September 2013
Geoff Huston

A Question of DNS Protocols

One of the most prominent denial of service attacks in recent months was one that occurred in March 2013, launched against Spamhaus and Cloudflare.

> One writeup of this attack is at *http://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet*. I'm not sure about the claim that this attack "almost broke the Internet," but with a peak volume of attack traffic of some 120Gbps, it was a very significant attack nevertheless.

How did the attackers generate such massive volumes of attack traffic? The answer lies in the Domain Name System (DNS). The attackers asked about domain names, and the DNS system answered. Something we all do all of the time on the Internet. So how can a conventional activity of translating a domain name into an IP address be turned into a massive attack?

There are a few aspects of the DNS that make it a readily coercible means of generating an attack;

- The DNS makes use of a very simple UDP transaction, where client send queries to resolvers and resolvers send back responses.
- Within the DNS it is possible to send a relatively small query packet and get the resolver to reply with a much larger response. The DNS resolver becomes, in effect, a traffic "amplifier."
- There many so-called "open resolvers, who are willing to respond to queries from any clients on the Internet. In other words, these resolvers will accept DNS queries from anyone and send DNS responses to anyone.

> The Open Resolver project (http://openresolverproject.org) claims that there are some 28 million open resolvers on the Internet at present). That's a disturbingly high number if you are worried about ways to subvert the DNS to launch a platform for such attacks.

- Finally, it appears that way too few networks implement source address egress filtering, as described in BCP38. This is a packet filtering mechanism, which if universally implemented, would make it far more challenging to mount attacks that relied on essentially lying about the source address in an IP packet. A network could only emit packets whose source address is reachable via the same network.

The combination of these four factors produces a comprehensive vulnerability for the Internet. In performing experiments about the behaviour of the DNS we see a background level of DNS "probes" that contain a query for "ANY", often querying the domain ISC.ORG. In this case the original UDP request of 64 bytes generates a UDP response of 3,475 bytes, or an amplification factor of 54. If an attacker can send this UDP query to 100 of these open resolvers each second, using a source address of

the intended victim, then the attacker will be generating a query traffic volume of some 51Kbps, and victim will receive an incoming traffic load of 2.78Mbps. If you then enlist a bot army to replicate this simple attack one thousand-fold, then the attack traffic volume has exceeded a gigabit per second. With DNSSEC the response sizes get larger, and the unwitting accomplices in such attacks can expand to include the authoritative name servers of DNSSEC-signed domain.

The problem is that, in flight, the traffic looks like all other traffic. These are simple DNS responses, and the network carries DNS responses as one of the more common packet types. So simple filtering is simply not an option, and we need to look deeper to see how we could mitigate this rather worrisome vulnerability. This has been a long-standing issue.

> For example, these was a presentation from May 2006 on this topic at an IETF meeting (http://www.iepg.org/july2006/1-frank-scalzo.pdf). The only difference is that the number of open resolvers has subsequently jumped from 500,000 to in excess of 25 million!

There have been a number of threads of discussion on this topic.

One thread of conversation goes along the lines that if everyone implemented the measures described in BCP38, this would prevent endpoints from emitting DNS query packets with a false source address, and thereby preventing these reflection attacks to be mounted using the DNS. Of course this is a long standing conversation that is older than BCP38 itself. BCP38 is now 13 years old as a document, and the somewhat worrisome observation is that nothing much appears to have happened in terms of improving the situation about source address spoofing over the past 13 years, so there is not a lot of optimism that anything will change in the coming months, if not years.

There is a conversation thread that says that resolvers should implement response rate limiting (RRL), and silently discard repetitive queries that exceed some locally configured threshold. While this is relatively effective in the case of authoritative name servers, it is less effective in the face of a massive pool of open recursive resolvers, as in this latter case the query load can be spread across the entire pool so that each resolver may not experience a detectable level of repeated queries. It is also possible to use a very pool of authoritative name servers in the same manner. However, this consideration does not weaken the advice that authoritative name servers should implement RRL in any case.

> There is an informative presentation by Randy Bush at the 2013 APRICOT conference (http://bit.ly/17QiNOV), illustrating the before and after state of the application of RRL for an authoritative name server.

Another thread of conversation is to shutdown the open recursive resolvers. The open resolver project (http://openresolverproject.org) is working on a "name and shame" approach to the problem, and is hopeful that by allowing individual resolver operators to check their own status, that these resolvers will be closed down. Like the universal adoption of BCP38, It's hard to be overly optimistic about this approach. Part of the issue is that there is a large volume of unmanaged or semi-managed systems connected to the Internet, and the vulnerabilities they create are not necessarily known to the parties who deployed these systems in the first place. For example, one way to create more robust "plug and play" systems is to reduce the amount of environmental configuration that needs to be loaded into such systems in the first place. Equipping such standalone units with their own DNS resolver appears to be a way to remove an additional configuration element from the unit. The downside is that if these units are configured as an open recursive resolver, then they become part of the overall problem of the massive population of open resolvers on today's Internet.

The behaviour of the DNS where a small-sized query generates a large response is one that appears to be intrinsic to the DNS, and particularly DNSSEC. If we want some form of security in the DNS so that the client can be assured that the response they receive is authentic and current, and has not been tampered with in any way, then the overheads of cryptographic signature blocks and the size these signature blocks take up appears to be an intrinsic part of the DNS' security architecture. So we want a lightweight fast DNS, so we appear like using UDP, coupled with a ubiquitous distribution of recursive resolvers and the associated resolver caches, but we also want DNS responses that can be verified, so we like DNSSEC. So the responses get larger, and the outcome is that the DNS is a highly effective platform for massive traffic attacks where there is a very limited space to mitigate the associated risks.

If we want to close the door on using the DNS to mount large scale attacks, there does not appear to be much space left to manoeuver here.

However, there is a conversation that has not quite petered out yet. That conversation is about the use of UDP in the DNS.

The original specification of the name system (RFC1123) allowed for the use of both UDP and TCP as the transport service for DNS queries and responses. The relevant part of this specification reads:

> "... it is also clear that some new DNS record types defined in the future will contain information exceeding the 512 byte limit that applies to UDP, and hence will require TCP. Thus, resolvers and name servers should implement TCP services as a backup to UDP today, with the knowledge that they will require the TCP service in the future."

Why 512 bytes? The 512 byte limit was based on the IPv4 Host Requirement Specification (RFC1122), where it is required that all IPv4 systems must accept an IP packet that is at least 576 octets in size. Allowing for 20 bytes of IP header, 8 bytes of UDP headers and room for the maximum of 40 bytes of IP options, this implies that the maximum payload in a UDP packet that will be accepted by all IPv4 hosts is restricted to 512 bytes.

It should be noted that it is theoretically possible that there are links that do not support IP packets of 576 bytes in size, so even these 576 byte IP packets may possibly be fragmented in flight. The limit being referred to here is the largest (possibly reassembled) packet that a host will assuredly accept.

Now of course it is possible to generate larger packets in IPv4, to a theoretical maximum of 65,535 bytes in size, which accommodates a UDP payload of 65,507 bytes, but such larger packets will probably be fragmented in flight. In such a case, when this is combined with typical firewall behaviour, then the trailing packet fragments may not be passed onward to a client at all, as many firewall configurations use acceptance rules based on UDP and TCP port address. As the trailing fragments of a fragmented IP datagram have no UDP or TCP packet header, this leaves with the firewall with a quandary. Should the firewall accept all IP fragments? In this case the security role of the firewall may be compromised through the transmission of fragments that form part of some form of hostile attack. Or should the firewall discard all IP fragments? In this case a sender of a large packet should be aware that if there is fragmentation, then the trailing packet fragments may not be passed to the receiver. So with the two considerations that hosts are not required to accept UDP datagrams that are larger than 576 bytes in size, and firewalls may discard trailing fragments of an IP datagram, the original DNS response to this situation was to limit all of its UDP responses to 512 bytes in size, and always use TCP as the backup plan if the DNS response was larger than 512 bytes in size.

However, clients may not know in advance that a DNS response is larger than 512 bytes in size. To signal to a client that it should use TCP to retrieve the complete DNS response, the DNS resolver will respond in UDP with a partial response, and set the "truncated" bit in the response.

We continued for some years with this approach. The DNS used UDP for the bulk of its transactions, which all fitted within the 512 byte limit, and for the relatively infrequent case where larger DNS responses were being generated, the UDP response was truncated, and the client was expected to repeat the question over a TCP connection.

This was not an altogether comfortable situation when we then considered adding security credentials to the DNS. The inclusion of digital signatures in DNS responses implied that very few DNSSEC responses would fit within this 512 byte limit. But if the process of switching to TCP was to respond to the UDP query with a UDP response that essentially said "Please use TCP" then this adds a considerable delay to the DNS function. Each query would now involve a minimum of 3 round-trip time interactions with the server rather than just the single round trip time interval for UDP (a TCP transaction still includes one transaction for the UDP query and truncated UDP response, one for the TCP connection establishment handshake, and one for the TCP query and response). The next refinement of the DNS was to include a way to signal that a client was able to handle larger DNS responses in UDP.

As pointed out in RFC5966:

> "Since the original core specifications for DNS were written, the Extension Mechanisms for DNS (EDNS0 [RFC2671]) have been introduced. These extensions can be used to indicate that the client is prepared to receive UDP responses larger than 512 bytes. An EDNS0-compatible server receiving a request from an EDNS0-compatible client may send UDP packets up to that client's announced buffer size without truncation."

EDNS0 allows for the UDP-based DNS response to grow to far higher sizes. The result of this extension is that it appears that the DNS largely uses UDP and EDNS0 is used to allow for the larger sized responses. TCP is now often regarded as a somewhat esoteric option, only being used for zone transfer operations, and if the zone does not want to support zone transfers as a matter of local policy, then it is commonly thought that the role of TCP is no longer essential.

Section 6.1.3.2 of the Host Requirements Specification that addresses requirements for applications and support (RFC1123) states:

> DNS resolvers and recursive servers MUST support UDP, and SHOULD support TCP, for sending (non-zone-transfer) queries.

In the world of standards specifications and so-called normative language, that "SHOULD" in the above text is different to a "MUST." It's a little stronger than saying "well, you can do that if you want to", but it's a little weaker than saying "You really have to do this. There is no option not to." Little wonder than some implementors of DNS resolvers and some folk who configure firewalls came to the conclusion that DNS over TCP was an optional part of the DNS specification that need not necessarily be supported.

But DNS over TCP is not only a tool to allow for large DNS responses. If we review the preconditions for the use of the DNS in large scale reflector attacks, the widespread support of UDP for large packet responses, and the relatively sparse use of BCP38, allows an attacker to mount a reflection attack by co-opting a large set of open resolvers to send their responses to the target system, by using UDP queries whose IP source address is the IP address of the intended victim.

TCP does not have the same vulnerability. If an attacker were to attempt to open up a TCP session using an IP source address of the intended victim, the victim would receive a short IP packet (IP and

TCP header only, which is a 40 byte packet) containing only the SYN and ACK flags set. As the victim system has no pre-existing state for this TCP connection, it will discard the packet. Depending on the local configuration, it may send a TCP RESET to the other end to indicate that it has no state, or the discard may be completely silent. This removes one of the essential preconditions for a reflector amplification attack. If the attack traffic with the spoofed source address of the intended victim uses a 40 byte SYN TCP packet then the victim will receive a 40 byte SYN/ACK TCP packet. The DNS attack amplification factor would be effectively removed.

If the DNS represents such a significant vulnerability for the Internet through these UDP-based reflection attacks, then does TCP represent a potential mitigation? Could we realistically contemplate moving away from the ubiquitous use of ENDS0 to support large DNS responses in UDP, and instead use DNS name servers that limit the maximal size of their UDP responses, and turn to TCP for larger responses?

Again, lets turn to RFC5966:

> "The majority of DNS server operators already support TCP and the default configuration for most software implementations is to support TCP. The primary audience for this document is those implementors whose failure to support TCP restricts interoperability and limits deployment of new DNS features."

The question we are looking at here is, can we quantify the extent to which DNS resolvers are capable of using TCP for DNS queries and responses? How big is this "majority of DNS server operators" being referred to above?

The Experiment

We conducted an experiment using a modified DNS name server, where the maximal UDP packet size, was configured to 512 bytes, and then set up an experiment where a simple query to resolve a DNS name would generate a response what could not fit within 512 bytes.

While this is relatively easy if the query includes DNSSEC, if we want to set up a condition where all DNS responses are larger than 512 bytes for a domain then we need to use a slightly different approach. The approach used in this iteration of the experiment is to use the DNS name alias function, the CNAME record.

Here is a sample zone:

```
$TTL 1h
@       IN      SOA     nsx.dotnxdomain.net. research.apnic.net.  (
                        2013011406      ; Serial
                        3600            ; Refresh
                        900             ; Retry
                        1               ; Expire
                        1 )             ; Minimum
        IN      NS      nsz1.z.dotnxdomain.net.

z1      IN      A       199.102.79.186

*       IN      A       199.102.79.186

4a9c317f.4f1e706a.6567c55c.0be33b7b.2b51341.a35a853f.59c4df1d.3b069e4e.87ea53bc.2b4cfb4f.987d5318.fc0f8f61.3cbe5065.8d9a9ec4.1ddfa
1c2.4fee4676.1ffb7fcc.ace02a11.a3277bf4.2252b9ed.9b15950d.db03a738.dde1f863.3b0bf729 IN CNAME
33d23a33.3b7acf35.9bd5b553.3ad4aa35.09207c36.a095a7ae.1dc33700.103ad556.3a564678.16395067.a12ec545.6183d935.c68cebfb.41a4008e.4f
291b87.479c6f9e.5ea48f86.7d1187f1.7572d59a.9d7d4ac3.06b70413.1706f018.0754fa29.9d24b07c

33d23a33.3b7acf35.9bd5b553.3ad4aa35.09207c36.a095a7ae.1dc33700.103ad556.3a564678.16395067.a12ec545.6183d935.c68cebfb.41a4008e.4f
291b87.479c6f9e.5ea48f86.7d1187f1.7572d59a.9d7d4ac3.06b70413.1706f018.0754fa29.9d24b07c IN A 199.102.79.187
```

The use of the combination of long label strings and a CNAME construct forces a large response, which, in turn, triggers DNS to send a truncated UDP response in response to a conventional query for

the address record for the original domain name. The truncated UDP response should force the client resolver to open a TCP session with the name server, and ask the same query again.

To ensure that the authoritative name server directly processed every name query we used unique labels for each presented experiment, and ensured that the DNSSEC singed zones were also unique. This ensures that local DNS resolver caches cannot respond to these name queries.

The first question we are interested is: How many clients were able to successfully switch to TCP following the receipt of a truncated response in UDP?

We conducted an experiment by embedding a number of test cases inside an online ad, and over 8 days at the end of July 2013 we presented these tests to 2,045,287 end clients. We used four experiments. Two experiments used a name where the query and response would fit within a 512 byte payload, as long as the query did not include a request for DNSSEC. One of these domain names was unsigned, while the other was signed. The other two experiments using the CNAME approach to ensure that the response would be larger than 512 bytes. Again, one zone was signed, while the other was unsigned.

Experiment	**UDP queries**	**Truncated UDP responses**	**TCP responses**	**Truncated UDP to TCP fail**
Short, unsigned	2,029,725	2	6	0
Short, signed	2,037,563	1,699,935 (83.4%)	1,660,754 (81.5%)	39,101 (1.9%)
Long, unsigned	2,023,205	2,021,212 (99.9%)	1,968,927 (97.3%)	52,285 (2.6%)
Long, signed	2,033,535	2,032,176 (99.9%)	1,978,396 (97.3%)	53,780 (2.6%)

This data appears to point to a level of failure to followup from a truncated UDP response to a TCP connection of some 2.6% of clients.

That level of failure to switch from a truncated UDP response to re-phrase the query over TCP is large enough to be significant. The first question is this failure due to some failure of the DNS authoritative name server or a failure of the client resolver? If the name server is experiencing a high TCP session load it will reject new TCP sessions. The way it does this is to respond to incoming TCP session establishment packets with a RESET. We saw no evidence of this session overload behaviour in the packet traces that were gathered by the authoritative name server. So the TCP failure looks to be some issue closer to the client resolver than to the authoritative name server.

We can also look at this from the perspective of the set of visible resolvers. How many resolvers will switch to use TCP when they receive a UDP response that is truncated? Before looking at the results, it needs to be noted that the only resolvers that are exposed in this experiment are those resolvers that query our authoritative name server (*visible* resolvers). If a resolver is configured to use a recursive resolver, then its behaviour will not be directly exposed in this experiment. It should also be noted that even when the visible recursive resolver is forced to use TCP to query the authoritative name server, the recursive resolver may still relay the response back to its client via UDP, using EDNS0 to ensure that the larger UDP response is acceptable to the client.

The 2 million clients used a total of 80,505 visible resolvers. Some 13,483 resolvers, or 17% of these visible resolvers did not generate any TCP transactions with the authoritative name server. These 13,483 UDP-only resolvers made a total of 4,446,670 queries, and of these some 4,269,495 responses were truncated, yet none of these resolvers switched to TCP.

There is a second class of filtering middleware that operates on incoming traffic. In such cases the authoritative server will see an incoming TCP SYN packet to establish the DNS connection, and the server will response with a SYN+ACK packet. Because this packet will be blocked by the filtering middleware, it will never get passed through to the resolver client, and the TCP connection will not be established. This SYN-only behaviour was observed in just 337 resolvers, which represents some 0.4% of the set of visible resolvers. These resolvers generated a total of 1,719,945 queries, and received 1,575,328 truncated UDP responses.

Why is the client-level TCP failure rate at some 2% - 3% of clients, while at the resolver level the TCP failure rate is some 17% of visible resolvers? There are at least three possible reasons for this.

Firstly, in some cases we observe service providers using DNS forwarder farms, where queries are spread across a number of query engines. When a DNS query is re-phrased using TCP, it may not use the same forwarder to make the query.

Secondly, we should factor in end client failover to another DNS resolver that can support DNS transactions over TCP. Most clients are configured with multiple resolvers, and when one resolver fails to provide a response the client will ask the query of the second and subsequent resolvers in its resolver set. If any of the visible resolvers associated with the resolvers listed in the client's resolver set are capable of using TCP, then at some stage we will see a TCP transaction at the authoritative name server. In this more prevalent case of TCP failure, either the resolver itself is not capable of generating a DNS query using TCP (due presumably to local limitations in the resolver software or local configuration settings), or some network middleware is preventing the resolver performing outgoing TCP connections to port 53.

Thirdly, the distribution of end clients across the set of visible resolvers is not even, and while some resolvers, such as the set used by Google's Public DNS service serve some 7% of all end clients, others serve a single end client. We observed that some 53,000 experiments, out of a total of some 2 million experiments, failing to complete a TCP-based DNS resolution, so it is also possible that these 13,483 visible resolvers that do not support TCP queries are entirely consistent in volume with this level of end client failure to resolve the experiment's DNS label.

There is a slightly different way to look at this question. While we saw some 53,000 experiments that failed to complete the DNS resolution at all, how many experiments were affected by this deliberate effort to force resolvers to use TCP? How many clients were impacted in terms of longer DNS resolution time through the use of DNS resolvers that fail to switch to use TCP?

Experiment	UDP queries	Truncated UDP responses	TCP responses	Truncated UDP to TCP fail	Used TCP-Fail Resolvers
Long, unsigned	2,023,205	2,021,212 (99.9%)	1,968,927 (97.3%)	52,285 (2.6%)	124,881 (6.1%)
Long, signed	2,033,535	2,032,176 (99.9%)	1,978,396 (97.3%)	53,780 (2.6%)	129,555 (6.4%)

This table shows that slightly more than 6% of all clients used a resolver that was unable to repeat the DNS query over TCP. Some 75,000 clients used an alternate resolver that was capable of performing the TCP query, while the remainder were unable to resolve the DNS name at all.

After running this initial experiment, we considered the use of the CNAME construct to inflate the DNS response to more that 512 bytes, and wondered if this additional DNS indirection created some problems for some resolver clients. Another approach to coerce client resolvers to use TCP is to modify the name server code and drop its UDP maximum size to 275 octets, so that the name server will truncate the UDP response for any response of 276 bytes or larger. In this way a DNS query for

the short unsigned name would fit within the new UDP limit, but in all other cases the UDP response would be truncated.

The results we saw for this second experiment, which removed the CNAME entry, and used an authoritative name server with a 275 byte UDP payload limit, with three days of collected data, are summarized in the following table.

Experiment	UDP queries	Truncated UDP responses	TCP responses	Truncated UDP to TCP fail
Short, unsigned	704,458	0	3	0
Short, signed	706,975	581,081 (82.2%)	568,704 (80.4%)	12,377 (1.8%)
Long, unsigned	703,384	571,182 (81.3%)	556,835 (79.2%))	14,977 (2.1%)
Long, signed	706,553	579,639 (82.0%)	564,716 (79.9%)	14,923 (2.1%)

These results are largely consistent with the results of the original experiment. The failure rate is slightly lower, and this may lead to one further possible reason why a DNS resolver will fail to re-phrase the query in TCP following making a query in UDP. DNS Resolvers may be configured to operate within the parameters of a maximal elapsed time to resolve a DNS name, and once this time has elapsed they may simply abort their efforts to resolve a DNS name. With the CNAME construct we have seen some resolvers perform an additional query. When the query the original DNS name, the name server returns the CNAME record and the A record for the target of the CNAME. A small proportion of resolvers appear not to use the supplied A record in the response, and take the CNAME target name and launch a second query for the A record of this CNAME target name. The increase in elapsed time to perform another DNS query may have some bearing on the triggering of the resolver client's maximum resolution timer. It appears that the number of clients who would be impacted by a forced switch to TCP is of the order of 2% of clients.

Conclusion

The original specification of the DNS called for resolvers to use UDP when the response was 512 bytes or smaller, and TCP was to be used for larger DNS transactions. DNS clients would interpret the truncated flag in a DNS UDP response to trigger a re-query using TCP.

With the introduction of EDNS0, clients can now signal their capability to accept larger UDP datagrams, with the result that the fallback to TCP for large DNS responses is used less frequently, to the extent that there is now a concern that a significant set of clients cannot resolve a DNS name if that resolution operation is required to occur using TCP.

However, DNS UDP is being used in various forms of malicious attack, using DNS queries where the response is far larger than the query. The combination of source address spoofing and DNS over UDP is presenting us with some significant issues. For that reason there is a renewed consideration of the viability of reverting to TCP for various forms of larger DNS responses, which effectively prevents source address spoofing in the DNS query / response interaction.

In this experiment we've looked at the impact a forced switch to DNS over TCP would have on clients. In particular, what proportion of clients would no longer be able to complete a DNS name resolution process if the process necessarily involved the use of TCP? Our measurements of a sample of 2 million clients in early August 2013 points to a DNS resolution failure rate for some 2% of clients.

The picture for individual DNS resolvers appears to be somewhat worse, in that some 17% of visible resolvers do not successfully followup with a TCP connection following the reception of a truncated UDP response.

While that 17% number is surprisingly high, there are two mitigating factors here.

It appears that clients make use of multiple DNS resolvers in their local DNS configuration, so that failure of an initially selected resolver to respond to a query due to lack of support for TCP may be resolved by the clients selecting the next resolver from their local resolver set. For this set of clients, which appears to encompass some 4% of the total client population, the penalty is increased DNS resolution time, where the resolution of a name requires the client to failover to the other resolvers listed in their local DNS resolver set.

Secondly, the more intensively used visible DNS resolvers appear to be perfectly capable of supporting TCP-based queries, so the issues with TCP support in the DNS appear to be predominately concerned with resolvers that are used by a relatively small pool of end clients.

Disclaimer

About the Author

Geoff Huston B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. He has been closely involved with the development of the Internet for many years, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector. He is author of a number of Internet-related books, and was a member of the Internet Architecture Board from 1999 until 2005, and served on the Board of Trustees of the Internet Society from 1992 until 2001.

www.potaroo.net