DNSOP Working Group
Internet-Draft
Updates: 1034, 1035 (if approved)
Intended status: Standards Track
Expires: August 27, 2019

D. Lawrence
Oracle
W. Kumari
P. Sood
Google

February 23, 2019

Serving Stale Data to Improve DNS Resiliency
draft-ietf-dnsop-serve-stale-03

Abstract

This draft defines a method for recursive resolvers to use stale DNS data to avoid outages when authoritative nameservers cannot be reached to refresh expired data. It updates the definition of TTL from [RFC1034], [RFC1035], and [RFC2181] to make it clear that data can be kept in the cache beyond the TTL expiry and used for responses when a refreshed answer is not readily available. One of the motivations for serve-stale is to make the DNS more resilient to DoS attacks, and thereby make them less attractive as an attack vector.

Ed note

Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication. This document is being collaborated on in GitHub at <https://github.com/vttale/serve-stale>. The most recent version of the document, open issues, etc should all be available here. The authors gratefully accept pull requests.

Copyright Notice

Table of Contents

1. Introduction

Traditionally the Time To Live (TTL) of a DNS resource record has been understood to represent the maximum number of seconds that a record can be used before it must be discarded, based on its description and usage in [RFC1035] and clarifications in [RFC2181].

This document proposes that the definition of the TTL be explicitly expanded to allow for expired data to be used in the exceptional circumstance that a recursive resolver is unable to refresh the information. It is predicated on the observation that authoritative

server unavailability can cause outages even when the underlying data those servers would return is typically unchanged.

We describe a method below for this use of stale data, balancing the competing needs of resiliency and freshness.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

3. Background

There are a number of reasons why an authoritative server may become unreachable, including Denial of Service (DoS) attacks, network issues, and so on. If the recursive server is unable to contact the authoritative servers for a query but still has relevant data that has aged past its TTL, that information can still be useful for generating an answer under the metaphorical assumption that "stale bread is better than no bread."

[RFC1035] Section 3.2.1 says that the TTL "specifies the time interval that the resource record may be cached before the source of the information should again be consulted", and Section 4.1.3 further says the TTL, "specifies the time interval (in seconds) that the resource record may be cached before it should be discarded."

A natural English interpretation of these remarks would seem to be clear enough that records past their TTL expiration must not be used. However, [RFC1035] predates the more rigorous terminology of [RFC2119] which softened the interpretation of "may" and "should".

[RFC2181] aimed to provide "the precise definition of the Time to Live", but in Section 8 was mostly concerned with the numeric range of values and the possibility that very large values should be capped. (It also has the curious suggestion that a value in the range 2147483648 to 4294967295 should be treated as zero.) It closes that section by noting, "The TTL specifies a maximum time to live, not a mandatory time to live." This is again not [RFC2119]-normative language, but does convey the natural language connotation that data becomes unusable past TTL expiry.

Several major recursive resolver operators currently use stale data for answers in some way, including Akamai (in three different resolver implementations), BIND, Knot, OpenDNS, and Unbound. Apple can also use stale data as part of the Happy Eyeballs algorithms in mDNSResponder. The collective operational experience is that it provides significant benefit with minimal downside.

4. Standards Action

The definition of TTL in [RFC1035] Sections 3.2.1 and 4.1.3 is amended to read:

TTL a 32-bit unsigned integer number of seconds that specifies the duration that the resource record MAY be cached before the source of the information MUST again be consulted. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached. Values SHOULD be capped on the orders of days to weeks, with a recommended cap of 604,800 seconds. If the authority for the data is unavailable when attempting to refresh, the record MAY be used as though it is unexpired.

Interpreting values which have the high order bit set as being positive, rather than 0, is a change from [RFC2181]. Suggesting a cap of seven days, rather than the 68 years allowed by [RFC2181], reflects the current practice of major modern DNS resolvers.

5. Example Method

There is conceivably more than one way a recursive resolver could responsibly implement this resiliency feature while still respecting the intent of the TTL as a signal for when data is to be refreshed.

In this example method four notable timers drive considerations for the use of stale data, as follows:

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

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

- A resolution recheck timer, which limits the frequency at which a failed lookup will be attempted.

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

Most recursive resolvers already have the query resolution timer, and effectively some kind of resolution recheck timer. The client response timer and maximum stale timer are new concepts for this mechanism.

When a request is received by the recursive resolver, it SHOULD start the client response timer. This timer is used to avoid client timeouts. It SHOULD be configurable, with a recommended value of 1.8 seconds as being just under a common timeout value of 2 seconds while still giving the resolver a fair shot at resolving the name.

The resolver then checks its cache for any unexpired data that satisfies the request and of course returns them if available. If it finds no relevant unexpired data and the Recursion Desired flag is not set in the request, it SHOULD immediately return the response without consulting the cache for expired records.

If iterative lookups will be, done then the resolution recheck timer is consulted. Attempts to refresh from the authorities are recommended to be done no more frequently than every 30 seconds. If this request was received within this period, the cache may be immediately consulted for stale data to satisfy the request.

Outside the period of the resolution recheck timer, the resolver SHOULD start the query resolution timer and begin the iterative resolution process. This timer bounds the work done by the resolver when contacting external authorities, and is commonly around 10 to 30 seconds.

If the answer has not been completely determined by the time the client response timer has elapsed, the resolver SHOULD then check its cache to see whether there is expired data that would satisfy the request. If so, it adds that data to the response message; it MUST set the TTL of each expired record in the message greater than 0, with 30 seconds recommended. The response is then sent to the client while the resolver continues its attempt to refresh the data.

When no authorities are able to be reached during a resolution attempt, the resolver SHOULD attempt to refresh the delegation.

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

days, which gives time for monitoring to notice the resolution problem and for human intervention to fix it.

6. Implementation Caveats

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

Stale data is used only when refreshing has failed, in order to adhere to the original intent of the design of the DNS and the behaviour expected by operators. If stale data were to always be used immediately and then a cache refresh attempted after the client response has been sent, the resolver would frequently be sending data that it would have had no trouble refreshing. As modern resolvers use techniques like pre-fetching and request coalescing for efficiency, it is not necessary that every client request needs to trigger a new lookup flow in the presence of stale data, but rather that a good-faith effort has been recently made to refresh the stale data before it is delivered to any client. The recommended period between attempting refreshes is 30 seconds.

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

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

6.1. Implementation Considerations

This document mainly describes the issues behind serving stale data and intentionally does not provide a formal algorithm. The concept is not overly complex, and the details are best left to resolver authors to implement in their codebases. The processing of serve-stale is a local operation, and consistent variables between deployments are not needed for interoperability. However, we would like to highlight the impact of various variables.

The most obvious of these is the maximum stale timer. If this variable is too large it could cause excessive cache memory usage, but if it is too small, the serve-stale technique becomes less effective, as the record may not be in the cache to be used if needed. Memory consumption could be mitigated by prioritizing removal of stale records over non-expired records during cache exhaustion. Implementations may also wish to consider whether to track the names in requests for their last time of use or their popularity, using that as an additional factor when considering cache eviction. A feature to manually flush only stale records could also be useful.

The client response timer is another variable which deserves consideration. If this value is too short, there exists the risk that stale answers may be used even when the authoritative server is actually reachable but slow; this may result in sub-optimal answers being returned. Conversely, waiting too long will negatively impact user experience.

The balance for the resolution recheck timer is responsiveness in detecting the renewed availability of authorities versus the extra resource use of resolution. If this variable is set too large, stale answers may continue to be returned even after the authoritative server is reachable. If this variable is too small, authoritative servers may be rapidly hit with a significant amount of traffic when they become reachable again.

Regarding the TTL to set on stale records in the response, historically TTLs of zero seconds have been problematic for some implementations, and negative values can't effectively be communicated to existing software. Other very short TTLs could lead to congestive collapse as TTL-respecting clients rapidly try to refresh. The recommended 30 seconds not only sidesteps those potential problems with no practical negative consequences, it also rate limits further queries from any client that honors the TTL, such as a forwarding resolver.

Apart from timers, one more implementation consideration is the use of stale nameserver addresses for lookups. This is mentioned explicitly because, in some resolvers, getting the addresses for nameservers is a separate path from a normal cache lookup. If authoritative server addresses are not able to be refreshed, resolution can possibly still be successful if the authoritative servers themselves are up. For instance, consider an attack on a toplevel domain that takes its nameservers offline; serve-stale resolvers that had expired glue addresses for subdomains within that TLD would still be able to resolve names within those subdomains, even those it had not previously looked up.

7. Implementation Status

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

The algorithm described in the Section 5 section was originally implemented as a patch to BIND 9.7.0. It has been in production on Akamai's production network since 2011, and effectively smoothed over transient failures and longer outages that would have resulted in major incidents. The patch was contributed to Internet Systems Consortium and the functionality is now available in BIND 9.12 via the options stale-answer-enable, stale-answer-ttl, and max-stale-ttl.

Unbound has a similar feature for serving stale answers, but will respond with stale data immediately if it has recently tried and failed to refresh the answer by pre-fetching.

Knot Resolver has a demo module here: https://knot-resolver.readthedocs.io/en/stable/modules.html#serve-stale

Details of Apple's implementation are not currently known.

In the research paper "When the Dike Breaks: Dissecting DNS Defenses During DDoS" [DikeBreaks], the authors detected some use of stale answers by resolvers when authorities came under attack. Their research results suggest that more widespread adoption of the technique would significantly improve resiliency for the large number of requests that fail or experience abnormally long resolution times during an attack.

8. EDNS Option

During the discussion of serve-stale in the IETF dnsop working group, it was suggested that an EDNS option should be available to either explicitly opt-in to getting data that is possibly stale, or at least

as a debugging tool to indicate when stale data has been used for a response.

The opt-in use case was rejected as the technique was meant to be immediately useful in improving DNS resiliency for all clients.

The reporting case was ultimately also rejected as working group participants determined that even the simpler version of a proposed option was still too much bother to implement for too little perceived value.

9. Security Considerations

The most obvious security issue is the increased likelihood of DNSSEC validation failures when using stale data because signatures could be returned outside their validity period. This would only be an issue if the authoritative servers are unreachable, the only time the techniques in this document are used, and thus does not introduce a new failure in place of what would have otherwise been success.

Additionally, bad actors have been known to use DNS caches to keep records alive even after their authorities have gone away. This potentially makes that easier, although without introducing a new risk.

In [CloudStrife] it was demonstrated how stale DNS data, namely hostnames pointing to addresses that are no longer in use by the owner of the name, can be used to co-opt security such as to get domain-validated certificates fraudulently issued to an attacker. While this RFC does not create a new vulnerability in this area, it does potentially enlarge the window in which such an attack could be made. An obvious mitigation is that not only should a certificate authority not use a resolver that has this feature enabled, it should probably not use a caching resolver at all and instead fully look up each name freshly from the root.

10. Privacy Considerations

This document does not add any practical new privacy issues.

11. NAT Considerations

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

12. IANA Considerations

There are no IANA considerations.

13. Acknowledgements

The authors wish to thank Robert Edmonds, Tony Finch, Bob Harold, Matti Klock, Jason Moreau, Giovane Moura, Jean Roy, Mukund Sivaraman, Davey Song, Paul Vixie, Ralf Weber and Paul Wouters for their review and feedback.

14. References

14.1. Normative References

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

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://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, <https://www.rfc-editor.org/info/rfc2119>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

14.2. Informative References

[CloudStrife]
Borgolte, K., Fiebig, T., Hao, S., Kruegel, C., and G. Vigna, "Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates", ACM 2018 Applied Networking Research Workshop, DOI 10.1145/3232755.3232859, July 2018, <https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_06A-4_Borgolte_paper.pdf>.

[DikeBreaks]
Moura, G., Heidemann, J., Mueller, M., Schmidt, R., and M. Davids, "When the Dike Breaks: Dissecting DNS Defenses During DDos", ACM 2018 Internet Measurement Conference, DOI 10.1145/3278532.3278534, October 2018, <https://www.isi.edu/~johnh/PAPERS/Moura18b.pdf>.

[RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <https://www.rfc-editor.org/info/rfc7719>.

Authors' Addresses

David C Lawrence
Oracle

Email: tale@dd.org

Warren "Ace" Kumari
Google
1600 Amphitheatre Parkway
Mountain View CA 94043
USA

Email: warren@kumari.net

Puneet Sood
Google

Email: puneets@google.com