

WORKSHOP ON OPEN SOURCE CRYPTOGRAPHY
30 MARCH 2023

MAKE IT MEMORY SAFE

Adapting curl
to use Rustls



Hi, I'm

J.C. JONES

- Cryptography Engineer & SRE @ Internet Security Research Group (Let's Encrypt)

BUT NOT HERE ON
THEIR BEHALF

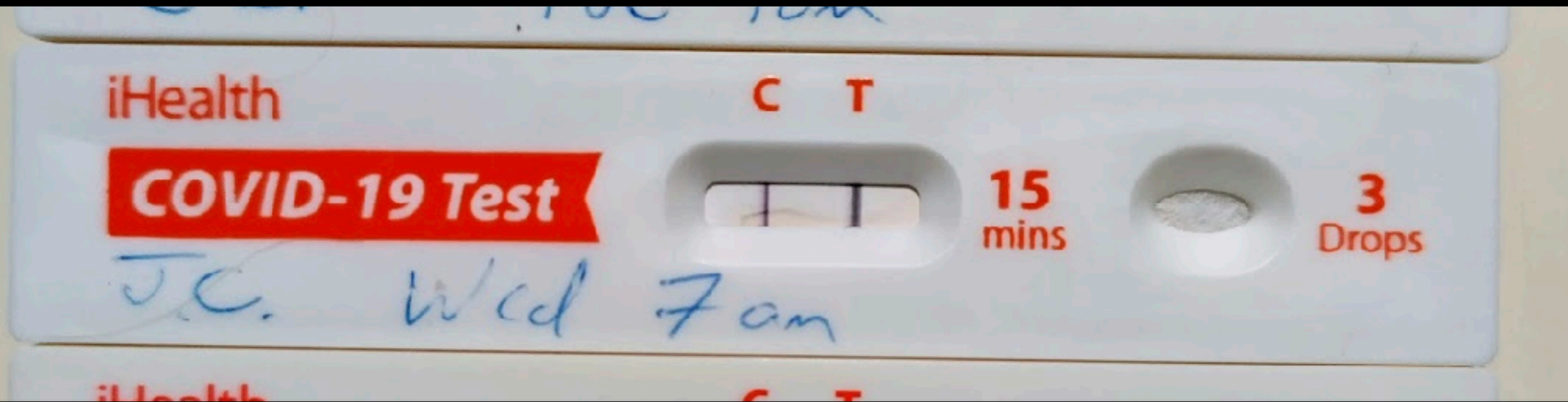
JC@


INSUFFICIENT.COFFEE
LETSencrypt.ORG

[HTTPS://INSUFFICIENT.COFFEE/SOCIAL/](https://insufficient.coffee/social/)



“Oh no”



curl:// + 
RUSTLS

CRYPTO



CRYPTO

OPERATING SYSTEMS

GENESIS OF RUSTLS-FFI

TLS IS
ALWAYS ON
A TRUST
BOUNDARY



PROSSIMO

memorysafety.org

WHY CURL

- Ubiquitous
- Routinely handles untrusted network data
- Mostly written in C



WHY RUST / RUSTLS?

- OpenSSL has no plans to become memory-safe
- Rustls is performant and compatible
- *ring* for crypto primitives



DESIGNING RUSTLS-FFI

REMINDER TO SELF: NOT
A RUST WORKSHOP...

KEEPING IT GENERIC

- FFI : Foreign Function Interface
 - C linking and headers
- Two immediate consumers:
 - libcurl's **vtls** interface
 - Apache's **mod_tls**



Apache Module **mod_tls**

Available Languages: [en](#)

Description:	TLS v1.2 and v1.3 implemented in memory-safe Rust via the rustls library
Status:	Experimental
Module Identifier:	tls_module
Source File:	mod_tls.c
Compatibility:	Available in version 2.4.52 and later

Summary

`mod_tls` is an alternative to [mod_ssl](#) for providing https to a server. It's feature set is a subset, described in more detail below. It can be used as a companion to [mod_ssl](#), e.g. both modules can be loaded at the same time.

`mod_tls`, being written in C, used the Rust implementation of TLS named [rustls](#) via its C interface [rustls-ffi](#). This gives *memory safe* cryptography and protocol handling at comparable performance.

It can be configured for frontend and backend connections. The configuration directive have been kept mostly similar to [mod_ssl](#) ones.

DON'T PANIC

- Exception handling is always a problem crossing languages
 - Panics are undefined across the FFI boundary
- Rustls panics are from memory allocation failures
 - If we catch a panic, can we discard the TLS connection and continue with others?

```
rustls_client_config_builder {
// Turn a *rustls_client_config_builder
// (read-only).
[no_mangle]
c extern "C" fn rustls_client_config_builder(
    builder: *mut rustls_client_config_builder)
-> *const rustls_client_config {
    ffi_panic_boundary! {
        let builder: Box<ClientConfigBuilder> =
            Box::new(builder);
        let config = builder.base.with_client_config(builder);
        let mut config = match builder.config {
            Some(r) => config.with_client_config(r),
            None => config.with_no_client_config(),
        };
        config.alpn_protocols = builder.alpn_protocols;
        config.enable_sni = builder.enable_sni;
        ArcCastPtr::to_const_ptr(config)
    }
}

// "Free" a client_config_builder without
// Normally builders are built into rustls
// and may not be free'd or otherwise used
// Use free only when the building of a
// was created.
[no_mangle]
c extern "C" fn rustls_client_config_builder_free(
    builder: *mut rustls_client_config_builder)
ffi_panic_boundary! {
    BoxCastPtr::to_box(builder);
}
```

AVOIDING SYMBOL AND LOGGING COLLISIONS

- Rustc codgen "metadata" option controls symbol mangling
- Rust logging relies on a singleton
- Allows linking multiple rust libs

```
st: all test-rust
    ./tests/verify-static-libraries.py
    ./tests/client-server.py ./target/client ./target/server

st-rust:
    ${CARGO} test

target:
    mkdir -p $@

src/rustls.h: src/*.rs cbindgen.toml
    cbindgen --lang C > $@

target/$(PROFILE)/librustls_ffi.a: src/*.rs Cargo.toml
    RUSTFLAGS="-C metadata=rustls-ffi" ${CARGO} build --target $@

target/%.o: tests/%.c tests/common.h | target
    $(CC) -o $@ -c $< $(CFLAGS)

target/client: target/client.o target/common.o
    $(CC) -o $@ $^ $(LDFLAGS)

target/server: target/server.o target/common.o
    $(CC) -o $@ $^ $(LDFLAGS)

install: target/$(PROFILE)/librustls_ffi.a
    mkdir -p $(DESTDIR)/lib
    install target/$(PROFILE)/librustls_ffi.a $(DESTDIR)/lib/
    mkdir -p $(DESTDIR)/include
    install src/rustls.h $(DESTDIR)/include/
```

USING RUSTLS-FFI FOR TLS

THE SUPER ABBREVIATED VERSION

REPRESENTING THE TLS STATE MACHINE

OBJECT ORIENTED C



"INTO" PATTERN

rustls_server_config_builder

mut

BUILD

rustls_server_config

rustls_client_config_builder

mut

BUILD

rustls_client_config

"INTO" PATTERN (SERVERS)

rustls_acceptor

mut

ACCEPT

rustls_accepted

mut

ClientHello

INTO(rustls_server_config)

rustls_connection

ACCEPTER, ACCEPTED, CONNECTION

- Allow nonblocking I/O for server connection setups

Struct rustls::server::Acceptor  [-][src]

```
pub struct Acceptor { /* fields omitted */ }
```

[-] Handle on a server-side connection before configuration is available.

The `Acceptor` allows the caller to provide a `ServerConfig` based on the `ClientHello` of the incoming connection.

Implementations

[-] `impl Acceptor` [src]

[+] `pub fn new() -> Result<Self, Error>` [src]

[+] `pub fn wants_read(&self) -> bool` [src]

[+] `pub fn read_tls(&mut self, rd: &mut dyn Read) -> Result<usize, Error>` [src]

[-] `pub fn accept(&mut self) -> Result<Option<Accepted>, Error>` [src]

Check if a `ClientHello` message has been received.

Returns an error if the `ClientHello` message is invalid or if the acceptor has already yielded an `Accepted`. Returns `Ok(None)` if no complete `ClientHello` has been received yet.

Auto Trait Implementations

```
impl !RefUnwindSafe for Acceptor
```

"INTO" PATTERN (CLIENTS)

rustls_client_config

NEW(**Hostname**)

rustls_connection



I/O

- Design of Rustls is agnostic to whether you use Rust's blocking or async I/O.



FOOTGUN: UNINITIALIZED MEMORY

```
/**
 * Read up to `count` plaintext bytes from the `rustls_connection` into `buf`.
 * On success, store the number of bytes read in *out_n (this may be less
 * than `count`). A success with *out_n set to 0 means "all bytes currently
 * available have been read, but more bytes may become available after
 * subsequent calls to rustls_connection_read_tls and
 * rustls_connection_process_new_packets."
 *
 * Subtle note: Even though this function only writes to `buf` and does not
 * read from it, the memory in `buf` must be initialized before the call (for
 * Rust-internal reasons). Initializing a buffer once and then using it
 * multiple times without zeroizing before each call is fine.
 * <https://docs.rs/rustls/0.20.0/rustls/struct.Reader.html#method.read>
 */
rustls_result rustls_connection_read(struct rustls_connection *conn,
                                   uint8_t *buf,
                                   size_t count,
                                   size_t *out_n);
```

WHAT WOULD WE DO DIFFERENTLY?

- Top request: Richer error reporting
 - Breaking API change

NEXT STEPS

- Advocate for more OS pickup
 - Leverage Rust in the Linux Kernel to pave the way
- Some small things need addressing to remove “experimental” markings from curl

EARLY OS ADOPTION



<https://wolfi.dev/>

ISRG / PROSSIMO PLANS

- Pluggable crypto backends, so you can choose alternatives to *ring*
 - `libcrux` :)
- Mutual authentication / Client certificates
- Rustls - OpenSSL C API compatibility layer
- And more:

<https://www.memorysafety.org/>

[initiative/rustls/rustls-work-plan/](https://www.memorysafety.org/initiative/rustls/rustls-work-plan/)

RUSTLS **CAN** REPLACE
OPENSSL

IT'S DANGEROUS TO GO
WITHOUT SAFETY. TAKE THIS.



This has been a lot of words from

J.C. JONES

- Cryptography Engineer & SRE @ Internet Security Research Group (Let's Encrypt)



JC@

INSUFFICIENT.COFFEE
LETS Encrypt.ORG

[HTTPS://INSUFFICIENT.COFFEE/SOCIAL/](https://insufficient.coffee/social/)

