Firecracker

Lessons from the Trenches

Agenda

- Firecracker Design
- Correction of Errors
- Two inglorious bugs

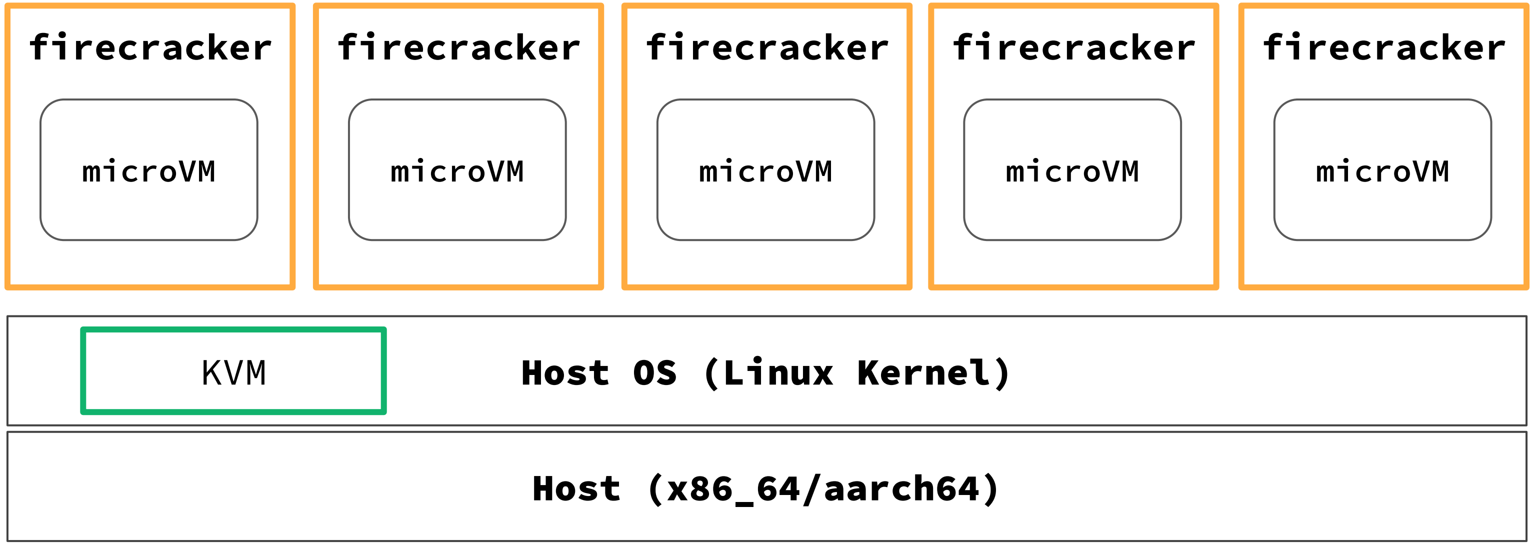
What is Firecracker?

What is Firecracker?

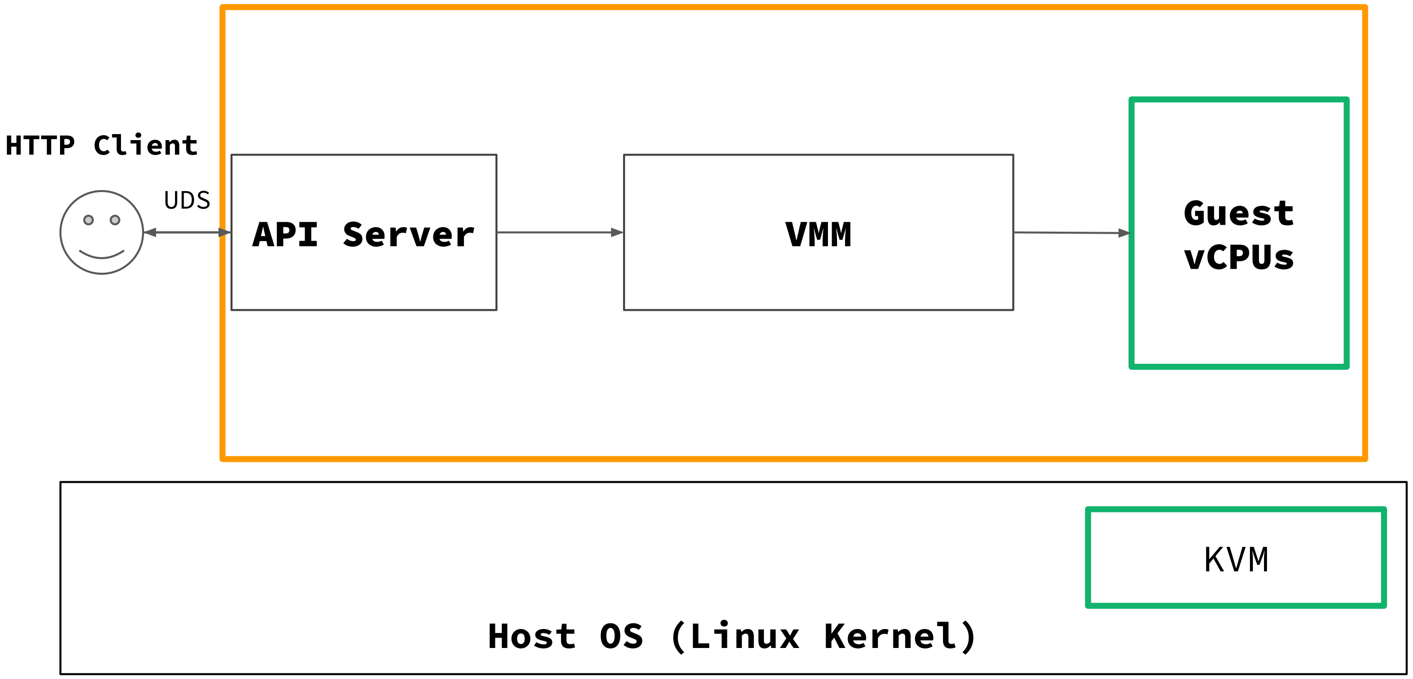
- Lightweight VMM written in Rust
- Multi-tenant cloud workloads (containers/functions)
- Used in production by AWS Lambda
- Open Source

https://github.com/firecracker-microvm/firecracker/

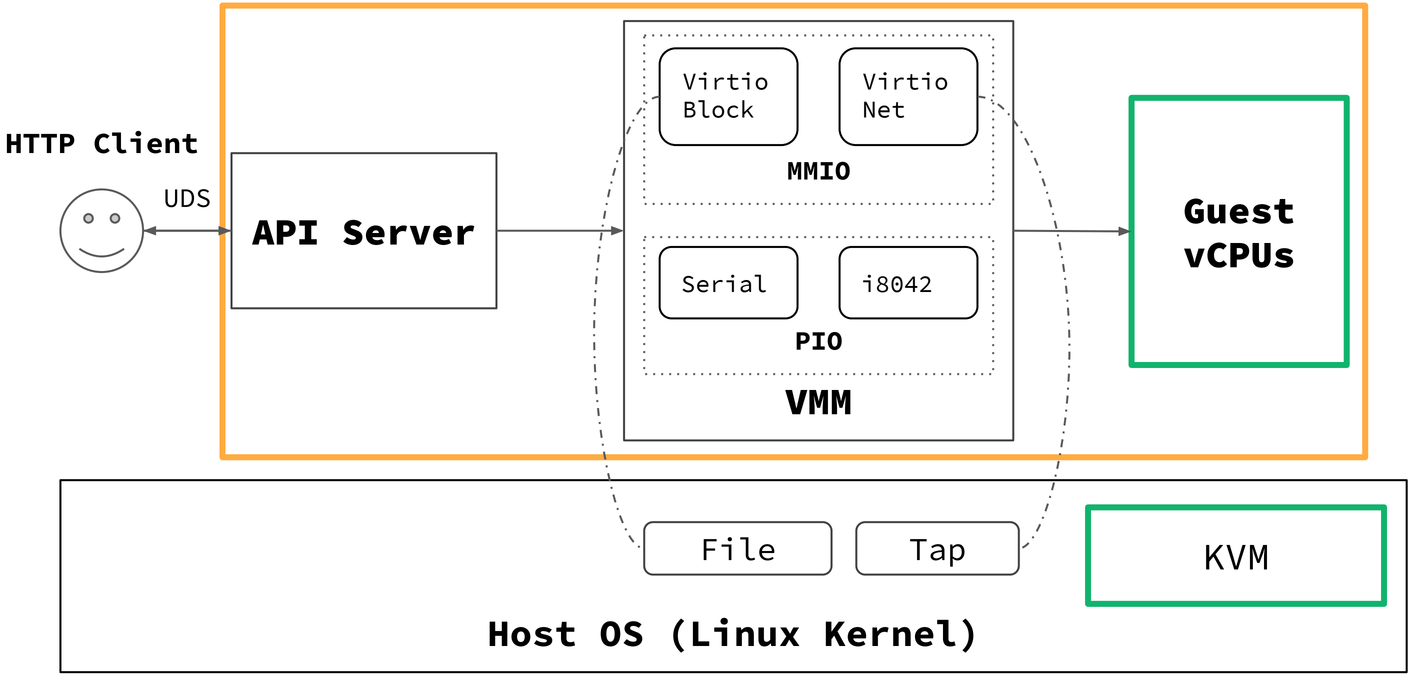
Firecracker Design



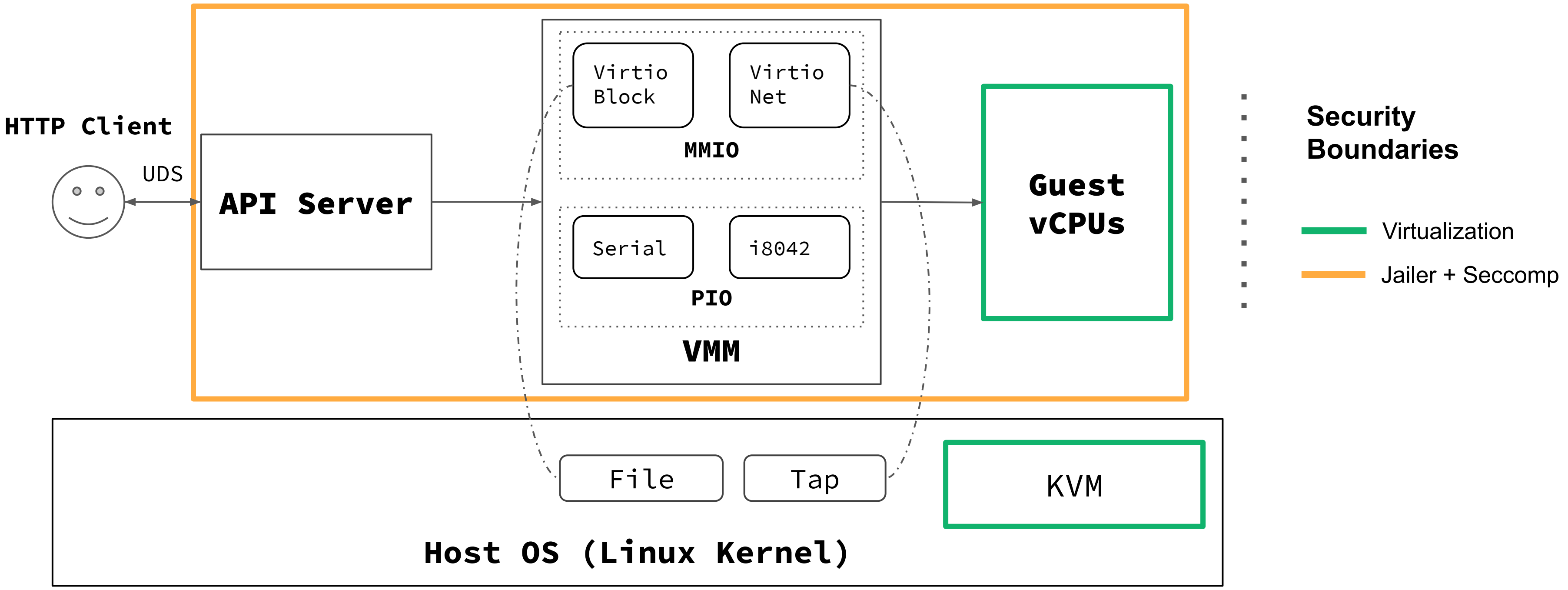
Firecracker Design - Threads



Firecracker Design - Devices



Firecracker Design - Security



Firecracker Properties

- Boot Time ~125ms*
- Low memory overhead ~3 MiB*
- Oversubscription CPU & Memory

* workload & configuration dependent; check out https://github.com/firecracker-microvm/firecracker/blob/master/SPECIFICATION.md

“You are destined to fail.”

L. David Marquet, *“Turn the Ship Around”*

COE

- Correction of Errors
- Understand the root cause
- 5 whys
- Take corrective actions & prevent same kind of mistakes

https://wa.aws.amazon.com/wat.concept.coe.en.html

Inglorious … Release

MADvise?

Problem:

- Firecracker intermittently exits with error code 128
- SYS_MADVISE is not whitelisted

Impact: Customers are unable to update Firecracker

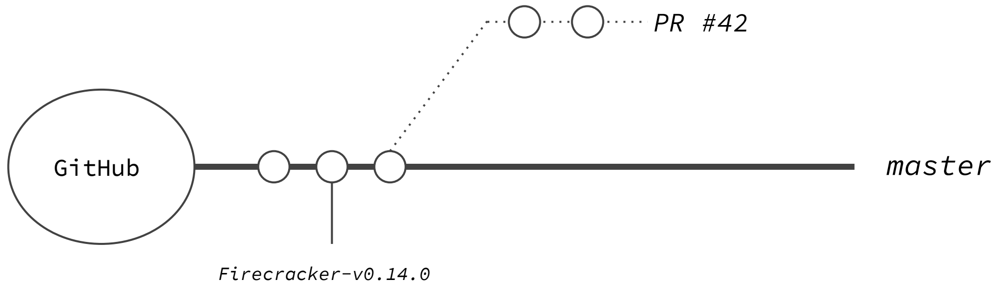
Fix: Whitelist SYS_MADVISE

Affected Versions: v0.15.0, v0.15.1

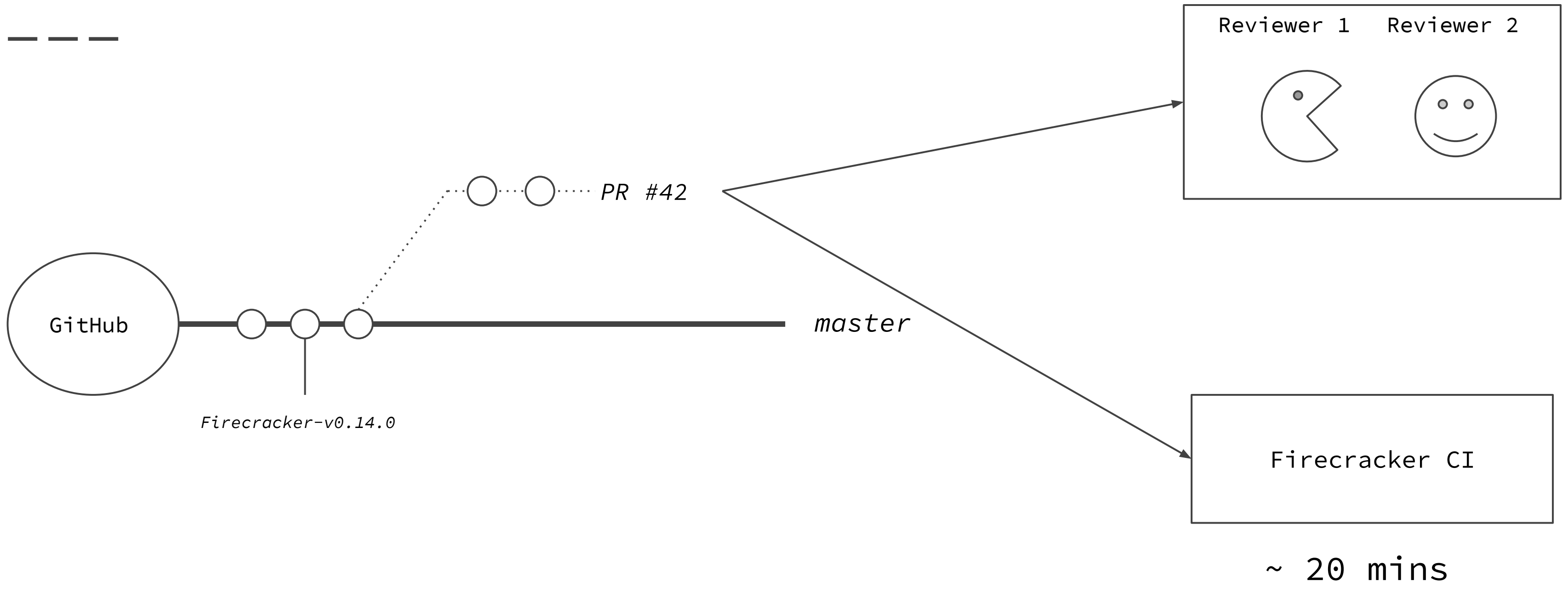
Firecracker - Seccomp

- Seccomp Filters:
  - None
  - Basic
  - Advanced (Default)
- Whitelist Approach
- ~30 whitelisted syscalls
- Seccomp Action:
  - Trap -> update metrics, log errors, exit with error code

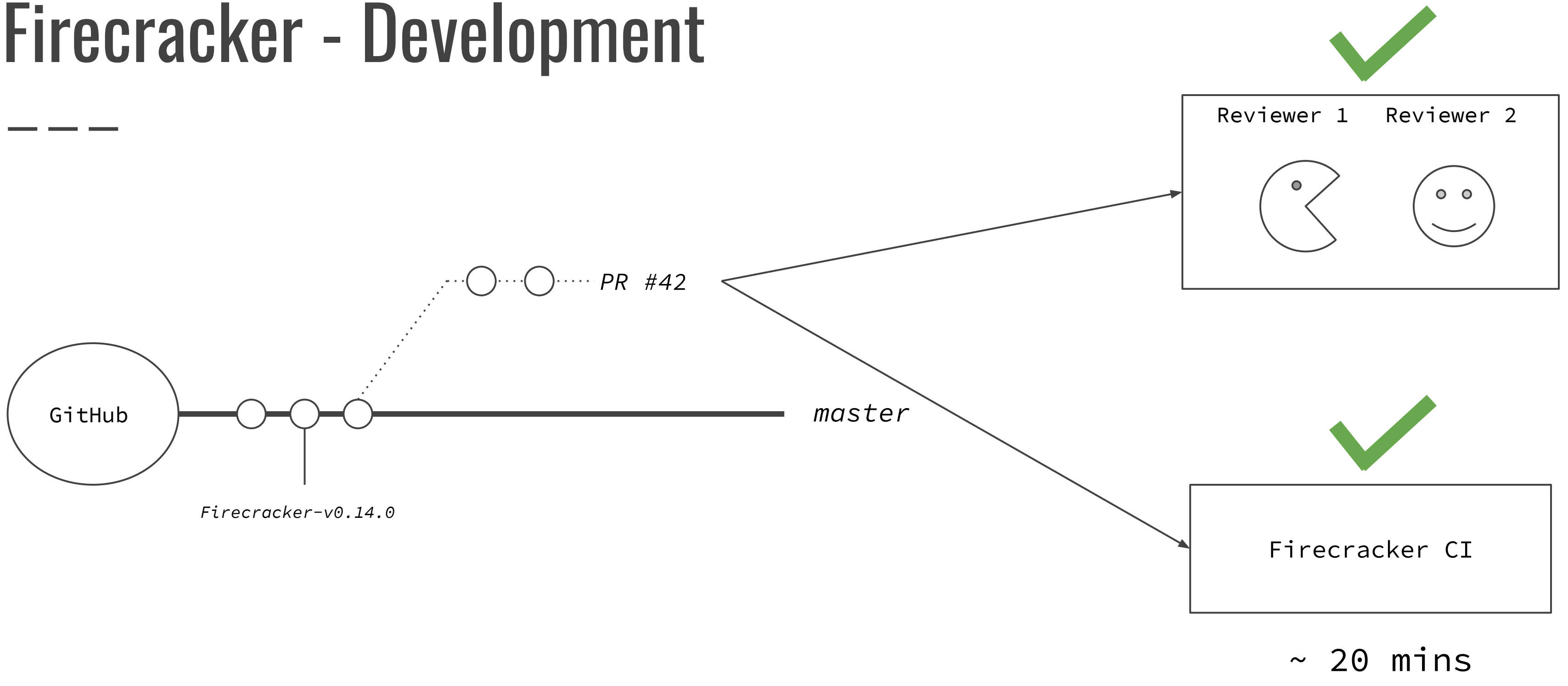
Firecracker - Development



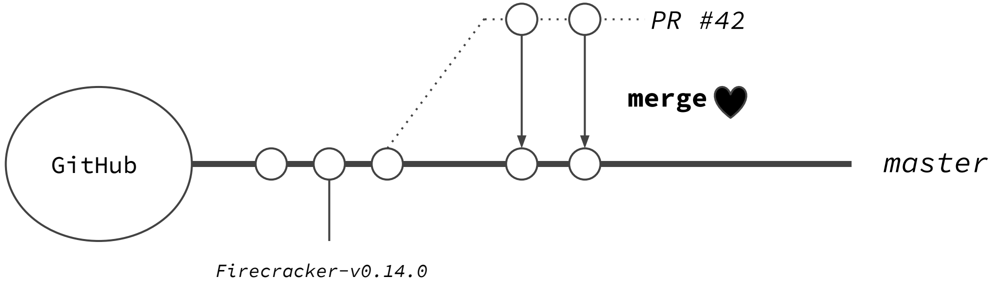
Firecracker - Development



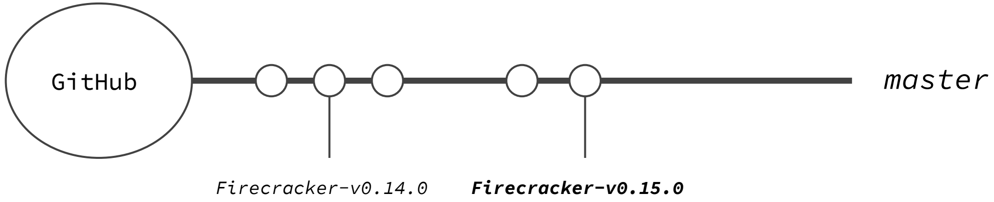
Firecracker - Development



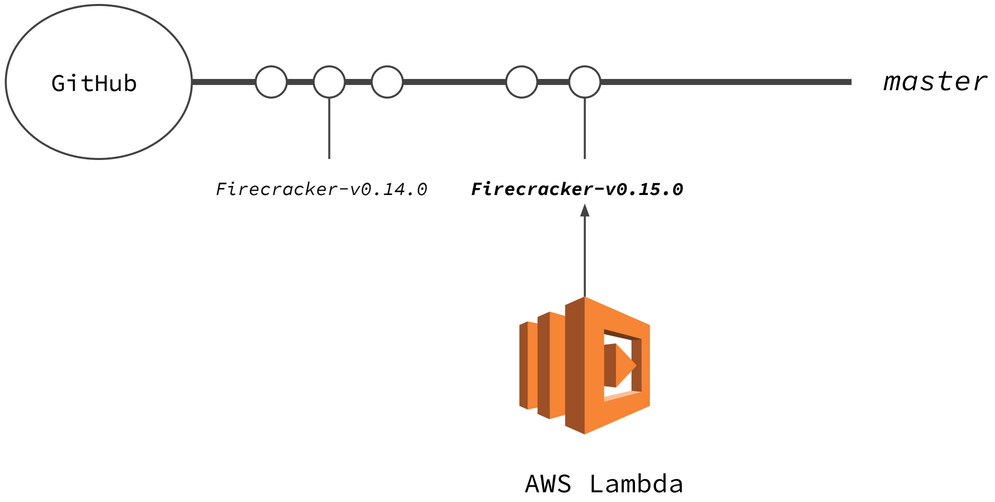
Firecracker - Development



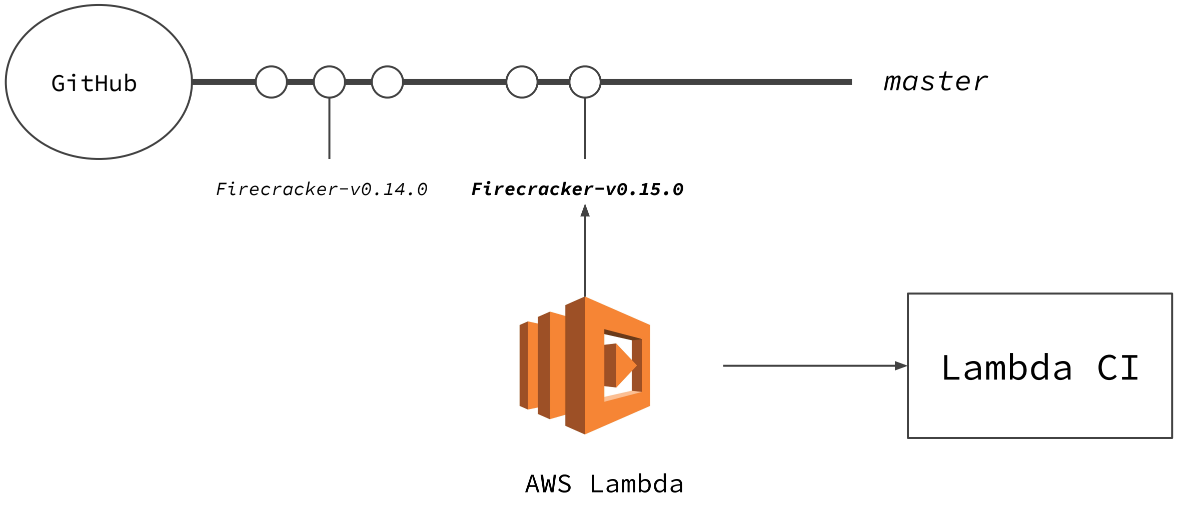
Firecracker - Development



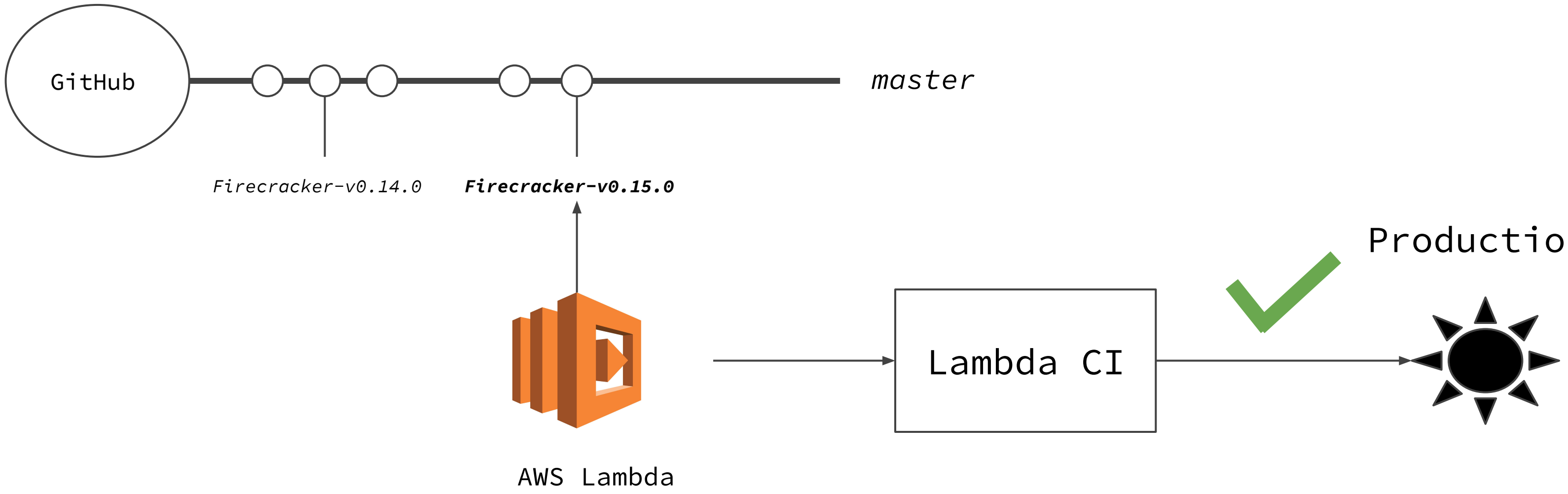
Firecracker - Development



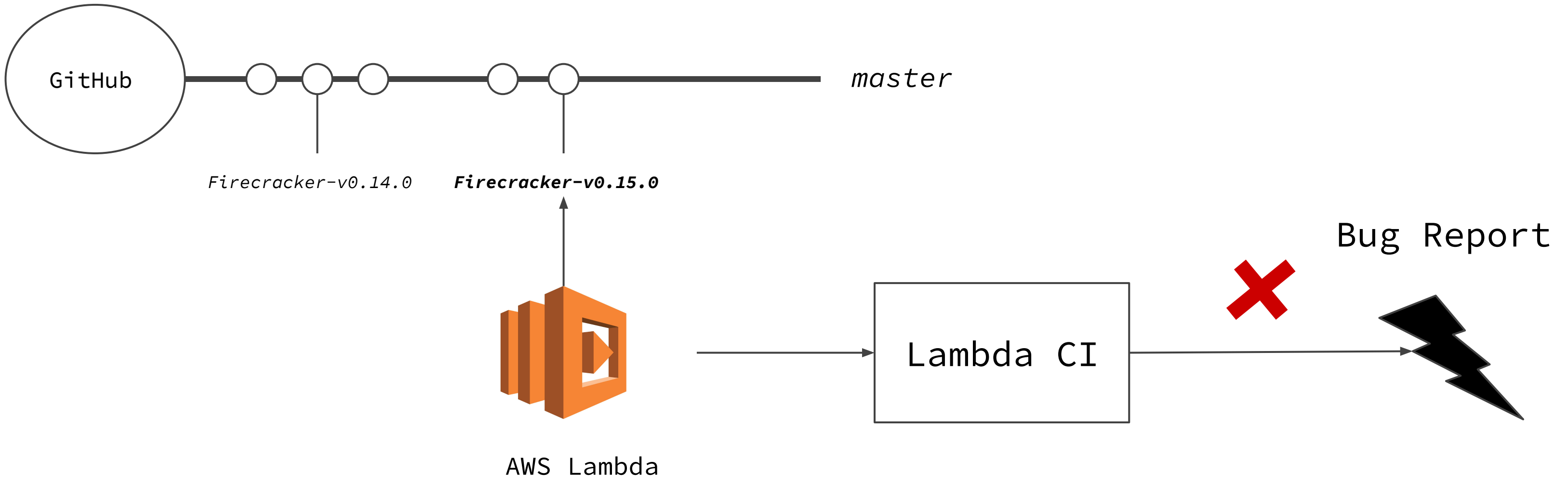
Firecracker - Development



Firecracker - Development



Firecracker - Development

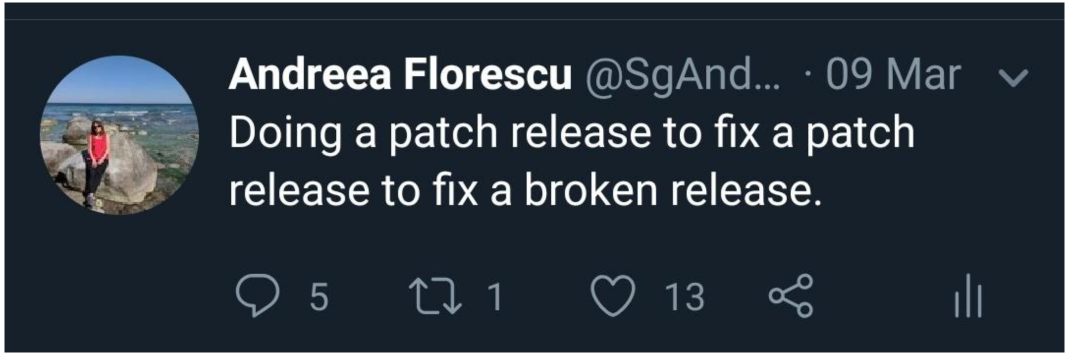


The Whys

- Why wasn’t SYS_madvise whitelisted?
    - V0.15.0 included an update to Rust 1.32
    - Changed out of memory handling in Rust runtime
- Why didn’t we catch it in Firecracker CI?
    - Compromise between CI time & coverage
    - Syscall triggered by specific workloads

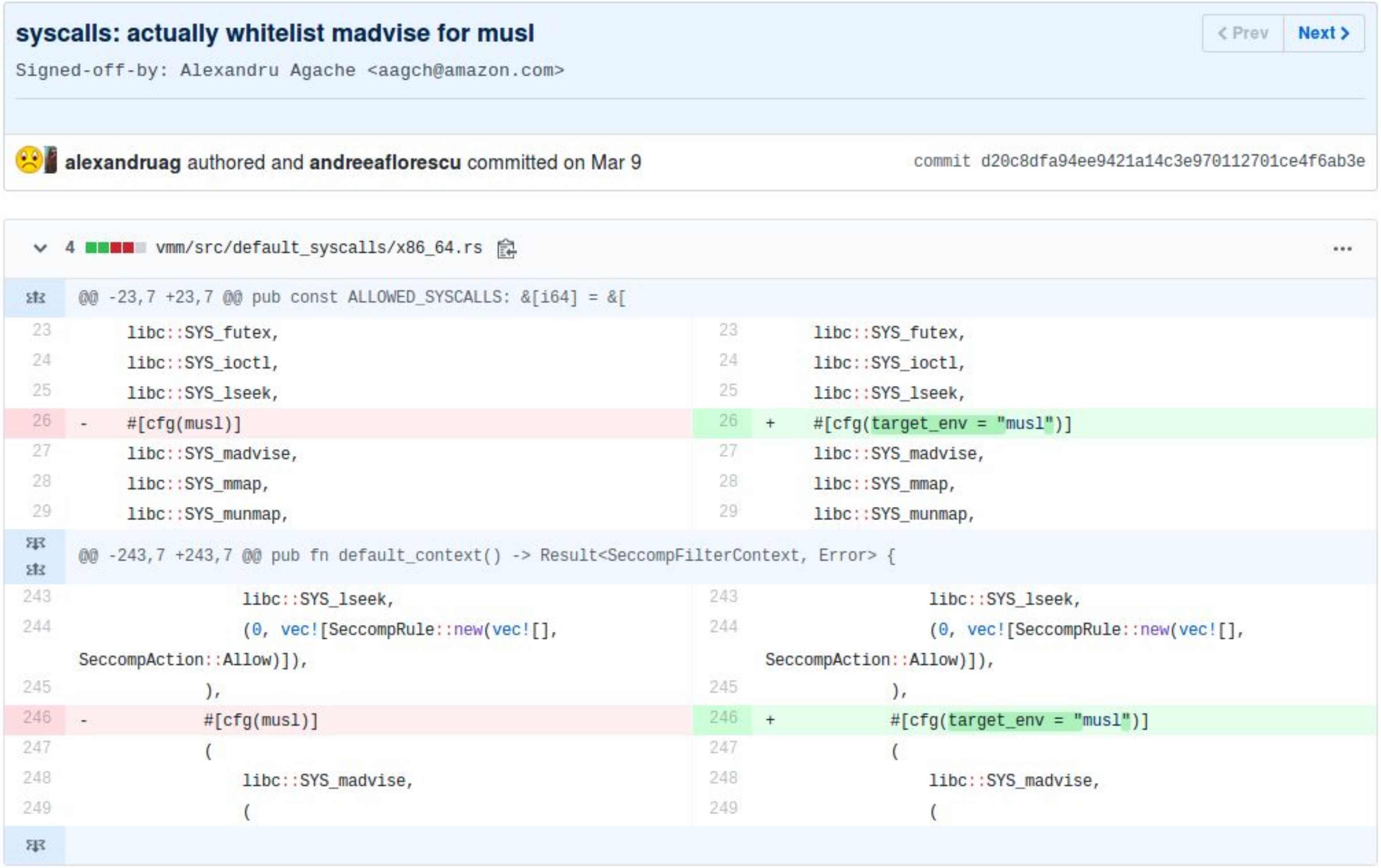
All good, right?

Wrong....


Andreea Florescu @SgAnd... · 09 Mar

Doing a patch release to fix a patch release to fix a broken release.

5 1 13



syscalls: actually whitelist madvise for musl

< Prev Next >

```
Signed-off-by: Alexandru Agache <aagch@amazon.com>
```

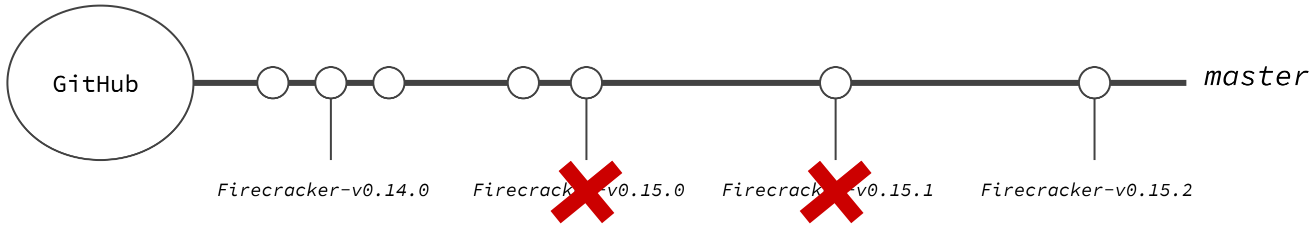
alexandruag authored and **andreeaflorescu** committed on Mar 9

`commit d20c8dfa94ee9421a14c3e970112701ce4f6ab3e`

4 🟩🟩🟥🟥 `vmm/src/default_syscalls/x86_64.rs`

```
@@ -23,7 +23,7 @@ pub const ALLOWED_SYSCALLS: &[i64] = &[
     libc::SYS_futex,                          libc::SYS_futex,
     libc::SYS_ioctl,                          libc::SYS_ioctl,
     libc::SYS_lseek,                          libc::SYS_lseek,
-    #[cfg(musl)]                           +  #[cfg(target_env = "musl")]
     libc::SYS_madvise,                        libc::SYS_madvise,
     libc::SYS_mmap,                           libc::SYS_mmap,
     libc::SYS_munmap,                         libc::SYS_munmap,
@@ -243,7 +243,7 @@ pub fn default_context() -> Result<SeccompFilterContext, Error> {
             libc::SYS_lseek,                          libc::SYS_lseek,
             (0, vec![SeccompRule::new(vec![],         (0, vec![SeccompRule::new(vec![],
SeccompAction::Allow)]),                  SeccompAction::Allow)]),
         ),                                        ),
-        #[cfg(musl)]                           +  #[cfg(target_env = "musl")]
         (                                         (
             libc::SYS_madvise,                        libc::SYS_madvise,
             (                                         (
```

Inglorious Releases



Corrective Actions

- Add long running tests
- Improve seccomp
  - Whitelist vs Blacklist
  - Auto-generate seccomp whitelist? -> 60 syscalls
  - Still discussing: https://github.com/firecracker-microvm/firecracker/issues/1177

Lessons Learned

- Testing, testing, testing!
- Use workloads as close as possible to production
- Logs and metrics saved the day (and engineering time)

“Math is hard.”

Everyone

Pesky Plus Sign

“+” vs “overflowing_add()”

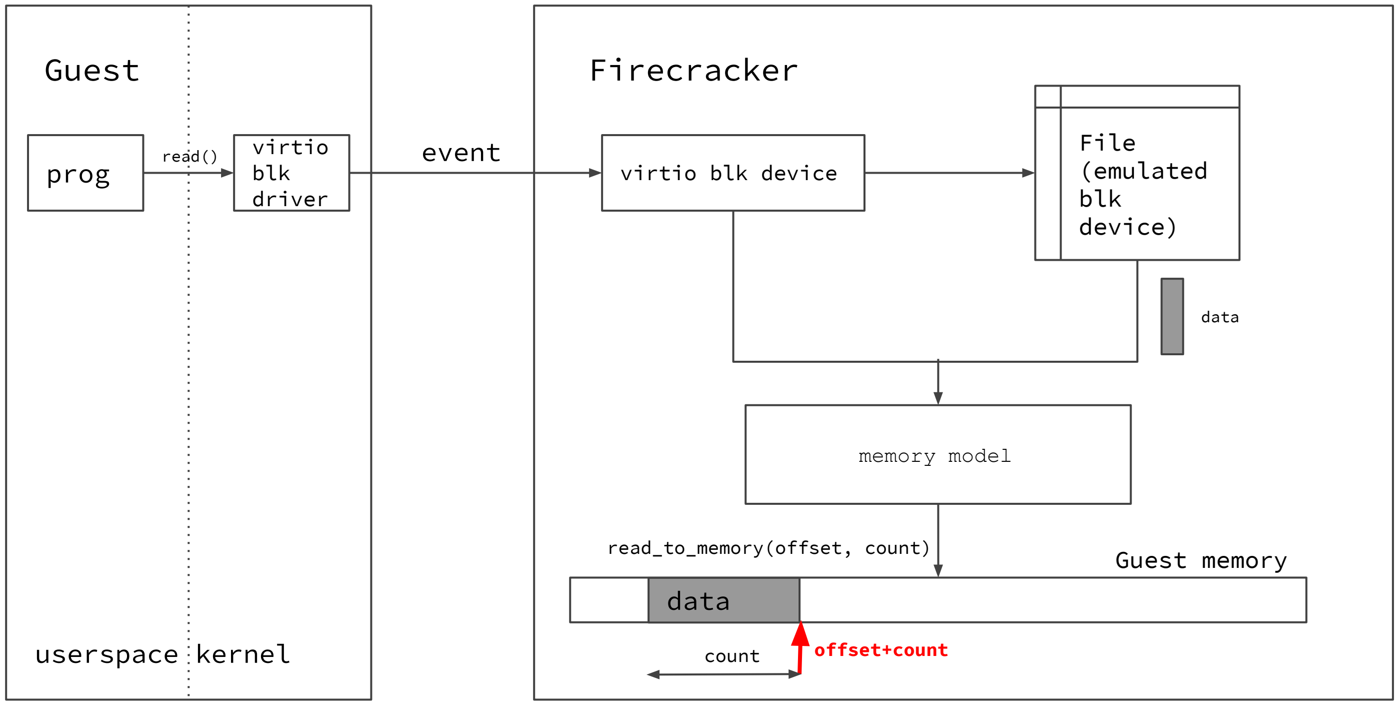
Problem: Unchecked arithmetic in memory model code

Potential Impact: Abrupt termination of guest OS

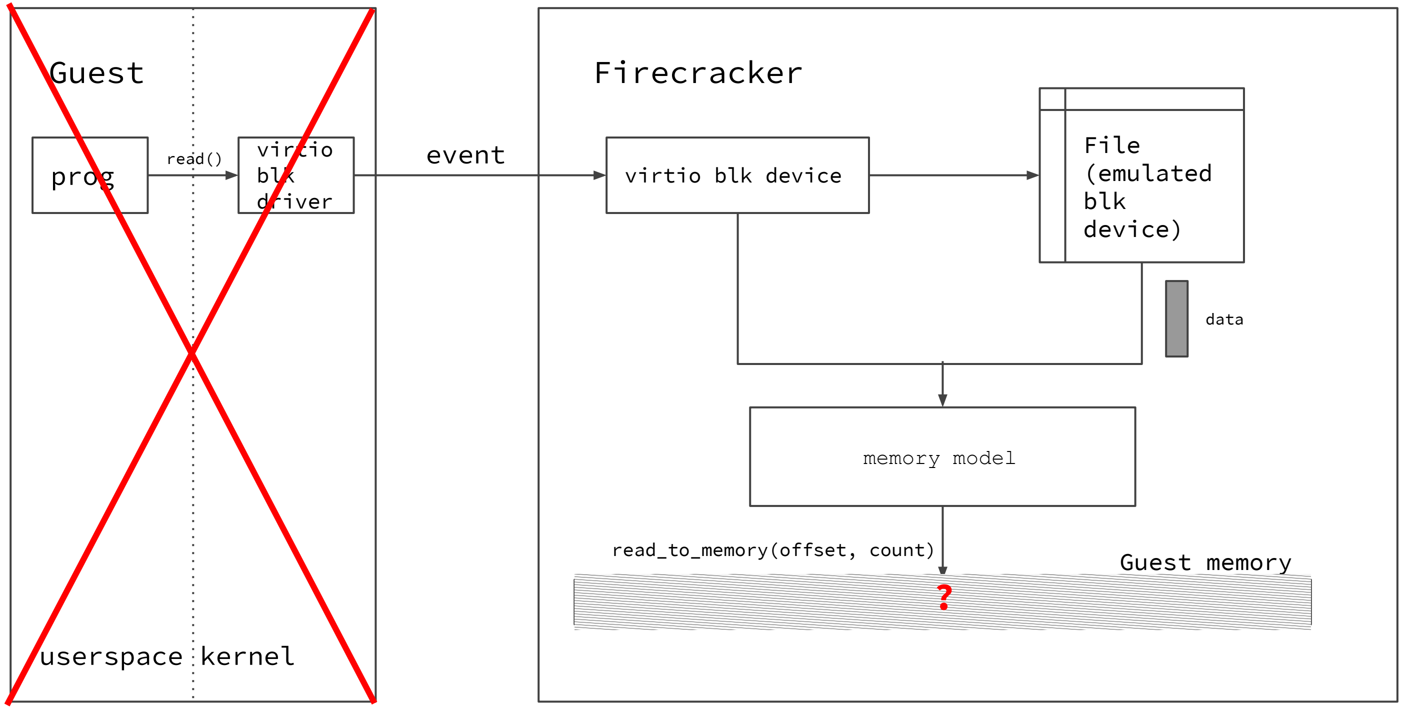
Fix: Checked arithmetic

Affected Versions: < v0.12.0

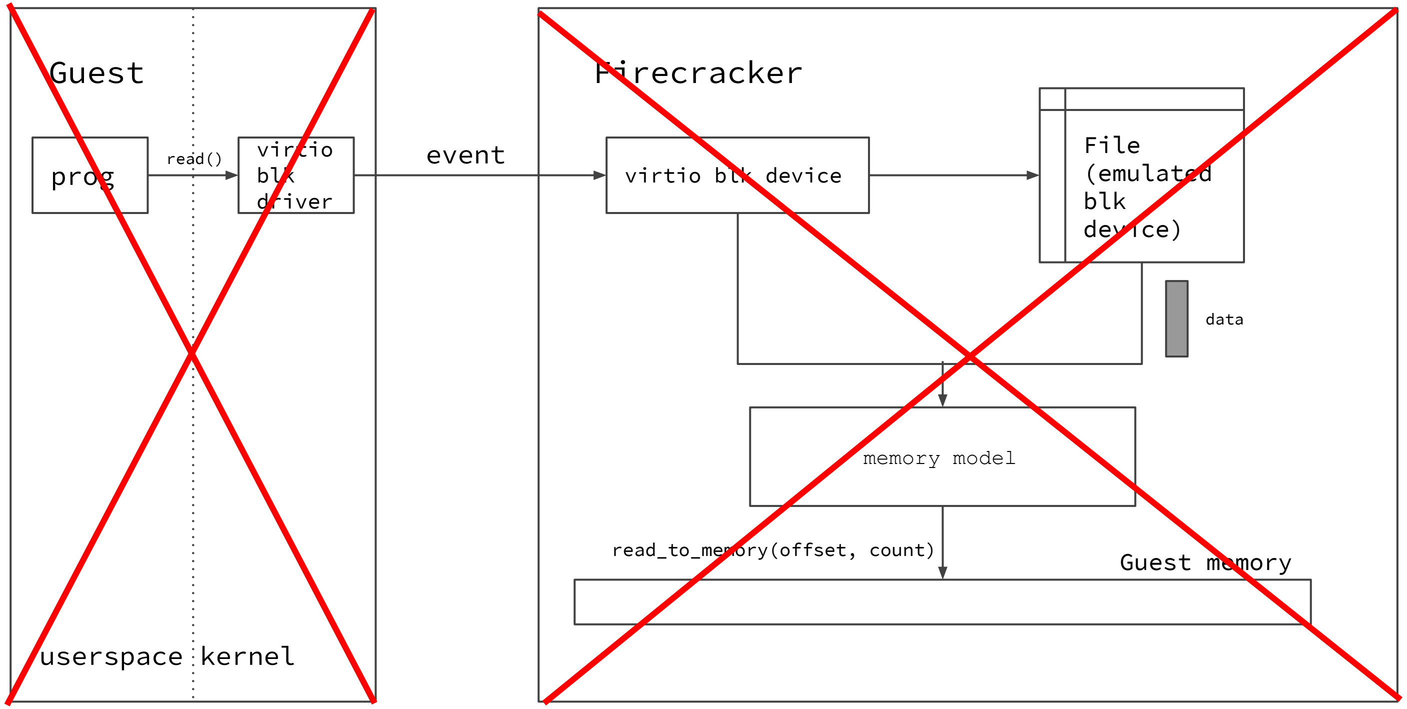
Anatomy of a read()



Anatomy of a read() - Silent Failure



Anatomy of a read() - Panic



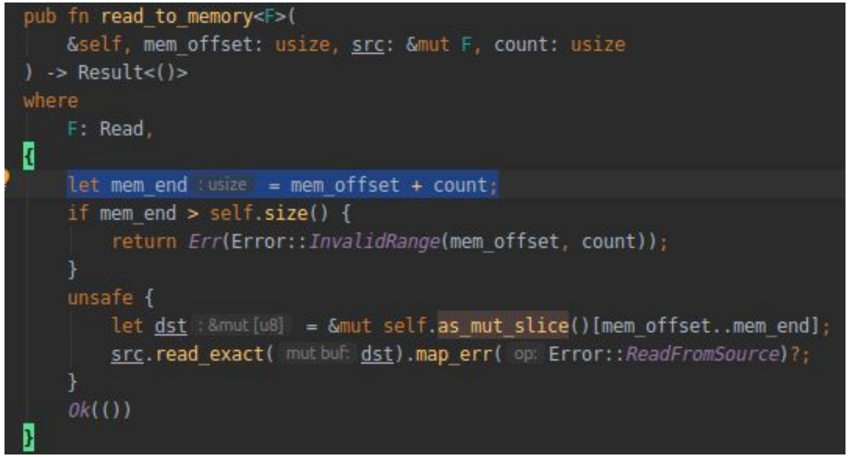
Rust Panic

- Expected problems: `Result` type, error propagation
- Unexpected problems: `panic`
  - Unwind: affects panicking thread, recoverable
  - Abort: SIGABRT, affects all threads, unrecoverable

Rust Panic

- Expected problems: `Result` type, error propagation
- Unexpected problems: `panic`
  - Unwind: affects panicking thread, recoverable
  - Abort: **SIGABRT**, affects all threads, irrecoverable
- Panic hook
  - Flush metrics
  - Say goodbye

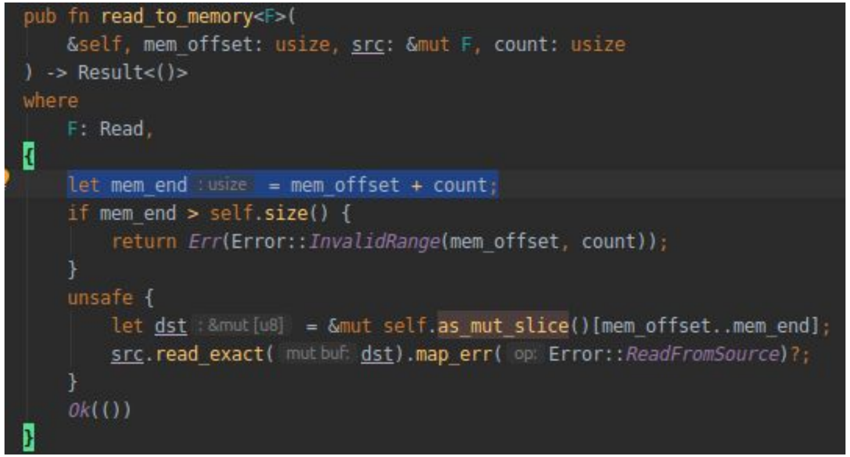
The Problem



```
pub fn read_to_memory<F>(
    &self, mem_offset: usize, src: &mut F, count: usize
) -> Result<()>
where
    F: Read,
{
    let mem_end : usize  = mem_offset + count;
    if mem_end > self.size() {
        return Err(Error::InvalidRange(mem_offset, count));
    }
    unsafe {
        let dst : &mut [u8]  = &mut self.as_mut_slice()[mem_offset..mem_end];
        src.read_exact( mut buf: dst).map_err( op: Error::ReadFromSource)?;
    }
    Ok(())
}
```

- Caller: virtio device code
- Faulty driver…

The Problem



```
pub fn read_to_memory<F>(
    &self, mem_offset: usize, src: &mut F, count: usize
) -> Result<()>
where
    F: Read,
{
    let mem_end :usize = mem_offset + count;
    if mem_end > self.size() {
        return Err(Error::InvalidRange(mem_offset, count));
    }
    unsafe {
        let dst :&mut [u8] = &mut self.as_mut_slice()[mem_offset..mem_end];
        src.read_exact( mut buf: dst).map_err( op: Error::ReadFromSource)?;
    }
    Ok(())
}
```

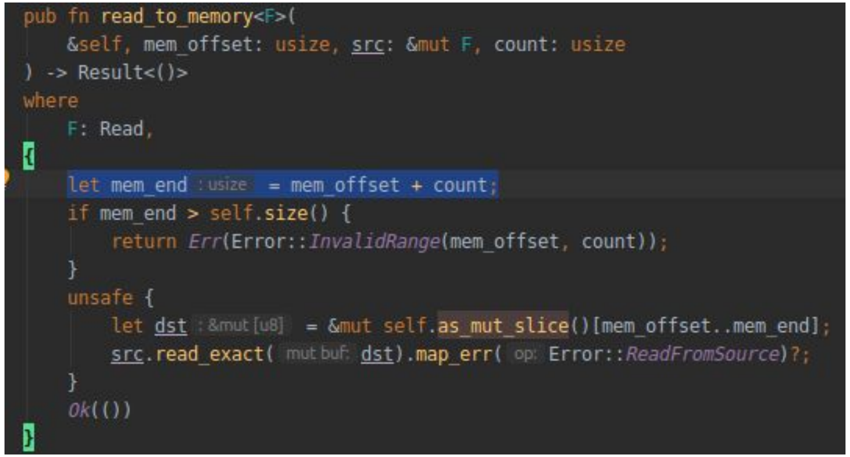
- Caller: virtio device code
- Faulty driver…

Debug build:

```
thread 'fc_vmm' panicked at
'attempt to add with overflow'
```

Very, very bad!

The Problem



```
pub fn read_to_memory<F>(
    &self, mem_offset: usize, src: &mut F, count: usize
) -> Result<()>
where
    F: Read,
{
    let mem_end : usize  = mem_offset + count;
    if mem_end > self.size() {
        return Err(Error::InvalidRange(mem_offset, count));
    }
    unsafe {
        let dst : &mut [u8]  = &mut self.as_mut_slice()[mem_offset..mem_end];
        src.read_exact( mut buf: dst).map_err( op: Error::ReadFromSource)?;
    }
    Ok(())
}
```

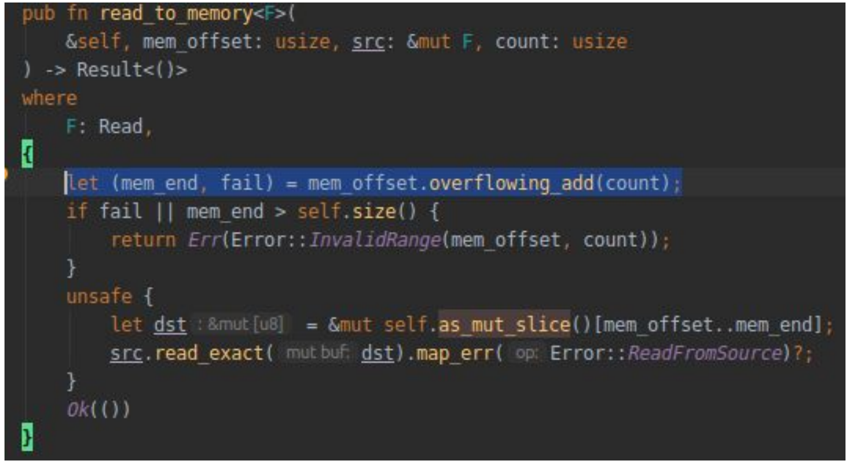
- Caller: virtio device code
- Faulty driver…

Release build:

- No symptoms until malfunction

Worse!

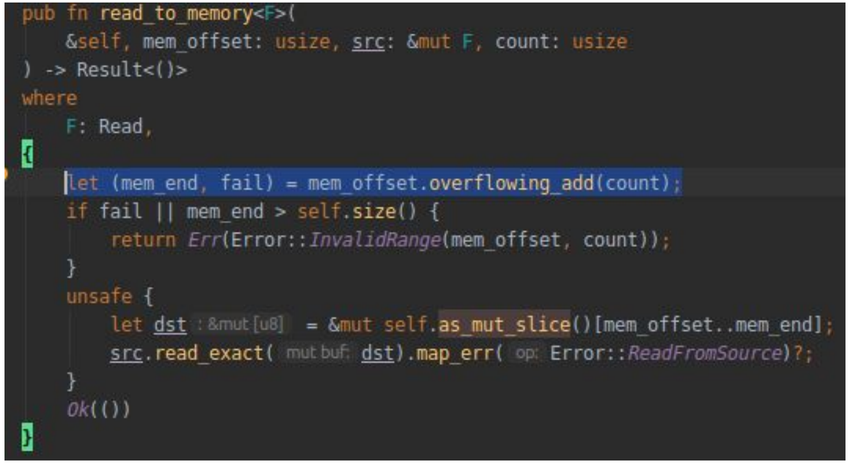
The Solution



```rust
pub fn read_to_memory<F>(
    &self, mem_offset: usize, src: &mut F, count: usize
) -> Result<()>
where
    F: Read,
{
    let (mem_end, fail) = mem_offset.overflowing_add(count);
    if fail || mem_end > self.size() {
        return Err(Error::InvalidRange(mem_offset, count));
    }
    unsafe {
        let dst : &mut [u8]  = &mut self.as_mut_slice()[mem_offset..mem_end];
        src.read_exact( mut buf: dst).map_err( op: Error::ReadFromSource)?;
    }
    Ok(())
}
```

- Checked arithmetic: Rust standard
- Turns a hidden panic condition into a gracefully handled `Result`
- Faulty driver…

The Solution



```rust
pub fn read_to_memory<F>(
    &self, mem_offset: usize, src: &mut F, count: usize
) -> Result<()>
where
    F: Read,
{
    let (mem_end, fail) = mem_offset.overflowing_add(count);
    if fail || mem_end > self.size() {
        return Err(Error::InvalidRange(mem_offset, count));
    }
    unsafe {
        let dst : &mut [u8]  = &mut self.as_mut_slice()[mem_offset..mem_end];
        src.read_exact( mut buf: dst).map_err( op: Error::ReadFromSource)?;
    }
    Ok(())
}
```

- Checked arithmetic: Rust standard
- Turns a hidden panic condition into a gracefully handled `Result`
- Faulty driver…
  - Logged error message
  - Incremented error metric

The Whys

- Why were there no overflow checks in place?
  - Hidden error condition
  - Code unchanged since #1
- Why didn’t we catch it in Firecracker CI?
  - Community report, community fix
  - Drivers in CI images didn’t trigger it
  - **CI didn’t lint Rust code**

rust-clippy

- Rust code linter, available as `cargo` subcommand
- `clippy::integer_arithmetic`
- **>200** warnings at the time this issue was fixed
  - Correctness, Restriction, Style and more

Corrective Actions

- `cargo clippy` test in Firecracker CI
  - Warnings treated as errors
  - Find and fix obscure error conditions
  - Improve overall code quality
- Replace panic conditions with error propagation
  - `unwrap`, `expect`
- Roadmap: virtio device input fuzzing

Lessons Learned

- Testing, testing, testing!
- Linting, linting, linting!
- The Rust compiler is strict, but doesn’t protect from everything

Conclusions

- Seccomp is hard
- Math is hard

The problem is not the problem, but your attitude about the problem.

Capt. Jack Sparrow

Thank you!

Andreea Florescu
fandree@amazon.com

Alexandra Iordache
aghecen@amazon.com