



Replacing OpenSSL,
one step at a time

RustNL 2024

this talk

1

recent rustls rundown

2

rustls-libssl: why

3

rustls-libssl: what

4

rustls-libssl: how

recent rustls rundown

Funding!

Joe [@ctz](#) and Daniel [@cpu](#) full time

Adolfo [@aochagavia](#) and Ferrous Systems
project contracts

Funded by ISRG Prossimo project:
memorysafety.org

recent rustls rundown

Shipped features!

Revocation support with CRLs

Pluggable cryptography providers

FIPS140 support

no_std support

Post-quantum cryptography support

Unbuffered API

} experimental

A large, irregular orange thought bubble with a black outline, containing the text "Let's replace OpenSSL". Below the main bubble are two smaller circles, also in orange with black outlines, connected by a line, suggesting a thought process or a sequence of ideas.

Let's replace
OpenSSL

rustls-libssl: why

non-Rust users *also* deserve memory-safe TLS!

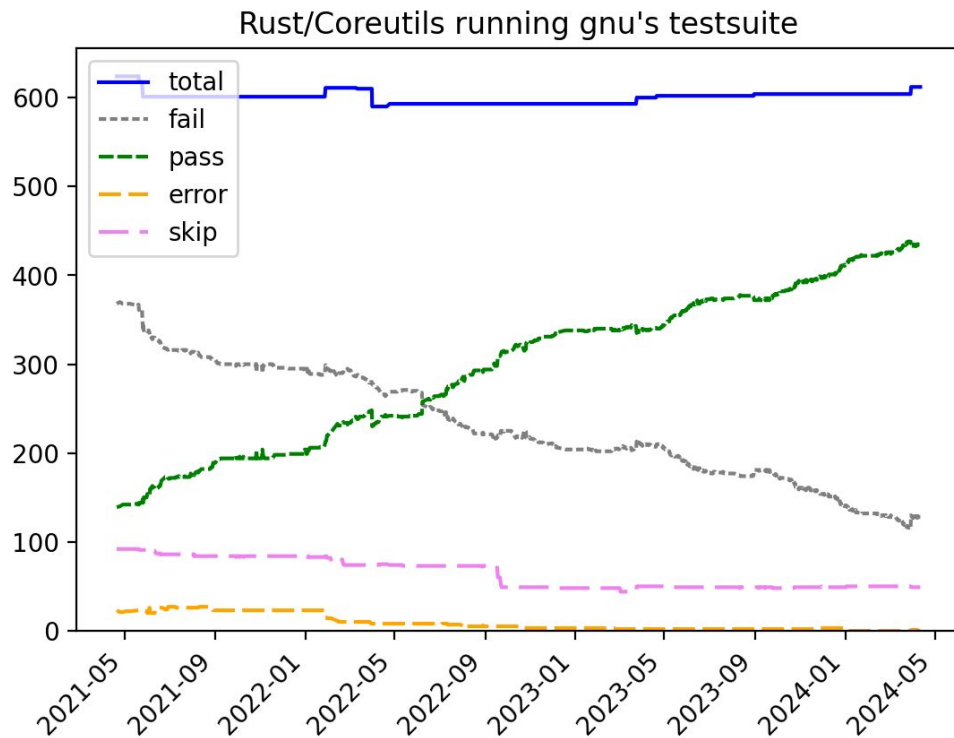
rustls-libssl: why

replacing a memory-unsafe component with a drop-in alternative is a quick way to reduce risks

rustls-lib

replaced
drop-

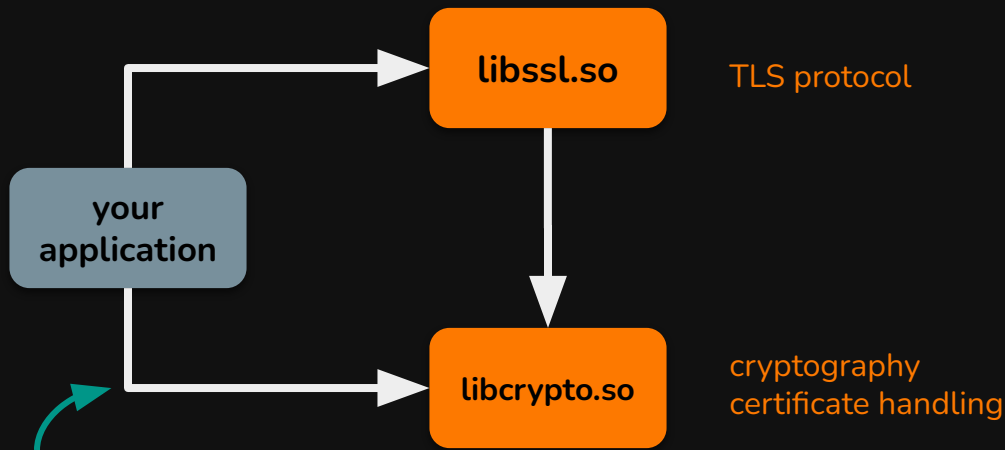
tasks



source: <https://github.com/uutils/coreutils>

rustls-libssl: what

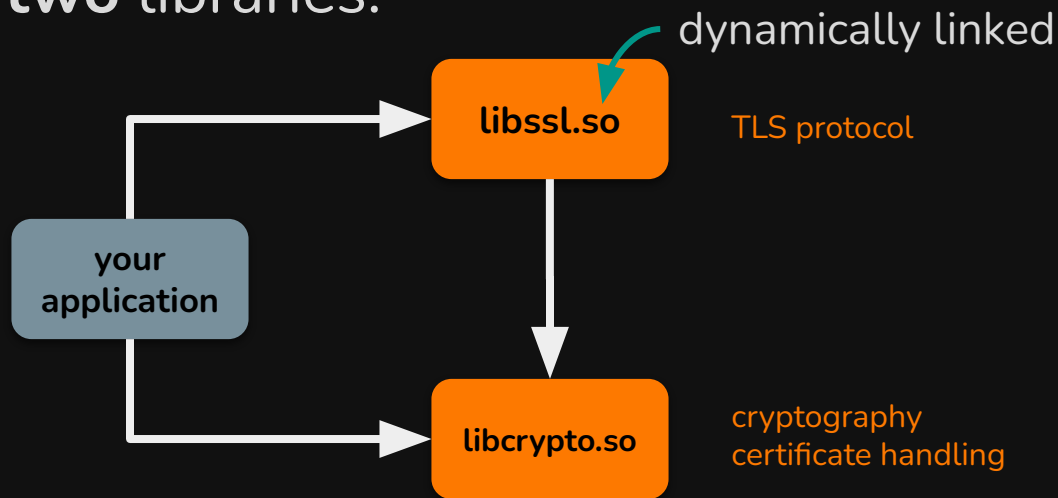
OpenSSL is **two** libraries:



Invariably an application that
uses libssl also uses libcrypto

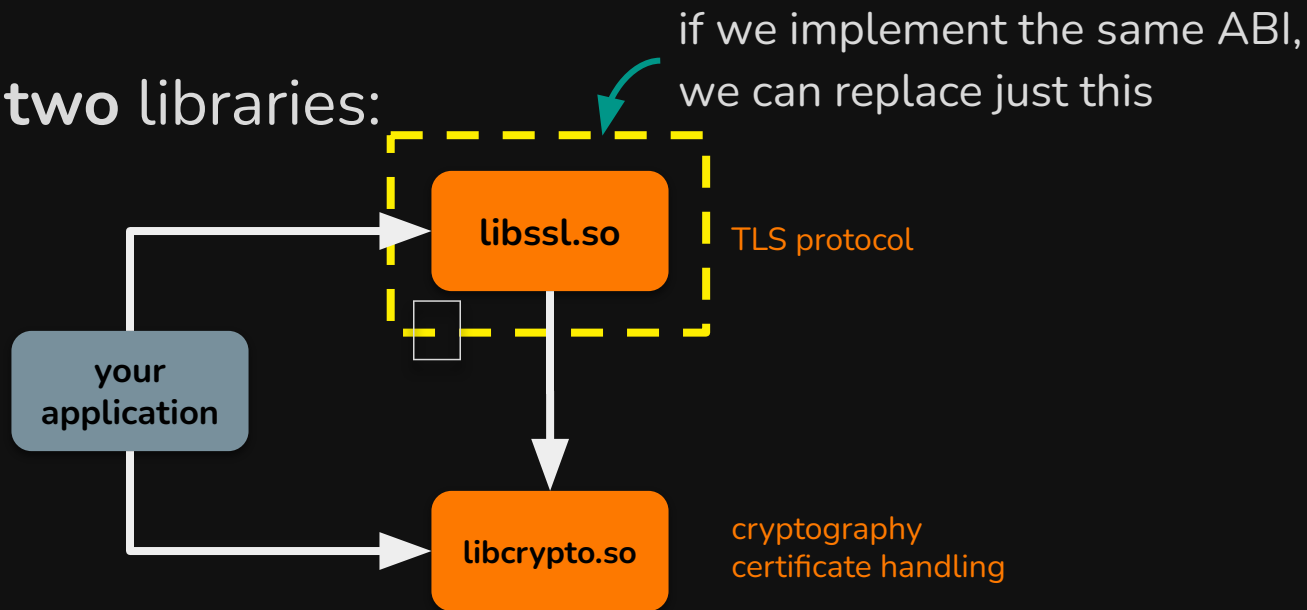
rustls-libssl: what

OpenSSL is **two** libraries:



rustls-libssl: what

OpenSSL is **two** libraries:



OpenSSL libssl is **big!**



136 additional functions are multiplexed!

```
#define SSL_CTX_sess_get_cache_size(ctx) \  
    SSL_CTX_ctrl(ctx,SSL_CTRL_GET_SESS_CACHE_SIZE,0,NULL)
```

(SSL_CTX_ctrl, SSL_ctrl, SSL_CTX_callback_ctrl and
SSL_callback_ctrl are magic like this.)

Fortunately, most of this API surface is rarely used.

eg. curl¹ uses just 57 functions:

BIO_f_ssl	SSL_CTX_set_keylog_callback	SSL_get_peer_cert_chain
OPENSSL_init_ssl	SSL_CTX_set_msg_callback	SSL_get_privatekey
SSL_alert_desc_string_long	SSL_CTX_set_next_proto_select_cb	SSL_get_shutdown
SSL_CIPHER_get_name	SSL_CTX_set_options	SSL_get_verify_result
SSL_connect	SSL_CTX_set_post_handshake_auth	SSL_get_version
SSL_ctrl	SSL_CTX_set_srp_password	SSL_new
SSL_CTX_add_client_CA	SSL_CTX_set_srp_username	SSL_pending
SSL_CTX_check_private_key	SSL_CTX_set_verify	SSL_read
SSL_CTX_ctrl	SSL_CTX_use_certificate_chain_file	SSL_SESSION_free
SSL_CTX_free	SSL_CTX_use_certificate_file	SSL_set_bio
SSL_CTX_get_cert_store	SSL_CTX_use_certificate	SSL_set_connect_state
SSL_CTX_load_verify_dir	SSL_CTX_use_PrivateKey_file	SSL_set_ex_data
SSL_CTX_load_verify_file	SSL_CTX_use_PrivateKey	SSL_set_fd
SSL_CTX_new	SSL_free	SSL_set_session
SSL_CTX_sess_set_new_cb	SSL_get0_alpn_selected	SSL_shutdown
SSL_CTX_set_alpn_protos	SSL_get1_peer_certificate	SSL_write
SSL_CTX_set_cipher_list	SSL_get_certificate	TLS_client_method
SSL_CTX_set_ciphersuites	SSL_get_current_cipher	
SSL_CTX_set_default_passwd_cb	SSL_get_error	
SSL_CTX_set_default_passwd_cb_userdata	SSL_get_ex_data	

¹ curl 7.81.0-1ubuntu1.15 as shipped on Ubuntu 22.04 LTS

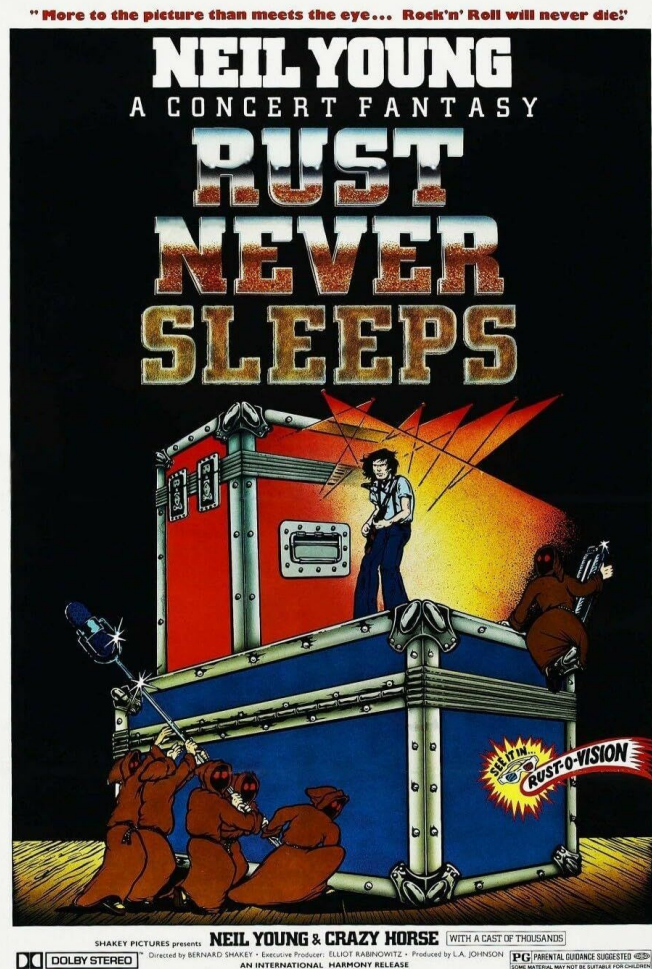
nginx¹ uses 90 functions.

significant overlap with curl: 114 functions cover both

that feels achievable!

¹ nginx 1.18.0-6ubuntu14.4 as shipped on Ubuntu 22.04LTS

let's make
libssl.so
(but in rust)



ingredients



build a C-ABI dynamic library

Cargo.toml:

```
[lib]
name = "ssl"
crate-type = ["cdylib"]
```

ingredients

- ✓ build a C-ABI dynamic library
- ✓ with correctly-versioned symbols

build.rs:

```
let filename = write_version_file();

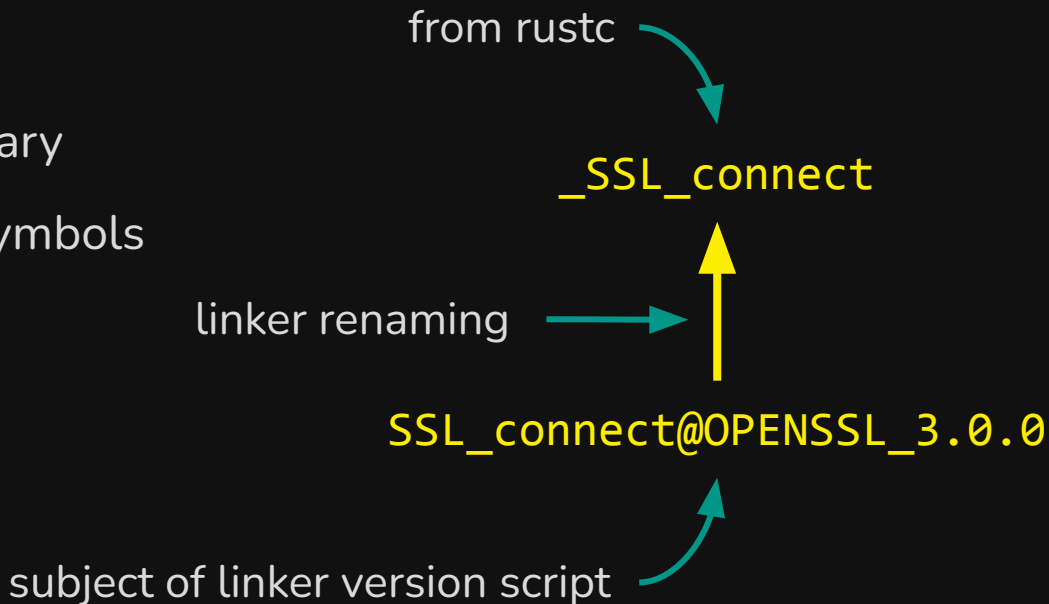
println!("cargo:rustc-cdylib-link-arg="
        "-Wl,--version-script={filename}");

for symbol in ENTRYPOINTS {
    println!(
        "cargo:rustc-cdylib-link-arg="
        "-Wl,--defsym={}=_{}",
        symbol, symbol
    );
}
```

(and then use lld)

ingredients

- ✓ build a C-ABI dynamic library
- ✓ with correctly-versioned symbols



ingredients

- ✓ build a C-ABI dynamic library
- ✓ with correctly-versioned symbols
- ✓ write some rust functions with C linkage

ingredients



build a C-ABI dynamic library



`entry! {` ← `#[no_mangle] and extern "C"`

```
    pub fn _SSL_alert_desc_string(value: c_int) -> *const c_char {
        crate::constants::alert_desc_to_short_string(value).as_ptr() as *const c_char
    }
}
```

ingredients

- ✓ build a C-ABI dynamic library
 - ✓ with correctly-versioned symbols
 - ✓ write some rust functions with C linkage
 - ✓ and avoid undefined behaviour!
-

ingredients




build a C-ABI dynamic library



also wrap the whole function body in
`std::panic::catch_unwind`

```
entry! {  
    pub fn _SSL_alert_desc_string(value: c_int) -> *const c_char {  
        crate::constants::alert_desc_to_short_string(value).as_ptr() as *const c_char  
    }  
}
```



differential testing



write some simple C programs
that use OpenSSL

```
#include <stdio.h>
```

```
#include <openssl/ssl.h>
```

```
int main(void) {  
    for (int i = -1; i < 260; i++) {  
        printf("%d: '%s' '%s'\n", i,  
               SSL_alert_desc_string(i),  
               SSL_alert_desc_string_long(i));  
    }  
    return 0;  
}
```

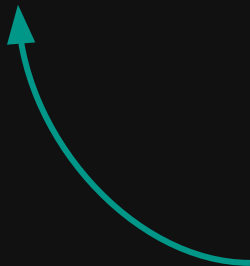
differential testing



write some simple C programs
that use OpenSSL



have them print all return values
and data



should be deterministic: so no pointer
values, or randomised data!

```
-1: 'UK' 'unknown'
0: 'CN' 'close notify'
1: 'UK' 'unknown'
2: 'UK' 'unknown'
3: 'UK' 'unknown'
4: 'UK' 'unknown'
5: 'UK' 'unknown'
6: 'UK' 'unknown'
7: 'UK' 'unknown'
...
```

differential testing



write some simple C programs
that use OpenSSL



have them print all return values
and data



run them against the real
libssl.so and ours

```
$ ./target/constants > original
```

```
$ LD_LIBRARY_PATH=target/debug  
./target/constants > ours
```

differential testing



write some simple C programs
that use OpenSSL



have them print all return values
and data



run them against the real
libssl.so and ours



the output *should* be identical!

```
$ ./target/constants > original
```

```
$ LD_LIBRARY_PATH=target/debug  
./target/constants > ours
```

```
$ diff -su original ours  
Files original and ours are identical
```

miri with FFI calls

miri works very well for pure-rust
libraries

...but not if you make an FFI call

miri with FFI calls

miri works very well for pure-rust
libraries

...but not if you make an FFI call

```
error: unsupported operation: can't call foreign function `X509_STORE_new` on OS `linux`
```

```
--> src/x509.rs:242:27
```

```
242 |         raw: unsafe { X509_STORE_new() },  
    |                        ^^^^^^^^^^^^^^^^^ can't call foreign function  
    |                                         `X509_STORE_new` on OS `linux`
```

```
= help: this is likely not a bug in the program; it indicates that the program  
       performed an operation that the interpreter does not support
```

miri with FFI calls

miri works very well for pure-rust
libraries

...but not if you make an FFI call



Let's send a
pull request
to miri

```
error: unsupported operation: can't call foreign function  
--> src/x509.rs:242:27
```

```
242 |         raw: unsafe { X509_STORE_new() },  
    |                        ^^^^^^^^^^^^^^^^^ can't call foreign function  
    |                                         `X509_STORE_new` on OS `linux`
```

= help: this is likely not a bug in the program; it indicates that the program
performed an operation that the interpreter does not support

miri with FFI calls

miri *already* looks in your crate for native function definitions!

```
#[cfg(miri)]
mod miri {
    pub struct X509_STORE(());

    #[no_mangle]
    pub extern "C" fn X509_STORE_new() -> *mut X509_STORE {
        Box::into_raw(Box::new(X509_STORE(())))
    }

    #[no_mangle]
    pub extern "C" fn X509_STORE_free(ptr: *mut X509_STORE) {
        if ptr.is_null() { return; }
        drop(unsafe { Box::from_raw(ptr) });
    }
}
```


conclusions




we've made a memory-safe¹ replacement for openssl 3.0 libssl.so

¹ mostly

conclusions

 we've made a memory-safe¹ replacement for openssl 3.0 libssl.so

 written in rust, using rustls

¹ mostly

conclusions

- 🦀 we've made a memory-safe¹ replacement for openssl 3.0 libssl.so
- 🦀 written in rust, using rustls
- 🦀 applications don't need recompilation

¹ mostly

conclusions

- 🦀 we've made a memory-safe¹ replacement for openssl 3.0 libssl.so
- 🦀 written in rust, using rustls
- 🦀 applications don't need recompilation
- 🦀 supports a small subset of the openssl 3.0 libssl API
(but enough for nginx and curl²)

¹ mostly

² ubuntu 22.04 LTS versions

conclusions



and it's easy to try:

```
$ with-rustls-libssl curl https://rustls.horse/
```

or

```
# rustls-libssl-nginx enable  
# systemctl daemon-reload  
# service nginx restart
```

thanks!

Repo: <https://github.com/rustls/rustls-openssl-compat>

Mastodon: [@jpixton@octodon.social](https://octodon.social/@jpixton)

Mail: jbp@jbp.io

Slides: <https://github.com/ctz/talks>

