November 30, 2020

The Lengths People Go To Just To Avoid DNSSEC

Connecting to a website, say example.com, over TLS is a relatively straightforward affair. The client looks up the DNS A/AAAA record for example.com, connects to the IP address over TLS, and confirms that the presented certificate is valid for example.com.

In contrast, connecting to other services, like XMPP or SMTP, over TLS is less straightforward. That's because clients don't directly look up the A/AAAA record for example.com. Instead they look up a SRV record (for XMPP) or an MX record (for SMTP) which contains the hostname of the XMPP or SMTP server. Then they look up the A/AAAA record of that hostname and connect to it. This layer of indirection makes it easy to delegate the operation of certain services to other hosts. For instance, if example.com wants to use Gmail for their email, their MX record would contain aspmx.l.google.com.

This raises the question of which hostname the certificate should certify: the original hostname (example.com), or the hostname listed in the SRV/MX record (aspmx.l.google.com). Both options have problems.

Approach 1, using the original hostname (example.com), is undesirable because there is no automated way for the operator of a service like SMTP or XMPP to obtain certificates for the hostnames which are delegated to them. Google can automatically get a certificate for aspmx.l.google.com because they own google.com, but they can't for example.com. The admins of example.com would have to request the certificate themselves and give it to their SMTP and XMPP providers. Such a manual approach is bound to cause outages as people forget to renew expiring certificates. But there's another problem: the certificate would permit the SMTP and XMPP providers to impersonate all services for example.com. You probably don't want your instant messaging provider to be able to impersonate your website.

Approach 2, using the SRV/MX hostname (aspmx.l.google.com), doesn't have these problems. It's quite easy for the service operator to automatically obtain a certificate for their own domain. Unfortunately, this approach is not secure. Since the DNS lookup for the SRV/MX record is most likely unauthenticated, a man-in-the-middle attacker could intercept the query and return a rogue record that says the SMTP or XMPP service for example.com is handled by a server operated by the attacker. The attacker would have no problem obtaining a valid certificate for their own domain.

Perhaps the most obvious solution to this problem is to just make the DNS lookup authenticated. The IETF has been trying to do this since 1997 with DNSSEC. More than 20 years later, the results are not promising: fewer than 2% of .com domains support DNSSEC, only 25% of Internet users validate DNSSEC even when it is supported, the use of insecure crypto like 1024 bit RSA and SHA-1 is still rampant, and DNSSEC is so hard to deploy correctly that outages are common.

Unsurprisingly, many people would like to avoid the DNSSEC quagmire, which has led to some very interesting workarounds...

POSH

One workaround is POSH, or "PKIX over Secure HTTP", standardized in RFC 7711. With POSH, the owner of example.com publishes a JSON document at *https://example.com/.well-known/posh/SERVICE.json* containing a list of certificate fingerprints which are allowed to be used for the given *SERVICE* (e.g. *xmpp-server*) on example.com. To connect to example.com's XMPP server, a POSH-aware client would first retrieve *https://example.com/.well-known/posh/xmpp-server.json*, ensuring that the HTTPS server presents a valid, publicly-trusted certificate for example.com. It would then connect to the XMPP server indicated in example.com's XMPP SRV record, and ensure that it presents a certificate whose fingerprint is listed in the JSON document.

POSH solves the problems presented above. It's secure, because the JSON document containing the fingerprints is authenticated by a certificate for example.com, which remains under the control of the owner of example.com. The operator of the XMPP service doesn't need to obtain a publicly-trusted certificate for example.com. To facilitate certificate rotation, POSH supports delegation: the JSON file at example.com can reference the URL of a different JSON file which is hosted by the XMPP operator. This gives the XMPP operator flexibility to rotate certificates at any time without needing to inform their customers to update their JSON documents.

Although POSH was designed to be protocol agnostic, it was only ever used with XMPP, and even then I could only find a few XMPP clients which support it. It's fair to say POSH never caught on.

MTA-STS

A more recent workaround, for server-to-server SMTP only, is MTA-STS, standardized in RFC 8461. The underlying concept of MTA-STS is the same as POSH: the owner of example.com publishes a document over HTTPS with information about how to validate secure SMTP connections to example.com's mail servers. Several details are different. One cosmetic difference is that the document is published at mta-sts.example.com rather than example.com, which simplifies deployment for domain owners who can't easily make changes to their main website. A more fundamental difference is that instead of listing certificate fingerprints, the document lists the hostnames which are allowed in the MX record set. When connecting to an SMTP server for example.com, the client verifies both that it's connecting to a server listed in the MTA-STS document at mta-sts.example.com, and that the server presents a publicly-trusted certificate valid for the SMTP server's hostname.

The amusing thing about MTA-STS is that it basically boils down to duplicating the contents of the MX record in a document that is published over HTTPS, leveraging the WebPKI to authenticate the MX record rather than DNSSEC. It's kind of incredible that this is considered easier than using DNSSEC, despite having more moving parts and requiring duplication. That says way more about DNSSEC being a failure than about MTA-STS being good, and I've written before about how I think MTA-STS will prove hard to deploy in practice. I suggested how DNS providers could alleviate the problems by automating MTA-STS for their customers, but I'm not aware of any DNS providers to do so.

I am happy to report that SSLMate now offers MTA-STS automation as part of Cert Spotter. It's not quite as seamless as what a DNS provider could offer, but it's pretty good: Cert Spotter continuously monitors your domains' MX records and automatically publishes an appropriate MTA-STS policy for you, obtaining and renewing the necessary SSL certificates. All you need to do is publish two CNAME records delegating the MTA-STS-related subdomains (*mta-sts* and *_mta-sts*) to SSLMate-operated servers. Cert Spotter updates the policy automatically any time it detects a change to your MX records, ensuring the policy never falls out of sync with your DNS. For transparency, Cert Spotter emails you when this happens, so you can detect unauthorized MX record changes. Cert Spotter will also alert you if any of your MX servers have a TLS or certificate problem that would prevent MTA-STS from working.

Looking forward: SRVName certificates

There is another solution which, if it's ever deployed, will be much nicer than POSH or MTA-STS: SRVName certificates. SRVName certificates authenticate not just a domain name, like normal certificate, but a particular service running on that domain. For example, you could get a certificate that's valid for only SMTP on example.com, or only XMPP on example.com. This solves the security problem of the first approach above: the owner of example.com can give their mail server operator a SRVName certificate that's valid only for SMTP, allowing the mail server to operate an SMTP service for example.com, but not impersonate any other example.com services. Assuming the validation rules are flexible enough, the SMTP service operator could even obtain the SRVName certificate themselves provided the certificate authority validates that they are listed in the MX record for example.com.

Technically speaking, SRVName is a type of subject alternative name (SAN) which can be placed in a certificate, akin to the DNS SAN which certificates use today for authenticating domain names. It's possible for a certificate to contain both SRVName and DNS SANs, and here I use "SRVName certificate" to mean a certificate containing a SRVName SAN.

Unfortunately, there's a major obstacle blocking SRVName certificates: technically-constrained subordinate certificate authorities. A technically-constrained sub-CA is a certificate authority which is restricted to issuing certificates only for namespaces that are enumerated in the sub-CA certificate's name constraints field. For example, an enterprise that needs to issue a large number of certificates might operate a publicly-trusted, technically-constrained sub-CA that is constrained to the enterprise's domains and IP address ranges. Since their sub-CA can only issue certificates for namespaces that they control, they're allowed to operate it under looser security standards than unconstrained publicly-trusted CAs.

The problem is that if a particular type of SAN (in this case, the SRVName SAN) isn't listed in the name constraints as either allowed or denied, the standard says that all instances of that SAN type are allowed by default. Allow-by-default is usually a bad idea when security is concerned, and this case is no different. Clients can't accept SRVName certificates because it would be unsafe: every existing technically-constrained sub-CA that doesn't have SRVName in its name constraints field has unconstrained ability to issue SRVName certificates. Unfortunately, the Baseline Requirements (the rules governing public certificate issuance) only require technically-constrained sub-CAs to have DNS, IP, and Directory Name constraints. Consequentially, there are many existing technically-constrained sub-CAs out there that would need to be revoked and reissued with SRVName constraints before it's safe to deploy SRVName certificates.

Will that ever happen? Who knows. In the meantime, we're stuck with hacks like MTA-STS.

Older (View Archive)

Writing an SNI Proxy in 115 Lines of Go

Newer

Security Vulnerabilities in Smallstep PKI Software

Comments

No comments yet.

Post a Comment

Your comment will be public. To contact me privately, email me. Please keep your comment polite, on-topic, and comprehensible. Your comment may be held for moderation before being published.

Your Name:

(Optional; will be published)

Your Email Address:

(Optional; will not be published)

Your Website:

(Optional; will be published)

- Blank lines separate paragraphs.
- Lines starting with > are indented as block quotes.
- Lines starting with two spaces are reproduced verbatim (good for code).
- Text surrounded by *asterisks* is *italicized*.
- Text surrounded by `back ticks` is *monospaced*.
- URLs are turned into links.
- Use the Preview button to check your formatting.

Post Preview