DIY Internet with MinimaLT

Low-latency secure networking

JSConf.EU 2013

Andy Wingo

wingo@igalia.com

Compiler hacker at Igalia

Recently: ES6 generators in V8, SpiderMonkey (sponsored by Bloomberg)

Not a cryptographer

This talk is for folks that deploy both endpoints, for cryptonerds, and for early-stage tinkerers

You are here

Context: Militarization of daily life

Generals peeping on your web searches

Read the wrong things and they send the SWAT team

what’s he building in there?

what’s he building in there?

He has subscriptions to those RSS feeds
And he’s been tweeting about MinimaLT
We’re in his router, and his mobile phone
You won’t believe what we got from the drone
What’s he building in there?
What the hell is he building in there?
We have a right to know

Solution?

Smash the state!

Meanwhile, let’s not make it easy for the NSA

HTTPS vs…

Attack vectors:

- Cryptanalysis (RC4)
- MITM via rogue certificates (DigiNotar &c)
- Use JavaScript! CRIME, BEAST, …
- Backdoors in TLS implementations (Windows?)

HTTPS vs…

Attack vectors:

- Cryptanalysis (RC4)
- MITM via rogue certificates (DigiNotar &c)
- Use JavaScript! CRIME, BEAST, …
- Backdoors in TLS implementations (Windows?)
- **HTTP**

HTTPS vs HTTP

"Cryptography that is not actually used can be viewed as the ultimate disaster" – DJB

`competitions.cr.yp.to/disasters.html`

How many of you...

HTTPS vs HTTP

"Cryptography that is not actually used can be viewed as the ultimate disaster" – DJB

`competitions.cr.yp.to/disasters.html`

How many of you…

❧ use EFF's "HTTPS everywhere" extension?

HTTPS vs HTTP

"Cryptography that is not actually used can be viewed as the ultimate disaster" – DJB

`competitions.cr.yp.to/disasters.html`

How many of you…

- use EFF's "HTTPS everywhere" extension?
- never use plain HTTP with Google?

HTTPS vs HTTP

“Cryptography that is not actually used can be viewed as the ultimate disaster” – DJB

`competitions.cr.yp.to/disasters.html`

How many of you…

- use EFF’s “HTTPS everywhere” extension?
- never use plain HTTP with Google?

There is a reason for this

Anatomy of a GET

`000.00  → www.gnu.org  TCP   SYN`

Visiting `http://www.gnu.org/` over French wired ADSL.

Anatomy of a GET

```
000.00  → www.gnu.org  TCP   SYN
130.50  ← www.gnu.org  TCP   SYN/ACK
```

130 ms RTT, ~65ms latency.

Remote server hosted in Boston, ~4000 miles away.

4000 miles is 22 light-milliseconds.

Anatomy of a GET

```
000.00  → www.gnu.org  TCP   SYN
130.50  ← www.gnu.org  TCP   SYN/ACK
130.78  → www.gnu.org  HTTP  GET /
```

The GET is delayed by 130 ms.

Anatomy of a GET

```
000.00  → www.gnu.org  TCP   SYN
130.50  ← www.gnu.org  TCP   SYN/ACK
130.78  → www.gnu.org  HTTP  GET /
278.00  ← www.gnu.org  TCP   [begin]
```

Begin receiving response. Early parsing.

Anatomy of a GET

```
000.00  → www.gnu.org  TCP   SYN
130.50  ← www.gnu.org  TCP   SYN/ACK
130.78  → www.gnu.org  HTTP  GET /
278.00  ← www.gnu.org  TCP   [begin]
282.00  → www.gnu.org  TCP   SYN x 3
```

Kick off more connections for parallel fetch.

Anatomy of a GET

```
000.00  → www.gnu.org  TCP   SYN
130.50  ← www.gnu.org  TCP   SYN/ACK
130.78  → www.gnu.org  HTTP  GET /
278.00  ← www.gnu.org  TCP   [begin]
282.00  → www.gnu.org  TCP   SYN x 3
410.71  ← www.gnu.org  HTTP  200 OK
```

Total: 7108 bytes over 411 milliseconds.

Anatomy of a GET

```
000.00  → www.gnu.org  TCP   SYN
130.50  ← www.gnu.org  TCP   SYN/ACK
130.78  → www.gnu.org  HTTP  GET /
278.00  ← www.gnu.org  TCP   [begin]
282.00  → www.gnu.org  TCP   SYN x 3
410.71  ← www.gnu.org  HTTP  200 OK
414.85  → www.gnu.org  TCP   SYN/ACK x 3
```

Initial round-trip kills parallel fetch :-(

HTTPS sadness

```
000.00  → www.gnu.org  TCP   SYN
```

HTTPS sadness

```
000.00  → www.gnu.org  TCP   SYN
129.91  ← www.gnu.org  TCP   SYN/ACK
130.46  → www.gnu.org  TLS   Client Hello
```

HTTPS sadness

```
000.00  → www.gnu.org  TCP   SYN
129.91  ← www.gnu.org  TCP   SYN/ACK
130.46  → www.gnu.org  TLS   Client Hello
266.13  ← www.gnu.org  TLS   Server Hello
267.08  ← www.gnu.org  TLS   Certificate
267.73  → www.gnu.org  TLS   Key Exchange
```

HTTPS sadness

```
000.00  → www.gnu.org  TCP   SYN
129.91  ← www.gnu.org  TCP   SYN/ACK
130.46  → www.gnu.org  TLS   Client Hello
266.13  ← www.gnu.org  TLS   Server Hello
267.08  ← www.gnu.org  TLS   Certificate
267.73  → www.gnu.org  TLS   Key Exchange
449.06  ← www.gnu.org  TCP   ACK (???)
449.10  → www.gnu.org  TLS   Change Cipher
```

HTTPS sadness

```
000.00  → www.gnu.org  TCP   SYN
129.91  ← www.gnu.org  TCP   SYN/ACK
130.46  → www.gnu.org  TLS   Client Hello
266.13  ← www.gnu.org  TLS   Server Hello
267.08  ← www.gnu.org  TLS   Certificate
267.73  → www.gnu.org  TLS   Key Exchange
449.06  ← www.gnu.org  TCP   ACK (???)
449.10  → www.gnu.org  TLS   Change Cipher
580.28  ← www.gnu.org  TLS   Change Cipher
583.72  → www.gnu.org  HTTPS GET /
```

HTTPS sadness

```
000.00  → www.gnu.org  TCP   SYN
129.91  ← www.gnu.org  TCP   SYN/ACK
130.46  → www.gnu.org  TLS   Client Hello
266.13  ← www.gnu.org  TLS   Server Hello
267.08  ← www.gnu.org  TLS   Certificate
267.73  → www.gnu.org  TLS   Key Exchange
449.06  ← www.gnu.org  TCP   ACK (???)
449.10  → www.gnu.org  TLS   Change Cipher
580.28  ← www.gnu.org  TLS   Change Cipher
583.72  → www.gnu.org  HTTPS GET /
764.97  ← www.gnu.org  HTTPS 200 OK
```

… and then the CSS, the JS, …

MinimaLT, a low-latency networking protocol

“properly implemented, strong crypto”

… that connects faster than TCP

SYN/ACK – Just say no!

Properly implemented, strong crypto

Uses high-level NaCl library from `@hashbreaker` and `@hyperelliptic`

Avoids many HTTPS/TLS pitfalls

- Well-chosen cyphers
- Timing-independent implementation
- No plaintext (HTTP) mode

MinimaLT adds forward secrecy

Minimal latency

1 round trip if you need “DNS” lookup

0 otherwise

Persistent tunnels

Tunnels can migrate over IP changes – invisible to applications

A protocol for today’s internet

UDP-based

Reliable: replaces TCP + TLS

Denial-of-Service (DoS) resistance

Low overhead, scales to tens of Gb/s

Tunnels and connections

Tunnels multiplex *connections*

Connection 0 is the control connection

- flow control
- connection creation
- authentication (client certs)

Multiple connections can proceed concurrently

QUIC more advanced here in some ways

Wire protocol

```
  +-----------------------+
c | Ethernet, IP, UDP     | 42 bytes
l |-----------------------|
e | Tunnel ID, Nonce      | 16 bytes
a |-----------------------|
r | Ephemeral public key  | 32 bytes (first)
  |=======================|
c | Checksum              | 16 bytes
y |-----------------------|
p | Seq, Ack              | 8 bytes
h |-----------------------|
e | Payload               |
r | ...                   |
```

Crypto

NaCl “box”:

```
+-------------+ C'→S'
| Cyphertext |    n
+-------------+
```

Tunnel ID (TID): a random 64-bit number, provided by client when creating the tunnel

After first packet, TID looks up `C'→S'`: the shared secret

Protocol to change TID and evolve shared secret for forward security

How to get server's public key?

TLS:

- Client knows address of DNS provider
- DNS gives server address (maybe)
- Client connects to server, server provides certificate
- Client verifies cert. using public key infrastructure (PKI)

How to get server's public key?

MinimaLT:

- Client knows address, long-term key of Directory Service
- Server registers address, port, long-term public key and ephemeral public key with DS
- Client asks DS for server info, trusts DS

Servers could register info in DNS records with suitably low TTL (TBD)

Directory server protocol

At first lookup of any name, or at boot:

- 1 round-trip to fetch DS's ephemeral key

To look up a name:

- 1 round-trip using fresh ephemeral client key, DS's ephemeral key

Authenticated and encrypted

Performance

The “expensive” part: establishing the shared secret via Curve25519, which happens when tunnels are created.

- 8000 connections/s/core on modern x86
- ~750 connections/s/core on modern ARM (estimate)

Afterwards, MinimaLT can saturate Gb/s links

Denial-of-Service

Why is MinimaLT able to avoid 3-way handshake?

- A server can slow down clients arbitrarily using puzzles
- Clients may have to "mine for bitcoins"
- Puzzles can be sent at any point (tunnel GC)
- Pre-RT responses should be smaller than requests (hello DNSSEC)

Amplification vs latency?

In general, response can be larger than the request (e.g. HTTP GET)

Does the client IP (spoofable cleartext) correspond to the client request (authenticated, tamper-proof)?

One round trip seems needed in general :-(

Mitigated by long-term tunnels, multiplexed connections

No worse than TCP

Faster than TCP

0RT connects faster than TCP at any latency above 0.5 ms (150 km)

Always faster than OpenSSL

At 64ms latency: 130ms full connection, request, response vs 516ms for OpenSSL

Compare to 278ms for HTTP

Tor-friendly

Project status

University of Illinois at Chicago research project (Jon Solworth)

Very 2013

Ethos, new Xen-based OS

- Security-focused
- Typed filesystem, typed IPC
- Written in C and Go

`http://ethos-os.org/`

W. Michael Petullo doing MinimaLT

MinimaLT: remote IPC for Ethos

```
res := <-Ipc("example.com", "http",
             "GET", "/")

res := <-Ipc("example.com", "foo",
             &Foo{bar:42, baz:"qux"})
```

And POSIX?

Ongoing work to make a shared library; expect it out shortly

```
minimalt_connection*
minimalt_connect_and_write
    (char *host, char *service,
     uint8_t *data, size_t count);
```

Probably not RPC-based – type tools are a mess

And JavaScript?? :)

Upcoming: Libuv integration, and from there to Node

- MinimaLT needs an event loop running, somehow

Pure-JS reliability layer?

- Experiments in congestion control

On the front lines

Bandwidth goes up, but latency stays the same.

There is demand for privacy at low latency: demand for a new protocol.

Go forth and hack!

MinimaLT @ ACM CCS 2013 – Here (Berlin) in Nov.

SYN/ACK – Just say no!

`@andywingo` for slides, upcoming lib release