# rustls

modern, fast, safer TLS

#### Introduction

#### This talk:

- Small dose of memory safety evangelism
- Currently implemented features
- TLS vulnerabilities & rationale for rustls's design
- Performance
- Testing
- Future work

#### Introduction

#### rustls:

- TLS1.2 and TLS1.3 TLS library
- Written in Rust
- MIT/Apache/ISC triple licensed

#### Timeline:

- Born: 2nd May 2016
- First handshake: 27th May 2016
- First release: 26th August 2016

# Let's talk about memory safety

### "About the security content of iOS 10.3.3"

Stand-in for a large piece of software written in memory-unsafe languages

Multiple vendors -> different dev methodologies, varying tooling, skill levels, etc.

- 47 CVEs total
- **36** CVEs are memory unsafety issues

"A memory corruption issue was addressed with improved bounds checking."

# STRIKE FORCE



# BUT NODE JS IS CRAZY

node.js memory unsafe in all versions prior to v8.0.0, May 2017!

#### Rust

- Memory safe & concurrency safe
- Good FFI for integration into existing systems
- Excellent type system
- Speed -- typically as fast as C, occasionally faster
- No runtime overhead (no GC or interpreter)

#### rustls -- design approach

make something that is easy and **safe** for 90% of uses

but perhaps might not ever cater to the remaining 10%

"if in doubt, leave it out"

don't require configuration for 90% of such uses

sane, well-regarded defaults

# "The most dangerous code in the world: validating SSL certificates in non-browser software" - Boneh et al, ACM CCS'12

"Our main conclusion is that SSL certificate validation is completely broken in many critical software applications and libraries" ...

"For the most part, the actual SSL libraries used in these programs are correct" ...

"The primary cause of these vulnerabilities is the developers' misunderstanding of the numerous options, parameters, and return values of SSL libraries" ...

"many SSL libraries are unsafe by default, requiring higher-level software to correctly set their options, provide hostname verification functions [...]"

"APIs should present high-level abstractions to developers, such as "confidential and authenticated tunnel," as opposed to requiring them to explicitly deal with low-level details such as hostname verification"

#### **Current TLS feature list**

TLS1.2 and TLS1.3-draft-18 (+ drafts 19, 20)

Client and server end

AES-128-GCM, AES-256-GCM, chacha20poly1305 suites (9 in total)

Forward secrecy, always: curve25519, nistp256 or nistp384

Server authentication, always -- integrated cert chain and hostname verification

Optional client authentication

All kinds of resumption

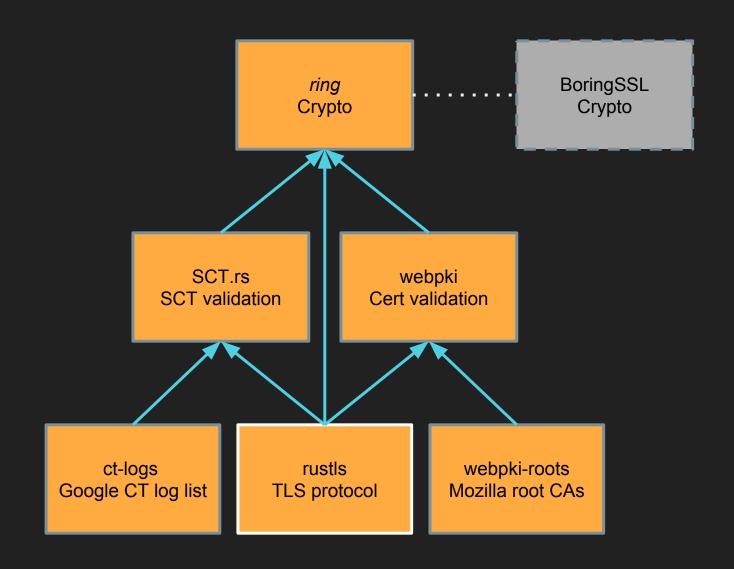
#### Current TLS feature list

Extended master secret support (RFC7627)

OCSP and SCT pinning

Integrated SCT verification

# Dependencies



# Past TLS problems

RC4 BEAST Lucky13 CRIME Logjam



GCM nonces Poodle DROWN SWEET 32 FREAK

GCM nonces

Poodle

DROWN

SWEET 32

FREAK

random start point

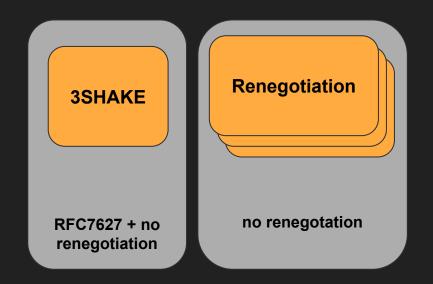
no SSL3

no SSL2

no 64-bit block ciphers

no export suites





# TLS1.3 vs TLS1.2 vs rustls's TLS1.2 subset

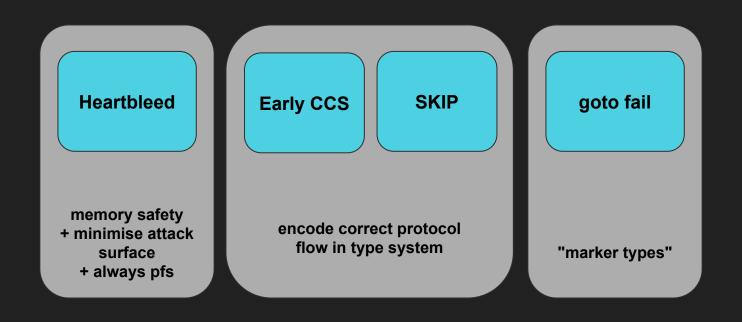
	TLS1.2 standard	TLS1.2 (rustls subset)	TLS1.3 standard
RC4	Yes	No	No
CBC	Mandatory	No	No
Compression	Yes	No	No
Reneg	Yes (+ 2 fixes)	No	No*
PFS	Yes	Mandatory	Mandatory

<sup>\*</sup> renegotation killed, replaced with simple rekeying sub-protocol

# Past TLS implementation problems



# Past TLS implementation problems



## "Marker types"

(please note: unlikely to be a new idea)

#### Generalised "goto fail" problem:

- "Adversary": duplicates any source line into adjacent position
- How do we make a signature verification function that deals with this?
- Problem is fundamentally: absence of an error is a poor indicator of signature validity

#### This idea:

- Unique, zero-sized, behaviourless, explicitly constructed type
- Represents **positive** outcome of signature verification

#### "Marker types"

#### In rustls:

- Protocol states after important verifications require values of these types
- This binds entering those states to the verification
- The compiler then checks we didn't skip verification somehow
- Code review: are these types only constructed at precisely the right point?

Zero run-time cost

### "Marker types"

### **Testing**

automated test suite made up of:

- integration tests against openssl and some public web servers
- api-level tests
- unit tests of library internals
- 'bogo' the BoringSSL test suite, built from the golang TLS stack
  - hugely comprehensive and impressive
  - found some good bugs in rustls
- 'TryTLS' python test suite concentrating on common implementation errors
- performance benchmarks

currently ~95% line coverage

#### Performance

date	send speed (Gbps per core)	recv speed (Gbps per core)
2016-09-04	1.5	2.3
2016-09-20	2.0	6.5
2016-09-27	4.4	6.2
2017-01-26	15.9	15.0
2017-05-10	22.7	21.4

all measurements taken on the same i5-6500 at 3.2GHz TLS1.2, suite TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 OpenSSL 1.1.0f does raw AES128-GCM at ~37 Gbps here

#### Future work

- Work out what to do with TLS1.3 0RTT data
  - feature that can break the normal TLS security guarantees, like replay protection
  - leaky abstraction -- application must know it is happening
  - requires application-level profile rfc -- none exist so currently shouldn't be deployed
- OCSP verification
- PSK support
- Write some glue for use from non-rust programs
  - libtls interface from libressl project seems like a good choice
- Third party audit

#### thanks

Repo: <a href="https://github.com/ctz/rustls">https://github.com/ctz/rustls</a>

Test server: <a href="https://rustls.jbp.io/">https://rustls.jbp.io/</a>

Twitter: @jpixton

Mail: jbp@jbp.io

Slides: <a href="https://github.com/ctz/talks">https://github.com/ctz/talks</a>

# detailed TLS non-feature list

feature	why unsupported
RC4	biases ruin confidentiality
DES/3DES	block size too small
export suites	deliberately bad
CBC suites	IV chaining, mac-then-encrypt,
renegotiatio n	no binding between sessions; underspecified

feature	why unsupported
compressio n	unsafe with mix of secret and attacker-controller data
MD5/SHA1	not collision resistant
static RSA KX	no forward secrecy
discrete log DH	cannot be negotiated in TLS1.2, poor performance

#### about TLS1.0/1.1

hard decision

no obviously safe ciphersuites

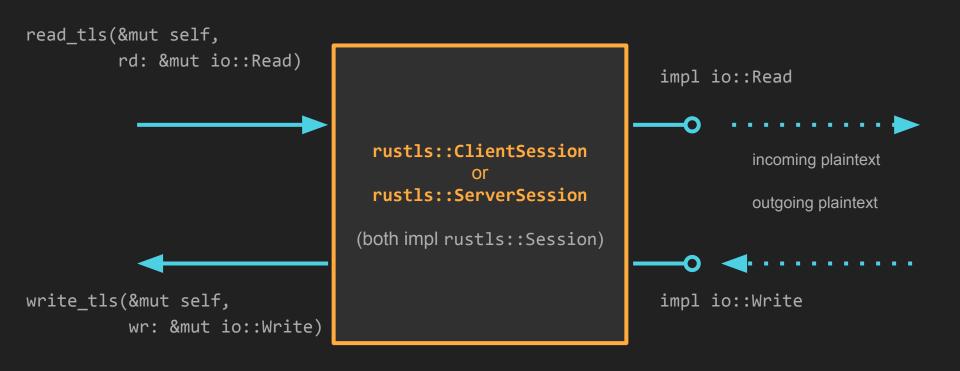
closest has design flaws that resulted in "Lucky13" vuln

countermeasures possible, but unconvincing

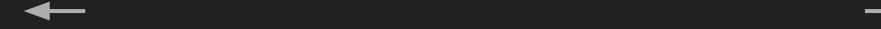
long tail of old/misconfigured servers

early stages of death? (PCI from June 2018, Apple ATS, Cloudflare 'Require Modern TLS')

#### api



network application



#### api

```
pub trait Session: Read + Write + Send {
    fn read_tls(&mut self, rd: &mut Read) -> Result<usize, io::Error>;
    fn write_tls(&mut self, wr: &mut Write) -> Result<usize, io::Error>;
    fn process_new_packets(&mut self) -> Result<(), TLSError>;
(...)
}
```

#### api

to make a rustls::ClientSession you need a rustls::ClientConfig and the server's dns name

- one ClientConfig per process, typically
- contains root certificates
- plus a trait impl that persists data between sessions for resumption

similarly for rustls::ServerSession and rustls::ServerConfig

ServerConfig contains a trait impl that chooses what certificate chain & private key to use for session

### api usability

a rustls::Session buffers outgoing data if needed

- so you can send your HTTP request before the TLS handshake completes
- data only sent if handshake successful
- data sent in same flight as last message in handshake

the 'bring your own IO' interface has moderate usability cost

- allows compatibility with all IO models: non-blocking, thready, async

#### internals

msgs/\*.rs: things that can de/serialise all the TLS types into convenient rust structs

```
Message { typ: Handshake, version: TLSv1_2, payload: Handshake(HandshakeMessagePayload { typ: ServerHello, payload: ServerHello(ServerHelloPayload { server_version: TLSv1_2, random: Random([87, ..., 186]), session_id: SessionID, cipher_suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, compression_method: Null, extensions: [RenegotiationInfo(PayloadU8([])), ServerNameAcknowledgement, SessionTicketAcknowledgement, ECPointFormats([Uncompressed])] }) }) })
```

#### internals

server\_hs.rs/client\_hs.rs: state machine for server/client

- complexity hotspot (~1300/~1500 lines each). ripe for refactoring.
- consists of functions of form:

```
fn handle_server_hello(sess: &mut ClientSessionImpl, m: Message) -> Result<&'static State, TLSError>
{ ... }

pub static EXPECT_SERVER_HELLO: State = State {
    expect: Expectation {
        content_types: &[ContentType::Handshake],
        handshake_types: &[HandshakeType::ServerHello],
    },
    handle: handle_server_hello,
};
```

#### internals

- cipher.rs: TLS record layer encryption. a few implementations of these traits:

```
pub trait MessageDecrypter : Send + Sync {
    fn decrypt(&self, m: Message, seq: u64) -> Result<Message, TLSError>;
}

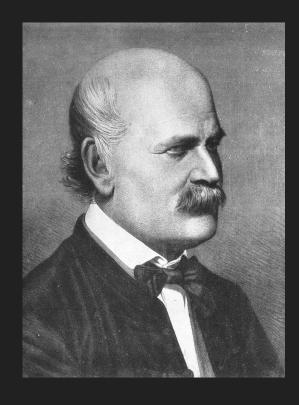
pub trait MessageEncrypter : Send + Sync {
    fn encrypt(&self, m: BorrowMessage, seq: u64) -> Result<Message, TLSError>;
}
```

#### Hand washing

Ignaz Semmelweis (1818-1865)

Noticed two adjacent maternity wards had different postpartum infection rates (10% vs 4% mortality)

One staffed solely by midwives, other by doctors and medical students.



Proposed hand washing between autopsies and deliveries

Mortality rate equalised

#### Hand washing

"Doctors are gentlemen and a gentleman's hands are clean"

"I am a good driver"

"I can write correct code"

Dr. Charles Meigs, 1854

Random driver

Random Programmer