Integrating Rust into Tor: Successes and Challenges

Chelsea Holland Komlo & isis agora lovecruft

Introduction

isis agora lovecruft: cryptographic design and implementations, security engineering

Chelsea Holland Komlo: Distributed systems, applied cryptography research and implementation.

Thank you

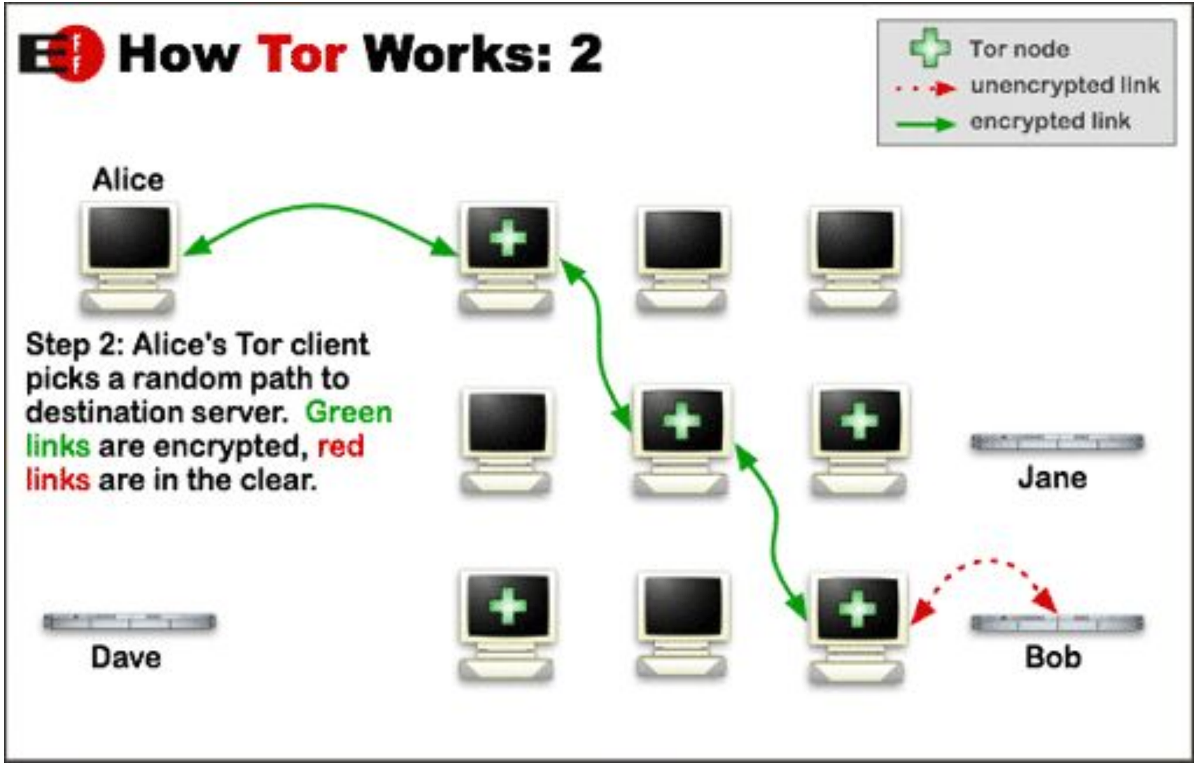
- To members of the Tor network team who provided feedback about issues and have contributed to this effort as well, including Tim Wilson-Brown, Nick Matthewson, Taylor Yu, David Goulet, George Kadianakis, Alexander Færøy, and Mike Perry.
- To Rust people who worked with us in getting through challenges and who were helpful, including Alex Crichton, Manish Goregaokar, Nika Layzell, without boats, Steve Klabnik, Alexis Beingessner, Patrick Walton, and many others. <3

Overview

- What is Tor?
- Where we started
- Where we are
- Where we are going
- What have we learned
- What we hope to see

Overview

- **What is Tor?**
- Where we started with Rust
- Where we are
- Where we are going
- What have we learned
- What we hope to see

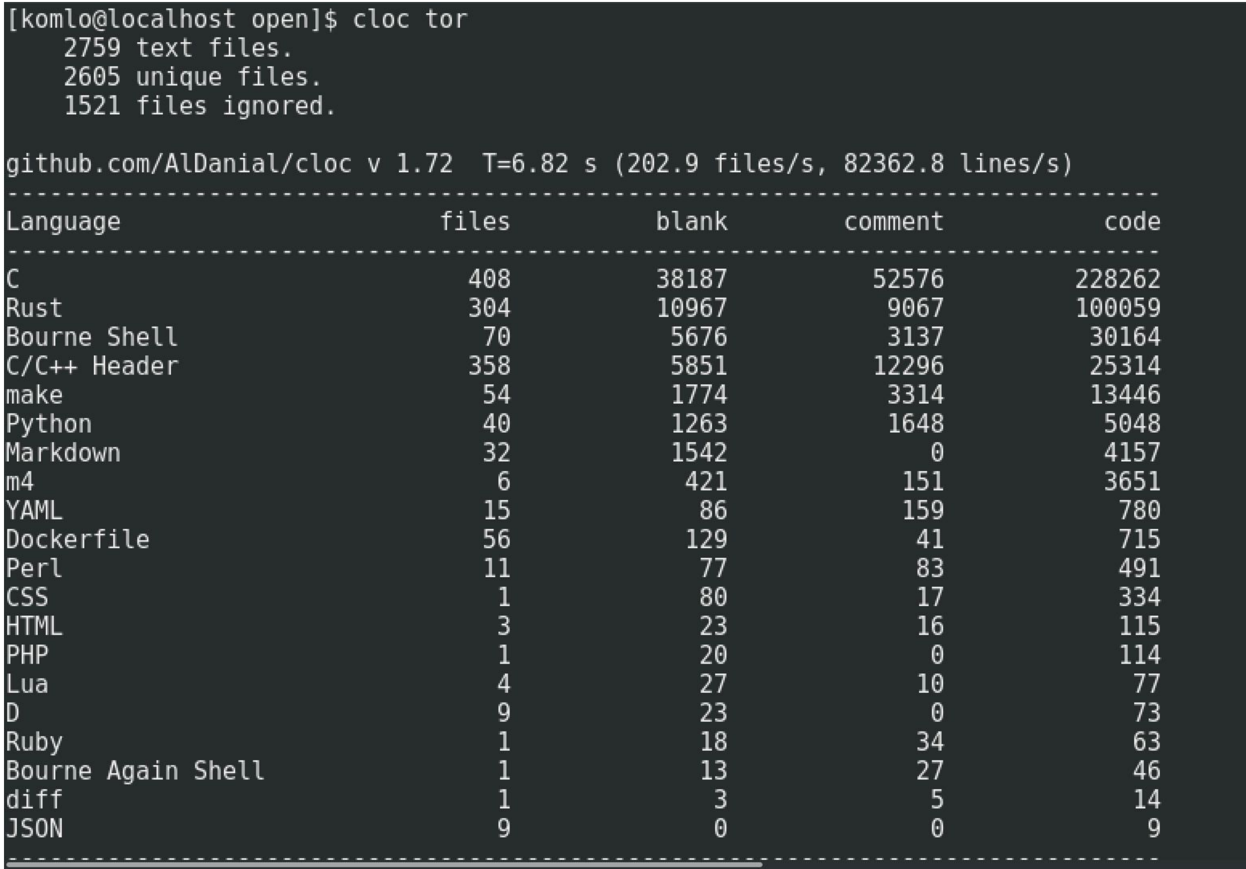


Step 2: Alice's Tor client picks a random path to destination server. Green links are encrypted, red links are in the clear.

Tor is provides anonymity online and (ideally) censorship circumvention via the Tor network, a network of relays run by volunteers

Source: https://www.torproject.org/about/overview

What is tor?



```
[komlo@localhost open]$ cloc tor
    2759 text files.
    2605 unique files.
    1521 files ignored.

github.com/AlDanial/cloc v 1.72  T=6.82 s (202.9 files/s, 82362.8 lines/s)
--------------------------------------------------------------------------------
Language                      files          blank        comment           code
--------------------------------------------------------------------------------
C                               408          38187          52576         228262
Rust                            304          10967           9067         100059
Bourne Shell                     70           5676           3137          30164
C/C++ Header                    358           5851          12296          25314
make                             54           1774           3314          13446
Python                           40           1263           1648           5048
Markdown                         32           1542              0           4157
m4                                6            421            151           3651
YAML                             15             86            159            780
Dockerfile                       56            129             41            715
Perl                             11             77             83            491
CSS                               1             80             17            334
HTML                              3             23             16            115
PHP                               1             20              0            114
Lua                               4             27             10             77
D                                 9             23              0             73
Ruby                              1             18             34             63
Bourne Again Shell                1             13             27             46
diff                              1              3              5             14
JSON                              9              0              0              9
--------------------------------------------------------------------------------
```

Overview

- What is Tor?
- **Where we started with Rust**
- Where we are
- Where we are going
- What have we learned
- What we hope to see

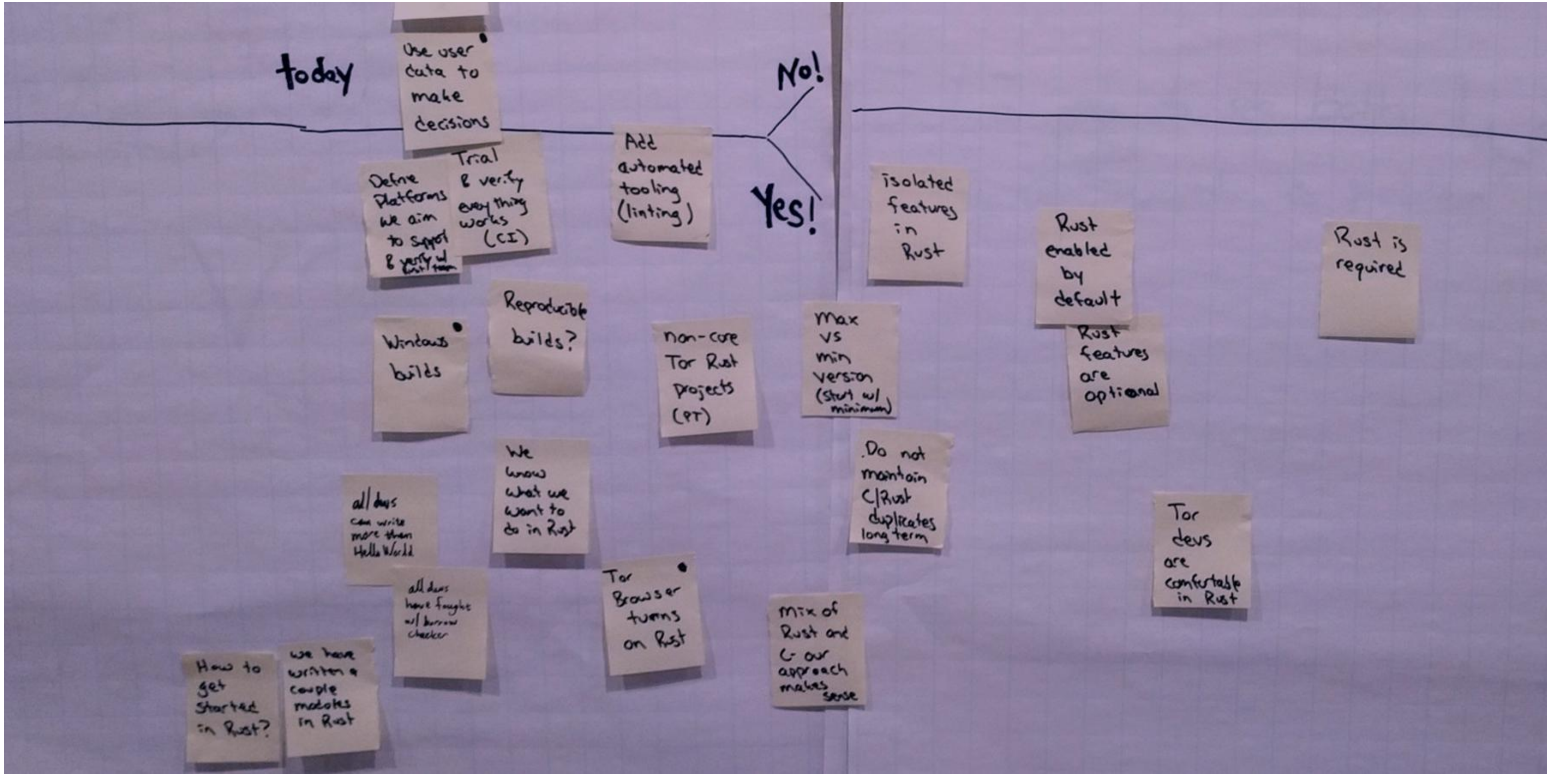
Deciding on Rust

We identified (some of) the following goals:

- Do no harm (the code that we deploy should not be a liability to the user)
- Have confidence in what is deployed
- Reduce size/memory requirements
- Developer friendliness
- Productivity
- Cross platform compatibility
- Not adding too much overhead
- Re-use existing test vectors
- Reproducibility

Source: https://trac.torproject.org/projects/tor/wiki/org/meetings/2017Amsterdam/Notes/MemorySafeLanguagesandTor

Identify and test requirements


today
No!
Yes!
Use user data to make decisions
Define Platforms we aim to support & verify w/ Rust team
Trial & verify every thing works (CI)
Add automated tooling (linting)
Windows builds
Reproducible builds?
non-core Tor Rust projects (PT)
all devs can write more than Hello World
We know what we want to do in Rust
all devs have fought w/ borrow checker
How to get Started in Rust?
we have written a couple modules in Rust
Tor Browser turns on Rust
Mix of Rust and C our approach makes sense
Isolated features in Rust
Max vs min version (start w/ minimum)
Do not maintain C/Rust duplicates long term
Rust enabled by default
Rust features are optional
Rust is required
Tor devs are comfortable in Rust

Source: https://blog.torproject.org/blog/network-team-hackfest-wilmington-watch

Critical questions

- How can we integrate Rust into the tor build system?

Critical questions

- How can we integrate Rust into the tor build system?
- What is the overhead to implement existing/new submodules in Rust?

Critical questions

- How can we integrate Rust into the tor build system?
- What is the overhead to implement existing/new submodules in Rust?
- Is Rust supported on platforms that Tor supports?

Critical questions

- How can we integrate Rust into the tor build system?
- What is the overhead to implement existing/new submodules in Rust?
- Is Rust supported on platforms that Tor supports?
- Can we reproducibly build tor with Rust enabled?

Overview

- What is Tor?
- Where we started with Rust
- **Where we are**
- Where we are going
- What have we learned
- What we hope to see

Experimental submodule rewrite

- Can we rewrite an existing submodule with little overhead/code changing?

Experimental submodule rewrite

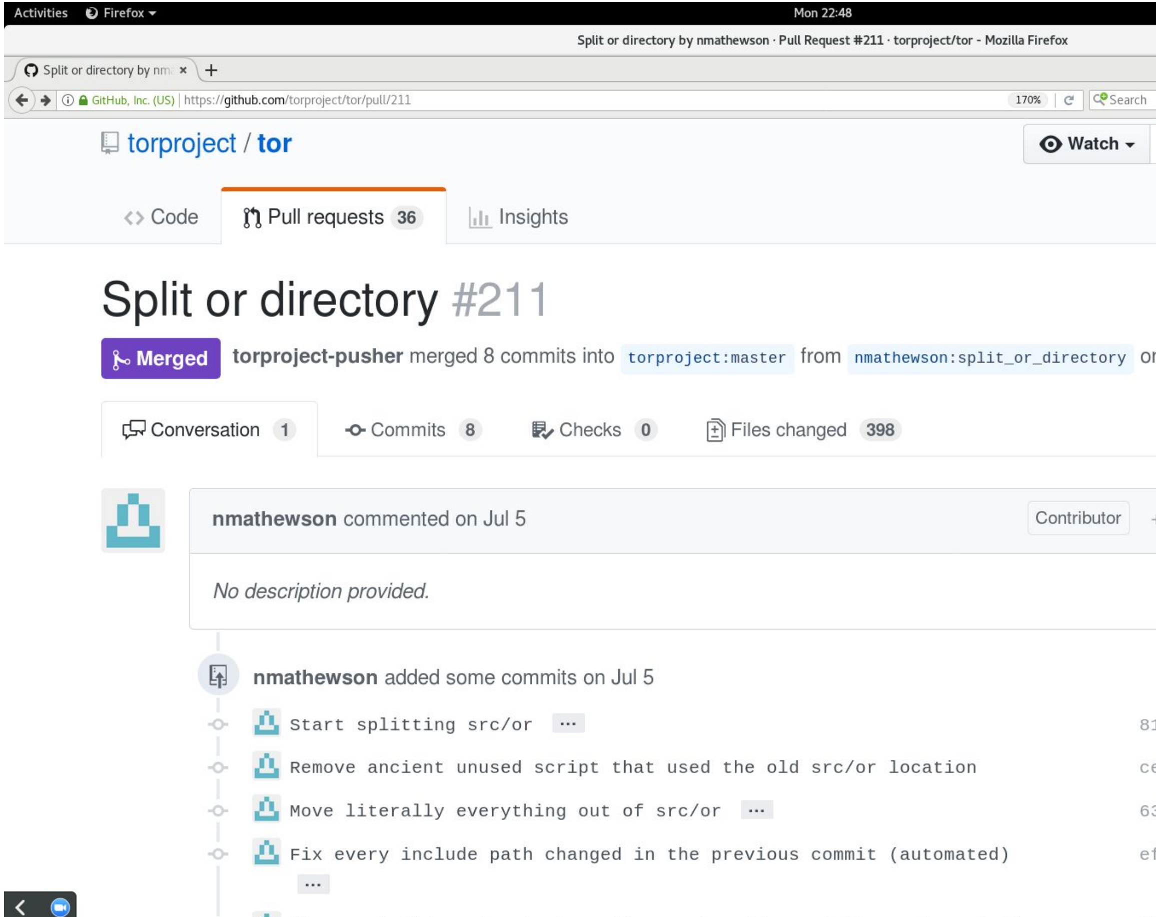
- Can we rewrite an existing submodule with little overhead/code changing?
- Choose one with limited dependencies and simple interface

Experimental submodule rewrite

- Can we rewrite an existing submodule with little overhead/code changing?
- Choose one with limited dependencies and simple interface
- Takeaway: Refactoring before porting would help make this easier in the future!

Modularization

- Improved modularization will help us move isolated functionality to Rust



Activities
Firefox
Mon 22:48
Split or directory by nmathewson · Pull Request #211 · torproject/tor - Mozilla Firefox
Split or directory by nm
GitHub, Inc. (US) | https://github.com/torproject/tor/pull/211
170%
Search
torproject / tor
Watch
Code
Pull requests 36
Insights

Split or directory #211

Merged torproject-pusher merged 8 commits into `torproject:master` from `nmathewson:split_or_directory` or

Conversation 1 | Commits 8 | Checks 0 | Files changed 398

nmathewson commented on Jul 5 — Contributor

No description provided.

nmathewson added some commits on Jul 5

```
Start splitting src/or ...
Remove ancient unused script that used the old src/or location
Move literally everything out of src/or ...
Fix every include path changed in the previous commit (automated)
...
```

Sorting out linking issues

Sorting out linking issues

- Building C code and Rust code as static libraries using the same sanitiser (e.g. UBSan, ASan) doesn’t currently have a configurable way to pass the same sanitiser options to the linker. This causes problems for unittest code where Rust code calls C code.

Sorting out linking issues

- Building C code and Rust code as static libraries using the same sanitiser (e.g. UBSan, ASan) doesn’t currently have a configurable way to pass the same sanitiser options to the linker. This causes problems for unittest code where Rust code calls C code.
- For doctests, we similarly need a way to pass arguments to the C linker (when Rust code in a doctest calls C).

Sorting out linking issues

- Building C code and Rust code as static libraries using the same sanitiser (e.g. UBSan, ASan) doesn't currently have a configurable way to pass the same sanitiser options to the linker. This causes problems for unittest code where Rust code calls C code.
- For doctests, we similarly need a way to pass arguments to the C linker (when Rust code in a doctest calls C).
- This has resulted in stubbing out `#[cfg(test)]` versions of Rust code which wraps C code. For example, we wrapped the usage of OpenSSL's (P)RNG and hash digests with code to implement the **`rand::Rng`** and **`digest::Digest`** traits, but we then had to substitute pure-Rust implementations during testing to avoid the linker errors.

Maintaining bitwise- and behaviourally- identical binary parsers in both C and Rust

Maintaining bitwise- and behaviourally- identical binary parsers in both C and Rust

- Turns out to be a bad idea!

Maintaining bitwise- and behaviourally- identical binary parsers in both C and Rust

- Turns out to be a bad idea! A *very, very* bad idea.
- So. bad. Do not do.

Maintaining bitwise- and behaviourally- identical binary parsers in both C and Rust

- Turns out to be a bad idea! A *very, very* bad idea.
- So. bad. Do not do.
- In maintaining bitwise- and behaviourally- identical binary parsers, we found bugs. In both implementations. Lots of bugs. Bugs that got CVE numbers assigned to them.

Maintaining bitwise- and behaviourally- identical binary parsers in both C and Rust

- Turns out to be a bad idea! A *very, very* bad idea.
- So. bad. Do not do.
- In maintaining bitwise- and behaviourally- identical binary parsers, we found bugs. In both implementations. Lots of bugs. Bugs that got CVE numbers assigned to them. 🐛

Maintaining bitwise- and behaviourally- identical binary parsers in both C and Rust

- Turns out to be a bad idea! A *very, very* bad idea.
- So. bad. Do not do.
- In maintaining bitwise- and behaviourally- identical binary parsers, we found bugs. In both implementations. Lots of bugs. Bugs that got CVE numbers assigned to them. 🐛🐞

Maintaining bitwise- and behaviourally- identical binary parsers in both C and Rust

- Turns out to be a bad idea! A *very, very* bad idea.
- So. bad. Do not do.
- In maintaining bitwise- and behaviourally- identical binary parsers, we found bugs. In both implementations. Lots of bugs. Bugs that got CVE numbers assigned to them. 🐛🐞🕷

Maintaining bitwise- and behaviourally- identical binary parsers in both C and Rust

- Turns out to be a bad idea! A *very, very* bad idea.
- So. bad. Do not do.
- In maintaining bitwise- and behaviourally- identical binary parsers, we found bugs. In both implementations. Lots of bugs. Bugs that got CVE numbers assigned to them. 🐛🐞🕷
- It's also a really good way to drive your developers up the wall as they manually fuzz for edge cases which produce different behaviours, which in the context of an anonymity system, could potentially be used as distinguishers between clients.

Maintaining bitwise- and behaviourally- identical binary parsers in both C and Rust

- Turns out to be a bad idea! A *very, very* bad idea.
- So. bad. Do not do.
- In maintaining bitwise- and behaviourally- identical binary parsers, we found bugs. In both implementations. Lots of bugs. Bugs that got CVE numbers assigned to them. 🐛🐞🕷
- It’s also a really good way to drive your developers up the wall as they manually fuzz for edge cases which produce different behaviours, which in the context of an anonymity system, could potentially be used as distinguishers between clients.
- We found a memory exhaustion attack, two different remote crashes due to null pointer deference, and a DoS attack by triggering an infinite loop.

Maintaining bitwise- and behaviourally- identical cryptographic protocol implementations

Maintaining bitwise- and behaviourally- identical cryptographic protocol implementations

- Not that bad of an idea, actually!

Maintaining bitwise- and behaviourally- identical cryptographic protocol implementations

- Not that bad of an idea, actually!
- We already had support for several different implementations of the ed25519 signature scheme, registering the external code via structs containing function pointers.

Maintaining bitwise- and behaviourally- identical cryptographic protocol implementations

- Not that bad of an idea, actually!
- We already had support for several different implementations of the ed25519 signature scheme, registering the external code via structs containing function pointers.
- All we had to do to integrate ed25519-dalek (and its underlying curve library, curve25519-dalek) was create FFI which presented the same interfaces as those defined in the function pointers.

Maintaining bitwise- and behaviourally- identical cryptographic protocol implementations

- Not that bad of an idea, actually!
- We already had support for several different implementations of the ed25519 signature scheme, registering the external code via structs containing function pointers.
- All we had to do to integrate ed25519-dalek (and its underlying curve library, curve25519-dalek) was create FFI which presented the same interfaces as those defined in the function pointers.
- This was a lot easier to implement and test, and should prove easier to maintain, largely due to the obvious requirement that cryptographic implementations match testvectors (and hopefully match in behaviour).

Privcount

Privcount

- Work done by our colleagues Tim “teor” Wilson-Brown and Nick Mathewson, and based upon work by our colleagues Rob Jansen and Aaron Johnson.

Privcount

- Work done by our colleagues Tim “teor” Wilson-Brown and Nick Mathewson, and based upon work by our colleagues Rob Jansen and Aaron Johnson.
- First stand-alone (i.e. not also implemented in C) pure-Rust module.

Privcount

- Work done by our colleagues Tim “teor” Wilson-Brown and Nick Mathewson, and based upon work by our colleagues Rob Jansen and Aaron Johnson.
- First stand-alone (i.e. not also implemented in C) pure-Rust module.
- In cryptography, differential privacy systems aim to provide means to maximize query accuracy from statistical databases while minimizing the chances of identifying specific records.

Privcount

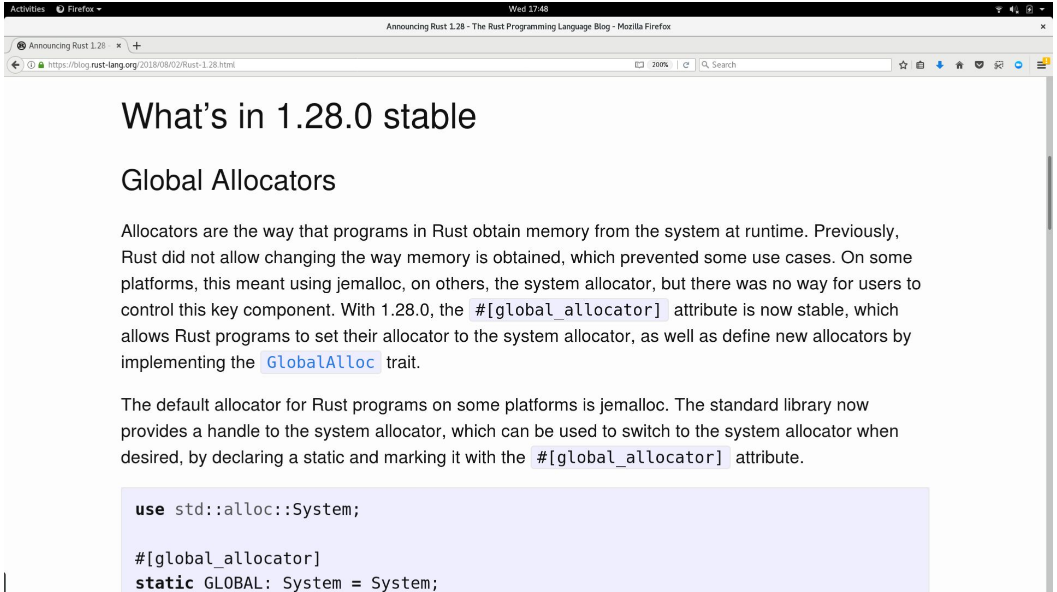
- Work done by our colleagues Tim “teor” Wilson-Brown and Nick Mathewson, and based upon work by our colleagues Rob Jansen and Aaron Johnson.
- First stand-alone (i.e. not also implemented in C) pure-Rust module.
- In cryptography, differential privacy systems aim to provide means to maximize query accuracy from statistical databases while minimizing the chances of identifying specific records.
- Privcount is aimed at safely gathering anonymised metrics on servers in a manner that is resistant to servers colluding to influence the results or learn things about the Privcount metrics gathered by other participants by combining Shamir secret sharing (for robustness) with the PrivEx algorithm developed by our colleagues Tariq Elahi, George Danezis, and Ian Goldberg.

Overview

- What is Tor?
- Where we started with Rust
- Where we are
- **Where we are going**
- What have we learned
- What we hope to see

Improved FFI Ergonomics

- Using the same allocator between Rust and C


Activities
Firefox
Wed 17:48
Announcing Rust 1.28 - The Rust Programming Language Blog - Mozilla Firefox
Announcing Rust 1.28
https://blog.rust-lang.org/2018/08/02/Rust-1.28.html
200%
Search

What’s in 1.28.0 stable

Global Allocators

Allocators are the way that programs in Rust obtain memory from the system at runtime. Previously, Rust did not allow changing the way memory is obtained, which prevented some use cases. On some platforms, this meant using jemalloc, on others, the system allocator, but there was no way for users to control this key component. With 1.28.0, the `#[global_allocator]` attribute is now stable, which allows Rust programs to set their allocator to the system allocator, as well as define new allocators by implementing the `GlobalAlloc` trait.

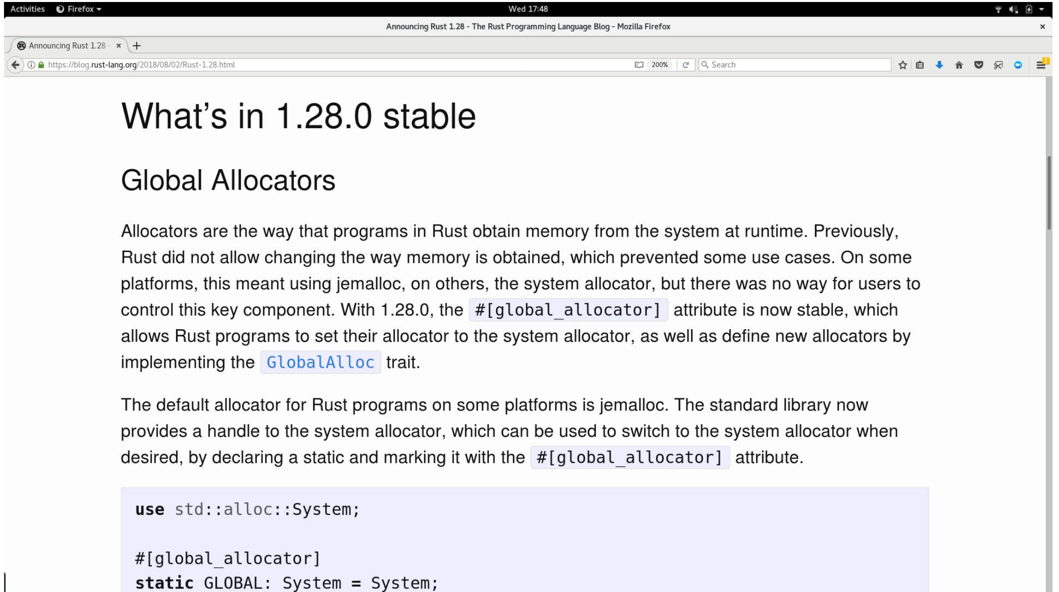
The default allocator for Rust programs on some platforms is jemalloc. The standard library now provides a handle to the system allocator, which can be used to switch to the system allocator when desired, by declaring a static and marking it with the `#[global_allocator]` attribute.

```
use std::alloc::System;

#[global_allocator]
static GLOBAL: System = System;
```

Improved FFI Ergonomics

- Using the same allocator between Rust and C
- Automated mechanism to keep C/Rust shared types in sync


Activities
Firefox
Wed 17:48
Announcing Rust 1.28 - The Rust Programming Language Blog - Mozilla Firefox
Announcing Rust 1.28 -
https://blog.rust-lang.org/2018/08/02/Rust-1.28.html
200%
Search

What's in 1.28.0 stable

Global Allocators

Allocators are the way that programs in Rust obtain memory from the system at runtime. Previously, Rust did not allow changing the way memory is obtained, which prevented some use cases. On some platforms, this meant using jemalloc, on others, the system allocator, but there was no way for users to control this key component. With 1.28.0, the `#[global_allocator]` attribute is now stable, which allows Rust programs to set their allocator to the system allocator, as well as define new allocators by implementing the `GlobalAlloc` trait.

The default allocator for Rust programs on some platforms is jemalloc. The standard library now provides a handle to the system allocator, which can be used to switch to the system allocator when desired, by declaring a static and marking it with the `#[global_allocator]` attribute.

```
use std::alloc::System;

#[global_allocator]
static GLOBAL: System = System;
```

Overview

- What is Tor?
- Where we started with Rust
- Where we are
- Where we are going
- **What have we learned**
- What we hope to see

Complex objects across FFI is hard!

Complex objects across FFI is hard!

- We started manually & used comments to indicate where types are kept in sync

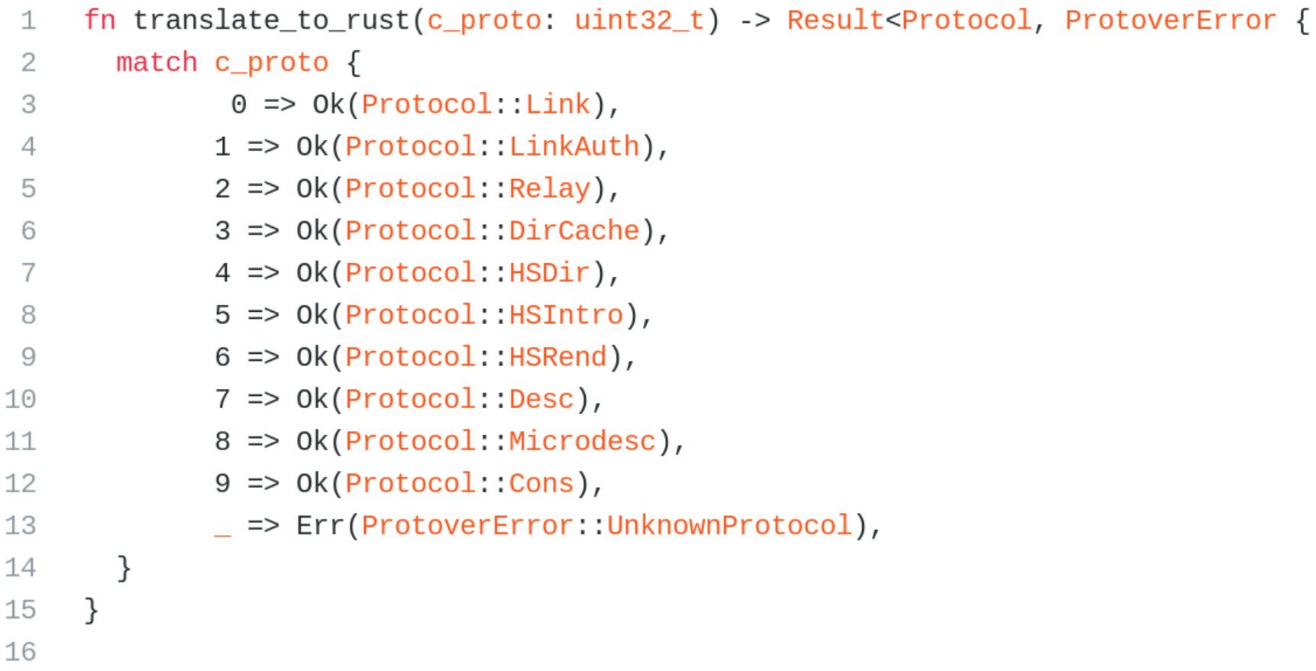
Complex objects across FFI is hard!

- We started manually & used comments to indicate where types are kept in sync
- Duplication across language boundaries (enums, constants, etc)

Complex objects across FFI is hard!

- We started manually & used comments to indicate where types are kept in sync
- Duplication across language boundaries (enums, constants, etc)
- Extra copies when converting types (Tor’s smartlist to a Rust vector)

Translating enums



```
fn translate_to_rust(c_proto: uint32_t) -> Result<Protocol, ProtoverError {
  match c_proto {
         0 => Ok(Protocol::Link),
        1 => Ok(Protocol::LinkAuth),
        2 => Ok(Protocol::Relay),
        3 => Ok(Protocol::DirCache),
        4 => Ok(Protocol::HSDir),
        5 => Ok(Protocol::HSIntro),
        6 => Ok(Protocol::HSRend),
        7 => Ok(Protocol::Desc),
        8 => Ok(Protocol::Microdesc),
        9 => Ok(Protocol::Cons),
        _ => Err(ProtoverError::UnknownProtocol),
  }
}
```

Keeping Rust API distinct from FFI

Keeping Rust API distinct from FFI

- At first, we wrote a lot of very suspiciously C-looking Rust code.

Keeping Rust API distinct from FFI

- At first, we wrote a lot of very suspiciously C-looking Rust code.
- Later, we defined coding standards which said that the pure-Rust implementations needed to present intuitive, easy-to-use, rusty APIs, and that code for FFI and interfacing with C needed to live elsewhere and do the work of safely translating between the C API and the Rust API.

Keeping Rust API distinct from FFI

- At first, we wrote a lot of very suspiciously C-looking Rust code.
- Later, we defined coding standards which said that the pure-Rust implementations needed to present intuitive, easy-to-use, rusty APIs, and that code for FFI and interfacing with C needed to live elsewhere and do the work of safely translating between the C API and the Rust API.
- This should work better moving forward, when we have more Rust code calling more Rust code, but seems to be slightly confusing to new contributors in the meantime.

Keeping Rust API distinct from FFI

- At first, we wrote a lot of very suspiciously C-looking Rust code.
- Later, we defined coding standards which said that the pure-Rust implementations needed to present intuitive, easy-to-use, rusty APIs, and that code for FFI and interfacing with C needed to live elsewhere and do the work of safely translating between the C API and the Rust API.
- This should work better moving forward, when we have more Rust code calling more Rust code, but seems to be slightly confusing to new contributors in the meantime.
- Our Rust coding standards also require that Rust-to-C FFI be kept separate to C-to-Rust FFI, and that Rust-to-C FFI all live in one crate (to avoid code duplication).

Write new features in Rust while RIRing

Write new features in Rust while RIRing

- Based on our experiences, if we had to do this again, we'd opt for writing new features in Rust while rewriting old code in Rust. (By rewriting, we mean replacing, i.e. *not* maintaining two implementations.)

Write new features in Rust while RIRing

- Based on our experiences, if we had to do this again, we'd opt for writing new features in Rust while rewriting old code in Rust. (By rewriting, we mean replacing, i.e. *not* maintaining two implementations.)
- More specifically, maintaining two bitwise- and behaviourally- identical binary parsers was bad and led to badness and sadness. Developing new features in Rust *only*, as well as rewriting things was awesome. (Toss that technical debt out the window, for ferris' sake!!)

Keep a running code standards guide

Keep a running code standards guide

- We found it important to document our standards and guidelines for Rust contributions from the outset, and point to them often.

Keep a running code standards guide

- We found it important to document our standards and guidelines for Rust contributions from the outset, and point to them often.
- Give developers clear explanations of what to do *and* what to not do, with code snippets and a concise reasoning for why we've made these choices.

Keep a running code standards guide: safety

- Educate developers on UB in Rust, avoiding unwinding across the FFI boundary, maintaining type safety, avoiding `unsafe` and `unwrap()`, whitelisting only C ABI compatible types for crossing the FFI boundary, no abusing `unsafe` to muck around with changing lifetimes, avoiding memory leaks in `CString` usage, performing allocations to copy buffers across the FFI boundary, enums are nobody's friend, downcasting floats to unsigned integers with `as`, and many more no-nos.

Keep a running code standards guide

The current guides can be found at:

https://github.com/torproject/tor/blob/master/doc/HACKING/GettingStartedRust.md

https://github.com/torproject/tor/blob/master/doc/HACKING/CodingStandardsRust.md

Overview

- What is Tor?
- Where we started with Rust
- Where we are
- Where we are going
- What have we learned
- **What we hope to see from Rust**

Fuzzing Assertions/test harness

- It’d be great to have an easy way to use `cargo-fuzz` or a similar project to take fuzzing inputs from afl or libsys-fuzzer and hand the same input to both a C and a Rust function, then test that the output matches.

Better FFI documentation

Better FFI documentation

- Writing pure-Rust is different than writing Rust that will be called across an FFI- important to document.

Better FFI documentation

- Writing pure-Rust is different than writing Rust that will be called across an FFI- important to document.
- Usage of bindgen could be better documented for newcomers.

Thank you!

isis agora lovecruft: cryptographic design and implementations, security engineering

https://twitter.com/isislovecruft
https://github.com/isislovecruft

Chelsea Holland Komlo: Distributed systems, applied cryptography research and implementation

https://twitter.com/chelseakomlo
https://github.com/chelseakomlo