August 2021
Geoff Huston

TLS with a side of DANE

Am I really talking to you? In a networked world that's an important question.

For example, where I'm located, when I look up the DNS name `www.google.com` I get the IPv6 address 2404:6800:4006:813::2004. This implies that when I send an IPv6 packet to this destination address I will reach a Google-operated server. Right? Well, most of the time that's probably a reasonable assumption but it's not always true. A packet's adventure through the Internet is way beyond my direct control and I have no idea if my packet might be captured and sent elsewhere, accidently, or maliciously. And this risk is true for every online service.

Of course, the implications for such potential misdirection vary according to the nature of the service. So let's get personal. What about my bank? When I enter the URL of my bank how do I know that the resultant session is a session with my bank?

You might think that's a silly question because for most of the time and for most users the Internet "just works". But there are times when trust turns into credulity. Exactly what are you trusting?

Let's take the DNS transforms used so-called "big four" retail banks in Australia as a good example of today's picture with the provision of services.

Commonwealth Bank of Australia:

```
www.commbank.com.au.    3599   IN      CNAME    prd.akamai.cba.commbank.edgekey.net.
prd.akamai.cba.commbank.edgekey.net. 300 IN CNAME e7049.x.akamaiedge.net.
e7049.x.akamaiedge.net.  20   IN      A        104.98.23.23
```

Westpac:

```
www.westpac.com.au.       10800      IN       CNAME    dplenhq18279.cloudfront.net.
dplenhq18279.cloudfront.net. 60      IN       A        13.224.175.48
dplenhq18279.cloudfront.net. 60      IN       A        13.224.175.62
dplenhq18279.cloudfront.net. 60      IN       A        13.224.175.22
dplenhq18279.cloudfront.net. 60      IN       A        13.224.175.58
```

ANZ:

```
www.anz.com.au.    1200      IN       CNAME    jtyncho.x.incapdns.net.
jtyncho.x.incapdns.net.     30       IN       A        45.60.126.46
```

NAB:

```
www.nab.com.au.  7200     IN       CNAME    www.nab.com.au.edgekey.net.
www.nab.com.au.edgekey.net. 300 INCNAME    e11252.x.akamaiedge.net.
e11252.x.akamaiedge.net. 20       IN       A        23.12.68.183
```

In none of these cases does the bank's DNS name map directly an IP address that is linked to servers that are part of the bank's name space. Each of these banking services use platforms operated by Akamai, or Amazon's Cloudfront, or Incapsula. No doubt there are good reasons why the banks choose to outsource their service platforms to a hosting company, including DDOS defence and cost-effective service resiliency in

a hostile environment. The days of running your own services on in-house operated infrastructure are long gone for these online service enterprises.

But the question remains: How can these cloud service platforms convince my browser that they are acting on behalf of my bank?

The way trust is inferred in this situation is through public key cryptography:

- The service provider signs a certificate signing request containing the public key part of a public/private key pair and passes this request to any one of a number of trusted Certification Authorities (CAs).

- The CA performs some form of check of the bona fides of the service provider and the domain name that want to be associated with.

- If the CA is satisfied it will sign a public key certificate with its own private key and pass this certificate back to the service provider.

When my browser opens a secure connection to the named service, using the TLS protocol, then in the initial part of the TLS the browser will tell the server the original domain name that my browser wants to establish a connection with, via the Server Name Indication (SNI) field in TLS. In response, the server provides my browser with a copy of this public key certificate and also encloses a piece of data encrypted using the associated private key.

The idea here is that you are communicating with is an endpoint that has knowledge of the private key of the key pair that the trusted CA was prepared to associate with the domain name of the service. Whatever aliasing transforms that were applied in resolving the DNS name, and wherever IP address is hosting the service, then as long as this service delivery endpoint demonstrates that is has control of the named service's private key then the browser is prepared to believe that this is the genuine service.

What could possibly go wrong?

Well, because you asked:

- We are often incapable of keeping a secret. Anyone who learns your private key can impersonate you and nobody else can tell the difference.

- The relationship or authority that a public key certificate is supposed to attest might be subverted and the wrong party might be certified by a CA.

- The CA might have been hacked, compromised, socially engineered, or otherwise coerced to issue certificates to the wrong party under false pretences. Nobody else witnesses the transaction between the applicant and the CA that results in a certificate. We just have to trust that the CA is operating with integrity.

- The architecture of the distributed trust system used by the Internet's PKI makes the system itself only as trustable as most vulnerable CA. It doesn't matter how good your CA might be, if every user can be duped by a falsely issued certificate from a corrupted CA, then the damage has been done.

- Trust Anchors (TAs) are distributed "blind" by browser vendors, and are axioms of trust: things certified by a trusted CA are automatically valid, but where a CA fails to maintain robust procedures and issues certificates under compromised conditions, we as users and end consumers of the security framework are exposed without any conscious buy-in: it just happens as a function of the existence of the CA as a point of trust in our system's software. We either have to be experts to know how to flush these out or rely on others to help us update our circle of credulity.

- When certificates are issued in error there is also the ability for the CA to revoke a certificate. However not all clients check the revocation status of a certificate, so my client might still trust a certificate that the CA has revoked. There is also the issue of how to manage the situation of a corrupted CA that issues spurious revocations as part of a denial-of-service attack.

Each time these incidents occur we castigate the errant CA. Sometimes we eject them from the trusted CA set, in the belief that these actions will fully restore our collective trust in this obviously corrupted framework. But there is trust and there is credulity, and here we've all been herded into the credulity pen.

We've seen two styles of response to these structural problems with the Internet's PKI. One is to try and fix these problems while leaving the basic design of the system in place. The other is to run away and try something completely different.

Fix it!

The fix it crew have come up with many ideas over the years. Much of the work has concerned CA *pinning*.

> The problem is that the client does not know which particular CA issued the authentic certificate. If any of the other trusted CA's have been coerced or fooled into issuing a false certificate, then the user would be none the wiser when presented with this fake certificate. With around one hundred generally trusted CAs out there, this represents an uncomfortably large attack surface. You don't have to knock them all off to launch an attack. Just one. Anyone will do. This vulnerability has proved to be a tough problem to solve in a robust manner.
>
> What the fixers want is to allow the certificate subject to be able to state, in a secure manner, which CA has certified them. That way an attacker who can successfully subvert a CA can only forge certificates that were issued by this subverted CA. Obviously it doesn't solve the problem of errant CAs but it limits the scope of damage from everyone to a smaller subset.

The various pinning solutions proposed so far rely on an initial leap of faith in the form of "trust on first use".

One deployed pinning solution is effective, namely the incorporation of the public key fingerprint for a number of domain names into the source code of the Google Chrome browser. While this works for Google's domain names when the user is a Chrome user, it obviously doesn't work for anyone else, so it's not a generally useful solution to the pinning problem inherent in a very diverse distributed trust framework.

HTTP Public Key Pinning (HPKP) (RFC7469) approach required a hash of the 'real' public key to be included in the delivered web content. If you trusted the web content you could trust the key. If you trusted the key, then you could trust the web content. As the RFC itself conceded it's not a perfect defence against MTIM attackers, and it's not a defence against compromised keys. If an attacker can intrude in this initial HTML exchange, then the user can still be misled.

Rolling your pinned key also implies rolling the content that references the key or upgrading the entire set of deployed applications that incorporate the hash of the key.

Certificate Transparency uses a different approach that requires all valid certificates to be published in a public log. Without evidence of such publication, in the form of a Signed Certificate Timestamp, which the browser is supposed to validated either through OSCP stapling in a TLS extension or from a SCT embedded in the certificate, then the certificate should not be regarded as trustable. This does not directly address the pinning problem per se, nor does it address the rogue CA problems, but it is intended to expose rougue actions that would otherwise go undetected.

We can confidently predict that errant CA incidents will continue to occur. But the worrisome observation is that the CA space is changing. Rather than many CAs each with a proportionate share of the total volume of issued certificates we are seeing aggregation and consolidation in the CA space. If the strength of the CA system in this PKI is the number and diversity of CAs then this particular strength is being rapidly eroded and the CA space is rapidly centralising toward just one CA (https://bit.ly/3du1TAb). Obviously, this CA is the Let's Encrypt CA, and their product is short term free certificates based on automated proof-of-control tests.

But perhaps this value destruction in the CA space is not only inevitable but long overdue. Users are generally completely unaware which CA issues a certificate, and a good case can be made that this is indeed something they shouldn't need to care about anyway. If the user can't tell the difference between using a free CA using automated checks and an expensive CA that may (or may not) have performed more rigorous checks they why bother with these expensive tests? If our entire security infrastructure for the Web PKI is based on the convenient fiction that spending more money to obtain precisely the same commodity item somehow imbues this item with magical security powers, then this PKI is in a truly bad place.

> It's my opinion that the "fix it" crew has failed, but they are just not ready to admit it yet! Even if the money is fleeing out the door due to free certificates there is still a heap of invested mind share in the PKI, and a lot of people who are still willing to insist that the Web PKI certificate boat is keeping itself above the water line. In my view they're wrong, but they're also keen to deny that there's any problem even as their vessel sinks down to the depths!

Run Away!

The DNS is truly magical - its massive, its fast, its timely, it seems to work despite being subject to consistent hostile attacks of various forms and various magnitudes. And finally, after some 20 years of playing around, we have DNSSEC. When I query your DNSSEC-signed zone I can choose to assure myself that the answer I get from the DNS is authentic, timely and unaltered. And all I need to trust to pull this off is my local copy of the root zone KSK key. Not a hundred or so trust points, none of which back each other up, creating a hundred or more points of vulnerability, but a single anchor of trust.

The DNS is almost the exact opposite of the PKI. In the PKI each CA has a single point of publication and offers a single service point. The diverse nature of the Internet PKI means that CAs do not back each other up and avail themselves of massively replicated service infrastructure. When I want to phrase an OCSP query I can't ask any CA about the revocation status of a given certificate. I have to ask only the CA that issued the certificate. The result is many trusted CAs, but a very limited set of CA publication points, each of which is a critical point of vulnerability. The DNS uses an antithetical approach. A single root of a name hierarchy, but with the content massively replicated in a publication structure that avails itself of mutual backup. DNSSEC has a single anchor of trust, but with many different ways to retrieve the data. Yes, you can manage your zone with a single authoritative server and a single unicast publication point and thereby create a single point of vulnerability, but you can also avail yourself of multiple secondary services, anycast-based load sharing, short TTLs giving the data publisher some degree of control over local caching behaviours.

> The single trust model was in fact a goal of the authors of Internet X.509 PKI specification (RFC3280). They apparently didn't expect an explosion of many points of trust and had hoped the IETF was going to step up and become some kind of de-facto community managed point of trust for most public Internet contexts. In retrospect this was a case of naively wishful thinking that typified much of the formative days of the Internet!

Let's look at DANE.

Let's not put these public keys in a PKI as a derivation of trust. Instead, let's put these public keys in the DNS. After all, the thing we are trying to associate securely is a public key to be used in TLS with a domain name. Why must we have these middleware notaries called CAs? Why not just put the key in the DNS and eliminate the intermediary?

> It is a venerable adage in Computer Science that any problem can be solved (or at least pushed to be a different problem!) by adding another layer of indirection. The IETF is nothing, if not experts at adding extra complexity, trowelling it on as an added complex layer of indirection.

DANE was always going to be provocative to the CA industry, and predictably they were vehemently opposed to the concept. There was strong resistance to adding DANE support into browsers: DNSSEC was insecure, the keys used to sign zones were too short, but the killer argument was "it takes too much time to validate an DNS answer". Which is true. The query intense DNSSEC validation process operated at a time scale that set new benchmarks in slow application behaviour. It wasn't geologically slow, but it certainly wasn't fast either. No doubt a DNSSEC validator could've been made a little faster by ganging up all the DNSSEC validation queries and sending them in parallel, but even if it did this there would still be a noticeable time penalty to perform DNSSEC validation.

In response to this criticism, the DNSSEC folk came up with a different approach to validation.

Rather than parallel queries, they proposed DNSSEC chained responses as additional data (RFC7901). This approach relies on the single DNSSEC trust anchor. Each signed name has a unique validation path so the queries to retrieve the chain of interlocking DNSKEY and DS records are predictable. It's not the queries that are important, nor to whom these queries are addresses. What's important are the responses. Because all these responses are themselves DNSSEC-signed it does not matter how the client gets these responses as DNSSEC validation will verify that these are authentic. It's quite feasible for an authoritative server to bundle these responses up together and hand them back with the response to the original query as additional data. Chaining is a nice idea as it removes any additional query overhead for DNSSEC validation. However, it's likely that these chained responses will be large, it becomes a point of strain in creating very large DNS responses that rely on UDP fragmentation. Yes, UDP truncation and TCP fallback can work, but that adds a further 2 round trip times for the DNS transaction and if the entire point of the exercise was to make all this faster, then it hardly appears like forward progress! However, with the current fascination with TLS-variants of DNS-over-TLS (DoT) and DNS-over-HTTPS (DoH) and DNS-over-QUIC (DoQ), adding a chained DNSSEXC validation package as additional data in a TCP/TLS response would be quite feasible.

However, the DNS has gone into camel-resistance mode these days and new features in the DNS are being regarded with suspicion bordering on paranoia. So far, the DNS vendors have not implemented RFC7901 support, which is a shame because eliminating the time penalty for validation makes the same good sense as multi-certificate stapled OCSP responses (RFC 6691 and RFC8445). It's often puzzling to see one community, TLS, say that a concept is a good idea and see the same concept be shunned in another community, namely the DNS developers.

But that's not all. It is argued that within the DNS resolution process DNSSEC validation is commonly implemented in the wrong way, as validation is commonly performed in the recursive resolver and not directly within the end client's system in the stub resolver). The problem here is that the end client has no reason to implicitly trust any recursive resolver, nor are there any grounds whatsoever to believe that an open unencrypted UDP exchange between a stub resolver and recursive resolver is not susceptible to a MITM attack. What we are doing today with DNSSEC validation in the DNS is just the wrong mode they claim, and there is a strong element of truth here. A more robust form of implementation of DNSSEC validation would have every endpoint performing DNSSEC validation for themselves.

> Maybe the DNS Camel folk have a point and perhaps we should try and spread the load. Why not have the application that wants to use DNS-provided data perform their own independent validation of this data? That way we can circumvent the issues with validation only being performed in DNS recursive resolvers and at the same time circumvent the issues with DNS and transport protocols. This appears to be the motivating thought process that lies behind the approach described in RFC9012.

RFC 9102 - TLS DNSSEC Chain Extension

This is a document that was originally posted as an individual Internet Draft in 2015 (https://datatracker.ietf.org/doc/html/draft-shore-tls-dnssec-chain-extension) that was submitted to the IETF's TLS Working Group, and adopted as a Working Group draft a year later (https://datatracker.ietf.org/doc/html/draft-ietf-tls-dnssec-chain-extension-00). However, progress within the TLS Working Group appeared to grind to a halt in 2018, after 7 revisions of this working draft. The document was then further developed over the ensuing three years (https://datatracker.ietf.org/doc/html/draft-dukhovni-tls-dnssec-chain) and published via the RFC's Independent Submission process as an experimental specification, RFC9102.

Placing the DNSSEC validation chained response into the TLS handshake can circumvent the issues with DNS transports and coping with large responses, and in theory directs the information directly to an intended client ("Here's why you can trust this proffered public key"). However, DNS transport issues and implicit trust in validating recursive resolvers were not the entirety of concerns with DNSSEC and simply addressing these DNS transport issues by removing them from the DNS still leaves some critical questions to be answered.

It is claimed that DNSSEC's commonly used crypto is too weak. There is a common belief, that everyone uses RSA-1024 to sign in DNSSEC with SHA-1 hashes, and these days that's not a very strong crypto setting. While we are looking at weaknesses, we should note that there is no explicit form of revocation of bad data in the DNS, and no clear equivalent of transparency associated with the signing of DNS zones, nor of the derivation of authority for the publication of DS and NS records in parent zones in the DNS. While Certificate Transparency may be considered to be a relatively weak security measure in the Web PKI, there appears to be no such equivalent of even that in the DNS. Bad deeds in the DNS can go largely unnoticed and unremarked.

There is the problem with stapled DNSSEC chain data that a man-in-middle can strip the stapled TLS extension as there is no proof of existence for TLS extensions. RFC9012 attempts to mitigate this issue through the use of a `ExtSupportLifetime` element of the TLS extension. Its value represents the number of hours that the TLS server commits to serving this extension in the future.

Where Now?

It's unlikely that RFC9102 represents a breakthrough of this impasse between the Web PKI model of trusted CAs and the DANE/DNSSEC model of placing the entirety of the trust infrastructure into the DNS.

It's possible that the Web PKI proponents have a good point when they argue that perhaps it's unwise to pin the entire Internet security framework onto a single key, the DNS KSK root zone key. Maybe it might be more resilient to use more than one TA so that we are not vulnerable to a single point of potential failure.

On the other hand, the Web PKI has largely adopted automated certificate issuance based on DNS proof-of-control tests, and the underlying reliance in a largely opaque transaction between the CA and the DNS in validating a certificate signing request is little different in the DANE/DNSSEC environment. Perhaps the only difference here is that the client is directly performing the validation of the data when using stapled DNSSEC validation data, rather than merely validating an otherwise untestable attestation made by the CA that they performed such a DNSSEC validation operation at some point in the past and the client should simply believe the CA's attestation.

Neither situation seems like a truly comfortable position to be in.

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