May 2021
Geoff Huston

DNS OARC 35

The DNS Operations, Analysis, and Research Centre (DNS-OARC) convened OARC-35 at the start of May. Here's some thoughts on a few presentations at that meeting that caught my attention.

TTL Snooping with the DNS

These days it seems that the term "the digital economy" is synonymous with "the surveillance economy". Many providers of services on the Internet spend a lot of time and effort assembling profiles of their customers, and these days it's not just data in terms of large-scale demographics but the assembling of large sets of individual profiles. We are all probably aware that we emit a steady stream of bits as a digital outflow when we use the Internet and there is a major ongoing effort to sniff this digital effluent and derive profiles of individual activities from this data. If an entity operates a recursive resolver in the DNS or operates a popular web service, then it's pretty clear how such use profiles can be assembled, if that's what they want to do. What is not so apparent is that our digital outflow can be sniffed by almost anyone. All it takes is a little ingenuity.

The presentation on "Trufflehunter" at DNS OARC 35 is a good case in point of being able to perform such indirect snooping. The question posed here is to what extent is *stalkerware* being used. By its very nature *stalkerware* is covert, as the intent is that the intended victim should be completely unaware that they have this app running on their device. So, is there a puff of tell tail smoke that can reveal *stalkerware* in action? The key observation is that often these apps use the DNS as a command-and-control channel. After all, the DNS is ubiquitous, and the total query volumes are truly prodigious. What's a few more queries in such a torrent of DNS? The app is simply hiding itself in a densely packed crowd. You might get a signal of active *stalkerware* if you operated a DNS resolver, but if you aren't the resolver operator than you just can't see the signal. Right?

Not true.

The critical piece of data that is used in this form of digital eavesdropping is the TTL (Time to Live) field in DNS responses. When a recursive resolver loads a response that was supplied by an authoritative server it will hold this item of data in its local cache for the count of seconds specified by the TTL field of the DNS record. The first time the recursive resolver loads the data from the authoritative server the recursive resolver will pass back the original TTL in its response as well as storing this record in its cache. If the recursive resolver receives a subsequent query for the same name within this TTL period it will use the local cached entry, but the TTL value it returns in the response will reflect the elapsed time since the original load.

For example:

```
$ dig www.potaroo.net @ns1.potaroo.net
www.potaroo.net.        6400    IN      A       203.133.248.108

$ dig www.potaroo.net @1.1.1.1
www.potaroo.net.        6240    IN      A       203.133.248.108
```

I can tell the TTL of the potaroo.net zone by querying the authoritative name server directly, and in this case, it returns the value 6400 seconds. The value of 6,240 seconds returned by the service behind 1.1.1.1 says that it's highly likely that somebody else has queried for this same name to this same recursive resolver service some 160 seconds earlier.

This way I can tell, approximately, if an application of service is being used. If I know the domain name(s) used by the application, then I can query for the same name to one or more of the commonly used open DNS resolvers. If the TTL is less than the TTL provided by the zone authoritative server, then I can infer that it is likely that the application is active in the area served by this open resolver as the name has already been loaded into the recursive resolver's cache.

The resolution of this technique to snoop on user activity is pretty coarse. To see if it can reveal some level of intensity of use of the name the snooper needs to understand the internal structure of the DNS resolver service. Interestingly, this is possible using this technique. Within each cache the TTL value of the name is reduced in a linear manner. Successive timed queries should see a linear decline of the TTL value over time. (Figure 1).

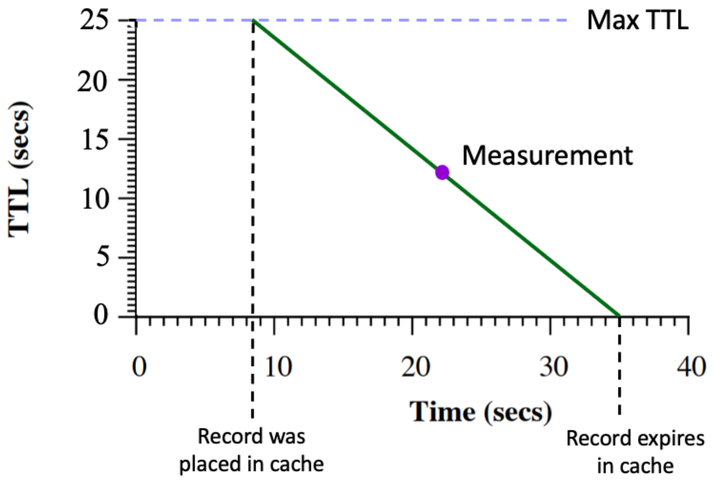


Figure 1 – Cache TTL behaviour, from: Audrey Randall, Trufflehunter
https://indico.dns-oarc.net/event/38/contributions/838/attachments/801/1441/Trufflehunter_OARC_20mins.pptx

When a number of less ordered TTL values is seen over time it is possible to match these observations to a collection of independent caches, and even to distinguish between font-side and back-end internal cache management in recursive resolver systems (Figure 2).

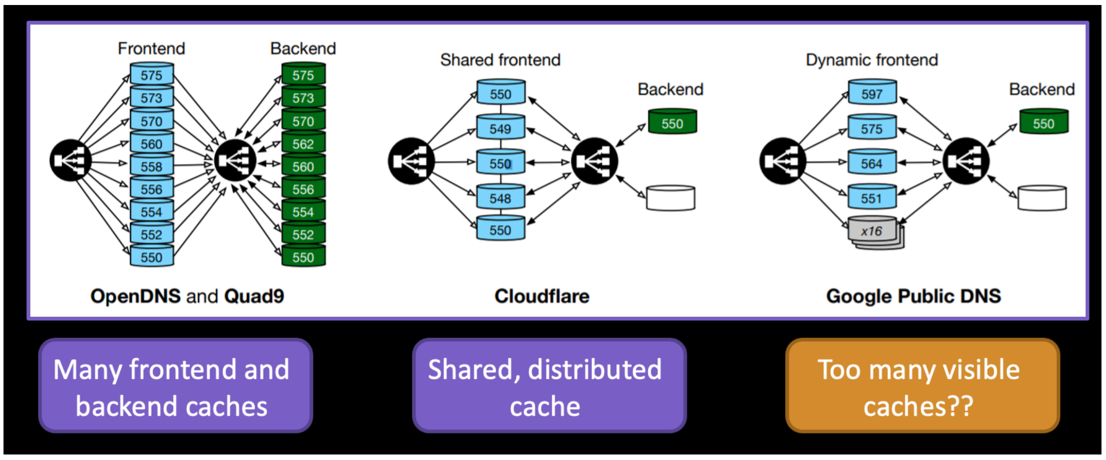


Figure 2 – Resolver internal structure inferred from Cache TTL behaviour, from: Audrey Randall, Trufflehunter
https://indico.dns-oarc.net/event/38/contributions/838/attachments/801/1441/Trufflehunter_OARC_20mins.pptx

This reconstruction of the likely internal architecture of the large open resolver systems from TTL measurement is a fascinating aspect of this study.

If one were to find this kind of indirect snooping using the cached record to be slightly unsettling, then the obvious counter is for the authoritative server to response with a TTL value in the response chosen at random between 0 and the cache lifetime TTL value. Yes, this reduces cache efficiency slightly, but it does appear to counter this form of indirect observation of name use.

However, in this case the information about usage that can be gleaned from this indirect form of measurement seems to be relatively minor in terms of detail and certainly appears to have minimal privacy concerns as far as I can tell.

DNS Cookies

Some years ago, the issue of abusing the DNS to perform various kinds of attacks was the most prominent theme in DNS meetings, including OARC. The use of UDP allows for a number of attacks that uses forged source addresses, such as reflection attacks to overwhelm a victim, or to perform a cache poisoning attack by an attacker misrepresenting itself as a server. A number of mechanisms were proposed at the time, including rate limiters, fallback to TCP and DNS Cookies.

DNS Cookies were standardised in the IETF in 2016 (RFC 7873) and are intended to be a lightweight security mechanism that provides some protection against some off-path attacks in the DNS, "Some level of protection" is a deliberately vague term, of course, but it is intended to describe that this is comparable to the protection provided by DNS over TCP, but without incurring the additional costs imposed by supporting TCP sessions. In other words, the server can assume that source address of a query is not spoofed, and this query is not part of a reflection attack.

In a DNS query sent from a client to a server when the client does not know the server's cookie value, the client uses an 8-byte random value, the client cookie value. This should be different for each server. When the client knows the server's server cookie value, the client also adds the server cookie value to its queries sent to this server.

The server cookie as defined in RFC7873 was constructed from a hashing function applied to the client's IP address and a random value determined by the server, and the client cookie. A subsequent refinement to this specification (RFC 9018) defines a 16-byte field with sub-fields of a version, a reserved section, a 4-byte timestamp and an 8-byte hash outcome (this is the result of a hash function applied to the client cookie, the other server cookie subfields, the client's IP address and a server secret), and is appended to the client cookie value, making a 24-byte field in total. This allows a server to associate a server cookie value with a client's IP address, and if a subsequent query from this client address does not contain this cookie, then the server can assume that the source address of the query has been spoofed.

The protection offered here is not against all forms of eavesdropping, but rather to ensure that any off-path potential attacker cannot misrepresent itself as either the client or the server, in a manner similar to the properties of a TCP session.

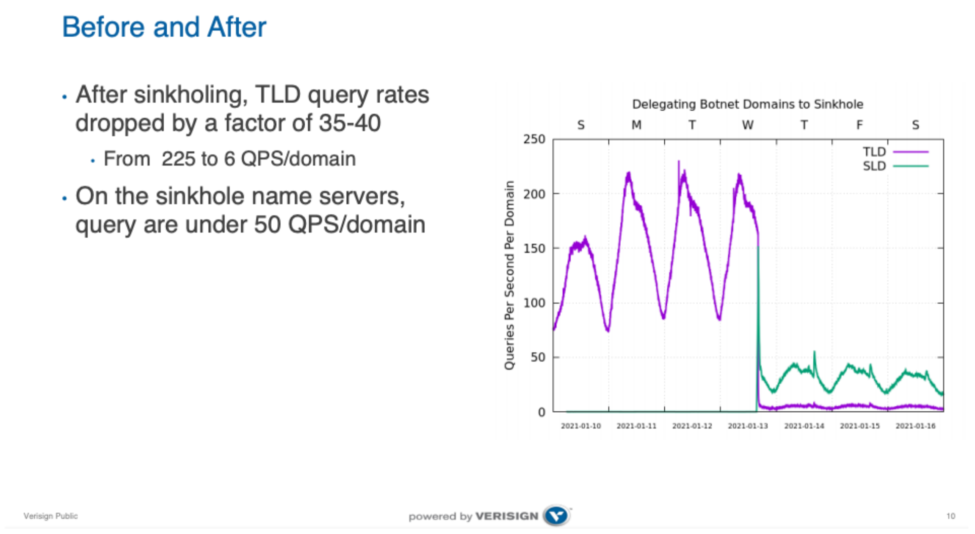
The question posed by Jacob Davis and Casey Deccio was about the level of support for Server and Client cookies. While some 98% of servers for the TLDs and the Alexa top million support EDNS options in queries, less than 30% support cookies. Of course, "support cookies" and "handle cookies correctly" are two different things, and in the case of client cookies the correct handling of cookies, namely to disregard cookies that do not faithfully use the client's cookies, was only supported in 20% of the cases they used for measurement. The result regarding server cookies was also surprising. They first established client's support of server cookies, then performed subsequent queries with no cookie and then a fake cookie value. More than 99% of the servers responded with a normal response to queries with the missing or fake cookie. If the guiding motivation for server cookies was to prevent reflection attacks using spoofed source addresses in queries, then the observed handling of server cookies in this work clearly indicated that there is no change to the susceptibility of these servers to participate in DNS reflection attacks.

It appears from this measurement exercise that the take up of cookies is still rather sparse, and around 30% of authoritative servers use DNS cookies and 10% of clients that ask queries use cookies. Perhaps more of a concern is their observation that less than 1% of authoritative servers enforce cookies by not responding to queries where the cookie values in a query is inconsistent with previous cookie values from the same client.

Botnet Traffic in the DNS

This was a presentation by Versign's Duane Wessels and Matt Thomas on the DNS query profile of a large active botnet. It's no secret that the DNS is one of the few universally accessible paths in today's Internet, and it should come as no surprise to observe that many botnets use the DNS as their command-and-control channel that they use to corral all these instances of the botnet code to act in a unified manner.

This particular case was a highly active botnet, generating some 375,000 queries per second to Verisign's .COM and .NET servers. However, it was also one where not all the domain names used in the bot code had been registered in the DNS, so most of these queries had a NXDOMAIN response. They observed some 71,000 unique three-label domain names, and a 900 second negative caching TTL applied to these domains. The queries for these domain names were diverted to a sinkhole, and the query rate per unique domain name dropped from 225 qps to 6 qps once the sinkhole was activated. (Figure 3)


Before and After

- After sinkholing, TLD query rates dropped by a factor of 35-40
  - From 225 to 6 QPS/domain
- On the sinkhole name servers, query are under 50 QPS/domain

Verisign Public — powered by VERISIGN — 10

Figure 3 – Sinkholing Botnet Traffic, from Duane Wessels, Matt Thomas, "Botnet Traffic" https://indico.dns-oarc.net/event/38/contributions/841/attachments/809/1430/verisign-sinkhole.pdf

The interesting aspect of this presentation was in looking at the chances in the traffic profile based on the way the sinkhole is implemented. How does the response code of the sinkhole affect the query rate? What's the difference between NXDOMAIN and NOERROR responses? And do the error code responses, SERVFAIL and REFUSED, alter the query behaviour?

It certainly appears that the DNS does not appreciate non-answers, and the query rate more than doubles when the query rates associated with NOERROR/NODATA and NXDOMAIN responses are compared to NOERROR/DATA responses (Figure 4).

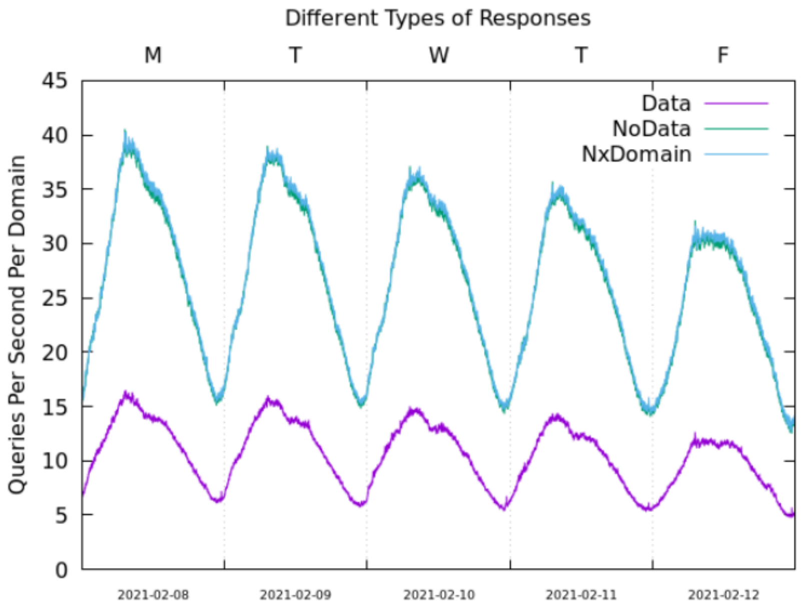


Figure 4 – Comparison of responses on Botnet Query Traffic, from Duane Wessels, Matt Thomas, "Botnet Traffic" https://indico.dns-oarc.net/event/38/contributions/841/attachments/809/1430/verisign-sinkhole.pdf

What if the answer is of the form "I am not going to answer this query." The options in the DNS are to respond with the REFUSED or SERVFAIL rcode values. DNS clients significantly increase their query volume in response to these two "no answer" response codes. Such responses do not have a cache TTL value, so this is an expected outcome. It also appears that both REFUSED and SERVFAIL are interpreted by client as a denial of that particular domain name, not a denial of the client itself for all queries posed by the client to this server.

It appears that the most "efficient" sinkhole in the content of botnets is to provide a misleading response, such as 127.0.0.1 I suppose. NXDOMAIN and NOERROR/NODATA create a slightly higher query rate, but the responses will still be cached in the DNS, so the query rate does not immediately blow out. The SERVFAIL and REFUSED codes generate the highest query rate. It was not reported here if no response at all would be worse than SERVFAIL and REFUSED. I suspect it would be, as the DNS client side is incredibly persistent in trying to get any response at all!

XoT

The DNS privacy effort does not apply only to DNS queries and responses. There are a number of scenarios where a zone admin may wish to keep the zone contents private during a zone transfer between the primary and secondary servers. One possible way to prevent unauthorized zone data collection during this zone transfer is to perform the transfer operation (AXFR and IXFR) using a TLS transport. This is being termed XFR-over-TLS, or XoT.

XoT complements existing TSIG, as it adds payload protection to TSIG's endpoint authentication. In terms of TLS, the draft specification (https://datatracker.ietf.org/doc/html/draft-ietf-dprive-xfr-over-tls) requires the use of TLS 1.3, using the DoT default port of 853. The client (secondary DNS server) must authenticate the server (primary DNS server). The server must authenticate the secondary through either mutual TLS authentication or some unspecified form of IP address authentication.

It is an interesting area of speculation whether we really need to continue with the concept of primaries and secondaries in the DNS, particularly if you combine zone signing (RFC8976) with XoT. If the outcome of the XFR process is to have a group of servers who all have the same zone data in order to respond to DNS queries then we could consider the use of closed groups as an alternative to the strict primary/secondary hierarchy, using the combination of the ZONEMD and SOA records as a means of assuring each group member that they are up to data with the current zone state.

Deployment of XoT does not pose the same huge logistical exercise of incremental deployment of DoH and DoT, as here we are looking at the more constrained network of primary and secondary servers, and the decision of zone to use XoT can be made independently of other zones.

While it could be argued that the need for secured XFR channels is not a high priority requirement in the effort to fix the privacy and tampering issues with the DNS, it is still an important component in the overall picture of a secure DNS. Support for XoT is coming from vendors of server software, including Bind (9.17), Unbound, and NSD.

There is a current debate in DNS circles whether vendors of DNS systems should implement proposals described in Internet drafts or await their formal adoption by the IETF standardisation process in the form of an RFC. Implementing every DNS draft as it progresses through the DNS channels of the IETF process is a daunting challenge that just gets worse as the number of drafts-in-progress in the IETF seems to be getting larger in recent times. This observation has provoked a conversation about the DNS turning into bloatware in the form of the "DNS Camel" conversation, and there is a thought to rely on the relatively sedate RFC publication rate as some form of rate containment of this incipient "bloating" of the DNS. On the other hand, pushing the IETF standardisation process into publishing specifications of technology without already having deployed implementations of the technology stands the risk of further pushing the RFC publication process into an even more protracted exercise of generating vacuous specifications without no visible application or use (Or, as was said many years ago when describing the OSI effort of the late 80's, just more "paperware about vapourware").

TCP Fast Open

TCP has a very mixed reputation. On the one hand, it is an extremely resilient protocol than can shift huge volumes of data, with sufficient flexibility to incorporate various forms of bolt-on security, such as TLS. On the other hand, it's clunky and slow for small transactions.

In some ways the getting the "just right" transport protocol for the DNS is the Goldilocks problem: UDP is just too small for the larger payloads we appear to want with DNSSEC, EDNS and expanding resource records, while TCP is just too big and heavy-handed. Sure, TCP can handle these larger DNS payloads with ease, but the overheads of setting up and maintaining TCP session state are a heavy price to pay in terms of setup latency and session maintenance. This observation is nothing new, and RFC955 from September 1985 explains this tension very clearly.

An earlier "solution" to this search for the "just right" transport protocol headed into two directions. Adding "reliability" to UDP (such as the Reliable Data Protocol (RFC908 asnd RFC1151 was one approach. The other was to strip put TCP, and the TCP for Transactions (T/TCP) was specified in RFC 1379 and subsequently RFC1644. T/TCP was certainly an interesting approach as it attempted to load the client's entire TCP interaction into a single opening packet, using the SYN, PSH, and FIN flags together with the data payload into its initial packet and the server was intended to respond with the SYN, FIN and PSH flags and the response data in its packet. All that was left was for the client to close off the T/TCP session with an ACK and it was done (Figure 5).

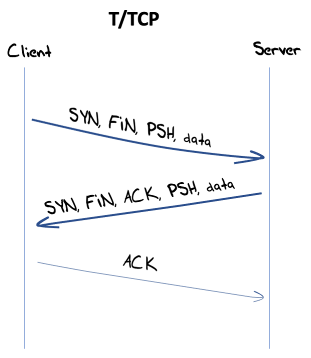


Figure 5 – T/TCP protocol exchange

T/TCP attracted some interest at the time, but the interest faded, and the protocol was declared "historic" in 2011 in RFC6247, citing security weaknesses. To put this into some perspective, T/TCP is at a par with UDP in terms of supporting a securable session, but if the case for T/TCP is that "it's about the same as UDP" then there seemed to be little point in pursuing it further.

The search for a faster version of TCP has continued, concentrating on the delays inherent in TCP's 3-way handshake. TCP Fast Open (TFO) (RFC7413) is the current preferred approach, which uses TCP cookies. The client requests a TCP cookie in the first open exchange, allowing the client to bypass the TCP handshake on subsequent connections and 'resume' the session by adding the cookie to the data in the first packet in the subsequent connection (Figure 6). TFO is probably entering the phase in its lifetime of being considered as a "mature protocol," with relatively widespread support across platforms and applications. In the DNS world TFO is supported in current releases from Bind, Knot, PowerDNS recursor and dnsmasq.

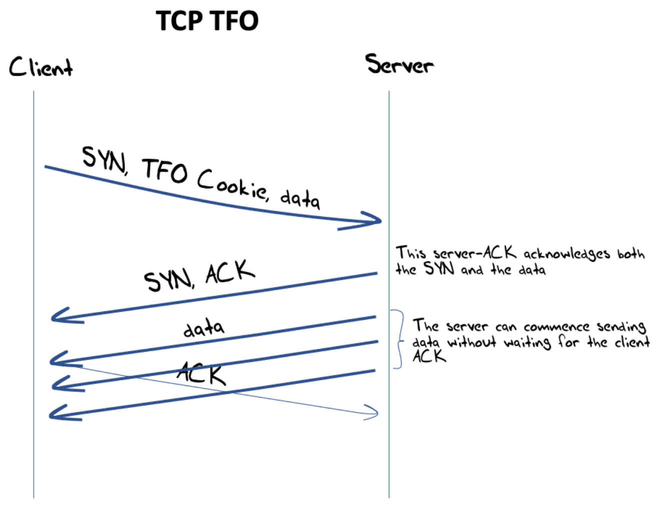

TCP TFO
Client
Server
SYN, TFO Cookie, data
This server-ACK acknowledges both the SYN and the data
SYN, ACK
data
The server can commence sending data without waiting for the client ACK
ACK

Figure 6 – TCP Fast Open Protocol Exchange

However, as Otto Moerbeek explained, TFO has encountered issues in today's DNS world. Where servers are implemented using anycast constellations some care must be taken to either steer the same end client to the same server engine where there is a record of the TFO cookie state, or the TFO information needs to be implemented in a shared state across all anycast engines in the anycast service constellation. On some platforms it appears that a single timeout on a TFO connect is enough for the platform to stop using TFO for all client connections for periods of minutes or ever hours. Otto's observations point to an outlook of a somewhat protracted deployment for TFO in the DNS, if at all.

It's more likely that efforts to optimise the performance of DNS over TCP will get lost in the larger effort to make DoH (DNS over HTTPS) more efficient, using TSL 1.3 and the 0-RTT session open options available with using DoH. It already takes some time to switch a DNS query from UDP to TCP (using the truncation flag in the UDP response) and I suspect that the pragmatic argument may well be that at this point the flipping between transport protocols is already exacting a latency cost on the DNS transaction. If we can make DoH sufficiently fast and efficient then we can bypass the entire UDP/TCP transport issue. Or at least that seems to be a strong line of thought in DNS realms at present.

DNS OARC 35

This is a small selection of those presentations from OARC 35 that caught my attention. The workshop program can be found at https://indico.dns-oarc.net/event/38/timetable/#20210507.detailed, and the recordings of presentations will be uploaded into the DNS-OARC YouTube channel (https://www.youtube.com/dns-oarc) in the near future.

Disclaimer

Author

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

www.potaroo.net