NOT-QUITE-SO-BROKEN TLS

David Kaloper Meršinjak
Hannes Mehnert
Peter Sewell
Anil Madhavapeddy

HCSSC, Annapolis, 6 May 2015
• Motivation
• Brief overview of Transport layer security
• Common implementation flaws
• \textit{nqsb}\textendash TLS
• Results: robust, modular, multi-purpose, and not too slow
MOTIVATION

• TLS is the most widely used security protocol on the Internet
• Long history of security issues (56 CVEs in 2014)
• Most implementations are old and written in C
• Specified in loose prose
• No comprehensive test suite
TLS BACKGROUND

CORE FEATURES:

- Secure connection between two endpoints
- Authentication and Confidentiality

LARGE PARAMETER SPACE:

- Family of protocols: SSLv3, TLS 1.0, 1.1, 1.2
- Algorithmic agility (key exchange, symmetric cipher, mac)
- Extensible (Server Name Identification, OCSP, ...)
TLS BACKGROUND

MAJOR TLS ASPECTS:

• Core protocol logic
• Wire representation, fragmentation
• Cryptography: AES/3DES/RC4 in CBC/GCM/CCM, DH/RSA/DSA, RNG
• Authentication via X.509 certificates
• ASN.1 parsing
TLS BACKGROUND

STRUCTURE:

Record layer multiplexes five sub-protocols:

- Handshake (negotiates security parameters)
- Change Cipher Spec (activates security parameters)
- Alert (error signalling)
- Heartbeat (keep-alive, PMTU discovery)
- Application Data (payload)

And *fragments* them on both record and sub-protocol level.
TLS BACKGROUND

HANDSHAKE

Live handshake visualisation

- Initiated by client
- Server selects version, ciphersuite
- All forms of authentication: mutual, one side, none
- Key exchange
WHY IS TLS BROKEN?

- Prose specification written as RFCs (30000 lines)
- Lack of test suites
- Absence of negative tests: security is undermined by bad error handling
- Implemented in error-prone languages, by large and weakly modular code bases
- Speed often takes precedence
### 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>
WHY IS TLS BROKEN?

OpenSSL tree has > 350000 LOC.

```c
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 the following
    * is in a single record, which is not guaranteed
    * the protocol specification:
    * Byte  Content
    * 0     type     \
    * 1/2   version   > record header
    * 3/4   length    /
    * 5     msg_type  \
    * 6-8   length    > Client Hello message
    * 9/10  client_version  /
    */
    char *buf = &buf_space[0];
    unsigned char *p,*d,*d_len,*dd;
    unsigned int i;
}
```

56835 of those lines are ASM-generating Perl.
EXAMPLE BROKENNESS:

MEMORY

Heartbleed (OpenSSL)

/* Read type and payload length first */

```
- hbtype = *p++;
- n2s(p, payload);
+ if (1 + 2 + 16 > s->s3->rrec.length)
  + return 0; /* silently discard */
+ hbtype = *p++;
+ n2s(p, payload);
+ if (1 + 2 + payload + 16 > s->s3->rrec.length)
  + return 0; /* silently discard per RFC 6520 sec. 4 */

pl = p;
```
EXAMPLE BROKENNESS:
ASN.1 PARSING
CVE-2014-1568 (NSS)

AlgorithmIdentifier ::= SEQUENCE {
  algorithm  OBJECT IDENTIFIER,
  parameters ANY DEFINED BY algorithm OPTIONAL
-- contains a value of the type
-- registered for use with the
-- algorithm object identifier value
}

The contents of the optional parameters field will vary according to the algorithm identified.
EXAMPLE BROKENNESS:
ASN.1 PARSING
CVE-2014-1568 (NSS)

/* make sure the "parameters" are not too bogus. */
if (di->digestAlgorithm.parameters.len > 2) {
  goto sigloser;
}

In reality, for TLS, it is OID for ECDSA or NULL for everything else.
EXAMPLE BROKENNESS:
STATE MACHINE ERRORS

#gotofail

```c
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
```
NQSB-TLS

TLS developed from scratch with dual goals:

- Act as specification
  - test oracle
  - test generator*
- And a usable TLS implementation
  - Unix
  - MirageOS
  - C shared object*

(*-work in progress)
ASIDE:

MIRAGEOS

A "library operating system".

- Runs OCaml programs directly on top of Xen hypervisor.
- OS services implemented as OCaml libraries.
- The runtime environment is assembled statically -- both the OS and application code compile down to a unikernel.
ASIDE:

MIRAGEOS

Diagram showing the structure of MirageOS, with layers of configuration files, application binary, language runtime, OCaml compiler, kernel threads, user processes, filesystem, network stack, hypervisor, and hardware. The diagram also shows the unikernel concept.
NQSB APPROACH

Explore using a memory-safe, functional language for the system itself, as opposed to the tooling.

Create an implementation that encodes the specification in a clear and concise way.
NQSB APPROACH

Designed for clarity and readability.

Written in a memory-safe language.

Mostly restricted to the pure subset of OCaml:

- Explicit flows of data
- No side-effects
- Value-based, explicit error handling
nqsb-TLS ML module layout
CORE

Is purely functional:

```ocaml
val handle_tls : state -> Cstruct.t ->
[ `Ok of state * Cstruct.t option * Cstruct.t option |
  `Fail of failure ]
```
CORE

Takes advantage of ADT and the type system help enforce the state-machine invariants.

```ocaml
let handle_handshake ss hs buf =  
  match parse_handshake buf with  
  | Or_error.Error re -> fail (`Fatal (`ReaderError re))  
  | Or_error.Ok handshake ->  
    match ss, handshake with  
    | AwaitClientHello, ClientHello ch ->  
      answer_client_hello hs ch buf  
      (* ... *)  
    | AwaitClientFinished (session, log), Finished fin ->  
      answer_client_finished hs session fin buf log  
    | Established, ClientHello ch ->  
      answer_client_hello_reneg hs ch buf  
    | AwaitClientHelloRenegotiate, ClientHello ch ->  
      answer_client_hello_reneg hs ch buf  
    | _, hs -> fail (`Fatal (`UnexpectedHandshake hs))
```
CRYPTOGRAPHY

Symmetric cipher and hash cores are implemented in C because:

- performance
- timing side-channel issues

However C code does not allocate and is restricted to minimal control-flow.

Public-key cryptographic primitives are, on the other hand, OCaml/GMP.
ASN.1

TBSertificate ::= SEQUENCE {

  version                [0]  Version DEFAULT v1,
  serialNumber           CertificateSerialNumber,
  signature               AlgorithmIdentifier,
  issuer                  Name,
  validity                Validity,
  subject                  Name,
  subjectPublicKeyInfo    SubjectPublicKeyInfo,
  issuerUniqueID          [1]  IMPLICIT UniqueIdentifier OPTIONAL,
  subjectUniqueID         [2]  IMPLICIT UniqueIdentifier OPTIONAL,
  extensions              [3]  Extensions OPTIONAL

}
Combinatory parsing -- bidirectional.
RESULTS

TEST ORACLE

- Successfully validated 30000 recorded traces of several prior versions.
- Analysis of failures discovered further deviations from the RFC in the deployed stacks:
  - Additional length field in padding extension
  - Unknown alert code
  - 0-padded DH representation
## RESULTS

Full system on top of Xen has 1/25 the TCB of OpenSSL on Linux.

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Linux/OpenSSL</th>
<th>nqsb Unikernel (C code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>1600</td>
<td>48 (36)</td>
</tr>
<tr>
<td>Runtime</td>
<td>689</td>
<td>25 (6)</td>
</tr>
<tr>
<td>Crypto</td>
<td>230</td>
<td>23 (14)</td>
</tr>
<tr>
<td>TLS</td>
<td>41</td>
<td>6 (0)</td>
</tr>
<tr>
<td>Total</td>
<td>2560</td>
<td>102 (56)</td>
</tr>
</tbody>
</table>

(numbers in kloc)
# RESULTS

## PERFORMANCE

<table>
<thead>
<tr>
<th></th>
<th>nqsb</th>
<th>OpenSSL/Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>639 hs/s</td>
<td>573 hs/s</td>
</tr>
<tr>
<td>RSA-DH</td>
<td>374 hs/s</td>
<td>360 hs/s</td>
</tr>
<tr>
<td>AES128-CBC/SHA1</td>
<td>203 MB/s</td>
<td>285 MB/s</td>
</tr>
</tbody>
</table>
RESULTS

PERFORMANCE

- Handshakes per second are similar to OpenSSL.
- Around 70% of OpenSSL throughput with AES-NI.
- 72% of time spent in the SHA1 core (in C).
RESULTS

BTC PIÑATA

- Online since 10th February 2015
- 50000 unique IPs
- 1000 unique IPs for TLS
  - 13500 client traces
  - 6500 server traces
- 42 different X.509 certificates recorded
PIÑATA RESULT

We were exposed to many attacks targeting flaws in the other stacks.

Unsurprisingly, without success.

(But we don't claim we can conclude much from that.)
VARIOUS ROLLOUTS

Slowly moving OCaml Labs infrastructure to nqsb-TLS.
It is being indirectly used by about 50 libraries in the OCaml package ecosystem.
FUTURE

- Finish the Unix .so version targeting OpenBSD's libtls
- Expand flexibility in choice points
- Re-purpose as test generator and run them against other stacks
- Implement known attacks as traces to replay
- Finish TLS features: resumption, ECC
CONCLUSION

- Robust, widely interoperable TLS implementation with tiny TCB
- Multi purpose: usable TLS library, test oracle
- Targets various platforms
- Applicable to other mainstream protocols as well
- Purely functional approach works out for systems software!
CONCLUSION

- nqsb-TLS - https://github.com/mirleft/ocaml-tls
- TLS Demo - https://tls.openmirage.org
- BTC Piñata - http://ownme.ipredator.se