RUST-VMM

A Security Journey

Andreea Florescu, fandree@amazon.com

What is rust-vmm?

A Short Intro

RUST-VMM

- Virtualization components written in Rust
- Focus on:
  - Quality vs Features
  - Extensibility and Usability
- Main customers: VMMs (e.g Cloud Hypervisor, Firecracker)

Components - Examples

RUST-VMM

- Hypervisor Support:
  - KVM -> kvm-ioctls & kvm-bindings
  - Microsoft Hyper-V -> mshv-ioctls & mshv-bindings
- Devices:
  - Serial Console, RTC -> vm-superio
  - MMIO Bus, PIO Bus, Device Managers
- Virtio:
  - Queues, Virtio Device -> vm-virtio
  - Vhost, Vhost User I2C, Vhost User Backend

The Security Story

Security Journey

- Applying security at multiple levels:
  - Organization Setup
  - Development
  - Documentation
  - Operating in production

Organization Setup

- Writing components in Rust
- One Rust package (crate) per component
- All components run the same set of tests (unit tests, build, linters)
- Audits for vulnerabilities in dependencies

Audit for Vulnerabilities

RUST-VMM

- cargo audit
  - Checks a Rust vulnerability database
  - Vulnerable versions of dependencies
- Dependency versions typically locked in Rust binaries
- Rust-vmm = library components => NO fixed dependencies
- Audit checks MUST be run in consumer products

Development

- Reduced number of (external) dependencies
  - Common dependencies: libc, serde
  - 0-dependency components: vm-fdt, vm-superio, vm-device
- Negative testing
- Reduce the usage of unsafe code

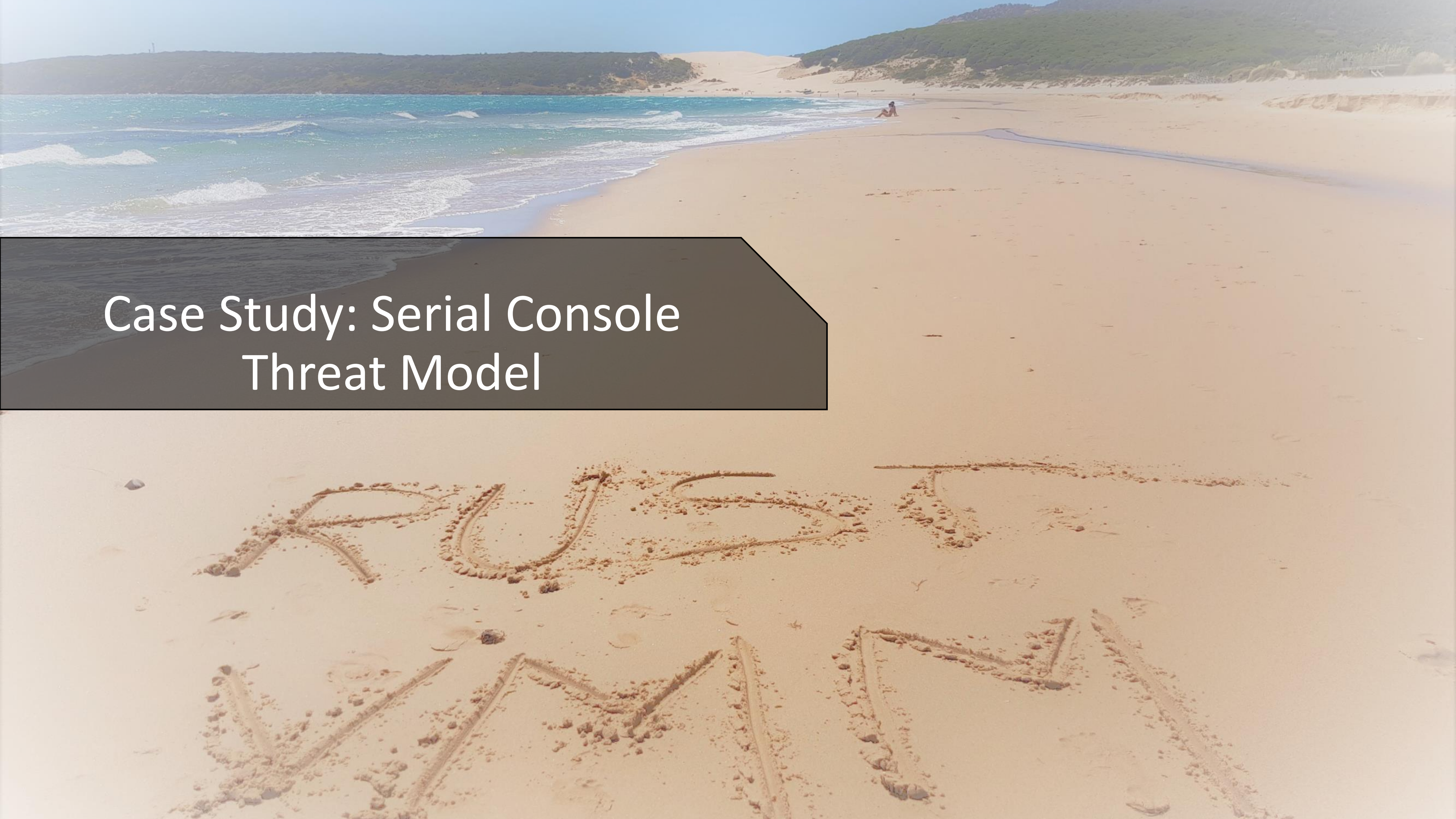
Reduce Unsafe Code

- DON’TS:
  - Write everything in a big unsafe block
- DOs:
  - Limit the unsafe code
  - Document why it’s safe/unsafe -> reduces the risk of code being misused

Documentation

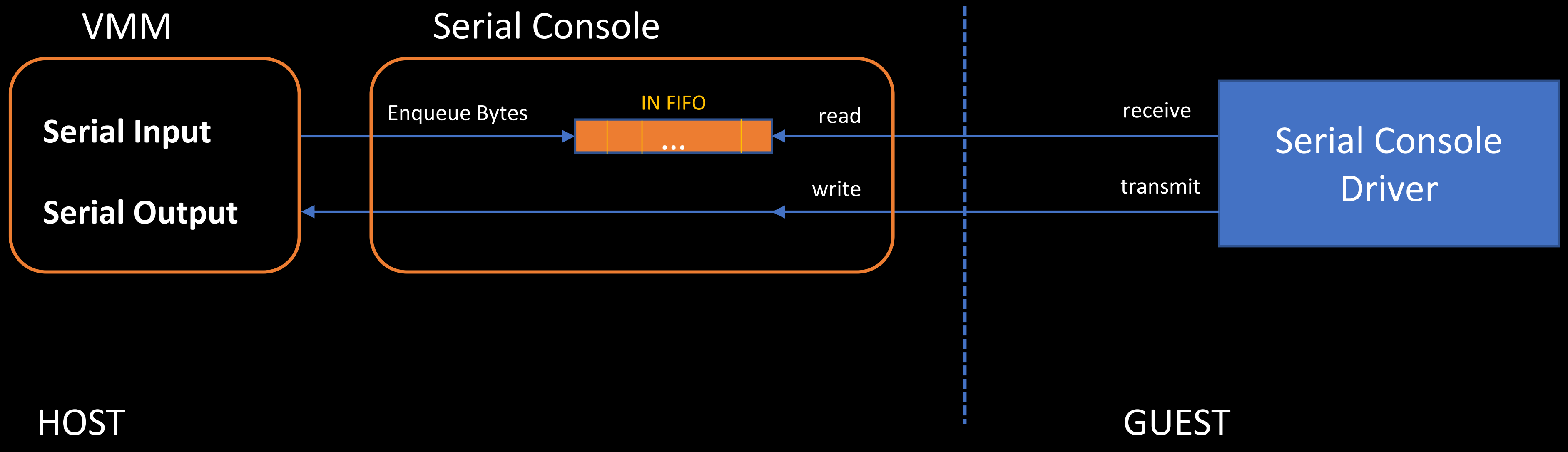
- Document unsafe public functions -> required by Rust
- Threat model documentation:
  - Trusted/untrusted
  - Threats and mitigations
  - Document expectations from consumer products

Case Study: Serial Console Threat Model



Overly Simplified Operation Mode

- UART 16550A serial port with a 64-byte FIFO
- Receiving/Transmitting Data



Serial Console – Threat Model

Threat model available at rust-vmm/vm-superio

1. A malicious guest generates large memory allocations by flooding the serial console input:

   - CVE-2020-27173
   - Fix at the emulation level: limit input FIFO & return errors when FIFO full
   - Fix at the VMM level: handle FIFO full errors

Serial Console – Threat Model (2)

2. A malicious guest can fill up the host disk by generating a high amount of data to be written to the serial output.

  - Output in full control of the consumer
  - Mitigation only possible at the VMM level
    - Rate limit the output (e.g. ring buffer, named pipe)

Lessons Learned

Read code with security in mind
Follow the input/output

Fuzzing Virtualization Components

RUST-VMM

- Component based fuzzing
- Advantages:
  - Fuzzing library code -> easy to pass input to target interface
  - Test components in isolation
  - Low level testing
- Disadvantage:
  - Testing side effects becomes harder
  - Identified issues might not reproduce
  - Mock driver code

Preparing Virtio Components for Fuzzing

RUST-VMM

- Identify the target interfaces:
  - Queues
  - Device Implementation (virtio-blk)
- Build reusable mock-ups:
  - Partially implemented as part of GSoC 2021
  - Create descriptor chains
  - Write arbitrary (fuzz) data in descriptor chains

Preparing Virtio Components for Fuzzing (2)

- Create a specialized mock for devices:
    - Balance between random data and useful data
    - Re-use mock for unit/integration tests

What if you discover a vulnerability?

Reporting Security Vulnerability

- Find the appropriate security vulnerability process
- rust-vmm/${name}/**security/policy**
  - https://github.com/rust-vmm/vm-virtio/security/policy
  - https://github.com/rust-vmm/vm-virtio/security/policy
  - …
- tl;dr: send encrypted email to rust-vmm maintainers

Key Takeaways

Apply security at all levels from project setup to development, and operation

Read code with a security hat on (and then write that threat model)

Use the security process for reporting vulnerabilities

Thank you!

P.S. Photo from Tarifa, Spain, 2019