FUZZING DEVICE MODELS IN RUST

Common Pitfalls

- Andreea Florescu, fandree@amazon.com -
Automated testing
- Check how system behaves with unexpected input
- Findings: crashes, hangs, timeouts

FUZZING - QUICK INTRO
FUZZING - START WITH CORPUS

First Run: Generate Corpus
FUZZING - START WITH CORPUS

First Run: Generate Corpus

Next Runs: Extend Corpus
Fuzzing in Rust-VMM/VM-Virtio

- Implemented fuzz targets for:
  - Virtio Queue
  - Virtio Queue Serialization
  - Virtio Vsock (Packet)
- Discovered 3 crashes – only 1 triggable by malicious drivers
- Fuzzing runs for 15 minutes on Pull Requests
- Fuzzing at the library level using libfuzzer
- Cloud Hypervisor discovered 1 timeout
P1: RUN TIMEOUT IS TOO LARGE

- Fuzzers have a default timeout for hangs/timeouts (i.e. libfuzzer 20 minutes)
- Instead: adjust timeout to suit your use case
Advantages

- It’s EASY to set up
- Can run on almost any host
- Runs in userspace

Disadvantages

- Doesn’t cover the whole virtio setup
- Needs mocking for the driver side of communication – complicated
- Can find false positives
EASY IS A GOOD THING!
Fuzzing at the Library Level

Advantages

- It’s EASY to set up
- Can run on almost any host
- Runs in userspace

Disadvantages

- Doesn’t cover the whole virtio setup
- Needs mocking for the driver side of communication – complicated
- Can find false positives
MOCKING THE DRIVER
MOCKING THE DRIVER

What we want to test
MOCKING THE DRIVER

What we need to mock

Driver
driver
write

Memory

Device
read
do stuff with data

What we want to test
MOCKING THE DRIVER

- Needed for unit tests as well
- Initial version of a mock interface from the beginning
- Evolve the mock interface as you implement features and devices
Fuzzing High Level Flow

1. Parse random bytes
2. Create queue
3. Call queue functions
4. Descriptors
5. Queue functions
6. Mock driver
P2: RETROFIT FUZZING WHEN THE PROJECT IS MATURE

- Instead: keep fuzzing in the back of your head, think about how mock objects can be reused
P3: CRASH ON INVALID INPUT

- Instead: return error to be processed at higher level
STRUCTURE AWARE FUZZING
STRUCTURE AWARE FUZZING

Without

fuzz_target!(|data: &[u8]| { // Interpret data as bytes // sense for the library // fuzz.
})

With

fuzz_target!(|color: Rgb| { // Data already parsed as // what you want to fuzz.
})
STRUCTURE AWARE FUZZING

- Implemented with Arbitrary
- Significantly reduces the code you need for parsing
- LOC: 270 vs 738
STRUCTURE AWARE FUZZING - PROBLEMS

- Not reproducible
- Reads introduce randomness
  - `read_corpus(write_corpus())` – not idempotent
- Cannot use it with a custom corpus
P4: RELY ON INCREMENTAL IMPROVEMENTS

- Instead: check that the tools you want to use have support for future extensions or allow sufficient time for changes
THE CURIOUS CASE OF AN OVERFLOW NOT FOUND BY FUZZING...
THE BUG: EXPLAINED

- Addition can overflow
- Bug found during code review
ANALYSIS AND IMPROVEMENTS

- With fuzzing: ~40 minutes to find the bug
- Added an optional fuzz session that runs for 24 hours:
  - Needs to be started by one of the maintainers
  - Should be started only when needed (changes impacting device implementation)
P5: NOT RUNNING FUZZING LONG ENOUGH

- Instead: figure out a way to include fuzzing for extended period of times without disrupting development
CODE COVERAGE FOR FUZZING
COVERAGE IN RUST

- Llvm-cov
- In rust: only line coverage
Coverage Report

- Coverage for queue.rs file

<table>
<thead>
<tr>
<th>Fuzzing Config</th>
<th>Missed Regions</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 minutes</td>
<td>34</td>
<td>81.82%</td>
</tr>
</tbody>
</table>
# Coverage Report

- Coverage for queue.rs file

<table>
<thead>
<tr>
<th>Fuzzing Config</th>
<th>Missed Regions</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 minutes</td>
<td>34</td>
<td>81.82%</td>
</tr>
</tbody>
</table>

😊
### Coverage Report

- Coverage for queue.rs file

<table>
<thead>
<tr>
<th>Fuzzing Config</th>
<th>Missed Regions</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 minutes</td>
<td>34</td>
<td>81.82%</td>
</tr>
<tr>
<td>15 minutes + minimal corpus</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

😊
### Coverage Report

- Coverage for queue.rs file

<table>
<thead>
<tr>
<th>Fuzzing Config</th>
<th>Missed Regions</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 minutes</td>
<td>34</td>
<td>81.82%</td>
</tr>
</tbody>
</table>
| 15 minutes + minimal corpus     | 34             | 81.82%   | 😞 😞
Coverage Report

- Coverage for queue.rs file

<table>
<thead>
<tr>
<th>Fuzzing Config</th>
<th>Missed Regions</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 minutes</td>
<td>34</td>
<td>81.82%</td>
</tr>
<tr>
<td>15 minutes + minimal corpus</td>
<td>34</td>
<td>81.82%</td>
</tr>
<tr>
<td>2 weeks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Coverage Report

- Coverage for `queue.rs` file

<table>
<thead>
<tr>
<th>Fuzzing Config</th>
<th>Missed Regions</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 minutes</td>
<td>34</td>
<td>81.82%</td>
</tr>
<tr>
<td>15 minutes + minimal corpus</td>
<td>34</td>
<td>81.82%</td>
</tr>
<tr>
<td>2 weeks</td>
<td>30</td>
<td>83.96%</td>
</tr>
</tbody>
</table>
Coverage Report

- Coverage for queue.rs file

<table>
<thead>
<tr>
<th>Fuzzing Config</th>
<th>Missed Regions</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 minutes</td>
<td>34</td>
<td>81.82%</td>
</tr>
<tr>
<td>15 minutes + minimal corpus</td>
<td>34</td>
<td>81.82%</td>
</tr>
<tr>
<td>2 weeks</td>
<td>30</td>
<td>83.96%</td>
</tr>
</tbody>
</table>
P6: USING COVERAGE TO DECIDE WHEN TO STOP FUZZING

- Instead: use coverage to understand how to extend fuzz targets
Summary

- P1: Run Timeout is too large
- P2: Retrofit Fuzzing when the project is mature
- P3: Crash on invalid input
- P4: Rely on incremental improvements
- P5: Not running fuzzing long enough
- P6: Using coverage to decide when to stop fuzzing

Fuzzing does not need to be hard to be useful.
WHY COVERAGE WAS 81.82%?

- Functions not called on purpose:
  - Iterators over descriptor chains -> the data needs to be interpreted by devices anyway
- Missed to call 1 function: desc_table

<table>
<thead>
<tr>
<th>Fuzzing Config</th>
<th>Missed Regions</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 minutes</td>
<td>34</td>
<td>81.82%</td>
</tr>
<tr>
<td>15 minutes + minimal corpus</td>
<td>34</td>
<td>81.82%</td>
</tr>
<tr>
<td>2 weeks</td>
<td>30</td>
<td>83.96%</td>
</tr>
<tr>
<td>15 mins + missing funcs</td>
<td>26</td>
<td>86.1%</td>
</tr>
</tbody>
</table>
Most of the missed coverage regions are macros that are printing errors

logger is not initialized when running fuzzing
Fuzzing Findings

- Index out of bounds access in the Virtio Mock implementation
  - https://github.com/rust-vmm/vm-virtio/pull/162/commits/e42fe6b3165aceec7183e206874d5970a6e591f7

- Panic when using wrong ordering in functions called by VMM:
  - https://github.com/rust-vmm/vm-virtio/pull/174
  - Also, excluded the invalid ordering in fuzzer

- Division by 0 in descriptor chains iterator:
  - https://github.com/rust-vmm/vm-virtio/pull/173