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 initial read. */
    /* We can detect SSL 3.0/TLS 1.0 Client Hellos ('type == 3') correctly only when */
    /* is in a single record, which is not guaranteed by the protocol specification: */
    /* Byte     Content                        */
    /* 0        type                            */
    /* 1/2      version                         */
    /* 3/4      length                          */
    /* 5        msg_type                        */
    /* 6-8      length                          */
    /* > record */
    /* > Client Hello message */
}
# Common CVE sources in 2014

<table>
<thead>
<tr>
<th>Class</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory safety</td>
<td>15</td>
</tr>
<tr>
<td>State-machine errors</td>
<td>10</td>
</tr>
<tr>
<td>Certificate validation</td>
<td>5</td>
</tr>
<tr>
<td>ASN.1 parsing</td>
<td>3</td>
</tr>
</tbody>
</table>

(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**

- **Serialise**
  -ASN.1
  -X.509

- **Parse**

- **Policy Config**

- **Flow**

- **Trust Anchors**
  - Nocrypto
  - CSPRNG

- **Entropy**
Core

Is purely functional:

```ocaml
core
val handle_tls :
  state -> buffer ->
  [ `Ok of state * buffer option * buffer option |
  `Fail of failure ]
```
OCaml helps to enforce state-machine invariants.

```ocaml
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
}


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_info)  
  @ (opt "issuerUID" (i 1 uniqueId))  
  @ (opt "subjectUID" (i 2 uniqueId))  
  -@ (opt "extensions" (e 3 Extension.extensions_info))  
)
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, 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

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Linux/OpenSSL</th>
<th>nqsb Unikernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>1600</td>
<td>48</td>
</tr>
<tr>
<td>Runtime</td>
<td>689</td>
<td>25</td>
</tr>
<tr>
<td>Crypto</td>
<td>230</td>
<td>23</td>
</tr>
<tr>
<td>TLS</td>
<td>41</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2560</strong></td>
<td><strong>102</strong></td>
</tr>
</tbody>
</table>

(numbers in kloc)

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

<table>
<thead>
<tr>
<th></th>
<th>nqsb</th>
<th>OpenSSL</th>
<th>PolarSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>698 hs/s</td>
<td>723 hs/s</td>
<td>672 hs/s</td>
</tr>
<tr>
<td>DHE-RSA</td>
<td>601 hs/s</td>
<td>515 hs/s</td>
<td>367 hs/s</td>
</tr>
</tbody>
</table>
Throughput

Throughput graph showing the performance of OpenSSL, PolarSSL, and nqsbtls with varying block sizes in 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