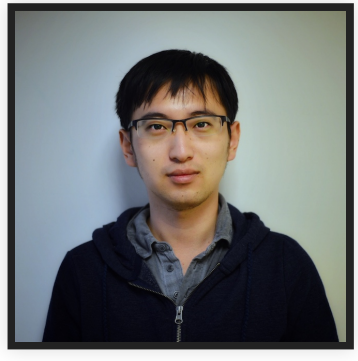
CARGO MEETS AUTOTOOLS (AND CMAKE)

Yiming Jing (荆一明)

百度安全 有AI更安全 Baidu X-Lab

ABOUT ME

- Security scientist at Baidu Security/X-Lab;
- Author of MesaLink, providing OpenSSL-compatible C APIs for the Rust TLS stack.



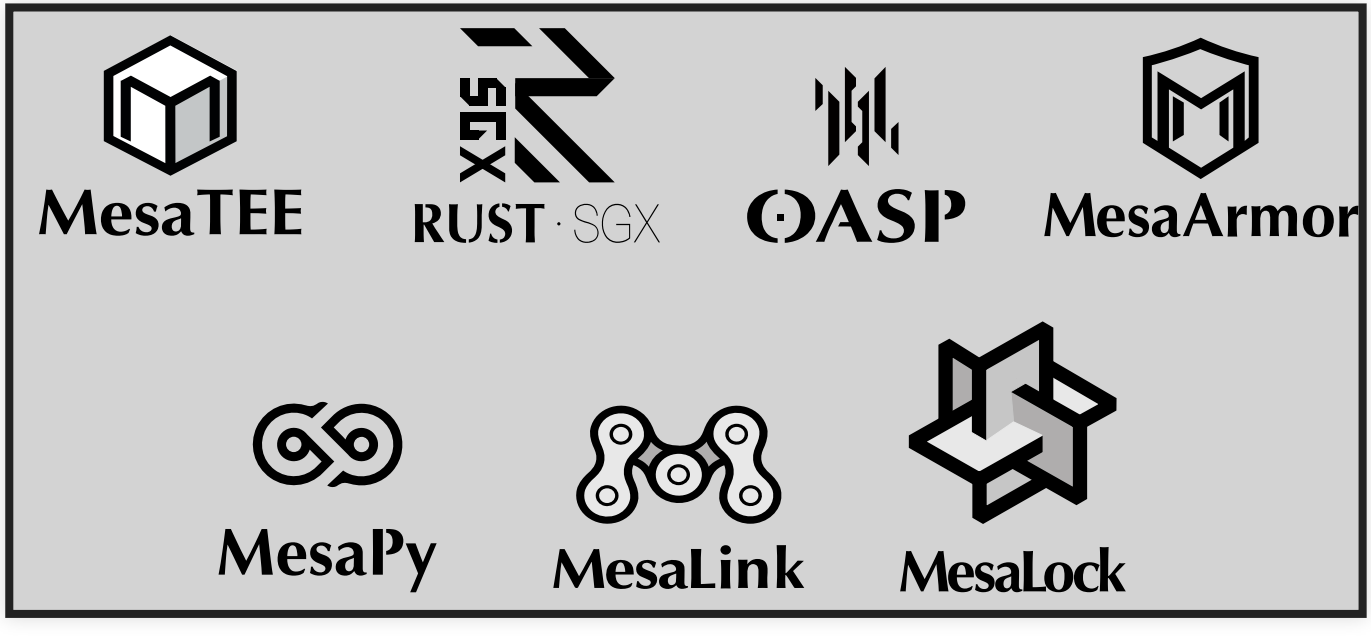
MesaLink

- In production at Baidu since 08/2018;
- 10 million monthly active users as of 12/2018;
- 1.0.0 released on 04/02/2019.

ABOUT BAIDU SECURITY/X-LAB

- Baidu is the 2nd largest search engine in the world;
- Baidu X-Lab is led by Dr. Lenx Wei (韦韬), Chief Security Scientist (T11) of Baidu.

Memory safe projects lineup



OUTLINE OF THIS TALK

Background: What is MesaLink?

"Hey, why don't you use `cargo install`?"

Autotools: ./configure && make

CMake: cmake .. && cmake --build .

Summary and Takeaways

BACKGROUND

- 2014: Heartbleed;
- 2015: BoringSSL forked from OpenSSL;
- 2017.10: We published a short paper on **Hybrid Memory Safety** at ACM CCS 2017;
- 2017.11: We implemented SSL/SSL_CTX with Rust FFI on rustls, webpki, and *ring*.

So we have the MesaLink project.

WHAT IS MESALINK

- A memory safe and OpenSSL-compatible TLS library
- C bindings for rustls, modeled after OpenSSL
- Long term goal: *drop-in replacement* for OpenSSL

WHAT IS THIS TALK ABOUT

- How we build and distribute MesaLink as a C library;
- Gluing Cargo and Autotools/CMake together.

More details about Rust FFI at my RustFest Rome talk,
"One Thousand Ways to Die in Rust FFI"

point · 7 hours ago

mesalink on crates.io, why?

Give Award Share Report Save

points · 3 hours ago

dn't really make sense because the top level interface of mesalink is not a rus
SL-compatible C library.

Give Award Share Report Save

milter 1 point · 3 hours ago

wouldn't really make sense because the top level interface of mesalink is no
's an OpenSSL-compatible C library

have no sense? There is a lot of `-sys` crates, it is just one of them. But in co
r `-sys` crates, there is no need for `pkg-config` inside `build.rs` and so on
ry, just add `c library` as `dependicies` and that's all. Much more simple fo
pare with `cargo install` + `pkg-config`

WHY DON'T YOU USE `cargo install`?

- *"Cargo can only install packages that have binary targets"*, the Rust book, §14.4;
- Unlike `openssl-sys`, MesaLink is for C callers. Rust crates should depend on rustls, not MesaLink;
- We'd like to hide rustc/cargo behind a build system that OpenSSL users are familiar with.

CARGO MEETS AUTOTOOLS

```
./configure && make
```

1. CARGO.TOML: A TALE OF TWO LIBS

```
[lib]
name = "mesalink"
crate-type = ["staticlib", "cdylib"]
```

- `cdylib`: a shared object **without SONAME or version info**; it can be dlopened but can't be linked;
- `staticlib`: an archive of objects.

But we need a linkable shared library.

2. BUILD.RS: GENERATING A .LA FILE

Use libtool-rust or draft your own

```
dlname='libmesalink.so.15'
library_names='libmesalink.so.15.0.0 \
               libmesalink.so.15 libmesalink.so'

old_library='libmesalink.a'

# Linux
inherited_linker_flags=' -pthread -lm -ldl'
# macOS
inherited_linker_flags=' -lm -ldl -lresolv \
                         -framework Security'
```

3. SRC/INCLUDE.AM: LIBTOOL TO HELP

```
libmesalink_la_SOURCES = \
    $(NULL)
libmesalink_la_LIBADD = \
    # libmesalink.la generated by build.rs
    $(MESALINK_LIB_LA)
libmesalink_la_LDFLAGS = \
    -export-dynamic \
    -version-info ${MESALINK_LIBRARY_VERSION} \
    -export-symbols-regex "^mesalink_.*" \
    -Wl,--gc-sections \ # LTO
    $(AM_LDFLAGS)
```

4. CONFIGURE.AC: SETTING OPTIONS

Pass options to cargo and headers:

```
AC_ARG_ENABLE([tls13],
    [AS_HELP_STRING([--enable-tls13],
    [Enable TLS 1.3 (default: enabled)])],
    [ ENABLE_TLS13=$enableval ],
    [ ENABLE_TLS13=yes ]
    )

if test "$ENABLE_TLS13" = "yes"
then
    CARGO_FEATURES=$CARGO_FEATURES" tls13"
    CONFIG_OPTIONS="$CONFIG_OPTIONS HAVE_TLS13"
else
    CONFIG_OPTIONS="$CONFIG_OPTIONS NO_TLS13"
fi
```

5. CALLING CARGO

Pass features, target, and linker options to rustc:

```
cargo rustc --release
    --no-default-features
    --features $(CARGO_FEATURES)

    ### Cross Compile ###
    --target arm-unknown-linux-gnueabi
    -- -C linker=arm-linux-gnu-eabi-gcc
```

BUILDING MESALINK WITH AUTOTOOLS

```
  * Installation prefix:        /usr/local
  * Host:                       x86_64-pc-linux-gnu
  * Rust Host:
  * C Compiler:                 gcc
  * C Compiler vendor:          gnu
  * C Flags:                    -Os -fvisibility=hidden -ffunction-sections -fdata-sections
  * Debug enabled:              no
  * Rust version:               rustc 1.33.0 (2aa4c46cf 2019-02-28)
  * Nightly Rust:               no
  * Examples:                   no

  Features
  * Jemalloc:                   no
  * Logging and error strings:  yes
  * AES-GCM:                    yes
  * Chacha20-Poly1305:          yes
  * TLS 1.3 (draft):            yes
  * X25519 key exchange:        yes
  * EC key exchange:            yes
  * RSA signature verification: yes
  * EC signature verification:  yes
  * SGX attestation:            no

---
yjing@Workstation:~/mesalink [master✓] »
```

CARGO MEETS CMAKE

```
cmake .. && cmake --build .
```

1. A TALE OF TWO LIBS, AGAIN

GNU Libtool does not get along with M$ Windows;

Without libtool, we have to use `cdylib`;

Luckily, we can pass linker options to `rustc`.

2. STAGING LINKER ARGS

Use `-soname` on Linux:

```
set(CARGO_LINKER_ARGS "\
    -C linker=${CMAKE_C_COMPILER} \
    -C link-arg=-Wl,-soname \
    -C link-arg=-Wl,${SONAME}" VERBATIM)
```

2. STAGING LINKER ARGS (CONT'D)

Use `-install_name` on macOS:

```
set(CARGO_LINKER_ARGS "\
    -C linker=${CMAKE_C_COMPILER} \
    -C link-arg=-Wl,-install_name \
    -C link-arg=-Wl,${SONAME} \
    -C link-arg=-Wl,-compatibility_version \
    -C link-arg=-Wl,${PROJECT_VERSION_MAJOR} \
    -C link-arg=-Wl,-current_version \
    -C link-arg=-Wl,${PROJECT_VERSION_MAJOR}.${PROJECT_VERSION
    VERBATIM)
```

3. CMAKELISTS.TXT: SETTING OPTIONS

Pass options to cargo and headers:

```
configure_file(${PROJECT_SOURCE_DIR}/mesalink/options.h.in \
    ${PROJECT_SOURCE_DIR}/mesalink/options.h @ONLY)

option(HAVE_TLS13 "Enable TLS 1.3 (default: enabled)" ON)
if(HAVE_TLS13)
    string(APPEND CONFIG_FEATURES tls13,)
endif()
```

4. CALLING CARGO

Pass features, target, and linker options to rustc:

```
RUSTFLAGS="-C linker=/usr/bin/cc \
    -C link-arg=-Wl,-soname \
    -C link-arg=-Wl,libmesalink.so.15"

cargo build --release
    --no-default-features
    --features $(CARGO_FEATURES)
```

5. CROSS COMPILING IN CMAKE

See cmake/arm-linux-gnueabi.toolchain.cmake

```
# apt-get install gcc-arm-linux-gnueabi g++-arm-linux-gnueabi
#    gcc-arm-linux-gnueabihf g++-arm-linux-gnueabihf \
#    libc6-armel-cross libc6-dev-armel-cross

# rustup target add arm-unknown-linux-gnueabi

set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR arm)
set(CMAKE_C_COMPILER arm-linux-gnueabi-gcc)
set(CMAKE_CXX_COMPILER arm-linux-gnueabi-g++)
set(RUST_TARGET arm-unknown-linux-gnueabi)
```

6. BONUS: WINDOWS INSTALLER

Generate an NSIS installer with CPack

```
mkdir build && cd build
cmake -G "Visual Studio 15 2017 Win64" ..
cmake --build .

cpack -D CPACK_GENERATOR="NSIS64"
```


MesaLink 1.0.0 Setup
Welcome to MesaLink 1.0.0 Setup
Setup will guide you through the installation of MesaLink 1.0.0.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

BUILDING MESALINK WITH CMAKE

```
yjing@Workstation:~/mesalink [master✓] » mkdir build
yjing@Workstation:~/mesalink [master✓] » cd build
yjing@Workstation:~/mesalink/build [master✓] » cmake ..
-- The C compiler identification is GNU 5.4.0
-- The CXX compiler identification is GNU 5.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- w
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Cargo Home: /home/yjing/.cargo
-- Rust Compiler Version: 1.33.0
-- Configuring done
-- Generating done
-- Build files have been written to: /home/yjing/mesalink/build
yjing@Workstation:~/mesalink/build [master✓] »
```

AUTOTOOLS VS CMAKE: A SUMMARY

	Crate-type	Library	LTO	Install
Autotools	`staticlib`	libtool	ld	libtool
CMake	`cdylib`	rustc	rustc	cmake/cp

TAKEAWAYS

- The Rust toolchain is a few steps away from distributing a configurable C library and its headers;
- Autotools/CMake bridges the gap and brings a better experience to Rust non-users.

ACKNOWLEDGEMENTS

- The GNOME librsvg project;
- Autotools: A Practitioner's Guide to GNU Autoconf, Automake, and Libtool;
- CMake Documentation.

THANK YOU

🔗 https://mesalink.io

✉️ jingyiming@baidu.com

百度安全 有AI更安全 Baidu X-Lab

Give us a ⭐ on Github if you like this project!