Recent Progress in and around LibreSSL

Theo Buehler
tb@openbsd.org

EuroBSDCon – September 17, 2022

About LibreSSL

One of the four major forks of OpenSSL

- 1998: OpenSSL forks from/continues SSLeay
  accumulates (more) shoddy code, cruft over next 16 years
  after lots of desasters, heartbleed makes people look and act
- Apr 2014: OpenBSD forks LibreSSL
- Jun 2014: Adam Langley (Google) makes BoringSSL public
- Nov 2021: Akamai / Microsoft want QUIC $\rightsquigarrow$ QuicTLS
  OpenSSL + patchset to add BoringSSL QUIC API

LibreSSL Main Features

- libtls: sane and easy-to-use wrapper of the SSL/TLS stack
- clean room implementation of TLSv1.3 stack (2018-2020)
  - centerpieces: record layer and handshake state machine
  - missing features: PSK (work in progress), ECH (complicated)
  - non-goal: early data
- new certificate validator
- documentation (unfortunately there's only one schwarze@)
- lots of code cleanup
- largely compatible with OpenSSL 1.1 on support intersection
- this improved a lot due to making structs in LibreSSL opaque
- ABI about as stable as OpenSSL 1.1

On OpenSSL compatibility

- OpenSSL 1.1 API: have what we need, more than we wanted
- No OpenSSL 3 API yet
- $>$ 2000 OpenBSD ports link against libcrypto or libssl
- $<$ 100 of these need patches ($<$ 5%)
- Painful: Qt, PyPy (because of py-cryptography), stunnel
- By far the most requested missing feature is Ed25519 ...
- ... followed by things like SHA-512/256, SHA-3, Blake, ...
- 6 ports link against OpenSSL:
  - mail/opensmtpd-filters/dkimsign flavor (Ed25519 signatures)
  - mail/postfix (DANE, mostly)
  - net/bro aka zeek: needs TLS-PRF API
  - lang/node: Ed25519 + a dozen API functions
  - net/nagios/nsca-ng: PSK
  - security/libretls: by design

Background: Anatomy of a Certificate

Certificates are a complicated data structure.
ASN.1 from RFC 5280:

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }
```

- sequence: basically a `struct`
- TBS: To Be Signed
- Contents of struct
  1. what is (to be) signed
  2. how is it signed
  3. signature

Background: Anatomy of a Certificate (continued)

```
TBSCertificate  ::=  SEQUENCE  {
     version          [0]  EXPLICIT Version DEFAULT v1,
     serialNumber          CertificateSerialNumber,
     signature             AlgorithmIdentifier,
     issuer                Name,
     validity              Validity,
     subject               Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     extensions      [3]  EXPLICIT Extensions OPTIONAL
                          -- If present, version MUST be v3
     }
```

Could go on forever.

RFC 5280: 151 pages

$>$ 80 of which are the details of this struct (and CRLs)

A PEM encoded certificate

Most of you will have seen something like this

```
-----BEGIN CERTIFICATE-----
MIIG4jCCBcqgAwIBAgISBJxsswkXmlb9UVavEeONm1EzMA0GCSqGSIb3DQE
MDIxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQswCQY
EwJSMzAeFw0yMjA3MjExNTQ3MTJaFw0yMjEwMTkxNTQ3MTFaMBoxGDAWBgN
...
-----END CERTIFICATE-----
```

- PEM: Privacy Enhanced Mail (see RFC 7468)
- Base64 encoded DER of certificate
- DER: Distinguished Encoding (Rules) of ASN.1 “struct”

Aside: Why do certs start with `MII`?

All 133 CA certs in OpenBSD's root bundle start with `MII`

```
$ grep -c -- -----BEGIN /etc/ssl/cert.pem
133
$ grep -A1 -- -----BEGIN /etc/ssl/cert.pem | grep -c MII
133

$ echo -n MIIG | b64decode -r | hexdump -Cv | head -n 1
00000000  30 82 06
```

- 30: DER: encoding of an ASN.1 `SEQUENCE`
- 82: DER: the length is described by the next two bytes
- `MII`: Base64 of 30 82 + 2 most significant bits of length
- length of a cert is > 127 bytes (so needs at least two bytes)
- length of a cert is usually < 16684 bytes,
  so the two most significant bits are 0

New Certificate Validator

- "Legacy validator" inherited from OpenSSL: unmaintainable
- During lockdown, beck@ wrote an RFC 5280 validator
- Initial code was correct. We only found minor bugs, ...
- ... then many months of whack-a-mole started
- Lots of software relies on
  - strange and overly specific error codes in certain situations
  - undocumented behavior of the verify callback
  - specific order of traversing the potential chains
- Took us two years to be reasonably compatible with the legacy validator
  - fix one thing, break ten others
  - one hole introduced in the process

Legacy Record Layer Rewrite (WIP)

- jsing@ wrote a very nice record layer underlying TLSv1.3
- Similar ideas can be used for old TLS versions and DTLS
- Goal: remove `ssl_pkt.c` and `d1_pkt.c` (terrible code)
- Uses `CBS` and `CBB` instead of explicit pointer manipulations
- With this work, DTLSv1.2 support came pretty much for free
  - landry@: linphone, baresip
  - kn@: tdesktop
  - missing bit: `BIO_ADDR` API, so Qt cannot yet use it

QUIC API

- De facto standard API by David Benjamin of BoringSSL
- OpenSSL PR 8797 (2019): port by Todd Short (Akamai)
- Had to wait for OpenSSL 3 (was already late at that point)
- Mai 2021: QUIC standardized in RFCs 9000 – 9002
- Sep 2021: OpenSSL 3 released
- Oct 2021: OpenSSL want their own stack
  - BoringSSL compatibility explicit non-goal
  - Unclear why. *Someone* must have a reason...
  - QUIC transport protocol not really within OpenSSL's expertise
- Nov 2021: QuicTLS announced in IETF side meeting

QUIC API (continued)

- beck@ and jsing@ ported BoringSSL API
- Plugged very nicely into jsing@'s record layer
- Needed `EVP_chacha20_poly1305` support in libcrypto
- Experimental version will be available in LibreSSL 3.6
  - curl can speak QUIC using ngtcp2
  - wlallemand added minimal working version to haproxy
    Needs `SSL_CTX_set_client_hello_cb` for full support
- BoringSSL API works, but is not great
  - exposes full structs and enums publically (sigh...)
  - BoringSSL and QuicTLS have already diverged
  - ngtcp2 initializes public struct without C99 initializers
  - BoringSSL open to improvements
  - QuicTLS probably set in stone

Primality Testing

Starting point: a 2018 preprint:
Prime and Prejudice: Primality Testing Under Adversarial Conditions.
Albrecht, Massimo, Paterson, Somorovsky:

- [. . .] construct 2048-bit composites that are declared prime with probability 1/16
- [. . .] the advertised performance [LibreSSL/OpenSSL] is 2^{-80}
- [. . .] for a number of libraries (Cryptlib, LibTomCrypt, JavaScript Big Number, WolfSSL), we can construct composites that always pass the supplied primality tests

Primality Testing (continued)

Tricky to fix

- Workaround: crank number of Miller-Rabin rounds (slow)
- Recommendation: Baillie–Pomerance–Selfridge–Wagstaff algorithm
- Problem: this isn't easy – someone needs time and skills

Primality Testing (continued)

Lucky coincidence: Martin Grenouilloux has time and skills

- background: espie@ finds preprint independently
- tells us he has a promising student with a knack for maths
- Martin already had a Python implementation
- a few weeks later: C implementation lands in my inbox
- work stalled for a few of weeks due to exams
- things become easier with a mostly correct implementation...
- clean up, optimize, simplify, fix, and commit
- result is one of the nicest pieces of code in libcrypto
- amazing work by Martin Grenouilloux

RFC 3779 support

- This is about routing and BGP
- X.509 Extensions for IP Addresses and AS Identifiers
- Issuer of certificate transfers "internet numbers" to subject
- Part of libcrypto, ported by job@ from OpenSSL
- Helps `rpki-client`, makes `openssl x509` output nicer
- Needed audit, cleanup, lots of fixes, regress
- Public API is pretty broken
- Downside: code is inefficient, hit by certificate validator
- `rpki-client`: spends $\sim 10\%$ of runtime in RFC 3779 code

Testing, CI and Coverity

- Ilya Shipitsin from haproxy has been tremendously helpful
  - Helped add ASAN CI, which has been invaluable
  - Also helps with triaging Coverity issues
- tlsfuzzer runs as part of daily regression tests
  - Tickles many corner cases
  - Helped improve standards compliance a lot
  - Hannes Mehnert mentioned it at BSDCan 2019, thanks!
- The Ruby OpenSSL Gem has a very useful test suite
- Joshua Sing rewrote and improved many of the old tests

Thanks

- LibreSSL core team: bcook@, beck@, inoguchi@, jsing@
- schwarze@ for awesome documentation and many bug fixes
- ajacoutot@, sthen@ for help with ports
- genua@ for testing infrastructure and for sponsoring work
- Martin Grenouilloux, espie@ for the work on primality testing
- Ilya Shipitsin for help with portable
- “orbea” for helping with upstream patches
- OpenBSD foundation for sponsoring bulk build machine