The ISP Column
A monthly column on things Internet

June 2019
Geoff Huston

Looking for What's Not There

DNSSEC is often viewed as a solution looking for a problem. It seems only logical that there is some intrinsic value in being able to explicitly verify the veracity and currency of responses received from DNS queries, yet fleshing this proposition out with practical examples has proved challenging. The relatively slow uptake of DNSSEC-signed zones in the DNS indicates that the value proposition presented by DNSSEC is yet to be proved, and there are frequently cited examples such as the unsigned name www.google.com as evidence that even well-used domain names do not see the value in being DNSSEC-signed. So where might the value of DNSSEC lie?

Some hope has been invested in DANE, or domain keys in the DNS. The credentials used to support the validity of a TLS-offered domain name certificate could also be published in the DNS, so a client could use the DNS to validate the TLS credentials of the remote end of a connection, and DNSSEC would be used to validate the DNS response. This has failed to gain traction in the browser world, for a set of reasons relating to vulnerability of various forms of stripping attacks by a hostile man-in-the-middle, as well as some concerns about the widespread use of small RSA key sizes in DNSSEC and of course the observation that entire DANE space is vulnerable to a registrar re-delegation attack. There has been some uptake of DANE in the mail space as an anti-spam measure, but the broader objective of using DANE and DNSSEC as a way of reinforcing or even superseding the CA WebPKI structure appears to have come to a grinding halt.

Given that the majority of DNS queries arise from a need to map a domain name to an IP address as a precursor to making some form of network connection, then a signed DNS response would mean that it would be harder for an attacker to substitute a different address into a DNS response and mislead the end user. But while this could've been a serious concern in the past, the widespread adoption of secure transport services in the form of Transport Layer Security (TLS) is often cited as a counter-measure to this form of DNS misdirection.

A TLS connection is expecting a set of credentials that it can use to validate that the remote side of the connection has control of a private key that has been associated with the domain name, demonstrated through the issuance of a X.509 Domain Name Certificate by a trusted Certificate Authority. With TLS, not only does the attacker need to coerce the DNS to provide fake responses, but the attacker needs to coerce other parts of the Internet's infrastructure in order to generate a plausible, but ultimately fake, domain name certificate for this form of misdirection to succeed.

The experience with name substitution attacks points to the observation that it's far easier to mislead a domain name registrar and have the registrar redelegate the entire zone to name servers operated by the attacker. If the zone is DNSSEC-signed, then the same registrar attack can substitute DS records and any defensive aspects of DNSSEC are negated. Online CA can then be used to issue certificates to the redelegated name and the attack is successful, at least in the short term.

Does DNSSEC have any other immediate uses? Is there any other value today in signing a domain name?

One unexpected potential use case has arisen from the way in which the DNS communicates the non-existence of a domain name. If a DNS authoritative name server serves pre-signed zones, then it cannot sign what is not in the static zone file, so it cannot sign in advance the collection of NXDOMAIN responses for all possible names that are not defined in a DNS zone. Instead, DNSSEC has defined an alternative approach where DNSSEC is used to sign the 'gaps' between the names in a zone when the names are lexicographically ordered. Even with NSEC3 this is effective, as NSEC3 simply redefines the ordering of a zone labels to make simple zone enumeration slightly harder. A DNSSEC-signed NXDOMAIN response says far more than the non-existence of a single name. It asserts that a range of names does not exist, and the same response can be used for a query for any name that sits within the range. A DNSSEC-aware recursive resolver could cache these negative range responses and re-use them in response to queries for any name that falls within these ranges without passing a query to the zone's authoritative name server.

This negative caching process still sounds pretty esoteric. However, the value here lies in defending against random name attacks in the DNS. If an attacker can orchestrate a set of slave bots to each offer a low rate of DNS queries to randomly generate names within the targeted zone, then the recursive resolvers will pass the queries onward to the zone’s authoritative servers as a local cache miss. With enough bots in the attack the overall result can be an overwhelming load on the zone’s authoritative servers, as illustrated in the October 2016 attack on DYN's DNS infrastructure.

How can we defend ourselves against such random name DNS attacks? We seem to be completely unable to stop the creation of bot armies. We also seem to be completely unable to stop these bots running scripts that generate random DNS query names. But perhaps we can make recursive resolvers more capable here, and task them with generating the NXDOMAIN responses to these random names, instead of passing the query onto the authoritative servers. If the zone is DNSSEC-signed, and the recursive resolver performs DNSSEC validation, and is also performing NSEC caching, as described in RFC 8198, then the recursive resolvers will directly answer these random name queries from their cache if they can with the result that most of these non-existent name queries will not be passed on to the authoritative name servers.

This NSEC caching behaviour of recursive resolvers was analyzed in a lab configuration by CZNIC’s Petr Špaček, using a query replay tool to feed queries into recursive resolvers, comparing a NSEC caching configuration against a conventional NXDOMAIN cache. [https://indico.dns-oarc.net/event/28/contributions/509/attachments/479/786/DNS-OARC-28-presentation-RFC8198.pdf] His conclusion was that NSEC caching is particularly effective in mitigating the effects of a random name attack.

A number of major providers of DNS recursive resolver tools, including BIND, Unbound and KNOT perform NSEC caching, either as a configuration option or by default.

Measuring NSEC Caching

At APNIC Labs we have been spending some time looking at the state of DNS infrastructure in the DNS, and we thought it would be helpful to see whether DNSSEC and NSEC caching was making a difference in the DNS these days. We were hoping to answer the question: **How effective is NSEC caching today?**

This has been a challenging question to answer, as we are trying to measure what can't be directly seen. We need to look for queries for non-existing names that are not passed to a zone's authoritative name server. In other words, we are looking for absent queries about absent names!

Once again, we've turned to the online advertising measurement platform to perform this measurement. The platform distributes some 5-10M ads per day across the entire "eyeball" internet, and the ad contains embedded Javascript that directs the test subject to fetch a small set of URLs. By using DNS names that are served by authoritative name servers that we manage, we can observe the DNS interactions between the recursive resolvers used by the end client and the experiment's DNS servers. We can't directly observe the client's actions, nor can we see what's happening in the DNS between the client and their recursive resolvers, as our only vantage point is our servers at the other end of the connection.

In this case the NSEC caching test is relatively simple to describe. The script requests the client to fetch an object from a URL where the domain name is signed, but the name itself does not exist. Two seconds later the script requests the client to fetch a second object from a URL where the domain name also does not exist, but it sits within the negative rage that is covered by an NSEC record from the first request. We are looking for clients where we see a recursive resolver fetch the first name, but not the second.

Looking for a query that is not meant to happen is not easy, as there is also an element of partial execution of these measurement scripts, so non-existent queries can be readily confused with experimental noise. To help with the interpretation of results, we use a two-step DNS process, where a unique name resolves to a CNAME response (DNSSEC signed) that maps into the non-existent name, and the experiment uses two passes, where the first pass is intended to load a cache and the second is intended to use the cache. Figure 1 shows the DNS query sequence.

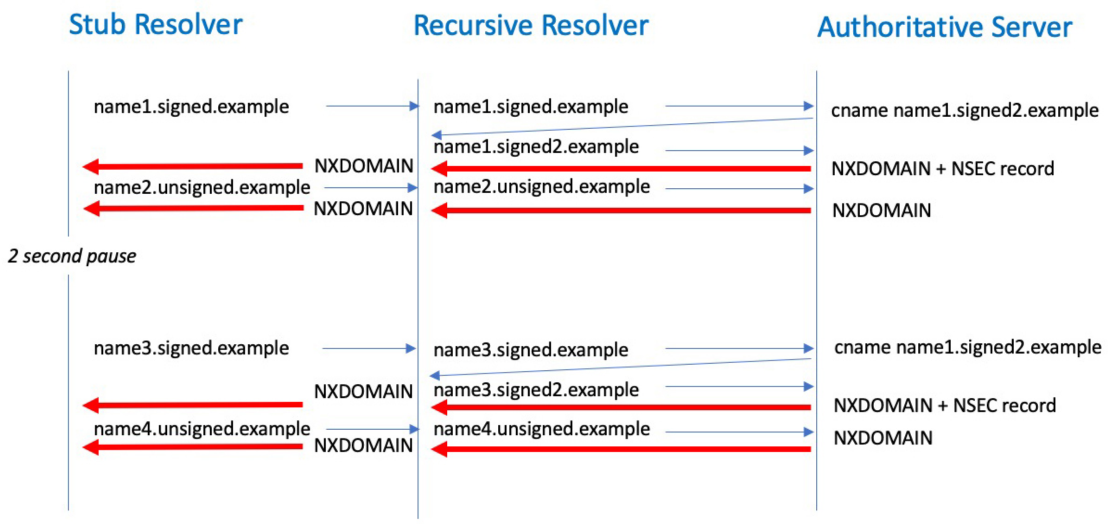

Figure 1 - NSEC Cache Measurement – DNS query model

The stub resolver running in the end client system is tasked to resolve two domain names, both using random labels. The first name is drawn from a DNSSEC-signed zone, while the second is not.

When the authoritative name server receives a query for the first name it will response with a CNAME record pointing to a name in a different signed zone. When this followup name is queried the authoritative server will respond with an NXDOMAIN code. If the query included the EDNS(0) DNSSEC OK flag (DO bit) then the authoritative server will also return an NSEC record that spans the name space of the followup zone.

The second name is unsigned, and always generates an NXDOMAIN response.

The experiment script will pause for 2 seconds and then repeat the two queries but use a slightly different query name. For the signed zone the CNAME response will use a name that sits within the span defined by the earlier NSEC record. If the recursive resolver is performing NSEC caching then the recursive resolver will generate a response based on this NSEC record and will not query the authoritative server. In the above example in Figure 1, the query for “name3.signed2.example” can be answered from the local NSEC cache.

We now have a test that in theory can identify where NSEC-caching recursive resolvers are being used. Such an NSEC-caching recursive resolver will only generate 5 queries to the authoritative name server, while non-security-aware recursive resolvers, and non-NSEC-caching resolvers will query for all 6 unique names.

Expectations

A surprisingly high 29% of the Internet's users use recursive resolvers that perform DNSSEC validation. Some 8% of users use a mixed environment where there are both DNSSEC-validating and non-valuing resolvers, such that when the DNSSEC-validating resolver returns SERVFAIL, indicating validation failure, the user’s stub resolver will re-query using a non-validating resolver. The other 21% of users exclusively use DNSSEC-validating recursive resolvers. (Figure 2)

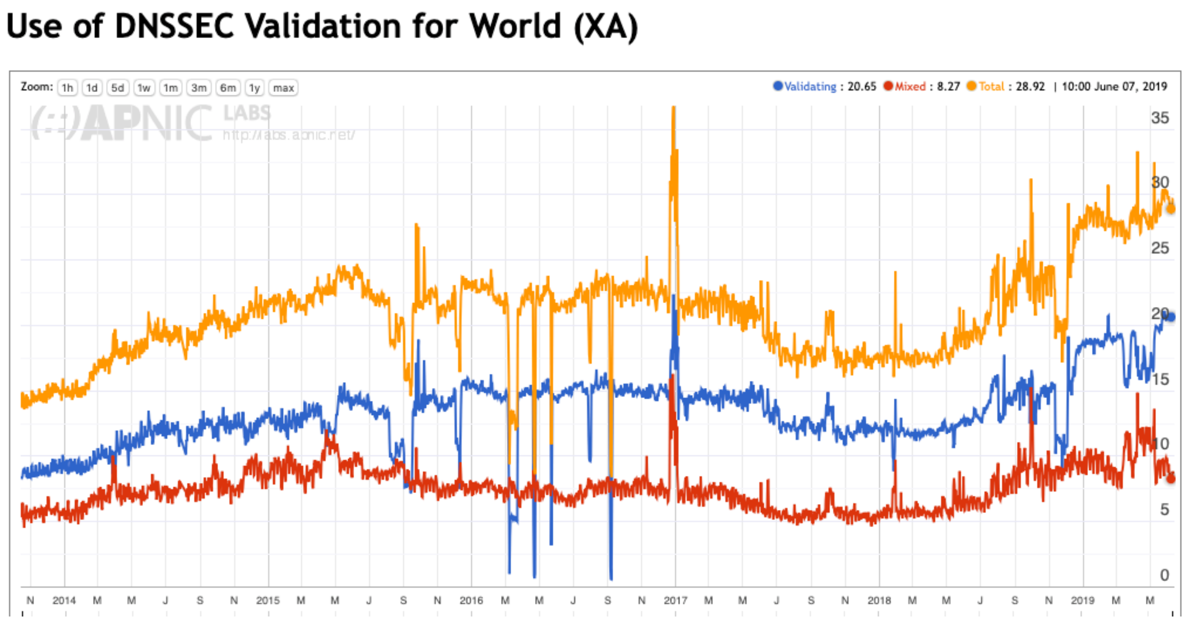

Figure 2 – Use of DNSSEC-validating recursive resolvers

The higher figure, 29% of users, is the number that interests us in this context, as the NXDOMAIN response should be accepted at face value, and the NSEC record has been delivered to the recursive resolver. In other words, even in partial use scenarios, the DNSSEC-validating resolver, if queried first, will generate a NXDOMAIN response, which will stop the user’s stub resolver from querying other resolvers.

Recursive resolvers are a highly skewed distribution, where a small collection of resolvers is used by a marge proposition of users. Some 12% of users pass their queries to Google's Public DNS servers as a first preference, and the 10 largest resolver sets serve one third of the Internet's users.

The upper bound on the use of NSEC caching should be the same as the use of DNSSEC validation, which is of the order of 29% of users. On the understanding that Google’s Public DNS resolvers perform NSEC caching, then the lower bound is around 10% or so. If we work on a rough estimate of 50% uptake of NSEC caching in DNSSEC-validating resolvers, then a reasonable expectation should be that some 15%- 20% of all users show a signal consistent with NSEC caching.

Results

The results of the experiment are shown in Figure 3.

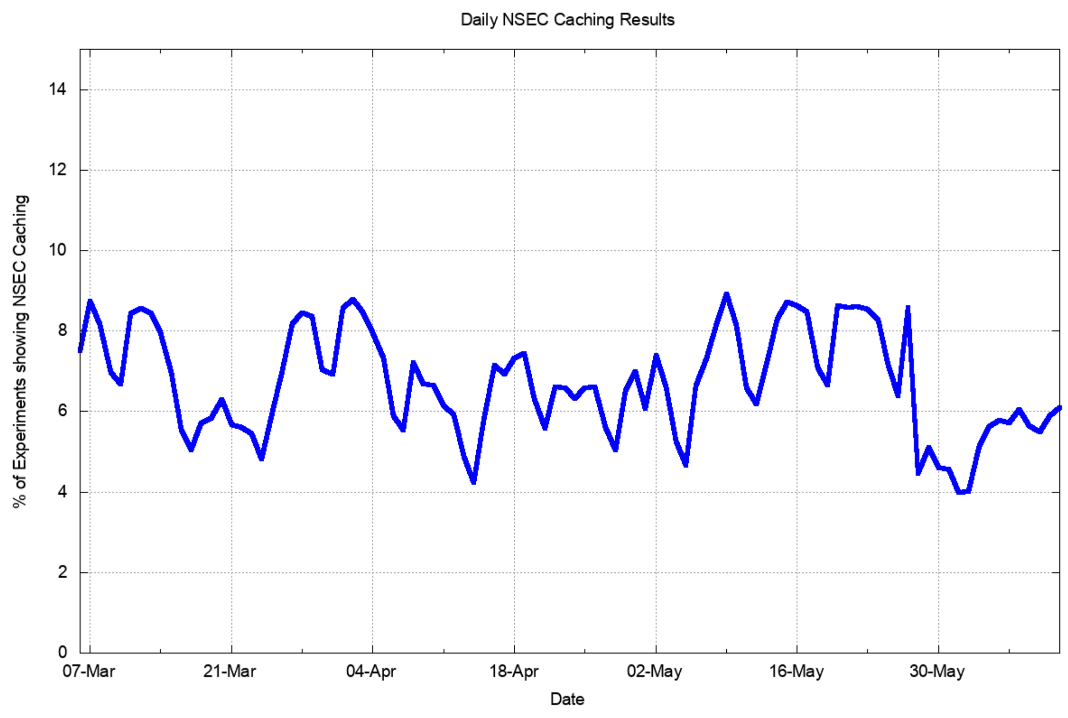

Figure 3 – Daily Percent of users who appear to use NSEC caching resolvers

Across the 98 days we ran the experiment some 266 million times. We observed in 6% - 9% of cases a query behaviour pattern that was consistent with NSEC caching being used, while the remainder of cases was pointed to no NSEC caching taking place. The figure is lower on weekends and higher on weekdays. When the daily experiment presentation volume was dropped at the end of May the NSEC caching rate also dropped.

Based in this data, it would be reasonable to conclude that the use of NSEC caching is small, and not growing, and our expectations were flawed in some fashion.

But this may not be the case, as in the DNS things are never quite what they seem.

Resolver Load Balancers and NSEC caching

The flaw in the model of the DNS that we are using is that the recursive DNS resolver is a single DNS engine. This is relatively uncommon these days, as the recursive resolver structure is often built as a 'farm' of DNS resolver engines

One any day these 3 million measurements pass their DNS queries towards some 100,000 recursive resolvers who are seen to ask authoritative servers. Over the 98-day period we observed 559,357 different recursive resolver IP addresses, which is to be expected from the long tail distribution of recursive resolvers in the Internet.

If we group these resolvers by their IPv4 or IPv6 subnet (set to /24 and /48 respectively) we see only 295,546 distinct resolver subnets. More than one half of the resolvers, or 347,302 resolvers by IP address, share a common subnet with one or more other resolvers, and it's likely that these are resolver engines that are members of a collective resolver farm. Many recursive resolver farms use a technique of distributed load balancing, where a number of these resolver engines lie behind a single client-facing service address. A scalable DNS recursive resolver system can be built in this manner with individual engines added to cope with peak demands on the recursive resolver service.

How do these farms distribute the query load? Round robin or other simple forms of query distribution can lead to poor system performance, particularly with respect to cache management, but at the same time can offer better load balancing properties under certain query patterns. More generally, each recursive resolve engine has to build up its own cache of names, and a sequence of queries for the same name may be distributed across multiple engines, negating the effectiveness of any single engine's cache if the distribution is based on relative current engine load across the engine farm

A common approach to query load distribution is to perform hashing of the query name, where queries for the same name are always sent to the same resolver engine. This way each resolver engine caches a consistent set of query names, and the collection of engines can be tuned to work very efficiently. Query name hashing works both for caching names that are 'defined' in the DNS to cache DNS responses on the same resolver engine every time and for negative names, where the non-existent name status is also cached on the same resolver engine.

What about NSEC caching and our NSEC experiment?

In this experiment we are using two separate query names: one query to load the recursive resolver with an NSEC response and the second query to supposedly query into the name span defined by the cached NSEC record from the first query, which is where the benefit of NSEC caching is intended to be found.

What happens if the two query names are hashed to different engines in a cluster of resolvers? Both queries then become cache misses and the authoritative server will see both queries, albeit from different resolvers. We would conclude from the queries seen at the authoritative server, possibly incorrectly, that no NSEC caching is taking place in these resolvers (Figure 4).

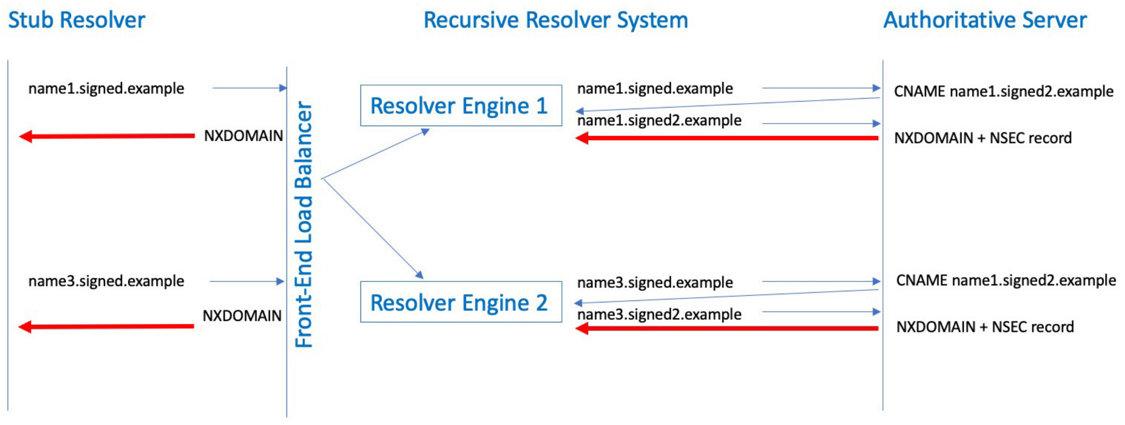

Figure 4 – Query Name Hashing load balancers and NSEC caching in resolver 'farms'

More generally, how do DNS load balancers and NSEC caching interact?

DNS questions of this form often have no clear black or white answer, and this question is no exception as there is no clear single answer. For the root zone, each resolver engine will assemble a complete set of entries for the zone from these NSEC responses very rapidly. But when the zone is used less intensively, these "cache-seeding" queries that have NSEC responses may be distributed across the entire set of engines in a resolver farm, and this may not result in any major leverage for NSEC caching. Our unexpectedly low result for observed NSEC caching from this experiment could be explained to some extent by the mismatch between query name hashing to load balance across resolver engines and NSEC range responses that describe a range of names with a relatively low query intensity.

In the normal course of queries for names lower down in the DNS name hierarchy NSEC caching may make no appreciable difference in the query load that is seen at the authoritative servers. Also, as the query volume falls the effectiveness of NSEC caching also falls. This explains the drop we saw in measured NSEC caching rates when the daily volume of experiments fell.

However, this sporadic low query volume scenario not the situation where NSEC caching is intended to provide significant benefit. If the zone's servers are experiencing a high volume random name attack then the larger query volume will interact with a query name hashing load balancer to ensure that all of a farms resolver engines will learn the entire contents of the zone, and at that point the recursive resolvers will be in a position to absorb the random name attack queries via NSEC caching.

Investigating attack scenario sounds like a fascinating further direction for this experiment, looking at query intensity levels and their impact on the caching effectiveness with resolver farms. But we are not in the business of using our measurement platform to launch a random name DNS attack, even if it is targeted on our own servers, so we are not in a position to measure how effective NSEC caching is under these more extreme conditions using the Internet. Perhaps this line of investigation is best left to a benchtop simulation.

Conclusion

Performing NSEC caching in recursive resolvers is probably better than not. If a recursive resolver is already performing DNSSEC validating, then there is no major difference to the recursive resolver to cache the NSEC-record range as compared to caching the query name. There does not appear to be any greater load imposed on the recursive resolver and it offer the potential to improve its cache performance.

But in isolation the "DNS Camel" argument comes into play. Yes, there may be marginal improvements in overall DNS performance, but at the same time it's one more software feature to manage, and the lifetime management cost of additional functionality into DNS resolution code has some significant fixed components for each feature. Is the marginal benefit worth the effort of implementation?

On the other hand, we really have little in the way of effective defence against DNS attacks based on random query name generation. Such an attack can be extremely simple to set up, and as we've seen with previous such attacks, with a large enough bot army a very unsophisticated attack can result in destructive havoc in the Internet's DNS infrastructure. In today's environment the DNS has become highly concentrates and a very small set of authoritative name servers service a large set of DNS names associated with mainstream services. An effective attack against just one of these service operators can be extremely effective, as we saw with the October 2016 attack against DYN. In such a scenario NSEC caching can be useful. If the benefit is one of countering a significant area of vulnerability in DNS infrastructure, then perhaps this is worth the incremental cost of yet another addition to the DNS resolver function.

Disclaimer

Author

Geoff Huston B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

www.potaroo.net