# Not-quite-so-broken TLS

## Lessons in re-engineering a security protocol specification and implementation

David Kaloper Meršinjak
Hannes Mehnert
Peter Sewell
Anil Madhavapeddy
*University of Cambridge, Computer Labs*

**Usenix Security, Washington DC, 12 August 2015**

```
int ssl23_get_client_hello(SSL *s)
    {
    char buf_space[11]; /* Request this many bytes in initia
                         * We can detect SSL 3.0/TLS 1.0 Cli
                         * ('type == 3') correctly only when
                         * is in a single record, which is n
                         * the protocol specification:
                         * Byte  Content
                         *  0     type            \
                         *  1/2   version          > record
                         *  3/4   length          /
                         *  5     msg_type        \
                         *  6-8   length           > Client
```

# Common CVE sources in 2014

| Class | # |
|---|---|
| Memory safety | 15 |
| State-machine errors | 10 |
| Certificate validation | 5 |
| ASN.1 parsing | 3 |

(OpenSSL, GnuTLS, SecureTransport, Secure Channel, NSS, JSSE)

# Root causes

- Error-prone languages
- Lack of separation
- Ambiguous and untestable specification

# nqsb approach

- Choice of language and idioms
- Separation and modular structure
- A precise and testable specification of TLS
- Reuse between specification and implementation

# Choice of language and idioms

OCaml: a memory-safe language with expressive static type system

- Well contained side-effects
- Explicit flows of data
- Value-based
- Explicit error handling

We leverage it for abstraction and automated resource management.

# Formal approaches

Either reason about a simplified model of the protocol;
or reason about small parts of OpenSSL.

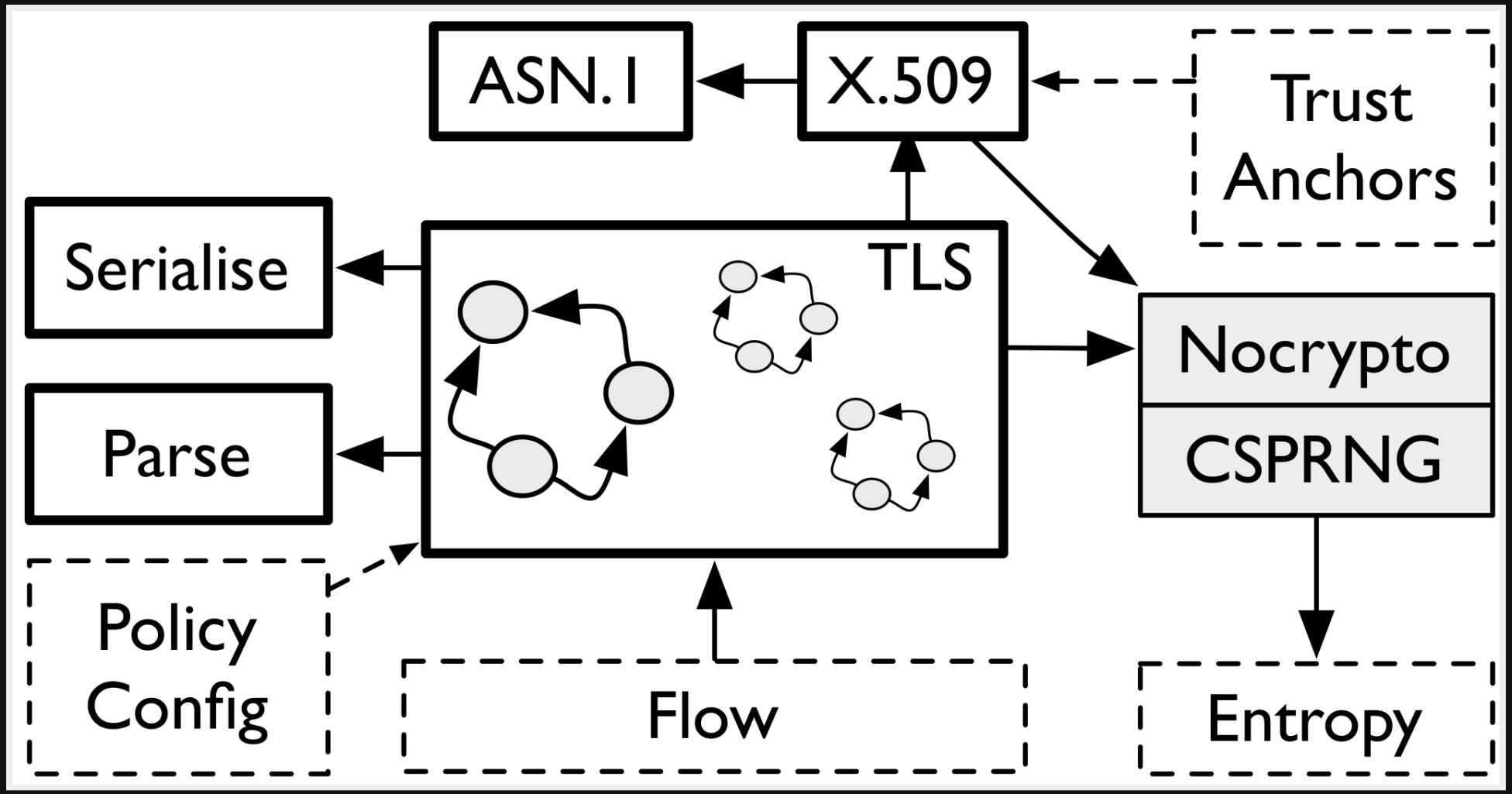
In contrast, we are engineering a deployable implementation.

# *nqsb-tls*

A TLS stack, developed from scratch, with dual goals:

- Executable specification
- Usable TLS implementation

# Structure




**nqsb-TLS** ML module layout

# Core

Is purely functional:

```
val handle_tls :
  state -> buffer ->
  [ `Ok   of state * buffer option * buffer option
  | `Fail of failure ]
```

# Core

OCaml helps to enforce state-machine invariants.

```
let handle_handshake ssn hs buf =
  match parse_handshake buf with
    Error -> fail (`Fatal `ReaderError)
  | Ok handshake ->
     match (ssn, handshake) with
       (AwaitClientHello, ClientHello ch) ->
        answer_client_hello hs ch buf
     | (AwaitClientFinished (sessn, log), Finished fin) ->
        answer_client_finished hs sessn fin buf log
     (* ... *)
     | _ -> fail (`Fatal `UnexpectedHandshake)
```

# ASN.1

```
    TBSCertificate ::= SEQUENCE {
         version           [0] Version,
         serialNumber          CertificateSerialNumber,
         signature             AlgorithmIdentifier,
         issuer                Name,
         validity              Validity,
         subject               Name,
         subjectPKInfo         SubjectPublicKeyInfo,
         issuerUniqueID    [1] IMPLICIT UniqueId OPTIONAL,
         subjectUniqueID   [2] IMPLICIT UniqueId OPTIONAL,
         extensions        [3] Extensions OPTIONAL
}
```

# ASN.1 in OCaml

```
let tbsCertificate   = sequence (
   (opt "version"         (e 0 version))
 @ (req "serialNumber"          certificate_sn)
 @ (req "signature"             Algorithm.identifier)
 @ (req "issuer"                Name.name)
 @ (req "validity"              validity)
 @ (req "subject"               Name.name)
 @ (req "subjectPKInfo"         PK.pk_info_der)
 @ (opt "issuerUID"       (i 1 uniqueId))
 @ (opt "subjectUID"      (i 2 uniqueId))
-@ (opt "extensions"      (e 3 Extension.extensions_der))
)
```

# X.509

```
let is_server_cert_valid host time cert =
  match
    validate_time time cert,
    maybe_validate_hostname cert host,
    version_matches_extensions cert,
    validate_server_extensions cert
  with
  | (true, true, true, true) -> success
  | (false, _, _, _) -> fail `CertificateExpired
  | (_, false, _, _) -> fail `InvalidServerName
  | (_, _, false, _) -> fail `InvalidVersion
  | (_, _, _, false) -> fail `InvalidServerExtensions
```

# Cryptography

- Cipher and hash cores in C
- Cipher modes (CTR, CBC, GCM, CCM) in OCaml
- Public-key cryptography in OCaml using GMP

# Timing side channels

- Symmetric ciphers: AES-NI
- MODP public-key: blinding
- PKCS, Protocol: deploying widely accepted mitigations
- lucky13: no mitigation in place yet

# Live handshake visualisation

Analysed 30000 recorded TLS sessions

# Trace checker

- Live demo here

# BTC Piñata

- Transparent Bitcoin bait
- Both client and server side are exposed
- Private BTC key when successfully authenticated

# Results: BTC Piñata

- Since February 2015
- Attacks included exploits in other stacks
- 20000 traces from 1000 IPs

(We **can't** infer security from the Piñata.)

# Trusted Computing Base

| Subsystem | Linux/OpenSSL | nqsb Unikernel |
|---|---|---|
| Kernel | 1600 | 48 |
| Runtime | 689 | 25 |
| Crypto | 230 | 23 |
| TLS | 41 | 6 |
| **Total** | **2560** | **102** |

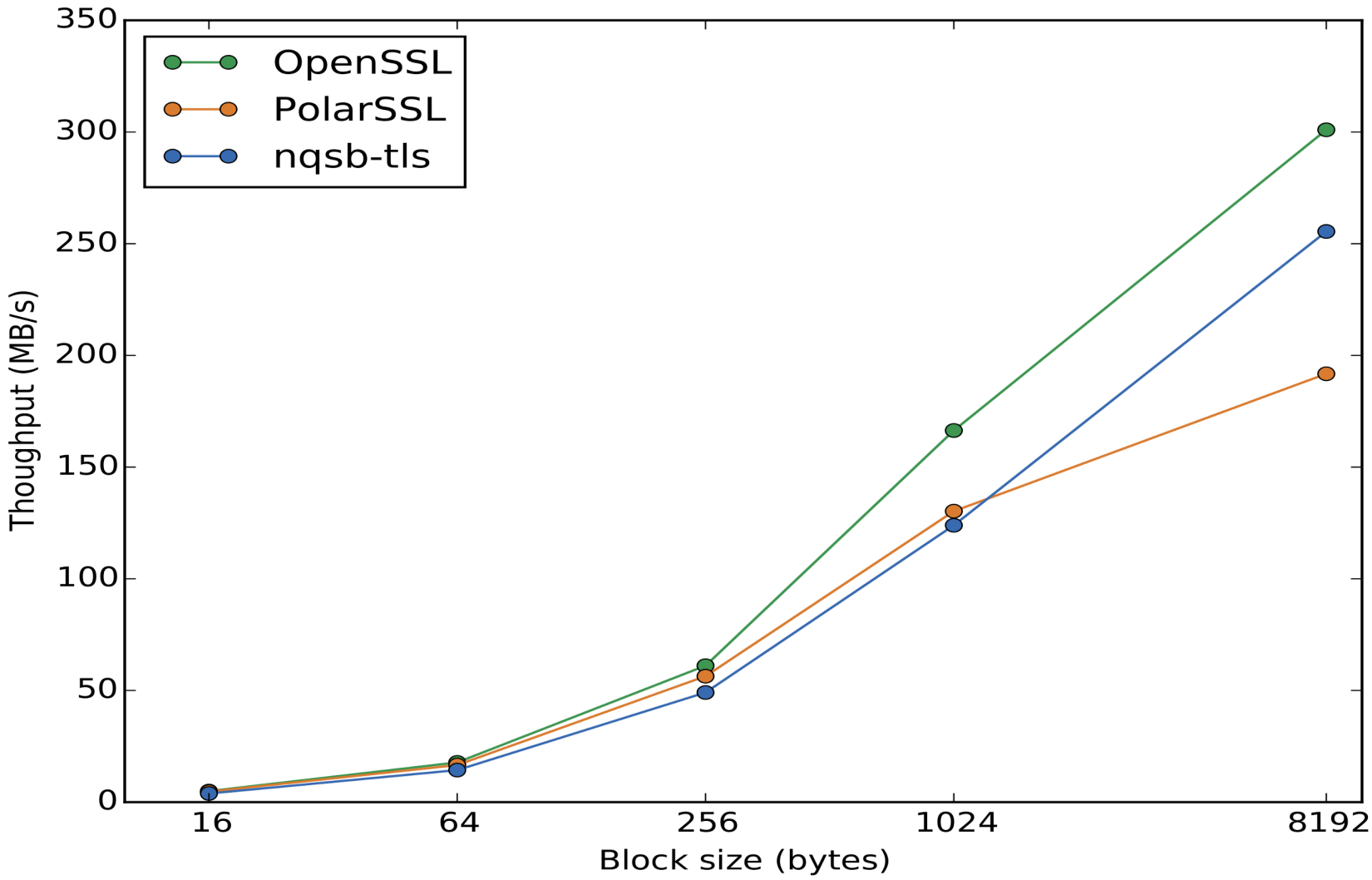
(numbers in **kloc**)

The code size of Piñata is 1/25 of OpenSSL on Linux.

# Handshake performance

| | nqsb | OpenSSL | PolarSSL |
|---|---|---|---|
| RSA | 698 hs/s | 723 hs/s | 672 hs/s |
| DHE-RSA | 601 hs/s | 515 hs/s | 367 hs/s |

# Throughput


OpenSSL
PolarSSL
nqsb-tls
Thoughput (MB/s)
350
300
250
200
150
100
50
0
16
64
256
1024
8192
Block size (bytes)

# Various rollouts

Moving OCaml Labs infrastructure to *nqsb-TLS*.

- tlstunnel a TLS tunnel (stud/stunnel)
- jackline an XMPP chat client
- trace-checker a TLS validator using packet traces
- certify a CA tool

# nqsb TLS conclusion

- Engineered using a radical approach for the systems community
- Purely functional, designed for clarity
- Usable both as specification and implementation
- Small TCB, reasonable performance, concise code
- Avoids root causes of common flaws

https://nqsb.io