's Address

Lee Howard
Time Warner Cable
13820 Sunrise Valley Drive
Herndon, VA 20171
US

Email: lee.howard@twcable.com

dnsop
Internet-Draft
Intended status: Standards Track
Expires: August 25, 2016

P. Wouters
Red Hat
J. Abley
Dyn, Inc.
S. Dickinson
Sinodun
R. Bellis
ISC

February 22, 2016

The edns-tcp-keepalive EDNS0 Option
draft-ietf-dnsop-edns-tcp-keepalive-06

Abstract

DNS messages between clients and servers may be received over either UDP or TCP. UDP transport involves keeping less state on a busy server, but can cause truncation and retries over TCP. Additionally, UDP can be exploited for reflection attacks. Using TCP would reduce retransmits and amplification. However, clients commonly use TCP only for retries and servers typically use idle timeouts on the order of seconds.

This document defines an EDNS0 option ("edns-tcp-keepalive") that allows DNS servers to signal a variable idle timeout. This signalling encourages the use of long-lived TCP connections by allowing the state associated with TCP transport to be managed effectively with minimal impact on the DNS transaction time.

Table of Contents

1. Introduction

DNS messages between clients and servers may be received over either UDP or TCP [RFC1035]. Historically, DNS clients used API's that only facilitated sending and receiving a single query over either UDP or TCP. New APIs and deployment of DNSSEC validating resolvers on hosts that in the past were using stub resolving only is increasing the DNS client base that prefer using long lived TCP connections. Long-lived TCP connections can result in lower request latency than the case where UDP transport is used and truncated responses are received. This is because clients that retry over TCP following a truncated UDP response typically only use the TCP session for a single (request, response) pair, continuing with UDP transport for subsequent queries.

The use of TCP transport requires state to be retained on DNS servers. If a server is to perform adequately with a significant query load received over TCP, it must manage its available resources to ensure that all established TCP sessions are well-used, and idle connections are closed after an appropriate amount of time.

UDP transport is stateless, and hence presents a much lower resource burden on a busy DNS server than TCP. An exchange of DNS messages over UDP can also be completed in a single round trip between communicating hosts, resulting in optimally-short transaction times. UDP transport is not without its risks, however.

A single-datagram exchange over UDP between two hosts can be exploited to enable a reflection attack on a third party. Response Rate Limiting [RRL] is designed to help mitigate such attacks against authoritative-only servers. One feature of RRL is to let some amount of responses "slip" through the rate limiter. These are returned with the TC (truncation) bit set, which causes legitimate clients to re-query using TCP transport.

[RFC1035] specified a maximum DNS message size over UDP transport of 512 bytes. Deployment of DNSSEC [RFC4033] and other protocols subsequently increased the observed frequency at which responses exceed this limit. EDNS0 [RFC6891] allows DNS messages larger than 512 bytes to be exchanged over UDP, with a corresponding increased incidence of fragmentation. Fragmentation is known to be problematic in general, and has also been implicated in increasing the risk of cache poisoning attacks [fragmentation-considered-poisonous].

TCP transport is less susceptible to the risks of fragmentation and reflection attacks. However, TCP transport for DNS as currently

deployed has expensive setup overhead, compared to using UDP (when no retry is required).

The overhead of the three-way TCP handshake for a single DNS transaction is substantial, increasing the transaction time for a single (request, response) pair of DNS messages from 1 x RTT to 2 x RTT. There is no such overhead for a session that is already established therefore the overhead of the initial TCP handshake is minimised when the resulting session is used to exchange multiple DNS message pairs over a single session. The extra RTT time for session setup can be represented as the equation (1 + N)/N, where N represents the number of DNS message pairs that utilize the session and the result approaches unity as N increases.

With increased deployment of DNSSEC and new RRtypes containing application specific cryptographic material, there is an increase in the prevalence of truncated responses received over UDP with retries over TCP. The overhead for a DNS transaction over UDP truncated due to RRL is 3x RTT, higher than the overhead imposed on the same transaction initiated over TCP.

This document proposes a signalling mechanism between DNS clients and servers that encourages the use of long-lived TCP connections by allowing the state associated with TCP transport to be managed effectively with minimal impact on the DNS transaction time.

This mechanism will be of benefit both for stub-resolver and resolver-authoritative TCP connections. In the latter case the persistent nature of the TCP connection can provide improved defence against attacks including DDoS.

The reduced overhead of this extension adds up significantly when combined with other EDNS0 extensions, such as [CHAIN-QUERY] and [DNS-over-TLS]. For example, the combination of these EDNS0 extensions make it possible for hosts on high-latency mobile networks to natively and efficiently perform DNSSEC validation and encrypt queries.

2. 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].

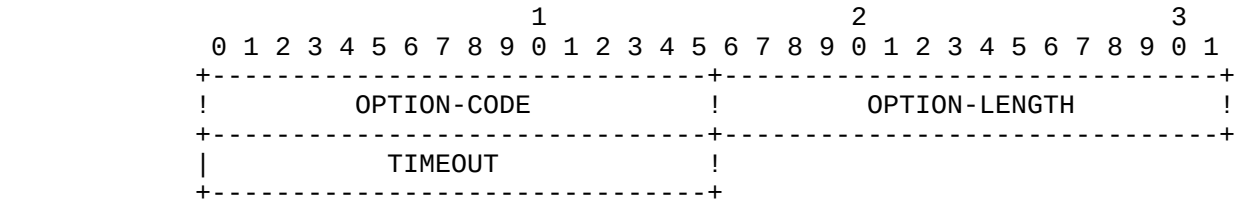
3. The edns-tcp-keepalive Option

This document specifies a new EDNS0 [RFC6891] option, edns-tcp-keepalive, which can be used by DNS clients and servers to signal a willingness to keep an idle TCP session open to conduct future DNS transactions, with the idle timeout being specified by the server. This specification does not distinguish between different types of DNS client and server in the use of this option.

[DRAFT-5966bis] defines an 'idle' DNS-over-TCP session from both the client and server perspective. The idle timeout described here begins when the idle condition is met per that definition and should be reset when that condition is lifted i.e. when a client sends a message or when a server receives a message on an idle connection.

3.1. Option Format

The edns-tcp-keepalive option is encoded as follows:



where:

OPTION-CODE: the EDNS0 option code assigned to edns-tcp-keepalive, TBD1

OPTION-LENGTH: the value 0 if the TIMEOUT is omitted, the value 2 if it is present;

TIMEOUT: an idle timeout value for the TCP connection, specified in units of 100 milliseconds, encoded in network byte order.

3.2. Use by DNS Clients

3.2.1. Sending Queries

DNS clients MUST NOT include the edns-tcp-keepalive option in queries sent using UDP transport.

DNS clients MAY include the edns-tcp-keepalive option in the first query sent to a server using TCP transport to signal their desire to keep the connection open when idle.

DNS clients MAY include the edns-tcp-keepalive option in subsequent queries sent to a server using TCP transport to signal their continued desire to keep the connection open when idle.

Clients MUST specify an OPTION-LENGTH of 0 and omit the TIMEOUT value.

3.2.2. Receiving Responses

A DNS client that receives a response using UDP transport that includes the edns-tcp-keepalive option MUST ignore the option.

A DNS client that receives a response using TCP transport that includes the edns-tcp-keepalive option MAY keep the existing TCP session open when it is idle. It SHOULD honour the timeout received in that response (overriding any previous timeout) and initiate close of the connection before the timeout expires.

A DNS client that receives a response that includes the edns-tcp-keepalive option with a TIMEOUT value of 0 SHOULD send no more queries on that connection and initiate closing the connection as soon as it has received all outstanding responses.

A DNS client that sent a query containing the edns-keepalive-option but receives a response that does not contain the edns-keepalive-option SHOULD assume the server does not support keepalive and behave following the guidance in [DRAFT-5966bis]. This holds true even if a previous edns-keepalive-option exchange occurred on the existing TCP connection.

3.3. Use by DNS Servers

3.3.1. Receiving Queries

A DNS server that receives a query using UDP transport that includes the edns-tcp-keepalive option MUST ignore the option.

A DNS server that receives a query using TCP transport that includes the edns-tcp-keepalive option MAY modify the local idle timeout associated with that TCP session if resources permit.

3.3.2. Sending Responses

A DNS server that receives a query sent using TCP transport that includes an OPT RR (with or without the edns-tcp-keepalive option) MAY include the edns-tcp-keepalive option in the response to signal the expected idle timeout on a connection. Servers MUST specify the TIMEOUT value that is currently associated with the TCP session. It

is reasonable for this value to change according to local resource constraints. The DNS server SHOULD send a edns-tcp-keepalive option with a timeout of 0 if it deems its local resources are too low to service more TCP keepalive sessions, or if it wants clients to close currently open connections.

3.4. TCP Session Management

Both DNS clients and servers are subject to resource constraints which will limit the extent to which TCP sessions can persist. Effective limits for the number of active sessions that can be maintained on individual clients and servers should be established, either as configuration options or by interrogation of process limits imposed by the operating system. Servers that implement edns-tcp-keepalive should also engage in TCP connection management by recycling existing connections when appropriate, closing connections gracefully and managing request queues to enable fair use.

In the event that there is greater demand for TCP sessions than can be accommodated, servers may reduce the TIMEOUT value signalled in successive DNS messages to minimise idle time on existing sessions. This also allows, for example, clients with other candidate servers to query to establish new TCP sessions with different servers in expectation that an existing session is likely to be closed, or to fall back to UDP.

Based on TCP session resources servers may signal a TIMEOUT value of 0 to request clients to close connections as soon as possible. This is useful when server resources become very low or a denial-of-service attack is detected and further maximises the shifting of TIME_WAIT state to well-behaved clients.

However it should be noted that RCF6891 states:

> Lack of presence of an OPT record in a request MUST be taken as an indication that the requestor does not implement any part of this specification and that the responder MUST NOT include an OPT record in its response.

Since servers must be faithful to this specification even on a persistent TCP connection it means that (following the initial exchange of timeouts) a server may not be presented with the opportunity to signal a change in the idle timeout associated with a connection if the client does not send any further requests containing EDNS0 OPT RRs. This limitation makes persistent connection handling via an initial idle timeout signal more attractive than a mechanism that establishes default persistence and

then uses a connection close signal (in a similar manner to HTTP 1.1 [RFC7320]).

If a client includes the edns-tcp-keepalive option in the first query, it SHOULD include an EDNS0 OPT RR periodically in any further messages it sends during the TCP session. This will increase the chance of the client being notified should the server modify the timeout associated with a session. The algorithm for choosing when to do this is out of scope of this document and is left up to the implementor and/or operator.

DNS clients and servers MAY close a TCP session at any time in order to manage local resource constraints. The algorithm by which clients and servers rank active TCP sessions in order to determine which to close is not specified in this document.

3.5. Non-Clean Paths

Many paths between DNS clients and servers suffer from poor hygiene, limiting the free flow of DNS messages that include particular EDNS0 options, or messages that exceed a particular size. A fallback strategy similar to that described in [RFC6891] section 6.2.2 SHOULD be employed to avoid persistent interference due to non-clean paths.

3.6. Anycast Considerations

DNS servers of various types are commonly deployed using anycast [RFC4786].

Changes in network topology between clients and anycast servers may cause disruption to TCP sessions making use of edns-tcp-keepalive more often than with TCP sessions that omit it, since the TCP sessions are expected to be longer-lived. It might be possible for anycast servers to avoid disruption due to topology changes by making use of TCP multipath [RFC6824] to anchor the server side of the TCP connection to an unambiguously-unicast address.

4. Intermediary Considerations

It is RECOMMENDED that DNS intermediaries which terminate TCP connections implement edns-tcp-keepalive. An intermediary that does not implement edns-tcp-keepalive but sits between a client and server that both support edns-tcp-keepalive might close idle connections unnecessarily.

5. Security Considerations

The edns-tcp-keepalive option can potentially be abused to request large numbers of long-lived sessions in a quick burst. When a DNS Server detects abusive behaviour, it SHOULD immediately close the TCP connection and free the resources used.

Servers could choose to monitor client behaviour with respect to the edns-tcp-keepalive option to build up profiles of clients that do not honour the specified timeout.

Readers are advised to familiarise themselves with the security considerations outlined in [DRAFT-5966bis]

6. IANA Considerations

The IANA is directed to assign an EDNS0 option code for the edns-tcp-keepalive option from the DNS EDNS0 Option Codes (OPT) registry as follows:

Value	Name	Status	Reference
TBD1	edns-tcp-keepalive	Standard	[This document]

7. Acknowledgements

The authors acknowledge the contributions of Jinmei TATUYA and Mark Andrews. Thanks to Duane Wessels for detailed review and the many others who contributed to the mailing list discussion.

8. References

8.1. Normative References

[DRAFT-5966bis] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", draft-ietf-dnsop-5966bis (work in progress), January 2016.

[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>.

[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>.

[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>.

[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>.

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <http://www.rfc-editor.org/info/rfc7320>.

8.2. Informative References

[CHAIN-QUERY] Wouters, P., "Chain Query requests in DNS", draft-ietf-dnsop-edns-chain-query (work in progress), January 2016.

[DNS-over-TLS] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "TLS for DNS: Initiation and Performance Considerations", draft-ietf-dprive-dns-over-tls (work in progress), January 2016.

[fragmentation-considered-poisonous] Herzberg, A. and H. Shulman, "Fragmentation Considered Poisonous", arXiv 1205.4011, May 2012, <http://arxiv.org/abs/1205.4011>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <http://www.rfc-editor.org/info/rfc6824>.

[RRL] Vixie, P. and V. Schryver, "DNS Response Rate Limiting (DNS RRL)", ISC-TN 2012-1-Draft1, April 2012, <http://ss.vix.su/~vixie/isc-tn-2012-1.txt>.

Appendix A. Editors' Notes

A.1. Abridged Change History

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

A.1.1. draft-ietf-dnsop-edns-tcp-keepalive-06

Introduction: Moved paragraph 8 to paragraph 2 for readability.

Introduction: clarified that TCP has expensive setup overhead compared to UDP.

Section 3: Add explicit description of the idle timeout.

Section 3.3.2, 1st para: make explicit that query may or may not contain edns-tcp-keepalive option.

Section 3.3.2: remove discussion of intermediary behaviour.

A.1.2. draft-ietf-dnsop-edns-tcp-keepalive-05

Reword Abstract and paragraph 9 in Introduction to remove discussion on balancing UDP/TCP and talk about encouraging use of long-lived TCP sessions.

Section 3.2.2: should -> SHOULD

Changed draft-ietf-dnsop-5966bis to be a normative reference, therefore adding a dependancy on publication of that as RFC.

Reword sentence referring to RFC6824 since it is informational.

Update IANA option to Standard.

Remove last sentence from 1st paragraph of introduction.

Reword paragraph 6 in Introduction, merge paragraph 7 and 8.

Reword Section 3, first sentence to clarify the timeout is specified by the server.

Correct missing URIs in 2 references.

Clarify statement in Section 3.2.2 as how clients should handle updating the timeout when receiving a response.

Reworded first paragraph of Introduction discussing TCP vs (UDP + retry over TCP). Changed 'fallback' to 'retry' in 2 places.

A.1.3. draft-ietf-dnsop-edns-tcp-keepalive-04

Adding wording to sections 3.2.1 and 3.4 to clarify client behaviour on subsequent queries on a TCP connection.

Changed the should to a SHOULD in section 3.2.2

Changed Nameserver to DNS server in section 5.

Updated references.

Changed reference to RFC6824 to be informative.

Corrected reference to requested EDNS0 option code to be 'TBD1'.

A.1.4. draft-ietf-dnsop-edns-tcp-keepalive-03

Clarified that a response to a query with any OPT RR may contain the ends-tcp-keepalive option.

Corrected TIMEOUT length from 4 to 2 in the diagram.

Updated references, including name change of STARTTLS -> DNS-over-TLS and adding reference for cache poisoning.

Updated wording in section on Intermediary Considerations.

Updated wording describing RRL.

Added paragraph to security section describing client behaviour profiles.

Added wording to introduction on use case for stub/resolver/ authoritative.

A.1.5. draft-ietf-dnsop-edns-tcp-keepalive-02

Changed timeout value to idle timeout and re-phrased document around this.

Changed units of timeout to 100ms to allow values less than 1 second.

Change specification to remove use of the option over UDP. This is potentially confusing, could cause issues with ALG's and adds only limited value.

Changed semantics so the client no longer sends a timeout. The client timeout is of limited value as servers should be managing connections based on their view of their resources, not on client requests as this is open to abuse. Additionally this identifies cases were the option is simply being reflected back.

Changed semantics for the meaning of a server sending a timeout of 0. The maximum timeout value of 6553.5s (~1.8h) is already large and a distinct 'connection close'-like signal is potentially more useful.

Added more detail on server side requirements when supporting keepalive in terms of resource and connection management.

Added discussion of EDNS0 per-message limitation and implications of this.

Added reference to STARTTLS draft and RFC7320.

A.1.6. draft-ietf-dnsop-edns-tcp-keepalive-01

Version bump with no changes

A.1.7. draft-ietf-dnsop-edns-tcp-keepalive-00

Clarifications, working group adoption.

A.1.8. draft-wouters-edns-tcp-keepalive-01

Also allow clients to specify KEEPALIVE timeout values, clarify motivation of document.

A.1.9. draft-wouters-edns-tcp-keepalive-00

Initial draft.

Authors' Addresses

Paul Wouters
Red Hat

Email: pwouters@redhat.com

Joe Abley
Dyn, Inc.
470 Moore Street
London, ON N6C 2C2
Canada

Phone: +1 519 670 9327
Email: jabley@dyn.com

Sara Dickinson
Sinodun Internet Technologies
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA
UK

Email: sara@sinodun.com
URI: http://sinodun.com

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

Phone: +1 650 423 1200
Email: ray@isc.org
URI: http://www.isc.org

Domain Name System Operations
Internet-Draft
Intended status: Informational
Expires: April 26, 2015

P. Ebersman
Comcast
C. Griffiths
Dyn
W. Kumari
Google
J. Livingood
Comcast
R. Weber
Nominum

October 23, 2014

Definition and Use of DNSSEC Negative Trust Anchors
draft-livingood-dnsop-negative-trust-anchors-01

Abstract

DNS Security Extensions (DNSSEC) is now entering widespread deployment. However, domain signing tools and processes are not yet as mature and reliable as those for non-DNSSEC-related domain administration tools and processes. Negative Trust Anchors (described in this document) can be used to mitigate DNSSEC validation failures.

[Editor note: This document was originally draft-livingood-negative-trust-anchors-07 - renamved at the request of the DNSOP chairs]

Table of Contents

1. Introduction

The Domain Name System (DNS), DNS Security Extensions (DNSSEC), and related operational practices are defined extensively [RFC1034] [RFC1035] [RFC4033] [RFC4034] [RFC4035] [RFC4398] [RFC4509] [RFC6781] [RFC5155].

This document defines a Negative Trust Anchor, which can be used during the transition to ubiquitous DNSSEC deployment. Negative Trust Anchors (NTAs) are configured locally on a validating DNS recursive resolver to shield end users from DNSSEC-related authoritative name server operational errors. Negative Trust Anchors are intended to be temporary, and should not be distributed by IANA or any other organization outside of the administrative boundary of the organization locally implementing a Negative Trust Anchor. Finally, Negative Trust Anchors pertain only to DNSSEC and not to Public Key Infrastructures (PKI) such as X.509.

DNSSEC has now entered widespread deployment. However, the DNSSEC signing tools and processes are less mature and reliable than those for non-DNSSEC-related administration. As a result, operators of DNS recursive resolvers, such as Internet Service Providers (ISPs), occasionally observe domains incorrectly managing DNSSEC-related resource records. This mismanagement triggers DNSSEC validation failures, and then causes large numbers of end users to be unable to reach a domain. Many end users tend to interpret this as a failure of their ISP or resolver operator, and may switch to a non-validating resolver or contact their ISP to complain, rather than seeing this as a failure on the part of the domain they wanted to reach. Without the techniques in this document, this pressure may cause the resolver operator to disable (or simply not deploy) DNSSEC validation. Use of a Negative Trust Anchor to temporarily disable DNSSEC validation for a specific misconfigured domain name immediately restores access for end users. This allows the domain’s administrators to fix their misconfiguration, while also allowing the organization using the Negative Trust Anchor to keep DNSSEC validation enabled and still reach the misconfigured domain.

2. Definition of a Negative Trust Anchor

Trust Anchors are defined in [RFC5914]. A trust anchor should be used by a validating caching resolver as a starting point for building the authentication chain for a signed DNS response. The inverse of this is a Negative Trust Anchor, which creates a stopping point for a caching resolver to end validation of the authentication chain. Instead, the resolver sends the response as if the zone is unsigned and does not set the AD bit. This Negative Trust Anchor can

potentially be placed at any level within the chain of trust and would stop validation from that point in the chain down.

3. delete

4. Domain Validation Failures

A domain name can fail validation for two general reasons: a legitimate security failure such as due to an attack or compromise of some sort, or as a result of misconfiguration on the part of an domain administrator. As domains transition to DNSSEC the most likely reason for a validation failure will be misconfiguration. Thus, domain administrators should be sure to read [RFC6781] in full. They should also pay special attention to Section 4.2, pertaining to key rollovers, which appear to be the cause of many recent validation failures.

It is also possible that some DNSSEC validation failures could arise due to differences in how different software developers interpret DNSSEC standards and/or how those developers choose to implement support for DNSSEC. For example, it is conceivable that a domain may be DNSSEC signed properly, and one vendor's DNS recursive resolvers will validate the domain but other vendors' software may fail to validate the domain.

5. End User Reaction

End users generally do not know what DNSSEC is, nor should they be expected to at the current time (especially absent widespread integration of DNSSEC indicators in end user software such as web browsers). As a result, end users may misinterpret the failure to reach a domain due to DNSSEC-related misconfiguration . They may (incorrectly) assume that their ISP is purposely blocking access to the domain or that it is a performance failure on the part of their ISP (especially of the ISP's DNS servers). They may contact their ISP to complain, which will incur cost for their ISP. In addition, they may use online tools and sites to complain of this problem, such as via a blog, web forum, or social media site, which may lead to dissatisfaction on the part of other end users or general criticism of an ISP or operator of a DNS recursive resolver.

As end users publicize these failures, others may recommend they switch from security-aware DNS resolvers to resolvers not performing DNSSEC validation. This is a shame since the ISP or other DNS recursive resolver operator is actually doing exactly what they are supposed to do in failing to resolve a domain name, as this is the expected result when a domain can no longer be validated, protecting end users from a potential security threat. Use of a Negative Trust

Anchor would allow the ISP to specifically remedy the failure to reach that domain, without compromising security for other sites. This would result in a satisfied end user, with minimal impact to the ISP, while maintaining the security of DNSSEC for correctly maintained domains.

6. Switching to a Non-Validating Resolver is Not Recommended

As noted in Section 5 some people may consider switching to an alternative, non-validating resolver themselves, or may recommend that others do so. But if a domain fails DNSSEC validation and is inaccessible, this could very well be due to a security-related issue. In order to be as safe and secure as possible, end users should not change to DNS servers that do not perform DNSSEC validation as a workaround, and people should not recommend that others do so either. Domains that fail DNSSEC for legitimate reasons (versus misconfiguration) may be in control of hackers or there could be other significant security issues with the domain.

Thus, switching to a non-validating resolver to restore access to a domain that fails DNSSEC validation is not a recommended practice, is bad advice to others, is potentially harmful to end user security, and is potentially harmful to DNSSEC adoption.

7. Responsibility for Failures

A domain administrator is solely and completely responsible for managing their domain name(s) and DNS resource records. This includes complete responsibility for the correctness of those resource records, the proper functioning of their DNS authoritative servers, and the correctness of DNS records linking their domain to a top-level domain (TLD) or other higher level domain. Even in cases where some error may be introduced by a third party, whether that is due to an authoritative server software vendor, software tools vendor, domain name registrar, or other organization, these are all parties that the domain administrator has selected or approved, and therefore is responsible for managing successfully.

There are some cases in which the domain administrator is not the same as the domain owner. In those cases, a domain owner has delegated operational responsibility to the domain administrator. So no matter whether a domain owner is also the domain administrator or not, the domain administrator is operationally responsible for the proper configuration operation of the domain.

So in the case of a domain name failing to successfully validate, when this is due to a misconfiguration of the domain, that is the sole responsibility of the domain administrator.

Any assistance or mitigation responses undertaken by other parties to mitigate the misconfiguration of a domain name by a domain administrator, especially operators of DNS recursive resolvers, are optional and at the pleasure of those parties.

8. Use of a Negative Trust Anchor

Technical personnel trained in the operation of DNS servers MUST confirm that a failure is due to misconfiguration, as a similar breakage could have occurred if an attacker gained access to a domain's authoritative servers and modified those records or had the domain pointed to their own rogue authoritative servers. They should also confirm that the domain is not intentionally broken, such as for testing purposes as noted in Section 12. Finally, they should make a reasonable attempt to contact the domain owner of the misconfigured zone, preferably prior to implementing the Negative Trust Anchor.

In the case of a validation failure due to misconfiguration of a TLD or popular domain name (such as a top 100 website), this could make content or services in the affected TLD or domain inaccessible for a large number of users. In such cases, it may be appropriate to use a Negative Trust Anchor as soon as the misconfiguration is confirmed.

Once a domain has been confirmed to fail DNSSEC validation due to a DNSSEC-related misconfiguration, an ISP or other DNS recursive resolver operator may elect to use a Negative Trust Anchor for that domain or sub-domain. This instructs their DNS recursive resolver to temporarily NOT perform DNSSEC validation at or in the misconfigured domain. This immediately restores access to the domain for end users while the domain's administrator corrects the misconfiguration(s). It does not and should not involve turning off validation more broadly.

A Negative Trust Anchor MUST only be used for a limited duration. Implementors SHOULD allow the operator using the Negative Trust Andhor to set an end time and date associated with any Negative Trust Anchor. Optimally this time and date is set in a DNS recursive resolver's configuration, though in the short-term this may also be achieved via other systems or supporting processes. Use of a Negative Trust Anchor MUST NOT be automatic.

Finally, a Negative Trust Anchor SHOULD be used only in a specific domain or sub-domain and MUST NOT affect validation of other names up the authentication chain. For example, a Negative Trust Anchor for zone1.example.com would affect only names at or below zone1.example.com, and validation would still be performed on example.com, .com, and the root (".") . In another example, a Negative Trust Anchor for example.com would affect only names within

example.com, and validation would still be performed on .com, and the root (".")

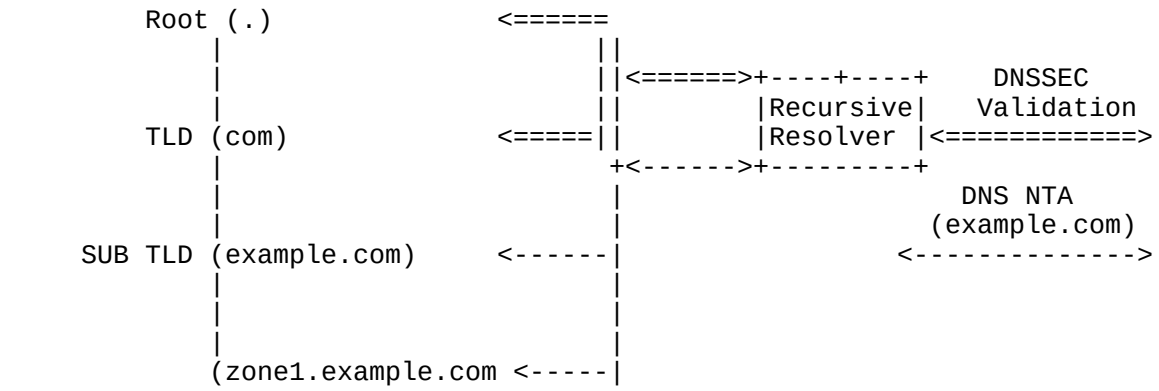


Figure 1: Negative Trust Anchor Diagram

9. Managing Negative Trust Anchors

While Negative Trust Anchors have proven useful during the early stages of DNSSEC adoption, domain owners are ultimately responsible for managing and ensuring their DNS records are configured correctly Section 7.

Most current implementations of DNS validating resolvers currently follow [RFC4033] on defining the implementation of Trust Anchor as either using Delegation Signer (DS), Key Signing Key (KSK), or Zone Signing Key (ZSK). A Negative Trust Anchor should use domain name formatting that signifies where in a delegation a validation process should be stopped.

Different DNS recursive resolvers may have different configuration names for a Negative Trust Anchor. For example, Unbound calls their configuration "domain-insecure."

[need to update reference to full Appendix A, not just unbound]

[Unbound-Configuration]

10. Removal of a Negative Trust Anchor

As explored in Section 13.1, using an NTA once the zone correctly validates can have security considerations. It is therefore recommended that NTA implementors should periodically attempt to validate the domain in question, for the period of time that the Negative Trust Anchor is in place, until such validation is again

successful. Before removing the Negative Trust Anchor, all authoritive resolvers listed in the zone should be checked. Due to AnyCast or load balancers, this may not be possible.

Once all testing succeeds, a Negative Trust Anchor should be removed as soon as is reasonably possible. Optimally this is automatic, though it may also be achieved via other systems or supporting processes.

11. Comparison to Other DNS Misconfigurations

As noted in Section 7 domain administrators are ultimately responsible for managing and ensuring their DNS records are configured correctly. ISPs or other DNS recursive resolver operators cannot and should not correct misconfigured A, CNAME, MX, or other resource records of domains for which they are not authoritative. Expecting non-authoritative entities to protect domain administrators from any misconfiguration of resource records is therefore unrealistic and unreasonable, and in the long-term is harmful to the delegated design of the DNS and could lead to extensive operational instability and/or variation.

12. Intentionally Broken Domains

Some domains, such as dnssec-failed.org, have been intentionally broken for testing purposes [Measuring-DNSSEC-Validation-of-Website-Visitors] [Netalyzr]. For example, dnssec-failed.org is a DNSSEC-signed domain that is broken. If an end user is querying a validating DNS recursive resolver, then this or other similarly intentionally broken domains should fail to resolve and should result in a SERVFAIL error. If such a domain resolved successfully, then it is a sign that the DNS recursive resolver is not fully validating.

Organizations that utilize Negative Trust Anchors should not add a Negative Trust Anchor for any intentionally broken domain.

Organizations operating an intentionally broken domain may wish to consider adding a TXT record for the domain to the effect of "This domain is purposely DNSSEC broken for testing purposes".

13. Other Considerations

13.1. Security Considerations

End to end DNSSEC validation will be disabled during the time that a Negative Trust Anchor is used. In addition, the Negative Trust Anchor may be in place after the point in time when the DNS

misconfiguration that caused validation to break has been fixed. Thus, there may be a gap between when a domain has have been re-secured and when a Negative Trust Anchor is removed. In addition, a Negative Trust Anchor may be put in place by DNS recursive resolver operators without the knowledge of the authoritative domain administrator for a given domain name. However, attempts SHOULD be made to contact and inform the domain administrator prior to putting the NTA in place.

End users of a DNS recursive resolver or other people may wonder why a domain that fails DNSSEC validation resolves with a supposedly validating resolver. As a result, implementors should consider transparently disclosing those Negative Trust Anchors which are currently in place or were in place in the past, such as on a website [Disclosure-Example]. This is particularly important since there is currently no special DNS query response code that could indicate to end users or applications that a Negative Trust Anchor is in place. Such disclosures should optimally include both the data and time that the Negative Trust Anchor was put in place and when it was removed.

13.2. Privacy Considerations

There are no privacy considerations in this document.

13.3. IANA Considerations

There are no IANA considerations in this document.

14. Acknowledgements

Several people made contributions of text to this document and/or played an important role in the development and evolution of this document. This in some cases included performing a detailed review of this document and then providing feedback and constructive criticism for future revisions, or engaging in a healthy debate over the subject of the document. All of this was helpful and therefore the following individuals merit acknowledgement:

- Joe Abley
- John Barnitz
- Tom Creighton
- Marco Davids
- Brian Dickson

- Patrik Falstrom
- Tony Finch
- Chris Ganster
- Olafur Gudmundsson
- Peter Hagopian
- Wes Hardaker
- Paul Hoffman
- Shane Kerr
- Murray Kucherawy
- Warren Kumari
- Rick Lamb
- Marc Lampo
- Ted Lemon
- Ed Lewis
- Antoin Verschuren
- Paul Vixie
- Patrik Wallstrom
- Nick Weaver
- Ralf Weber

15. References

15.1. Normative References

[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.

[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.

[RFC4398] Josefsson, S., "Storing Certificates in the Domain Name System (DNS)", RFC 4398, March 2006.

[RFC4509] Hardaker, W., "Use of SHA-256 in DNSSEC Delegation Signer (DS) Resource Records (RRs)", RFC 4509, May 2006.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, March 2008.

[RFC5914] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", RFC 5914, June 2010.

[RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, December 2012.

15.2. Informative References

[DNSSEC-Validation-Failure-Analysis]
Barnitz, J., Creighton, T., Ganster, C., Griffiths, C., and J. Livingood, "Analysis of DNSSEC Validation Failure - NASA.GOV", Comcast , January 2012, <http://www.dnssec.comcast.net/DNSSEC_Validation_Failure_NASAGOV_20120118_FINAL.pdf>.

[Disclosure-Example]
Comcast, "faa.gov Failing DNSSEC Validation (Fixed)", Comcast , February 2013, <http://dns.comcast.net/index.php/entry/faa-gov-failing-dnssec-validation-fixed>.

[Measuring-DNSSEC-Validation-of-Website-Visitors]
Mens, J., "Is my Web site being used via a DNSSEC-validator?", July 2012, <http://jpmens.net/2012/07/30/is-my-web-site-being-used-via-dnssec-validator/>.

[Netalyzr]
Weaver, N., Kreibich, C., Nechaev, B., and V. Paxson, "Implications of Netalyzr's DNS Measurements", Securing and Trusting Internet Names, SATIN 2011 SATIN 2011, April 2011, <http://conferences.npl.co.uk/satin/presentations/satin2011slides-Weaver.pdf>.

[Unbound-Configuration]
Wijngaards, W., "Unbound: How to Turn Off DNSSEC", June 2010, <http://unbound.net/documentation/howto_turnoff_dnssec.html>.

Appendix A. Configuration Examples

The section contains example configurations to achive Negative Trust Anchor funcationality for the zone foo.example.com.

Please note: These are simply examples - nameserver operators are expected to test and understand the implications of these operations.

A.1. NLNet Labs Unbound

Unbound lets us simply disable validation eching for a specific zone. See: <http://unbound.net/documentation/howto_turnoff_dnssec.html> [TODO(WK): Make this a "real" reference]

```
server:
        domain-insecure: "foo.example.com"
```

A.2. ISC BIND

Use the "rndc" command:

rndc nta [-lifetime duration] [-force] foo.example.com [view]

Set a negative trust anchor, disabling DNSSEC validation for the given domain. Using -lifetime specifies the duration of the NTA, up to one day. Using -force prevents the NTA from expiring before its full lifetime, even if the domain can validate sooner.

A.3. Nominum Vantio

**

negative-trust-anchors

Format: name

Command Channel: view.update name=world negative-trust-anchors=(foo.example.com)

Command Channel: resolver.update name=res1 negative-trust-anchors=(foo.example.com)

Description: Disables DNSSEC validation for a domain, even if the domain is under an existing security root.

Appendix B. Document Change Log

[RFC Editor: This section is to be removed before publication]

Individual-00: First version published as an individual draft.

Individual-01: Fixed minor typos and grammatical nits. Closed all open editorial items.

Individual-02: Simple date change to keep doc from expiring. Substantive updates planned.

Individual-03: Changes to address feedback from Paul Vixie, by adding a new section "Limited Time and Scope of Use". Changes to address issues raised by Antoin Verschuren and Patrik Wallstrom, by adding a new section "Intentionally Broken Domains" and added two related references. Added text to address the need for manual investigation, as suggested by Patrik Falstrom. Added a suggestion on notification as suggested by Marc Lampo. Made several additions and changes suggested by Ralf Weber, Wes Hardaker, Nick Weaver, Tony Finch, Shane Kerr, Joe Abley, Murray Kucherawy, Olafur Gudmundsson.

Individual-04: Moved the section defining a NTA forward, and added new text to the Abstract and Introduction per feedback from Paul Hoffman.

Individual-05: Incorporated feedback from the DNSOP WG list received on 2/17/13 and 2/18/13. This is likely the final version before the IETF 86 draft cutoff date. Updated references to RFC6781 to RFC6781, per March Davids.

Individual-06: Added more OPEN issues to continue tracking WG discussion. No changes in the main document - just expanded issue tracking.

Individual-07: Refresh document - needs revision and rework before IETF-91. Planning to add more contributors.

WG-00: Renamed at request of DNSOP co-chairs, added co-authors

WG-00 (cont):

- Using github issue tracker - go see https://github.com/wkumari/draft-livingood-dnsop-negative-trust-anchors/issues for more details.
- A bunch of readability improvments.
- Issue: Notify the domain owner of the validation failure - resolved.
- Issue: Make the NTA as specific as possible - resolved.

Appendix C. Open Issues

[RFC Editor: This section is to be removed before publication]

Determine whether RFC 2119 language should be used or not when describing things like the duration of a NTA.

The DNSOP WG should discuss whether a 1 day limit is reasonable, whether a different time (more or less than 1 day, such as 1 hour or 1 week) should be specified, or whether no time should be specified (just a recommendation that it SHOULD generally be limited to X).

Olafur Gudmundsson has suggested that we may want to consider whether a non validatable RRSIG should be returned or not when a NTA is in place. This was raised in the context of NLnet Labs' DNSSEC-Trigger, which apparently acts like forwarding stub-validator. He said, "The reason for this is if NTA strips signatures the stub-validator thinks it is under attack and may a) go into recursive mode to try to resolve the domain, getting to the right answer the long way. b) Give the wrong error "Missing signatures" instead of the real error. If all the validator does is not to set the AD bit for RRsets at and below the NTA, stub-resolvers (and cascading resolvers) should be happy."

Determine whether an informative reference to X.509 in the Introduction is necessary.

Is it desirable to say that NTAs should not be distributed across organizational boundaries?

Per Warren Kumari, add examples to appendix. "it would be very helpful to actually show how this is used, with e.g and example in an Appendix, for -insert favorite resolver here-. The document contains a lot of really useful content about why you might use one, how to minimize damage, etc but (IMO) does't do a great job of explaining

how to actually do so". Rick Lamb and Joe Abley also agreed on the need for this.

Per Rick Lamb, "it might be useful to have section 2 "Definition .." make that clear for slow people like me - that the NTA is not an RR and is more of a configuration. Maybe simply replacing "placed" with "implemented" in section 2? "This NTA can potentially be -placed/ implemented- at any level within the chain of trust"

Per Olafur Gudmundsson, address fact that ALL authoritative name servers must be working. "section 10 you talk about possible early removal the NTA when validation succeeds but there may be instances where validation succeeds when using a sub-set of the authoritative servers thus NTA should only be removed if all servers are providing "good" signatures."

Per Olafur Gudmundsson, "Furthermore what to do if some names work but others do not, for example I remember a case where the records at the apex worked but all names below the apex were signed by a key not in the DNSKEY RRset, thus it is possible that either human or automated checks may assume there is no problem when there actually is one. What this is bringing to my mind is maybe you want a new section with guidelines on how to test for failures and in what cases failure justifies NTA and what tests MUST pass before preemttive removal of an NTA."

Per Olafur Gudmundsson, "Also should there be guidance that removal of NTA should include cleaning the caches of all RRsets below the name?"

Reference and text per Ed Lewis: One thing that seems to need repeating from time to time is this passage in RFC 4033. ... In the final analysis, however, authenticating both DNS keys and data is a matter of local policy, which may extend or even override the protocol extensions defined in this document set. See Section 5 for further discussion. A responsibility (one of many) of a caching server operator is to "protect the integrity of the cache." DNSSEC is just a tool to help accomplish that. It carries ancillary data that a local cache administrator may use to filter out undesired responses. DNSSEC is not an enforcement mechanism, it’s a resource. When I see folks voice opinions that DNSSEC’s recommended operation has to strictly followed, my gut reaction is that these folks have forgotten the purpose of all of our efforts. We don’t secure protocols to make things work better. We don’t operate the DNS because we like to run a well run machine. We don’t run the Internet for the fun of it. (Some might enjoy running it, that’s job satisfaction to some extent.) At the end of the day all that matters is that what is being done benefits society. We run the Internet to

enrich society. We prefer a well run DNS because it saps less resources than a poorly run DNS. We prefer secure protocols so that people don't become victims (in some sense of the word). Make it work. Do what it takes to make it work. "Local policy" rules.

Per David Conrad: I'd suggest that in the BCP/RFC/whatever, in addition to recommending that NTAs be time capped and not written to permanent storage, it should also recommend NTAs be written as specifically as possible. (Should be obvious, but doesn't hurt to reiterate I suppose).

Per Ralf Weber: Informing the domain owner on the validation failure. There should be a section in the document that the operator deploying an NTA has to inform the domain owner of the problem. (JL note: would prefer to say operator SHOULD take reasonable steps to notify the domain owner, etc.)

Authors' Addresses

Paul Ebersman
Comcast
One Comcast Center
1701 John F. Kennedy Boulevard
Philadelphia, PA 19103
US

Email: ebersman-ietf@dragon.net

Chris Griffiths
Dyn
150 Dow Street
Tower Two
Manchester, NH 03101
US

Email: cgriffiths@gmail.com
URI: http://www.dyn.com

Warren Kumari
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
US

Email: warren@kumari.net
URI: http://www.google.com

Jason Livingood
Comcast
One Comcast Center
1701 John F. Kennedy Boulevard
Philadelphia, PA 19103
US

Email: jason_livingood@cable.comcast.com
URI: http://www.comcast.com

Ralf Weber
Nominum

Email: Ralf.Weber@nominum.com
URI: http://www.nominum.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 30, 2015

W. Kumari
Google
P. Hoffman
VPN Consortium

November 26, 2014

Decreasing Access Time to Root Servers by Running One on Loopback
draft-wkumari-dnsop-root-loopback-02

Abstract

Some DNS recursive resolvers have longer-than-desired round trip times to the closest DNS root server. Such resolvers can greatly decrease the round trip time by running a copy of the full root zone on a loopback address (such as 127.0.0.1). Typically, the vast majority of queries going to the root are for names that do not exist in the root zone, and the negative answers are cached for a much shorter period of time. This document shows how to start and maintain such a copy of the root zone in a manner that is secure for the operator of the recursive resolver and does not pose a threat to other users of the DNS.

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 May 30, 2015.

Table of Contents

1. Introduction

DNS recursive resolvers have to answer all queries from their customers, even those which are for domain names that do not exist. For each queried name that has a top level domain (TLD) that is not in the recursive resolver's cache, the resolver must send a query to a root server to get the information for that TLD, or to find out that the TLD does not exist. If there is a slow path between the recursive resolver and the closest root server, getting slow responses to these queries has a negative effect on the resolver's customers.

This document describes a method for the operator of a recursive resolver to greatly speed these queries. The basic idea is to create an up-to-date root zone server on a loopback address on the same host as the recursive server, and that server is used when the recursive resolver uses for looking up root information. The recursive resolver validates all responses from the root server on the loopback address, just as it would all responses from a remote root server.

The primary goal of this design is to provide faster negative responses to stub resolver queries that contain junk queries. This design will probably have little effect on getting faster positive

responses to stub resolver for good queries on TLDs, because the data for those zones is usually long-lived and already in the cache of the recursive resolver; thus, getting faster positive responses is a non-goal of this design.

This design explicitly only allows the new root zone server to be run on a loopback address. This prevents the server from serving authoritative answers to any system other than the recursive resolver.

This design requires the addition of authoritative name server software running on the same machine as the recursive resolver. Thus, recursive resolver software such as BIND will not need to add much new functionality, but recursive resolver software such as Unbound will need to be able to talk to an authoritative server (such as NSD) running on the same host.

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. Requirements

In order to implement the mechanism described in this document:

- The system MUST be able to validate a zone with DNSSEC.
- The system MUST have an up-to-date copy of the DNS root key.
- The system MUST be able to retrieve a copy of the entire root zone (including all DNSSEC-related records).
- The system MUST be able to run an authoritative server on one of the IPv4 loopback addresses (that is, an address in the range 127/8).

A corollary of the above list is that authoritative data in the root zone used on the local authoritative server MUST be identical to the same data in the root zone for the DNS. It is possible to change the unsigned data (the glue records) in the copy of the root zone, but such changes are likely to cause problems for the recursive server that accesses the local root zone.

3. Operation of the Root Zone on the Loopback Address

The operation of an authoritative server for the root in the system described here can be done separately from the operation of the recursive resolver.

The steps to set up the root zone are:

1. Retrieve a copy of the root zone. (See Appendix A for some current locations of sources.)

2. Start the authoritative server with the root zone on a loopback address that is not in use. This would typically be 127.0.0.1, but if that address is in use, any address in 127/8 is acceptable.

The contents of the root zone must be refreshed using the timers from the SOA record in root zone, as described in [RFC1035]. If the contents of the zone cannot be refreshed before the expire time, the server MUST return a SERVFAIL error response for all queries until the zone can be successfully be set up again.

4. Using the Root Zone Server on the Loopback Address

A recursive resolver that wants to use a root zone server operating as described in Section 3 simply specifies the local address as the place to look when it is looking for information from the root. All responses from the root server must be validated using DNSSEC.

Note that using this configuration will cause the recursive resolver to fail if the local root zone server fails. See Appendix B for more discussion of this for specific software.

To test the proper operation of the recursive resolver with the local root server, use a DNS client to send a query for the SOA of the root to the recursive server. Make sure the response that comes back does not have the AD bit in the message header set.

5. IANA Considerations

This document requires no action from the IANA.

6. Security Considerations

A system that does not follow the DNSSEC-related requirements given in Section 2 can be fooled into giving bad responses in the same way as any recursive resolver that does not do DNSSEC validation on responses from a remote root server.

7. Acknowledgements

The editors fully acknowledge that this is not a new concept, and that we have chatted with many people about this. In fact, this concept may already have been implemented without the knowledge of the authors. For example, Bill Manning described something similar in his doctoral dissertation in 2013.

Evan Hunt contributed greatly to the logic in the requirements. Other significant contributors include Wouter Wijngaards, Tony Hain Doug Barton, and Greg Lindsay.

8. Normative References

[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.

Appendix A. Current Sources of the Root Zone

The root zone can be retrieved from anywhere as long as it comes with all the DNSSEC records needed for validation. Currently, there are three sources of the root zone supported by ICANN:

- From ICANN via FTP at ftp://rs.internic.net/domain/root.zone
- From ICANN via HTTP at http://www.internic.net/domain/root.zone
- From ICANN by AXFR from DNS servers at xfr.lax.dns.icann.org and xfr.cjr.dns.icann.org

Currently, the root can be retrieved by zone transfer (AXFR) from the following root server operators:

- b.root-servers.net
- c.root-servers.net
- f.root-servers.net
- g.root-servers.net
- k.root-servers.net

Appendix B. Example Configurations of Common Implementations

This section shows fragments of configurations for some popular recursive server software that is believed to correctly implement the requirements given in this document.

The IPv4 and IPv6 addresses in this section were checked recently by testing for AXFR over TCP from each address for the known single-letter names in the root-servers.net zone.

The examples here use a loopback address of 127.12.12.12, but typical installations will use 127.0.0.1. The different address is used in order to emphasize that the root server does not need to be on the device at "localhost".

B.1. Example Configuration: BIND 9.9

BIND acts both as a recursive resolver and an authoritative server. Because of this, there is "fate sharing" between the two servers in the following configuration. That is, if the root server dies, it is likely that all of BIND is dead.

Using this configuration, queries for information in the root zone are returned with the AA bit not set.

When slaving a zone, BIND will treat zone data differently if it is slaved into a separate view (or a separate instance of the software) versus slaving the zone into the same view or instance that is also performing the recursion.

Validation: When using separate views or separate instances, the DS records in the slaved zone will be validated as the zone data is accessed by the recursive server. When using the same view, this validation does not occur for the slaved zone.

Caching: When using separate views or instances, the recursive server will cache all of the queries for the slaved zone, just as it would using the traditional root hints method. Thus, as the zone in the other view or instance is refreshed or updated, changed information will not appear in the recursive server until the TTL of the old record times out. Currently the TTL for DS and delegation NS records is two days. When using the same view, all zone data in the recursive server will be updated as soon as it receives its copy of the zone.

```
view root {
    match-destinations { 127.12.12.12; };
    zone "." {
        type slave;
        file "rootzone.db";
        notify no;
        masters {
            192.228.79.201; # b.root-servers.net
            192.33.4.12;    # c.root-servers.net
            192.5.5.241;    # f.root-servers.net
            192.112.36.4;   # g.root-servers.net
            193.0.14.129;   # k.root-servers.net
            2001:500:84::b; # b.root-servers.net
            2001:500:2f::f; # f.root-servers.net
            2001:7fd::1;    # k.root-servers.net
        };
    };
};

view recursive {
    dnssec-validation auto;
    allow-recursion { any; };
    recursion yes;
    zone "." {
        type static-stub;
        server-addresses { 127.12.12.12; };
    };
};
```

B.2. Example Configuration: Unbound 1.4 and NSD 4

Unbound and NSD are separate software packages. Because of this, there is no "fate sharing" between the two servers in the following configurations. That is, if the root server instance (NSD) dies, the recursive resolver instance (Unbound) will probably keep running, but will not be able to resolve any queries for the root zone. Therefore, the administrator of this configuration might want to carefully monitor the NSD instance and restart it immediately if it dies.

Using this configuration, queries for information in the root zone are returned with the AA bit not set.

```
# Configuration for Unbound
server:
    do-not-query-localhost: no
stub-zone:
    name: "."
    stub-prime: no
    stub-addr: 127.12.12.12

# Configuration for NSD
server:
    ip-address: 127.12.12.12
zone:
    name: "."
    request-xfr: 192.228.79.201 NOKEY # b.root-servers.net
    request-xfr: 192.33.4.12 NOKEY    # c.root-servers.net
    request-xfr: 192.5.5.241 NOKEY    # f.root-servers.net
    request-xfr: 192.112.36.4 NOKEY   # g.root-servers.net
    request-xfr: 193.0.14.129 NOKEY   # k.root-servers.net
    request-xfr: 2001:500:84::b NOKEY # b.root-servers.net
    request-xfr: 2001:500:2f::f NOKEY # f.root-servers.net
    request-xfr: 2001:7fd::1 NOKEY    # k.root-servers.net
```

B.3. Example Configuration: Microsoft Windows Server 2012

Windows Server 2012 contains a DNS server in the "DNS Manager" component. When activated, that component acts as a recursive server. DNS Manager can also act as an authoritative server.

Using this configuration, queries for information in the root zone are returned with the AA bit set.

The steps to configure DNS Manager to implement the requirements in this document are:

1. Launch the DNS Manager GUI. This can be done from the command line ("dnsmgmt.msc") or from the Service Manager (the "DNS" command in the "Tools" menu).

2. In the hierarchy under the server on which the service is running, right-click on the "Forward Lookup Zones", and select "New Zone". This brings up a succession of dialog boxes.

3. In the "Zone Type" dialog box, select "Secondary zone".

4. In the "Zone Name" dialog box, enter ".".

5. In the "Master DNS Servers" dialog box, enter "b.root-servers.net". The system validates that it can do a zone

transfer from that server. (After this configuration is completed, DNS Manager will attempt to transfer from all of the root zone servers.)

6. In the "Completing the New Zone Wizard" dialog box, click "Finish".

7. Verify that the DNS Manager is acting as a recursive resolver. Right-click on the server name in the hierarch, choosing the "Advanced" tab in the dialog box. See that "Disable recursion (also disables forwarders)" is not selected, and that "Enable DNSSEC validation for remote responses" is selected.

Authors' Addresses

Warren Kumari
Google

Email: Warren@kumari.net

Paul Hoffman
VPN Consortium

Email: paul.hoffman@vpnc.org