MapGuard dereferences to a dangling pointer #45

Closed **Qwaz** opened this issue on Dec 9, 2020 · 5 comments

Qwaz commented on Dec 9, 2020

Hello fellow Rustacean,
we (Rust group **@sslab-gatech**) are scanning Rust code on crates.io for potential memory safety and soundness bugs and found an issue in this crate which allows safe Rust code to exhibit an undefined behavior.

Issue Description

arc-swap/src/access.rs
Lines 279 to 293 in b5ec44c

```rust
impl<A, T, F, R> Access<R> for Map<A, T, F>
where
    A: Access<T>,
    F: Fn(&T) -> &R,
{
    type Guard = MapGuard<A::Guard, R>;
    fn load(&self) -> Self::Guard {
        let guard = self.access.load();
        let value: *const _ = (self.projection)(&guard);
        MapGuard {
            _guard: guard,
            value,
```

arc-swap/src/access.rs
Lines 230 to 242 in b5ec44c

```rust
impl<G, T> Deref for MapGuard<G, T> {
    type Target = T;
    fn deref(&self) -> &T {
        // Why this is safe:
        // * The pointer is originally converted from a reference. It's not null, it's aligned,
        //   it's the right type, etc.
        // * The pointee couldn't have gone away - the guard keeps the original reference alive, so
        //   must the new one still be alive too. Moving the guard is fine, we assume the RefCnt is
        //   Pin (because it's Arc or Rc or something like that - when that one moves, the data it
        //   points to stay at the same place).
        unsafe { &*self.value }
    }
```

As noted in the comment, unsafe code in `MapGuard` expects the underlying guard type to dereferences to the same value even when the guard object is moved. However, `Map` uses `Access` as a trait bound which does not guarantee this property. As a result, `Map` accesses a dangling pointer when it is used with an `Access` implementation that does not dereferences to the same value when moved.

arc-swap/src/access.rs
Lines 295 to 322 in b5ec44c

```rust
#[doc(hidden)]
#[derive(Copy, Clone, Debug, Eq, PartialEq, Ord, PartialOrd, Hash)]
pub struct ConstantDeref<T>(T);

impl<T> Deref for ConstantDeref<T> {
    type Target = T;
    fn deref(&self) -> &T {
        &self.0
    }
}

/// Access to an constant.
```

`Constant` seems to be the only type in this crate that implements `Access` in this way, but there can be other user types that implements `Access` on their own.

Reproduction

Below is an example program that segfaults, written only with safe APIs of `arc-swap`.

▶ Show Detail

vorner commented on Dec 10, 2020 Owner

Hello

It's a great discovery, though disturbing this has slipped through. I'll have to think how to plug this problem in a way that doesn't disrupt users too much, but I'll have a look at it today or tomorrow.

Thank you for finding it.

vorner added a commit that referenced this issue on Dec 10, 2020

`Fix soundness hole around access::Map` … baff7e8

vorner closed this as completed in `dfeb84b` on Dec 11, 2020

vorner commented on Dec 11, 2020 Owner

Fix released as 1.1.0 (by eliminating all unsafe code in that file).

It is technically a breaking change, but the chance of actually breaking code is low and the chance people will migrate from the broken version are higher this way, and even rustc makes an exception for soundness issues in the stability guarantees. It seems less bad to release as part of the 1. version.

dbanty mentioned this issue on Dec 11, 2020

Update arc-swap XAMPPRocky/octocrab#48

Closed

vorner added a commit that referenced this issue on Dec 11, 2020

`Fix soundness hole around access::Map` ··· ✕ `34b809f`

vorner commented on Dec 11, 2020 Owner

Added a backport, released as `0.4.8`, for all the reverse dependencies that didn't migrate to the `1.` version yet.

Qwaz commented on Dec 12, 2020 Author

Thank you for the quick fix!

abergmann commented on Dec 28, 2020

CVE-2020-35711 was assigned to this issue.

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

3 participants