The ISP Column
A monthly column on things Internet

March 2020
Geoff Huston
George Michaelson

Insecurity

A couple of weeks ago I wrote an article about some issues with the Internet's Public Key Infrastructure. In particular, I was looking at what happens if you want to "unsay" a public key certificate and proclaim to the rest of the Internet that henceforth this certificate should no longer be trusted. In other words, I was looking at approaches to certificate revocation. Revocation is challenging in many respects, not the least of which is the observation that some browsers and platforms simply do not use any method to check the revocation status of a certificate and the resultant trust in public key certificates is uncomfortably unconditional.

I've had a number of conversations on this topic since posting that article, and I thought I would collect my own opinions of how we managed to create this rather odd situation where a system designed to instil trust and integrity in the digital environment has evidently failed in that endeavour.

I should admit at the outset that I have a pretty low opinion of the webPKI, where all of us are essentially forced to trust what is for me a festering mess of inconsistent behaviours and some poor operational practices that fail to even provide the palliative veneer of universal trust, let alone being capable of being a robust, secure and trustable framework.

You've been warned: this is a strongly opinionated opinion piece!

Geoff

We need a secure and trustable infrastructure. We need to be able to provide assurance that the service we are contacting is genuine, that the transaction is secured from eavesdroppers and that we leave no useful traces behind us. Why has our public key certificate system failed the Internet so badly?

Is cryptography letting us down?

It doesn't appear to be the case. The underpinnings of public/private key cryptography are relatively robust, providing of course that we choose key lengths and algorithms that are computationally infeasible to break.

This form of cryptography is a feat worthy of any magical trick: we have a robust system where the algorithm is published, and even one of the two keys is published, but even when you provide both of these components and provide material that was encrypted with this algorithm using the associated private key, this body of data still makes the task of computing the private key practically infeasible. It's not that the task is theoretically impossible, but it is intended to be practically impossible. The effort to exhaustively check every possible candidate value is intentionally impractical with today's compute power and even with the compute power we can envisage in the coming years.

This bar of impracticality is getting higher because of the continually increasing computational capability, and with the looming prospect of quantum computing. It's already a four-year old document, but the US NSA report published in January 2016 (NSA Suite and Quantum Computing FAQ) proposes that a secure system with an anticipated 20 year secure lifetime should use RSA with key lengths be 3072 bits or larger and Elliptical Curve Cryptography using ECDSA with NIST P-384.

Let's assume that we can keep ahead of this escalation in computing capability and continue to ensure that in our crypto systems the task of the attacker is orders of magnitude harder than the task of the user. So it's not the crypto itself that is failing us. Cryptography is the foundation of this secure framework, but it also relies on many other components.

It's these other related aspects of the PKI infrastructure that are experiencing problems and issues. Here's a few:

- 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 certification authority. And we've seen instances where trusted Certification Authorities have been hacked, compromised or coerced to issue certificates to the wrong party under false pretences.

- The architecture of the distributed trust system used by the Internet's PKI makes the system itself only as trustable as the worst performing Certification Authority (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.

- For many years these domain name certificates were orders of magnitude more expensive than domain name registrations, yet the system was continually undermined by poor operational practices with a result that these expensive instruments of trust were in fact untrustable.

- Trust Anchors are distributed "blind" by browser vendors, and are axioms of trust: things certified by a trust anchor 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 their trust anchor in our systems software. We either have to be experts to know how to flush these out, or rely on others to help us update.

- Not all trusted CAs operate as diligently as our implicit trust in each and every one of their actions requires of them. There was the episode of the issuing of fake certificates for several Google domains by an Egyptian intermediary using CNNIC as the trusted CA in order to eavesdrop on Egyptian citizens (https://security.googleblog.com/2015/03/maintaining-digital-certificate-security.html). There was the issue when Symantec issued certificates that had never been requested (https://security.googleblog.com/2018/03/distrust-of-symantec-pki-immediate.html). And of course there was the hacking of Diginotar's CA back in 2011 (https://blog.mozilla.org/security/2011/09/02/diginotar-removal-follow-up/).

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. 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.

Let's Fix this Mess!

The fix 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. A trusted CA has issued this certificate: good enough, so lets proceed! 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. Any one. This vulnerability has proved to be a tough problem to solve in a robust manner.

The PKI structure we use requires us to implicitly trust each CA's actions all of the time, for all of the CA's in the trust collection. That's a lot of trust, and as we've already noted that trust is violated on a seemingly regular basis. So perhaps what we would like to do is to refine this trust. 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. This approach is termed *pinning*. The various pinning solutions proposed so far rely on an initial leap of faith in the form of "trust on first use".

HTTP Public Key Pinning (HPKP) (RFC7469) enjoyed some favor for a while, but it has since been deprecated. The 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 you could trust the web content. Spot the problem? 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.

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.

Even if the pinning issue can be solved don't forget that pinning does not fix the problem of errant CAs. 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. Taken to the extreme to illustrate the problem here, if there was only one CA left in the marketplace, then pinning would be useless! We are not at this extreme position yet. But we are inexorably heading there (https://bit.ly/3du1TAb). It's a rather odd race condition here that is illustrative of the rather demented state of the Internet PKI itself, namely a race to see if we can devise some secure form of CA pinning before the CA market has consolidated to the point where any form of CA pinning is completely useless!

The fix crew also came up with Certificate Transparency (RFC6962). The idea is that all issued certificates should be logged, and the log receipt attached to the certificate. Users should not trust a certificate unless there is a log receipt attached to the certificate. A fraudulently issued certificate would not be accepted by a user unless it also had a duly signed log receipt. So even though a bad actor might be able to coerce a CA to issue a fake certificate, to ensure that users will trust this certificate the bad actor will still have to log the certificate and attach the log receipt to the certificate in order to have the intended victim(s) accept the certificate. Each log entry is a certificate and its validated certificate chain. The logs are Merkle Tree Hash logs so that any form of tampering with the log will break the Merkle chain. The receipt of lodgement in one or more transparency logs is attached to the certificate as an extension. All this is intended to produce the result that an incorrectly issued certificate will be noticed. Users should not accept certificates that do not have an attached log receipt. A log may accept certificates that are not yet fully valid and certificates that have expired. As a log is irrevocable, revoked certificates are also maintained in the log.

Again, like HPKP, all this sounds far better than it really is. The case of Symantec certifying `example.com` is a good illustration as to why this approach has its weaknesses. It took 6 months for someone to notice that particular entry in the transparency logs! Yes, that's 6 months! As long as attacks extend over weeks or months then these transparency logs might be useful, but in a world where an attack takes just a few minutes and where the attacker really doesn't care about the trail they leave behind (https://go.icann.org/33EyD4U), these certificate transparency logs are again merely palliative measures.

The fix crew attacked the weak enrolment processes in certificates by creating a more rigorous form of enrolment termed "Extended Validation" certificate. Aside from being a cynical exercise on the part of the certificate industry to create a more expensive class of certificates, these EV certificates appear to have been a complete failure. Users hardly notice the lock icon in the browser bar, and whether the lock is green yellow or a shade of chartreuse is completely unnoticed. the idea of making the certificate's subject undertake more work and spend a lot more money to generate a subtly distinguished public key certificate that produces invisible results for end users seems like a bonanza for some CA's, but a dud deal for everyone else. EV is indeed dead! (https://bit.ly/39cGfNe)

And then there's Let's Encrypt who took the exact opposite path to try and fix this mess. Instead of expensive certificates that have a high touch enrolment procedure, Let's Encrypt went the other way with plentiful, free short-lived certificates issued through a fully automated process. Their hearts are clearly in a good place. Security should not be a luxury item but a universally affordable high-quality commodity. These are laudable sentiments. But that does not necessarily mean that the Internet is a better place as a result. It's not that other CA's hadn't fully automated their enrolment process, it's just that Let's Encrypt went there openly. The obvious outcome is that Let's Encrypt is destroying any residual value in supposedly "high trust" long term certificates by flooding the Internet with low trust (if any) short term certificates. The proof of possession tests for such certificates are readily circumvented through either DNS attacks or host attacks on the web server systems. The counter argument is that the certificates are short-lived and any damage from such a falsely issued certificate is time limited. These certificates are good enough for low trust situations and nothing more, insofar as they provide good channel security, but only mediocre authenticity. But we are now dominated by the race to the bottom and these low trust certificates are now being used for everything, including fast attacks. After all, it's not the CA you are using that determines your vulnerability to such attacks, but the CA that the attacker can use. A cynic might call this move to abundant free certificates with lightweight enrolment procedures a case of destruction from the inside.

But perhaps this value destruction in issuing certificates 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 and an extortionately expensive CA then what's the deal? If our entire security infrastructure based on the convenient fiction that spending more money to obtain precisely the same commodity item somehow imbues this item with magical powers then the PKI is in a truly bad place.

No matter how hard the "let's fix this" crew try, the window of vulnerability of fraudulently issued certificates is still around a minimum of a week, and the certificate system is groaning under even that modest objective. It looks pretty much as if the fix crew has failed. 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 PKI certificate boat is still keeping itself above the water line. They're wrong, but they're keen to deny that there's any problem even as their vessel plummets down to the depths!

Run Away!

The run away crew headed to the DNS.

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 name 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 tenet, 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 open-Internet contexts. This 'one size fits all' model for the entire X.509 world was never going to be accepted by banking and finance (who had already formed their closed group for credit cards) or the military (who had already adopted PKI for armed forces identity cards) or governments, but for common use amongst users of Internet services, it would have been interesting had it become true.

So DANE.

Let's put these public keys in the DNS. After all, the thing we are trying to associate securely is a TLS public key with a domain name. Why must we have these middleware notaries called CAs? Why not just put the key in the DNS?

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 and 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. Any user of CZNIC's TLSA validator extension in their browser found that the results were hardly encouraging as the DNSSEC validation process operated at a time scale that set new benchmarks in slow browsing behaviour. It wasn't geologically slow, but it certainly wasn't fast either. No doubt the validator could've been made faster by ganging up all the DNSSEC validation queries and sending them in parallel, but even if it did this the additional DNS round trip time would still have been noticeable.

The DNSSEC folk came up with a different approach. 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, and it's not the queries that are important, it's the responses. Because all these responses are themselves DNSSEC-signed it does not matter how the client gets these responses - DNSSEC validation will verify that these are authentic, so it's quite feasible for an authoritative server to bundle these responses up together with the original query. It's a nice idea as it cuts the DNSSEC validation overhead to 0 additional RTTs. The only issue is that it becomes a point of strain to create very large UDP responses, because the Internet is just a little too hostile to fragmented UDP packets. DNS over TCP makes this simple, and with the current fascination with TLS-variants of DNS over TLS (DOT) and DNS over HTTPS (DOH), adding a chained 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 (the TLS folk) say that a concept is a good idea and see the same concept be shunned in another (the DNS folk).

Then came DANE plus DNSSEC chain stapling as a TLS extension, similar to OCSP stapling. The fix folk were vehemently opposed. They argued that DNSSEC is commonly implemented in the wrong way (DNSSEC validation is commonly implemented in the recursive resolver not with the client's system in the stub resolver). The problem with today's model of DNSSEC validation 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. So 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. Every end point needs to perform DNSSEC validation for themselves.

We think that DNSSEC validation could scale from a few thousand recursive resolvers performing validation to a few billion end clients performing validation as the additional load would be absorbed by these same recursive resolvers. But that's not the only problem of scaling up the system to reach all the endpoints. For example, is a KSK roll still feasible when there are a few billion relying parties that need to track the state of the transition of trust from one key to the next?

But the current model of misplaced trust is not the only criticism of DNSSEC. DNSSEC's crypto was too weak, they say. There is a common belief, that everyone uses RSA-1024 to sign in DNSSEC and these days that's not a very strong crypto setting. 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. None of these are in and of themselves major issues, although the stripping issue is substantive and would require some signalling of existence in the signed part of the certificate, but it looks strongly that the PKI folk want a

PKI solution, not a DNS solution and the DNS folk have largely given up trying to convince the PKI and browser folk to change their minds. Firefox and Chrome continue to follow the fix it path.

The TLS DNSSEC Chain Extension never got past a draft (draft-ietf-tls-dnssec-chain-extension-07) because the DNS proponents of that approach appeared to come to the realisation that the PKI/browser folk were just too locked in to a PKI-based approach ("pig-headed" comes to mind) and the PKI folk were convinced that patching up the increasing mess of PKI insecurity was a "better" approach than letting the DNS and DNSSEC into their tent.

But maybe the PKI folk have a good point. Maybe it's unwise to pin the entire Internet security framework into a single key, the DNS KSK root zone key, and hang all Internet security off this. Maybe it might be more resilient to use more than one approach so that we are not vulnerable to a single point of potential failure. Maybe we shouldn't ignore the constant bleating of enterprises (and one or two national environments) who want forced HTTPS proxies so that they can spy on what their users are doing. After all, they argue, deliberately compromised security for the "right" motives is not a compromise at all!

Scaling is Hard

The fundamental problem here is not (as was said at the start) the mathematics behind the cryptography. The problem is the organizational dynamics of managing these systems at scale, in a worldwide context. We not only have a distribution and management problem; we have different goals and intent. Some people want to provide strong hierarchical controls on the certificates and keys because it entrenches their role in providing services. Some want to do it because it gives them a point of control to intrude into the conversation. Others want to exploit weaknesses in the system to leverage an advantage. But end users are simple. Users just want to be able to trust that the websites and services that they connect to and share their credentials, passwords and content with are truly the ones they expected to be using. If we can't trust our communications infrastructure, then we don't have a useful communications infrastructure.

What a dysfunctional mess we've created!

Disclaimer

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