

DNS Operations(dnsop)
Internet-Draft
Intended status: Informational
Expires: January 4, 2015

K. Fujiwara
JPRS
July 3, 2014

Detection and countermeasure of forged response cache poisoning attacks
draft-fujiwara-dnsop-poisoning-measures-00.txt

Abstract

Although the Domain Name System Security (DNSSEC) Extensions has been implemented, cache poisoning is still a big issue. "ID Guessing and Query Prediction" type cache poisoning is detectable on a full resolver. TCP transport has strong resistance to cache poisoning attacks. This document proposes an improvement of full resolvers about the detection and the measure against forged response cache poisoning attacks.

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 4, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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. Introduction	2
2. Detection	2
3. Measures to forged response attacks	3
4. Possible solution	3
5. Security considerations	3
6. IANA Considerations	4
7. Normative References	4

1. Introduction

"Threat Analysis of the Domain Name System (DNS)" [RFC3833] described "ID Guessing and Query Prediction" and brute force attacks. Dan Kaminsky proposed effective attack method [DK2008]. "Wikipedia DNS_spoofing" [Wikipedia_DNS_spoofing] describes concrete attack patterns.

It is difficult to distinguish a forged response from an authentic response as the identity fields such as port number and query ID can be guessed easily under certain circumstances. "Redirect the target domain's name server" attack is effective because it forges delegation information. Kaminsky offered the continuation attack method to increase an attack probability.

"Detection" of forged response attacks is described in Section 2. A Measure to forged response attacks is described in Section 3. A possible solution is described in Section 4.

2. Detection

Attacks described in Section 1 hardly success by one-time trial in almost all cases. The probability of success by one-time trial is $1 / (\text{number of Query IDs}, 2^{16}) / (\text{number of ports}, 2^{16} - 1024) / (\text{number of DNS servers of the domain name})$. A full resolver under attack receives many unmatched responses which have different query IDs, port numbers, IP addresses, or query names. Most of unmatched responses are cache poisoning attacks.

These responses contain resource records which attackers want to inject to the cache of the full resolver. Attacked domain names can be picked up by parsing unmatched responses.

Detailed logs are useful for DNS server operations. They should contain resource records which attackers want to inject.

Log aggregation is important since number of forged responses may be too many and logging takes many resources.

The log should contain summarized data from source IP addresses, destination IP address, destination port number, query names, query types, NS and glue RRs.

3. Measures to forged response attacks

Using TCP as a DNS transport is a good countermeasure against forged responses attacks. First, each TCP packet has 32bit sequence number field and predicting sequence numbers and timing control are very hard. Second, the attacker need to inject at least two packets: one is to establish a TCP connection and the other is to send a forged response.

Using TCP transport may cause two issues. First, it increases query response time. Second, it causes performance issues to both full resolvers and authoritative DNS servers.

4. Possible solution

A feasible measure is a combination of the detection and the use of TCP transport. A full resolver detects forged response attacks described in Section 2. If an attack is detected, the full resolver invalidate name resolution states which contain target-of-attack domain names and restart the name resolution using TCP transport. If forged response attacks are stopped, the full resolver detects it and resume to use UDP transport for the attacked domains. The changing delay may be a same value as timeout of the waiting for the response from authoritative DNS servers.

This idea may be well known and some products may implement it already. They may have patents.

Encryption of DNS traffic discussed on the dns-privacy mailing list [dns-privacy] is good countermeasure against forged response attacks.

5. Security considerations

Idea described in Section 4 may cause a new weak point. Attackers can force the full resolver to use TCP transport for a domain name by sending small number of forged responses. This attack increases the full resolver's state and load, authoritative DNS servers' states.

6. IANA Considerations

7. Normative References

- [DK2008] "DNS 2008 and the new (old) nature of critical
 infrastructure,
 <http://www.slideshare.net/dakami/dmk-bo2-k8bhfed>", July
 2008.
- [RFC3833] Atkins, D. and R. Austein, "DNS Threat Analysis", RFC
 3383, August 2004.
- [Wikipedia_DNS_spoofing]
 "DNS spoofing, http://en.wikipedia.org/wiki/DNS_spoofing",
 .

Author's Address

Kazunori Fujiwara
Japan Registry Services Co., Ltd.
Chiyoda First Bldg. East 13F, 3-8-1 Nishi-Kanda
Chiyoda-ku, Tokyo 101-0065
Japan

Phone: +81 3 5215 8451
EMail: fujiwara@jprs.co.jp

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

L. Howard
Time Warner Cable
June 26, 2014

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

Abstract

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

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 25, 2014.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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.

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

Table of Contents

1. Introduction	3
1.1. Reverse DNS in IPv4	3
1.2. Reverse DNS Considerations in IPv6	4
2. Alternatives in IPv6	5
2.1. No Response	5
2.2. Wildcard match	5
2.3. Dynamic DNS	6
2.3.1. Dynamic DNS from Individual Hosts	6
2.3.2. Dynamic DNS through Residential Gateways	7
2.3.3. Dynamic DNS Delegations	7
2.3.4. Generate Dynamic Records	8
2.3.5. Populate from DHCP Server	8
2.3.6. Populate from RADIUS Server	9
2.4. Delegate DNS	9
2.5. Dynamically Generate PTR When Queried ("On the Fly")	9
3. Recommendations	10
4. Security Considerations	10
4.1. Using Reverse DNS for Security	10
4.2. DNS Security with Dynamic DNS	10
4.3. Considerations for Other Uses of the DNS	11
5. Acknowledgements	11
6. IANA Considerations	11
7. References	11
7.1. Normative References	11
7.2. Informative References	12
Author's Address	12

1. Introduction

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

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

1.1. Reverse DNS in IPv4

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

```
1.2.0.192.IN-ADDR.ARPA.  IN PTR 1.user.town.AW.example.com.
2.2.0.192.IN-ADDR.ARPA.  IN PTR 2.user.town.AW.example.com.
3.2.0.192.IN-ADDR.ARPA.  IN PTR 3.user.town.AW.example.com.
.
.
.
254.2.0.192.IN-ADDR.ARPA.  IN PTR 254.user.town.AW.example.com.
```

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

```
1.user.town.AW.example.com.  IN A 192.0.2.1
2.user.town.AW.example.com.  IN A 192.0.2.2
3.user.town.AW.example.com.  IN A 192.0.2.3
.
```

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

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

1.2. Reverse DNS Considerations in IPv6

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

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

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

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

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

2. Alternatives in IPv6

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

2.1. Negative Response

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

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

2.2. Wildcard match

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

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

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

2.3. Dynamic DNS

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

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

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

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

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

2.3.1. Dynamic DNS from Individual Hosts

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

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

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

2.3.2. Dynamic DNS through Residential Gateways

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

2.3.3. Automatic DNS Delegations

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

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

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

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

2.3.4. Generate Dynamic Records

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

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

2.3.5. Populate from DHCP Server

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

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

2.3.6. Populate from RADIUS Server

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

2.4. Delegate DNS

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

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

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

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

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

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

be acceptable.

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

3. Recommendations

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

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

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

4. Security Considerations

4.1. Using Reverse DNS for Security

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

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

4.2. DNS Security with Dynamic DNS

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

Interactions with DNSsec are described throughout this document.

4.3. Considerations for Other Uses of the DNS

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

5. Acknowledgements

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

6. IANA Considerations

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

7. References

7.1. Normative References

- [RFC1033] Lottor, M., "Domain Administrators Operators Guide", November 1987.
- [RFC1912] Barr, D., "Common DNS Operational and Configuration Errors", February 1996.
- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", April 1997.
- [RFC2845] "Secret Key Transaction Authentication for DNS (TSIG)".
- [RFC2865] "Remote Authentication Dial In User Service (RADIUS)".

- [RFC2866] "RADIUS Accounting".
- [RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", November 2000.
- [RFC3646] Droms, R., Ed., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", December 2003.
- [RFC4033] "DNS Security Introduction and Requirements".
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", March 2005.
- [RFC4592] Lewis, E., "The Role of Wildcards in the Domain Name System", July 2006.
- [RFC4704] Stapp, M., Volz, Y., and Y. Rekhter, "The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option".
- [RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", September 2007.
- [RFC4941] "Privacy Extensions for Stateless Address Autoconfiguration in IPv6".
- [RFC6106] "IPv6 Router Advertisement Options for DNS Configuration".

7.2. Informative References

- [RFC2317] Eidnes, H., de Groot, G., and P. Vixie, "Classless IN-ADDR.ARPA delegation", March 1998.
- [RFC2535] Eastlake, D., "Domain Name System Security Extensions", March 1999.
- [RFC4472] Durand, A., Ihren, J., and P. Savola, "Operational Considerations and Issues with IPv6 DNS", April 2006.
- [RFC6092] Woodyatt, J., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", January 2011.
- [inaddr-reqd] Senie, D., "draft-ietf-dnsop-inaddr-required-07",

August 2005.

[rmap-consider]

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

Author's Address

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

Email: lee.howard@twcable.com

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 16, 2015

S. Morris
ISC
J. Ihren
Netnod
J. Dickinson
Sinodun
W. Mekking
NLnet Labs
October 13, 2014

DNSSEC Key Rollover Timing Considerations
draft-ietf-dnsop-dnssec-key-timing-06.txt

Abstract

This document describes the issues surrounding the timing of events in the rolling of a key in a DNSSEC-secured zone. It presents timelines for the key rollover and explicitly identifies the relationships between the various parameters affecting the process.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 16, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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. Introduction	3
1.1. Key Rolling Considerations	3
1.2. Types of Keys	4
1.3. Terminology	4
1.4. Limitation of Scope	4
2. Rollover Methods	5
2.1. ZSK Rollovers	5
2.2. KSK Rollovers	6
3. Key Rollover Timelines	8
3.1. Key States	8
3.2. Zone-Signing Key Timelines	9
3.2.1. Pre-Publication Method	9
3.2.2. Double-Signature Method	12
3.3. Key-Signing Key Rollover Timelines	14
3.3.1. Double-KSK Method	14
3.3.2. Double-DS Method	16
3.3.3. Double-RRset Method	19
3.3.4. Interaction with Configured Trust Anchors	21
3.3.5. Introduction of First Keys	22
4. Standby Keys	23
5. Algorithm Considerations	24
6. Summary	24
7. IANA Considerations	25
8. Security Considerations	25
9. Acknowledgements	25
10. Normative References	25
Appendix A. List of Symbols	25
Appendix B. Change History (To be removed on publication)	28
Authors' Addresses	30

1. Introduction

1.1. Key Rolling Considerations

When a zone is secured with DNSSEC, the zone manager must be prepared to replace ("roll") the keys used in the signing process. The rolling of keys may be caused by compromise of one or more of the existing keys, or it may be due to a management policy that demands periodic key replacement for security or operational reasons. In order to implement a key rollover, the keys need to be introduced into and removed from the zone at the appropriate times.

Considerations that must be taken into account are:

- o DNSKEY records and associated information (such as the DS records or RRSIG records created with the key) are not only held at the authoritative nameserver, they are also cached by resolvers. The data on these systems can be interlinked, e.g., a validating resolver may try to validate a signature retrieved from a cache with a key obtained separately.
- o Zone "boot-strapping" events, where a zone is signed for the first time, can be common in configurations where a large number of zones are being served. Procedures should be able to cope with the introduction of keys into the zone for the first time as well as "steady-state", where the records are being replaced as part of normal zone maintenance.
- o To allow for an emergency re-signing of the zone as soon as possible after a key compromise has been detected, standby keys (additional keys over and above those used to sign the zone) need to be present.
- o A query for the DNSKEY RRset returns all DNSKEY records in the zone. As there is limited space in the UDP packet (even with EDNS0 support), key records no longer needed must be periodically removed. (For the same reason, the number of standby keys in the zone should be restricted to the minimum required to support the key management policy.)

Management policy, e.g., how long a key is used for, also needs to be considered. However, the point of key management logic is not to ensure that a rollover is completed at a certain time but rather to ensure that no changes are made to the state of keys published in the zone until it is "safe" to do so ("safe" in this context meaning that at no time during the rollover process does any part of the zone ever go bogus). In other words, although key management logic enforces policy, it may not enforce it strictly.

A high-level overview of key rollover can be found in [RFC6781]. In contrast, this document focuses on the low-level timing detail of two classes of operations described there, the rollover of zone-signing keys (ZSKs), and the rollover of key-signing keys (KSKs).

Note that the process for the introduction of keys into a zone is different to that of rolling a key; see Section 3.3.5 for more information about that topic.

1.2. Types of Keys

Although DNSSEC validation treats all keys equally, [RFC4033] recognises the broad classification of ZSKs and KSKs. A ZSK is used to authenticate information within the zone; a KSK is used to authenticate the zone's DNSKEY RRset. The main implication for this distinction concerns the consistency of information during a rollover.

During operation, a validating resolver must use separate pieces of information to perform an authentication. At the time of authentication, each piece of information may be in its cache or may need to be retrieved from an authoritative server. The rollover process needs to happen in such a way that at all times during the rollover the information is consistent. With a ZSK, the information is the RRSIG (plus associated RRset) and the DNSKEY. These are both obtained from the same zone. In the case of the KSK, the information is the DNSKEY and DS RRset with the latter being obtained from a different zone.

Although there are similarities in the algorithms to roll ZSKs and KSKs, there are a number of differences. For this reason, the two types of rollovers are described separately.

1.3. Terminology

The terminology used in this document is as defined in [RFC4033] and [RFC5011].

A number of symbols are used to identify times, intervals, etc. All are listed in Appendix A.

1.4. Limitation of Scope

This document represents current thinking at the time of publication. However, the subject matter is evolving and it is not possible for the document to be comprehensive. In particular, it does not cover:

- o Rolling a key that is used as both a ZSK and KSK.
- o Algorithm rollovers. Only the rolling of keys of the same algorithm are described here, not transitions between algorithms.

The reader is therefore reminded that DNSSEC is, as of date of publication, in the early stages of deployment, and best practices may further develop over time.

2. Rollover Methods

2.1. ZSK Rollovers

For ZSKs, the issue for the zone operator/signer is to ensure that any caching validator which has access to a particular signature also has access to the corresponding valid ZSK.

A ZSK can be rolled in one of three ways:

- o Pre-Publication: described in [RFC6781], the new key is introduced into the DNSKEY RRset which is then re-signed. This state of affairs remains in place for long enough to ensure that any cached DNSKEY RRsets contain both keys. At that point signatures created with the old key can be replaced by those created with the new key, and the old signatures removed. During the re-signing process (which may or may not be atomic depending on how the zone is managed), it doesn't matter which key an RRSIG record retrieved by a resolver was created with; cached copies of the DNSKEY RRset will contain both the old and new keys.

Once the zone contains only signatures created with the new key, there is an interval during which RRSIG records created with the old key expire from caches. After this, there will be no signatures anywhere that were created using the old key, and it can be removed from the DNSKEY RRset.

- o Double-Signature: also mentioned in [RFC6781], this involves introducing the new key into the zone and using it to create additional RRSIG records; the old key and existing RRSIG records are retained. During the period in which the zone is being signed (again, the signing process may not be atomic), validating resolvers are always able to validate RRSIGs: any combination of old and new DNSKEY RRset and RRSIGs allows at least one signature to be validated.

Once the signing process is complete and enough time has elapsed to make sure that all validators that have the DNSKEY and

signatures in cache have both the old and new information, the old key and signatures can be removed from the zone. As before, during this period any combination of DNSKEY RRset and RRSIGs will allow validation of at least one signature.

- o Double-RRSIG: strictly speaking, the use of the term "Double-Signature" above is a misnomer as the method is not only double signature, it is also double key as well. A true Double-Signature method (here called the Double-RRSIG method) involves introducing new signatures in the zone (while still retaining the old ones) but not introducing the new key.

Once the signing process is complete and enough time has elapsed to ensure that all caches that may contain an RR and associated RRSIG have a copy of both signatures, the key is changed. After a further interval during which the old DNSKEY RRset expires from caches, the old signatures are removed from the zone.

Of the three methods, Double-Signature is conceptually the simplest - introduce the new key and new signatures, then approximately one TTL later remove the old key and old signatures. It is also the fastest, but suffers from increasing the size of the zone and the size of responses.

Pre-Publication is more complex - introduce the new key, approximately one TTL later sign the records, and approximately one TTL after that remove the old key. It does however keep the zone and response sizes to a minimum.

Double-RRSIG is essentially the reverse of Pre-Publication - introduce the new signatures, approximately one TTL later change the key, and approximately one TTL after that remove the old signatures. However, it has the disadvantage of the Pre-Publication method in terms of time taken to perform the rollover, the disadvantage of the Double-Signature rollover in terms of zone and response sizes, and none of the advantages of either. For these reasons, it is unlikely to be used in any real-world situations and so will not be considered further in this document.

2.2. KSK Rollovers

In the KSK case, there should not be a problem that a caching validator does not have access to a particular signature that corresponds to a valid KSK. The KSK is only used for one signature (that over the DNSKEY RRset) and both the key and the signature travel together. Instead, the issue is to ensure that the KSK is trusted.

Trust in the KSK is either due to the existence of a signed and validated DS record in the parent zone or an explicitly configured trust anchor. If the former, the rollover algorithm will need to involve the parent zone in the addition and removal of DS records, so timings are not wholly under the control of the zone manager. If the latter, [RFC5011] timings will be needed to roll the keys. (Even in the case where authentication is via a DS record, the zone manager may elect to include [RFC5011] timings in the key rolling process so as to cope with the possibility that the key has also been explicitly configured as a trust anchor.)

It is important to note that the need to interact with the parent does not preclude the development of key rollover logic; in accordance with the goal of the rollover logic being able to determine when a state change is "safe", the only effect of being dependent on the parent is that there may be a period of waiting for the parent to respond in addition to any delay the key rollover logic requires. Although this introduces additional delays, even with a parent that is less than ideally responsive the only effect will be a slowdown in the rollover state transitions. This may cause a policy violation, but will not cause any operational problems.

Like the ZSK case, there are three methods for rolling a KSK:

- o Double-KSK: the new KSK is added to the DNSKEY RRset which is then signed with both the old and new key. After waiting for the old RRset to expire from caches, the DS record in the parent zone is changed. After waiting a further interval for this change to be reflected in caches, the old key is removed from the RRset.
- o Double-DS: the new DS record is published. After waiting for this change to propagate into caches, the KSK is changed. After a further interval during which the old DNSKEY RRset expires from caches, the old DS record is removed.
- o Double-RRset: the new KSK is added to the DNSKEY RRset which is then signed with both the old and new key, and the new DS record added to the parent zone. After waiting a suitable interval for the old DS and DNSKEY RRsets to expire from caches, the old DNSKEY and DS record are removed.

In essence, Double-KSK means that the new KSK is introduced first and used to sign the DNSKEY RRset. The DS record is changed, and finally the old KSK removed. It limits interactions with the parent to a minimum but, for the duration of the rollover, the size of the DNSKEY RRset is increased.

With Double-DS, the order of operations is the other way round:

introduce the new DS, change the DNSKEY, then remove the old DS. The size of the DNSKEY RRset is kept to a minimum, but two interactions are required with the parent.

Finally, Double-RRset is the fastest way to roll the KSK, but has the drawbacks of both of the other methods: a larger DNSKEY RRset and two interactions with the parent.

3. Key Rollover Timelines

3.1. Key States

DNSSEC validation requires both the DNSKEY and information created from it (referred to as "associated data" in this section). In the case of validation of an RR, the data associated with the the key is the corresponding RRSIG. Where there is a need to validate a chain of trust, the associated data is the DS record.

During the rolling process, keys move through different states. The defined states are:

Generated	Although keys may be created immediately prior to first use, some implementations may find it convenient to create a pool of keys in one operation and draw from it as required. (Note: such a pre-generated pool must be secured against surreptitious use.) Keys that have been created but not yet used are said to be in the "Generated" state.
Published	A key enters the published state when either it or its associated data first appears in the appropriate zone.
Ready	The DNSKEY or its associated data have been published for long enough to guarantee that copies of the key(s) it is replacing (or associated data related to that key) have expired from caches.
Active	The data is starting to be used for validation. In the case of a ZSK, it means that the key is now being be used to sign RRsets and that both it and the created RRSIGs appear in the zone. In the case of a KSK, it means that it is possible to use it to validate a DNSKEY RRset as both the DNSKEY and DS records are present in their relevant zones. Note that when this state is entered, it may not be possible for validating resolvers to use the data for validation in all cases: the zone signing may not have finished, or the data might not have reached the

resolver because of propagation delays and/or caching issues. If this is the case, the resolver will have to rely on the predecessor data instead.

Retired The data has ceased to be used for validation. In the case of a ZSK, it means that the key is no longer used to sign RRsets. In the case of a KSK, it means that the successor DNSKEY and DS records are in place. In both cases, the key (and its associated data) can be removed as soon as it is safe to do so, i.e. when all validating resolvers are able to use the new key and associated data to validate the zone. However, until this happens, the current key and associated data must remain in their respective zones.

Dead The key and its associated data are present in their respective zones, but there is no longer information anywhere that requires their presence for use in validation. Hence they can be removed at any time.

Removed Both the DNSKEY and its associated data have been removed from their respective zones.

There is one additional state, used where [RFC5011] considerations are in effect (see Section 3.3.4):

Revoked The DNSKEY is published for a period with the "revoke" bit set as a way of notifying validating resolvers that have configured it as an [RFC5011] trust anchor that it is about to be removed from the zone.

3.2. Zone-Signing Key Timelines

The following sections describe the rolling of a ZSK. They show the events in the lifetime of a key (referred to as "key N") and cover its replacement by its successor (key N+1).

3.2.1. Pre-Publication Method

In this method, the new key is introduced into the DNSKEY RRset. After enough time to ensure that any cached DNSKEY RRsets contain both keys, the zone is signed using the new key and the old signatures are removed. Finally, when all signatures created with the old key have expired from caches, the old key is removed.

The following diagram shows the timeline of a Pre-Publication rollover. Time increases along the horizontal scale from left to right and the vertical lines indicate events in the process.

Significant times and time intervals are marked.

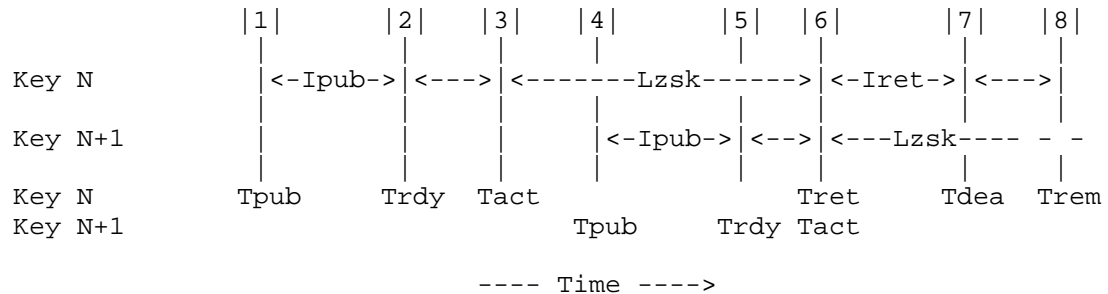


Figure 1: Timeline for a Pre-Publication ZSK rollover.

Event 1: Key N's DNSKEY record is put into the zone, i.e. it is added to the DNSKEY RRset which is then re-signed with the currently active key-signing keys. The time at which this occurs is the publication time (Tpub), and the key is now said to be published. Note that the key is not yet used to sign records.

Event 2: Before it can be used, the key must be published for long enough to guarantee that any cached version of the zone's DNSKEY RRset includes this key.

This interval is the publication interval (Ipub) and, for the second or subsequent keys in the zone, is given by:

$$I_{pub} = D_{prp} + TTL_{key}$$

Here, Dprp is the propagation delay - the time taken for a change introduced at the master to replicate to all name servers. TTLkey is the time-to-live (TTL) for the DNSKEY records in the zone. The sum is therefore the maximum time taken for existing DNSKEY records to expire from caches, regardless of the nameserver from which they were retrieved.

(The case of introducing the first ZSK into the zone is discussed in Section 3.3.5.)

After a delay of Ipub, the key is said to be ready and could be used to sign records. The time at which this event occurs is key N's ready time (Trdy), which is given by:

$$\text{Trdy}(N) = \text{Tpub}(N) + \text{Ipub}$$

Event 3: At some later time, the key starts being used to sign RRs. This point is the activation time (Tact) and after this, key N is said to be active.

$$\text{Tact}(N) \geq \text{Trdy}(N)$$

Event 4: At some point a key must be given to its successor (key $N+1$). As with the introduction of the currently active key into the zone, the successor key will need to be published at least Ipub before it is activated. The publication time of key $N+1$ depends on the activation time of key N :

$$\text{Tpub}(N+1) \leq \text{Tact}(N) + \text{Lzsk} - \text{Ipub}$$

Here, Lzsk is the length of time for which a ZSK will be used (the ZSK lifetime). It should be noted that in the diagrams the actual key lifetime is represented; this may differ slightly from the intended lifetime set by key management policy.

Event 5: While key N is still active, its successor becomes ready. From this time onwards, key $N+1$ could be used to sign the zone.

Event 6: When key N has been in use for an interval equal to the ZSK lifetime, it is retired (i.e. it will never again be used to generate new signatures) and key $N+1$ activated and used to sign the zone. This is the retire time of key N (Tret), and is given by:

$$\text{Tret}(N) = \text{Tact}(N) + \text{Lzsk}$$

It is also the activation time of the successor key $N+1$. Note that operational considerations may cause key N to remain in use for a longer (or shorter) time than the lifetime set by the key management policy.

Event 7: The retired key needs to be retained in the zone whilst any RRSIG records created using this key are still published in the zone or held in caches. (It is possible that a validating resolver could have an old RRSIG record in the cache, but the old DNSKEY RRset has expired when it is asked to provide both to a client. In this case the DNSKEY RRset would need to be looked up again.) This means that once the key is no longer used to sign records, it should be retained in the zone for at least the retire interval (Iret) given by:

$$\text{Iret} = \text{Dsgn} + \text{Dprp} + \text{TTLsig}$$

Dsgn is the delay needed to ensure that all existing RRs have been

re-signed with the new key. Dprp is the propagation delay, required to guarantee that the updated zone information has reached all slave servers, and TTLsig is the maximum TTL of all the RRSIG records in the zone created with the retiring key.

The time at which all RRSIG records created with this key have expired from resolver caches is the dead time (Tdea), given by:

$$Tdea(N) = Tret(N) + Iret$$

... at which point the key is said to be dead.

Event 8: At any time after the key becomes dead, it can be removed from the zone's DNSKEY RRset, which must then be re-signed with the current key-signing key. This time is the removal time (Trem), given by:

$$Trem(N) \geq Tdea(N)$$

... at which time the key is said to be removed.

3.2.2. Double-Signature Method

In this rollover, a new key is introduced and used to sign the zone; the old key and signatures are retained. Once all cached DNSKEY and/or RRSIG information contains copies of the new DNSKEY and RRSIGs created with it, the old DNSKEY and RRSIGs can be removed from the zone.

The timeline for a double-signature rollover is shown below. The diagram follows the convention described in Section 3.2.1

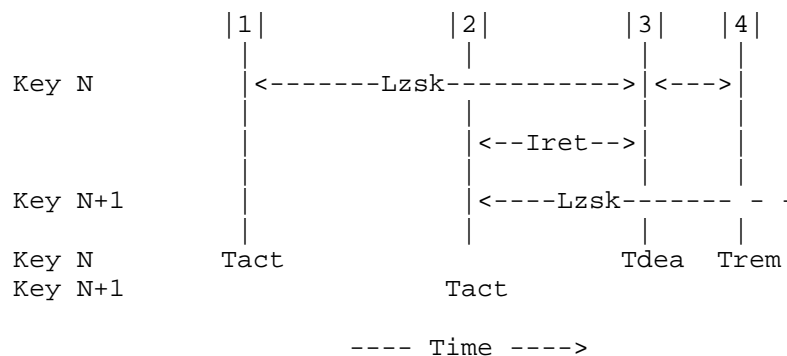


Figure 2: Timeline for a Double-Signature ZSK rollover.

Event 1: Key N is added to the DNSKEY RRset and is then used to sign the zone; existing signatures in the zone are not removed. The key is published and active: this is key N's activation time (Tact), after which the key is said to be active.

Event 2: As the current key (key N) approaches the end of its actual lifetime (Lzsk), the successor key (key N+1) is introduced into the zone and starts being used to sign RRsets: neither the current key nor the signatures created with it are removed. The successor key is now also active.

$$\text{Tact}(N+1) = \text{Tact}(N) + \text{Lzsk} - \text{Iret}$$

Event 3: Before key N can be withdrawn from the zone, all RRsets that need to be signed must have been signed by the successor key (key N+1) and any old RRsets that do not include the new key or new RRSIGs must have expired from caches. Note that the signatures are not replaced - each RRset is signed by both the old and new key.

This takes Iret, the retire interval, given by the expression:

$$\text{Iret} = \text{Dsgn} + \text{Dprp} + \max(\text{TTLkey}, \text{TTLsig})$$

As before, Dsgn is the delay needed to ensure that all existing RRsets have been signed with the new key and Dprp is the propagation delay, required to guarantee that the updated zone information has reached all slave servers. The final term (the maximum of TTLkey and TTLsig) is the period to wait for key and signature data associated with key N to expire from caches. (TTLkey is the TTL of the DNSKEY RRset and TTLsig is the maximum TTL of all the RRSIG records in the zone created with the ZSK. The two may be different: although the TTL of an RRSIG is equal to the TTL of the RRs in the associated RRset [RFC4034], the DNSKEY RRset only needs to be signed with the KSK.)

At the end of this interval, key N is said to be dead. This occurs at the dead time (Tdea) so:

$$\text{Tdea}(N) = \text{Tact}(N+1) + \text{Iret}$$

Event 4: At some later time key N and the signatures generated with it can be removed from the zone. This is the removal time (Trem), given by:

$$\text{Trem}(N) \geq \text{Tdea}(N)$$

3.3. Key-Signing Key Rollover Timelines

The following sections describe the rolling of a KSK. They show the events in the lifetime of a key (referred to as "key N") and cover its replacement by its successor (key N+1). (The case of introducing the first KSK into the zone is discussed in Section 3.3.5.)

3.3.1. Double-KSK Method

In this rollover, The new DNSKEY is added to the zone. After an interval long enough to guarantee that any cached DNSKEY RRsets contain the new DNSKEY, the DS record in the parent zone is changed. After a further interval to allow the old DS record to expire from caches, the old DNSKEY is removed from the zone.

The timeline for a Double-KSK rollover is shown below. The diagram follows the convention described in Section 3.2.1.

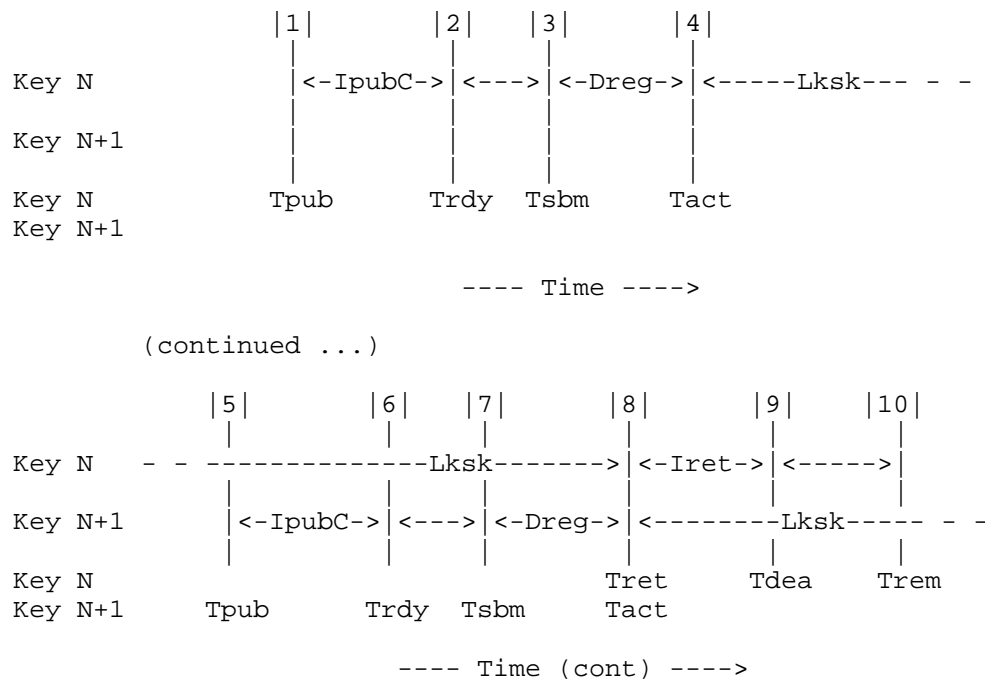


Figure 3: Timeline for a Double-KSK rollover.

Event 1: Key N is introduced into the zone; it is added to the DNSKEY RRset, which is then signed by all currently active KSKs. (So at

this point, the DNSKEY RRset is signed by both key N and its predecessor KSK. If other KSKs were active, it is signed by these as well.) This is the publication time of key N (T_{pub}); after this the key is said to be published.

Event 2: Before it can be used, the key must be published for long enough to guarantee that any validating resolver that has a copy of the DNSKEY RRset in its cache will have a copy of the RRset that includes this key: in other words, that any prior cached information about the DNSKEY RRset has expired.

The interval is the publication interval in the child zone (I_{pubC}) and is given by:

$$I_{pubC} = D_{prpC} + TTL_{key}$$

... where D_{prpC} is the propagation delay for the child zone (the zone containing the KSK being rolled) and TTL_{key} the TTL for the DNSKEY RRset. The time at which this occurs is the key N's ready time, $Trdy$, given by:

$$Trdy(N) = T_{pub}(N) + I_{pubC}$$

Event 3: At some later time, the DS record corresponding to the new KSK is submitted to the parent zone for publication. This time is the submission time, T_{sbm} :

$$T_{sbm}(N) \geq Trdy(N)$$

Event 4: The DS record is published in the parent zone. As this is the point at which all information for authentication - both DNSKEY and DS record - is available in the two zones, in analogy with other rollover methods, this is called the activation time of key N (T_{act}):

$$T_{act}(N) = T_{sbm}(N) + D_{reg}$$

... where D_{reg} is the registration delay, the time taken after the DS record has been submitted to the parent zone manager for it to be placed in the zone. (Parent zones are often managed by different entities, and this term accounts for the organisational overhead of transferring a record. In practice, D_{reg} will not be a fixed time: instead, the end of D_{reg} will be signalled by the appearance of the DS record in the parent zone.)

Event 5: While key N is active, thought needs to be given to its successor (key N+1). At some time before the scheduled end of the KSK lifetime, the successor KSK is published in the zone. (As before, this means that the DNSKEY RRset is signed by all KSKs.)

This time is the publication time of the successor key N+1, given by:

$$T_{\text{pub}}(N+1) \leq T_{\text{act}}(N) + L_{\text{sk}} - D_{\text{reg}} - I_{\text{pubC}}$$

... where L_{sk} is the actual lifetime of the KSK, and D_{reg} the registration delay.

Event 6: After an interval I_{pubC} , key N+1 becomes ready (in that all caches that have a copy of the DNSKEY RRset have a copy of this key). This time is the ready time of the successor key N+1 (T_{rdy}).

Event 7: At the submission time of the successor key N+1, $T_{\text{sbm}}(N+1)$, the DS record corresponding to key N+1 is submitted to the parent zone.

Event 8: The successor DS record is published in the parent zone and the current DS record withdrawn. Key N is said to be retired and the time at which this occurs is $T_{\text{ret}}(N)$, given by:

$$T_{\text{ret}}(N) = T_{\text{sbm}}(N+1) + D_{\text{reg}}$$

Event 9: Key N must remain in the zone until any caches that contain a copy of the DS RRset have a copy containing the new DS record. This interval is the retire interval, given by:

$$I_{\text{ret}} = D_{\text{prpP}} + T_{\text{TLDs}}$$

... where D_{prpP} is the propagation delay in the parent zone and T_{TLDs} the TTL of a DS record in the parent zone.

As the key is no longer used for anything, it is said to be dead. This point is the dead time (T_{dea}), given by:

$$T_{\text{dea}}(N) = T_{\text{ret}}(N) + I_{\text{ret}}$$

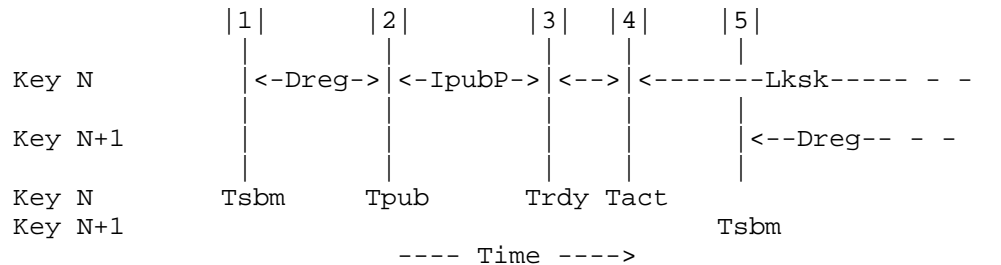
Event 10: At some later time, key N is removed from the zone's DNSKEY RRset (at the remove time T_{rem}); the key is now said to be removed.

$$T_{\text{rem}}(N) \geq T_{\text{dea}}(N)$$

3.3.2. Double-DS Method

In this rollover, the new DS record is published in the parent zone. When any caches that contain the DS RRset contain a copy of the new record, the KSK in the zone is changed. After a further interval for the old DNSKEY RRset to expire from caches, the old DS record is removed from the parent.

The timeline for a Double-DS rollover is shown below. The diagram follows the convention described in Section 3.2.1



(continued ...)

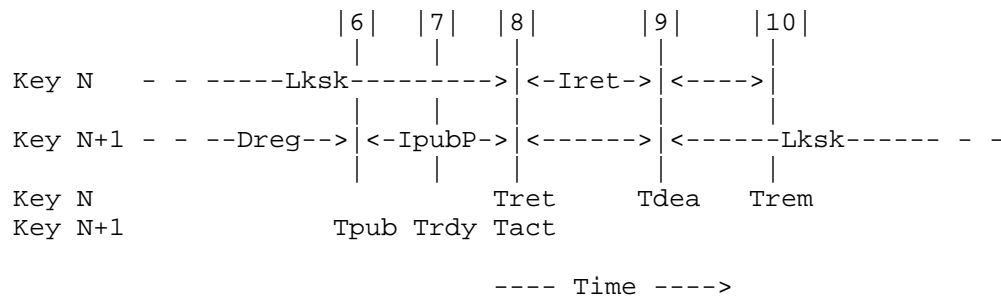


Figure 4: Timeline for a Double-DS KSK rollover.

Event 1: The DS RR is submitted to the parent zone for publication. This time is the submission time, T_{sbm} .

Event 2: After the registration delay, Dreg, the DS record is published in the parent zone. This is the publication time (T_{pub}) of key N, given by:

$$T_{pub}(N) = T_{sbm}(N) + Dreg$$

As before, in practice Dreg will not be a fixed time. Instead, the end of Dreg will be signalled by the appearance of the DS record in the parent zone.

Event 3: At some later time, any cache that has a copy of the DS RRset will have a copy of the DS record for key N. At this point, key N, if introduced into the DNSKEY RRset, could be used to validate the zone. For this reason, this time is known as the ready time, T_{rdy} , and is given by:

$$\text{Trdy}(N) = \text{Tpub}(N) + \text{IpubP}$$

IpubP is the publication interval of the DS record (in the parent zone) and is given by the expression:

$$\text{IpubP} = \text{DprpP} + \text{TTLds}$$

... where DprpP is the propagation delay for the parent zone and TTLds the TTL assigned to DS records in that zone.

Event 4: At some later time, the key rollover takes place and the new key (key N) is introduced into the DNSKEY RRset and used to sign it. This time is key N's activation time (Tact) and at this point key N is said to be active:

$$\text{Tact}(N) \geq \text{Trdy}(N)$$

Event 5: At some point thought must be given to key replacement. The DS record for the successor key must be submitted to the parent zone at a time such that when the current key is withdrawn, any cache that contains the zone's DS records has data about the DS record of the successor key. The time at which this occurs is the submission time of the successor key N+1, given by:

$$\text{Tsbm}(N+1) \leq \text{Tact}(N) + \text{Lksk} - \text{IpubP} - \text{Dreg}$$

... where Lksk is the actual lifetime of key N (which may differ slightly from the lifetime set in the key management policy) and Dreg is the registration delay.

Event 6. After an interval Dreg, the successor DS record is published in the zone.

Event 7: The successor key (key N+1) enters the ready state, i.e. its DS record is now in caches that contain the parent DS RRset.

Event 8: When key N has been active for its lifetime (Lksk), it is replaced in the DNSKEY RRset by key N+1; the RRset is then signed with the new key. At this point, as both the old and new DS records have been in the parent zone long enough to ensure that they are in caches that contain the DS RRset, the zone can be authenticated throughout the rollover. A validating resolver can authenticate either the old or new KSK.

This time is the retire time (Tret) of key N, given by:

$$Tret(N) = Tact(N) + Lksk$$

This is also the activation time of the successor key N+1.

Event 9: At some later time, all copies of the old DNSKEY RRset have expired from caches and the old DS record is no longer needed. In analogy with other rollover methods, this is called the dead time, Tdea, and is given by:

$$Tdea(N) = Tret(N) + Iret$$

... where Iret is the retire interval of the key, given by:

$$Iret = DprpC + TTLkey$$

As before, this term includes DprpC, the time taken to propagate the RRset change through the master-slave hierarchy of the child zone and TTLkey, the time taken for the DNSKEY RRset to expire from caches.

Event 10: At some later time, the DS record is removed from the parent zone. In analogy with other rollover methods, this is the removal time (Trem), given by:

$$Trem(N) \geq Tdea(N)$$

3.3.3. Double-RRset Method

In the Double-RRset rollover, the new DNSKEY and DS records are published simultaneously in the appropriate zones. Once enough time has elapsed for the old DNSKEY and DS RRsets to expire from caches, the old DNSKEY and DS records are removed from their respective zones.

The timeline for this rollover is shown below. The diagram follows the convention described in Section 3.2.1

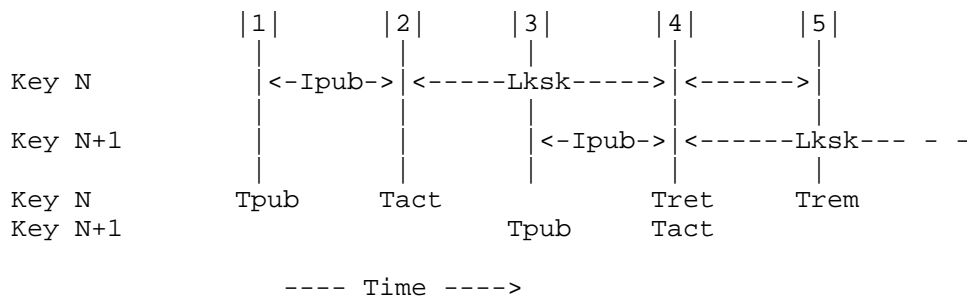


Figure 5: Timeline for a Double-RRset KSK rollover.

Event 1: The key is added to and used for signing the DNSKEY RRset and is thereby published in the zone. At the same time the corresponding DS record is submitted to the parent zone for publication. This time is the publish time for key N (T_{pub}) and the key is said to be published.

Event 2: At some later time, the DS record is published in the parent zone and at some time after that, the updated information has reached all caches: any cache that holds a DNSKEY RRset from the child zone will have a copy that includes the new KSK, and any cache that has a copy of the parent DS RRset will have a copy that includes the new DS record.

The time at which this occurs is called the activation time of key N (T_{act}), given by:

$$T_{act}(N) = T_{pub}(N) + I_{pub}$$

... where I_{pub} is the composite publication interval for the DNSKEY and DS records, given by:

$$I_{pub} = \max(I_{pubP}, I_{pubC}),$$

I_{pubP} being the publication interval of the DS record in the parent zone and I_{pubC} the publication interval of the DNSKEY in the child zone. The parent zone's publication interval is given by:

$$I_{pubP} = D_{reg} + D_{prpP} + TTL_{ds}$$

where D_{reg} is the registration delay, the time taken for the DS record to be published in the parent zone. D_{prpP} is the parent zone's propagation delay and TTL_{ds} is the TTL of the DS record in that zone.

The child zone's publication interval is given by a similar equation:

$$\text{IpubC} = \text{DprpC} + \text{TTLkey}$$

... where DprpC is the propagation delay in the child zone and TTLkey the TTL of a DNSKEY record.

Event 3: At some point we need to give thought to key replacement. The successor key (key N+1) must be introduced into the zone (and its DS record submitted to the parent) at a time such that it becomes active when the current key has been active for its actual lifetime, Lksk. This is the publication time (Tpub) of the successor key, and is given by:

$$\text{Tpub}(N+1) \leq \text{Tact}(N) + \text{Lksk} - \text{Ipub}$$

... where Lksk is the actual lifetime of the KSK and Ipub is as defined above.

Event 4: Key N+1's DNSKEY and DS records are now in caches that contain the child zone DNSKEY and/or the parent zone DS RR, and so the zone can be validated with the new key. This is the activation time (Tact) of the successor key N+1 and by analogy with other rollover methods, it is also the dead time of key N:

$$\text{Tdea}(N) = \text{Tact}(N) + \text{Lksk}$$

Event 5: At some later time, the key N's DS and DNSKEY records are removed from their respective zones. In analogy with other rollover methods, this is the removal time (Trem), given by:

$$\text{Trem}(N) \geq \text{Tdea}(N)$$

3.3.4. Interaction with Configured Trust Anchors

Although the preceding sections have been concerned with rolling KSKs where the trust anchor is a DS record in the parent zone, zone managers may want to take account of the possibility that some validating resolvers may have configured trust anchors directly.

Rolling a configured trust anchor is dealt with in [RFC5011]. It requires introducing the KSK to be used as the trust anchor into the zone for a period of time before use, and retaining it (with the "revoke" bit set) for some time after use.

3.3.4.1. Addition of KSK

When the new key is introduced, the expression for the publication interval of the DNSKEY(IpubC) in the Double-KSK and Double-RRset methods is modified to:

$$\text{IpubC} \geq \text{DprpC} + \max(\text{Itrp}, \text{TTLkey})$$

... where the right hand side of the expression now includes the "trust point" interval. This term is the interval required to guarantee that a resolver configured for the automatic update of keys from a particular trust point will see at least two validated DNSKEY RRsets containing the new key (a requirement from [RFC5011], section 2.4.1). It is defined by the expression:

$$\text{Itrp} \geq (2 * \text{queryInterval}) + (n * \text{retryTime})$$

... where queryInterval and retryTime are as defined in section 2.3 of [RFC5011]. "n" is the total number of retries needed by the resolver during the two attempts to get the DNSKEY RRset.

The first term of the expression $(2 * \text{queryInterval})$ represents the time to obtain two validated DNSKEY RRsets. The second term $(n * \text{retryTime})$ is a safety margin, with the value of "n" reflecting the degree of confidence in the communication between a resolver and the trust point.

In the Double-DS method, instead of swapping the KSK RRs in a single step, there must now be a period of overlap. In other words, the new KSK must be introduced into the zone at least:

$$\text{DprpC} + \max(\text{Itrp}, \text{TTLkey})$$

... before the switch is made.

3.3.4.2. Removal of KSK

The timeline for the removal of the key in all methods is modified by introducing a new state, "revoked". When the key reaches its dead time, instead of being declared "dead", it is revoked; the "revoke" bit is set in the published DNSKEY RR, and the DNSKEY RRset re-signed with the current and revoked keys. The key is maintained in this state for the "revoke" interval, Irev, given by:

$$\text{Irev} \geq 30 \text{ days}$$

... 30 days being the [RFC5011] remove hold-down time. After this time, the key is dead and can be removed from the zone.

3.3.5. Introduction of First Keys

There are no timing considerations associated with the introduction of the first keys into a zone other than they must be introduced and the zone validly signed before a chain of trust to the zone is

created.

This is important: in the case of a secure parent, it means ensuring that the DS record is not published in the parent zone until there is no possibility that a validating resolver can obtain the record yet is not able to obtain the corresponding DNSKEY. In the case of an insecure parent, i.e. the initial creation of a new security apex, it is not possible to guarantee this. It is up to the operator of the validating resolver to wait for the new KSK to appear at all servers for the zone before configuring the trust anchor.

4. Standby Keys

Although keys will usually be rolled according to some regular schedule, there may be occasions when an emergency rollover is required, e.g., if the active key is suspected of being compromised. The aim of the emergency rollover is to allow the zone to be re-signed with a new key as soon as possible. As a key must be in the ready state to sign the zone, having at least one additional key (a standby key) in this state at all times will minimise delay.

In the case of a ZSK, a standby key only really makes sense with the Pre-Publication method. A permanent standby DNSKEY RR should be included in the zone or successor keys could be introduced as soon as possible after a key becomes active. Either way results in one or more additional ZSKs in the DNSKEY RRset that can immediately be used to sign the zone if the current key is compromised.

(Although in theory the mechanism could be used with both the Double-Signature and Double-RRSIG methods, it would require pre-publication of the signatures. Essentially, the standby key would be permanently active, as it would have to be periodically used to renew signatures. Zones would also permanently require two sets of signatures.)

It is also possible to have a standby KSK. The Double-KSK method requires that the standby KSK be included in the DNSKEY RRset; rolling the key then requires just the introduction of the DS record in the parent. Note that the standby KSK should also be used to sign the DNSKEY RRset. As the RRset and its signatures travel together, merely adding the KSK without using it to sign the DNSKEY RRset does not provide the desired time saving: for a KSK to be used in a rollover the DNSKEY RRset must be signed with it, and this would introduce a delay while the old RRset (not signed with the new key) expires from caches.

The idea of a standby KSK in the Double-RRset rollover method effectively means having two active keys (as the standby KSK and

associated DS record would both be published at the same time in their respective zones).

Finally, in the Double-DS method of rolling a KSK, it is not a standby key that is present, it is a standby DS record in the parent zone.

Whatever algorithm is used, the standby item of data can be included in the zone on a permanent basis, or be a successor introduced as early as possible.

5. Algorithm Considerations

The preceding sections have implicitly assumed that all keys and signatures are created using a single algorithm. However, [RFC4035] (section 2.2) requires that there be an RRSIG for each RRset using at least one DNSKEY of each algorithm in the zone apex DNSKEY RRset.

Except in the case of an algorithm rollover - where the algorithms used to create the signatures are being changed - there is no relationship between the keys of different algorithms. This means that they can be rolled independently of one another. In other words, the key rollover logic described above should be run separately for each algorithm; the union of the results is included in the zone, which is signed using the active key for each algorithm.

6. Summary

For ZSKs, "Pre-Publication" is generally considered to be the preferred way of rolling keys. As shown in this document, the time taken to roll is wholly dependent on parameters under the control of the zone manager.

In contrast, "Double-RRset" is the most efficient method for KSK rollover due to the ability to have new DS records and DNSKEY RRsets propagate in parallel. The time taken to roll KSKs may depend on factors related to the parent zone if the parent is signed. For zones that intend to comply with the recommendations of [RFC5011], in virtually all cases the rollover time will be determined by the RFC5011 "add hold-down" and "remove hold-down" times. It should be emphasized that this delay is a policy choice and not a function of timing values and that it also requires changes to the rollover process due to the need to manage revocation of trust anchors.

Finally, the treatment of emergency key rollover is significantly simplified by the introduction of standby keys as standard practice

during all types of rollovers.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

This document does not introduce any new security issues beyond those already discussed in [RFC4033], [RFC4034], [RFC4035] and [RFC5011].

9. Acknowledgements

The authors gratefully acknowledge help and contributions from Roy Arends and Wouter Wijngaards.

10. Normative References

- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.
- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.
- [RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", STD 74, RFC 5011, September 2007.
- [RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, December 2012.

Appendix A. List of Symbols

The document defines a number of symbols, all of which are listed here. All are of the form:

All symbols used in the text are of the form:

<TYPE><id><ZONE>

where:

<TYPE> is an upper-case character indicating what type the symbol is. Defined types are:

D	delay: interval that is a feature of the process
I	interval between two events
L	lifetime: interval set by the zone manager
T	a point in time
TTL	TTL of a record

I, T and TTL are self-explanatory. Like I, both D and L are time periods, but whereas I values are intervals between two events (even if the events are defined in terms of the interval, e.g., the dead time occurs "retire interval" after the retire time), D and L are fixed intervals: a "D" interval (delay) is a feature of the process, probably outside control of the zone manager, whereas an "L" interval (lifetime) is chosen by the zone manager and is a feature of policy.

<id> is lower-case and defines what object or event the variable is related to, e.g.,

act	activation
pub	publication
ret	retire

<ZONE> is an optional capital letter that distinguishes between the same variable applied to different zones and is one of:

C	child
P	parent

Within the rollover descriptions, times may be suffixed by a number in brackets indicating the instance of the key to which they apply, e.g. Tact(N) is the activation time of key N, Tpub(N+1) the publication time of key N+1 etc.

The list of variables used in the text given below.

Dprp	Propagation delay. The amount of time for a change made at a master nameserver to propagate to all the slave nameservers.
DprpC	Propagation delay in the child zone.
DprpP	Propagation delay in the parent zone.
Dreg	Registration delay: the time taken for a DS record submitted to a parent zone to appear in it. As a parent zone is often managed by a different organisation to that managing the child zone, the delays associated with passing data between organisations is captured by this term.
Dsgn	Signing delay. After the introduction of a new ZSK, the amount of time taken for all the RRs in the zone to be signed with it.
Ipub	Publication interval. The amount of time that must elapse after the publication of a DNSKEY and/or its associated data before it can be assumed that any resolvers that have the relevant RRset cached have a copy of the new information.
IpubC	Publication interval in the child zone.
IpubP	Publication interval in the parent zone.
Iret	Retire interval. The amount of time that must elapse after a DNSKEY or associated data enters the retire state for any dependent information (e.g. RRSIG for a ZSK) to be purged from validating resolver caches.
Irev	Revoke interval. The amount of time that a KSK must remain published with the revoke bit set to satisfy [RFC5011] considerations.
Itrp	Trust-point interval. The amount of time that a trust anchor must be published for to guarantee that a resolver configured for an automatic update of keys will see the new key at least twice.
Lksk	Lifetime of a key-signing key. This is the actual amount of time for which this particular KSK is regarded as the active KSK. Depending on when the key is rolled-over, the actual lifetime may be longer or shorter than the intended key lifetime indicated by management policy.

Lzsk	Lifetime of a zone-signing key. This is the actual amount of time for which the ZSK is used to sign the zone. Depending on when the key is rolled-over, the actual lifetime may be longer or shorter than the intended key lifetime indicated by management policy.
Tact	Activation time. The time at which the key is regarded as the principal key for the zone.
Tdea	Dead time. The time at which any information held in validating resolver caches is guaranteed to contain information related to the successor key. At this point, the current key and its associated information are not longed required for validation purposes.
Tpub	Publication time. The time that the key or associated data appears in the zone for the first time.
Trem	Removal time. The time at which the key and its associated information starts being removed from their respective zones.
Tret	Retire time. The time at which successor information starts being used.
Trdy	Ready time. The time at which it can be guaranteed that validating resolvers that have information about the key and/or associated data cached have a copy of the new information.
Tsbm	Submission time. The time at which the DS record of a KSK is submitted to the parent zone.
TTLds	Time to live of a DS record.
TTLkey	Time to live of a DNSKEY record. (By implication, this is also the time to live of the signatures on the DNSKEY RRset.)
TTLsig	The maximum time to live of all the RRSIG records in the zone that were created with the ZSK.

Appendix B. Change History (To be removed on publication)

- o draft-ietf-dnsop-dnssec-key-timing-06
 - * Clarifications to various text, as identified in WGLC.
 - * Moved "Limitation of Scope" section to be a subsection of

section 1.

- o draft-ietf-dnsop-dnssec-key-timing-05
 - * Some more renamings of "Double-Signature" KSK rollover to "Double-KSK".
 - * Remove Tgen from diagrams.
 - * Review by Richard Lamb.
 - * Updated KSK rollover summary text.
 - * Updated variable descriptions in the appendix.
- o draft-ietf-dnsop-dnssec-key-timing-04
 - * Renamed to "DNSSEC Key Rollover Timing Considerations" to emphasise that this draft concerns rollover timings.
 - * Updated 4641bis reference to RFC 6781.
 - * Add introductory paragraph to each rollover description summarising its essential features.
 - * Remove detailed description of double-RRSIG ZSK rollover. It is extremely unlikely to be used in any practical situation.
 - * "Double-Signature" KSK rollover renamed to "Double-KSK" to avoid confusion with the ZSK rollover of the same name.
 - * Removed section 2.3 (rollover summary) which just listed the order in which records are published.
 - * Matthijs Mekking added as co-author.
 - * Changed Lzsk and Kzsk definitions: actual lifetime instead of intended lifetime.
 - * Update diagrams and text to better reflect key states and key lifetimes.
- o draft-ietf-dnsop-dnssec-key-timing-03
 - * Clarifications of and corrections to wording (Marc Lampo, Alfred Hoenes).
 - * Updated timings related to trust anchor interaction (Matthijs Mekking).
 - * Updated RFC 4641 reference to 4641bis (Alfred Hoenes).
 - * Moved change history to end of document (Alfred Hoenes).
- o draft-ietf-dnsop-dnssec-key-timing-02
 - * Significant re-wording of some sections.
 - * Removal of events noting change of state of predecessor key from ZSK Double-RRSIG and Double-Signature methods.
 - * Change order of bullet points (and some wording) in section 1.1.
 - * Remove discussion of advantages and disadvantages of key roll methods from section 2: draft is informative and does not give recommendations.
 - * Removal of discussion of upper limit to retire time relationship to signature lifetime.
 - * Remove timing details of first key in the zone and move discussion of first signing of a zone to later in the document.

(Matthijs Mekking)

* Removal of redundant symbols from Appendix A.

- o draft-ietf-dnsop-dnssec-key-timing-01
 - * Added section on limitation of scope.
- o draft-ietf-dnsop-dnssec-key-timing-00
 - * Change to author contact details.
- o draft-morris-dnsop-dnssec-key-timing-02
 - * General restructuring.
 - * Added descriptions of more rollovers (IETF-76 meeting).
 - * Improved description of key states and removed diagram.
 - * Provided simpler description of standby keys.
 - * Added section concerning first key in a zone.
 - * Moved [RFC5011] to a separate section.
 - * Various nits fixed (Alfred Hoenes, Jeremy Reed, Scott Rose, Sion Lloyd, Tony Finch).
- o draft-morris-dnsop-dnssec-key-timing-01
 - * Use latest boilerplate for IPR text.
 - * List different ways to roll a KSK (acknowledgements to Mark Andrews).
 - * Restructure to concentrate on key timing, not management procedures.
 - * Change symbol notation (Diane Davidowicz and others).
 - * Added key state transition diagram (Diane Davidowicz).
 - * Corrected spelling, formatting, grammatical and style errors (Diane Davidowicz, Alfred Hoenes and Jinmei Tatuya).
 - * Added note that in the case of multiple algorithms, the signatures and rollovers for each algorithm can be considered as more or less independent (Alfred Hoenes).
 - * Take account of the fact that signing a zone is not atomic (Chris Thompson).
 - * Add section contrasting pre-publication rollover with double signature rollover (Matthijs Mekking).
 - * Retained distinction between first and subsequent keys in definition of initial publication interval (Matthijs Mekking).
- o draft-morris-dnsop-dnssec-key-timing-00
 - Initial draft.

Authors' Addresses

Stephen Morris
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
USA

Email: stephen@isc.org
URI: <http://www.isc.org>

Johan Ihren
Netnod
Franzengatan 5
Stockholm, SE-112 51
Sweden

Email: johani@netnod.se
URI: <http://www.netnod.se>

John Dickinson
Sinodun Internet Technologies Ltd
Magdalen Centre
Oxford Science Park
Robert Robertson Avenue
Oxford, Oxfordshire OX4 4GA
UK

Email: jad@sinodun.com
URI: <http://www.sinodun.com>

W. (Matthijs) Mekking
NLnet Labs
Science Park 400
Amsterdam 1098 XH
The Netherlands

Email: matthijs@nlnetlabs.nl
URI: <http://www.nlnetlabs.nl>

DNSOP
Internet-Draft
Intended status: Best Current Practice
Expires: February 12, 2017

W. Hardaker
Parsons
O. Gudmundsson
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

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 12, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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. Introduction	3
1.1. Notation	3
1.2. Background	3
1.3. Implementation experiences	4
1.3.1. Test Zone Implementation	4
2. Goals	5
3. Detecting DNSSEC Non-Compliance	5
3.1. Determining DNSSEC support in recursive resolvers	6
3.1.1. Supports UDP answers	6
3.1.2. Supports TCP answers	6
3.1.3. Supports EDNS0	7
3.1.4. Supports the DO bit	7
3.1.5. Supports the AD bit DNSKEY algorithm 5 and 8	7
3.1.6. Returns RRSig for signed answer	8
3.1.7. Supports querying for DNSKEY records	8
3.1.8. Supports querying for DS records	8
3.1.9. Supports negative answers with NSEC records	9
3.1.10. Supports negative answers with NSEC3 records	9
3.1.11. Supports queries where DNAME records lead to an answer	10
3.1.12. Permissive DNSSEC	10
3.1.13. Supports Unknown RRtypes	10
3.2. Direct Network Queries	10
3.2.1. Support for Remote UDP Over Port 53	11
3.2.2. Support for Remote UDP With Fragmentation	11
3.2.3. Support for Outbound TCP Over Port 53	11
3.3. Support for DNSKEY and DS combinations	12
4. Aggregating The Results	12
4.1. Resolver capability description	12
5. Roadblock Avoidance	13
5.1. Partial Resolver Usage	16
5.1.1. Known Insecure Lookups	16
5.1.2. Partial NSEC/NSEC3 Support	16
6. Start-Up and Network Connectivity Issues	16
6.1. What To Do	17
7. Quick Test	17
7.1. Test negative answers Algorithm 5	18
7.2. Test Algorithm 8	18
7.3. Test Algorithm 13	18
7.4. Fails when DNSSEC does not validate	18
8. Security Considerations	18

9. IANA Considerations	18
10. Acknowledgments	18
11. Normative References	19
Authors' Addresses	19

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 is 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 is 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 has 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

dnsop
Internet-Draft
Intended status: Experimental
Expires: August 21, 2016

P. Wouters
Red Hat
February 18, 2016

Chain Query requests in DNS
draft-ietf-dnsop-edns-chain-query-07

Abstract

This document defines an EDNS0 extension that can be used by a security-aware validating resolver configured to use a Forwarder to send a single query, requesting a complete validation path along with the regular query answer. The reduction in queries potentially lowers the latency and reduces the need to send multiple queries at once. This extension mandates the use of source IP verified transport such as TCP or UDP with EDNS-COOKIE so it cannot be abused in amplification attacks.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 21, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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. Introduction	2
1.1. Requirements Notation	3
2. Terminology	3
3. Overview	4
4. Option Format	5
5. Protocol Description	5
5.1. Discovery of Support	5
5.2. Generate a Query	6
5.3. Send the Option	6
5.4. Generate a Response	6
6. Protocol Considerations	7
6.1. DNSSEC Considerations	7
6.2. NS record Considerations	8
6.3. Session Management	8
6.4. Negative Trust Anchors	8
6.5. Anycast Considerations	9
7. Implementation Status	9
8. Security Considerations	10
8.1. Additional work and bandwidth	10
8.2. Amplification Attacks	10
8.3. Privacy Considerations	10
9. Examples	10
9.1. Simple Query for example.com	10
9.2. Out-of-path Query for example.com	12
9.3. Non-existent data	13
10. IANA Considerations	14
10.1. EDNS0 option code for CHAIN	14
11. Acknowledgements	14
12. Normative References	14
Author's Address	16

1. Introduction

Traditionally, a DNS client operates in stub-mode. For each DNS question the DNS client needs to resolve, it sends a single query to an upstream Recursive Resolver to obtain a single DNS answer. When DNSSEC [RFC4033] is deployed on such DNS clients, validation requires that the client obtains all the intermediate information from the DNS root down to the queried-for hostname so it can perform DNSSEC validation on the complete chain of trust.

Currently, applications send out many UDP requests concurrently. This requires more resources on the DNS client with respect to state (cpu, memory, battery) and bandwidth. There is also no guarantee that the initial set of UDP questions will result in all the records required for DNSSEC validation. More round trips could be required depending on the resulting DNS answers. This especially affects high-latency links.

This document specifies an EDNS0 extension that allows a validating Resolver running as a Forwarder to open a TCP connection to another Resolver and request a DNS chain answer using one DNS query/answer pair. This reduces the number of round trips to two. If combined with long lived TCP or [TCP-KEEPALIVE] there is only one round trip. While the upstream Resolver still needs to perform all the individual queries required for the complete answer, it usually has a much bigger cache and does not experience significant slowdown from last-mile latency.

This EDNS0 extension allows the Forwarder to indicate which part of the DNS hierarchy it already contains in its cache. This reduces the amount of data required to be transferred and reduces the work the upstream Recursive Resolver has to perform.

This EDNS0 extension is only intended to be sent by Forwarders to Recursive Resolvers. It MUST be ignored by Authoritative Servers.

1.1. Requirements Notation

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

2. Terminology

The DNS terminology used in this document is that of [RFC7719]. Additionally, the following terms are used:

Recursive Resolver: A nameserver that is responsible for resolving domain names for clients by following the domain's delegation chain, starting at the root. Recursive Resolvers frequently use caches to be able to respond to client queries quickly. Described in [RFC1035] chapter 7.

Validating Resolver: A recursive nameserver that also performs DNSSEC [RFC4033] validation. Also known as "security-aware resolver".

3. Overview

When DNSSEC is deployed on a host, it can no longer delegate all DNS work to the upstream Recursive Resolver. Obtaining just the DNS answer itself is not enough to validate that answer using DNSSEC. For DNSSEC validation, the DNS client requires a locally running validating Resolver so it can confirm DNSSEC validation of all intermediary DNS answers. It can configure itself as a Forwarder if it obtains the IP addresses of one or more Recursive Resolvers that are available, or as a stand-alone Recursive Resolver if no functional Recursive Resolvers were obtained. Generating the required queries for validation adds a significant delay in answering the DNS question of the locally running application. The application must wait while the Resolver validates all intermediate answers. Each round-trip adds to the total time waiting on DNS resolution with validation to complete. This makes DNSSEC resolving impractical for devices on networks with a high latency.

This document defines the CHAIN option that allows the Resolver to request all intermediate DNS data it requires to resolve and validate a particular DNS answer in a single round-trip. The Resolver could be part of the application or a Recursive Resolver running on the host.

Servers answering with CHAIN data should ensure that the transport is TCP or source IP address verified UDP. See Section 8. This avoids abuse in DNS amplification attacks.

Applications that support CHAIN internally can perform validation without requiring the host to run a Recursive Resolver. This is particularly useful for virtual servers in a cloud or container based deployment where it is undesirable to run a Recursive Resolver per virtual machine.

The format of this option is described in Section 4.

As described in Section 5.4, a Recursive Resolver could use this EDNS0 option to include additional data required by the Resolver in the Authority Section of the DNS answer packet when using a source IP verified transport. The Answer Section remains unchanged from a traditional DNS answer and contains the answer and related DNSSEC entries.

An empty CHAIN EDNS0 option MAY be sent over any transport as a discovery method. A DNS server receiving such an empty CHAIN option SHOULD add an empty CHAIN option in its answer to indicate that it supports CHAIN for source IP address verified transports.

The mechanisms provided by CHAIN raise various security related concerns, related to the additional work, bandwidth, amplification attacks as well as privacy issues with the cache. These concerns are described in Section 8.

4. Option Format

This draft uses an EDNS0 [RFC6891] option to include client IP information in DNS messages. The option is structured as follows:

```

      1                               2                               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
!           OPTION-CODE           !           OPTION-LENGTH           !
+-----+-----+-----+-----+-----+-----+-----+-----+
~                               Closest Trust Point (FQDN)                               ~
+-----+-----+-----+-----+-----+-----+-----+-----+

```

- o OPTION-CODE, 2 octets, for CHAIN is 13.
- o OPTION-LENGTH, 2 octets, contains the length of the payload (everything after Option-length) in octets.
- o Closest Trust Point, a variable length Fully Qualified Domain Name ("FQDN") in DNS wire format of the requested start point of the chain. This entry is the 'lowest' known entry in the DNS chain known by the recursive server seeking a CHAIN answer for which it has a validated DS and DNSKEY record. The end point of the chain is obtained from the DNS Query Section itself. No DNS name compression is allowed for this value.

5. Protocol Description

5.1. Discovery of Support

A Forwarder may include a zero-length CHAIN option in a regular query over any transport to discover the DNS server capability for CHAIN. Recursive Resolvers that support and are willing to accept CHAIN queries over source IP verified transport respond to a zero-length CHAIN received by including a zero-length CHAIN option in the answer. If not already using a source IP verified transport, the Forwarder MAY then switch to a source IP verified transport and start sending queries with the CHAIN option to request a CHAIN response from the Recursive Resolver. Examples of source IP verification are the 3-way TCP handshake and UDP with [EDNS-COOKIE].

5.2. Generate a Query

In this option value, the Forwarder sets the Closest Trust Point in the chain - furthest from the root - that it already has a DNSSEC validated (secure or not) answer for in its cache. The upstream Recursive Resolver does not need to include any part of the chain from the root down to this option's FQDN. A complete example is described in Section 9.1.

The CHAIN option should generally be sent by system Forwarders and Resolvers within an application that also perform DNSSEC validation.

5.3. Send the Option

When CHAIN is available, the downstream Recursive Resolver can adjust its query strategy based on the desired queries and its cache contents.

A Forwarder can request the CHAIN option with every outgoing DNS query. However, it is RECOMMENDED that Forwarders remember which upstream Recursive Resolvers did not return the option (and additional data) with their response. The Forwarder SHOULD fallback to regular DNS for subsequent queries to those Recursive Resolvers. It MAY switch to another Recursive Resolver that does support the CHAIN option or try again later to see if the server has become less loaded and is now willing to answer with Query Chains. A fallback strategy similar to that described in [RFC6891] section 6.2.2 SHOULD be employed to avoid persistent interference due to non-clean paths.

5.4. Generate a Response

When a query containing a non-zero CHAIN option is received from a Forwarder, the upstream Recursive Resolver supporting CHAIN MAY respond by confirming that it is returning a CHAIN. To do so, it MUST set the CHAIN option to the lowest Trust Point sent as part of the chain, with its corresponding OPTION-LENGTH. It extends the Authority Section in the DNS answer packet with the DNS RRsets required for validating the answer. The DNS RRsets added start with the first chain element below the received Closest Trust Point up to and including the NS and DS RRsets that represent the zone cut (authoritative servers) of the QNAME. The added RRsets MAY be added in matching hierarchical order but a DNS client MUST NOT depend on the order of the added RRsets for validation. The actual DNS answer to the question in the Query Section is placed in the DNS Answer Section identical to the traditional DNS answer. All required DNSSEC related records must be added to their appropriate sections. This includes records required for proof of non-existence of regular and/or wildcard records, such as NSEC or NSEC3 records.

Recursive Resolvers that have not implemented or enabled support for the CHAIN option, or are otherwise unwilling to perform the additional work for a Chain Query due to work load, may safely ignore the option in the incoming queries. Such a server MUST NOT include an CHAIN option when sending DNS answer replies back, thus indicating it is not able or willing to support Chain Queries at this time.

Requests with wrongly formatted options (i.e. bogus FQDN) MUST be rejected and a FORMERR response must be returned to the sender, as described by [RFC6891].

Requests resulting in chains that the receiving resolver is unwilling to serve can be rejected by answering the query as a regular DNS reply but with an empty CHAIN payload. Replying with an empty CHAIN can be used for chains that would be too big or chains that would reveal too much information considered private.

At any time, a Recursive Resolver that has determined that it is running low on resources can refuse CHAIN queries by replying with a regular DNS reply with an empty CHAIN payload.

If a CHAIN answer would be bigger than the Recursive Resolver is willing to serve, it SHOULD send a partial chain starting with the data closest to the top of the chain. The client MAY re-send the query with an updated Closest Trust Point until it has received the full chain. The CHAIN response will contain the lowest Closest Trust Point that was included in the CHAIN answer.

If the DNS request results in a CNAME or DNAME for the Answer Section, the Recursive Resolver MUST return these records in the Answer Section similar to regular DNS processing. The CNAME or DNAME target MAY be placed in the Additional Section only if all supporting records for DNSSEC validation of the CNAME or DNAME target are also added to the Authority Section.

The response from a Recursive Resolver to a Resolver MUST NOT contain the CHAIN option if none was present in the Resolver's original request.

A DNS query that contains the CHAIN option MUST also have the DNSSEC OK ("OK") bit set. If this bit is not set, or if the Checking Disabled ("CD") bit is set, the CHAIN option received MUST be ignored.

6. Protocol Considerations

6.1. DNSSEC Considerations

The presence or absence of an OPT resource record containing an CHAIN option in a DNS query does not change the usage of those resource records and mechanisms used to provide data origin authentication and data integrity to the DNS, as described in [RFC4033], [RFC4034] and [RFC4035].

6.2. NS record Considerations

CHAIN responses SHOULD include the NS RRset from the zone itself including the RRSIG records required for validation. It MUST NOT include the NS RRset from parent zone, as this RRset is not signed. If the size of the answer is an important factor, the NS RRset MAY be omitted.

When a DNSSEC chain is supplied via CHAIN, the Forwarder is no longer required to use the NS RRset, as it can construct the validation path via the DNSKEY and DS RRsets without using the NS RRset. However, the Forwarder might be forced to switch from Forwarder mode to Recursive Resolver mode due to a network topology change. In Recursive Resolver mode, the NS RRsets are needed to find and query Authoritative Servers directly. It is RECOMMENDED that the DNS Forwarder populate its cache with this information to avoid requiring future queries to obtain any missing NS records. Therefore, CHAIN responses MUST include the NS RRset from the child zone, including the RRSIG records required for validation.

6.3. Session Management

The use of [TCP-KEEPALIVE] on DNS TCP sessions is RECOMMENDED, and thus TCP sessions should not immediately be closed after the DNS answer to the first query is received.

Both DNS clients and servers are subject to resource constraints which will limit the extent to which Chain Queries can be executed. Effective limits for the number of active sessions that can be maintained on individual clients and servers should be established, either as configuration options or by interrogation of process limits imposed by the operating system.

In the event that there is greater demand for Chain Queries than can be accommodated, DNS servers may stop advertising the CHAIN option in successive DNS messages. This allows, for example, clients with other candidate servers to query to establish new sessions with different servers in expectation that those servers might still allow Chain Queries.

6.4. Negative Trust Anchors

If a CHAIN answer would intersect with a Negative Trust Anchor [RFC7646], a partial CHAIN up to the node above the Negative Trust Anchor should be returned.

6.5. Anycast Considerations

Recursive Resolvers of various types are commonly deployed using anycast [RFC4786].

Successive DNS transactions between a client and server using UDP transport may involve responses generated by different anycast nodes, and the use of anycast in the implementation of a DNS server is effectively undetectable by the client. The CHAIN option SHOULD NOT be included in responses using UDP transport from servers provisioned using anycast unless all anycast server nodes are capable of processing the CHAIN option.

Since DNS queries using CHAIN may result in longer TCP sessions, network topology changes may disrupt them more frequently. Anycast servers MAY make use of TCP multipath [RFC6824] to anchor the server side of the TCP connection to an unambiguously-unicast address in order to avoid disruption due to topology changes.

7. Implementation Status

[RFC Editor Note: Please remove this entire section prior to publication as an RFC.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

[While there is some interest, no work has started yet]

8. Security Considerations

8.1. Additional work and bandwidth

Producing CHAIN answers incurs additional load and bandwidth on the Recursive Resolver. At any time, a Recursive Resolver may decide to no longer answer with CHAIN answers and fall back to traditional DNS answers.

8.2. Amplification Attacks

Chain Queries can potentially send very large DNS answers. Attackers could abuse this using spoofed source IP addresses to inflict large Distributed Denial of Service attacks using query-chains as an amplification vector in their attack. While TCP is not vulnerable for this type of abuse, the UDP protocol is vulnerable to this.

A Recursive Resolver MUST NOT return CHAIN answers to clients over UDP without source IP address verification. An example of UDP based source IP address verification is [EDNS-COOKIE]. A Recursive Resolver refusing a CHAIN option MUST respond with a zero-length CHAIN option to indicate support for CHAIN queries when a proper transport is used. It MUST NOT send an RCODE of REFUSED.

8.3. Privacy Considerations

A client producing CHAIN queries reveals a little more information about its cache contents than regular DNS clients. This could be used the fingerprint a client across network reconnections. If DNS privacy is a concern, a CHAIN query client MAY try to use a DNS transport that provides privacy, such as [DNS-over-TLS] or a trusted DNS server that is contacted through a VPN connection such as IPsec.

9. Examples

9.1. Simple Query for example.com

- o A web browser on a client machine asks the Forwarder running on localhost to resolve the A record of "www.example.com." by sending a regular DNS UDP query on port 53 to 127.0.0.1.
- o The Resolver on the client machine checks its cache, and notices it already has a DNSSEC validated entry of "com." in its cache. This includes the DNSKEY RRset with its RRSIG records. In other words, according to its cache, ".com" is DNSSEC validated as "secure" and can be used to continue a DNSSEC validated chain.

- o The Resolver on the client opens a TCP connection to its upstream Recursive Resolver on port 53. It adds the CHAIN option as follows:
 - * Option-code, set to 13
 - * Option-length, set to 5
 - * Closest Trust Point set to "com." (0x03 0x63 0x6f 0x6d 0x00)
- o The upstream Recursive Resolver receives a DNS query over TCP with the CHAIN Closest Trust Point set to "com.". After accepting the query it starts constructing a DNS reply packet.
- o The upstream Recursive Resolver performs all the regular work to ensure it has all the answers to the query for the A record of "www.example.com.". It does so without using the CHAIN option - unless it is also configured as a Forwarder. The answer to the original DNS question could be the actual A record, the DNSSEC proof of non-existence, or an insecure NXDOMAIN response.
- o The upstream Recursive Resolver adds the CHAIN option to the DNS response as follows:
 - * Option-code, set to 13
 - * Option-length, set to 5
 - * The Closest Trust Point is set to "com." (0x03 0x63 0x6f 0x6d 0x00)
- o The upstream Recursive Resolver constructs the DNS Authority Section and fills it (in any order) with:
 - * The DS RRset for "example.com." and its corresponding RRSIGs (made by the "com." DNSKEY(s))
 - * The DNSKEY RRset for "example.com." and its corresponding RRSIGs (made by the "example.com" DNSKEY(s))
 - * The authoritative NS RRset for "example.com." and its corresponding RRSIGs (from the child zone)

If the answer does not exist, and the zone uses DNSSEC, it also adds the proof of non-existence, such as NSEC or NSEC3 records, to the Authority Section.

- o The upstream Recursive Resolver constructs the DNS Answer Section and fills it with:
 - * The A record of "www.example.com." and its corresponding RRSIGs
- If the answer does not exist (NODATA or NXDOMAIN), the Answer Section remains empty. For the NXDOMAIN case, the RCode of the DNS answer packet is set to NXDOMAIN. Otherwise it remains NOERROR.
- o The upstream Recursive Resolver returns the DNS answer over the existing TCP connection. When all data is sent, it SHOULD keep the TCP connection open to allow for additional incoming DNS queries - provided it has enough resources to do so.
- o The Resolver on the client receives the DNS answer. It processes the Authority Section and the Answer Section and places the information in its local cache. It ensures that no data is accepted into the cache without having proper DNSSEC validation. It MAY do so by looping over the entries in the Authority and Answer Sections. When an entry is validated for its cache, it is removed from the processing list. If an entry cannot be validated it is left in the process list. When the end of the list is reached, the list is processed again until either all entries are placed in the cache, or the remaining items cannot be placed in the cache due to lack of validation. Those entries are then discarded.
- o If the cache contains a valid answer to the application's query, this answer is returned to the application via a regular DNS answer packet. This packet MUST NOT contain a CHAIN option. If no valid answer can be returned, normal error processing is done. For example, an NXDOMAIN or an empty Answer Section could be returned depending on the error condition.

9.2. Out-of-path Query for example.com

A Recursive Resolver receives a query for the A record for example.com. It includes the CHAIN option with the following parameters:

- o Option-code, set to 13
- o Option-length, set to 14
- o The Closest Trust Point set to 'unrelated.ca.' (0x09 0x75 0x6e 0x72 0x65 0x6c 0x61 0x74 0x65 0x64 0x03 0x63 0x61 0x00)

As there is no chain that leads from "unrelated.ca." to "example.com", the Resolving Nameserver answers with an empty CHAIN specified using:

- o Option-code, set to 13
- o Option-length, set to 0x00 0x00
- o The Closest Trust Point is omitted (zero length)

Note that the regular answer is still present just as it would be for a query that did not specify the CHAIN option.

9.3. Non-existent data

A Recursive Resolver receives a query for the A record for "ipv6.toronto.redhat.ca". It includes the CHAIN option with the following parameters:

- o Option-code, set to 13
- o Option-length, set to 0x00 0x03
- o The Closest Trust Point set to 'ca.'

Using regular UDP queries towards Authoritative Nameservers, it locates the NS RRset for "toronto.redhat.ca.". When querying for the A record it receives a reply with RCODE "NoError" and an empty Answer Section. The Authority Section contains NSEC3 and RRSIG records proving there is no A RRtype for the QNAME "ipv6.toronto.redhat.ca".

The Recursive Resolver constructs a DNS reply with the following CHAIN option parameters:

- o Option-code, set to 13
- o Option-length, set to 0x00 0x00
- o The Closest Trust Point is omitted (zero length)

The RCODE is set to "NoError". The Authority Section is filled in with:

- o The DS RRset for "redhat.ca." plus RRSIGs
- o The DNSKEY RRset for "redhat.ca." plus RRSIGs
- o The NS RRset for "redhat.ca." plus RRSIGs (eg ns[01].redhat.ca)

- o The A RRset for "ns0.redhat.ca." and "ns1.redhat.ca." plus RRSIGs
- o The DS RRset for "toronto.redhat.ca." plus RRSIGs
- o The NS RRset for "toronto.redhat.ca." plus RRSIGs (eg ns[01].toronto.redhat.ca)
- o The DNSKEY RRset for "toronto.redhat.ca." plus RRSIGs
- o The A RRset and/or AAAA RRset for "ns0.toronto.redhat.ca." and "ns1.toronto.redhat.ca." plus RRSIGs
- o The NSEC record for "ipv6.toronto.redhat.ca." (proves what RRTYPEs do exist, does not include A)
- o The NSEC record for "toronto.redhat.ca." (proves no wildcard exists)

The Answer Section is empty. The RCode is set to NOERROR.

10. IANA Considerations

10.1. EDNS0 option code for CHAIN

IANA has assigned option code 13 in the "DNS EDNS0 Option Codes (OPT)" registry to CHAIN.

11. Acknowledgements

Andrew Sullivan pointed out that we do not need any new data formats to support DNS chains. Olafur Gudmundsson ensured the RRsets are returned in the proper Sections. Thanks to Tim Wicinski for his thorough review.

12. Normative References

[DNS-over-TLS]

Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "DNS over TLS: Initiation and Performance Considerations", draft-ietf-dprive-dns-over-tls-05 (work in progress), January 2016.

[EDNS-COOKIE]

Eastlake, Donald., "Domain Name System (DNS) Cookies", draft-ietf-dnsop-cookies (work in progress), January 2016.

- [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>>.
- [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>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<http://www.rfc-editor.org/info/rfc6824>>.
- [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>>.
- [RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <<http://www.rfc-editor.org/info/rfc6982>>.

- [RFC7646] Ebersman, P., Kumari, W., Griffiths, C., Livingood, J., and R. Weber, "Definition and Use of DNSSEC Negative Trust Anchors", RFC 7646, DOI 10.17487/RFC7646, September 2015, <<http://www.rfc-editor.org/info/rfc7646>>.
- [RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <<http://www.rfc-editor.org/info/rfc7719>>.
- [TCP-KEEPAALIVE] Wouters, P., Abley, J., Dickinson, S., and R. Bellis, "The edns-tcp-keepalive EDNS0 Option", draft-ietf-dnsop-edns-tcp-keepalive-05 (work in progress), January 2016.

Author's Address

Paul Wouters
Red Hat

Email: pwouters@redhat.com

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

P. Wouters
Red Hat
J. Abley
Dyn, Inc.
S. Dickinson
Sinodun
R. Bellis
ISC
February 22, 2016

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

Abstract

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

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

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 25, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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. Introduction	3
2. Requirements Notation	4
3. The edns-tcp-keepalive Option	5
3.1. Option Format	5
3.2. Use by DNS Clients	5
3.2.1. Sending Queries	5
3.2.2. Receiving Responses	6
3.3. Use by DNS Servers	6
3.3.1. Receiving Queries	6
3.3.2. Sending Responses	6
3.4. TCP Session Management	7
3.5. Non-Clean Paths	8
3.6. Anycast Considerations	8
4. Intermediary Considerations	8
5. Security Considerations	9
6. IANA Considerations	9
7. Acknowledgements	9
8. References	9
8.1. Normative References	9
8.2. Informative References	10
Appendix A. Editors' Notes	11
A.1. Abridged Change History	11
A.1.1. draft-ietf-dnsop-edns-tcp-keepalive-06	11
A.1.2. draft-ietf-dnsop-edns-tcp-keepalive-05	11
A.1.3. draft-ietf-dnsop-edns-tcp-keepalive-04	12
A.1.4. draft-ietf-dnsop-edns-tcp-keepalive-03	12
A.1.5. draft-ietf-dnsop-edns-tcp-keepalive-02	12
A.1.6. draft-ietf-dnsop-edns-tcp-keepalive-01	13
A.1.7. draft-ietf-dnsop-edns-tcp-keepalive-00	13
A.1.8. draft-wouters-edns-tcp-keepalive-01	13
A.1.9. draft-wouters-edns-tcp-keepalive-00	13

Authors' Addresses	13
--------------------	----

1. Introduction

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

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

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

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

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

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

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

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

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

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

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

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

2. Requirements Notation

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

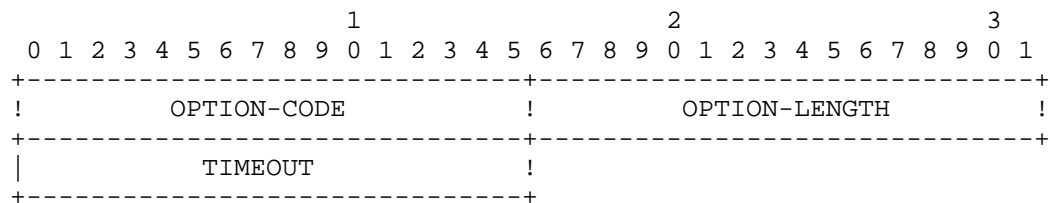
3. The edns-tcp-keepalive Option

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

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

3.1. Option Format

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



where:

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

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

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

3.2. Use by DNS Clients

3.2.1. Sending Queries

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

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

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

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

3.2.2. Receiving Responses

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

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

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

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

3.3. Use by DNS Servers

3.3.1. Receiving Queries

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

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

3.3.2. Sending Responses

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

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

3.4. TCP Session Management

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

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

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

However it should be noted that RCF6891 states:

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

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

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

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

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

3.5. Non-Clean Paths

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

3.6. Anycast Considerations

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

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

4. Intermediary Considerations

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

5. Security Considerations

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

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

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

6. IANA Considerations

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

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

7. Acknowledgements

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

8. References

8.1. Normative References

- [DRAFT-5966bis] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", draft-ietf-dnsop-5966bis (work in progress), January 2016.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<http://www.rfc-editor.org/info/rfc1035>>.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <<http://www.rfc-editor.org/info/rfc4033>>.
- [RFC4786] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, DOI 10.17487/RFC4786, December 2006, <<http://www.rfc-editor.org/info/rfc4786>>.
- [RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <<http://www.rfc-editor.org/info/rfc6891>>.
- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<http://www.rfc-editor.org/info/rfc7320>>.

8.2. Informative References

- [CHAIN-QUERY] Wouters, P., "Chain Query requests in DNS", draft-ietf-dnsop-edns-chain-query (work in progress), January 2016.
- [DNS-over-TLS] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "TLS for DNS: Initiation and Performance Considerations", draft-ietf-dprive-dns-over-tls (work in progress), January 2016.
- [fragmentation-considered-poisonous] Herzberg, A. and H. Shulman, "Fragmentation Considered Poisonous", arXiv 1205.4011, May 2012, <<http://arxiv.org/abs/1205.4011>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<http://www.rfc-editor.org/info/rfc6824>>.
- [RRL] Vixie, P. and V. Schryver, "DNS Response Rate Limiting (DNS RRL)", ISC-TN 2012-1-Draft1, April 2012, <<http://ss.vix.su/~vixie/isc-tn-2012-1.txt>>.

Appendix A. Editors' Notes

A.1. Abridged Change History

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

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

Introduction: Moved paragraph 8 to paragraph 2 for readability.

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

Section 3: Add explicit description of the idle timeout.

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

Section 3.3.2: remove discussion of intermediary behaviour.

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

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

Section 3.2.2: should -> SHOULD

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

Reword sentence referring to RFC6824 since it is informational.

Update IANA option to Standard.

Remove last sentence from 1st paragraph of introduction.

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

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

Correct missing URIs in 2 references.

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

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

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

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

Changed the should to a SHOULD in section 3.2.2

Changed Nameserver to DNS server in section 5.

Updated references.

Changed reference to RFC6824 to be informative.

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

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

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

Corrected TIMEOUT length from 4 to 2 in the diagram.

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

Updated wording in section on Intermediary Considerations.

Updated wording describing RRL.

Added paragraph to security section describing client behaviour profiles.

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

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

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

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

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

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

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

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

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

Added reference to STARTTLS draft and RFC7320.

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

Version bump with no changes

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

Clarifications, working group adoption.

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

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

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

Initial draft.

Authors' Addresses

Paul Wouters
Red Hat

Email: pwouters@redhat.com

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

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

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

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

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

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

Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: June 27, 2017

P. Koch
DENIC eG
M. Larson
P. Hoffman
ICANN
December 24, 2016

Initializing a DNS Resolver with Priming Queries
draft-ietf-dnsop-resolver-priming-11

Abstract

This document describes the queries that a DNS resolver should emit to initialize its cache. The result is that the resolver gets both a current NS RRSet for the root zone and the necessary address information for reaching the root servers.

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 June 27, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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.

1. Introduction

Recursive DNS resolvers need a starting point to resolve queries. [RFC1034] describes a common scenario for recursive resolvers: they begin with an empty cache and some configuration for finding the names and addresses of the DNS root servers. [RFC1034] describes that configuration as a list of servers that will give authoritative answers to queries about the root. This has become a common implementation choice for recursive resolvers, and is the topic of this document.

This document describes the steps needed for this common implementation choice. Note that this is not the only way to start a recursive name server with an empty cache, but it is the only one described in [RFC1034]. Some implementers have chosen other directions, some of which work well and others of which fail (sometimes disastrously) under different conditions. For example, an implementation that only gets the addresses of the root name servers from configuration, not from the DNS as described in this document, will have stale data that could cause slower resolution.

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].

This document only deals with recursive name servers (recursive resolvers, resolvers) for the IN class.

2. Description of Priming

Priming is the act of finding the list of root servers from a configuration that lists some or all of the purported IP addresses of some or all of those root servers. A recursive resolver starts with no information about the root servers, and ends up with a list of their names and their addresses.

Priming is described in Sections 5.3.2 and 5.3.3 of [RFC1034]. The scenario used in that description, that of a recursive server that is also authoritative, is no longer as common.

The configured list of IP addresses for the root servers usually comes from the vendor or distributor of the recursive server software. This list is usually correct and complete when shipped, but may become out of date over time.

The list of root server operators and the domain name associated with each one has been stable since 1997. However, there are address changes for the root server domain names, both for IPv4 and IPv6 addresses. However, research shows that after those addresses change, some resolvers never get the new addresses. Therefore, it is important that resolvers be able to cope with change, even without relying upon configuration updates to be applied by their operator. Root server change is the main reason that resolvers need to do priming instead of just going from a configured list to get a full and accurate list of root servers.

3. Priming Queries

A priming query is a DNS query used to get the root server information in a resolver. It has a QNAME of "." and a QTYPE of NS, and is sent to one of the addresses in the configuration for the recursive resolver. The priming query can be sent over either UDP or TCP. If the query is sent over UDP, the source port SHOULD be randomly selected (see [RFC5452]). The RD bit MAY be set to 0 or 1, although the meaning of it being set to 1 is undefined for priming queries.

The recursive resolver SHOULD use EDNS0 [RFC6891] for priming queries and SHOULD announce and handle a reassembly size of at least 1024 octets [RFC3226]. Doing so allows responses that cover the size of a full priming response (see Section 4.2) for the current set of root servers. See Section 3.3 for discussion of setting the DNSSEC OK (DO) bit (defined in [RFC4033]).

3.1. Repeating Priming Queries

The recursive resolver SHOULD send a priming query only when it is needed, such as when the resolver starts with an empty cache and when the NS RRset for the root zone has expired. Because the NS records for the root are not special, the recursive resolver expires those NS records according to their TTL values. (Note that a recursive resolver MAY pre-fetch the NS RRset before it expires.)

If a priming query does not get a response, the recursive resolver needs to retry the query with a different target address from the configuration.

3.2. Target Selection

In order to spread the load across all the root server domain names, the recursive resolver SHOULD select the target for a priming query randomly from the list of addresses. The recursive resolver might choose either IPv4 and IPv6 addresses based on its knowledge of

whether the system on which it is running has adequate connectivity on either type of address.

Note that this recommended method is not the only way to choose from the list in a recursive resolver's configuration. Two other common methods include picking the first from the list, and remembering which address in the list gave the fastest response earlier and using that one. There are probably other methods in use today. However, the random method listed above SHOULD be used for priming.

3.3. DNSSEC with Priming Queries

The resolver MAY set the DNSSEC OK (DO) bit. At the time this document is being published, there is little use to performing DNSSEC validation on the priming query. Currently all root name server names end in "root-servers.net" and the AAAA and A RRsets for the root server names reside in the "root-servers.net" zone. All root servers are also authoritative for this zone, allowing priming responses to include the appropriate root name server A and AAAA RRsets. But because the "root-servers.net" zone is not currently signed, these RRsets cannot be validated.

A man-in-the-middle attack on the priming query could direct a resolver to a rogue root name server. Note, however, that a validating resolver will not accept responses from rogue root name servers if they are different from the real responses because the resolver has a trust anchor for the root and the answers from the root are signed. Thus, if there is a man-in-the-middle attack on the priming query, the only result for a validating resolver will be a denial of service, not the resolver's accepting the bad responses.

If the "root-servers.net" zone is later signed, or if the root servers are named in a different zone and that zone is signed, having DNSSEC validation for the priming queries might be valuable.

4. Priming Responses

A priming query is a normal DNS query. Thus, a root name server cannot distinguish a priming query from any other query for the root NS RRSet. Thus, the root server's response will also be a normal DNS response.

4.1. Expected Properties of the Priming Response

The priming response is expected to have an RCODE of NOERROR, and to have the AA bit set. Also, it is expected to have an NS RRSet in the Answer section (because the NS RRSet originates from the root zone), and an empty Authority section (because the NS RRSet already appears

in the Answer section). There will also be an Additional section with A and/or AAAA RRSets for the root name servers pointed at by the NS RRSets.

Resolver software SHOULD treat the response to the priming query as a normal DNS response, just as it would use any other data fed to its cache. Resolver software SHOULD NOT expect exactly 13 NS RRs because historically some root servers have returned fewer.

4.2. Completeness of the Response

There are currently 13 root servers. All have one IPv4 address and one IPv6 address. Not even counting the NS RRSets, the combined size of all the A and AAAA RRSets exceeds the original 512 octet payload limit from [RFC1035].

In the event of a response where the Additional section omits certain root server address information, re-issuing of the priming query does not help with those root name servers that respond with a fixed order of addresses in the Additional section. Instead, the recursive resolver needs to issue direct queries for A and AAAA RRSets for the remaining names. Currently, these RRSets would be authoritatively available from the root name servers.

5. Security Considerations

Spoofing a response to a priming query can be used to redirect all of the queries originating from a victim recursive resolver to one or more servers for the attacker. Until the responses to priming queries are protected with DNSSEC, there is no definitive way to prevent such redirection.

An on-path attacker who sees a priming query coming from a resolver can inject false answers before a root server can give correct answers. If the attacker's answers are accepted, this can set up the ability to give further false answers for future queries to the resolver. False answers for root servers are more dangerous than, say, false answers for TLDs, because the root is the highest node of the DNS. See Section 3.3 for more discussion.

In both of the scenarios above, a validating resolver will be able to detect the attack if its chain of queries comes to a zone that is signed, but not for those that are unsigned.

6. IANA Considerations

None.

7. 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>>.
- [RFC3226] Gudmundsson, O., "DNSSEC and IPv6 A6 aware server/resolver message size requirements", RFC 3226, DOI 10.17487/RFC3226, December 2001, <<http://www.rfc-editor.org/info/rfc3226>>.
- [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>>.
- [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>>.

Appendix A. Acknowledgements

This document is the product of the DNSOP WG and benefitted from the reviews done there.

Authors' Addresses

Peter Koch
DENIC eG
Kaiserstrasse 75-77
Frankfurt 60329
DE

Phone: +49 69 27235 0
Email: pk@DENIC.DE

Matt Larson
ICANN

Email: matt.larson@icann.org

Paul Hoffman
ICANN

Email: paul.hoffman@icann.org

Network Working Group
Internet-Draft
Intended status: BCP
Expires: April 21, 2014

J. Abley
Dyn, Inc.
October 18, 2013

DNS Reverse Mapping for Multicast Addresses
draft-jabley-multicast-ptr-00

Abstract

The mapping of IPv4 and IPv6 addresses to names using the Domain Name System (DNS) is colloquially known as "Reverse Mapping". Reverse Mapping support for registered multicast address assignments in IPv4 is currently incomplete and ad-hoc; in IPv6 there is no support at all.

This document describes procedures to be followed that will result in more systematic and predictable support for Reverse Mapping for IPv4 multicast address assignments, and introduces analogous support for IPv6.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

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. Introduction	3
2. General Approach	4
3. Naming Scheme	5
3.1. IPv4 Multicast Addresses	5
3.2. IPv6 Multicast Addresses	5
4. Use of MCAST.ARPA	6
5. IAB Considerations	7
6. IANA Considerations	8
6.1. Registry Changes	8
6.1.1. IPv6 Multicast Scope Registry	8
6.1.2. IPv4 Multicast Address Space Registries	8
6.1.3. IPv6 Multicast Address Space Registries	9
6.2. Delegation of MCAST.ARPA and MCAST6.ARPA	9
6.3. Initial Zone Contents	9
6.4. Process Changes	11
6.5. Ongoing Support for MCAST.NET	11
7. Security Considerations	13
8. Acknowledgements	14
9. References	15
9.1. Normative References	15
9.2. Informative References	15
Appendix A. Editorial Notes	16
A.1. Change History	16
Author's Address	17

1. Introduction

The Domain Name System (DNS), as originally specified in [RFC1034] and [RFC1035], provides support for the mapping of IPv4 addresses to names using a namespace convention within the IN-ADDR.ARPR domain and the PTR resource record type.

The analogous mapping of IPv6 address to names is specified in [RFC3596], adopting a similar namespace convention within the IP6.ARPA domain.

Multicast addresses are assigned by the IANA, and assignments are documented in various IANA registries.

For IPv4, Reverse Mapping of assigned multicast addresses to names has historically been provided in an ad-hoc and incomplete fashion, without tight coordination with IANA multicast address assignment processes. Names assigned to IPv4 multicast addresses have been chosen somewhat arbitrarily within the MCAST.NET domain. For IPv6, no Reverse Mapping is provided.

This document describes procedures to be followed by the IANA to support predictable and consistent Reverse Mapping for registered multicast addresses in IPv4 and IPv6.

2. General Approach

This document specifies extensions to existing IPv4 and IPv6 multicast registries to include a mandatory column "DNS Label". This field is required to be populated with a unique, valid DNS label for all future multicast address assignments except in the case where reverse mapping for an address is explicitly not desirable.

The procedures at the IANA relating to multicast address assignment are extended to include the provisioning of appropriate changes in the DNS at the time of registration or de-registration of any multicast addresses. Specific actions requested of the IANA are described in Section 6.

Names for multicast addresses are assigned under MCAST.ARPA for IPv4 addresses, and MCAST6.ARPA for IPv6 addresses. The naming schemes to be used in each case are described in Section 3. The use of MCAST.ARPA rather than MCAST.NET is discussed in Section 4.

3. Naming Scheme

3.1. IPv4 Multicast Addresses

Each assigned IPv4 multicast address has an accompanying DNS Label. The name associated with an IPv4 multicast address with DNS Label SOMENAME is SOMENAME.MCAST.ARPA.

For example, suppose the assigned IPv4 multicast address 224.0.1.1 has the DNS Label "NTP". The address 224.0.1.1 maps to the name "NTP.MCAST.ARPA"; the name "NTP.MCAST.ARPA" maps to the address 224.0.1.1.

3.2. IPv6 Multicast Addresses

Each assigned IPv6 multicast address has an accompanying DNS Label ("Address Label").

IPv6 multicast addresses may be of fixed or variable scope. The naming scheme for these addresses incorporates a scope identifier using an additional DNS label ("Scope Label"), specified in a dedicated registry (see Section 6.1.1). Both fixed and variable scope multicast addresses use the same naming scheme.

The name associated with an IPv6 multicast address with Scope Label SCOPE and Address Label SOMENAME is SOMENAME.SCOPE.MCAST6.ARPA.

For example, suppose ff01::1 is an assigned IPv6 multicast address with Scope Label "NODE-LOCAL" and Address Label "ALL-NODES". The address ff01::1 maps to the name "ALL-NODES.NODES-LOCAL.MCAST6.ARPA"; the name "ALL-NODES.NODES-LOCAL.MCAST6.ARPA" maps to the address ff01::1.

The variable-scope multicast address ff0x::fb will have different Reverse Mapping depending on the scope specified in the address (i.e. the value of x), although the Address Label in each case will be the same ("MDNSV6"). The Site-Local address ff05::fb has an associated Scope Label "SITE-LOCAL", and is therefore named MDNSV6.SITE-LOCAL.MCAST6.ARPA. The Link-Local address ff02::fb has an associated Scope Label "LINK-LOCAL" and hence is named MDNSV6.LINK-LOCAL.MCAST6.ARPA.

4. Use of MCAST.ARPA

Use of the MCAST.ARPA domain rather than MCAST.NET for IPv4 multicast addresses is specified for the same reasons that led IP6.INT to be superceded by "IP6.ARPA" [RFC3152].

It is prudent to assume that hard-coded assumptions about names in MCAST.NET exist, and will persist for some time. This document specifies that names in the MCAST.ARPA domain also be available in the MCAST.NET domain, to provide support for software with those assumptions. Ongoing support for the MCAST.NET zone is described in Section 6.5.

It is possible that in the future empirical measurement will confirm that the use of names under MCAST.NET is no longer required and that provisioning of the MCAST.NET domain can safely cease. This document provides no such measurement and makes no such recommendation, however.

5. IAB Considerations

This document proposes a delegation within the ARPA domain, and, in accordance with [RFC3172], IAB review and approval of the delegation of MCAST.ARPA and MCAST6.ARPA as described in Section 6.2 is required.

Once IAB approval has been obtained, this section may be removed prior to publication or updated to include text that confirms the IAB's decision, at the IAB's discretion.

6. IANA Considerations

6.1. Registry Changes

6.1.1. IPv6 Multicast Scope Registry

The IANA is directed to create a new registry as follows:

Registry Name: IPv6 Multicast Address Scopes

Registration Procedure: Standards Action

Reference: This document

Schema: See initial contents, below. Note that the Scope Value and DNS Label fields are mandatory for all rows, and that values chosen for future DNS Label fields are required to be unique within this registry.

The initial contents of this new registry should be:

Scope Value	DNS Label	Scope Name	Reference
0x0	(none)	Reserved	[RFC4291]
0x1	NODE-LOCAL	Node-Local Scope	[RFC4291]
0x2	LINK-LOCAL	Link-Local Scope	[RFC4291]
0x3	(none)	Reserved	[RFC4291]
0x4	ADMIN-LOCAL	Admin-Local Scope	[RFC4291]
0x5	SITE-LOCAL	Site-Local Scope	[RFC4291]
0x8	ORG-LOCAL	Organisation-Local Scope	[RFC4291]
0xE	GLOBAL	Global Scope	[RFC4291]

The IANA may add "Date Registered" and "Last Revised" columns to the schema at its discretion.

6.1.2. IPv4 Multicast Address Space Registries

The IANA is directed to add a mandatory "DNS Label" column to all IPv4 Multicast Address Space registries. The initial contents of the

DNS Label field for each row should be taken from the corresponding MCAST.NET zone owner names where available; addresses with no existing mapping in MCAST.NET should have DNS Labels assigned by the IANA at their discretion.

All existing assignments should have a DNS Label assigned. A DNS Label should be mandatory for all future registrations. DNS Labels are required to be unique for all IPv4 multicast address assignments.

6.1.3. IPv6 Multicast Address Space Registries

The IANA is directed to add a mandatory "DNS Label" column to all IPv6 Multicast Address Space registries. The initial contents of the DNS Label field for each row should be assigned by the IANA at their discretion.

All existing assignments should have a DNS Label assigned. A DNS Label should be mandatory for all future registrations. DNS Labels are required to be unique for all IPv6 multicast address assignments.

6.2. Delegation of MCAST.ARPA and MCAST6.ARPA

The IANA is directed to create and host the MCAST.ARPA and MCAST6.ARPA zones on name servers of their choosing. The MCAST.ARPA and MCAST6.ARPA zones should be signed using DNSSEC, with DNSSEC parameters chosen by the IANA. The initial zone contents should be as described in Section 6.3.

The IANA is directed to provision secure delegations for the MCAST.ARPA and MCAST6.ARPA zones from the ARPA zone (i.e. delegations with accompanying DS RRsets).

6.3. Initial Zone Contents

The IANA is directed to populate the MCAST.ARPA and MCAST6.ARPA zones, and the corresponding reverse mapping zones under IN-ADDR.ARPA and IP6.ARPA, directly from the IPv4 and IPv6 Multicast Address Registries, amended as described in Section 6.1.

As an example, if the IPv6 Variable Scope Multicast Addresses sub-registry contained the following entry:

Address(es)	Description	DNS Label
FF0X::FB	mDNSv6	MDNSV6


```
$ORIGIN MCAST.ARPA.  
;  
; SGI-Dogfight address  
;  
SGI-DOG                A          224.0.1.2  
  
$ORIGIN 224.IN-ADDR.ARPA.  
;  
; SGI-Dogfight address  
;  
2.1.0                  PTR        SGI-DOG.MCAST.ARPA.
```

6.4. Process Changes

The IANA is directed to require a valid and unique DNS Label to be specified within the existing processes of multicast address assignment in IPv4 and IPv6.

The IANA is further directed to maintain the MCAST.ARPA, MCAST6.ARPA and related domains under IP6.ARPA and IN-ADDR.ARPA such that any additions, changes or deletions from the corresponding address registries are reflected accurately in the DNS.

6.5. Ongoing Support for MCAST.NET

IANA is directed to remove all non-apex resource records from the MCAST.NET zone and to add an apex DNAME [RFC6672] with target MCAST.ARPA. The intention is to provide backwards compatibility for software that has hard-coded assumptions about naming conventions for IPv4 multicast addresses.

For example, the following describes the result of this change for MCAST.NET SOA serial 2012123836, with DNSSEC resource records omitted for clarity:

```
$ORIGIN MCAST.NET.
```

```
; beginning of zone
```

```
@      SOA      SNS.DNS.ICANN.ORG. NOC.DNS.ICANN.ORG. (  
                                2012123836  
                                7200  
                                3600  
                                604800  
                                3600 )
```

```
NS      A.IANA-SERVERS.NET.  
NS      B.IANA-SERVERS.NET.  
NS      C.IANA-SERVERS.NET.  
NS      NS.ICANN.ORG.
```

```
DNAME   MCAST.ARPA.
```

```
; end of zone
```

7. Security Considerations

This document presents no known additional security concerns to the Internet.

8. Acknowledgements

Your name here, etc.

9. References

9.1. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
- [RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", RFC 3596, October 2003.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

9.2. Informative References

- [RFC3152] Bush, R., "Delegation of IP6.ARPA", BCP 49, RFC 3152, August 2001.
- [RFC3172] Huston, G., "Management Guidelines & Operational Requirements for the Address and Routing Parameter Area Domain ("arpa")", BCP 52, RFC 3172, September 2001.
- [RFC6672] Rose, S. and W. Wijngaards, "DNAME Redirection in the DNS", RFC 6672, June 2012.

Appendix A. Editorial Notes

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

A.1. Change History

00 Initial draft, circulated for the purposes of entertainment.

Author's Address

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

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

DNSOP Working Group
Internet Draft
Expires: January 2015

Xiaodong Lee
CNNIC
Vixie Paul
Farsight Security, Inc.
Zhiwei Yan(Ed.)
CNNIC
July 3, 2014

How to scale the DNS root system?
draft-lee-dnsop-scalingroot-00.txt

Abstract

In Domain Name System (DNS), 13 root servers are deployed in order to provide the entrance of domain name resolution and they are denoted by 13 letters. From 2000 or so, the anycast technology has been adopted to extend the number of root servers with the explosive development of Internet. To date, 11 of the 13 letters are hosted at multiple sites, and the root zone is served at about 380 sites around the globe. However, increasing mirror sites is not a perfect solution to scale the DNS root servers because the geographical distribution of the current 13 root servers is uneven and only increasing mirror sites cannot solve the efficiency and scalability issues caused by that. Then we propose a new DNS root system in this draft based on the widely deployed Domain Name System Security Extensions (DNSSEC). The proposed architecture is scalable and secure enough to meet the current and future needs to scale the DNS root system in an easy-deployment way.

Requirements Language

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

Status of this Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents

at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on January, 2015.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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. Introduction	2
2. Motivations	3
3. Proposed architecture.....	5
4. Management considerations.....	8
5. Security considerations.....	9
6. References	9
Author's Address	10
Acknowledgment	11

1. Introduction

In Domain Name System (DNS), 13 root servers are deployed and they are denoted by 13 letters from A to M. From 2000 or so, the anycast technology has been adopted to extend the number of root servers with the explosive development of Internet. To date, 11 of the 13 letters are hosted at multiple sites, and the root zone is served at about 380 sites around the globe [1,2].

With the continuous development of Internet, more and more DNS root servers have to be deployment in order to meet the efficiency and stability requirements of the DNS root system. On one hand, the DNS resolution efficiency should be high enough to support more and more Internet users and their sophisticated applications. On the other hand, the DNS root system has to be stable enough to face the cyber attacks which are becoming more and more serious [3]. Then, how about deploying more anycast mirror sites? Without doubt, this scheme can improve the efficiency and stability of DNS root system as usual, but its problems are:

a) This scheme is not open enough to satisfy the special requirements of a country or district or organization (CDO). If a CDO wants to deploy a root server to serve its local users, the current procedure is complex and it is almost impossible for the CDO to control the deployed site no matter the type of the site is local or global. Specially, the current root name servers have respective Root Name Server Operators (RNSOs), to whom IP address blocks have been allocated. The CDO wants a root server mirror site must coordinate anycast routing configuration and server placement with the corresponding RNSO.

b) This scheme is not stable enough. When one root server is attacked and fails down, its anycast mirrors cannot do the zone file synchronization and the DNS root service may be affected.

2. Motivations

During recent years, many CDOs declare their anticipation to host a DNS root server. And how to adjust the placement of the current 13 DNS root servers also drew a lot of attention from both academic and technical worlds [2]. However, it's very difficult to change the current belonging situation of DNS root servers. If possible, it's a good idea to deploy more root servers (not just the mirror sites) to solve or relieve the above problems.

In the IPv4 based Internet, only 13 root servers are allowed due to the 512-byte limitation. However, adding the IPv6 address information for more than two root name servers will increase the size of the DNS priming response to exceed this maximum. Ultimately, when all 13 root name servers assign IPv6 addresses, the priming response will be increased to 811 bytes [3]. Specially, if a message cannot fit in the 512 bytes, the server must set a special flag indicating that the message was truncated. When receiving such a message, a DNS resolver will normally retry the transaction using TCP instead of UDP [4]. However, the costs of using TCP rather than UDP, in terms of system and network resources, are much higher and

can have significant impact on systems such as name servers that may receive several thousands of queries per second during normal operations. At the same time, with the promotion from ICANN, work is in progress to put forward a recommendation that will enable the publication of IPv6 addresses for, at least, DNS root servers in as short a time as possible. Besides, with the exhaustion of IPv4 address, the IPv6 is promoted from all parties in the community. Currently, 10 IPv6 addresses have been configured on the root servers and the total size of the DNS priming response message has been increased to above 512 bytes (We even do not consider the DNSSEC herein, which increases the DNS response message more tremendously).

Above background means that the 512-byte limitation does not actually exist anymore in the DNS [5]. Then how many DNS root server can be increased further? We take the IPv6 supported Internet as an example, then the DNS priming response message structure is shown in the following if we set the maximum number of root servers is N [6].

--Required Header: 12 bytes

--Query: 5 bytes

--Answer: $31+15*(N-1)$ [the first answer is 31 bytes, the sequent answers are reduced by 16 bytes due to compression]

--Additional Records: $16*N$ [each A record is 16 bytes]

--Additional Records: $28*N$ [each AAAA record is 28 bytes]

The IPv6 supported node is not required to reduce the size of subsequent packets to less than 1280 bytes, but must include a "Fragment Header" in those packets so that the IPv6-to-IPv4 translating router can obtain a suitable "Identification" value to use in resulted IPv4 fragments. Note that this means the payload may have to be reduced to 1232 bytes (1280 minus 40 for the IPv6 header and 8 for the Fragment header) [7].

Then we get the following formula,

$$12+5+31+15*(N-1)+16*N+28*N<1232$$

Accordingly, $N=20$, which means that the root servers can only be increased by 7 more in the IPv6 transport environment (without EDNS0 [8] and TCP supporting).

In advance, DNSSEC should be considered because it is necessary to guarantee the security of the DNS information and it has been widely deployed (and will be more widely deployed in the future). If only one kind of signature algorithm is used to generate the RRSIG resource record, the size of DNS response message will be increased to about 7 to 10 times comparing with the original one without signing [9]. It means that it is unpromising to increase the root servers further in the current architecture.

In the security aspect, criticisms of the current and historical root name server system is lack of resistance to DDoS attack, noting that even with the current wide scale anycasting by every RNSO, there are still only a few hundred name servers in the world to answer authoritatively for the DNS root zone. We are also concerned that reachability of the root name server system is required even for purely local communication, since otherwise local clients have no way to discover local services. In a world sized distributed system like the Internet, critical services such as DNS root system ought to be extremely well distributed.

In a word, we have to design a new architecture to scale the DNS root name server system and in this way the root server deployment balance can be reconsidered (the future deployment can refer the actual requirements such as the number of Internet users, amount of Internet traffic and so on). Besides, this architecture should scale the DNS root servers without the technology limitations as illustrated above.

3. Proposed architecture

The proposed architecture is strongly based on the widely deployed DNSSEC and shown in Figure 1. With DNSSEC, it is now possible for recursive name server operators to configure DNSSEC validation, such that any gTLD information heard from a root server (or mirror site) must be IANA-approved as indicated by DNSSEC signatures made with IANA's root Zone Signing Key (ZSK).

For the universal anycast root server nodes deployment, we here provide two actual solutions:

- 1) The DNS root service manager (may still be the IANA) would generate and digitally sign (with DNSSEC) an additional version of the root zone that has a different set of NS records at its apex. These NS records will denote name servers whose addresses are not assigned to any current RNSO but are instead held in trust by IANA for use by any or all interested CDOs (Global Root X in Figure 1). IANA would request infrastructure micro-allocations from an RIR

(such as ARIN or APNIC), as several IPv4 24-bit prefixes and several IPv6 48-bit prefixes, for use in universal anycasting of the root zone. For example, the following configuration can be used corresponding to the universal root server (Global Root X):

```
. IN NS anycast-X1.iana-servers.net.
```

```
. IN NS anycast-X2.iana-servers.net.
```

```
$ORIGIN iana-servers.net.
```

```
anycast-x1 IN AAAA 2001::1::1
```

```
anycast-x1 IN A   ??.1.1
```

```
anycast-x2 IN AAAA 2001::2::2
```

```
anycast-x2 IN A   ??.2.2
```

- 2) Based on the contract or approval of one or multiple RNSO, the related root server can also be globally anycasted or locally deployed and totally managed and controlled by a CDO (Root A1 in Figure 1). This case is backward-compatible and does not need to change the current DNS root system.

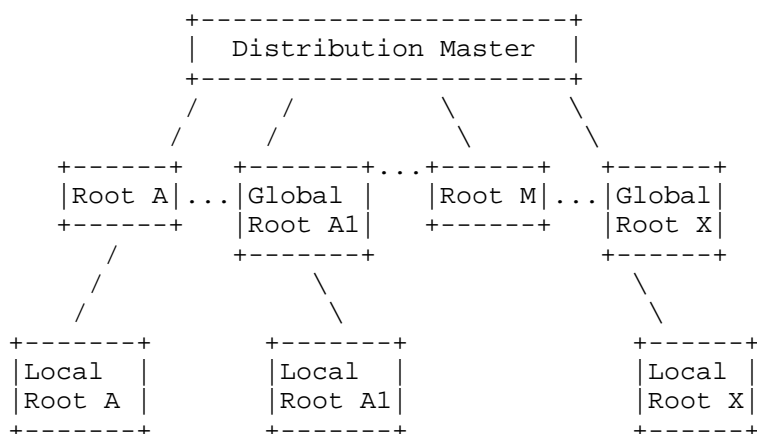


Figure 1. DNS root server architecture

We herein assume that the DNSSEC is widely deployed. In this way, the root zone file will not be tampered or misused. If the DNS root

currently operated by the root server, the root server starts the Zone Transfer (XFR) to retrieve the new root zone file. If this serial number in the SOA response is not higher than the root server's currently used version, the root server will take no further action. If this scheme is adopted, each CDO root server has to register with the DM in order to receive the notify message and which is the requirement of the first deployment solution illustrated above. While, for the second deployment solution, the DM function may be the root server managed by the current RNSO.

Of course, a more open scheme is needed in the future for the root zone file synchronization. Because the root zone file data is solid and cannot be tampered, the CDO can actively fetch the root zone file data according to its special requirement and configuration. We mention this synchronization type because the CDO root servers may be increased significantly in the future and the traditional scheme explained above is not flexible and efficient enough.

In a word, DNSSEC makes it possible to deploy the DNS root servers in a more distributed and flat manner, because any server can synchronize the root zone file without modification.

4. Management considerations

Considering the deployment of the architecture proposed in this draft, the following management questions should be answered:

1) How to manage the universal anycast DNS root servers?

We introduce the idea to globally distribute the root servers and locally deploy multiple local nodes. In the routing aspect, the anycast addresses will be broadcasted more widely for the global nodes to be accessed anywhere. However, the addresses of the local nodes have to be controlled in the local area (with the no-export attribute in BGP routing protocol) to serve the requests from the particular local area.

2) What are the principles to deploy the universal DNS root servers?

The principles or rules to select the service provider (CDO) and deploy the root servers can refer to [10,11]. Besides, more actual aspects can be considered in this new architecture due to its minimized limitation to deploy a root server.

Other management and deployment issues will be added further.

5. Security considerations

Although this architecture maintains the basic operations of the current DNS root system and follows the standard DNS protocols, it changes the current DNS root system because we want this mature system to be flexibly scaled with the Internet. Then the following issues related to security and stability may be caused:

1) Routing table: according to this architecture, more than 13 root servers may broadcast their IP addresses globally. This will increase the entries of the BGP routing table (even maybe only one pair of additional anycast addresses (IPv4 and IPv6)).

2) Attack defense: due to the deployment of anycast servers is more widely and the anycast nodes are increased a lot. How to secure these nodes will be harder and more important. In order to manage the increased global/local nodes, more sophisticated tools (such as CHOAS resource record, Traceroute command, special monitoring and analyzing platform) should be adopted and developed. In this way, the anycast nodes can be identified and monitored effectively and efficiently.

Of course, we believe that the future DNS root service provider (many ccTLD and gTLD have deployed anycast service and manage it well) for the global or local anycast nodes will have the experience and ability to monitor and manage the servers and the possible attack (such as DDoS) can be effectively detected and defended [12].

Other security issues will be discussed and detailed further.

6. References

- [1] <http://www.root-servers.org/>.
- [2] T. Lee, B. Huffaker, M. Fomenkov, and k. claffy. On the problem of optimization of DNS root servers' placement. In Proc. of Passive and Active Network Measurement Workshop (PAM). April 2003.
- [3] P. Vixie and A. Kato. DNS Response Size Issues. draft-ietf-dnsop-respsize-15.txt. February 2014.
- [4] R. Bellis. DNS Transport over TCP - Implementation Requirements. IETF RFC 5966. August 2010.
- [5] CAIDA. Analysis of the DNS root and gTLD nameserver system: status and progress report. May 2008.

- [6] ICANN. Accommodating IP Version 6 Address Resource Records for the Root of the Domain Name System. January 2007.
- [7] S. Deering, and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. IETF RFC 2460. December 1998.
- [8] P. Vixie. Extension Mechanisms for DNS (EDNS0). IETF RFC 2671. August 1999.
- [9] http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_7-2/dnssec.html.
- [10] S. Sato, T. Matsuura, and Y. Morishita. BGP Anycast Node Requirements for Authoritative Name Servers draft-sato-dnsop-anycast-node-requirements-02.txt. September 2007.
- [11] R. Bush, D. Karrenberg, M. Koster, and R. Plzak. Root Name Server Operational Requirements. IETF RFC 2870. June 2000.
- [12] USC/ISI Technical Report. Identifying and Characterizing Anycast in the Domain Name System. May 2011.

Author's Address

Xiaodong Lee
China Internet Network Information Center
No.4 South 4th Street, Zhongguancun
Beijing, P. R. China
Email: xl@cnnic.cn

Paul Vixie
Farsight Security, Inc.
155 Bovet Road, #476
San Mateo, CA 94402, USA
Email: vixie@farsightsecurity.com

Zhiwei Yan
China Internet Network Information Center
No.4 South 4th Street, Zhongguancun
Beijing, P. R. China
Email: yanzhiwei@cnnic.cn

Acknowledgment

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

DNSOP
Internet-Draft
Intended status: Experimental
Expires: January 5, 2015

J. Lundstroem
.SE
W. Mekking
NLnet Labs
July 4, 2014

DNSSEC Key and Signing Policies
draft-mekking-dnsop-kasp-00

Abstract

This document describes how key policies should look like.

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 5, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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. Introduction	2
1.1. Key Words	2
1.2. Terminology	2
1.3. Data Modeling	3
1.3.1. Data Modeling Types	3
1.3.2. Data Modeling Arguments	3
2. KASP Contents	3
2.1. Preamble	3
2.1.1. Policies	4
2.1.1.1. Signatures	4
2.1.1.2. Authenticated Denial of Existence	6
2.1.1.3. Keys	8
2.1.1.4. Zone	11
2.1.1.5. Parent	13
3. References	14
3.1. Informative References	14
3.2. Normative References	14
Appendix A. Changelog	16

1. Introduction

A key and signing policy (KASP) defines a DNSSEC [RFC4033] [RFC4034] [RFC4035] policy for one or more zones.

1.1. Key Words

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

1.2. Terminology

The reader is assumed to be familiar with DNSSEC described in [RFC4033] [RFC4034] [RFC4035], [RFC5155] and [RFC6781].

The following terminology is used throughout this document:

KASP: Key And Signing Policy, describes a DNSSEC policy that can be applied to one or more zones.

A key and signing policy can be expressed in any format. This document uses XML as example.

1.3. Data Modeling

1.3.1. Data Modeling Types

This document reuses the modeling types described in [RFC6020].

The following modeling types are used:

container: A container node does not have a value, but it has a list of child nodes in the data tree. The child nodes are defined in the container's substatements.

leaf: A leaf node has a value, but no child nodes in the data tree. A leaf node exists in zero or one instances in the data tree.

list: The "list" statement is used to define an interior data node in the schema tree. A list node may exist in multiple instances in the data tree. Each such instance is known as a list entry.

choice: The "choice" statement defines a set of alternatives, only one of which may exist at any one time. A choice node does not exist in the data tree.

A choice consists of a number of branches, defined with the "case" substatement. Each branch contains a number of child nodes. The nodes from at most one of the choice's branches exist at the same time.

1.3.2. Data Modeling Arguments

The following arguments are used:

string: A string.

duration: A duration, as specified in ISO 8601 [REF].

empty: An empty type.

integer: An integer.

2. KASP Contents

2.1. Preamble

All policies MUST be enclosed in a KASP container.

```
container KASP {  
    list Policy { ... }  
}
```

A KASP container MUST contain a sequence of policy entries and MUST NOT contain any other modeling types.

2.1.1. Policies

Each policy MUST have a "name" leaf which contains the name of the policy. The name is used to link a policy and the zones signed using it; each policy MUST have a unique name. A policy named "default" MAY be used to associate with all zones that do not have a policy explicitly configured. A policy SHOULD have a description associated with it. Furthermore, a policy MUST have the containers Signatures, Denial, Keys, Zone and Parent. These containers are described in the forthcoming sections.

```
list Policy {  
    key "name";  
    leaf name {  
        mandatory true;  
        type string;  
    }  
    leaf description {  
        type string;  
    }  
    container Signatures { ... }  
    container Denial { ... }  
    container Keys { ... }  
    container Zone { ... }  
    container Parent { ... }  
}
```

2.1.1.1. Signatures

A Signatures container defines the policy parameters for creating RRSIG records and MUST be included. It MUST contain the following leaf nodes: Resign, Refresh, InceptionOffset. It MAY have a leaf node Jitter. It MUST contain a Validity container that includes leaf nodes for the validity periods of certain type of resource record sets (RRsets). The Default leaf node sets the validity period for all RRsets that do not have a specific leaf node in this Validity container. The Denial leaf node sets the validity period for all NSEC and NSEC3 RRsets.

The Validity container MUST include leaf nodes Default and Denial and MAY include other leaf nodes to differentiate between even more types of RRsets.

```
container Signatures {
  leaf Resign {
    mandatory true;
    type duration;
  }
  leaf Refresh {
    mandatory true;
    type duration;
  }
  leaf Jitter {
    type duration;
  }
  leaf InceptionOffset {
    mandatory true;
    type duration;
  }
  container Validity {
    leaf Default {
      mandatory true;
      type duration;
    }
    leaf Denial {
      mandatory true;
      type duration;
    }
  }
}
```

Here:

1. Resign - the re-sign interval, which is the interval when the signer MUST re-sign the zone.
2. Refresh - the refresh interval, detailing when a signature MUST be refreshed. As signatures are typically valid for much longer than the interval between runs of the signer, there is no need to re-generate the signatures each time the signer runs. The signature MUST be refreshed when the time until the signature expiration is closer than the refresh interval or when the data has been changed. A value of zero (PT0S) MUST be interpreted as to refresh the signatures each re-sign interval.
3. Jitter - If present, the value added to the expiration time of signatures to ensure that not all signatures expire at the same

time. The actual value of Jitter to be added MUST be $-j + (r \% 2j)$, where j is the jitter value from the policy and r a random duration, uniformly ranging between $-j$ and j , is added to signature validity period to get the signature expiration time.

4. InceptionOffset - a duration that MUST be subtracted from the time at which a record is signed to give the start time of the record. This is required to allow for clock skew between the signing system and the system on which the signature is checked. Without it, the possibility exists that the checking system could retrieve a signature whose start time is later than the current time.
5. Validity - groups two or more elements of information related to how long the signatures are valid for - Denial is the validity period for all NSEC and NSEC3 RRsets, Default is the validity period for all other RRset.

[MM: Add MaxZoneTTL]

The relationship between these elements is shown [RFC6781], Figure 11.

2.1.1.2. Authenticated Denial of Existence

Authenticated denial of existence information is included within a Denial container. It MUST contain either an empty leaf node NSEC or a container NSEC3. A NSEC3 container MUST include a leaf node for TTL and Resalt and MUST include a container Hash. Additionally, it MAY include an OptOut leaf node.

The Hash container MUST include three leaf nodes: Algorithm, Iterations and SaltLength.

```
container Denial {
  choice RRtype {
    case plain {
      leaf NSEC {
        mandatory true;
        type empty;
      }
    }
    case hashed {
      container NSEC3 {
        leaf TTL {
          mandatory true;
          type duration;
        }
        leaf OptOut {
          type empty;
        }
        leaf Resalt {
          mandatory true;
          type duration;
        }
      }
      container Hash {
        leaf Algorithm {
          mandatory true;
          type integer;
        }
        leaf Iterations {
          mandatory true;
          type integer;
        }
        leaf SaltLength {
          mandatory true;
          type integer;
        }
      }
    }
  }
}
```

If NSEC is used, zones with this policy MUST include NSEC records when signing the zone. If NSEC3 is used, zones with this policy MUST include NSEC3 and NSEC3PARAM records with the appropriate policy values:

1. TTL - if present, the TTL for the NSEC3PARAM resource record MUST be set to this value. If not present, PT0S (0) SHOULD be used as TTL.

2. OptOut - if present, all included NSEC3 records SHOULD set the Opt-Out bit. The signer SHOULD NOT include NSEC3 records for insecure delegations.
3. Resalt - A new salt value MUST be generated each Resalt interval value.
4. Algorithm, Iterations, SaltLength MUST be used as the parameters to the hash algorithm.

The choice and case modeling types are not included in the actual data tree. In the case that NSEC is used, the XML example would be:

```
<Denial>
  <NSEC/>
</Denial>
```

2.1.1.3. Keys

Parameters relating to keys can be found in Keys container. This container MUST include leaf node for TTL, and MAY include leaf nodes for PublishSafety, RetireSafety, ShareKeys and Purge. Furthermore, it MAY include a KSK list, a ZSK list and/or a CSK list. The latter indicates that a Single-Type Signing Scheme is used. If no such lists are included in the policy, the zone remains unsigned.

```
container Keys {
  leaf TTL {
    mandatory true;
    type duration;
  }
  leaf PublishSafety {
    type duration;
  }
  leaf RetireSafety {
    type duration;
  }
  leaf ShareKeys {
    type empty;
  }
  leaf Purge {
    type duration;
  }
  list KSK { ... }
  list ZSK { ... }
  list CSK { ... }
}
```


The leaf nodes specify the following behavior:

1. TTL - This value MUST be used as the TTL of the DNSKEY resource records.
2. PublishSafety - If present, the publish safety margin is used to give some extra time to cover unforeseen events and MUST be added to the publish interval in key timing equations.
3. RetireSafety - If present and similar to PublishSafety, the retire safety margin MUST be added to the retire interval in key timing equations.
4. ShareKeys - if multiple zones are associated with the same policy, the presence of a ShareKeys node indicates that a physical key can be shared between zones.
5. Purge - if present, keys in the dead state (as defined in key-timing-bis) will be automatically purged from the database after this interval.

A CSK, KSK or ZSK list MUST include leaf nodes for Algorithm, Length, Lifetime and Repository. Additionally, they MAY include a RollType leaf, Standby leaf and ManualRollover leaf. A CSK or KSK list MAY also include a RFC5011 leaf.

```
list CSK {  
  leaf Algorithm {  
    mandatory true;  
    type integer;  
  }  
  leaf Length {  
    mandatory true;  
    type integer;  
  }  
  leaf Lifetime {  
    mandatory true;  
    type duration;  
  }  
  leaf Repository {  
    mandatory true;  
    type string;  
  }  
  leaf RollType {  
    type string;  
  }  
  leaf Standby {  
    type empty;  
  }  
}
```

```
    }
    leaf ManualRollover {
        type empty;
    }
    leaf RFC5011 {
        type empty;
    }
}

list KSK {
    leaf Algorithm {
        mandatory true;
        type integer;
    }
    leaf Length {
        mandatory true;
        type integer;
    }
    leaf Lifetime {
        mandatory true;
        type duration;
    }
    leaf Repository {
        mandatory true;
        type string;
    }
    leaf RollType {
        type string;
    }
    leaf Standby {
        type empty;
    }
    leaf ManualRollover {
        type empty;
    }
    leaf RFC5011 {
        type empty;
    }
}

list ZSK {
    leaf Algorithm {
        mandatory true;
        type integer;
    }
    leaf Length {
        mandatory true;
        type integer;
    }
}
```

```
    }  
    leaf Lifetime {  
        mandatory true;  
        type duration;  
    }  
    leaf Repository {  
        mandatory true;  
        type string;  
    }  
    leaf RollType {  
        type string;  
    }  
    leaf Standby {  
        type empty;  
    }  
    leaf ManualRollover {  
        type empty;  
    }  
}
```

where:

1. Algorithm - this algorithm MUST be used.
2. Length - this key length MUST be used.
3. Lifetime - determines how long the key SHOULD be used for before it is rolled.
4. Repository - determines the location of the physical keys. The corresponding type of keys MUST be stored in this repository.
5. Standby - if present, determines the number of standby keys MUST be held in the zone.
6. ManualRollover - if present, this indicates that the key rollover MUST NOT occur automatically, it may only be initiated by the operator.
7. RFC5011 - if present, this indicates that the key rollover MUST include additional time for key publication and retirement and MUST revoke the predecessor key accordingly.

2.1.1.4. Zone

The Zone container encloses general information concerning the zone. It MAY include a PropagationDelay leaf and MUST have a SOA container.

The SOA container MUST include a TTL leaf, Minimum leaf and Serial leaf.

```
container Zone {
  leaf PropagationDelay {
    type duration;
  }
  container SOA {
    leaf TTL {
      mandatory true;
      type duration;
    }
    leaf Minimum {
      mandatory true;
      type duration;
    }
    leaf Serial {
      mandatory true;
      type string;
    }
  }
}
```

The PropagationDelay leaf holds the amount of time needed for information changes at the master server for the zone to work its way through to all the secondary servers. The value MAY be used in equations related to key timings during a rollover. If PropagationDelay is not used, a signer MUST use different heuristics to make sure key timings during a rollover are correct, for example by querying the name servers for the required records to exist.

The SOA container gives values of parameters for the SOA record in the signed zone. These values will override values set for the SOA record in the unsigned zone file:

1. TTL - The TTL of the SOA record MUST be set to the value of the TTL leaf.
2. Minimum - The MINIMUM RDATA field of the SOA record MUST be set to the value of the Minimum leaf.
3. Serial - The format of the serial number in the signed zone. This is one of: "counter", datecounter, unixtime, keep.

When Serial is set to "counter", the SOA serial MUST be incremented by one every re-sign.

When Serial is set to "datecounter", the SOA serial MUST be set to YYYYMMDDCC, where YYYYMMDD represents the current date and CC the number of new re-signs that day. If there are more than 100 re-signs a day, the date MUST rollover to the day after and count is reset to 00.

When Serial is "unixtime", the SOA serial MUST be set to the seconds since the epoch (1970-01-01 UTC).

When Serial is "keep" the SOA serial MUST be set to the SERIAL RDATA field of the SOA record in the unsigned zone. If this does not increment the last used serial, a signer MUST NOT produce a signed zone.

2.1.1.5. Parent

If a DNSSEC zone is in a chain of trust, digest information about the KSKs used in the zone will be published in DS records in the parent zone. To properly roll keys, timing information about the parent zone must be configured in the Parent container. A Parent container MAY include a PropagationDelay leaf, a DS container and a SOA container. The DS container MAY include a TTL leaf and the SOA container MAY include TTL and Minimum leafs.

```
container Parent {
  leaf PropagationDelay {
    type duration;
  }
  container DS {
    leaf TTL {
      type duration;
    }
  }
  container SOA {
    leaf TTL {
      type duration;
    }
    leaf Minimum {
      type duration;
    }
  }
}
```

Here, the PropagationDelay leaf configures how long it takes to get a DS record published in the parent zone after submitting the DNSKEY or DS record to the parent zone manager. The value SHOULD be used in equations related to key timings during a rollover. If PropagationDelay is not used, a signer MUST use different heuristics

to make sure key timings during a rollover are correct, for example by querying the name servers for the required records to exist.

The DS and SOA containers give values of parameters for the DS record and SOA record in the parent zone. These values SHOULD be used in equations related to key timings during a rollover. If these values are not used, a signer MUST query the parent name servers in order to retrieve the correct values.

A before:

1. TTL - The TTL of the DS and SOA records MUST be set to the value of the corresponding TTL leaf.
2. Minimum - The MINIMUM RDATA field of the SOA record MUST be set to the value of the Minimum leaf.

3. References

3.1. Informative References

- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.
- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.
- [RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, March 2008.
- [RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, December 2012.

3.2. Normative References

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

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

Appendix A. Changelog

- o Initial version

Authors' Addresses

Jerry Lundstroem
.SE
Ringvaegen 100
Box 7399
Stockholm 103 91
SE

EMail: jerry.lundstrom@iis.se
URI: <http://www.iis.se/>

W. (Matthijs) Mekking
NLnet Labs
Science Park 400
Amsterdam 1098 XH
NL

EMail: matthijs@nlnetlabs.nl
URI: <http://www.nlnetlabs.nl/>

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2015

W. Kumari, Ed.
Google
P. Hoffman, Ed.
VPN Consortium
July 3, 2014

Securely Distributing the DNS Root
draft-wkumari-dnsop-dist-root-01

Abstract

This document recommends that recursive DNS resolvers get copies of the root zone, validate it using DNSSEC, populate their caches with the information, and also give negative responses from the validated zone.

[[Note: This document is largely a discussion starting point.]]

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 4, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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. Introduction	2
1.1. Requirements notation	3
2. Requirements	3
3. Open Question: How Should the Root Zone Be Distributed? . . .	5
4. Open Question: Should Responses Have the AA Bit Set?	5
5. Pros and Cons of this Technique	6
5.1. Pros	6
5.2. Cons	6
6. IANA Considerations	7
7. Security Considerations	7
8. Acknowledgements	7
9. Contributors	7
10. Normative References	8
Authors' Addresses	8

1. Introduction

One of the main advantages of a DNSSEC-signed root zone is that it doesn't matter where you get the data from, as long as you validate the contents of the zone using DNSSEC information. From that point on, you know all of the contents of the root zone at the time that you retrieve and validated the zone.

When a typical recursive resolver starts up, it has an empty cache, the addresses of the root servers. As it begins answering queries, it populates its cache by making a number of queries to the set of root servers, and caching the results. All queries for root zone names that come to the recursive resolver that are not in either its positive or negative cache are sent to one of the root servers. This process cause a large number of the queries that hit the root are so called "junk" queries, such as queries for second-level domains in non-existent TLDs.

This document is describes a mechanism to populate caches in recursive resolvers with the verified contents of the full root zone so that the recursive resolvers have the all of root zone content cached. This technique can be viewed as pre-populating a resolver's cache with the root zone information by retrieving a signed copy of the root zone and verifying the contents.

The two goals of this mechanism are to provide faster negative responses to stub resolver queries that contain junk queries, and to reduce the number of junk queries sent to the root servers. The

mechanism has other minor advantages, but those two are the focus of this document.

1.1. Requirements notation

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

2. Requirements

In the discussion below, the term "legacy operation" means the way that a recursive resolver acts when it is not using the mechanism describe in this document, namely as a normal validating recursive resolver with no other special features.

In order to implement the mechanism described in this document, a recursive resolver MUST support DNSSEC, and MUST have an up-to-date copy of the DNS root key.

A recursive resolver using this mechanism MUST follow these steps at startup or after clearing its cache:

1. The resolver determines the list of root zone delivery servers. The delivery mechanism is not yet defined in this document, and some possible options for it are described in Section 3.
2. The resolver SHOULD randomly sort the list of zone delivery servers so that all the servers get a fairly even distribution of queries.
3. The resolver SHOULD attempt to transfer the signed root zone using the transfer protocol from each one of the servers until either success is achieved or the list has been exhausted. The resolver MAY attempt to transfer in parallel to minimize startup latency. If the root zone cannot be transferred, the resolver logs this as an error, and MUST fall back to legacy operation.
4. The resolver MUST validate the records in the zone using DNSSEC. If any of the records do not validate, the resolver MUST discard all records received, MUST log an error, and SHOULD try the next server in the list. If no transferred copy of the root zone can be validated, the resolver logs this as an error, and falls back to legacy operation. Note that the resolver MUST validate all of the zone contents, and MUST NOT start using the new contents until all have been validated; the resolver MUST NOT use "lazy validation". This means that the addition of the zone data MUST be an atomic operation.

The resolver MAY store the contents of the validated root zone to disk. If the resolver has a stored copy of the root zone, and the data in the zone is not expired, and that copy was written within the refresh time listed in the zone, the resolver MAY load that zone instead of transferring.

Once the resolver has transferred and validated the zone, it MUST attempt to keep its copy of the root zone up to date. This includes following the refresh, retry, expire logic, with certain modifications:

- o If the zone expires (for example, because it cannot retransfer because of blocked TCP connections), the resolver MUST fall back to legacy operation and MUST log an error. It MUST NOT return SERVFAIL to queries only due to its copy of the root zone being expired.
- o The resolver MUST validate the contents of the records in the zone using DNSSEC for every transfer. The resolver SHOULD try alternate servers if the validation fails. If the resolver is unable to transfer a copy of the zone that validates, it MUST treat this as an error, MUST discard the received records, and MUST fail back to legacy operation. Note that the resolver MUST validate all of the zone contents, and MUST NOT start using the new contents until all have been validated; the resolver MUST NOT use "lazy validation". This means that the replacement of the current zone data MUST be an atomic operation.
- o The resolver SHOULD attempt to restart this process at every retry interval for the root zone.
- o The resolver MUST set the AD bit on responses to queries for records in the root zone. This action is the same as if it had inserted the entry into its cache through a "normal" query that received a DNSSEC-validated answer.
- o The resolver MUST set the TTL on responses in the same fashion as it would in legacy operation. The difference here is that, when the TTL times out, instead of fetching the new answer from the root, the resolver simply starts the TTL at the maximum listed in the root zone.

Compliant nameservers software MUST include an option to securely cache the root zone (an example name for this option could be "transfer-and-validate-root [yes|no]"). That is, the mechanism described in this document MUST be optional, and the cache operator MUST be able to turn it off and on.

3. Open Question: How Should the Root Zone Be Distributed?

The signed root zone can be distributed over almost any protocol. Because the zone is signed, the distribution protocol does not need to be authenticated. Suggestions for the distribution mechanism include:

AXFR zone transfer within the DNS

HTTP, most likely with appropriately-tuned caching

FTP

[[Others...]]

Note that with any of these methods, the zone does not need to be transferred from the root servers themselves. Instead, a simple discovery mechanism can be built into the protocol that lets a recursive resolver discover where there are servers that will let it transfer the root zone.

4. Open Question: Should Responses Have the AA Bit Set?

A recursive resolver that has a securely validated copy of the root can be thought of in at least two ways: as a smarter cache, or as a pseudo-slave server for the root. This section discusses the ramifications of those two choices. In both scenarios, the resolver will send back NXDOMAIN responses for junk queries without sending queries to the root and the resolver will set the AD bit on the responses. However, the two scenarios differ in whether or not the responses have the AA bit set.

A smarter cache does not set the AA bit. The responses for any query for a name in the root or an NXDOMAIN that is being sent because the TLD is junk come back with the AD bit set but the AA bit not set, just as it would in legacy operation.

A pseudo-slave to the root sets the AA bit in response to any query for a name in the root or an NXDOMAIN that is being sent because the TLD is junk. The reason that this is called a pseudo-slave instead of a slave is that there is a general expectation that a slave has a relationship with the master that would cause the slave to be notified of changes in the master with a NOTIFY announcement; that is not the case here. It acts a slave because it knows exactly how the master would reply at the time that it retrieve the signed zone, but it is a pseudo-slave because the master has no way of alerting it of changes.

The advantage of a recursive resolver acting as a pseudo-slave is that other resolvers that demand authoritative answers can ask if for those. However, there are few scenarios in which those demanding resolvers exist. The disadvantage of a recursive resolver acting as a pseudo-slave is that there is no way to signal that it is a pseudo-slave and not a real slave. Thus, someone seeing the AA bit set might think that the resolver is a real slave. This opens the can of worms about trusting the settings of the AA and AD bits in responses.

5. Pros and Cons of this Technique

This is primarily a tracking / discussion section, and the text is kept even looser than in the rest of this doc. These are not ordered.

5.1. Pros

- o Junk queries / negative caching - Currently, a significant number of queries to the root servers are "junk" queries. Many of these queries are TLDs that do not (and may never) exist in the root. Another significant source of junk is queries where the negative TLD answer did not get cached because the queries are for second-level domains (a negative cache entry for "foo.example" will not cover a subsequent query for "bar.example").
- o DoS against the root service - By distributing the contents of the root to many recursive resolvers, the DoS protection for customers of the root servers is significantly increased. A DDoS may still be able to take down some recursive servers, but there is much more root service infrastructure to attack in order to be effective. Of course, there is still a zone distribution system that could be attacked (but it would need to be kept down for a much longer time to cause significant damage, and so far the root has stood up just fine to DDoS).
- o Small increase to privacy of requests - This also removes a place where attackers could collect information. Although query name minimization also achieves some of this, it does still leak the TLDs that people behind a resolver are querying for, which may in itself be a concern (for example someone in a homophobic country who is querying for a name in .gay).

5.2. Cons

- o Loss of agility in making root zone changes - Currently, if there is an error in the root zone (or someone needs to make an emergency change), a new root zone can be created, and the root server operators can be notified and start serving the new zone

quickly. Of course, this does not invalidate the bad information in (long TTL) cached answers. Notifying every recursive resolver is not feasible. Currently, an "oops" in the root zone will be cached for the TTL of the record by some percentage of servers. Using the technique described above, the information may be cached (by the same percentage of servers) for the refresh time + the TTL of the record

- o No central monitoring point - DNS operators lose the ability to monitor the root system. While there is work underway to implement better instrumentation of the root server system, this (potentially) removes the thing to monitor.
- o Increased complexity in nameserver software and their operations - Any proposal for recursive servers to copy and serve the root inherently means more code to write and execute. Note that many recursive resolvers are on inexpensive home routers that are rarely (if ever) updated.
- o Changes the nature and distribution of traffic hitting the root servers - If all the "good" recursive resolvers deploy root copying, then root servers end up servicing only "bad" recursive resolvers and attack traffic. The roots (could) become what AS112 is for RFC1918.

6. IANA Considerations

This document requires no action from the IANA.

7. Security Considerations

A resolver that uses this mechanism but does not do full DNSSEC validation on the data it uses can obviously cause serious security issues because it can be fooled into giving wrong answers.

[[More?]]

8. Acknowledgements

The editors fully acknowledge that this is not a new concept, and that we have chatted with many people about this. If we have spoken to you and your name is not listed below, let us know.

9. Contributors

The general concept in this document is not new; there have been discussions regarding recursive resolvers copying the root zone for

many years. The fact that the root zone is now signed with DNSSEC makes implementing some of these techniques more feasible.

The following is an unordered list of individuals have contributed text and / or significant discussions to this document.

Steve Crocker - Shinkuro

Jaap Akkerhuis - NLnet Labs

David Conrad - Virtualized, LLC.

Lars-Johan Liman - Netnod

Suzanne Woolf - Individual

Roy Arends - Nominet

Olaf Kolkman - NLnet Labs

Danny McPherson - Verisign

Joe Abley - Dyn

Jim Martin - ISC

Jared Mauch - NTT America

Rob Austien - Dragon Research Labs

Sam Weiler - Parsons

10. Normative References

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

Authors' Addresses

Warren Kumari (editor)
Google
1600 Amphitheatre Parkway
Mountain View, Ca 94043
US

Email: Warren@kumari.net

Paul Hoffman (editor)
VPN Consortium

Email: paul.hoffman@vpnc.org