November 2024
Geoff Huston

DNS OARC 43

The DNS Operations, Analysis, and Research Center (DNS-OARC) brings together DNS service operators, DNS software implementors, and researchers together to share concerns, information and learn together about the operation and evolution of the DNS. They meet between two to three times a year in a workshops format. The most recent workshop was held in Prague, October 2024, and all the presentations from the workshop can be found here. Here are my thoughts on some of the material that was presented and discussed at this workshop.

DNS as an Attack Platform

Akamai's Richard Meeus provided the somewhat disturbing statistic that some 65% of the DDOS (Distributed Denial of Service) attacks seen in Q3 of 2024 leveraged the DNS, either as DNS resource exhaustion attacks or as DNS reflection attacks. Neither of these techniques are novel attack vectors. The resource exhaustion attack attempts to exhaust the resources of either a DNS recursive resolver or an authoritative DNS server through a high volume of queries that take some time, or server resources, to formulate a response (such as CNAME chains, or missing glue record chains in referral responses), or the key trap scenario where DNSSEC validation consumers significant resources on the part of the validator. DNS reflection attacks are very much old school these days. They exploit the observation that in the DNS it's possible to generate a response that is significantly larger than the query. By using the IP address of the intended victim as the source address of attack traffic DNS servers can be coopted into sending high volumes of traffic to the victim.

The trends in such attacks show increasing packet intensity over longer periods of time with increasing frequency (Figure 1).

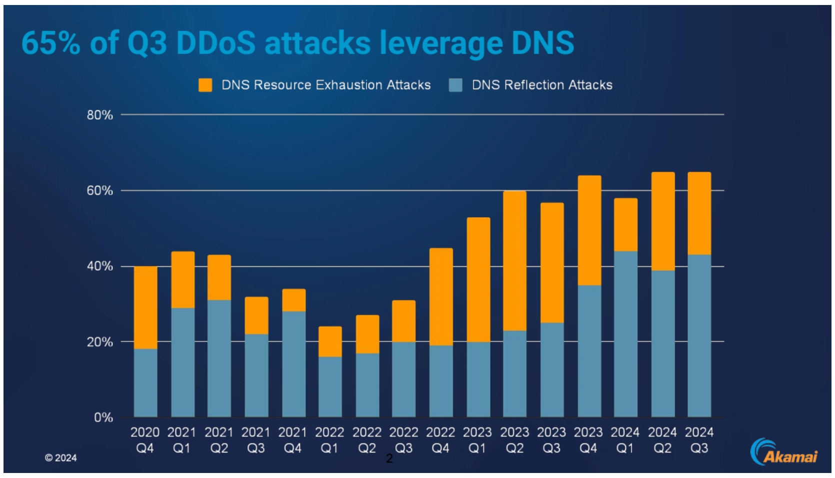


Figure 1 – Profile of recent DDOS Attacks, Richard Meeus, Akamai (OARC 43 Presentation)

There are no novel responses here. The old fallback is to build enough capacity in the server infrastructure such that is extremely challenging to generate a query load that will exhaust the server capacity. Of course this generates escalation pressure in the attack/defence process. Higher capacity defences motivate more intense attacks, which in turn motivates higher capacity defences, and so on. As the stakes escalate, effective defence becomes a more challenging proposition that only the largest platform operators can provide, and the side effect of this escalation is yet another factor that motivates further centralisation in the provision of DNS infrastructure.

Somewhat disturbingly, it appears that to some extent the promulgation of novel attacks can be attributed to academic security researchers whose supposed role is to defend against such attacks. It appears that their self-justification is that in order to improve security, then a deep understanding of the problem at hand is required and this level of understanding entails finding a way to weaponize it, so that it then becomes an attack. Their rationale appears to be that a simple description of threats and vulnerabilities is typically inadequate to generate a corrective response from vendors, and it is only when a method to exploit such a vulnerability and turn it into a major attack that vendors take it seriously!

Yes, there is an argument that security by obscurity is no defence, but the extent to which the discovery of vulnerabilities and their exploitation as disruptive attacks has become a path for funding of research activities is morphed this research activity into one that has crossed the line from defence into attack many years ago! We are no longer working on making a "safer" space, but diligently working on making far more insecure, and far more unsafe! These days, we appear to be our own worst enemy!

In a similar vein, there was a presentation of the *DNSBomb* attack, concentrating a low-rate query traffic into a high-rate response by pulsing these responses in time. *Key Trap* is an instance of a toxic zone configuration designed to force a recursive resolve into consuming resources to the extent that would impair its normal function and is again an instance of the same class of researcher-lead attack vectors. There is also *NRDelegation*, *CacheFlush*, *CAMP*, *NXNSAttack* and simple non-existent name queries. It seems to me that the blindingly obvious response for a resolver against all of these resource exhaustion attacks is to impose limits on the effort associated with resolving a name and stopping resolution once any such limit is exceeded. It is a characteristic of DNS transactions that the information-bearing response will inevitably be larger than a DNS query. Equally, it's a weakness of DNS UDP-based transactions that the responder is placed in the position of having to respond to a query where neither party authenticates each other.

Is the issue here a lack of fine-grained detail in the standard specification of DNS resolver and server implementations. Should we generate more RFCs to codify aspects of DNS behaviour that have been leveraged to form and attack vector? Or is the problem in the lack of clear specification in the content of the DNS database elements, in the zone configuration files? For example, how many name servers for a domain is too many? And does the answer to this question change if there are no glue records, forcing the resolver to resolve each of these name server names. Similarly, what if the name server name is served from a zone that itself does not contain glue records, and so on? How many stacked referral queries is too many, and should a resolver stop at a certain length of a chained referral sequence? Efforts to address these issues in standards-based specifications don't seem to be gaining sufficient traction.

Should we try to maintain a list of past and current attack vectors and a matching set of specifications for DNS resolvers and authoritative server, as well as specifications for the content of DNS zone files? This could represent a significant amount of effort (not to mention a continuous sequence of DNS-related RFCs to add to the few hundred RFCs that have already been published).

There is an underlying tension here between a resolver being diligent in attempting to resolve a name by exploring every available option until the name is resolved, against the resolver deliberately stopping its resolution process at some set of threshold points to defend itself against against resource exhaustion attacks. Do we try to codify all such possible attacks and potential responses? Or should we ask resolver implementations to simply exercise common sense? I suspect that the temptation to over-think this prescriptive style of approach is a constant issue, and increasing the volume of specification documents does not necessarily lead to more robust implementations.

I had thought that the 1987 advice from RFC 1035, section 7.1, was still perfectly adequate here:

> "The amount of work which a resolver will do in response to a client request must be limited to guard against errors in the database, such as circular CNAME references, and operational problems, such as network partition which prevents the resolver from accessing the name servers it needs. While local limits on the number of times a resolver will retransmit a particular query to a particular name server address are essential, the resolver should have a global per-request counter to limit work on a single request. The counter should be set to some initial value and decremented whenever the resolver performs any action (retransmission timeout, retransmission, etc.) If the counter passes zero, the request is terminated with a temporary error."
>
> RFC 1035, "Domain Names – Implementation and Specification", Paul Mockapetris, 1987

The root cause, as far as I can see, is that these strictures described RFC1035 are often harder to implement at scale than they might look, but it we want to get past chasing our tail with an ever-expanding set of specific responses intended to address specific behaviours, then the more generic advice of the need to resolver to perform self-protection needs to be given more serious implementation attention than it appears to have garnered so far.

This topic was a panel discussion at OARC 43, and the slides can be found here.

DNS Privacy and Anonymity

The DNS is placed in an invidious position with respect to privacy. If you consider that almost every network transaction with a remote service starts with a call to the DNS to resolve a service name into a set of service parameters, including the IP address of course, then a user's DNS data stream is in effect a log of the user's activity.

However, the DNS is so much more than this. The DNS is a ubiquitous signalling system and has been used for decades as a command-and-control channel for all kinds of distributed systems, many of which have turns out to be malicious in various ways. Looking at a DNS query stream can identify such instances of malware and the systems that have been compromised by such malware.

The industry does not have a consistent response to this. Some operators of DNS infrastructure operate on the principle that all DNS log data generated by user activity is strictly private, and the DNS query data is not just never divulged, its rapidly destroyed. Others operators have tried to provide some level of visibility of DNS query data, and they attempt to obscure who is performing the queries by performing some form of hashing of the client's IP address. Such obfuscation is not all that effective, and if my IP address always hashes to the same has code point then the collection of my queries in the query log are still identified as a distinct query set, which does not say much about the level of obscurity of such supposedly anonymised data.

As an open DNS resolver that is co-funded by the EU, the DNS4EU project has been caught up in this dilemma. They obviously wish to maintain the privacy of those users who elect to use this server as their DNS resolver, but at the same time provide a useful stream of available data to inform the efforts to bring to light aspects of large pool of totally toxic activity which is interweaved into DNS queries and responses.

The current efforts in DNS4EU, presented by Whalebone's Robert Šefr, are to look at more sophisticated forms of querier IP address anonymisation, mapping all IP addresses in a IPv6 pseudo address for use on generated logs.

Is this response to potential privacy concerns good enough? If it's the combination of *who is asking?* and *what question they are asking?*, then it seems to me that double layer obfuscation, and the cooperation of trusted network agents would be the minimum level that would be needed to preserve privacy. It's not just a case of post-processing the logs of DNS queries, but withholding this information from the DNS in the first place.

We appear to be working between extremes here. The earlier open trusting culture of the Internet was so comprehensive abused by all kinds of private and public entities that any residual user confidence on the fidelity and privacy of their transactions on the Internet was completely eroded. There is no going back to these days. But at the same time a blanket model of comprehensive obscurity, where all deeds, good or bad, go by unnoticed, has its own perils. If bad deeds can occur with impunity, then bad deeds will inevitably proliferate and overwhelm the entire online environment. I am not convinced that these measures used by DNS4EU to anonymise their data is the best answer we can ever come up with, but they have thought through these issues with care and attention, and it certainly appears to be a positive step in balancing these conflicting demands.

Presentation: DNS4EU for public anonymization

Authoritative Server Performance

Over the years much effort has gone into studying the performance of the root server system for the root of the DNS, yet the next level down, that of the performance of the nameservers that serve the top-level domains has received far less attention.

There are currently 1,445 top level domains in the root zone, and on average each TLD has 5.31 name servers. There are 4,656 unique IPv4 addresses and 4.353 unique IPv6 addresses.

What is the "right" number of name servers? There is no single answer here, and in looking at the TLDs opinions differ. 93 TLDs have 2 or 3 nameservers, and 173 TLDs have more than 6 nameservers (none have more than 13). At either extreme we see:

# IP addresses for nameservers	TLD
2	et kp pf sl
24	arpa
26	com edu in md mg mr mx net

A small collection of nameservers appears to offer little in the way of resilience, even if these name servers are configured using anycast service constellations, so the four TLDs with just 23 nameserver addresses seems to be operationally fragile. On the other hand, using 13 dual stack name servers, combined with anycast services would be heading to the other extreme. DNS clients are just not that patient to perform failover 25 times to arrive at an answer. The complete distribution of TLD nameservers by IP address is shown in Figure 2.

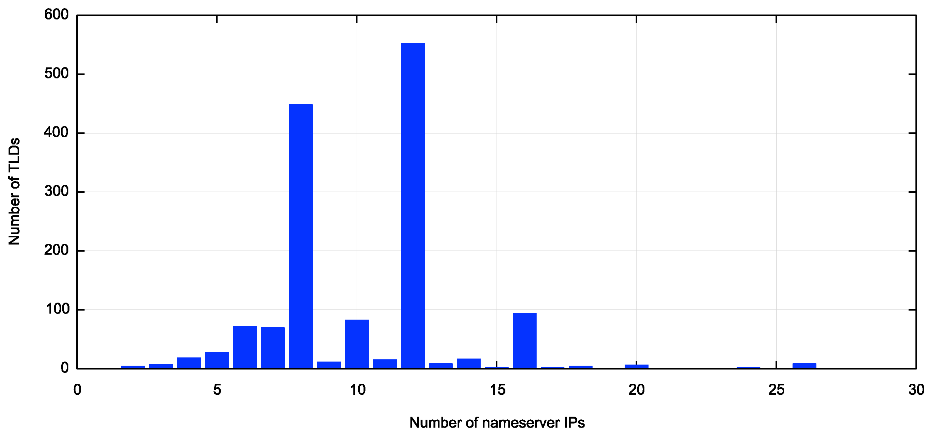


Figure 2 – Distribution of nameservers per TLD by IP Address

What can this tell us about the expectations of performance for serving TLDs?

If *performance* means *resilience*, then it appears that most TLDs use either 4 or 6 dual stack nameservers, and this would imply there is some provision for client failover in the case of one or two unresponsive nameservers.

However, it we equate *performance* to *responsiveness* then this is not so obvious. If all of these name servers were deployed on single server unicast platforms then the relative locations of these platforms in relation to the querier is the determining factor in the responsiveness of the service. One might assume that if TLD was served by multiple nameservers then if would query the nameserver that is the fastest to respond. If that were so, then using many servers that are distributed across the network would try and optimise the responsiveness for all queriers. However, the querier has no a priori knowledge as to which nameserver is the fastest to respond. The list of nameservers in a referral response is randomly ordered for each query, and the client will normally work though this list in a serial fashion until it gets an answer. Normally this would be the first server on the list, all other factors being equal.

There are no standards that govern this nameserver selection procedure. RFC 1034 suggests that resolvers should "find the best server to ask" but there is no further specification of what *best* means in this context now how such a selection is to take place. Recursive resolvers have made their own decisions here.

BIND9 selects the nameserver with the lowest statistical latency, through the maintenance of the Smoothed Round Trip Time (SRTT) to each nameserver. It updates a nameserver's SRTT after each query based on the latency experienced for the query, using an Exponentially Weighted Moving Average. Bind selects the nameserver with the lowest SRTT for query assignment. Those nameservers that have not yet been queried are assigned a random SRTT value ranging from 1 to 32 milliseconds, causing BIND to be biased to query all nameserver early, and then latch onto the nameserver that is fastest to respond. This algorithm treats all nameservers as IP addresses, so a dual-stacked nameserver counts as two distinct nameservers in this process.

PowerDNS uses a name based nameserver selection algorithm. Unqueried nameservers have an initial SRTT of zero, and the SRTT decays over time to ensure all nameservers are periodically queried to refresh their SRTT values.

Knot has a couple of unique features, in that it has a bias to use IPv6 addresses over IPv4, and the server selection algorithm will select the fastest nameserver in around 95% of cases but will randomly select any one of the nameservers in the other 5% of cases.

Unbound will assign untested nameservers an initial SRTT of 376ms. The selection of a nameserver is a random selection from all host nameservers that are no slower than 400ms more than the fastest nameserver.

What this means is that the observed performance of the nameservers for a TLD not only depends on the number and distribution of nameservers for that domain, but the resolver software that is being used to make the specific nameserver selection.

There are two further factors that should be considered here as well. The first is the use of anycast nameservers. Anycast systems leverage the routing system to make the selection of which instance of a nameserver is the "closest" to the resolver client. The resultant performance of the anycast service depends on the density of the anycast service constellation and the relative distance from the client to the closest instance of the anycast service. The numbers given in Figure 2 do not identify where servers are anycast, nor do they attempt to quantify the anycast constellation density. Secondly, the DNS makes extensive use of caching, and the longer the cache times the longer the zone's records can reside in a recursive resolver's local cache the better the average response times for clients of the recursive resolver. The intent of the DNS is, to the maximal extent possible for each domain, push the domain information into the recursive resolvers and serve the data from the resolver. Domains with a short TTL will trigger more frequent cache expiry and consequent refresh from the authoritative nameservers, at the cost of query performance.

All of which brings me to the presentation by Cloudflare's Wouter de Vries on authoritative nameserver performance of TLDs. This approach used Cloudflare's own anycast recursive resolver platform and looked at the response times per TLD and grouped these response times into percentiles on an hourly basis per TLD and per nameserver IP address.

Frankly, I'm not sure what to make of the results of this measurement exercise. Obviously TLDs with a small number of unicast nameservers had a poor average performance relative to a TLD that used a dense anycast constellation, and a TLD with a long TTL will produce better average response times than one with a short TTL due to caching.

A more thorough, but older (2017), study of the interaction between recursive resolvers and authoritative nameservers was published in the paper "Recursives in the Wild: Engineer Authoritative DNS Servers" and if you are interested in this topic then this paper is a good starting point.

Presentation: Authoritative Nameserver Performance of TLDs

AI and the DNS

These days its fashionable to include at least one AI presentation in a workshop program. This presentation compared the responses from Chat BGP and Meta AI on relatively simple DNS questions to the base DNS specifications. This included questions such as the possibility of CNAMEs at the zone apex (not allowed in the standards) and the label depth of wildcards in the DNS (unbounded). Unsurprisingly some generated answers correlated well with standard specifications, some did not.

Of course, the real issue here is that all of the answers, whether they were complete fabrications or not, all sounded totally plausible.

There are two kinds of responses to this observation. One response is "Don't be lazy!" Not everything a generative AI tool spits out is true, and you need to do the work yourself to validate these responses. Another response is to propose yet another AI tool that combines a large language model with DNS-specific RFCs, as proposed by Salesforce's Pallavi Aras. Personally, I'm pretty cynical about the latter option, but I'm also pretty clearly on the cynical side of AI ever getting beyond mindless pattern matching and demonstrating non-trivial inductive reasoning!

Presentation: Generative AI and the DNS

CNAMES in the Wild

There is a granularity mismatch in the DNS between the protocol model and operational practice. The assumption in the DNS was that entire zones would be delegated when you wanted to hand operational control over to another party. For example, if you wanted to pass www.example.com to a CDN then the DNS model is to make the www label a delegation point and use the CDN's nameservers for the delegated domain. Operational practice has simplified this using CNAME records so that individual labels in a zone can be passed over the CDN's control without delegating the entire name. In our example the label www would be a CNAME record to domain within the CDN. But what if the CDN operator wanted to change the target name withinb the CDN sphere? Getting the original domain administrator to make the change is operationally expensive. Its far easier to start "CNAME chaining" and make the original target of the CNAME a CDN entry point and then CNAME this name to the CDN service point, creating a CNAME chain of length 2.

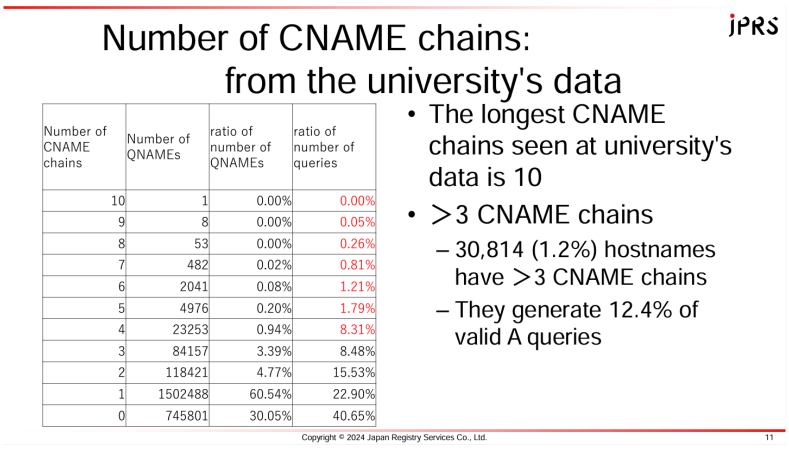
It was said in Computer Science that there were only 3 useful numbers: 0, 1 and infinity! The same applies here. If you can have CNAME chains of length 2, then why not of length 3, or 4, and so on. Kazunori Fujiwara's presentation on this topic at OARC 43 used the example of www.brother.in:

$ dig www.brother.in

www.brother.in.	600	IN	CNAME	mc-12265895-control-tm.trafficmanager.net.
mc-12265895-control-tm.trafficmanager.net.	30	IN	CNAME	mc-12265895-6e1d-4319-8534-7755-cdn-endpoint.azureedge.net.
mc-12265895-6e1d-4319-8534-7755-cdn-endpoint.azureedge.net.	1800	IN	CNAME	mc-12265895-6e1d-4319-8534-7755-cdn-endpoint.afd.azureedge.net.
mc-12265895-6e1d-4319-8534-7755-cdn-endpoint.afd.azureedge.net.	60	IN	CNAME	reserved-g01.afd.azureedge.net.
reserved-g01.afd.azureedge.net.	60	IN	CNAME	star-t-g.trafficmanager.net.
star-t-g.trafficmanager.net.	60	IN	CNAME	shed.dual-low.s-part-0003.t-0009.t-msedge.net.
shed.dual-low.s-part-0003.t-0009.t-msedge.net.	17	IN	CNAME	s-part-0003.t-0009.t-msedge.net.
s-part-0003.t-0009.t-msedge.net.	17	IN	A	13.107.246.31

That's a CNAME chain of length 7, and a lookup of this name entails a total of 78 DNS queries. What is also interesting is that the last two records have a relatively short TTL (of 17 seconds in this snapshot), and the other TTLs are variously 30, 60, 600 and 1,800 seconds. Long CNAME chains represent a hidden load on DNS resolvers where the long chains represent additional queries (and delay) to resolve names.

The extent of the use of CNAMES is shown in a data collection from two weeks at a Japanese university's recursive resolution service, spanning some 2.4M query names. The surprising result was that non-CNAME queries represent only some 41% of the query volume, while the remaining 59% comes from following CNAME chains (Figure 3)



Number of CNAME chains: from the university's data

Number of CNAME chains	Number of QNAMEs	ratio of number of QNAMEs	ratio of number of queries
10	1	0.00%	0.00%
9	8	0.00%	0.05%
8	53	0.00%	0.26%
7	482	0.02%	0.81%
6	2041	0.08%	1.21%
5	4976	0.20%	1.79%
4	23253	0.94%	8.31%
3	84157	3.39%	8.48%
2	118421	4.77%	15.53%
1	1502488	60.54%	22.90%
0	745801	30.05%	40.65%

- The longest CNAME chains seen at university's data is 10
- ＞3 CNAME chains
  - 30,814 (1.2%) hostnames have ＞3 CNAME chains
  - They generate 12.4% of valid A queries

Copyright © 2024 Japan Registry Services Co., Ltd. 11

Figure 3 – CNAME Chains in DNS DNS Data Capture (From "CNAMES in the Wild")

The other somewhat surprising observation was the incidence of long CNAME chains of length 4 or more. Services that use Microsoft, Amazon, Akamai and Apple's CDN platforms feature in this long CNAME set.

CNAME chains represent a compromise between operational flexibility and resolver load. This presentation by Kazunori Fujiwara of JPRS advocates a universal upper limit on the length of CNAME chains, rather than leaving it to individual resolver implementations.

Presentation: CNAMES in the Wild

Shuffling Nameservers

It is a common expectation that if you list multiple nameservers for a delegated domain then resolvers will somehow distribute their queries across all nameservers, and not concentrate upon the first server. Equally, if a nameserver name has multiple IP address, then the query load will be distributed across these IP addresses. However, this behaviour does not appear to be standardized.

The question posed by Shane Kerr of IBM's NS1 is: "Whose role is it to shuffle nameserver records?" When a nameserver delivers a referral response that lists the nameservers for a domain, or when it is explicitly queried for NS records, then the nameserver could perform a shuffle of the records in preparing the response to each query. It is also an option to perform this shuffling of nameservers in the resolver, shuffling the nameserver set each time the resolver needs to send a query to a nameserver.

The problem here is that if an implementation of an authoritative nameserver relies on resolvers to perform this shuffling and always lists the nameservers in the order as per the database, and if the resolver accepts the order as provided by the nameserver, then the query load will favour the first listed nameserver. A way to correct this is for all authoritative nameservers to shuffle the list of nameserver provided in referral responses as well as in response to a specific query for NS records.

Presentation: MUST Shuffle Resource Records

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

Author

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

www.potaroo.net