TLS v1.2
...as of March 2020

Contents

- Current BCP usage of TLSv1.2
- TLSv1.2 = RFC 5246
- TLSv1.3 = RFC 8446 (after 4 years)

TLS Best Current Practice

- BCP195/RFC 7525 - Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) – May 2015
  - Quite a few (50+) references to this in other RFCs and Internet-drafts
  - https://datatracker.ietf.org/doc/rfc7525/referencedby/
- Best Current Practice (BCP) that says what options to use and not use in general in applications using TLS
  - Goal was to choose best options that are usable (which means implemented and deployed)
  - No changes to TLSv1.2 allowed, just profiling for what ‘s the best available options
- Uses in mail, web etc., properly dominated discussion but other applications were considered too (e.g VoIP)
  - Not all applications have e.g. a user or GUI
  - It’s just fine if applications impose additional constraints, but best if they still adhere to the recommendations in 7525
- This work was done at the same time as work on TLSv1.3 started, expectation is that there’ll be an update later for TLSv1.3 when that’s more widely deployed and we learn from those deployments
- IETF process minutiae: RFC 7525 can be updated and BCP195 will then point to the new RFC number but will remain BCP195

Handling Versions...

- SSLv2, SSLv3: MUST NOT negotiate
- TLSv1.0, TLSv1.1: SHOULD NOT negotiate if possible
- TLSv1.2: MUST support, MUST prefer over TLSv1.0/TLSv1.1
- DTLSv1.0: SHOULD NOT
  - DTLSv1.1: doesn't exist:-) DTLSv1.0 ~= TLSv1.1
- DTLSv1.2: MUST support, MUST prefer
- Fallback: "ick but sometimes you gotta"
  - Mainly, but not only, for browsers/web servers
  - But MUST NOT goto SSLv3 or earlier

Strict TLS

- HSTS = HTTP Strict Transport Security
  - RFC 6797, Nov 2012
  - Allows a web site to say that “you better use TLS for me (and these subdomains) for the next <many> seconds”
- Applications SHOULD do <something like> HSTS
  - Work to define that for mail transport/IMAP under way still
- HTTP clients/servers MUST support
- HTTP servers SHOULD use
  - Unless you’re stuck with self-signed certs, in which case, that’d be dim (get a real cert first from LE, then do HSTS)

Compression

- SHOULD disable compression
  - Due to CRIME (but be careful about BREACH too!)
  - Some applications are still ok (but few) and better to compress at application layer usually
  - Some applications needed work to adhere to this recommendation, e.g. Usenet/NNTP, (RFC 8054, January 2017)

Session Resumption/Renegotiation

- Be careful with that!
- Session ticket (RFC 5077, Jan 2008) MUST use as good a cipher, tickets MUST be regularly changed (e.g. weekly) and validity MUST be similarly limited
- TLS renegotiation – MUST implement renegotiation_info extension/hack (RFC 5746, Feb 2010)

Server Name Indication

- SNI (RFC 6066, Jan 2011) MUST be supported, but use of SNI is a “local matter”
  - SNI needed to support e.g. Apache VirtualHost to serve >1 web server on one Apache instance
- SNI has privacy implications – TLSv1.2 does not provide confidentiality for handshake extenstions like SNI
  - There was lots of debate as to how to improve that, but it’s **hard**, given the deployed web
- Current thinking for how to handle SNI confidentiality for TLSv1.3 is a work-in-progress
  - https://tools.ietf.org/html/draft-ietf-tls-sni-encryption
  - https://tools.ietf.org/html/draft-ietf-tls-esni
  - Current thinking is to have an “inner” Encrypted Client Hello (ECHO)

Application Layer Protocol Negotiation (ALPN)

- ALPN (RFC 7301, July 2014) is a TLS extension that allows a client and server to agree on which application layer protocol (primarily http/1.1 or h2) they want to use
- Needed because neither new port numbers nor HTTP UPGRADE work so well due to f/w's and middleboxes, and for performance (server gets to know h2 will work during TLS h/s)
  - Port 443 is most of the Internet these days:-)
- Sent in clear in TLSv1.2 & TLSv1.3, NPN was an equivalent proposal that saw some initital use and was encrypted, but somewhat problematic
  - ECHO would add confidentiality for this too

Ciphersuite Guidelines

- NULL encryption: MUST NOT
  - Had to be debated – the debuggers of the world weren't happy – you need to know HOWTO use wireshark with a private key file when testing
- RC4 – MUST NOT
- <112 bits of "strength" - MUST NOT
- <128 bits of "strength" - SHOULD NOT
  - Some 3DES still needed in the wild
- RSA Key Transport – SHOULD NOT
  - Now that one's still interesting!!!
- Forward Secrecy (i.e. DH/ECDH) – MUST support, MUST prefer

Cipheruites

- RECOMMENDED:
  - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
  - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- RFC 5246 (Aug 2008), RECOMMENDED TLS_RSA_WITH_AES_128_CBC_SHA
  - That's now a SHOULD NOT, but practically remains "mandatory to implement" (MTI)
  - Similar things true of many RFCs for applications that use TLS – as those are updated they ought not pick ciphersuites, but rather refer to BCP195
- To get this you might need to tweak how your code orders ciphersuites – there are HOWTOs for various applications, e.g. nginx, Apache
  - https://bettercrypto.org/ but maintaining that is a hard problem
- AES-CCM suites: "sigh, out of scope"
- Symmetric (PSK) suites: "sigh, out of scope"
- If you wanna prefer AES256 suites, go right ahead

Lengths/Strengths

- Integer DH: primes>=2048 bits long RECOMMENDED
  - Some 1024 bit stuff lingers, sadly, including clients that might barf on longer primes, Java VMs were a problem here, not sure if fully solved in deployments
- ECDH: >=192 bit curves RECOMMENDED
- RSA: >=2048 bit modulus RECOMMENDED
- ECDH preferred over integer DH, where possible
  - EC complexity/IPR has held back enough folks to be an issue still

7525 Misc. Stuff

- Doesn’t diss opportunistic security (RFC 7435, Dec 2014) - mostly orthogonal
- Do check the bleedin’ host names in certs!
  - And be careful about e.g. MX indirections
- Don’t re-use DH public values too much as that breaks FS
- Consider certificate revocation, esp., if dealing with long-lived TLS sessions – do OCSP
  - Hard-fail is still not RECOMMENDED though as it’s too hard still to do reliably
- OCSP stapling is nice! (but still not that widely used yet)

TLS Measurement

- There are now lots of ways to measure your deployment of TLS, typically these scan a web site and tell you what TLS options you've enabled
- zmap/zgrab https://github.com/zmap/
- Censys.io – built from those (now commercial)
- Qualsys ssllabs - https://www.ssllabs.com/
- Mozilla observatory - https://observatory.mozilla.org/
- Privacy Score - https://privacyscore.org/
- Part of a more general trend that whatever is visible on the public Internet will **constantly** get attacked/probed/measured
- Some of the above have command line cilents:
  - e.g. pip install httpobs-cli
- Aspects of the above are IPv4 oriented as it's easy to scan the address space, but IPv6 scanning techniques are developing, so it's not safe to assume that using IPv6 means you're not going to be scanned