DNSSEC Deconstructed

by David Kane-Parry, 18F

A Brief History Of DNS & DNSSEC

It is easy to look back at the naivete of the Internet's early infrastructure. Depending on which sources you look at[1][2][3], either in 1989 or 1990 Steven Bellovin noticed that services like rlogin (which we have since replaced with ssh) were "fundamentally flawed" for their "reliance on host addresses or host names for authentication". At the time, however, UC Berkeley's Computer Systems Research Group (which was funded by DARPA to improve AT&T's Unix operating system) instead settled for doing address-to-name *and* name-to-address DNS queries, under the mistaken impression that an attacker could not forge both an DNS name record *and* a DNS address record.

In 1993, Bellovin followed up with an explanation as to how DNS cache poisoning would work. In a nutshell, a DNS server under the attacker's control could, when queried for a DNS name record, also provide a DNS address record. Both these records would be cached by the victim's local DNS resolver, and so when rlogin would ask for the address record for the attacker's host, it would be given the attacker's record from the cache instead of from the authoritative DNS server for that host.

Bellovin emphasized again in 1995 that "the problem here is not DNS". Foretelling the replacement of rlogin with ssh, he wrote, "If strong (i.e., cryptographic) mechanisms were used, people playing games with [DNS] would be notable solely for their nuisance value".

Then in early August of 2008, Dan Kaminsky publicly disclosed a method for an attacker to poison DNS caches without a DNS server under their control. By flooding a DNS resolver with responses to a query the attacker had asked it to make, each response containing a unique guess as to what the acceptable query ID is, the attacker could bet that their response would be accepted by the resolver before the authoritative nameserver's response was received. This particular method was effectively fixed by randomizing the source port used by the server, so that between the query ID and source port, the attacker was faced with more than twice the original space to search.

Nonetheless, while it had been over a decade since Bellovin first called attention to the untrustworthiness of DNS, in that time the Internet had grown into the public and commercial

[1] https://www.nlnetlabs.nl/projects/dnssec/history.html
[2] https://wiki.tools.isoc.org/DNSSEC_History_Project
[3] https://en.wikipedia.org/wiki/Domain_Name_System_Security_Extensions#History

sphere we know today. Later that month, OMB would issue OMB Memo M-08-23[4] requiring the government deploy DNSSEC.

The Limitations Of DNSSEC

The effort to take Bellovin's advice to heart and employ cryptographic mechanisms was formalized in 1997 when RFC 2065, "Domain Name System Security Extensions" was published by the IETF, and has undergone significant revisions ever since then as more and more issues have been found; some issues have still not been addressed. For the sake of this document, it will limit itself to DNSSEC as it stands today, per RFC 4033 "DNS Security Introduction and Requirements" also known as "DNSSEC-bis".

The problem began with the shortcut of trying to employ the cryptographic mechanisms in DNS. Recall that services that were trusting DNS had been doing so in error, for it was not designed to be trustworthy. Here, the path forks: do we re-engineer our services to not trust DNS, or do we try to re-engineer DNS to be trustworthy? Re-engineering DNS can only get us so far because the scope of DNS is limited; a theoretically perfectly secure DNS utilized by a perfectly insecure rlogin is still vulnerable to all of the attacks that necessitated its replacement by ssh, and perfectly insecure HTTP is still vulnerable to all of the attacks that necessitated its tunneling via TLS.

Chief among the security properties of those two protocols is that they do not require trust in lower-level protocols like DNS or even in IP or UDP. Both protocols establish point-to-point security based on verifiable key material, and so if people play games with DNS while these protocols are employed, it is only for the nuisance value.

DNSSEC, in a most confusing contrast, requires a chain of trust across multiple points but does not include the "last mile" chain to the requesting client. While each of the DNS servers involved in resolving a query must use verifiable key material, no material is available to the client to establish trust that the response to their query is coming from the server they queried. Instead, DNSSEC-aware servers include an Authenticated Data (AD) bit to indicate whether they were able to validate the signatures for all of the data in their response. It is then up to the resolver to trust the response or not, effectively moving the cache poisoning problem from the server to the resolver without actually fixing it.

This design branches into problems in two different directions. In one direction, we have the failure modes of DNS servers in the event that signatures cannot be validated. For servers that respond with unsigned domains, their responses will be carried back to the resolver; for servers that respond with invalidly signed domains, this will cause a SERVFAIL error and the client will be unable to proceed. This disparity in behavior is an attractive nuisance, enabling attackers to both continue to forge unsigned responses for targeted domains, with the expectation that the

[4] https://georgewbush-whitehouse.archives.gov/omb/memoranda/fy2008/m08-23.pdf

client will ignore the AD bit or lack the means to surface its status to the relying application, as well as enabling attackers to provide forged invalidly signed responses for targeted domains, with the expectation that servers will return SERVFAIL to their clients. This kind of denial of service doesn't even need to be intentional. DNSSEC already has an illustrious history for being the root cause of many outages thanks to configuration changes that resulted in signature validation failures. A near-comprehensive list can be found here[5].

DNSSEC is a vector for denial of service from another vector as well. Distributed Denial of Service attacks typically attempt to overwhelm their target through sheer force of combined bandwidth; the target has a bandwidth peak of X Gbps and by adding more and more nodes into their net the DDoS perpetrators hope to exceed X. DNSSEC, in this context, can also be leveraged as an amplifier, since a small-sized request from a client will trigger a much-larger sized response from the server. By reflecting their small requests, attackers can direct DNSSEC-enabled servers to deliver their much larger responses to another target, enabling a swarm of DDoS nodes to leverage much more bandwidth than their numbers would strictly allow. In fact, this exact situation befell cspc.gov[6].

DNSSEC also signs responses when the domain does _not_ exist, but since the non-existent domains that clients might ask about can't be known in advance, the only way to handle them would be to keep the signing key online for the server to use as needed. For organizations who have adopted DNSSEC despite the above issues, keeping the signing key online is an understandably unacceptable risk. Instead, what DNSSEC does is return a response that in effect says "the domain you asked about, which would exist between these two other domains you didn't ask about, doesn't exist". This effectively removes the need of keeping the signing key online, but at the cost of leaking possibly internal domain names to external clients. After this flaw was acknowledged, the response was modified to cryptographically hash the domain names, but advances in GPU and ASIC hardware have made it cost-effective to brute-force those hashes in offline dictionary attacks.

More could be said about the cryptographic lapses in DNSSEC, but they are attempting distraction from the fundamental issues. Yes, DNSSEC requires support of SHA-1, which is cryptographically broken[7], and NIST has directed agencies to stop using it in digital signature schemes[8], but requiring better algorithms will not address all or even any of the preceding issues. It is noted only as further indication of how disconnected DNSSEC is from the kind of solution we want and need.

---

[5] https://ianix.com/pub/dnssec-outages.html

[6] https://blog.cloudflare.com/how-the-consumer-product-safety-commission-is-inadvertently-behind-the-internets-largest-ddos-attacks/

[7] https://shattered.io/static/shattered.pdf

[8] http://csrc.nist.gov/groups/ST/hash/policy.html

The Alternative To DNSSEC

The good news is that, were DNSSEC to disappear tomorrow, the world would not be at greater risk. Pinning our hopes on DNSSEC, however, does put us at greater risk because it steals the air from more effective practices. Wider deployment of protocols like ssh and TLS (especially HTTPS) that do not require a trust in DNS is the single best step. We have already made tremendous strides in that regard, and there's even more to come. We can't forget the legacy services, however, that cannot be upgraded to avail themselves of those kinds of protocols directly. Opportunities exist, though, to have those services tunneled through secure protocols to provide them with the kind of security DNSSEC never could.