DNSOP   W. Hardaker
Internet-Draft   Parsons
Intended status: Best Current Practice   O. Gudmundsson
Expires: February 12, 2017   CloudFlare
                                                            S. Krishnaswamy
                                                                   Parsons
                                                           August 11, 2016

                        DNSSEC Roadblock Avoidance
              draft-ietf-dnsop-dnssec-roadblock-avoidance-05.txt

   Abstract

      This document describes problems that a Validating DNS resolver,
      stub-resolver or application might run into within a non-compliant
      infrastructure. It outlines potential detection and mitigation
      techniques. The scope of the document is to create a shared approach
      to detect and overcome network issues that a DNSSEC software/system
      may face.

   Status of This Memo

   Copyright Notice

   to this document. Code Components extracted from this document must
   include Simplified BSD License text as described in Section 4.e of
   the Trust Legal Provisions and are provided without warranty as
   described in the Simplified BSD License.

Table of Contents

1. Introduction

   This document describes problems observable during DNSSEC ([RFC4034],
   [RFC4035]) deployment that derive from non-compliant infrastructure.
   It poses potential detection and mitigation techniques.

1.1. Notation

   In this document a "Host Validator" can either be a validating stub-
   resolver, such as library that an application has linked in, or a
   validating resolver daemon running on the same machine. It may or
   may not be trying to use upstream caching resolvers during its own
   resolution process; both cases are covered by the tests defined in
   this document.

   The sub-variant of this is a "Validating Forwarding Resolver", which
   is a resolver that is configured to use upstream Resolvers when
   possible. A Validating Forward Resolver also needs to perform the
   tests outlined in this document before using an upstream recursive
   resolver.

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

1.2. Background

   Deployment of DNSSEC validation is hampered by network components
   that make it difficult or sometimes impossible for validating
   resolvers to effectively obtain the DNSSEC data they need. This can
   occur for many different reasons including, but not limited to:

   o Because recursive resolvers and DNS proxies [RFC5625] are not
      fully DNSSEC compliant

   o Because resolvers are not DNSSEC aware

   o Because "middle-boxes" actively block, modify and/or restrict
      outbound traffic to the DNS port (53) either UDP and/or TCP .

   o In-path network components do not allow UDP fragments

This document talks about ways that a Host Validator can detect the
state of the network it is attached to, and ways to hopefully
circumvent the problems associated with the network defects it
discovers. The tests described in this document may be performed on
any validating resolver to detect and prevent problems. While these
recommendations are mainly aimed at Host Validators it it prudent to
perform these tests from regular Validating Resolvers before enabling
just to make sure things work.

There are situations where a host can not talk directly to a
Resolver; the tests below can not address how to overcome that, and
inconsistent results can be seen in such cases. This can happen, for
instance, when there are DNS proxies/forwarders between the user and
the actual resolvers.

1.3. Implementation experiences

Multiple lessons learned from multiple implementations led to the
development of this document, including (in alphabetical order)
DNSSEC-Tools' DNSSEC-Check, DNSSEC_Resolver_Check, dnssec-trigger,
FCC_Grade.

Detecting lack of support for specified DNSKEY algorithms and DS
digest algorithms is outside the scope of this document but the
document provides information on how to do that, see sample test
tool: https://github.com/ogud/DNSSEC_ALG_Check

This document does describe compliance tests for algorithms 5, 7 and
13 with DS digest algorithms 1 and 2.

1.3.1. Test Zone Implementation

In this document, the "test.example.com" domain is used to refer to
DNS records which contain test records that have known DNSSEC
properties associated with them. For example, the "badsign-
a.test.example.com" domain is used below to refer to a DNS A record
where the signatures published for it are invalid (i.e., they are
"bad signatures" that should cause a validation failure).

At the time of this publication, the "test.dnssec-tools.org" domain
implements all of these test records. Thus, it may be possible to
replace "test.example.com" in this document with "test.dnssec-
tools.org" when performing real-world tests.

2. Goals

   This document is intended to show how a Host Validator can detect the
   capabilities of a recursive resolver, and work around any problems
   that could potentially affect DNSSEC resolution. This enables the
   Host Validator to make use of the caching functionality of the
   recursive resolver, which is desirable in that it decreases network
   traffic and improves response times.

   A Host Validator has two choices: it can wait to determine that it
   has problems with a recursive resolver based on the results that it
   is getting from real-world queries issued to it, or it can
   proactively test for problems (Section 3) to build a work around list
   ahead of time (Section 5). There are pros and cons to both of these
   paths that are application specific, and this document does not
   attempt to provide guidance about whether proactive tests should or
   should not be used. Either way, DNSSEC roadblock avoidance
   techniques ought to be used when needed and if possible.

   Note: Besides being useful for Host Validators, the same tests can be
   used for a recursive resolver to check if its upstream connections
   hinder DNSSEC validation.

3. Detecting DNSSEC Non-Compliance

   A Host Validator may choose to determine early-on what roadblocks
   exist that may hamper its ability to perform DNSSEC look-ups. This
   section outlines tests that can be done to test certain features of
   the surrounding network.

   These tests should be performed when a resolver determines its
   network infrastructure has changed. Certainly a resolver should
   perform these tests when first starting, but MAY also perform these
   tests when they've detected network changes (e.g. address changes, or
   network reattachment, etc).

   NOTE: when performing these tests against an address, we make the
   following assumption about that address: It is a uni-cast address or
   an any-cast [RFC4786] cluster where all servers have identical
   configuration and connectivity.

   NOTE: when performing these tests we also assume that the path is
   clear of "DNS interfering" middle-boxes, like firewalls, proxies,
   forwarders. Presence of such infrastructure can easily make a
   recursive resolver appear to be improperly performing. It is beyond
   the scope of the document as how to work around such interference,
   although the tests defined in this document may indicate when such
   misbehaving middle-ware is causing interference.

NOTE: This document specifies two sets of tests to perform: a
comprehensive one and a fast one. The fast one will detect most
common problems, thus if the fast one passes then the comprehensive
MAY be considered passed as well.

3.1. Determining DNSSEC support in recursive resolvers

Ideally, a Host Validator can make use of the caching present in
recursive resolvers. This section discusses the tests that a
recursive resolver MUST pass in order to be fully usable as a DNS
cache.

Unless stated otherwise, all of the following tests SHOULD have the
Recursion Desired (RD) flag set when sending out a query and SHOULD
be sent over UDP. Unless otherwise stated, the tests MUST NOT have
the DO bit set or utilize any of the other DNSSEC related
requirements, like EDNS0, unless otherwise specified. The tests are
designed to check for support of one feature at a time.

3.1.1. Supports UDP answers

Purpose: This tests basic DNS over UDP functionality to a resolver.

Test: A DNS request is sent to the resolver under test for an A
record for a known existing domain, such as good-a.test.example.com.

SUCCESS: A DNS response was received that contains an A record in the
answer section. (The data itself does not need to be checked.)

Note: an implementation MAY chose to not perform the rest of the
tests if this test fails, as it is highly unlikely that the resolver
under test will pass any of the remaining tests.

3.1.2. Supports TCP answers

Purpose: This tests basic TCP functionality to a resolver.

Test: A DNS request is sent over TCP to the resolver under test for
an A record for a known existing domain, such as good-
a.test.example.com.

SUCCESS: A DNS response was received that contains an A record in the
answer section. (The data itself does not need to be checked.)

3.1.3. Supports EDNS0

   Purpose: Test whether a resolver properly supports the EDNS0
   extension option.

   Pre-requisite: "Supports UDP or TCP".

   Test: Send a request to the resolver under test for an A record for a
   known existing domain, such as good-a.test.example.com, with an EDNS0
   OPT record in the additional section.

   SUCCESS: A DNS response was received that contains an EDNS0 option
   with version number 0.

3.1.4. Supports the DO bit

   Purpose: This tests whether a resolver has minimal support of the DO
   bit.

   Pre-requisite: "Supports EDNS0".

   Test: Send a request to the resolver under test for an A record for a
   known existing domain such as good-a.test.example.com. Set the DO
   bit in the outgoing query.

   SUCCESS: A DNS response was received that contains the DO bit set.

   Note: this only tests that the resolver sets the DO bit in the
   response. Later tests will determine if the DO bit was actually made
   use of. Some resolvers successfully pass this test because they
   simply copy the unknown flags into the response. These resolvers
   will fail the later tests.

3.1.5. Supports the AD bit DNSKEY algorithm 5 and 8

   Purpose: This tests whether the resolver is a validating resolver.

   Pre-requisite: "Supports the DO bit".

   Test: Send requests to the resolver under test for an A record for a
   known existing domain in a DNSSEC signed zone which is verifiable to
   a configured trust anchor, such as good-a.test.example.com using the
   root's published DNSKEY or DS record as a trust anchor. Set the DO
   bit in the outgoing query. This test should be done twice, once for
   a zone that contains algorithm 5 (RSASHA1) and another for algorithm
   8 (RSASHA256).

   SUCCESS: A DNS response was received that contains the AD bit set for
   algorithm 5 (RSASHA1).

   BONUS: The AD bit is set for a resolver that supports Algorithm 8
   RSASHA256

3.1.6. Returns RRsig for signed answer

   Purpose: This tests whether a resolver will properly return RRSIG
   records when the DO bit is set.

   Pre-requisite: "Supports the DO bit".

   Test: Send a request to the resolver under test for an A record for a
   known existing domain in a DNSSEC signed zone, such as good-
   a.test.example.com. Set the DO bit in the outgoing query.

   SUCCESS: A DNS response was received that contains at least one RRSIG
   record.

3.1.7. Supports querying for DNSKEY records

   Purpose: This tests whether a resolver can query for and receive a
   DNSKEY record from a signed zone.

   Pre-requisite: "Supports the DO bit."

   Test: Send a request to the resolver under test for an DNSKEY record
   which is known to exist in a signed zone, such as test.example.com/
   DNSKEY. Set the DO bit in the outgoing query.

   SUCCESS: A DNS response was received that contains a DNSKEY record in
   the answer section.

   Note: Some DNSKEY RRset's are large and if the network path has
   problems with large answers this query may result in either false
   positive or false negative. In general the DNSKEY queried for should
   be small enough to fit into a 1220 byte answer, to avoid false
   negative result when TCP is disabled. However, querying many zones
   will result in answers greater than 1220 bytes so DNS over TCP MUST
   be available for DNSSEC use in general.

3.1.8. Supports querying for DS records

   Purpose: This tests whether a resolver can query for and receive a DS
   record from a signed zone.

   Pre-requisite: "Supports the DO bit."

Test: Send a request to the resolver under test for an DS record
which is known to exist in a signed zone, such as test.example.com/
DS. Set the DO bit in the outgoing query.

SUCCESS: A DNS response was received that contains a DS record in the
answer section.

3.1.9. Supports negative answers with NSEC records

Purpose: This tests whether a resolver properly returns NSEC records
for a non-existing domain in a DNSSEC signed zone.

Pre-requisite: "Supports the DO bit."

Test: Send a request to the resolver under test for an A record which
is known to not exist in an NSEC signed zone, such as non-
existent.test.example.com. Set the DO bit in the outgoing query.

SUCCESS: A DNS response was received that contains an NSEC record.

Note: The query issued in this test MUST be sent to a NSEC signed
zone. Getting back appropriate NSEC3 records does not indicate a
failure, but a bad test.

3.1.10. Supports negative answers with NSEC3 records

Purpose: This tests whether a resolver properly returns NSEC3 records
([RFC5155]) for a non-existing domain in a DNSSEC signed zone.

Pre-requisite: "Supports the DO bit."

Test: Send a request to the resolver under test for an A record which
is known to be non-existent in a zone signed using NSEC3, such as
non-existent.nsec3-ns.test.example.com. Set the DO bit in the
outgoing query.

SUCCESS: A DNS response was received that contains an NSEC3 record.

Bonus: If the AD bit is set, this validator supports algorithm 7
RSASHA1-NSEC3-SHA1

Note: The query issued in this test MUST be sent to a NSEC3 signed
zone. Getting back appropriate NSEC records does not indicate a
failure, but a bad test.

3.1.11. Supports queries where DNAME records lead to an answer

   Purpose: This tests whether a resolver can query for an A record in a
   zone with a known DNAME referral for the record's parent.

   Test: Send a request to the resolver under test for an A record which
   is known to exist in a signed zone within a DNAME referral child
   zone, such as good-a.dname-good-ns.test.example.com.

   SUCCESS: A DNS response was received that contains a DNAME in the
   answer section. An RRSIG MUST also be received in the answer section
   that covers the DNAME record.

3.1.12. Permissive DNSSEC

   Purpose: To see if a validating resolver is ignoring DNSSEC
   validation failures.

   Pre-requisite: Supports the AD bit.

   Test: ask for data from a broken DNSSEC delegation such as badsign-
   a.test.example.com.

   SUCCESS: A reply was received with the Rcode set to SERVFAIL

3.1.13. Supports Unknown RRtypes

   Purpose: Some DNS Resolvers/gateways only support some RRtypes. This
   causes problems for applications that need recently defined types.

   Pre-requisite: "Supports UDP or TCP".

   Test: Send a request for recently defined type or unknown type in the
   20000-22000 range, that resolves to a server that will return an
   answer for all types, such as alltypes.example.com (a server today
   that supports this: alltypes.res.dnssecready.org)

   SUCCESS: A DNS response was retrieved that contains the type
   requested in the answer section.

3.2. Direct Network Queries

   If need be, a Host Validator may need to make direct queries to
   authoritative servers or known Open Recursive Resolvers in order to
   collect data. To do that, a number of key network features MUST be
   functional.

3.2.1. Support for Remote UDP Over Port 53

   Purpose: This tests basic UDP functionality to outside the local
   network.

   Test: A DNS request is sent to a known distant authoritative server
   for a record known to be within that server's authoritative data.
   Example: send a query to the address of ns1.test.example.com for the
   good-a.test.example.com/A record.

   SUCCESS: A DNS response was received that contains an A record in the
   answer section.

   Note: an implementation can use the local resolvers for determining
   the address of the name server that is authoritative for the given
   zone. The recursive bit MAY be set for this request, but does not
   need to be.

3.2.2. Support for Remote UDP With Fragmentation

   Purpose: This tests if the local network can receive fragmented UDP
   answers

   Pre-requisite: Local UDP traffic > 1500 in size is possible

   Test: A DNS request is sent over UDP to a known distant DNS address
   asking for a record that has answer larger than 2000 bytes. For
   example, send a query for the test.example.com/DNSKEY record with the
   DO bit set in the outgoing query.

   Success: A DNS response was received that contains the large answer.

   Note: A failure in getting large answers over UDP is not a serious
   problem if TCP is working.

3.2.3. Support for Outbound TCP Over Port 53

   Purpose: This tests basic TCP functionality to outside the local
   network.

   Test: A DNS request is sent over TCP to a known distant authoritative
   server for a record known to be within that server's authoritative
   data. Example: send a query to the address of ns1.test.example.com
   for the good-a.test.example.com/A record.

   SUCCESS: A DNS response was received that contains an A record in the
   answer section.

   Note: an implementation can use the local resolvers for determining
   the address of the name server that is authoritative for the given
   zone. The recursive bit MAY be set for this request, but does not
   need to be.

3.3. Support for DNSKEY and DS combinations

   Purpose: These tests can check what algorithm combinations are
   supported.

   Pre-requisite: At least one of above tests has returned the AD bit
   set proving that the upstream is validating

   Test: A DNS request is sent over UDP to the resolver under test for a
   known combination of the DS algorithm number (N) and DNSKEY algorithm
   number (M) of the example form ds-N.alg-M-nsec.test.example.com.
   Examples:

           ds-2.alg-13-nsec.test.example.com TXT
                or
           ds-4.alg-13-nsec3.test.example.com TXT.

   SUCCESS: a DNS response is received with the AD bit set and with a
   matching record type in the answer section.

   Note: for algorithms 6 and 7, NSEC is not defined thus query for alg-
   M-nsec3 is required. Similarly NSEC3 is not defined for algorithms
   1, 3 and 5. Furthermore algorithms 2, 4, 9, 11 do not currently have
   definitions for signed zones.

4. Aggregating The Results

   Some conclusions can be drawn from the results of the above tests in
   an "aggregated" form. This section defines some labels to assign to
   a resolver under test given the results of the tests run against
   them.

4.1. Resolver capability description

   This section will group and label certain common results

   Resolvers are classified into following broad behaviors:

   Validator: The resolver passes all DNSSEC tests and had the AD bit
      appropriately set.

   DNSSEC Aware: The resolver passes all DNSSEC tests, but does not
      appropriately set the AD bit on answers, indicating it is not

validating. A Host Validator will function fine using this resolver as a forwarder.

Non-DNSSEC capable: The resolver is not DNSSEC aware and will make it hard for a Host Validator to operate behind it. It MAY be usable for querying for data that is in known insecure sections of the DNS tree.

Not a DNS Resolver: This is a improperly behaving resolver and not should not be used at all.

While it would be great if all resolvers fell cleanly into one of the broad categories above, that is not the case. For that reason it is necessary to augment the classification with more descriptive result, this is done by adding the word "Partial" in front of Validator/ DNSSEC Aware classifications, followed by sub-descriptors of what is not working.

Unknown: Failed the Unknown test

DNAME: Failed the DNAME test

NSEC3: Failed the NSEC3 test

TCP: TCP not available

SlowBig: UDP is size limited but TCP fallback works

NoBig: TCP not available and UDP is size limited

Permissive: Passes data known to fail validation

5. Roadblock Avoidance

The goal of this document is to tie the above tests and aggregations to avoidance practices; however the document does not specify exactly how to do that.

Once we have determined what level of support is available in the network, we can determine what must be done in order to effectively act as a validating resolver. This section discusses some of the options available given the results from the previous sections.

The general fallback approach can be described by the following sequence:

If the resolver is labeled as "Validator" or "DNSSEC aware":

Send queries through this resolver and perform local
validation on the results.

If validation fails, try the next resolver

Else if the resolver is labeled "Not a DNS Resolver" or
"Non-DNSSEC capable":

Mark it as unusable and try next resolver

Else if no more resolvers are configured and if direct queries
are supported:

1. Try iterating from the Root

2. If the answer is SECURE/BOGUS:
   Return the result of the iteration

3. If the answer is INSECURE:
   Re-query "Non-DNSSEC capable" servers and return
   answers from them w/o the AD bit set to the client.

This will increase the likelihood that split-view unsigned
answers are found.

Else:

Return an error code and log a failure

While attempting resolution through a particular recursive name
server with a particular transport method that worked, any transport-
specific parameters MUST be remembered in order to avoid any
unnecessary fallback attempts.

Transport-specific parameters MUST also be remembered for each
authoritative name server that is queried while performing an
iterative mode lookup.

Any transport settings that are remembered for a particular name
server MUST be periodically refreshed; they should also be refreshed
when an error is encountered as described below.

For a stub resolver, problems with the name server can manifest
themselves under the following types of error conditions:

o No Response, error response or missing DNSSEC meta-data

o Illegal Response: An illegal response is received, which prevents
   the validator from fetching all necessary records required for
   constructing an authentication chain. This could result when
   referral loops are encountered, when any of the antecedent zone
   delegations are lame, when aliases are erroneously followed for
   certain RRtypes (such as SOA, DNSKEYs or DS records), or when
   resource records for certain types (e.g. DS) are returned from a
   zone that is not authoritative for such records.

o Bogus Response: A Bogus Response is received, when the
   cryptographic assertions in the authentication chain do not
   validate properly.

For each of the above error conditions a validator MAY adopt the
following dynamic fallback technique, preferring a particular
approach if it is known to work for a given name server or zone from
previous attempts.

o No response, error response, or missing DNSSEC meta-data:

   * Re-try with different EDNS0 sizes (4096, 1492, None)

   * Re-try with TCP only

   * Perform an iterative query starting from the Root if the
      previous error was returned from a lookup that had recursion
      enabled.

   * Re-try using an alternative transport method, if this
      alternative method is known (configured) to be supported by the
      nameserver in question.

o Illegal Response

   * Perform an iterative query starting from the Root if the
      previous error was returned from a lookup that had recursion
      enabled.

   * Check if any of the antecedent zones up to the closest
      configured trust anchor are provably insecure.

o Bogus Response

   * Perform an iterative query starting from the Root if the
      previous error was returned from a lookup that had recursion
      enabled.

For each fallback technique, attempts to multiple potential name servers should be skewed such that the next name server is tried when the previous one encounters an error, a timeout is reached, or whichever is earlier.

The validator SHOULD remember, in its zone-specific fallback cache, any broken behavior identified for a particular zone for a duration of that zone's SOA negative TTL.

The validator MAY place name servers that exhibit broken behavior into a blacklist, and bypass these name servers for all zones that they are authoritative for. The validator MUST time out entries in this name server blacklist periodically, where this interval could be set to be the same as the DNSSEC BAD cache default TTL.

5.1. Partial Resolver Usage

It may be possible to use Non-DNSSEC Capable caching resolvers in careful ways if maximum optimization is desired. This section describes some of the advanced techniques that could be used to use a resolver in at least a minimal way. Most of the time this would be unnecessary, except in the case where none of the resolvers are fully compliant and thus the choices would be to use them at least minimally or not at all (and no caching benefits would be available).

5.1.1. Known Insecure Lookups

If a resolver is Non-DNSSEC Capable but a section of the DNS tree has been determined to be Provably Insecure [RFC4035], then queries to this section of the tree MAY be sent through Non-DNSSEC Capable caching resolver.

5.1.2. Partial NSEC/NSEC3 Support

Resolvers that understand DNSSEC generally, and understand NSEC but not NSEC3 are partially usable. These resolvers generally also lack support for Unknown types, rendering them mostly useless and to be avoided.

6. Start-Up and Network Connectivity Issues

A number of scenarios will produce either short-term or long-term connectivity issues with respect to DNSSEC validation. Consider the following cases:

Time Synchronization: Time synchronization problems can occur when a device which has been off for a period of time and the clock is no longer in close synchronization with "real time" or when a

device always has clock set to the same time during start-up.
This will cause problems when the device needs to resolve their
source of time synchronization, such as "ntp.example.com".

Changing Network Properties: A newly established network
connection may change state shortly after a HTTP-based pay-wall
authentication system has been used. This especially common in
hotel, airport and coffee-shop style networks, where DNSSEC,
validation and even DNS are not functional until the user proceeds
through a series of forced web pages used to enable their network.
The tests in Section 3 will produce very different results before
and after the network authorization has succeeded. APIs exist on
many operating systems to detect initial network device status
changes, such as right after DHCP has finished, but few (none?)
exist to detect that authentication through a pay-wall has
succeeded.

   There are only two choices when situations like this happen:

Continue to perform DNSSEC processing, which will likely result in
all DNS requests failing. This is the most secure route, but
causes the most operational grief for users.

Turn off DNSSEC support until the network proves to be usable.
This allows the user to continue using the network, at the
sacrifice of security. It also allows for a denial of security-
service attack if a man-in-the-middle can convince a device that
DNSSEC is impossible.

6.1. What To Do

   If the Host Validator detects that DNSSEC resolution is not possible
   it SHOULD log the event and/or SHOULD report an error to the user.
   In the case there is no user, then no reporting can be performed and
   thus the device MAY have a policy of action, like continue or fail.
   Until middle boxes allow DNSSEC protected information to traverse
   them consistently, software implementations may need to offer this
   choice to let users pick the security level they require. Note that
   continuing without DNSSEC protection in the absence of a notification
   or report could lead to situations where users assume a level of
   security that does not exist.

7. Quick Test

   The quick tests defined below make the assumption that the questions
   to be asked are of a real resolver and the only real question is:
   "how complete is the DNSSEC support?". This quick test as been
   implemented in few programs developed at IETF hackthons at IETF-91

and IETF-92. The programs use a common grading method. For each
question that returns expected answer the resolver gets a point. If
the AD bit is set as expected the resolver gets a second point.

7.1. Test negative answers Algorithm 5

   Query: realy-doesnotexist.test.example.com. A

   Answer: RCODE= NXDOMAIN, Empty Answer, Authority: NSEC proof

7.2. Test Algorithm 8

   Query: alg-8-nsec3.test.example.com. SOA

   Answer: RCODE= 0, Answer: SOA record

7.3. Test Algorithm 13

   Query: alg-13-nsec.test.example.com. SOA

   Answer: RCODE= 0, Answer: SOA record

7.4. Fails when DNSSEC does not validate

   Query: dnssec-failed.test.example.com. SOA

   Answer: RCODE= SERVFAIL, empty answer, and authority, AD=0

8. Security Considerations

   This document discusses problems that may occur while deploying the
   DNSSEC protocol. It describes what may be possible to help detect
   and mitigate these problems. Following the outlined suggestions will
   result in a more secure DNSSEC operational environment than if DNSSEC
   was simply disabled.

9. IANA Considerations

   No IANA actions are required.

10. Acknowledgments

   We thank the IESG and DNSOP working group members for their extensive
   comments and suggestions.

11. Normative References

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

   [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.

   [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>.

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

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

Authors' Addresses

   Wes Hardaker
   Parsons
   P.O. Box 382
   Davis, CA 95617
   US

   Email: ietf@hardakers.net

   Olafur Gudmundsson
   CloudFlare
   San Francisco, CA 94107
   USA

   Email: olafur+ietf@cloudflare.com

   Suresh Krishnaswamy
   Parsons
   7110 Samuel Morse Dr
   Columbia, MD 21046
   US

   Email: suresh@tislabs.com

                    Ordering of RRSets in DNS Messages
                   draft-jabley-dnsop-ordered-answers-00

Abstract

   The existing Domain Name System (DNS) specifications lack some
   clarity in their description of the process by which individual
   sections of a DNS message are constructed.

   This document updates RFC 1034 and RFC 1035 to provide a clearer
   specification, consistent with deployed implementations.

Status of this Memo

Copyright Notice

the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Terminology

    This document uses terminology specific to the Domain Name System
    (DNS), descriptions of which can be found in
    [I-D.ietf-dnsop-dns-terminology].

    In an exchange of DNS messages between two hosts, this document
    refers to the host sending a DNS request as the initiator, and the
    host sending a DNS response as the responder.

    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. Introduction

   [RFC1034] specifies an algorithm for use by responders when
   constructing response to a DNS QUERY. This algorithm in some cases
   can result in multiple RRSets being included in a single section of a
   DNS message, e.g. when handling CNAME resource records.

   Many responder implementations have interpreted the direction to copy
   or store particular RRSets in the answer section of a DNS response to
   mean "append", treating each section as an ordered list of RRSets.
   Many initiators, in particular stub resolvers, are known to rely upon
   that interpretation when processing DNS responses received from
   responders.

   Some DNS implementations employ algorithms in other sections that aim
   to optimise processing of responses received by initiators, e.g.
   NAPTR before SRV before A/AAAA in the additional section of a
   response. This behaviour has not been observed to cause any
   interoperability problems, and is explicitly permitted by this
   document.

   This document updates [RFC1035] to specify that the answer section in
   a DNS message is an ordered list of RRSets, but that other sections
   may be constructed differently, and clarifies the directions provided
   in [RFC1034] to match the observed behaviour and expectations of
   deployed software.

3. Updates to RFC 1034

   [RFC1034] specifies the algorithms by which sections of a DNS
   response are constructed by a responder. For example, step 3 of the
   algorithm described in [RFC1034] section 4.3.2 contains the direction
   "copy all RRs which match QTYPE into answer section".

   In this case, and in all other cases where [RFC1034] specifies that
   particular RRSets be included in the answer section of a DNS message,
   the section MUST be treated as an ordered list of RRSets. When it is
   necessary to include new RRSets in a section of a DNS message that is
   under construction, those RRSets MUST be appended. The receiver of a
   DNS message MAY refuse to process DNS messages that have been
   constructed differently.

   When constructing other sections of a DNS message, each section MAY
   be treated as a non-ordered list, and a receiver of a DNS message
   MUST NOT reject a DNS message on the basis of the order of RRSets in
   those sections.

4. Updates to RFC 1035

   In a DNS message, the answer section MUST be considered to be an
   ordered set of RRSets; all other sections MUST be considered to be a
   non-ordered set.

   DNS implementations MUST construct each section in a DNS response
   according to the algorithms specified in [RFC1034], as clarified in
   Section 3 of this document.

5. Security Considerations

   The recommendations contained in this document have no known security
   implications.

6. IANA Considerations

   This document has no IANA actions.

7. Acknowledgements

   The contributions of Mark Andrews and Paul Vixie to this document are
   acknowledged.

8. References

8.1. Normative References

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

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

8.2. Informative References

   [I-D.ietf-dnsop-dns-terminology]
              Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS
              Terminology", draft-ietf-dnsop-dns-terminology-05 (work in
              progress), September 2015.

Appendix A. Editorial Notes

   This section (and sub-sections) to be removed prior to publication.

A.1. Venue

   An appropriate forum for discussion of this draft is the dnsop
   working group.

A.2. Change History

A.2.1. draft-jabley-dnsop-ordered-answers-00

   Initial draft circulated for comment.

Author's Address

   Joe Abley
   Dyn, Inc.
   103-186 Albert Street
   London, ON N6A 1M1
   Canada

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

           Providing Minimal-Sized Responses to DNS Queries with QTYPE=ANY
                      draft-jabley-dnsop-refuse-any-01

Abstract

   The Domain Name System (DNS) specifies a query type (QTYPE) "ANY".
   The operator of an authoritative DNS server might choose not to
   respond to such queries for reasons of local policy, motivated by
   security, performance or other reasons.

   The DNS specification does not include specific guidance for the
   behaviour of DNS servers or clients in this situation. This document
   aims to provide such guidance.

Status of this Memo

Copyright Notice

carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Terminology

   This document uses terminology specific to the Domain Name System
   (DNS), descriptions of which can be found in
   [I-D.ietf-dnsop-dns-terminology].

   In this document, "ANY Query" refers to a DNS query with QTYPE=ANY.
   An "ANY Response" is a response to such a query.

   In an exchange of DNS messages between two hosts, this document
   refers to the host sending a DNS request as the initiator, and the
   host sending a DNS response as the responder.

   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. Introduction

   The Domain Name System (DNS) specifies a query type (QTYPE) "ANY".
   The operator of an authoritative DNS server might choose not to
   respond to such queries for reasons of local policy, motivated by
   security, performance or other reasons.

   The DNS specification [RFC1034] [RFC1035] does not include specific
   guidance for the behaviour of DNS servers or clients in this
   situation. This document aims to provide such guidance.

3. Motivations

   ANY queries are legitimately used for debugging and checking the
   state of a DNS server for a particular owner name. ANY queries are
   sometimes used as a attempt to reduce the number of queries needed to
   get information, e.g. to obtain MX, A and AAAA RRSets for a mail
   domain in a single query, although there is no documented guidance
   available for this use case and some implementations have been
   observed that appear not to function as perhaps their developers
   expected.

   ANY queries are also frequently used to exploit the amplification
   potential of DNS servers using spoofed source addresses and UDP
   transport (see [RFC5358]). Having the ability to return small
   responses to such queries makes DNS servers less attractive
   amplifiers.

   ANY queries are sometimes used to help mine authoritative-only DNS
   servers for zone data, since they return all RRSets for a particular
   owner name. A DNS zone maintainer might prefer not to send full ANY
   responses to reduce the potential for such information leaks.

   Some authoritative-only DNS server implementations require additional
   processing in order to send a conventional ANY response, and avoiding
   that processing expense may be desirable.

4. General Approach

   This proposal provides a mechanism for an authority server to signal
   that conventional ANY queries are not supported for a particular
   QNAME, and to do so in such a way that is both compatible with and
   triggers desirable behaviour by unmodified clients (e.g. DNS
   resolvers).

   Alternative proposals for dealing with ANY queries have been
   discussed. One approach proposed using a new RCODE to signal that an
   authortitaive server did not answer ANY queries in the standard way.
   This approach was found to have an undesirable effect on both
   resolvers and authoritative-only servers; resolvers receiving an
   unknown RCODE caused them to re-send the same query to all available
   authoritative servers, rather than suppress future such ANY queries
   for the same QNAME.

   This proposal avoids that outcome by returning a non-empty RRSet in
   the ANY response, providing resolvers with something to cache and
   effectively suppressing repeat queries to the same or different
   authority servers.

   This proposal specifies two different modes of behaviour by DNS
   responders, and operators are free to choose whichever mechanism best
   suits their environment.

   1. A DNS responder may choose to search for an owner name that
       matches the QNAME and, if that name owns multiple RRs, return
       just one of them.

   2. A DNS responder for whom a search for an owner name with an
       existing resource record is expensive may instead synthesise an
       HINFO resource record and return that instead. See Section 7 for
       discussion of the use of HINFO.

5. Behaviour of DNS Responders

   A DNS responder which receives an ANY query MAY decline to provide a
   conventional response, and MAY instead send a response with a single
   RRSet in the answer section.

   The RRSet returned in the answer section of the response MAY be a
   single RRSet owned by the name specified in the QNAME. Where
   mulitple RRSets exist, the responder MAY choose a small one to reduce
   its amplification potential.

   If there is no CNAME present at the owner name matching the QNAME,
   the resource record returned in the response MAY instead synthesised,
   in which case a single HINFO resource record should be returned. The
   CPU field of the HINFO RDATA SHOULD be set to RFCXXXX [note to RFC
   Editor, replace with RFC number assigned to this document]. The OS
   field of the HINFO RDATA SHOULD be set to the null string to minimise
   the size of the response.

   The TTL encoded for a synthesised RR SHOULD be chosen by the operator
   of the DNS responder to be large enough to suppress frequent
   subsequent ANY queries from the same initiator with the same QNAME,
   understanding that a TTL that is too long might make policy changes
   relating to ANY queries difficult to change in the future. The
   specific value used is hence a familiar balance when choosing TTLs
   for any RR in any zone, and should be specified according to local
   policy.

   If the DNS query includes DO=1 and the QNAME corresponds to a zone
   that is known by the responder to be signed, a valid RRSIG for the
   RRSets in the answer section MUST be returned.

   Except as described in this section, the DNS responder MUST follow
   the standard algorithms when constructing a response.

6. Behaviour of DNS Initiators

   XXX consider whether separate text here is required depending on
   whether the initiator is a non-caching stub resolver or a caching
   recursive resolver.

   A DNS initator which sends a query with QTYPE=ANY and receives a
   response containing an HINFO, as described in Section 5, MAY cache
   the HINFO response in the normal way. Such cached HINFO resource
   records SHOULD be retained in the cache following normal caching
   semantics, as it would with any other response received from a DNS
   responder.

   A DNS initiator MAY suppress queries with QTYPE=ANY in the event that
   the local cache contains a matching HINFO resource record with
   RDATA.CPU field, as described in Section 5.

7. HINFO Considerations

   In the case where a zone that contains HINFO RRSets is served from an
   authority server that does not provide conventional ANY responses, it
   is possible that the HINFO RRSet in an ANY response, once cached by
   the initiator, might suppress subsequent queries from the same
   initiator with QTYPE=HINFO. The use of HINFO in this proposal would
   hence have effectively masked the HINFO RRSet present in the zone.

   Authority-server operators who serve zones that rely upon
   conventional use of the HINFO RRType might sensibly choose not to
   deploy the mechanism described in this document.

   The HINFO RRType is believed to be rarely used in the DNS at the time
   of writing, based on observations made both at recursive servers and
   authority servers.

8. Changes to RFC 1035

   It is important to note that returning a subset of available RRSets
   when processing an ANY query is legitimate and consistent with
   [RFC1035]; ANY does not mean ALL.

   This document describes optional behaviour for both DNS initators and
   responders, and implementation of the guidance provided by this
   document is OPTIONAL.

9. Security Considerations

   Queries with QTYPE=ANY are frequently observed as part of reflection
   attacks, since a relatively small query can be used to elicit a large
   response; this is a desirable characteristic if the goal is to
   maximise the amplification potential of a DNS server as part of a
   volumetric attack. The ability of a DNS operator to suppress such
   responses on a particular server makes that server a less useful
   amplifier.

   The optional behaviour described in this document to reduce the size
   of responses to queries with QTYPE=ANY is compatible with the use of
   DNSSEC by both initiator and responder.

10. IANA Considerations

   This document has no IANA actions.

11. Acknowledgements

   Evan Hunt and David Lawrence provided valuable observations.

12. References

12.1. Normative References

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

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

12.2. Informative References

   [I-D.ietf-dnsop-dns-terminology]
              Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS
              Terminology", draft-ietf-dnsop-dns-terminology-05 (work in
              progress), September 2015.

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

Appendix A. Editorial Notes

   This section (and sub-sections) to be removed prior to publication.

A.1. Venue

   An appropriate forum for discussion of this draft is the dnsop
   working group.

A.2. Change History

A.2.1. draft-jabley-dnsop-refuse-any-01

   Make signing of RRSets in answers from signed zones mandatory.

   Document the option of returning an existing RRSet in place of a
   synthesised one.

A.2.2. draft-jabley-dnsop-refuse-any-00

   Initial draft circulated for comment.

Authors' Addresses

    Joe Abley
    Dyn, Inc.
    103-186 Albert Street
    London, ON   N6A 1M1
    Canada

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

    Olafur Gudmundsson
    CloudFlare Inc.

    Email: olafur@cloudflare.com

    Marek Majkowski
    CloudFlare Inc.

    Email: marek@cloudflare.com

Internet Engineering Task Force   M. Sivaraman
Internet-Draft   Internet Systems Consortium
Intended status: Experimental   S. Kerr
Expires: January 21, 2016   L. Song
                                            Beijing Internet Institute
                                                        July 20, 2015

                        DNS message fragments
                  draft-muks-dns-message-fragments-00

Abstract

   This document describes a method to transmit DNS messages over
   multiple UDP datagrams by fragmenting them at the application layer.
   The objective is to allow authoriative servers to successfully reply
   to DNS queries via UDP using multiple smaller datagrams, where larger
   datagrams may not pass through the network successfully.

Status of This Memo

Copyright Notice

the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction

1.1. Background

   [RFC1035] describes how DNS messages are to be transmitted over UDP.
   A DNS query message is transmitted using one UDP datagram from client
   to server, and a corresponding DNS reply message is transmitted using
   one UDP datagram from server to client.

   The upper limit on the size of a DNS message that can be transmitted
   thus depends on the maximum size of the UDP datagram that can be
   transmitted successfully from the sender to the receiver. Typically
   any size limit only matters for DNS replies, as DNS queries are
   usually small.

As a UDP datagram is transmitted in a single IP PDU, in theory the
size of a UDP datagram (including various lower internet layer
headers) can be as large as 64 KiB. But practically, if the datagram
size exceeds the path MTU, then the datagram will either be
fragmented at the IP layer, or worse dropped, by a forwarder. In the
case of IPv6, DNS packets are fragmented by the sender only. If a
packet's size exceeds the path MTU, a Packet Too Big (PTB) ICMP
message will be received by sender without any clue to the sender to
reply again with a smaller sized message, due to the stateless
feature of DNS. In addition, IP-level fragmentation caused by large
DNS response packet will introduce risk of cache poisoning
[Fragment-Poisonous], in which the attacker can circumvent some
defense mechanisms (like port, IP, and query randomization
[RFC5452]).

As a result, a practical DNS payload size limitation is necessary.
[RFC1035] limited DNS message UDP datagram lengths to a maximum of
512 bytes. Although EDNS(0) [RFC6891] allows an initiator to
advertise the capability of receiving lager packets (up to 4096
bytes), it leads to fragmentation because practically most packets
are limited to 1500 byte size due to host Ethernet interfaces, or
1280 byte size due to minimum IPv6 MTU in the IPv6 stack [RFC3542].

According to DNS specifications [RFC1035], if the DNS response
message can not fit within the packet's size limit, the response is
truncated and the initiator will have to use TCP as a fallback to re-
query to receive large response. However, not to mention the high
setup cost introduced by TCP due to additional roundtrips, some
firewalls and middle boxes even block TCP/53 which cause no responses
to be received as well. It becomes a significant issue when the DNS
response size inevitably increases with DNSSEC deployment.

In this memo, DNS message fragmentation attempts to work around
middle box misbehavior by splitting a single DNS message across
multiple UDP datagrams. Note that to avoid DNS amplification and
reflection attacks, DNS cookies [I-D.ietf-dnsop-cookies] is a
mandatory requirement when using DNS message fragments.

1.2. Motivation

It is not a new topic regarding large DNS packets(>512B) issue
[I-D.ietf-dnsop-respsize], starting from introduction of IPv6,
EDNS(0) [SAC016], and DNSSEC deployment [SAC035]. In current
production networks, using DNSSEC with longer DNSKEYs (ZSK>1024B and
KSK>2048B) will result in response packets no smaller than 1500B
[T-DNS]. Especially during the KSK rollover process, responses to
the query of DNSKEY RRset will be enlarged as they contain both the
new and old KSK.

When possible, we should avoid dropped packets as this means the
client must wait for a timeout, which incurs a high cost. For
example, a validator behind a firewall suffers waiting till the
timeout with no response, if the firewall drops large EDNS(0) packets
and IP fragments. It may even cause disaster when the validator can
not recieve response for new trust anchor KSK due to the extreme case
of bad middle boxes which also drop TCP/53.

Since UDP requires fewer packets on the wire and less state on
servers than TCP, in this memo we propose continuing to use UDP for
transmission but fragment the larger DNS packets into smaller DNS
packets at the application layer. We would like the fragments to
easily go through middle boxes and avoid falling back to TCP.

2. DNS Message Fragmentation Method

2.1. Client Behavior

   Clients supporting DNS message fragmentation add an EDNS option to
   their queries, which declares their support for this feature.

   If a DNS reply is received that has been fragmented, it will consist
   of multiple DNS message fragments (each transmitted in a respective
   UDP packet), and every fragment contain an EDNS option which says how
   many total fragments there are, and the identifier of the fragment
   that the current packet represents. The client collects all of the
   fragments and uses them to reconstruct the full DNS message. Clients
   MUST maintain a timeout when waiting for the fragments to arrive.

   Clients that support DNS message fragments MUST be able to reassemble
   fragments into a DNS message of any size, up to the maximum of 64KiB.

   The client MAY save information about what sizes of packets have been
   received from a given server. If saved, this information MUST have a
   limited duration.

   Any DNSSEC validation is performed on the reassembled DNS message.

2.2. Server Behavior

   Servers supporting DNS message fragmentation will look for the EDNS
   option which declares client support for the feature. If not
   present, the server MUST NOT use DNS message fragmentation. The
   server MUST check that DNS cookies are supported. [**FIXME**]
   Implementation of the first request case, where no existing
   established cookie is available needs discussion; we want to avoid
   additional round-trips here. Shane: don't cookies already handle
   this case?

   The server prepares the response DNS message normally. If the
   message exceeds the maximum UDP payload size specified by the client,
   then it should fragment the message into multiple UDP datagrams.

   Each fragment contains an identical DNS header with TC=1, possibly
   varying only in the section counts. Setting the TC flag in this way
   insures that clients which do not support DNS fragments can fallback
   to TCP transparently.

   As many RR are included in each fragment as are possible without
   going over the desired size of the fragment. An EDNS option is added
   to every fragment, that includes both the fragment identifier and the
   total number of fragments.

   The server needs to know how many total fragments there are to insert
   into each fragment. A simple approach would be to generate all
   fragments, and then count the total number at the end, and update the
   previously-generated fragments with the total number of fragments.
   Other techniques may be possible.

   The server MUST limit the number of fragments that it uses in a
   reply. (See "Open Issues and Discussion" for remaining work.)

   The server MUST NOT exceed the maximum fragment size requested by a
   client.

    The server should use the following sizes for each fragment in the
                         sequence in IPv4:

          +-------------+-------------------------------+
          | Fragment ID | Size |
          +-------------+-------------------------------+
          | 1 | min(512, client_specified_max) |
          | 2 | min(1460, client_specified_max) |
          | 3 | min(1480, client_specified_max) |
          | N | min(1480, client_specified_max) |
          +-------------+-------------------------------+

   The rationale is that the first packet will always get through, since
     if a 512 octet packet doesn't work, DNS cannot function. We then
    increase to sizes that are likely to get through. 1460 is the 1500
    octet Ethernet packet size, minus the IP header overhead and enough
    space to support tunneled traffic. 1480 is the 1500 octet Ethernet
    packet size, minus the IP header overhead. [**FIXME**] Why not add
    1240 here? Shane answers: 1280 is not any kind of limit in IPv4, as
                            far as I know.

The server should use the following sizes for each packet in the
sequence in IPv6:

```
+-------------+-------------------------------+
| Fragment ID | Size                          |
+-------------+-------------------------------+
| 1           | min(1240, client_specified_max) |
| 2           | min(1420, client_specified_max) |
| 3           | min(1460, client_specified_max) |
| N           | min(1460, client_specified_max) |
+-------------+-------------------------------+
```

Like with IPv4, the idea is that the first packet will always get
through. In this case we use the IPv6-mandated 1280 octets, minus
the IP header overhead. We then increase to 1420, which is the 1500
octet Ethernet packet size, minus the IP header overhead and enough
space to support tunneled traffic. 1460 is the 1500 octet Ethernet
packet size, minus the IP header overhead.

2.3. Other Notes

   o The FRAGMENT option MUST NOT be present in DNS query messages,
      i.e., when QR=0. If a DNS implementation notices the FRAGMENT
      option in a DNS query message, it MUST ignore it.

   o In DNS reply messages, the FRAGMENT option MUST NOT be present in
      datagrams when truncation is not done, i.e., when TC=0. If a DNS
      implementation notices the FRAGMENT option in a DNS reply message
      fragment datagram that is not truncated, i.e, when TC=0, it MUST
      drop all DNS reply message fragment datagrams received so far
      (awaiting assembly) for that message's corresponding question
      tuple (server IP, port, message ID) without using any data from
      them. [**FIXME**] Dropping fragments to be received yet will be
      problematic for implementations, but dropping fragments received
      so far ought to be sufficient.

   o More than one FRAGMENT option MUST NOT be present in a DNS reply
      message fragment datagram. If a DNS implementation notices
      multiple FRAGMENT options in a DNS reply message fragment
      datagram, it MUST drop all reply datagrams received for that
      message's corresponding question tuple (server IP, port, message
      ID) without using any data from them. [**FIXME**] Dropping
      fragments to be received yet will be problematic for
      implementations, but dropping fragments received so far ought to
      be sufficient.

3. The ALLOW-FRAGMENTS EDNS(0) Option

   ALLOW-FRAGMENTS is an EDNS(0) [RFC6891] option that a client uses to
   inform a server that it supports fragmented responses. [**FIXME**]
   Why not simply use the FRAGMENT option here with count=0,
   identifier=ignored and avoid using another option code? Shane: There
   are no shortage of options. Plus, if we want to include a maximum
   fragment size value in the ALLOW-FRAGMENTS then we really need a
   separate option.

3.1. Wire Format

   TBD.

3.2. Option Fields

3.2.1. Maximum Fragment Size

   The Maximum Fragment Size field is represented as an unsigned 16-bit
   integer. This is the maximum size used by any given fragment the
   server returns. [**FIXME**] This field's purpose has to be explained.
   Shane: discussed in the discussion section now.

3.3. Presentation Format

   As with other EDNS(0) options, the ALLOW-FRAGMENTS option does not
   have a presentation format.

4. The FRAGMENT EDNS(0) Option

   FRAGMENT is an EDNS(0) [RFC6891] option that assists a client in
   gathering the various fragments of a DNS message from multiple UDP
   datagrams. It is described in a previous section. Here, its syntax
   is provided.

4.1. Wire Format

   TBD.

4.2. Option Fields

4.2.1. Fragment Identifier

   The Fragment Identifier field is represented as an unsigned 8-bit
   integer. The first fragment is identified as 1. Values in the range
   [1,255] can be used to identify the various fragments. Value 0 is
   used for signalling purposes.

4.2.2. Fragment Count

   The Fragment Count field is represented as an unsigned 8-bit integer.
   It contains the number of fragments in the range [1,255] that make up
   the DNS message. Value 0 is used for signalling purposes.

4.3. Presentation Format

   As with other EDNS(0) options, the FRAGMENT option does not have a
   presentation format.

5. Network Considerations

5.1. Background

   TCP-based application protocols co-exist well with competing traffic
   flows in the internet due to congestion control methods such as in
   [RFC5681] that are present in TCP implementations.

   UDP-based application protocols have no restrictions in lower layers
   to stop them from flooding datagrams into a network and causing
   congestion. So applications that use UDP have to check themselves
   from causing congestion so that their traffic is not disruptive.

   In the case of [RFC1035], only one reply UDP datagram was sent per
   request UDP datagram, and so the lock-step flow control automatically
   ensured that UDP DNS traffic didn't lead to congestion. When DNS
   clients didn't hear back from the server, and had to retransmit the
   question, they typically paced themselves by using methods such as a
   retransmission timer based on a smoothed round-trip time between
   client and server.

   Due to the message fragmentation described in this document, when a
   DNS query causes multiple DNS reply datagrams to be sent back to the
   client, there is a risk that without effective control of flow, DNS
   traffic could cause problems to competing flows along the network
   path.

   Because UDP does not guarantee delivery of datagrams, there is a
   possibility that one or more fragments of a DNS message will be lost
   during transfer. This is especially a problem on some wireless
   networks where a rate of datagrams can continually be lost due to
   interference and other environmental factors. With larger numbers of
   message fragments, the probability of fragment loss increases.

5.2. Implementation Requirements

   TBD.

6. Open Issues and Discussion

   1.   Resolver behavior

        We need some more discussion of resolver behavior in general, at
        least to the point of making things clear to an implementor.

   2.   The use of DNS fragments mechanism

        Is this mechanism designed for all DNS transactions, or only
        used in some event or special cases like a key rollover process?
        If the mechanism is designed for general DNS transactions, when
        is it triggered and how is it integrated with existing patterns?

        One option is that DNS fragments mechanism works as a backup
        with EDNS, and triggered only when a larger packet fails in the
        middle. It will be orthogonal with TCP which provide additional
        context that TC bit will be used in server side.

   3.   What is the size of fragments?

        Generally speaking the number of fragment increases if fragment
        size is small (512 bytes, or other empirical value), which makes
        the mechanism less efficient. If the size can changed
        dynamically according to negotiation or some detection, it will
        introduce more cost and round trip time.

   4.   What happens if a client that does not support DNS fragments
        receives an out-of-order or partial fragment?

        We need to consider what happens when a client that does not
        support DNS fragments gets a partial response, possibly even out
        of order.

   5.   We should explain risk of congestion, packet loss, etc. when
        introducing the limit on the number of fragments. We might also
        set specific upper limits for number of fragments.

6.   EDNS buffer sizes vs. maximum fragmentation sizes

     Mukund: We need further discussion about the sizes; also an
     upper limit for each *fragment* has to be the client's UDP
     payload size as it is the driver and it alone knows the ultimate
     success/failure of message delivery. So if it sets a maximum
     payload size of 1200, there's no point in trying 1460. Clients
     that support DNS message fragments (and signal support using the
     EDNS option) should adapt their UDP payload size discovery
     algorithm to work with this feature, as the following splits on
     sizes will assist PMTU discovery.

     Shane: I think we need to separate the EDNS maximum UDP payload
     size from the maximum fragment size. I think that it is quite
     likely that (for example) we will want to restrict each fragment
     to 1480 bytes, but that the EDNS buffer size might remain at 4
     kibibytes.

7.   TSIG should be addressed

     We need to document how to handle TSIG, even though this is not
     likely to be a real-world issue. Probably each fragment should
     be TSIG signed, as this makes it harder for an attacker to
     inject bogus packets that a client will have to process.

8.   RR splitting should be addressed

     We need to document whether or not RR can be split. Probably it
     makes sense not to allow this, although this will reduce the
     effectiveness of the fragmentation, as the units that can be
     packed into each fragment will be bigger.

9.   We need to document that some messages may not be possible to
     split.

     Some messages may be too large to split. A trivial example is a
     TXT record that is larger than the buffer size. Probably the
     best behavior here is to truncate.

10. DNSSEC checks

DNSSEC checks should be done on the final reassembled packet.
This needs to be documented.

11. Name compression

Name compression should be done on the each fragment separately.
This needs to be documented.

12. OPT-RR

Some OPT-RR seem to be oriented at the entire message, others
make more sense per packet. This needs to be sorted out. Also
we need to investigate the edge case where fragments have
conflicting options (Mukund thinks that we can copy the approach
in the EDNS specification and use the same rules about
conflicting OPT-RR that it uses.)

7. Security Considerations

To avoid DNS amplification or reflection attacks, DNS cookies
[I-D.ietf-dnsop-cookies] must be used. The DNS cookie EDNS option is
identical in all fragments that make up a DNS message. The
duplication of the same cookie values in all fragments that make up
the message is not expected to introduce a security weakness in the
case of off-path attacks.

8. IANA Considerations

The ALLOW-FRAGMENTS and FRAGMENT EDNS(0) options require option codes
to be assigned for them.

9. Acknowledgements

Thanks to Stephen Morris, JINMEI Tatuya, Paul Vixie, Mark Andrews,
and David Dragon for reviewing a pre-draft proposal and providing
support, comments and suggestions.

10. References

[Fragment-Poisonous]
          Herzberg, A. and H. Shulman, "Fragmentation Considered
          Poisonous", 2012.

   [I-D.ietf-dnsop-cookies]
              Eastlake, D. and M. Andrews, "Domain Name System (DNS)
              Cookies", draft-ietf-dnsop-cookies-04 (work in progress),
              July 2015.

   [I-D.ietf-dnsop-respsize]
              Vixie, P., Kato, A., and J. Abley, "DNS Referral Response
              Size Issues", draft-ietf-dnsop-respsize-15 (work in
              progress), February 2014.

   [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.

   [RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei,
              "Advanced Sockets Application Program Interface (API) for
              IPv6", RFC 3542, May 2003.

   [RFC5452] Hubert, A. and R. van Mook, "Measures for Making DNS More
              Resilient against Forged Answers", RFC 5452, January 2009.

   [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion
              Control", RFC 5681, September 2009.

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

   [SAC016]   ICANN Security and Stability Advisory Committee, "Testing
              Firewalls for IPv6 and EDNS0 Support", 2007.

   [SAC035]   ICANN Security and Stability Advisory Committee, "DNSSEC
              Impact on Broadband Routers and Firewalls", 2008.

   [T-DNS]   Zhu, L., Hu, Z., and J. Heidemann, "T-DNS: Connection-
              Oriented DNS to Improve Privacy and Security (extended)",
              2007, <http://www.isi.edu/~johnh/PAPERS/Zhu14b.pdf>.

Appendix A. Change History (to be removed before publication)

   o draft-muks-dns-message-fragments-00
      Initial draft.

Authors' Addresses

    Mukund Sivaraman
    Internet Systems Consortium
    950 Charter Street
    Redwood City, CA 94063
    US

    Email: muks@isc.org
    URI: http://www.isc.org/

    Shane Kerr
    Beijing Internet Institute
    2/F, Building 5, No.58 Jinghai Road, BDA
    Beijing 100176
    CN

    Email: shane@biigroup.cn
    URI: http://www.biigroup.com/

    Linjian Song
    Beijing Internet Institute
    2/F, Building 5, No.58 Jinghai Road, BDA
    Beijing 100176
    CN

    Email: songlinjian@gmail.com
    URI: http://www.biigroup.com/

Internet Engineering Task Force   M. Sivaraman
Internet-Draft   Internet Systems Consortium
Intended status: Experimental   October 13, 2015
Expires: April 15, 2016

                       DNS message checksums
               draft-muks-dnsop-dns-message-checksums-01

Abstract

   This document describes a method for a client to be able to verify
   that IP-layer PDU fragments of a UDP DNS message have not been
   spoofed by an off-path attacker.

Table of Contents

1. Introduction

   [RFC1035] describes how DNS messages are to be transmitted over UDP.
   A DNS query message is transmitted using one UDP datagram from client
   to server, and a corresponding DNS reply message is transmitted using
   one UDP datagram from server to client.

   As a UDP datagram is transmitted in a single IP PDU, in theory the
   size of a UDP datagram (including various lower internet layer
   headers) can be as large as 64 KiB. But practically, if the datagram
   size exceeds the path MTU, then the datagram will either be
   fragmented at the IP layer, or dropped by a forwarder. In the case
   of IPv4, DNS datagrams may be fragmented by a sender or a forwarder.
   In the case of IPv6, DNS datagrams are fragmented by the sender only.

   IP-layer fragmentation for large DNS response datagrams introduces
   risk of cache poisoning by off-path attackers [Fragment-Poisonous] in
   which an attacker can circumvent some defense mechanisms like source
   port and query ID randomization [RFC5452].

   This memo introduces the concept of a DNS message checksum which may
   be used to stop the effects of such off-path attacks.

   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. DNS message checksum method

   Clients supporting DNS message checksums add an EDNS option to their
   queries, which signals their support for this feature.

   The CHECKSUM EDNS option contains 3 fields: NONCE, ALGORITHM, and
   DIGEST. These fields are described in Section 3.

   It is OPTIONAL for a client to add a CHECKSUM EDNS option to DNS
   query messages. If it adds such an option, it MUST set the NONCE
   field to a random value. The ALGORITHM field MUST be set to 0 and
   the DIGEST field MUST be left empty. The entire NONCE field MUST be
   randomly generated (i.e., in no predictable sequence and the random
   value must fill all bits of the field) for each query for which the
   client uses a CHECKSUM EDNS option. The client is expected to
   remember the per-query NONCE field's value to be used in verifying
   the reply to this query message.

   A client MUST NOT send multiple DNS query messages with the NONCE set
   to a fixed unchanging value. Instead, it must not send the option at
   all.

   The server SHOULD add a CHECKSUM EDNS option in the reply message to
   a corresponding query that arrived with this option present. The
   NONCE field MUST be copied verbatim from the query message to the
   corresponding reply message. A checksum is computed over the DNS
   reply message as described in Section 4 and the ALGORITHM and DIGEST
   fields MUST be set using the resulting checksum as given in
   Section 3. The server is at liberty to choose any checksum algorithm
   it wants to from the list of supported algorithms given in
   Appendix A.

   If a server receives a query containing a CHECKSUM EDNS option with
   an ALGORITHM field that is not set to 0, it MUST ignore this option
   and process the request as if there were no CHECKSUM EDNS option in
   the query.

   When a client receives a reply message for which it sent a CHECKSUM
   EDNS option in the corresponding query, it SHOULD look for the
   presence of the CHECKSUM EDNS option in the reply.

   The client may handle the lack of a CHECKSUM EDNS option in the reply
   as it chooses to. It is currently not specified, but may be updated
   in the future.

   If a client receives a reply containing a CHECKSUM EDNS option with
   an unknown ALGORITHM value, it MUST ignore this option and handle the
   reply as if there were no CHECKSUM EDNS option in it. From the

previous paragraph, it follows that the client behavior in this case
is also currently not specified, but may be updated in the future.

If a CHECKSUM EDNS option is present in the reply, the client SHOULD
first check and ensure that the NONCE field contains the same nonce
value that was sent in the corresponding query message. If the value
in the NONCE field is different, the reply message MUST be discarded.
Afterwards, the client SHOULD proceed to compute a checksum over the
reply message as described in Section 4 using the checksum algorithm
in the ALGORITHM field. It SHOULD then compare the checksum value
with the value that was received in the DIGEST field for equality.
If they are not equal, the reply message MUST be discarded. If they
are equal, the reply message can be used normally as the client
intends to use it.

3. The CHECKSUM EDNS(0) option

   CHECKSUM is an EDNS(0) [RFC6891] option that is used to transmit a
   digest of a DNS message in replies. Client and server behavior are
   described in Section 2. In this section, the option's syntax is
   provided.

3.1. Wire format

   The following describes the wire format of the OPTION-DATA field
   [RFC6891] of the CHECKSUM EDNS option. All CHECKSUM option fields
   must be represented in network byte order.

```
       +--------------+------------------+-------------------+
       | Option field | Type             | Field size        |
       +--------------+------------------+-------------------+
       | NONCE        | unsigned integer | 64 bits (8 octets)|
       | ALGORITHM    | unsigned integer | 8 bits (1 octet)  |
       | DIGEST       | byte array       | Variable length   |
       +--------------+------------------+-------------------+
```

3.2. Option fields

3.2.1. NONCE

   The NONCE field is represented as an unsigned 64-bit integer in
   network byte order. It MUST be randomly computed for each query
   message which a client sends out, and is copied verbatim from the
   query to the corresponding reply DNS message by the server.

3.2.2. ALGORITHM

   The ALGORITHM field is represented as an unsigned 8-bit integer in
   network byte order. In query messages, it MUST be set to 0. In
   reply messages, it MUST contain the numeric value of the algorithm
   used to compute the DIGEST field. A list of algorithms and their
   values is given in Appendix A.

3.2.3. DIGEST

   The DIGEST field is represented as a variable-length sequence of
   octets present after the NONCE and ALGORITHM fields. Its size is
   implicitly computed from the value in the OPTION-LENGTH field
   [RFC6891] for the CHECKSUM EDNS option minus the size of the NONCE
   and ALGORITHM fields. In query messages, it MUST be empty. In reply
   messages, it MUST contain the digest of the reply message which is
   computed as described in Section 4.

3.3. Presentation format

   As with other EDNS(0) options, the CHECKSUM EDNS option does not have
   a presentation format.

4. Checksum computation

   To generate the checksum digest to be placed in the DIGEST field,
   first the entire DNS message must be prepared (rendered) along with
   the CHECKSUM option embedded in it to the point that it is ready to
   be sent out on the wire. In this CHECKSUM option, initially the
   DIGEST field must be filled with zero values and its size must be
   reserved equal to the size expected for the digest from the checksum
   algorithm intended to be used. The NONCE field MUST be set to the
   value of the nonce from the query DNS message. The ALGORITHM field
   MUST be set to the checksum algorithm intended to be used. After
   this, the whole message contents (from the start of the DNS message
   header onwards) must be input to the checksum algorithm and the
   calculated checksum must be patched into the DIGEST field, space for
   which was reserved before.

   To verify the checksum digest from a DNS message that was received,
   first the DIGEST field is copied to a temporary location and the
   DIGEST field in the message is patched with zero values. After this,
   the whole message contents (from the start of the DNS message header
   onwards) must be input to the checksum algorithm specified in the
   ALGORITHM field. The calculated checksum must be compared for
   equality with the checksum originally received in the DIGEST field,
   the content of which was earlier saved to a temporary location. If
   both are equal, the checksum matches.

5. Security considerations

   The methods in this memo are designed to thwart off-path spoofing
   attacks which may lead to cache-poisoning, including the specific
   case when IP-layer PDU fragmentation occurs.

   The CHECKSUM EDNS option is not designed to offer any protection
   against on-path attackers. Very little can be done without using
   shared-secret or public key cryptography for this case.

   Checksum computation may increase resource usage on servers and
   clients. It is thus desirable to use fast checksum algorithms that
   meet the requirements of Appendix A.

   The entropy source used for generating random values for use in the
   NONCE field may be chosen similarly to provide ample security to
   verify a short-lived DNS message.

   The NONCE field effectively extends the ID field [RFC1035] in the DNS
   message header.

   As a side-effect of using checksums, resolver cache poisoning attacks
   are made more difficult due to the presence of the NONCE field.

   There is a risk of downgrade attack when the IP fragment containing
   the CHECKSUM EDNS option is spoofed, deleting this option. This risk
   would exist until the presence of the CHECKSUM option in replies is
   made mandatory when a corresponding option is sent in the query.
   This can be made so right from the start, or after an adoption
   period. At that time, it may be stated that a client that does not
   receive a CHECKSUM EDNS option in a reply would discard the reply
   message and retry the query using TCP.

   The CHECKSUM EDNS option cannot prevent some kinds of attack such as
   response and NS blocking and NS pinning as described in
   [Fragment-Poisonous].

6. IANA considerations

   This document defines a new EDNS(0) option, titled CHECKSUM (see
   Section 3), assigned a value of <TBD> from the DNS EDNS0 Option Codes
   (OPT) space [to be removed upon publication:
   https://www.iana.org/assignments/dns-parameters/dns-
   parameters.xhtml#dns-parameters-11].

+------+----------+--------+-------------------------------------+
| Valu | Name | Status | Reference |
| e | | | |
+------+----------+--------+-------------------------------------+
| TBD | CHECKSUM | TBD | [draft-muks-dnsop-dns-message- |
| | | | checksums] |
+------+----------+--------+-------------------------------------+

The CHECKSUM EDNS(0) option also defines an 8-bit ALGORITHM field,
for which IANA is to create and maintain a new sub-registry entitled
"DNS message checksum algorithms" under the Domain Name System (DNS)
Parameters. Initial values for the DNS message checksum algorithms
registry are given in Appendix A; future assignments are to be made
through Expert Review as in BCP 26 [RFC5226]. Assignments consist of
a DNS message checksum algorithm name and its associated value.

7. Acknowledgements

Tomek Mrugalski offered tips on draft naming and upload process. Joe
Abley reviewed the draft and pointed out some nits that were not
detected automatically. Ray Bellis, Robert Edmonds, Tony Finch, Paul
Hoffman, Evan Hunt, Paul Vixie, and Paul Wouters reviewed drafts and
sent in comments and opinions. Mark Andrews mentioned an alternate
method at the same time (on an internal mailing list) to address
spoofing issues that provided further support to the idea that
CHECKSUM was worth pursuing.

8. References

8.1. Normative references

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

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

   [RFC5452] Hubert, A. and R. van Mook, "Measures for Making DNS More
              Resilient against Forged Answers", RFC 5452, DOI 10.17487/
              RFC5452, January 2009,
              <http://www.rfc-editor.org/info/rfc5452>.

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

8.2. Informative references

   [Fragment-Poisonous]
              Herzberg, A. and H. Shulman, "Fragmentation Considered
              Poisonous", 2012.

   [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an
              IANA Considerations Section in RFCs", BCP 26, RFC 5226,
              DOI 10.17487/RFC5226, May 2008,
              <http://www.rfc-editor.org/info/rfc5226>.

Appendix A. Checksum algorithms

   The ALGORITHM field as specified in Section 3 identifies the checksum
   algorithm that is used to compute the checksum digest for a DNS
   message.

   The following table lists the currently defined checksum algorithm
   types. Candidate checksum algorithms that are chosen for inclusion
   in this list MUST be one-way cryptographic hash functions that may be
   used by a client to securely verify a short-lived DNS message with a
   maximum message size constraint of 64 KiB.

   +----------+-------+-----------+-----------------------------+
   | Value(s) | Name | Length | Status, Remarks |
   +----------+-------+-----------+-----------------------------+
0	EMPTY	0 octets	Empty digest (query only)
1	SHA-1	20 octets	Mandatory
2-239			Unassigned
240-254			Reserved for experimental use
255			Reserved
   +----------+-------+-----------+-----------------------------+

Appendix B. Change history (to be removed before publication)

   o draft-muks-dnsop-dns-message-checksums-01
      Reduced NONCE field to 8 bytes. Reduced ALGORITHM field to 1
      byte. Added note about risk of downgrade attack. Expanded IANA
      considerations section and algorithms appendix. Described
      behaviors further. Added notes on picking a suitable checksum
      algorithm. Updated cross references, language and grammar.

   o draft-muks-dnsop-dns-message-checksums-00

Initial draft (renamed version). Removed the NONCE-COPY field as
it is no longer necessary. Doubled the size of the NONCE field to
128 bits. Added sample checksum algorithms. Fixed incorrect
reference, language and grammar.

Author's Address

   Mukund Sivaraman
   Internet Systems Consortium
   950 Charter Street
   Redwood City, CA 94063
   US

   Email: muks@mukund.org
   URI: http://www.isc.org/

                Managing DS records from parent via CDS/CDNSKEY
                      draft-ogud-dnsop-maintain-ds-00

Abstract

   RFC7344 specifies how DNS trust can be maintained in-band between
   parent and child. There are two features missing in that
   specification: initial trust setup and removal of trust anchor. This
   document addresses both these omissions.

   Changing a domain's DNSSEC status can be a complicated matter
   involving many parties. Some of these parties, such as the DNS
   operator, might not even be known by all organisations involved. The
   inability to enable or disable DNSSEC via in-band signalling is seen
   as a problem or liability that prevents DNSSEC adoption at large
   scale. This document adds a method for in-band signalling of DNSSEC
   status changes.

   Initial trust is considered a much harder problem, this document will
   seek to clarify and simplify the initial acceptance policy.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   CDS/CDNSKEY [RFC7344] records are used to signal changes in trust
   anchors, this is a great way to maintain delegations when the DNS
   operator has no other way to inform the parent that changes are
   needed. RFC7344 contains no "delete" signal for the child to tell
   the parent that it wants to change the DNSSEC security of its domain.

   [RFC7344] punted the Initial Trust establishment question and left it
   to each parent to come up with an acceptance policy.

1.1. Removing DS

   This document introduces the delete option for both CDS and CDNSKEY.
   to allow a child to signal the parent to turn off DNSSEC. When a
   domain is moved from one DNS operator to another one, sometimes it is
   necessary to turn off DNSSEC to facilitate the change of DNS
   operator. Common scenarios include:

   1 moving from a DNSSEC operator to a non-DNSSEC capable one or one
      that does not support the same algorithms as the old one.

   2 moving to one that cannot/does-not-want to do a proper DNSSEC
      rollover.

   3 the domain holder does not want DNSSEC.

   4 when moving between two DNS operators that use disjoint sets of
      algorithms to sign the zone, thus algorithm roll can not be
      performed.

   Whatever the reason, the lack of a "remove my DNSSEC" option is
   turning into the latest excuse as why DNSSEC cannot be deployed.

   Turing off DNSSEC reduces the security of the domain and thus should
   only be done carefully, and that decision should be fully under the
   child domain's control.

1.2. Introducing DS

   The converse issue is how does a child domain instruct the parent it
   wants to have a DS record added. This problem is not as hard as many
   have assumed, given a few simplifying assumptions. This document
   makes the assumption that there are reasonable policies that can be
   applied and will allow automation of trust introduction.

   Not being able to enable trust via an easily automated mechanism is
   hindering DNSSEC at scale by anyone that does not have automated
   access to its parent's "registry".

1.3. Notation

   When this document uses the word CDS it implies that the same applies
   to CDNSKEY and vice versa, the only difference between the two
   records is how information is represented.

   When the document uses the word "parent" it implies an entity that is
   authorized to insert into parent zone information about this child
   domain. Which entity this is exactly does not matter. It could be

the Registrar or Reseller that the child domain was purchased from.
It could be the Registry that the domain is registered in when
allowed. It could be some other entity when the RRR framework is not
used.

We use RRR to mean Registry Registrar Reseller in the context of DNS
domain markets.

1.4. 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].

2. The Three Uses of CDS

In general there are three operations that a domain wants to
influence on its parent:

1 Roll over KSK, this means updating the DS records in the parent to
   reflect the new set of KSK's at the child. This could be an ADD
   operation, a Delete operation on one or more records while keeping
   at least one DS RR, or a full Replace operation

2 Turn off DNSSEC validation, i.e. delete all the DS records

3 Enable DNSSEC validation, i.e. place initial DS RRset in the
   parent.

Operation 1 is covered in [RFC7344], operations 2 and 3 are defined
in this document. In many people's minds, those two later operations
carry more risk than the first one. This document argues that 2 is
identical to 1 and the final one is different (but not that
different).

2.1. The meaning of CDS ?

The fundamental question is what is the semantic meaning of
publishing a CDS RRset in a zone? We offer the following
interpretation:

"Publishing a CDS or CDNSKEY record signifies to the parent that the
child is ready for the corresponding DS records to be synchronized.
Every parent or parental agent should have an acceptance policy of
these records for the three different use cases involved: Initial DS
publication, Key rollover, and Returning to Insecure."

In short, the CDS RRset is an instruction to the parent to modify DS
RRset if the CDS and DS RRsets differ. The acceptance policy for CDS
in the rollover case is "seeing" according to [RFC7344]. The
acceptance policy in the Delete case is just seeing a CDS RRset with
the delete operation specified in this document.

3. Enabling DNSSEC via CDS/CDNSKEY

There are number of different models for managing initial trust, but
in the general case, the child wants to enable global validation for
the future. Thus during the period from the time the child publishes
the CDS until the corresponding DS is published is the period that
DNS answers for the child could be forged. The goal is to keep this
period as short as possible.

One important case is how a 3rd party DNS operator can upload its
DNSSEC information to the parent, so the parent can publish a DS
record for the child. In this case there is a possibility of setting
up some kind of authentication mechanism and submission mechanism
that is outside the scope of this document.

Below are some policies that parents can use. These policies assume
that the notifications are can be authenticated and/or identified.

3.1. Accept policy via authenticated channel

In this case the parent is notified via UI/API that CDS exists, the
parent retrieves the CDS and inserts the DS record as requested, if
the request comes over an authenticated channel.

3.2. Accept with extra checks

In this case the parent checks that the source of the notification is
allowed to request the DS insertion. The checks could include
whether this is a trusted entity, whether the nameservers correspond
to the requestor, whether there have been any changes in registration
in the last few days, etc, or the parent can send a notification
requesting an confirmation.

The end result is that the CDS is accepted at the end of the checks
or when the out-of-band confirmation is received.

3.3. Accept after delay

In this case, if the parent deems the request valid, it starts
monitoring the CDS records at the child nameservers over period of
time to make sure nothing changes. After number of checks,

preferably from different vantage points, the parent accepts the CDS
records as a valid signal to update.

3.4. Accept with challenge

In this case the parent instructs the requestor to insert some record
into the child domain to prove it has the ability to do so (i.e., it
is the operator of the zone).

4. DNSSEC Delete Algorithm

The DNSKEY algorithm registry contains two reserved values: 0 and
255[RFC4034]. The CERT record [RFC4398] defines the value 0 to mean
the algorithm in the CERT record is not defined in DNSSEC.

[rfc-editor remove before publication] For this reason, using the
value 0 in CDS/CDNSKEY delete operations is potentially problematic,
but we propose that here anyway as the risk is minimal. The
alternative is to reserve one DNSSEC algorithm number for this
purpose. [rfc-editor end remove]

Right now, no DNSSEC validator understands algorithm 0 as a valid
signature algorithm, thus if the validator sees a DNSKEY or DS record
with this value, it will treat it as unknown. Accordingly, the zone
is treated as unsigned unless there are other algorithms present.

In the context of CDS and CDNSKEY records, DNSSEC algorithm 0 is
defined and means the entire DS set MUST be removed. The contents of
the records MUST contain only the fixed fields as show below.

1 CDS 0 0 0

2 CDNSKEY 0 3 0

There is no keying material payload in the records, just the command
to delete all DS records. This record is signed in the same way as
CDS/CDNSKEY is signed.

Strictly speaking the CDS record could be "CDS X 0 X" as only the
DNSKEY algorithm is what signals the delete operation, but for
clarity the "0 0 0" notation is mandated, this is not a definition of
DS Digest algorithm 0. Same argument applies to "CDNSKEY 0 3 0".

Once the parent has verified the CDS/CDNSKEY record and it has passed
other acceptance tests, the DS record MUST be removed. At this point
the child can start the process of turning DNSSEC off.

5. Security considerations

   This document is about avoiding validation failures when a domain
   moves from one DNS operator to another one. Turing off DNSSEC
   reduces the security of the domain and thus should only be done as a
   last resort.

   In most cases it is preferable that operators collaborate on the
   rollover by doing a KSK+ZSK rollover as part of the handoff, but that
   is not always possible. This document addresses the case where
   unsigned state is needed.

   Users SHOULD keep in mind that re-establishing trust in delegation
   can be hard and take a long time thus before going to unsigned all
   options SHOULD be considered.

   A parent should ensure that when it is allowing a child to become
   securely delegated, that it has a reasonable assurance that the CDS/
   CDNSKEY that is used to bootstrap the security on is visible from a
   geographically and network topology diverse view. It should also
   ensure the the zone would validate if the parent published the DS
   record. A parent zone might also consider sending an email to its
   contact addresses to give the child a warning that security will be
   enabled after a certain about of wait time - thus allowing a child
   administrator to cancel the request.

   This document does not introduce any new problems, but like Negative
   Trust Anchor[I-D.ietf-dnsop-negative-trust-anchors], it addresses
   operational reality.

6. IANA considerations

   This document updates the following IANA registries: "DNS Security
   Algorithm Numbers"

   Algorithm 0 adds a reference to this document.

7. References

7.1. Normative References

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

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

7.2. Informative References

   [I-D.ietf-dnsop-negative-trust-anchors]
               Ebersman, P., Kumari, W., Griffiths, C., Livingood, J.,
               and R. Weber, "Definition and Use of DNSSEC Negative Trust
               Anchors", draft-ietf-dnsop-negative-trust-anchors-13 (work
               in progress), August 2015.

   [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>.

   [RFC4398]   Josefsson, S., "Storing Certificates in the Domain Name
               System (DNS)", RFC 4398, DOI 10.17487/RFC4398, March 2006,
               <http://www.rfc-editor.org/info/rfc4398>.

Appendix A. Acknowledgements

   This document is generated using the mmark tool that Miek Gieben has
   developed.

Authors' Addresses

   Olafur Gudmundsson
   CloudFlare

   Email: olafur+ietf@cloudflare.com

   Paul Wouters
   Red Hat

   Email: pwouters@redhat.com

        Clarifications to the Dynamic Updates in the Domain Name System (DNS
                          UPDATE) specification
                    draft-spacek-dnsop-update-clarif-01

Abstract

   This document clarifies interaction among Dynamic Updates in the
   Domain Name System (DNS UPDATE), Classless IN-ADDR.ARPA delegation,
   and Secure Domain Name System (DNS) Dynamic Update in the presence of
   CNAME/DNAME redirections.

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 February 28, 2016.

Copyright Notice

Table of Contents

1. Introduction

   This document clarifies interaction among Dynamic Updates in the
   Domain Name System (DNS UPDATE) [RFC2136], Classless IN-ADDR.ARPA
   delegation [RFC2317], and Secure Domain Name System (DNS) Dynamic
   Update [RFC3007].

   It was identified that common implementations using DNS update
   protocol often ignore existence of CNAME/DNAME redirections and, as a
   result, fail to update records if redirection is used. One common
   example is failure to update PTR records in classless IN-ADDR.ARPA
   zones.

   [RFC2317] describes how to use the CNAME records in IN-ADDR.ARPA DNS
   zones to split administrative control over IN-ADDR.ARPA data for
   classless networks. The described method is perfectly compatible
   with standard DNS resolution but DNS update requests need special
   handling described in this document.

   This clarification is applicable to parties wanting to update records
   in IN-ADDR.ARPA and other zones without changing existing CNAME/DNAME
   redirections. A typical example are PTR record updates in zones
   which might potentially use [RFC2317]. This clarification is not
   applicable to cases where the purpose of the DNS update is to change
   CNAME/DNAME redirection.

2. Document Conventions

   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].

   Terms "requestor", "update message", and names of update message's
   sections are used in the same sense as in [RFC2136].

   Examples involving IN-ADDR.ARPA zone and PTR records are referring to
   [RFC2317].

3. Problem Description

   The problem described herein typically occurs when an implementation
   intends to update resource records resolvable by using particular
   owner name while keeping all CNAME/DNAME redirections intact. In
   other words, the purpose of the update is to change resource records
   associated with terminal node of (potential) chain of redirections
   starting at a known owner name.

   Typically, this is the case when the resource records are associated
   with a known owner name or an owner name that is derived from data
   obtained outside of DNS. For example, implementations often
   translate IPv4 address to DNS owner name using the algorithm from
   [RFC1034] section 5.2.1.4:

   192.0.2.1 -> 1.2.0.192.in-addr.arpa.

   The problem is that implementations often use this original node name
   in an Update Message without checking for redirections. If the
   original owner name contains redirection, then this behavior results
   in an attempt to add or delete another record to or from a node that
   already contains the CNAME record, and the update fails.

   Such inappropriately constructed update request will be silently
   ignored in accordance with [RFC2136] section 3.4.2.2. Alternatively,
   an error will be reported to the requestor if the non-existence of
   the CNAME record was added as a prerequisite to the Update Message.

4. Clarification to Requestor Behaviour

   Please see applicability note in Introduction (Section 1, Paragraph
   4).

   A Requestor MUST resolve (canonicalize) the original owner name (e.g.
   the one derived from an IPv4 address) to a canonical owner name
   before constructing the Update Message. The requestor MUST follow
   whole chain of redirections until the terminal node of the chain is
   reached and use canonical name found at the terminal node.
   Implementations MUST detect infinite loops.

   Canonical owner name MUST be used instead of the original owner name
   in the resulting Update Message:

   o All names used in the Prerequisite and Update sections MUST be
      canonicalized as specified above. Only prerequisites concerning
      the CNAME or DNAME records are an exception to this rule and
      should not be canonicalized.

o ZNAME in the Zone Section has to contain the name of the zone that
   encloses the canonical owner names.

o An implementation MAY chose to use canonicalized names in RDATA
   and an Additional Section. This is an application specific
   decision.

5. IANA Considerations

   This draft does not involve IANA Considerations.

6. Security Considerations

   Canonicalization process changes the owner name which is going to be
   affected by the update. An active attacker might interfere with the
   canonicalization process and trick the requestor to update a node of
   the attacker's choice if the canonicalization process is not secured
   by using DNSSEC or by other means.

   Security properties of DNS updates using only DNS UPDATE [RFC2136]
   without any security mechanisms on top of it are vulnerable anyway
   because an active attacker can very well modify the update message
   itself.

   Canonicalization generally increases overall risk for implementations
   of Secure DNS Dynamic Update [RFC3007] because an attacker might have
   a chance to modify the owner name in an Update Message before the
   message is signed by the requestor. An implementation might decide
   to accept canonicalized names only on condition that the overall
   security status of the canonicalization process is sufficient
   according to the local policy. Because the chain of redirections
   might involve multiple DNS zones, implementations MUST use the lowest
   security status from all links in the chain of redirections when
   doing security decisions.

   For example, a strict implementation might accept canonicalized names
   only on condition that all redirections were secured by DNSSEC and
   the security state of all redirections was "secure". Another
   implementation might decide that security checks on a server side are
   sufficient, so requestors will accept canonical names obtained using
   insecure protocols. In case of PTR records, a server might require
   the TCP transport and map an IP address of the requestor to the
   canonical owner name and/or check data in an Update Message with the
   requestor's identity.

7. Normative References

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

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

   [RFC2136] Vixie, P., Thomson, S., Rekhter, Y., and J. Bound,
              "Dynamic Updates in the Domain Name System (DNS UPDATE)",
              RFC 2136, April 1997.

   [RFC2317] Eidnes, H., de Groot, G., and P. Vixie, "Classless IN-
              ADDR.ARPA delegation", BCP 20, RFC 2317, March 1998.

   [RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic
              Update", RFC 3007, November 2000.

Author's Address

   Petr Spacek
   Red Hat, Inc.

   Email: pspacek@redhat.com

                        The EDNS Key Tag Option
                      draft-wessels-edns-key-tag-00

Abstract

   The DNS Security Extensions (DNSSEC) were developed to provide origin
   authentication and integrity protection for DNS data by using digital
   signatures. These digital signatures can be verified by building a
   chain-of-trust starting from a trust anchor and proceeding down to a
   particular node in the DNS. This document specifies a way for
   validating end-system resolvers to signal to a server which keys are
   referenced in their chain-of-trust. The extensions allow zone
   administrators to monitor the progress of rollovers in a DNSSEC-
   signed zone.

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 January 30, 2016.

   to this document. Code Components extracted from this document must
   include Simplified BSD License text as described in Section 4.e of
   the Trust Legal Provisions and are provided without warranty as
   described in the Simplified BSD License.

Table of Contents

1. Introduction

   The DNS Security Extensions (DNSSEC) [RFC4033], [RFC4034] and
   [RFC4035] were developed to provide origin authentication and
   integrity protection for DNS data by using digital signatures.
   DNSSEC uses Key Tags to efficiently match signatures to the keys from
   which they are generated. The Key Tag is a 16-bit value computed
   from the RDATA portion of a DNSKEY RR using a formula not unlike a
   ones-complement checksum. RRSIG RRs contain a Key Tag field whose
   value is equal to the Key Tag of the DNSKEY RR that validates the
   signature.

   Likewise, Delegation Signer (DS) RRs also contain a Key Tag field
   whose value is equal to the Key Tag of the DNSKEY RR to which it
   refers.

   This draft sets out to specify a way for validating end-system
   resolvers to tell a server in a DNS query which DNSSEC key(s) they
   would use to validate the expected response. This is done using the
   new EDNS option specified below in Section 4 for use in the OPT
   meta-RR [RFC6891]. This new EDNS option code is OPTIONAL to
   implement and use.

   This proposed EDNS option serves to measure the acceptance and use of
   new trust anchors and key signing keys (KSKs). This signaling option
   can be used by zone administrators as a gauge to measure the
   successful deployment of new keys. This is of particular interest
   for the DNS root zone in the event of key and/or algorithm rollovers
   which relies on [RFC5011] to automatically update a validating end-
   system's trust anchor.

   This draft does not seek to introduce another process for rolling
   keys or updating trust anchors. Rather, this document specifies a
   means by which a client query can signal the set of keys that a
   client uses for DNSSEC validation.

2. Requirements 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].
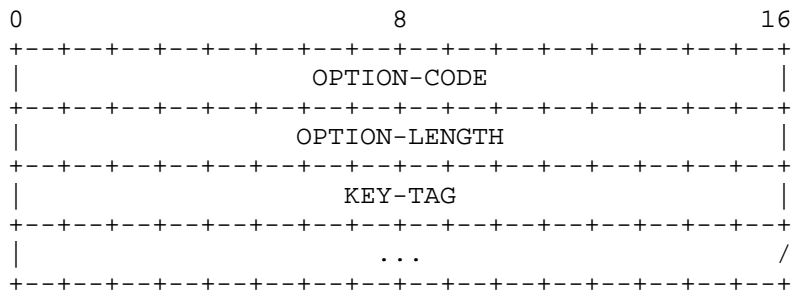

3. Terminology

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

Key Tag: A 16-bit integer that identifies and enables efficient
   selection of DNSSEC public keys. A Key Tag value can be computed
   over the RDATA of a DNSKEY RR. The Key Tag field in the RRSIG and
   DS records can be used to help select the corresponding DNSKEY RR
   efficiently when more than one candidate DNSKEY RR is available.
   For most algorithms the Key Tag is a simple 16-bit modular sum of
   the DNSKEY RDATA. See [RFC4034] Appendix B.

4. Option Format

The edns-key-tag option is encoded as follows:

```
0                       8                      16
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                 OPTION-CODE                   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                OPTION-LENGTH                  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                   KEY-TAG                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ...                        /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

OPTION-CODE: The EDNS0 option code assigned to edns-key-tag, [TBD].

OPTION-LENGTH: The value 2 x number of key-tag values present.

KEY-TAG: One or more 16-bit Key Tag values ([RFC4034], Appendix B).

5. Use By Queriers

A validating end-system resolver sets the edns-key-tag option in the
OPT meta-RR when sending a DNSKEY query. The validating end-system
resolver SHOULD also set the DNSSEC OK bit [RFC4034] to indicate that

it wishes to receive DNSSEC RRs in the response.

A DNS client MUST NOT include the edns-key-tag option for non-DNSKEY queries.

The KEY-TAG value(s) included in the edns-key-tag option represent the Key Tag of the Trust Anchor or DNSKEY RR that will be used to validate the expected response. When the client sends a DNSKEY query, the edns-key-tag option represents the Key Tag(s) of the KSK(s) of the zone for which the server is authoritative. A validating end-system resolver learns the Key Tag(s) of the KSK(s) from the zone's DS record(s) (found in the parent), or from a configured trust anchor.

A DNS client SHOULD include the edns-key-tag option when issuing a DNSKEY query for a zone corresponding to a configured Trust Anchor.

A DNS client MAY include the edns-key-tag option when issuing a DNSKEY query for a non-Trust Anchor zone (i.e., Key Tags learned via DS records). Since some DNSSEC validators implement bottom-up validation, non-Trust Anchor Key Tags zone might not be known at the time of the query. Such a validator can include the edns-key-tag option based on previously cached data.

A DNS client MUST NOT include Key Tag(s) for keys which are not learned via either configured Trust Anchor or DS records.

Since the edns-key-tag option is only set in the query, if a client sees these options in the response, no action needs to be taken and the client MUST ignore the option values.

5.1. Stub Resolvers

Typically, stub resolvers rely on an upstream recursive server (or cache) to provide a response. Optimal setting of the edns-key-tag option depends on whether the stub resolver elects to perform its own validation.

5.1.1. Validating Stub Resolvers

A validating stub resolver sets the DNSSEC OK (DO) bit [RFC4034] to indicate that it wishes to receive additional DNSSEC RRs (i.e., RRSIG RRs) in the response. Such validating resolvers SHOULD include the edns-key-tag option in the OPT RR when sending a DNSKEY query.

5.1.2. Non-validating Stub Resolvers

   The edns-key-tag option MUST NOT be included by non-validating stub
   resolvers.

5.2. Recursive Resolvers

5.2.1. Validating Recursive Resolvers

   A validating recursive resolver sets the edns-key-tag option when
   performing recursion based on relevant keys it knows and any edns-
   key-tag values in the stub client query. When the recursive server
   receives a query with the option set, the recursive server SHOULD set
   the edns-key-tag list for any outgoing iterative queries for that
   resolution chain to a union of the stub client's Key Tag(s) and the
   validating recursive resolver's Key Tag(s). For example, if the
   recursive resolver's Key Tag list is (19036, 12345) and the stub's
   list is (19036, 34567), the final edns-key-tag list would be (19036,
   12345, 34567).

   If the client included the DO and Checking Disabled (CD) bits, but
   did not include the edns-key-tag option in the query, the validating
   recursive resolver MAY include the option with its own Key Tag values
   in full.

   Validating recursive resolvers MUST NOT set the edns-key-tag option
   in the final response to the stub client.

5.2.2. Non-validating Recursive Resolvers

   Recursive resolvers that do not validate responses SHOULD copy the
   edns-key-tag option seen in received queries, as they represent the
   wishes of the validating downstream resolver that issued the original
   query.

6. Use By Responders

   An authoritative name server receiving queries with the edns-key-tag
   option MAY log or otherwise collect the Key Tag values to provide
   information to the zone operator.

   A responder MUST NOT include the edns-key-tag option in any DNS
   response.

7. IANA Considerations

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

```
+-------+--------------+----------+-----------------+
| Value | Name         | Status   | Reference       |
+-------+--------------+----------+-----------------+
| [TBA] | edns-key-tag | Optional | [This document] |
+-------+--------------+----------+-----------------+
```

8. Security Considerations

   This document specifies a way for a client to signal its trust anchor
   knowledge to a cache or server. The signals are optional codes
   contained in the OPT meta-RR used with EDNS. The goal of these
   options is to signal new trust anchor uptake in client code to allow
   zone administrators to know when it is possible to complete a key
   rollover in a DNSSEC-signed zone.

   There is a possibility that an eavesdropper or server could infer the
   validator in use by a client by the Key Tag list seen in queries.
   This may allow an attacker to find validators using old, possibly
   broken, keys. It could also be used to identify the validator or
   narrow down the possible validator implementations in use by a
   client, which could have a known vulnerability that could be
   exploited by the attacker.

   Consumers of data collected from the edns-key-tag option are advised
   that provided Key Tag values might be "made up" by some DNS clients
   with malicious or at least mischievous intentions.

   DNSSEC does not require keys in a zone to have unique Key Tags.
   During a rollover there is a small possibility that an old key and a
   new key will have identical Key Tag values. Zone operators relying
   on the edns-key-tag mechanism SHOULD take care to ensure that new
   keys have unique Key Tag values.

9. Privacy Considerations

   This proposal adds additional "signaling" to DNS queries in the form
   of Key Tag values. While Key Tag values themselves are not
   considered private information, it may be possible for an
   eavesdropper to use Key Tag values as a fingerprinting technique to
   identify particular DNS validating clients. This may be especially
   true if the validator is configured with trust anchor for zones in

addition to the root zone.

A validating end-system resolver need not transmit the edns-key-tag
option in every applicable query. Due to privacy concerns, such a
resolver MAY choose to transmit the edns-key-tag option for a subset
of queries (e.g., every 25th time), or by random chance with a
certain probability (e.g., 5%).

Implementations of this specification MAY be administratively
configured to only transmit the edns-key-tag option for certain
zones. For example, the software's configuration file may specify a
list of zones for which use of the option is allowed or denied.
Since the primary motivation for this specification is to provide
operational measurement data for root zone key rollovers, it is
RECOMMENDED that implementations at least include the edns-key-tag
option for root zone DNSKEY queries.

10. Acknowledgments

This document was inspired by and borrows heavily from [RFC6975] by
Scott Rose and Steve Crocker. The author would like to thank to
Casey Deccio and Burt Kalisky for early feedback.

11. References

11.1. Normative References

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

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

   [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S.

          Rose, "Resource Records for the DNS Security Extensions",
          RFC 4034, DOI 10.17487/RFC4034, March 2005,
          <http://www.rfc-editor.org/info/rfc4034>.

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

   [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>.

11.2. Informative References

   [RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC)
              Trust Anchors", STD 74, RFC 5011, DOI 10.17487/RFC5011,
              September 2007, <http://www.rfc-editor.org/info/rfc5011>.

   [RFC6975] Crocker, S. and S. Rose, "Signaling Cryptographic
              Algorithm Understanding in DNS Security Extensions
              (DNSSEC)", RFC 6975, DOI 10.17487/RFC6975, July 2013,
              <http://www.rfc-editor.org/info/rfc6975>.

Author's Address

   Duane Wessels
   Verisign Labs
   12061 Bluemont Way
   Reston, VA 20190

   Phone: +1 703 948-3200
   Email: dwessels@verisign.com
   URI: http://verisigninc.com