The ISP Column
A monthly column on things Internet

October 2019
Geoff Huston

Notes from OARC 31

DNS OARC held its 31st meeting in Austin, Texas on 31 October to 1 November. Here are some of my highlights from two full days of DNS presentations at this workshop.

Building a New Nameserver

There are two parts to DNS infrastructure. One is the infrastructure that supports resolving queries and the other is the infrastructure that serves the authoritative data that is used to answer these queries.

Facebook talked about their experience moving to a new DNS server platform. They have been a long-time user of tinydns. As any cursory search will reveal there are a variety of opinions about tinydns and its licensing, but it does appear to be a fast and efficient service. For many years, tinydns was able to meet Facebook's requirements, including DNS over IPv6, CIDR-based and ECS marking. However, simplicity comes with an implicit cost of extensions and with emerging functional requirements that exceeded tinydns' extension capabilities Facebook commenced a search across open-source servers. There are monolithic DNS servers in the open-source domain, including Bind, Knot, NSD and PowerDNS, but there is also the approach of plugins, where customised logic can be assembled across a substrate. The set of such module-based servers includes KnotDNS, PowerDNS and CoreDNS.

Facebook has found that CoreDNS is a good match for their requirements. There was a period of prototyping and testing and staging deployment that confirmed the viability of this selection. The result is a system that serves Facebook's query load with less memory and less processor impact, which has been in operation at Facebook for over a year.

DNS Transparency

There are a number of aspects of the DNS infrastructure that represent points of vulnerability. DNSSEC can certainly protect DNS responses such that a user who is prepared to validate a DNSSEC-signed response can assure themselves that the response is genuine and timely. But this is a qualified level of assurance.

If an attacker can convince a DNS registry that they are the subject of a name delegation then they can alter the delegation points, and potentially alter the DNSSEC record and assume control of the validation of the domain with their own key. It is then possible to use this altered DNS to get new web domain name certificates issued, given that a DNS-based proof-of-possession test can be met using this re-delegated DNS. At that point, TLS is pretty much hopelessly compromised and the hijack attack is complete for many services and sites.

It would be good to stop this form of attack, but the pragmatic observation is that weaknesses of the initial registrar-to-name-holder relationship is continuing and major vulnerability in the DNS ecosystem. A hopeful number registrars have adopted multi-factor authentication as a way of buttressing up this relationship. Registry Lock is also a good idea, so that attempts to alter the delegation state trigger notification and an imposed delay to ensure that the change has been authorised by the original name holder. However, such practices are not universally adopted, and the vulnerability remains present in far too many individual cases.

Similar problems exist in web security and web certificate issuers have adopted *certificate transparency*. All issued certificates are published in one of a small number of immutable public certificate stores. A certificate holder may constantly monitor the publication points of such certificate stores and if a certificate for their domain name is re-issued, they can take remedial steps. I'm unconvinced, and indeed more sceptical than not, about the efficacy of this approach. Such a measure does not stop bad actions in certificate issuance, nor does it necessarily make certificate issuance any more robust than it is today. It doesn't. Given the extremely fast set of actions by attackers, certificate transparency and the "well after the event" outcomes seem to be a response from the last century! It's hard to see *certificate transparency* as anything other than a rather meaningless charade.

This proposal (https://dnstransparency.org) advocates the same actions in the DNS for name delegations, where changes to delegations are logged in a public log. Such logs can be monitored, and domain name holders can use this to inform themselves after an attempt to perform a name hijack succeeds. Like *certificate transparency*, it does not prevent bad actors and bad actions, and as far as I can tell it looks more like a forensic tool than a prevention tool, but its proponents argue it has a role in today's DNS. I am, as you can probably guess from these comments, somewhat dubious about the value proposition of this proposal in the same way that I'b dubious about *certificate transparency*. Same-day or even same-week service in a world of incredibly fast attacks seems to represent an overwhelming mismatch between the capabilities of the attacker and the defensive capabilities of the soon-to-be victim!

DNSSEC Deployment in a Large Enterprise

How do you support a DNSSEC DNS server environment when you are a large hosting enterprise? *Large* in that there is a requirement to use multiple DNS providers for resilience, and *large* in the sense of supporting a DNS environment with up to a million changes per day and up to 700 changes per minute.

The DNSSEC model allows for both dynamic signing, where the RRSIG credentials are added to the response as the response is generated, and static signing, where the zone is signed in its entirety and the query generates a lookup and selects the relevant response. While the static model can be tuned to operate efficiently it is a tough ask for highly dynamic multi-provider zones. The presentation offered some paths through the various constraints and obstacles, but it underlined the old observation that the single challenge in the Internet is scaling and scaling in large dynamic DNSSEC-signed zones is absolutely no exception here. It's a formidable challenge.

DOH and DoT at Scale

The single DNS topic that dominates all other DNS conversations today is that of DNS channel encryption and in particular the use of TLS and HTTPS to carry DNS queries and responses.

It is part of venerable DNS mythology that UDP is the ideal transport protocol for the DNS. UDP does not impose a per-session state on the server and this, in turn, allows for DNS servers to handle a high query load when compared to the dimensions of the server platform. It was part of this mythology that TCP was definitely not the preferred DNS protocol, and only used as a last resort.

When we consider DNS over TLS (DoT) and DNS over HTTPS (DoH) there is both a TCP component, and also an encryption component. It was often asserted that the DNS server infrastructure was just not built to cope with the query load of all of today's query load was shifted onto DoT and/or DoH channels. But such assertions were often made in the absence of testing and measurement.

Comcast's DNS infrastructure handles the query load of tens of millions of customers. The DNS infrastructure serves some 600B queries per day and the peak load is some 10M queries per second. What is required to shift this load across to protected DNS query channels? In October 2019, Comcast started public tests of both DoH and DoT access to recursive DNS services, with an intention to bring this into production through 2020. They want to provide this with very high availability and no performance degradation.

Their first focus is DoH, using DoH-equipped frontends to existing DNS resolver infrastructure. The external facing interfaces provide HTTPS, while the connections back into the server infrastructure will use conventional DNS over UDP port 53. There are issues with split-horizon DNS, opt-in filtering services, localisation and accommodation of user preferences, and doubtless more issues will emerge. There are no clear answers here, but it's apparent that Comcast recognises that improving DNS privacy is a compelling story in today's retail services market and they are working hard to meet this demand and continue to provide infrastructure-based DNS services as part of their service offering.

Identifying the technical challenges of scaling the DNS infrastructure to match or exceed the current DNS capability with DoH and DoT services is the overall objective of this exercise.

DoH and DNS Preferences

The model of the Internet's namespace is that we have one. By that I mean we don't have 2, 3 or some other number. We have a single namespace. If I pass a DNS name to you then it acts as a consistent reference, in that the service or resource located at that named network point will remain the same for you as it is for me.

That's the theory in any case. In practice, this has not been the case for many years. There is the widespread practice of content filtering through DNS filters, where in certain parts of the world certain domain names won't resolve at all. Such DNS filtering measures exist in many countries for many reasons, not only for the protection of the exclusive rights of copyright holders, but also for reasons of societal norms, malware defence and similar. As well as removing DNS names, names have been added. A common ISP practice for a while was NXDOMAIN substitution where a query for a non-existent name would result in a redirection to a search service operated by the ISP or operated by an entity who paid the ISP for such invisible referrals. But there is still the concept of the DNS as infrastructure. Everyone in a particular domain or country or customers of the same ISP saw a largely similar Internet. At the same time, everyone leaked much the same information to the infrastructure provider of the DNS.

Various open DNS resolvers appear to provide some alternatives to users who prefer a different name resolution framework, such a cleanbrowsing or quad 9. They distinguish their service by adding (or removing) name filtering, or tailor responses based on assumed geolocation of the client, or they might apply different privacy frameworks or perform NXDOMAIN substitution. The concept was that it was the responsibility of the user to informed themselves of the policies and practices of the available open DNS resolvers and decide whether to configure them to the user's local configuration or not based on the match the user's requirements to those of the open resolver operator.

But what if the content wants to express a preference for the DNS resolver to choose for its content? (yes, I think this is a really odd proposition too!) In the wonderful world of HTML, there is an HSTS directive, where content can list the trusted Certification Authority (CA) that should be used to access this content. The value of this HSTS directive is entirely questionable in that it's a case of trust on first use, and an agile and capable attacker would strip the HSTS header from manipulated content (or even substitute its own). But despite these weaknesses in HSTS, this particular OARC presentation advocated a similar approach in DoH, where a DoH-Preference header would provide the URL of a DNS recursive resolver.

This seems to me to be completely bonkers! Why would content be allowed to alter the DNS resolution configuration of my browser application? Therein lies perhaps a plausible answer. Most users don't play with the configuration of their platform. Only a small set of users twiddle with the nerd knobs. So if you have built, at great expense no doubt, this wonderful open Internet-wide recursive resolver system, the problem still remains that no-one will use it. Some providers work on ISPs and come to an arrangement where the ISP passes all of its customers' queries to an open resolver, presumably for a financial consideration of some sort. Some come to an arrangement with an application vendor where in exchange for some support the application has a built-in preference to use an over-the-top DoH recursive resolver, and again in this world nothing is free so presumably, financial considerations would apply. But that's just not enough. What if content can provide steerage to an open DNS resolver?

It seems that the myriad of commercial interests out there in an unregulated Internet has unleashed a rather ruthless form of competitive interest where the end-user is the victim and the platform, the service providers, the infrastructure provider and now the content providers all are trying to find mechanisms to improve their leverage in a rather cynical game of increasing the level of ruthless exploitation of this benighted end user. In a previous age and a previous technology regime, the regulatory framework was perhaps the only way to protect consumers. These days no such impositions or regulations appear to hold sway, and we are seeing stranger and stranger outcomes in the battle over consumer exploitation.

This particular proposal is another rather sad and definitely depressing case of such aberrant industry actions in my opinion!

Authoritative DNS over TLS Operational Considerations

The DNS Privacy Working Group, is now entering the second phase of its operation, looking at the relationship between the recursive resolver and the authoritative server, and seeing how standard specifications could be drafted that would allow this connection to be secured. This proposal describes one submission to this work, proposing DoT between the recursive resolver and the authoritative. This is a mixed package in the light of the earlier DoT work. The selection of TCP port 853 for stub to recursive DoT is both criticised and applauded for much the same reason: it's a dedicated port. It's easy to stand up a dedicated server to listen on this port as all incoming traffic is assumed to be DNS traffic and there is no need for an application-level wrapper and application-level demuxing. However, it's also easy to identify and block all traffic on this port.

Treating DoT separately, and in consideration that this is a greenfield space at present allows the consideration of specifying TLS 1.3 as mandatory, and session resumption using 0-RTT. It is unlikely that an authoritative would exclusively listen on port 853, and then there is the consideration of how to configure the server environment. Should UDP and TCP port 53 coexist on the same infrastructure, or should an authoritative server look at service separation using distinct IP addresses? The latter requires service discovery handling for the recursive client to perform query steerage to the right server. There is little to be gained by pushing TLS all the way to the backend servers, and large servers would presumably use a TLS proxy server at the front end and pass queries into the server farms using conventional DNS over UDP port 53.

Connection management is an interesting aspect of this work. Studies of resolver centrality point to a centralised model of a heavily used core client set and a long tail of sparse clients. If this is the case then scaling a DoT authoritative server would present far fewer issues, allowing the overall majority of queries to be funnelled into the TLS sessions from a smaller number of recursive resolvers.

Authority Selection

I referred to DNS mythology earlier, and in the area of authoritative servers it comes up again. How many authoritative servers should be used? Is it ok to use just one authoritative server if it is anycast? Are more authoritative servers better? When do more servers become too many servers? What are the objectives that lie behind the use of multiple authoritative servers? Is it service resilience? Or optimal performance? Do recursive resolvers really pick the best authoritative server in any case?

The answers to these questions are found in the behaviour of the recursive resolvers. The folk at cz.nic have constructed a new tool, DNS Maze, to create a parameterised version of network characteristics that can allow recursive resolver implementations to be tested in a controlled reproducible environment.

They've used this tool to test the four open-source recursive resolvers, Unbound, PowerDNS, Bind and Knot in various scenarios.

When the authoritative servers are situated at progressive delays away from the recursive client. A Bind client will latch on to the lowest delay server pretty tightly (95% to the fastest server). and only periodically check the other servers. Knot will share its queries around more liberally (24% of queries to the fastest server). PowerDNS latches on every tighter than Bind (98%), while Unbound shows no obvious latching behaviour at all and shares its favours evenly over all servers irrespective of delay. Similar experiments using Atlas probes and the Internet show behaviours similar to the Maze outcomes, which is reassuring.

It appears that only some recursive resolvers will tend to concentrate their queries on the fastest authoritative server, namely Bind and PowerDNS while Knot and Unbound tend to increase the relative share the query set across all servers.

One wonders if an inspection of the code in each of these implementations would've predicted these outcomes more directly! I suspect so.

No Frag Flag Day!

Packet fragmentation has always been both an asset and a weakness of the Internet protocol. In its original sense of the Internet as a network of networks, there was always the problem of packet size mismatch. When a large packet had to be passed into a network that did not support such a large packet size the options are to drop the packet or adapt the packet to fit. IPv4 chose the second option, and routers can split up an IP packet into smaller fragments by replicating the IP packet header into each of the smaller fragments and writing fragmentation onto the IPv4 fragmentation control field to describe the fragment context. Fragments could also be further divided into smaller fragments. The task of reassembly of these fragments was left to the destination. For all its flexibility it was also a liability, both in performance and security. At the time of the design of IPv6, IP packet fragmentation had become unfashionable. The IPv4 "Don't Fragment" field was jammed to "on" and removed from the header and the fragmentation control fields were relocated into an extension header that sat between the IPv6 header and the transport control fields only if the source had fragmented the packet.

But despite the sentiment that fragmentation is to be avoided, fragmentation is still important. And it's important in a packet datagram protocol such as DNS over UDP. If there is a problem with fragmentation between a server and a client, then the packet is silently discarded. Of all the forms of packet damage, silent discard is the more troublesome. The client can only detect this condition by a timeout, and even when a timeout occurs it is not obvious where the issue may be.

Think about the plight of a DNSSEC-validating recursive resolver where a local firewall discards all IP fragmented packets. When it attempts to validate any signed name in `.org` the resolver will eventually ask for the DNSKEY record for `.org`. The DNS response is 1,625 octets and fragmentation is inevitable. As the local firewall is discarding all fragments what it will see is a timeout, so the resolver may conclude that the authoritative nameserver for `.org` is offline and try another. This may take a long time as there are 8 distinct name servers for this domain, and each has both IPv4 and IPv6 records. By the time that the resolver has concluded that all authoritative name servers being offline is just too much of a coincidence, way too much time has elapsed, and the query context may well have timed out.

How can we work around this? One solution I still like is ATR (written up as the IETF draft draft-song-atr-large-resp-03), but it appears that the DNS community does not share my personal preferences and they've opted for a more direct approach: don't send fragmented packets!

The proposed approach for this next DNS Flag Day is to set the default EDNS UDP Buffer Size to a very low value of 1,232. Consequently, larger UDP responses will be truncated and the client is expected to retry using TCP.

This buffer size of 1,232 bytes is a very conservative choice. QUIC, which has an even more critical intolerance to IP fragmentation uses a 1,340 MTU size, so one has to wonder why this particular number was chosen. Yes, it relates to the IPv6 minimum MTU of 1,280 bytes, and even this 1,280 number was simply plucked from thin air by the IPv6 design team back in the early days of IPv6 design. Oddly enough, the behaviour is largely similar to ATR, but slower by 1 RTT interval when the DNS response is between 1232 and 1432 octets and may be slower if the client sits behind a fragment-dropping network for larger packets.

If this is all about UDP and fragmentation, then this effort is open to the criticism that this appears to be a poorly thought-through response to a relatively small problem. On the other hand, if this is the first step in a Very Grand Plan to migrate the DNS from open UDP to secured TCP, then why not just say so and head there directly?

ECS and Evil

Extended DNS Client Subnet (ECS) had a rather difficult path through the IETF when it was brought there to generate a standard specification of the technology. The ECS specification, RFC7871, notes that "If we were just beginning to design this mechanism, and not documenting existing protocol, it is unlikely that we would have done things exactly this way...We recommend that the feature be turned off by default in all nameserver software." Not exactly a ringing endorsement.

Cloudflare's analysis of ECS in queries showed some interesting variation in behaviour. They compared the queries seen for two served zones, and while one zone saw 4% of queries with ECS, the other saw a rate of 32% of queries. Interestingly, the use of ECS is clustered around a small number of origin networks. Of the 42,338 seen origin ASNs in this measurement, 92 sent ECSD with IPv4 and 25 sent ECS with IPv6. ECS is not meant to be used for NS queries, yet they saw 0.05% in one zone and 0.5% in the other using ECS for NS queries. Where ECS was very prevalent (32%) the majority of these queries was for A RRS. In the other zone, where ECS was not observed at the same rate, the proportion of A queries was roughly equivalent to that for AAAA queries.

To avoid totally gratuitous information leakage ECS is intended to be associated with a Query Name Minimisation approach. However, they observed behaviour where ECS is applied more liberally than this, although they note that this is an infrequent case and most networks use ECS in a more careful manner that avoids more information leakage than is the case in a "minimal" ECS framework.

ECS is still woeful in terms of information leakage, and the use of DNS as a proxy for geolocation is undoubtedly a poor practice, but while some large scale CDNs use it to perform client steerage there is still a case to continue to support it. Part of the problem here is that most privacy frameworks use a model of advice and informed consent from the user, while ECS appears to be used in a far less overt manner. Most clients who are sitting behind ECS-marking resolvers are probably completely ignorant of this and the implications in terms of privacy control.

NSEC Caching Revisited

It has been challenging to make the case that DNSSEC is worth the additional effort. Some 30% of users may sit behind DNSSEC-validating resolvers, but far fewer DNS domains are DNSSEC signed. Whatever the case may be for DNSSEC signing, it seems that it’s just not enough in many if not most cases.

If we are looking for more benefits for DNSSEC signing a zone, then RFC8198 aggressive NSEC caching seemed to be exactly that. To allow offline signers, it was necessary to generate a signed response to authenticate non-existence without any prior knowledge of the non-existent queried name. The solution was to generate a range response that asserted that all names between two labels in an ordered view of the zone did not exist. Recursive resolvers could cache this information and use the cached NSEC response for any name that fell within the range.

Random surname attacks are a real problem for the DNS, and the various forms of query filtering to fend off such attacks is just one more episode of the cat and mouse game between attackers and defenders. NSEC caching offered a way to cut through this and allow the recursive resolver to answer such random names from its cache once the spanning NSEC record has been loaded.

All good in theory. What about in practice?

The result is not so good. The issue lies in the widespread use of DNS query distributors that farm queries out to a set of resolver engines. To operate efficiently it appears to be a common approach to hash the query name so that all queries for the same name are directed to the same resolver engine. This does not work as well in the case of non-existent names, where the hash function may pass subsequent queries to any of the resolver engines. In the simple low volume cases the level of NSEC caching is far less than what we would otherwise expect.

There are answers here, and the best I've heard is from PowerDNS's DNSdist, which places a cache into the DNS load distributor. That way the resolver engines are employed to fetch cache misses from a common front-end caching load distributor. As a DNS community, we appear to have avoided studying DNS load managers and we've not drafted a standard specification about desirable behaviours and the ways that can help the DNS to scale and fend off certain forms of attacks. Maybe it's time.

Root Name Queries

The queries received at the root name servers have been an area of study for many years. Root servers are critical to the operation of the DNS as they effectively load recursive resolvers with the primary mapping of the top-level domains to the associated top-level domain name servers. Every query starts with a notional call to a root server. This is termed 'notional' in that a recursive resolver will use its local cache whenever it can, for obvious reasons, so what the root server should see is in effect just the cache misses from recursive resolvers.

However, the profile of traffic directed to the root name servers appears to be at some variance to what a simple cache miss operating model might expect. An overwhelming volume of queries to the root server is for names that do not exist in the root zone. This study looked at the actual names used in these queries.

Chrome is a very dominant browser in today's Internet. When Chrome starts up it generates 3 nonsense queries to detect a paywall or DNS rewriting. The browser generates three random hostnames (dotless DNS labels) to detect if the local network is trapping and rewriting NXDOMAIN responses. The length of these text strings is between 7 and 15 characters, and the result is that the overwhelming majority of single label queries seen at the root is due to this particular Chrome behaviour.

Taking this approach and looking at names with an increasing number of labels yields some interesting outcomes. While single label queries dominate the query traffic, the next highest is 3, then 4 then 2-label queries. Most of these queries are uniformly distributed, except for queries of 8 and 9 labels, which show pronounced 'spikes' in the query volume. These appear to be linked to a single AS and potentially point to a single application.

Even longer query name lengths were seen, and the conclusion drawn is that a lot of these queries are simply the result of leakage from poor coding practices that result in corrupted memory, which in turn leads to DNS names that are formed into queries. But an observation that coding practices are less than perfect and bad code running on many devices generates major levels of network junk is not exactly a novel observation. Does the combination of cheapest possible code and device, coupled with massive volumes remind anyone of something? IoT perhaps? Expect more garbage.

How Big is that Zone?

The NSEC3 DNSSEC signing algorithm has prompted some interest from researchers into the actual level of obfuscation it really provides. Despite the design objectives it has to be observed that NSEC3 is a failure. The SHA-1 hashing algorithm is not exactly a major impediment and tools (of varying efficacy) are around to reverse NSEC3 hashing (such as https://github.com/anonion0/nsec3map). So NSEC3 can't stop domain enumeration if the enumerator has sufficient resources and patience. Supposedly NSEC3 also obscured the size of the zone file. But is there a fast way to estimate the size of an NSEC3-signed zone?

In May 2017, Bert Hubert shared a HyperLogLog technique to estimate its number of entries in a zone (https://indico.dns-oarc.net/event/26/contributions/437/attachments/395/671/hyperloglog_1.pdf). Casey Deccio reported on a different approach. One interesting property of the SHA-1 hash function is that it produces hash values that are uniformly distributed across the hash space in a normal distribution. If you look at the distances between these hash values, they are exponentially distributed across the hash space. What this means is that a small number of queries can reveal this distribution and therefore reveal a reasonable estimate of the size of the zone.

Given that NSEC3 hides nothing that can't be found out, why do folk bother to use it? It strikes me that if you really wanted to obscure the zone you would use a technique similar to that used by Cloudflare, namely to return an NSEC spanning record that is completely minimal (https://blog.cloudflare.com/black-lies/).

Web Material

This is by no means a comprehensive report of all the presentations across two very busy days. The full collection of presentation material can be found at the DNS OARC 32 web site at https://indico.dns-oarc.net/event/32/overview.

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