6-3: Useful crates and tools

Artem Pavlov, TII, Abu Dhabi, 21.04.2025

getrandom

- Retrieves random data from (operating) system sources
- Supports a wide range of targets (approximately the same as Rust's std)
- Can be used in no-std environments (backend must be specified by an application programmer)
- Repository: https://github.com/rust-random/getrandom
- Docs: https://docs.rs/getrandom

subtle

- Traits and utilities for constant-time cryptographic implementations
- Provides traits for constant-time comparison and selection
- Provides constant-time alternatives to bool (Choice) and Option (CtOption)
- Repository: https://github.com/dalek-cryptography/subtle
- Docs: https://docs.rs/subtle

hex-literal

- Provides the hex! macro for converting hexadecimal string literals to a byte array at compile time
- Accepts multiple string literals
- Implemented using const fns
- Docs: https://docs.rs/hex-literal

cpufeatures

- Lightweight and efficient runtime CPU feature detection for aarch64, loongarch64, and x86/x86_64 targets
- An alternative to the is_x86_feature_detected! macro
- Provides “tokens” which serve as a proof that target feature storage was already initialized
- Docs: https://docs.rs/cpufeatures

base16ct, base32ct, base64ct

- Crates for constant time encoding and decoding of base16 (hex), base32, and base64 formats
- Supports different format flavors encountered in the wild
- Constant time only in respect to input content, not length
- Because of the constant time property, the crates do not implement “early failure”
- Repository: https://github.com/RustCrypto/formats

serdect

- Constant-time serde serializer/deserializer helpers for binary secret data
- For human-readable formats (e.g. JSON) serializes data as hex using base16ct
- For binary formats uses binary data as-is
- Docs: https://docs.rs/serdect
- Repository: https://github.com/RustCrypto/formats

zeroize

- Provides functions and traits for zeroization of secrets which will not be optimized aways by compiler
- Zeroization support in cryptographic crates is often optional and enabled using zeroize crate feature
- Warning: does not prevent potential leaks caused by moves and stack spilling!
- Repository: https://github.com/RustCrypto/utils
- Docs: https://docs.rs/zeroize

Memory sanitizers

- Rust is compatible with external memory sanitizers such (e.g. Valgrind or LLVM sanitizers)
- https://doc.rust-lang.org/beta/unstable-book/compiler-flags/sanitizer.html

MIRI

- An interpreter for Rust's mid-level intermediate representation
- Repository: https://github.com/rust-lang/miri
- Installation: rustup +nightly component add miri
- Execution: cargo miri test, cargo miri run

proptest

- A framework for property testing
- Allows to easily write tests which check range of inputs
- Book: https://proptest-rs.github.io/proptest/intro.html
- Docs: https://docs.rs/proptest

proptest example

```
proptest! {
    #[test]
    fn doesnt_crash(s in "\\PC*") {
        parse_date(&s);
    }

    #[test]
    fn parses_date_back_to_original(y in 0u32..10000,
                                    m in 1u32..13, d in 1u32..32) {
        let (y2, m2, d2) = parse_date(
            &format!("{:04}-{:02}-{:02}", y, m, d)).unwrap();
        // prop_assert_eq! is basically the same as assert_eq!, but doesn't
        // cause a bunch of panic messages to be printed on intermediate
        // test failures. Which one to use is largely a matter of taste.
        prop_assert_eq!((y, m, d), (y2, m2, d2));
    }
}
```

cargo fuzz

- Fuzzing with libFuzzer
- libFuzzer requires LLVM sanitizer support, only works on x86-64 Linux, x86-64 macOS and Apple-Silicon (aarch64) macOS
- Also needs a C++ compiler with C++11 support
- Repository: https://github.com/rust-fuzz/cargo-fuzz
- Rust Fuzz Book: https://rust-fuzz.github.io/book/afl.html

cargo afl

- Fuzzing cargo plugin based on American Fuzzy Lop (AFL)
- Requires C compiler (e.g. gcc or clang) and cmake
- Repository: https://github.com/rust-fuzz/afl.rs

Questions?