December 2021
Geoff Huston

DNS-OARC 36

It's conference and workshop season right now, and November has been unusually busy this year. At the end of the month was the DNS Operations and Research meeting, DNS-OARC 36. These are my notes from those presentations at the meeting that I found to be of interest.

Slack's DNSSEC Debacle

It's a pretty conventional approach to change management in operational environments: Understand precisely what you want to achieve, and how you will observe that you have succeeded in making the change. At the same time prepare a list of observable behaviours that will indicate that the change has not gone according to the plan, and if you see any of these outcomes then be prepared to back out of the change, restore the original state and then take your time to work out what went wrong and why.

It's a conservative and proven approach and works most of the time. Except of course for those times when it doesn't. There are, from time to time, those nightmare scenarios where the change has created a state where you can't back out, and if the change has gone astray you are left with a broken service. In Slack's case the change on the 30th September produced a broken DNS outcome where the `slack.com` domain name was not resolving for 24 hours and it simply couldn't be undone during this period.

What they were trying to do was simple. They wanted to DNSSEC-sign `slack.com`. Their motive is obvious. There are many forms of malicious attacks on the DNS these days all of which attempt to deceive users and redirect them to the attacker's resources. Yes, if you are using TLS (and you really should be using TLS, no excuses any more), then the domain name certificate is meant to protect the user from being deceived in this manner, but there are still two residual problems with using TLS. Firstly, users have an annoying tendency to click through the "This site looks like it's a fake. The certificate does not match. Are you sure you want to proceed?" warnings. Secondly if you want to be a little more elaborate in your attack then you only need to pervert the DNS as seen by a CA and coerce that CA to issue a certificate for your fake credentials to make these often-disregarded warnings disappear. DNSSEC-signing your domain name makes it harder for the attacker. It's not perfect, and there are still many caveats about the effectiveness of DNSSEC, but it will work for some 30% of the Internet's user base and most CAs, so it's better than doing nothing at all.

Except when it all Goes Wrong!

These days it's not fashionable, and perhaps more importantly it's not even sensible to run your own DNS infrastructure on your own servers. Serving a global user base requires a high capacity widely distributed service platform that can not only serve normal DNS query traffic but can withstand all but the most extreme forms of hostile attack. So, like everyone else, Slack outsources its DNS to one of a small number of large-scale DNS hosting platforms, and in their case, they chose Amazon's Route 53 service (https://aws.amazon.com/route53/). Their choice is not usual by any means and Amazon is the 9th largest such provider in this space, serving a little over a million domains (see Table 1 below). The next step was slightly risky, in so far as increasing the number of external dependencies has its attendant

risks, but their decision to use a different platform, NS1 (https://ns1.com/), to host some subdomains of `slack.com` is still pretty conventional.

In setting up signing for `slack.com` they devised a robust plan: external monitoring and alerting using the various open DNS resolvers, signature coherency monitoring using *dnsviz* (https://dnsviz.net/) and the Verisign *DNSSEC debugger* (https://dnssec-debugger.verisignlabs.com/) and propagation checking using *dnschecker* (https://dnschecker.org/), and a staged rollout over a number of domain names operated by Slack, culminating in slack.com (Figure 1).

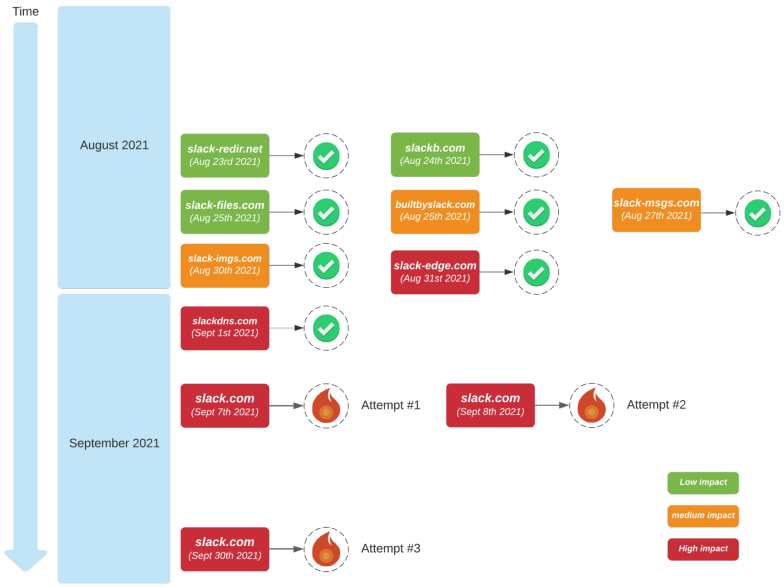

Figure 1 – Slack.com DNSSEC signing timeline (from https://slack.engineering/what-happened-during-slacks-dnssec-rollout/)

How do you "sign" a domain name? It seems like an odd question, but in this case it's an important question. It's a sequence of operations. Firstly, generate a keypair (or two if you are using separate ZSKs and KSKs) and add these keys to your zone as DNSKEY RRs. Secondly, assuming you are signing in advance, generate a signed zone by adding RRSIG RRs to all signable entries in the sone. Or if you are using dynamic signing, then prime the front-end signer with the signing keys. At this point the domain is "signed" but others are not in a position to rely on it yet. The final step is to pass a DS record to the parent domain. Once the DS record is in place DNSSEC validation will take place for validation-enabled clients. If you are using a DNS hosting provider, then the sequence of steps is much the same, but the levers you need to pull to get the hosting provider to do this on your behalf may be a little more indirect.

So, in the first attempt, on the 7th September, immediately prior to pushing the DS record to their DNS registrar for inclusion into the .com zone, a large ISP outage in the US was already causing access issues for the Slack app, so the change was stopped.

The second attempt on the next day fell at the CNAME hurdle. If you use CNAME to map a DNS name to a canonical name, then the only record associated with the name is the CNAME record and any DNSSEC signature records (RFC2181). The implication is that a CNAME cannot be at a zone apex. Now many DNS admins ignore this stricture and place CNAME records at the zone apex and most of the time it Just Works. Many resolvers are tolerant of CNAMES at the zone apex and will follow the CNAME pointer to resolve the name in any case. Which is just as well as CNAMES are a critical part of the larger story of hosted content on CDNs. However, as Slack found out, if the zone has DNSSEC signature records some resolvers apply a stricter check and fail to resolve a name that uses a CNAME at the zone apex, even if the DS record is not present at the zone's parent. This is a common issue and despite previous attempts to provide a substitute for CNAME records (such as ANAME) nothing useful has been standardised so far. NS1 uses a locally defined synthetic record, an ALIAS record, that uses

server-side aliasing to work around this issue. So, once more, Slack stopped the DNSSEC-signing process, backed out and then replaced the CNAME records with ALIAS records.

At this point Slack had twice successfully backed out of the DNSSEC-signing process and appeared to have mistakenly formed the view that they could stop and back out at any point in the process if they encountered unforeseen issues in future signing attempts.

On the 30th September they tried once more, enabling signature generation on all sub-zones of `slack.com`, performed validation steps and then passed the DS record to the DNS registrar to complete the process by initiating the process to place the DS record into the .com zone.

At this point DNS breakage was being reported, and let me quote directly from Slack's own account of what transpired and their response from their engineering note:

> Soon afterwards, Customer Experience started getting reports from users seeing "ERR_NAME_NOT_RESOLVED" errors in Slack clients. Slack applications are based on Chromium and integrate a way to export a NetLog capture, which is incredibly helpful when debugging network problems and analyzing performance. NetLogs are extensively used by Customer Experience and Engineering at Slack.
>
> 'ERR_NAME_NOT_RESOLVED' type errors can have multiple causes. In this specific case, it was caused by NODATA DNS responses from some DNS recursive resolvers. Our engineering teams started to investigate.
>
> During our initial investigation we saw most reports were from a private (corporate) DNS recursive resolver and Google's Public Resolver (8.8.8.8). This was surprising, as impact to Google DNS *should* mean a much larger impact than our monitoring and customer reports indicated.
>
> At this point we decided to pull the DS record from the DNS registrar, effectively removing the slack.com DS record at the '.com.' zone to 'stop the bleeding', and later focus on the 'NODATA' DNS responses reported by customers.
>
> *https://slack.engineering/what-happened-during-slacks-dnssec-rollout/*

The problem was that pulling the DS record from the DNS registrar is not an immediate cause and effect thing. Caching is taking its inevitable toll, and the timeline of the cached lifetime of the DS record is often far longer than one hour. Recursive resolvers have been known to interpret source-specified cache lifetime values as more like a guideline rather than a rule and cache the record for their own preferred time. Sometimes it may be shorter, other times it may be a lot longer.

For example, my local recursive name server, running Bind 9.16 returns a TTL on a DS record for one of my signed names of 3,216 seconds, or a little under one hour, but when I ask Google's public DNS server the TTL of the same record I get back a value of 21,600 seconds, or 6 hours. By default, the .com zone uses a 24-hour TTL value on DS records.

The next operational behaviour that contributed was than we you are on Plan B and it is not working then there is an overwhelming sense of panic and you find yourself heading off-plan and doing ill-considered things. Because panic. In Slack's situation they had expected the withdrawal of the DS record to take effect within one hour, as they had been led to understand that DS records were never cached. Withdrawal of the DS record from the .com zone would cause the `slack.com` zone to revert to being unsigned. But this was not happening. So, then they Made It Worse:

> After nearly an hour, there were no signs of improvement and error rates remained stable, trending neither upwards nor downwards.

> Our Traffic team was confident that reverting the DS record published in MarkMonitor was sufficient to eventually resolve DNS resolution problems. As things were not getting any better, and given the severity of the incident, we decided to rollback DNSSEC signing in our slack.com authoritative and delegated zones, wanting to recover our DNS configuration to the last previous healthy state, allowing us to completely rule out DNSSEC as a problem.
>
> *https://slack.engineering/what-happened-during-slacks-dnssec-rollout/*

What happened instead was that they had a DS record out there with a 24-hour cache time and by withdrawing the DNSKEY RR from `slack.com` all validating recursive resolvers were now encountering a validation failure and returning a SERVFAIL error code. At this point they entered the next phase of incident management, otherwise known as Clutching at Straws! They seriously thought of restoring the previously removed DS record. What stopped them was a Route 53 operational behaviour that did not keep a copy of the old ZSK value once the zone was no longer signed. If they had effectively re-signed the zone with a new ZSK the presence of old cached entries would still hamper resolution. So, the resolution of the problem was ultimately to wait through the 24-hour cache period and wait for resolvers to pick up on the new unsigned state of `slack.com`.

> There is an underlying TTL issue here, and some inherent problems with NS and DS records. These two records are "borrowed" from the child zone and inserted into the parent zone to facilitate name server discovery in the case of NS record and signature chaining in the case of the DS record. But in the process the TTL values for these parent zone records are specified by the parent, not the delegated child zone. Even the use of CDS/CDNSKEY does not allow the child to specify the TTL of the parent zone DS record. Instead, these TTL values are defined by the parent. The problem is that parent defined TTL values are always wrong! When the child needs to perform a change the TTL value is too long and slows the change through extended caching of the old value. When the child has an outage at the source then the TTL is invariably too short, and they give the child no time to repair the problem before the cache expires (just think of Facebook's recent outage to see the implications of a short TTL to cause cascading failure). To compound this problem, we have recursive resolver behaviours that reject the authoritative server's value of TTL reality and substitute their own! There is no good value here for the parent to use

Slack Engineering then performed a more detailed analysis of the original error. The analysis quickly centred on the behaviour of the Route 53 authoritative servers when returning NSEC records from `slack.com`. The `slack.com` zone uses a wildcard so all names in `slack.com`, including `literallyanything.slack.com` should resolve successfully. Why then are we talking about NSEC records when there is a wildcard in the zone?

We now need to introduce the next contributory factor, namely the lack of IPv6 address records for `slack.com`. For (reasons) `slack.com` is a IPv4-only service. If you ask for a AAAA RR for `slack.com` you get a NODATA response. NODATA means that the name itself exists in the zone, but the query type you have asked for does not exist. If the zone is DNSSEC-signed, then there needs to be a way to respond with a signed NODATA response. The DNSSEC design achieves this by using the same NSEC form of response that would be used if the domain name did not exist at all. In this case the NSEC record that is generated for the signed NODATA response has the query name as the target name in the NSEC response, rather than the lexicographically prior name in the zone file that would be used for an NXDOMAIN response. In the wildcard case this is the wildcard name record. All NSEC records used

in responses (both NODATA and NXDOMAIN responses) include a bit-vector that maps to the resource record types that are defined for the label. In the case of the NODATA response this is critical, as the NSEC record then "proves" that the name exists in the zone, even if it a wildcard, and the signed resource record bit-vector "proves" that there is no such RR set for this name by listing all the resource records that do exist for the name. If a resource record type is not listed in this NSEC response, then the client can safely assume that this resource record type is not defined for this name in the zone.

Now you can lie with NSEC records, as Cloudflare has shown with their *DNS Shotgun* mechanism (https://blog.cloudflare.com/black-lies/), where the signed NSEC NODATA response claims that every resource record type, except for the one being queried, exists. But what a server should never do is return an empty bit-vector in the NSEC record. Because some resolvers, including Google's Public DNS service interpret an empty NSEC bit-vector as claiming that there are no resource records at all for that domain name. This is not a Google DNS bug. It's a perfectly legitimate interpretation of the DNSSEC specification. The problem that Slack encountered was that the Route 53 server was returning a NSEC response with an almost empty RR-type bit-vector when the wildcard entry was used to form the response and the query type was not defined for the wildcard resource. This was a bug in the Route 53 implementation.

So, to complete the issue we need to understand the behaviour of dual stack clients and *Happy Eyeballs* (RFC 8305). Clients use the DNS to figure out whether they can use IPv6, so they query first for an AAAA record, then they subsequently query for an A record. If the AAAA answer comes first, then they use it. Even if the A answer comes back up to 50ms after the AAAA answer, then they still prefer the AAAA record. But what if the response to the AAAA query primed the recursive resolved to believe that there are no records at all for this name, including A records. Then the resolver simply will not resolve the name at all.

The original bug that was exposed when the DS record for `slack.com` was placed into the `.com` zone file was a Route 53 bug, triggered by the wildcard in `slack.com`. When clients asked for a AAAA record a validating recursive resolver would receive an NSEC record to indicate that there was no AAAA records in the wildcard, but the Route 53 bug meant that the NSEC response omitted the entire RRset bit-vector for the wildcard, claiming that not only was there no AAAA record, but no RR set at all was associated with this wildcard name.

This was exacerbated by the increasing number of dual-stack end clients who have IPv6 and who are asking for AAAA records and Slack's position where the service was only defined on IPv4.

Now we add the next compounding factor. This situation was further exacerbated by recursive resolvers that use so-called aggressive NSEC caching (RFC 8198) that reuse cached NSEC responses with NSEC-type bitmap inheritance. Once the "no RR-type" field is associated with a name in the cache then it's going to be used for all further queries for this query name until the cache lifetime expires. So, it takes just one AAAA query to poison the cache and then all subsequent queries return the same NODATA no-RR signed NSEC record from the local cache for the cache lifetime.

The ability to back out of the mess by yanking the DS record was thwarted by the setting of a 24-hour cache expiry time on this DS record, so the problem persisted for as long as resolvers' caches contained this record.

Has Slack tried once more to DNSSEC-sign `slack.com`?

Not yet.

I guess it's still a tender subject for them!

Slack's report on this outage can be found at: https://slack.engineering/what-happened-during-slacks-dnssec-rollout/

NSEC Compliance Survey

The major problem at the heart of the Slack outage was an error in the NSEC type bitmap produced by the Route 53 server for wildcard domains, which falsely denied the existence of all resource record types for a domain name, causing some resolvers to reuse this cached NSEC record and falsely response that some resource record types did not exist. This is a problem for any resolver that is using a local NSEC cache (RFC 8198).

How common is this form of error? What other forms of misrepresentation exist in signed domain names? ISC's Petr Špaček combed through the Tranco 1M domain name list looking for domains with a DS record, and found 35,668 of them, or 3.57% of the total set. These names are then analysed with *dnsviz* to identify instances where the NSEC bitmap asserts that no such RR set exists for this name, yet the RR set with an RRSIG exists for the domain name. 187 instances were found with this type of failure, or 0.52% of signed domains.

These NSEC problems were located in the hosting services operated by ArvaCloud (17%), DNSimple (75%), Alibaba DNS (6%) and NIC.br (which had 1 such instance and was fixed in 2 hours!). A similar issue is also apparent in F5 load balancers (https://en.blog.nic.cz/2019/07/10/error-in-dnssec-implementation-on-f5-big-ip-load-balancers/).

Measuring DNSSEC via Web Content

How do you "measure" the DNS? If you are wanting to make a measurement claim, such as "20.3% of DNS names are signed using a red pen" then how do you collect these DNS names in for first place? The most common approaches are to use the Alexa or Tranco Top N domain names and apply your measurement to these names. But in so many ways this does not correspond to actual DNS use. For example, you might go to a popular web site, but that web page references a whole set of other resources, many of which are in different DNS domains. If you considered all these "secondary" names in web content would this change some of the conclusions that were based on measurements conducted using the simple Alexa or Tranco lists?

The crawl data of web sites is assembled at common crawl (https://commoncrawl.org/). The analysis task was a case of applying a taxonomy to the web source, looking at domains used in scripts, images, and other forms of embedded content. In this study they analysed 13.5M web sites, spread across 1,113 different TLDs. In a division that is perhaps reflective of the domain name usage, 58% of these web sites used the original .com / .net / .org TLDs, 37% used ccTLDs and just 5% used other gTLDs.

Some 70% of these websites used up to 5 different content domains, some 20% used between 5 and 10 content domains and the remainder had more than 10.

Some 7.7% of the website domain names were DNSSEC signed, using either RSA/SHA-256 or ECDSA P-256. When we also look at the DNSSEC signed status of the content domains the picture changes somewhat. Only some 1.6% of web sites have all their content domains also DNSSEC-signed, while the other 6.1% had at least one unsigned content domain.

Anycast in the DNS

These days *anycast* is a prevalent approach to scaling DNS infrastructure, but it wasn't always so. The progress of the draft that led to the publication of RFC4786, Operation of Anycast Services, in December 2006 was anything but straightforward. There were doubts about the stability of TCP sessions with anycast servers, doubts about the ability to predict the load balanced outcome across anycast instances and doubts about the manageability of the anycast server constellation. These days anycast is commonplace and we appear to be confident in managing it. For the DNS it's been incredibly useful in

scaling the DNS, and it's hard to see how we could've achieved the same outcome with unicast solutions. Increasing the number of name server records for a domain or increasing the number of A and AAAA records for name server names does not directly result in better load bearing capacity, or faster DNS responses. It increases the depth of failover alternatives, and, in theory, increases service resiliency, but at the cost of resolution speed. These days the primary engineering objectives are speed and capacity, so anycast is the logical tool of choice for the DNS infrastructure.

In the world of Top Level Domains (TLDs) 97% of these domains are served with anycast authoritative server constellations, and the populations of each of these constellations are increasing over time. It's not just the TLDs and moving one level down to SLDs, some 62% of the 187M surveyed SLDs use anycast servers.

Does this imply a diverse set of hosting solutions enterprises that use anycast services, or a small set of hosting enterprises that dominate the hosting market? The answer is self-evident in today's highly centralised Internet where the top 10 anycast domain hosting services have 92% of the anycast adoption (Table 1).

Org	SLD Count	%
GoDaddy	44,145,357	54.5%
CloudFlare	6,955,596	8.6%
1&1 IONOS	4,808,600	5.9%
DynDNS	3,883,403	4.8%
Verisign	3,878,585	4.8%
Google	3,433,523	4.2%
Uniregistry	2,376,567	2.9%
Akamai	1,451,470	1.8%
Amazon	1,068,653	1.3%
One.com	1,016,796	1.7%

Table 1 – Top 10 Anycast Hosting Providers and Domain Counts, from Sommese et al, Characaterisation of Anycast in the DNS

The advantages of anycast lie in scalability and improved response times, and the denser the anycast constellation the greater the query capacity and the faster the responses times, on average. But the finer details of how to optimise an anycast deployment to maximise query capacity and minimise response times still appears to be a dark art, rather than a precise engineering exercise.

Enhancing DNS Cookies

Some years ago, OARC meetings were dominated by various approaches to mitigate DDOS, with various forms of rate limiting, cookies and similar being used to discard what is assumed to be attack traffic and still answer conventional queries. This work has been focussed on the server, be it an authoritative server or a recursive resolver, where the server profiles the query traffic and forms models of what is considered to be abuse traffic that is can discard when it us under extreme load.

Casey Deccio presented an alternative approach to this profiling exercise, where the client can assist the server by flagging its intention to use DNS cookies in its UDP queries. Queries make using the clients IP address that do not use queries can then be assumed to be source-address spoofing attack traffic, and can be safely dropped by the server.

I must admit to a certain degree of cynicism here. The reverse DNS space has been a solution looking for a problem since its inception, and for me this is not that problem. It seems to shift the onus of work (registering their query profile in reverse DNS records) onto a set of clients who have no particular motive to perform the work in the first place. It's not the clients who have the problem here, but the servers, yet the client is being asked to do the heavy lifting. Personally, I just can't see this idea gaining traction.

DNS-OARC 36

The set of presentations for OARC 36 can be found at DNS-OARC's website (https://indico.dns-oarc.net/event/40/timetable/#20211129.detailed). The recordings of the meeting appear on the OARC YouTube channel (https://www.youtube.com/dns-oarc) a few weeks after the meeting.

Disclaimer

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

Author

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

www.potaroo.net