February 2023
Geoff Huston

To DNSSEC or Not?

The early days of the Internet were marked by a constant churn of technology. For example, routing protocols came and went in rapid succession, transmission technologies were in a state of constant flux, the devices we used to interact with the emerging digital environment were changing, and the applications we used also changed. It's therefore somewhat of a surprise to note that at least one protocol has remained relatively constant across the more than forty years of the Internet, and that's the Domain Name System (DNS) and the associated DNS name resolution protocol.

> I suspect that we could throw the transport protocols UDP and TCP into the same category of protocols that while they are not yet in their dotage, Internet-wise, they are certainly among the oldest protocols still being used intensively today. It is quite a testament to the basic design of these protocols that while the network and its use has grown by a billion-fold or more, these protocols remain largely unchanged.

The canonical specification of the DNS that is normally cited are the pair of RFCs, RFC 1034, "Domain names - concepts and facilities", and RFC 1035, "Domain names - implementation and specification", both published in November 1987. However, these two core specifications are just the tip of a rather large iceberg. One compendium of all the RFCs that touch upon the DNS lists some 292 RFCs (https://www.statdns.com/rfc/). That implies that to claim that the DNS is essentially unchanged over this forty-year period might be a bit of a stretch, but nevertheless the fundamentals of the DNS have been constant. Those additional 290 RFCs illustrate the observation that we've spent a huge amount of time and effort over these forty years focused on tinkering at the edges!

Maybe this is a bit of a harsh judgement. There have been some changes to the DNS in this period, and if I were to nominate one change to be considered as the major DNS innovation in this period, then I would have to nominate the security framework for DNS, DNSSEC.

DNSSEC adds a digital signature to DNS resource records, allowing a client resolver to determine the authenticity of a DNS answer, if they so choose. You would think that by now, with a widespread appreciation of just how toxic the Internet can be, anything that allows a user to validate the response they receive from a DNS query would be seen as a huge step forward, and we would all be clamoring to use it. Yet the take up of DNSSEC has been less than enthusiastic. Operators of recursive DNS resolvers are reluctant to add the resolution steps to request digital signatures of DNS records and validate them, and very few stub resolvers at the edge have similar functionality. Over on the signing side, the uptake of adding DNSSEC signatures to DNS zones is equally unenthusiastic. An exact count of all domain names in the Internet is a practical impossibility, so an precise calculation of the percent of DNSSEC-signed names is equally challenging, but there is a rough consensus in the DNSSEC community that just some 10% of DNS names are DNSSEC signed.

Why is this? Why is the response to DNSSEC so apparently unenthusiastic?

DNSSEC has been around long enough that ignorance of DNSSEC is no excuse. The zone administrators who do not sign their zones no doubt have their reasons why not. And the DNS resolvers that do not perform validation of DNS responses also do so deliberately, I would guess.

It seems that there is much uncertainty over whether to enable DNSSEC signing and validation. Some resolver operators appear to have embraced DNSSEC and use it as a point of principle, including the large open resolver networks operated by Google and Cloudflare. On the other hand, there are many resolvers that do not perform DNSSEC validation. Our measurements at APNIC Labs of the rate of DNSSEC validation point to a current validation rate of some 30% of users who will not load a URL if the DNSSEC signature of the signed DNS name cannot be validated (Figure 1.

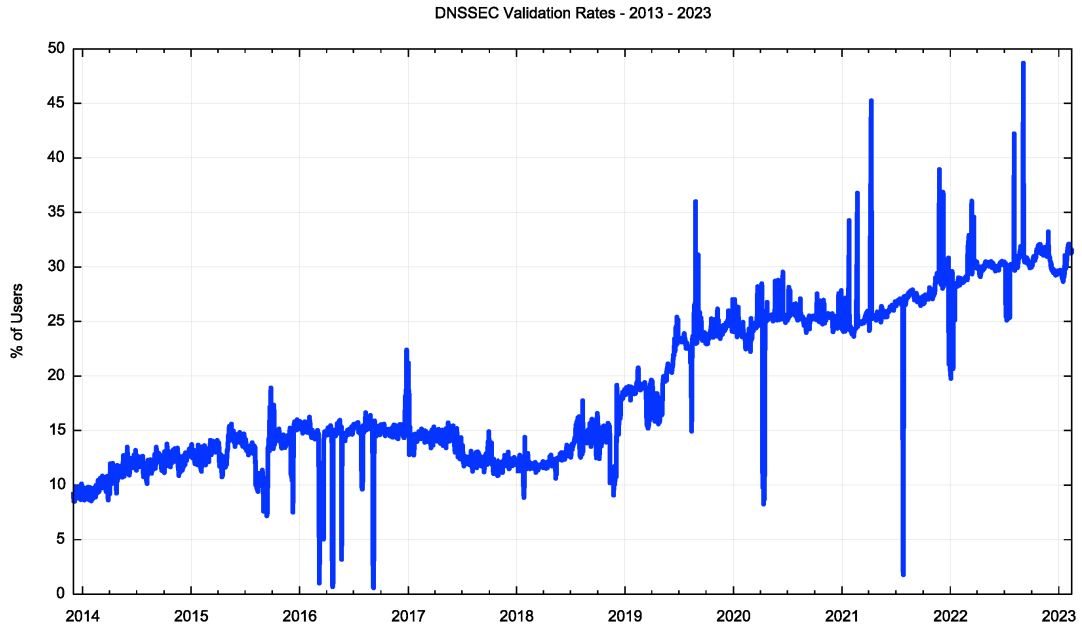

Figure 1 – DNSSEC Validation Rates (from https://stats.labs.apnic.net/dnssec/XA)

So, who's right? Is DNSSEC a good idea? Or is it nothing more than a whole lot of effort with little in the way of tangible benefit? Why aren't www.google.com, www.amazon.com or www.microsoft.com DNSSEC signed? Or, if we turn to retail banks, then why aren't www.bankofamerica.com, www.hsbc.com or www.bnpparibas.com DNSSEC-signed? Surely users of the associated online services rely completely on verifiable authenticity, and you would think that being able to validate that the DNS mapping of the service name to an IP address would be an unavoidable first step in creating the necessary assurance that their online services authentic. Yet, these names, and of course many others, are not DNSSEC-signed.

I don't think there is a clear answer to this question of whether it's a good idea to DNSSEC-sign your domain name. Yes, DNSSEC offers a more resilient and trustable DNS where users can trust that the DNS answers that they receive exactly match the current authoritative zone contents. But this comes at a cost, and the issue is whether the benefits are worth the incremental costs of adding DNSSEC signatures in DNS zones and validating these signatures in DNS responses.

Let's look at both sides of the issue of whether to sign or not to sign your DNS domain name.

Is DNSSEC worth it? The Case for "No!"

It's easy to see DNSSEC as a case of one more thing to go wrong in the DNS. For DNS zone administrators it's another set of zone administration tasks, adding key management, regular key updates, key rollover, and coordination of keys with the parent zone and delegated zones. Given that the simple elements of zone delegation and zone contents are already mis-configured across much of the DNS, then

adding elements of management of cryptographic keys and digital signatures into the set of management tasks only adds to the odds of inadvertent zone failure for those who choose to DNSSEC sign their zone.

There is the issue of how to sign a zone. Whole-of-zone signing might be straightforward in small zones, but for larger zones even the task of assembling the entire zone as a snapshot image of the zone text file and then passing a signer across the file is a logistic feat. Many larger zones are subject to a constant flow of updates, but whole-of-zone signing operations require the updates to be batched up and applied to the zone in a single pass. The signed zone is then frozen until the next scheduled update and signing pass. This operational practice may not be feasible for a large zone with a large client base with varying expectations of timeliness of changes to their entries. A number of zones have taken advantage of a *dynamic signer* where the zone's DNS servers operate as normal, and if the query requests DNSSEC credentials, then the response is passed from the server to a DNSSEC-signing front end, and the digital signature record is added to the DNS response as the final step. This is not quite so straightforward in the case of NSEC records, where it is normally expected the NSEC response lists all the resource records associated with a name as well as the next name in lexicographic order in the zone. In this case the dynamic signer may elect to provide a minimal NSEC record that meets the requirements of such negative records but describes a minimal span of just one byte on the label space, and a minimal set of resource records, as the full span information may not be available to the signer. Such deliberate lies in the NSEC records are not necessarily helpful when using these NSEC records in the recursive resolver to block related queries to non-existent labels and resource records, but the advantages to the zone administrator in such minimal NSEC answers are generally seen to outweigh the benefits in having some queries offloaded to recursive resolvers who cache these NSEC records and reuse then for related queries.

> See RFC 8198 for a description of aggressive NSEC caching and https://blog.cloudflare.com/black-lies/ for Cloudflare's description of dynamically generated minimal NSEC records.

While mentioning these NSEC records, I should also note the issue of NSEC records and zone enumeration. Conventional spanning NSEC records allow an external observer to assemble the complete list of all labels contained in a zone. This poses some issues for some zone operators who have business, commercial or security reasons for wishing to keep the zone contents private. This prompted the development of the NSEC3 record, which uses an ordering of the zone labels that is based on their hash values. This added complexity in zone signing again sounds a whole lot better than it is, as breaking this hash function is feasible with current compute power. There has been a more recent proposal to pile in more complexity onto the zone enumeration vulnerability with a proposed NSEC5 record (https://www.cs.bu.edu/~goldbe/papers/nsec5faq.html), but there has been little in the way of widespread interest in this proposal.

There is also the issue of how to add robustness to serving a zone when it is DNSSEC-signed. The conventional approach is to use a number of secondary servers that act in a way that is identical to the actions of the primary server, and they can do so by sharing a copy of the current zone file. If DNSSEC is being used, and a copy of the signed zone file has been shared then nothing changes. The secondary servers do not need to be in on the secret of the private part of the zone's Zone Signing Key (ZSK) nor the private part of the zone's Key Signing Key (KSK). However, if front-end dynamic signers are being used, then all these signing font-ends need to be configured with the ZSK private key. If all of these signers are operated by the same entity, then this is relatively straightforward, but if the zone administrator wanted greater diversity and has recruited different entities to operate secondaries, then they may want to operate these secondary systems with their own ZSK key. This can be achieved in DNSSEC by placing the collection of ZSK public keys into the zone's DNSKEY record, but the size of this DNSKEY record increases with the addition of these keys, and DNSEC response sizes are a major issue for DNSSEC. Large DNS responses impact the reliability and performance of the DNS, and DNSSEC can certainly trigger this behaviour.

These larger DNS responses also create issues with DNS performance. DNS over UDP is meant to fit DNS responses within 512 bytes. Adding DNSSEC digital signatures to a response may cause the response size to exceed this limit. DNS queriers need to use EDNS extensions to indicate their capability to handle large UDP responses. Our experience with larger DNS response that use this extension indicate that DNS responses up to 1,420 octets in IPv4 and 1,400 octets in IPv6 appear to be adequately robust, but larger responses are more likely to exceed the underlying IP Path MTU size, and the DNS response will need to use IP fragmentation. In IPv4, IP fragmentation can happen on the fly, and the router that performs the fragmentation will reproduce the IP header and rely on the receiver of the fragmented packets to reassemble the fragments and present the entire original IP payload to the upper protocol layer. This is not exactly a robust situation and fragmented DNS UDP responses have systemic weaknesses which expose the requestor to DNS cache poisoning from off-path attackers (see Appendix A of the Internet Draft draft-ietf-dnsop-avoid-fragmentation for some references to fragmentation attack vectors). The situation is perhaps worse in IPv6, in that the router that is unable to forward the packet needs to send an ICMPv6 Packet Too Big message to the original sender. As the transport protocol is UDP, the original UDP packet has been discarded, so retransmission is not possible. The resolution of this situation is to wait for the receiver to time out, and re-query. This time the sender should have cached the new path MTU value in its forwarding table and will perform IP fragmentation of the UDP packet before sending it. The fragmentation attacks already noted also apply here. So the pragmatic advice is for the DNS to avoid this situation altogether and avoid sending UDP packets that may be fragmented.

DNS has an inbuilt response to this situation, namely the use of the Truncation bit in the DNS response. The sender sends a truncated UDP packet that is a size that has a higher assurance level that it will be received. The receiver of this truncated DNS response is expected to re-query using TCP, and at this point its TCP that handles the larger response rather than IP fragmentation. The cost is delay, in that there is a round-trip time interval to send the query and receive a truncated response and a further round time trip time interval of perform a TCP handshake before it can pose the query once more. Not only does this take more time, but it also poses additional unreliability to the operation, as DNS over TCP is not universally supported in all DNS resolvers.

However, the additional time spent in handling large DNSSEC responses is not the least of the issues here. The DNSSEC validation function also takes additional time. The validating client needs the value of the ZSK used to generate the record's digital signature, and this is not provided in the original response. So, to validate the digital signature, the client needs to query the zone's DNSKEY record to obtain the current ZSK key values. But this poses a further question: how can the client validate the authenticity of the DNSKEY response it receives? This DNSKEY response has been signed by the zone's KSK, which is also stored in the zone's DNSKEY record, but the authenticity test requires more data, zs we need to establish that these are the "correct" keys and not some fake keys that have been injected by an adversary. DNSSEC requires that the client then ask the zone's immediate parent for the Delegation Signer (DS) record for this zone name. The answer is the hash of the zone's KSK, signed by the ZSK of the parent zone. To validate this signature the client needs to ask for the DNSKEY of the parent zone, and perform the same sequence of operations on the parent zone ZSK and KSK key values, and do so successively on the chain of parent zones until it reaches the root zone. The root zone KSK record is already loaded into the trusted key set of the validating resolver, and the process stops when this KSK value validates the signature of the root zone DNSKEY record. Unless an adversary has successfully stolen a copy of the private key part of the root zone KSK, which is the underlying article of faith in the integrity of DNSSEC signatures, an adversary cannot provide a synthetic signature path that leads to the stored KSK value.

This assembling of the DNSSEC signature path can represent lot of DNS queries. The need to assemble the DNSKEY and DS records of each of the parent zones would present an insurmountable time penalty were it not for resolver caching. The use of resolver caches partly mitigates this additional penalty in resolution time, but in a world where every millisecond matters DNSSEC is an extravagant time waster!

Most end systems do not perform DNSSEC validation directly. They rely on their DNS resolver to perform DNSSEC validation on their behalf, and they implicitly trust in the resolver to perform this with

appropriate levels of integrity. Of course, the issue here is that a man-in-the-middle attack between the end host and the validating resolver is still potentially effective: the end host is not validating the DNS response and cannot detect if a response is genuine or if it has been tampered with. Perhaps end systems could perform DNSSEC validation directly, but the local stub resolver cache is not as densely populated so the cache hit rate for validation queries would be lower, making the validation query time greater. Thre is also a concern that the quality of DNS implementations drops dramatically out to the edge of the network where low cost (and low quality) DNS forwarders are commonly used as part of network access devices. There is not a high degree of confidence that large DNS responses can be successfully carried through to the network's edge. So, we don't DNSSEC validate at the edge because there is a widely held suspicion that it would simply break DNS at the edge.

We generally leave the task of DNSSEC validation to the recursive resolver. When the recursive resolver function resides in the same network as the stub resolver client, then the opportunities for adverse attack are limited to some extent. When the resolver is an open resolver and located across the open Internet, then the issue of trust is even more prominent. DNS queries are carried unencrypted, and UDP is a naively credulous transport protocol. There are still a set of vulnerabilities that exist on the open path between the validating recursive resolver and the trusting stub resolver. In this scenario then there is also a question of how the recursive resolver signals a failure to validate the DNS response. Considerations of backward compatibility led to the reuse of an existing error code to indicate validation failure, namely the SERVFAIL response code. This does not specifically signal that the DNSSEC credential check failed, but that this server has failed. This error code invites the resolver client to spend more time performing re-queries. Stub resolvers will re-query using alternate recursive resolvers, while recursive resolvers will re-query using alternate authoritative servers for the zone, both of which just take more time (admittedly, is an improvement on an earlier behaviour when the resolver would exhaustively check every possible delegation path!).

DNSSEC validation is variable. When and how do resolvers perform validation? Do they perform all the queries for DNSKEY and DS records before attempting validation? Do they serialize these DNS queries or perform them in parallel? What about CNAME records? Is the first name validated before following the CNAME or is the CNAME record followed and then both names validated? How do these additional tasks and the time taken to complete them interact with existing timers in the DNS? It appears that DNSSEC causes additional query load in the DNS because of this interaction between aggressive timers in the client and time to complete validation functions that need to be performed by resolvers.

There is also some confusion over the provisioning of the DS record. This is the record in the parent zone that is the hash of the KSK public key value in the delegated child zone. In many ways the DS record is analogous to the NS delegation record, but there are some subtle differences. The NS record in the parent zone is a duplication of the same record in the child zone, and the copy that is held in the child zone if the authoritative version (which is why NS records are not signed in the parent zone but are signed in the child zone). DS records only exist in the parent zone, so they are signed by the parent zone ZSK. But their value is a hash of the child zone KSK and are therefore indirectly controlled by the child zone. It is common for zone administrators to use the same tools to manage the DS record as they use for the NS record. This is typically undertaken by a custom UI, or at best a semi-automated UI, where the risk of transcription errors is ever-present. The consequence of errors in the DS record fall into the category of a major error, as a mismatch between the DS record and the child zone KSK cause the entire zone to be considered to be invalid and unresolvable.

In response, there has been the introduction of an automated process of DS provisioning, where the child calculates the hash of its own KSK and publishes it as a signed CDS record in the child zone. The parent periodically polls the child zone's CDS record to detect new CDS values (indicating a pending roll of the child zone KSK) and validates the CDS record's DNSSEC signature through conventional DNSSEC validation. The validated CDS record is published in the parent zone (signed by the parent zone ZSK), and this then permits the child zone to complete the process of the KSK roll. This does remove the potential for various form of transcription errors in a manual DS management process, but at the expense of slightly greater operational complexity.

All of this is intended to show in detail that DNSSEC is not simple, either in design or in operations. There is much to get right and very little in the way of tolerance if errors occur. This adds to the fragility of the DNS. So, if we again ask the question about the value of DNSSEC then the answer is unclear.

There is also the really tough question: What threat is DNSSEC protecting you against? The original textbook answer is to protect resolvers against the so-called "Kaminsky attack" that injects bad data into a recursive recover's cache. DNSSEC can certainly provide this protection, but only in a limited context. As we have already observed, DNSSEC can protect the recursive resolver, but the non-validating client stub resolver is still as vulnerable as ever. Therefore, this is not a comprehensive solution to the problem. It's a step in the direction of threat mitigation by potentially protecting recursive resolvers against man-in-the middle attacks. If a DNS response fails DNSSEC validation the DNS will not inform the client what the "right" response may be. It simply withholds the response that cannot be validated and leaves it up to the client to determine what to do next.

Is the cost of this DNSSEC response commensurate with the nature of the threat? This particular DNS attack on cache integrity appears to be a rather esoteric attack vector and the use of randomised source ports in resolvers already adds sufficient randomness to make the Kaminsky guessing attack somewhat ineffective in any case.

The overall impression from this negative perspective is that DNSSEC is half-cooked. It is not a clean synthesis of security and DNS functionality, but a rather awkward and klunky tweak placed in an uncomfortable manner on top of the DNS. The incremental costs and fragility of DNSSEC far outweigh the potential benefit of risk mitigation from a rather obscure threat model.

But I suspect that I've been a bit too enthusiastic here, and maybe I've overstated the case for "No!" There are very real potential benefits for users and the Internet and a whole with the adoption of DNSSEC. Let's look at the other side of this conversation.

Is DNSSEC worth it? The Case for "Yes!"

The overall picture of security for the Internet is pretty woeful. The path between recognising a URL on a screen and clicking on it and believing that the presented result is actually the genuine service that the user had intended to access requires a fair amount of nothing less than blind trust. We are trusting that the DNS mapping of the name to an IP address is genuine, trusting that the routing system is passing the IP packets to the 'correct' endpoint, trusting that the representation of the name on your screen is actually the name of the service you intended to go to, and trusting that the TLS connection is genuine, trusting that people that you have never met, and people who you are probably not even aware of their existence, and who often only do things for money, never ever lie, to name but a few of these points of blind trust.

That's a lot of trust and many would argue it's just too much trust. Its opaque trust, in that no matter who hard you try, some of these relationships and actions where you are trusting were taken with integrity cannot be audited, or even exposed. We just hope that everyone is acting with the best of intentions.

> Yes, that's a cue for a mention of Voltaire's parody novella Candide. The character Dr. Pangloss repeats the claim that we live in the best of all possible worlds like a mantra when catastrophes keep happening to him and Candide. Derived from this character, the adjective "Panglossian" describes a person who believes that this actual world is the best of all possible worlds or is otherwise excessively optimistic. These days it seems that our unquestioning reliance on nothing more than the assumed good intentions of unseen others on the Internet is probably also "Panglossian!"

As we place more and more personal and social functions into a world of connected computers, we place more and more reliance on the integrity of the Internet. If an adversary can subvert the Internet's functions, then there is considerable potential for disruption and damage. The experience from repeated attacks so far is that adversaries, whether its talented hackers, criminal enterprises, or state actors, can subvert the Internet's operation and infrastructure and can create considerable damage. And with the much-touted Internet of Things underway we are now over-populating this already compromised environment with even more devices, and placing even greater levels of reliance on a foundation that is simply incapable of withstanding the pressure.

Don't we have a robust and trustable name infrastructure already? Doesn't that tiny padlock icon in my browser mean anything at all? The case for DNSSEC in my mind rests in the weaknesses in the existing trust model of the Internet's name infrastructure.

The existing trust structure is based on domain name X.509 public key certificates that attempt to create an association between a domain name, its holder, and a public/private key pair. If a client uses a domain name as the key for a DNS query, and uses the resultant IP address to connect to a server, then as long as the server can demonstrate that it has knowledge of the private key that is associated with this domain name then the domain name certificate should give the client the necessary assurance that it has reached a service point that the domain name holder has associated with this domain name. The IP address used to access this service is not relevant in this form of authentication. It's the subsequent security handshake that presents the client with the service's public key, the domain name certificate and a 'puzzle" that was generated with the service's private key that provides the client with the assurance that it has reached the authentic service. The client does not simply accept this proffered information as trustable information. The client is also pre-configured with a set of trusted certificate issuers (certification authorities) in the form of a collection of public keys. As long as the client can validate that the proffered domain name certificate was issued by one of these trusted certificate issuers, then it will be able to trust in the authenticity of the service point as "belonging" to this domain name.

The problem here is that all this sounds a whole lot better than it actually is. There are hundreds of trusted certificate issuers and each of these issuers is able to mint a public key certificate for any domain name. If any of these certificate issuers is compromised and issues fake certificates, then these fake certificates are indistinguishable from all the other issued certificates. If a certificate issuer is included in the client's trust collection of issuers, then the client unconditionally trusts this issuer, and accepts all its issued certificates as genuine, without qualification. So, we have a "weakest link" situation where it really does not matter how good a job is being performed by the certificate issuer used by a domain name holder. What matters is the quality of the job performed by the least reliable of the set of certificate issuers, as if they are compromised then they could be compelled to issue a fake certificate for any domain name.

There are a couple of further factors that undermine the trust in this domain name certificate framework. The first factor is the erosion of price. There has been a concerted effort to bring down the price of certificates, on the basis that providing security to online services should not be an expensive luxury that is accessible to a few, but a universally affordable commodity. This has been taken to its extreme case with the introduction of free domain name certificates provided through the automated portals operated by Let's Encrypt. Why should anyone pay for their domain name certificate when a functionally identical domain name certificate can be generated using a free service? The tests provided by these free services are functional tests where the applicant needs to demonstrate that they can add a record into the relevant domain name zone file or generate a URL page within the name space of the domain name. These free certificates are equivalent to any other domain name certificate, including in the way that an indication of certification is provided in the user interface of the client-side tool. The implication is that the ability to compromise a web site or intrude into the contents of a DNS zone file can result in the issuance of a fake domain name certificate. Supposedly this risk is mitigated by the observation that if an issuer is notified that a certificate should no longer be regarded as valid it can be revoked by the issuer.

The second factor is that revocation is not universally supported any more. Certificate revocation describes the process where a certificate issuer regularly publishes a list of certificate serial numbers that should no longer be trusted. When a client validates a certificate, it should consult the Certificate authority's (CA's) certificate revocation list (CRL), which can be found via a pointer contained in the certificate, retrieve this CRL, validate its content via its signature, and then look up the certificate's serial number in this list. If it cannot be found in the CRL then either the certificate has not been revoked, or, more accurately, it had not been revoked at the time of the CRL creation date (which is generally some days in the past). The more the number of unexpired revoked certificates the larger the CRL will get. For a large CA the workload associated with CRLs can be significant. Delivering a complete list of all revoked certificates seems to be a case of over-answering, particularly if all the querier wanted to know was the revocation status of a single certificate. Also, the generation of CRLs is not a mandatory requirement for CAs. A CA may elect to regularly publish CRLs, or may elect to publish CRLs and update them with delta CRLs, or may not publish CRLs at all! For these reasons CRLs are typically not used by end clients when setting up a TLS session. So, nobody uses CRLs. In its place we were meant to use the Online Certificate Status Protocol (OCSP). The certificate then contains an OCSP URL, and the client connects to this URL and sends it the serial number to be checked. The response is either an indication that the certificate has been revoked, or it has not been revoked.

Again, OCSP sounds better than it really is. There are privacy concerns with OCSP in having the client contact the CA, in that the CA is then aware of the identity of clients using this certificate via the source of the OCSP request and also aware of when the client is using the certificate. Not only is there a significant privacy leak each time the client needs to access the OCSP data for a certificate, but there are also performance issues with the additional time taken to generate the OCSP request and waiting for the response. This is not necessarily a single request, as a prudent client would check not only the revocation status of the certificate used by the server, but also check the revocation status of all the CA certificates used by the client to assemble the validation chain from a Trust Anchor to this certificate. It is also worth remembering that this is not necessarily a query as to the current revocation status of this certificate. The OCSP response is generally an extract from the CA's current CRL, and it reflects the certificate's revocation status at the time of the creation of the CRL, not the revocation status at the precise time of the OCSP query. It's still effectively a CRL lookup, but now the lookup using the larger CRL list is being performed by the CA, not the client.

We then turned to "stapled OCSP" where the server performs the OCSP check on behalf of the client and sends the signed OCSP response alongside the certificate. If the certificate is not revoked at the time of the CRL generation, then, as already noted, the OCSP reported status is good, which adds nothing to the information already contained in the certificate. In this case stapled OCSP is adding nothing. If the OCSP reported certificate status is revoked, then the CA is saying that this certificate should not be used. If that is the case, then why should the server convey the certificate and the OCSP status to the client and defer to the client on the decision not to proceed with the TLS connection? Why shouldn't the server simply terminate the TLS connection immediately itself? In other words, why should the server pass a revoked certificate to the client? In this case stapled OCSP is a long way around to reach the inevitable outcome of a failed TLS connection attempt. So why bother?

Is revocation worth the effort? Chrome, the major browser out there, has concluded that it's not worth the effort. So, what can we do about valid certificates that were issued under false pretences or use compromised key values? Nothing!

There is no panacea here, and every approach to certificate revocation represents some level of compromise.

We can live with long-lived certificates with high enrolment costs but only if we can support a robust and fast revocation mechanisms, which to date has been an elusive goal. CRLs and OCSP are not instant responses but they can reduce the timeframe of vulnerability arising from a compromised private key, even though there are some clear issues with robustness with both CRLs and various flavours of OCSP.

We can head down the path forged by Let's Encrypt and only use short lived certificates generated by highly automated processes (where "short" is still a somewhat lengthy 90 days!). Given their short lifetime revocation is not as big an issue, and it's possible to contemplate even shorter certificate lifetimes. If revocation lists are refreshed at one-week intervals, then a one-week certificate could simply be allowed to expire as revocation would not substantially alter the situation for relying parties.

But if we head down this path of short-lived certificates, then why do we need to bother with the X.509 wrapper at all? Why don't we take the approach of packaging OCSP responses in the DNS one step further and dispense with the X.509 packaging completely and place the entire public key infrastructure into the DNS? We could certainly go in that direction, but to go there requires DNSSEC to protect this information once it as been loaded into the DNS.

The essential differences between the security framework provided by certificates and that provided by DNSSEC can now be summarized.

The domain name certificate framework uses a distributed trust framework where a certificate can be issued by any CA. This creates a framework where the robustness of each certificate is based on the resilience of the weakest CA in the system. This has caused issued in the past. By comparison the trust framework of DNSSEC is based on a single private key, namely the Key Signing Key of the root zone.

Domain Name certificates may be revoked by the issuer, but revocation status checking has proved to be operationally challenging. The practical response has been to limit validity periods in issued certificates to a small number of months and dispense with revocation checks. The underlying DNS framework is designed to force holders of DNS information to periodically check the validity of their cached information. Cache lifetimes of the order of hours or days are feasible in the DNSSEC framework.

Domain Name certificates are a detached commentary on the delegation status of domain names. DNSSEC is represented in the DNS itself, and the security credentials become an attribute of the DNS name.

The case for DNSSEC lies for me in the observation that DNSSEC is a more unified way of adding credentials to domain names, where the properties of the credentials are closely links to the properties of the name itself. Domain Name certificates find it much more challenging to sustain this close association.

The problem for DNSSEC lies in the nature of the DNS protocol, and in particular the prevalent use of an on-demand lightweight query/response set of transactions. This approach has not handled large DNS payloads nor navigation across inter-linked relationships (as required for validation) efficiently.

Where now?

Let's not underestimate the value of a robust trustable name infrastructure. The trust infrastructure of the Web is subject to continual episodes of corrupted certificate issuance, and the efforts to introduce certificate transparency, HPKP records and CAA records are desperate and to my mind largely ineffectual measures that fail even when using a limited objective of palliative mitigation. The real question here is whether we have any more bits of string and wax that we might want to apply to this domain name PKI, or whether we are willing to head in an alternate direction and dispense with this third-party commentary on domain name control altogether.

Logically if we want to avoid this scenario of a third-party commentary on domain name tenure then it makes sense to fold the security credentials of a domain name into the DNS, and treat the name holder's public key as an attribute of the name. It appears that domain keys in the DNS (DANE) are the best response we have here to the issue, and that means we need to have a trustable DNS where users can verify DNS data as being authentic. And DNSSEC is the only way we know how to achieve that.

For use in TLS, it's possible to take the OCSP stapling approach and staple the server's public key response as a DANE resource record, a DNSSEC signature and the associated DNSSEC validation responses as a chain extension data block. It's the collection of DNS query responses that the client would've assembled if they were using the DNS directly, but in this case it's assembled by the server and passed to the client within the TLS handshake. The advantage of a DANE approach is to support a far shorter timeframe of local credential autonomy. The server who is assembling this material is limited by the DNS local copy expiration timer (TTL) and it is required to refresh the DNS data within this interval. Because it's the server, not the client, who is assembling this material, the client's identity is not being exposed into the DNS or tto he DNS servers.

More generally, are the issues of large responses in the DNS insurmountable., or are there ways around these issues? Can we avoid large DNSSEC responses yet still use DNSSEC? Surprisingly, the answer is "yes", at least to some extent. A very common cryptographic algorithm is RSA (named after the surnames of the original authors of the 1978 paper that described the algorithm, Ron Rivest, Adi Shamir and Leonard Adelman). RSA is a cryptographic algorithm based on prime number operations using modular exponentiation that does both encryption and decryption using a variable key length. A shorter RSA key is more efficient in terms of encryption and decryption but is not as robust. Longer keys are more expensive to use but offer greater robustness against efforts to break encoded data, and computers become more capable shorter keys become more vulnerable to attack.

This trend to use increasingly large RSA keys has led to effort to break away from prime number cryptography and look at other mathematical functions, including elliptical curves. The elliptical curve function ECDSA P-256 permits all of the DNSSEC resource records, namely RRSIG, NSEC(3), DNSKEY and DS records to all be under 512 bytes in length in most circumstances (the DNSKEY record during a keyroll is the exceptional case here), so for the moment the answer is "yes", we can avoid large DNSSEC responses if we use elliptical curve crypto algorithms.

> Current estimates are that ECDSA with curve P-256 has an approximate equivalent strength to RSA with 3072-bit keys. Using ECDSA with curve P-256 in DNSSEC has some advantages and disadvantages relative to using RSA with SHA-256 and with 3072-bit keys. ECDSA keys are much shorter than RSA keys; at this size, the difference is 256 versus 3072 bits. Similarly, ECDSA signatures are much shorter than RSA signatures. This is relevant because DNSSEC stores and transmits both keys and signatures."
>
> RFC 6605, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", P. Hoffman, W.C.A. Wijngaards, April 2012

> Taking the possible future use of quantum computers into account dramatically increases the resources required to prime factor 2048-bit numbers. In 2015, researchers estimated that a quantum computer would need a billion qubits to do the job reliably. That's significantly more than the 70 qubits in today's state-of-the-art quantum computers. On that basis, security experts might well have been able to justify the idea that it would be decades before messages with 2048-bit RSA encryption could be broken by a quantum computer.
>
> [Researchers have shown] how a quantum computer could do the calculation with just 20 million qubits. Indeed, they show that such a device would take just eight hours to complete the calculation. "[As a result], the worst case estimate of how many qubits will be needed to factor 2048 bit RSA integers has dropped nearly two orders of magnitude," they say.
>
> MIT Technology Review, May 2019

While elliptical curve crypto algorithms can allow DNSSEC signed responses to be carried in unfragmented DNS over UDP, the issue of the time taken to perform DNSSEC validation remains. The on-demand "just in time" model of the DNS protocol is product of the constrained computation and communications environment of the 1980's. In an environment of abundant computation and communications it's possible to contemplate a service model where the responses are pre-computed in advance "just in case" a client might ask for the information. This is the thinking behind the use of the DNS Chain Extension model that the server assembles the complete collection of the signed DNSKEY and DS resource records along with the record signature, for the zones that the server is authoritative for, just in case a client requests the DNSSEC credentials associated with a response. In this case the server can respond with the entire collection immediately, and the client can validate the signed response without any further need to perform DNS queries.

It's unlikely that this collection of information can fit into a UDP response, and it is unwise to even attempt to do so from the perspective of using this UDP behaviour as part of a DOS attack. Such a move to use DNSSEC chained extensions in responses would make a lot of sense in a context of DNS-over-TLS or DNS-over-HTTPs, or as a stapled attribute in a TLS certificate exchange to facilitate the use of DANE as a CA-pinning measure.

A refinement of the DNS error codes to explicitly signal DNSSEC validation failure would prevent the resolver re-query behaviour that we see with SERVFAIL signalling. Work is also underway to equip end hosts with DNSSEC validation capability, so that end hosts are not reliant on an untrusted (and vulnerable) connection between the host and their DNS resolver. And let's not forget that caching in the DNS is incredibly effective. The digital signatures in DNSSEC are cached in the same way as delegation and address records are held in the cache, so there will be no real time penalty for validated resolution of a signed DNS name if the relevant resource records are already held in the local cache.

Without a secure and trustable name infrastructure for the Internet, the prospects for the Internet don't look all that good. DNSSEC is not the complete answer here, but it sure looks as if it's an essential element of a robustly secure and trustable Internet. And maybe that's sufficient reason for us to adopt it. We can and should put in the technical effort to make DNSSEC more efficient and make it easier and faster to use. But we shouldn't let the perfect be the enemy of the good. There is no point in waiting for a "better" DNSSEC. We'll be waiting indefinitely, and the problems associated with a compromised digital infrastructure will persist. The alternative is to simply use what we have at hand and use our ongoing experiences to shape our further efforts to harden up both the DNS and the larger Internet infrastructure.

Conclusions

Should you DNSSEC-sign your domain name? The answer lies in understanding what you are trying to achieve in the first place. If it's critically important for you to protect the mapping of your domain names to an IP address then DNSSEC undoubtedly can do that, to some extent. If it's important to protect your DNS zone from interference and manipulation than DNSSEC can do that, again to some extent.

The qualifications here is that DNSSEC cannot counter any such third-party manipulation of DNS responses, but it can detect when such interference has occurred, and withhold the DNS response is such cases. Secondly, the assurances that DNSSEC offers as to the veracity of the provided DNS information relies on the client (or the client's agent) performing DNSSEC validation. If the client does not validate the signed DNS responses, then the client is no better off in terms of such assurance. Thirdly, the client needs to accept that the withholding of a DNS response is due to DNSSEC validation failure, rather than some transient error condition in the DNS resolution framework. Until DNS validating resolvers and their clients support the DNS extended error codes then this will always be a sore point for DNSSEC adoption. When the resolution framework encounters an error state that DNSSEC is designed to detect,

then the use of the default SERFAIL error code prompts the DNS system to take more time to perform a broader search for a server that can provide the answer that the client is seeking. And finally, as long as the stub resolver in the client host does not perform DNSSEC validation directly, then the client is still vulnerable to attacks on the DNS that can occur when the response is being passed from the validating resolver to the stub client. So that's a highly qualified understanding of the extent to which DNSSEC can provide additional protection to the client.

We can take a broader view and ask the question of where and why DNSSEC adds benefit to the Internet environment. Prior to the widespread adoption of secured transport sessions that perform server authentication, the the underlying common assumption for Internet applications and services was that if a client addresses a packet to the "correct" destination address, then it would obtain a "correct" response. Within such a framework, protecting this mapping from a name to an IP address was important. If an attacker could furnish an alternate IP address in a way that was not detectable by the client, then the client would assume that the substitute IP address was the "correct" service delivery point for this service.

The web environment has largely dispensed with this highly naïve assumption, and browsers in particularly are more stringent in demanding that the server perform a TLS handshake, requiring that the server provide credentials in the form of a domain name certificate and a signed digital object that allows the client to verify using third-party trust anchors that the service is indeed the named service that the client is seeking. In this environment the authenticity of the name-to-IP address is no longer critical. If the server can furnish the appropriate credentials, then the client will accept the server as a genuine instance of the service, irrespective of the IP address used to reach that service. If all DNSSEC can do is protect the mapping from a name to an IP address, then the service it provides is largely anachronistic in today's service environment. Perhaps all these services that choose not to DNSSEC-sign their name are making a wise choice, avoiding the pitfalls of an increasingly fragile DNS when DNSSEC signatures are folded into responses and avoiding clients spending time to validate these DNSSEC credentials. If the service is not the authentic service, then the TLS handshake will detect this attempted subterfuge in any case.

While the Domain Name Certificate Public Key Infrastructure (PKI) represents the foundation of today's security framework for the Internet, that does not mean that it is always effective in what it is trying to achieve. The widely distributed model of trust in this PKI, and the lack of CA pinning means that any individual compromised CA can undermine the authenticity of any service, and such actions will be largely unnoticed by end clients. The lack of an effective model of revocation of a certificate means that when a certificate should not be trusted, the overall majority of clients simply do not perform any form of revocation status checking. The demands of enterprise environments have meant that hosts can augment the locally maintained set of trusted CAs, which, in turn, have meant that host-based malware can create additional vectors of attack by adding rogue CAs to the local trusted CA set. The certificate system has responded with a collection of measures, such as CA pinning though HPKP records, CAA records and certificate transparency, but such initiatives have not been seen to represent effective counter-measures to these issues.

How can we "fix" these issues in the domain name PKI framework? How can we improve the robustness of the existing PKI? It's here that the DNS re-enters the conversation, and DNSSEC along with it. By using DANE (DNS-Based Authentication for TLS), the DNS can be loaded with the certificate of the trusted CA that issued the TLS certificate (or its public key), addressing the CA-pinning issue. Or the DNS can be loaded with the end-entity certificate (or its public key) that should be used by TLS, addressing both the pinning and the certificate revocation issues. Or the DNS can provide a self-signed certificate, or a public key, that should be used by TLS, bypassing all the domain Name PKI validation checks. While these DANE options can address TLS' shortcomings, they expose some new issues. To trust these DANE records it is imperative that these DANE records in the DNS are DNSSEC-signed, and the TLS client performs DNSSEC validation directly.

But it is completely unrealistic to contemplate adding any form of DNS queries into a TLS handshake. We need to use the "stapling" approach used by OCSP, where the additional information, and all the

data that should be used to validate this information, is loaded into the TLS handshake. How could we counter a "stripping" attack that removed this information from the TLS server hello packet? One proposed approach was to require DANE validation by a flag in the signed end-entity certificate.

If this was the only use of DNSSEC, and we altered the default query setting of DNS clients to clear the DNSSEC OK from conventional DNS queries we might well be in a far better place. If DNSSEC was used in a mode of a stapled set of credentials used outside of the incremental DNS query/response transactions, the pitfalls of DNSSEC would be avoided, while the performance of certificate validation in TLS handshakes would not be impacted in any significant manner.

So, with DANE, we know what we need to do to address the vulnerability issues with the TLS use of the domain name certificate framework, and this DANE approach necessarily includes the use of DNSSEC in a non-query mode. But will we all go there? Will we see TLS implementations that integrate stapled DANE and DNSSEC information. Or are we willing to continue to tolerate these fractures within the existing PKI framework, on the basis that the cost of exploits of these vulnerabilities, malicious or otherwise, represents a path of lesser effort than the path of adding a new validation framework into TLS coupled with the addition of DNSSEC into larger parts of the DNS space. I suspect that it's this question that is the key question behind the future prospects for DNSSEC.

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