Real Issues with TLS (and Fixes)

Oh crap, the TLS ecosystem isn't near-perfect.
Some people work on fixes.
Others work on finding more problems.

That's life as a really popular security protocol.

TLS Issues

- RFC 7457 - Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS) – Feb 2015
  - Good source of references to attacks/bugs as of that date
  - More continues to happen of course...

Contents

- Issues in rough age-order
  - Weak key generation
  - CA screw-ups
  - Bleichenbacher
  - Re-negotiation
  - CA screw-ups
  - Weak key generation
  - Stuxnet/Flame
  - BEAST
  - CRIME
  - Lucky-13
  - DROWN
- More to come no doubt
- Some possible Futures for TLS

Don't panic!

- Many issues have been found over the years with TLS implementations and a few issues with the protocol itself
- It is still far far easier to properly use TLS and get security than it is to re-invent TLS
  - So use it!
  - But make sure you're using it properly

Ancient Crappy Key Generation

- 1996-era Netscape mucked up key generation

https://people.eecs.berkeley.edu/~daw/papers/ddj-netscape.html

  Problem: TLS pre-master secret could be guessed

  - Only really 47 bits of entropy and not 128

- Fix: More entropy and better code

  - No protocol change needed

Ancient CA Screw-Ups

- 2001, social engineering works then and now
- VeriSign issued code-signing certs to someone who was not microsoft.com
- Fixes:
  - VeriSign revoked certs
  - MSFT issued a patch:

https://technet.microsoft.com/library/security/ms01-017

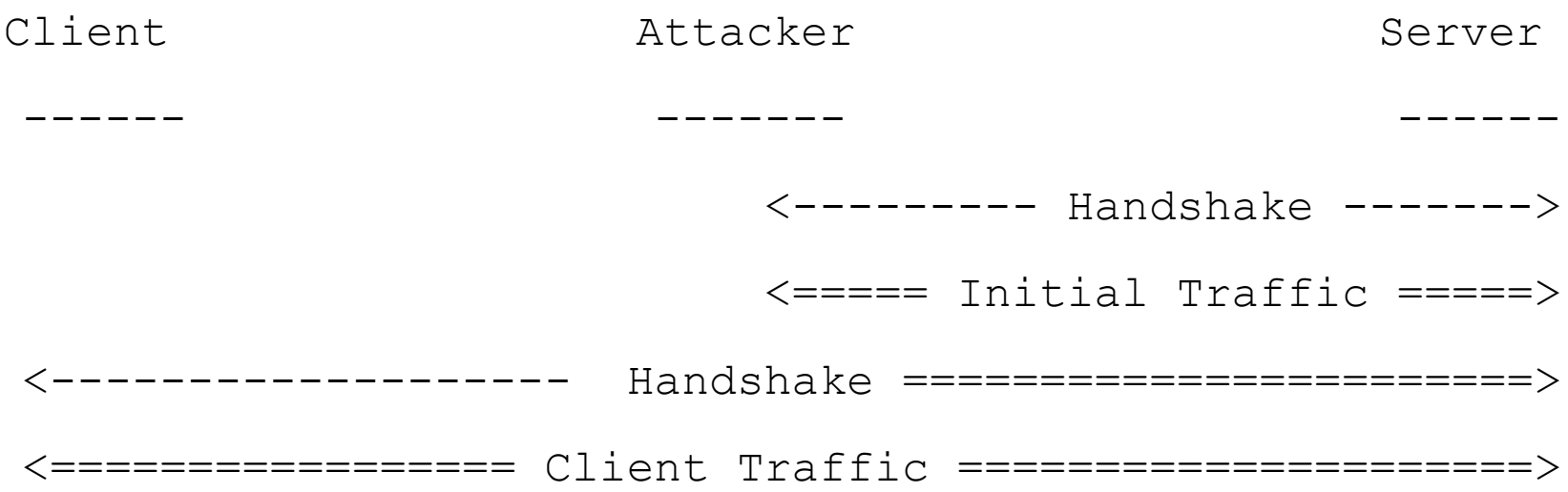
- Interesting sign of how bad revocation really was (and is!), user-agents need patching for serious revocations!

Bleichenbacher

- 1998 Adaptive chosen ciphertext attack
  - First of those afaik that was near-practical
  - Paper/ppt in course materials
- Really an attack on PKCS#1 and not TLS, but impacts TLS implemtentations
- Problem: responding to bad padding too early in handshake
- Bleichenbacher just keeps on giving (to security researchers):
  - Timing attacks on code, not just protocol
  - Java exception handling to generate random PMK
  - Cross-protocol attacks and fallback attacks – DROWN (later)
  - ROBOT (later)
- Fixes:
  - Don't do that – RFC 3218
  - Use OAEP – still not really done today because of “legacy” APIs

TLS Re-Negotiation Bug

- Unlike others, this one was a protocol-design flaw
- Re-negotiation - new session not bound to initial TLS session
  - Attacker can force renego but URL from original session used with 2nd session's privileges
  - Fix: RFC 5746, developed in 3 months start to finish, a breaking backwards incompatible protocol change
- Traffic pattern:



CA Screw-Ups (1 - diginotar)

- 2011, diginotar.nl were a public CA present in browser root stores

  https://en.wikipedia.org/wiki/DigiNotar
- Someone hacks into diginotar servers, issues certificates for google.com, microsoft.com etc., abuses those certificates
- Weeks later diginotar confess and are kicked from browser root stores and go bust
- Problem: every CA can certify any DNS name in the web PKI
- Fixes:
  - diginotar liquidated
  - CAA RR in DNS

CA Screw-Ups (2 - “comodogate”)

- 2011 again, good/bad year for this
- Not Commodo's direct fault, but their responsibility

https://www.theregister.co.uk/2011/03/28/comodo_gate_hacker_breaks_cover

- Comodo partner/RA hacked, certificate requests for google.com, microsoft.com etc., submitted from RA to CA and certificates issued
- Certificates revoked quickly but lack of sanity checks within CA infrastructure damaging
- Fixes:
  - Commodo now do more sanity checking on requests from Ras
  - CAA again

Internet-Scale Weak-Key Generation

- 2012 – Build DB of O(10^6) keys and look for problems...

  https://freedom-to-tinker.com/blog/nadiah/new-research-theres-no-need-panic-over-factorable-keys-just-mind-your-ps-and-qs/

- Problem: many embedded devices generate keys before they have any real entropy on first boot; with RSA, 1st prime from pair likely to collide with someone else when there are millions of devices (TLS/SSH servers) visible on the Internet
  - >1% of 15 million keys had common factors
  - Common RSA modulus factors => easy to factor, GCD calculation is easy regardless of size
- Fixes:
  - More entropy (somehow;-)
  - Better code (somehow:-)
- Interesting thing:
  - Building a DB of almost all public keys used on the public Internet is quite feasible
  - Now there's a thought! More on that later.

Stuxnet/Flame – BIG Bad Actors

- 199x-201x: Stuxnet and Flame
- Stuxnet: involved compromised code-signing private keys
- Flame: Stuxnet + new MD5 prefix-collision attack for out of date Windows deployments + really really stupid conflation of different PKIs in MSFT ecosystem

https://arstechnica.com/information-technology/2012/06/flame-crypto-breakthrough/

  - Above is a bit overstated maybe on the “breakthrough” part
- Problems:
  - Private key handling (stuxnet)
  - Outmoded PKIs (MD5 – sheesh!)
  - Major state actors have major resources and do not have the interests of the Internet at heart (at least for the parts of their organisations that create attacks)
- Fixes:
  - Do PKI better (we all wish;-)
  - Don't stick with crap algorithms like MD5

BEAST – Back to TLS

- 2011 – BEAST

https://www.ietf.org/proceedings/85/slides/slides-85-saag-1.pdf

- BEAST combines active content on page with a network MITM who can passively watch to allow for cookie recovery within TLS
  - TLS with CBC, next block IV is last block's ciphertext; URLConnection class allows client code to split info into small chunks; allows messing with TLS application layer PDUs; allows client code that can't read cookie to emit data so that collaborating MITM can detect when client code guessed correct cookie
  - Complicated – see the slides referenced above
- Fix:
  - Partly: Use TLS 1.2 and not earlier

CRIME – TLS Compression Not So Good in the end

- 2012 – CRIME

https://www.ietf.org/proceedings/85/slides/slides-85-saag-1.pdf

- Easier to grok than BEAST – if you compress things then they get smaller; that allows you to guess and detect good guesses from ciphertext size
  - Feed back HTML to client with guesses, each embedded in an img reference and see which gets smaller; smaller one is the one that's been seen before so you've learned a character of a cookie; repeat for next characters
- Fixes:
  - Turn off compression
  - What to do for HTTP/2.0? HPACK for headers, application layer and no compression in TLS

Lucky-13 – CBC Problematic Again!

- 2013 – Lucky-13 CBC timing side-channel

https://en.wikipedia.org/wiki/Lucky_Thirteen_attack

- CBC ciphersuites allow for a timing attack based on the size of HMAC-SHA1 input and timing on SHA1; millisecond differences bettwen good and bad padding expose plaintext
- Attack more practical against DTLS (UDP/connectionless) than TLS (TCP/connection-oriented)
- Fixes:
  - Use TLS1.2 with AEAD ciphers
- Of note:
  - Researchers worked with vendors/standards community to get fixes out while resarch paper embargoed
  - Everyone loved 'em for that

RC4 Problematic in TLS

- 2013, mid-March

https://www.isg.rhul.ac.uk/~sean/Mantin-RC4.pdf

- Found position-based biases in RC4 ciphertext bytes, based on analysis of ~2^40 inputs; repeated encryption of same plaintext at same position (e.g. Cookie) vulnerable, more problematic if limited alphabet (e.g., base32); result is plaintext recovery with about 2^24 or 2^30 TLS sessions
- Fixes:
  - Don't use RC4
- Of note:
  - Same researchers as Lucky-13, same modus-operandi, same love

DROWN – SSLv2!!

- March 2016
- Old SSLv2 code that can still be accessed re-enables Bleichenbacher
- If same RSA private key on >1 host, any of those willing to play SSLv2 could be attacked
- Yes, they have a web site: https://drownattack.com/
- Nice cross-protocol attack, similar to https://www.nds.rub.de/research/publications/ccs15/
- Affected millions of sites on the Internet
- Fix:
  - Get rid of old, old code!
  - Don't over-prioritize interop and ignore everything else

startcom/wosign - 2016

- Startcom CA sold on the q.t.
- Buyer a CA that had “interesting” practices
- CA (WoSign) couldn’t justify those to browsers
- Dropped from root stores, e.g.:

  https://threatpost.com/apple-to-block-wosign-intermediate-certificates/121044/

The good news: CT helped!

ROBOT - 2017

- Return Of Bleichenbacher's Oracle Threat

  https://robotattack.org/
- Mostly re-testing for Bleichenbacher on the Alexa top 1M
- Main point: demonstrated real issues remaining in real deployments
- To avoid: just don’t support RSA key transport at all

Trustico - 2018

- CA re-seller (an RA) emailed 23k private keys to it's CA
- Apparently wanting those revoked due to business changes
- And of course someone showed their web site was extremely borkable...
    - https://arstechnica.com/information-technology/2018/03/trustico-website-goes-dark-after-someone-drops-critical-flaw-on-twitter/

eTLS or ETS or static D-H

- As TLS1.3 was nearing completion, some folks (from BITS, a US financial services organisation) turned up who wanted to keep RSA key transport in TLS1.3, so they could eavesdrop “within their enterprises”
    - The TLS WG said “no”; They then wanted to use static D-H TLS server shares so they could eavesdrop “within their enterprises”; The TLS WG said “no” again (or rather, did not have rough consensus to work on that, which likely involved attempts to pack rooms by both sides of the argument)
    - See https://github.com/sftcd/tinfoil for my take on all that
- So they went off to ETSI who speedily rubber-stamped their ideas and called that eTLS (later ETS or MSP when the IETF complained about abusing the name TLS)
    - https://www.eff.org/deeplinks/2019/02/ets-isnt-tls-and-you-shouldnt-use-it
    - https://www.etsi.org/deliver/etsi_ts/103500_103599/10352303/01.01.01_60/ts_10352303v010101p.pdf
    - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9191 (slightly amusing:-)
- Don’t implement, deploy or use that is my advice – disparage it instead

TLS MITM Products

- 20xx-date: Vendors sell TLS MITM products for “corporate policy enforcement”

https://directorblue.blogspot.ie/2006/07/think-your-ssl-traffic-is-secure-if.html

- Reasons:
  - Good: inbound malware scanning
  - Bad: everything else
- Some CA operators “collaborated” with this bad practice:
  - https://bugzilla.mozilla.org/show_bug.cgi?id=724929
- Getting more common, but IETF have refused numerous vendor requests to standardise this behaviour
- Fixes:
  - New PKI approaches/features that detect country-level misbehaviour will hit this
  - End-to-end uise of TLS for sped and avoiding advertising will hit this

TLS API attack on UPS

- 2022: Security researcher deep-dive on UPS found various issues, including application-layer failure to react to TLS library errors.
- Tech report:
  - https://info.armis.com/rs/645-PDC-047/images/Armis-TLStorm-WP%20%281%29.pdf
- Result (in the face of active attac populated TLS session object that has a master secret of all zeros – oops
- Bonus: the tech report ends with smoke emerging from the UPS:-)

UPS in smoke after overheating the DC link capacitor

So where are we?

- (D)TLS is imperfect in various ways
  - PKI wrinkles and other imperfections
  - Too many TLS ciphersuites (with weaknesses)
  - Significant change takes years (before old bad code can be deleted)
- (D)TLS is still the best bet for many things
  - PKI is better than non-existent non-PKI (old-bad-but-existing-infrastructure often wins)
  - Plenty of good TLS ciphersuites esp. TLS 1.3 AEAD
  - Change only takes years because of success!

Aside: HTTP/2

- HTTP is definitely one of the most important protocols on the Internet
- HTTP/1.1 and earlier are not very efficient in various ways, e.g., header verbosity, head-of-line blocking
- HTTP/2 recently finished (RFC7540), based on SPDY, to make improvements (without semantic change vs. HTTP/1.1)
- Part of that work involved negotiating HTTP/2 usage and not HTTP/1.x usage
  - Doing do without additional round-trips is a MUST
- One important way to negotiate HTTP/2 is via TLS handshake
  - Claim: HTTP/TLS gives better speed and connectivity than cleartext HTTP!
  - Done via “Application Layer Protocol Negotiation” (ALPN) as TLS handshake extension (RFC7301)
- Notes:
  - This replaces TCP port numbers – architects faint!
  - This makes TLS criticial for deployment of HTTP/2
  - This annoys middlebox vendors/operators - Hmmmmmmmm
    - QUIC (RFC9000) probably annoys them even more

Strict TLS

- If a web site offers TLS then great, that's hard to spoof. If however, I spoof that same content but witout TLS, users won't notice.
  - “Studies” show users don't notice lock-icon, green URL bar, etc.
- However, browser can notice that a site I normally access over TLS is now in clear
  - Browser can't by itself know that's bad for sure
- But sites might be able to tell browser “once you've ever been here via TLS, you should never believe if you see this site in clear”
  - For example, paypal.com are keen on this
  - Creates potential new way to shoot oneself in the foot – turn on HSTS with the wrong key; sub-/parent-domain handling within same “administrative” domain
    - Sidebar: public-suffix list is a pain
- Result: HSTS HTTP header – HTTP Strict Transport Security (RFC 6797)
  - Site says: TLS MUST be used to access this site (and optionally sub-domains) between now and now+timeout and browser MUST NOT allow cleartext access to the above
  - Newer HSTS headers win over old (pushes duration out to future indefinitely)
  - HSTS MUST be first seen over TLS of course

TLS Key Pinning

- Subtly different idea...
- Provide an HTTP header that allows browser to “pin” site TLS server certificate to one or two site public keys or CA public key (RFC 7469)
- Allows a site to “brick” itself, so HPKP is mostly deprecated by sensible deployments
- Note: HSTS is recommended, HPKP is not

CAA – Certification Authority Authorization

- If a site publishes the list of CAs with which it wants to do business, then other (non-stupid) CAs can detect some badness that might currently pass unnoticed
  - Some compromised RA asks company-CA to issue certificate for example.com but example.com has publicly said they get certs from Foo and Bar CAs only
  - RFC 6844, is now being deployed due to CAB forum preferences, and it’s not hard (if you can write to DNS)

Conclusions

- TLS and PKI are and will continue to be attacked
  - The most commonly used end-to-end security mechanism is an attractive target for researchers and bad actors
- 20 years of experience seems to show:
  - TLS is robust enough and can evolve slowly
  - Evolution is too slow esp., for deprecation
  - TLS is too hard to use properly for some application developers
  - TLS1.3 (modulo 0rtt) will be a fine improvement
- Don't panic:-)