Challenges in Fuzzing the Linux Kernel Networking Subsystem with Syzkaller

2019-18873
송수빈

Introduction

Syzkaller is a kernel fuzzer that uses syscall descriptions to effectively fuzz kernel code by executing random sequences of system calls. While it is possible to fuzz the Linux kernel networking subsystem with Syzkaller, there are several challenges in doing so, compared to other subsystems. In this research, I try to fuzz the Linux kernel network subsystem using Syzkaller, inspect the resulting coverage, and analyze why it is difficult to effectively achieve high coverage. Also, based on the prior analysis result, I propose some solutions to improve the effectiveness of the fuzzing process.

Background

Syzkaller

- A coverage-guided kernel fuzzer made by Google
  - Found hundreds of bugs in the Linux kernel so far
- Uses sequence of system calls as an input; Dependencies and requirements for them are manually documented as **"syscall descriptions"**
- **Architecture**
  - Actual fuzzing is done inside isolated VMs; The host OS only controls the lifecycle of VMs through Manager
  - Inside the VM, Fuzzer generates/mutates inputs, and each Executor process executes the input

External Network Fuzzing with Syzkaller

- Due to Syzkaller's architecture, we can't send packets from outside of the VM to the guest kernel
- Instead, Syzkaller uses the **TUN/TAP interface** to simulate packets coming from outside of the kernel

TUN/TAP

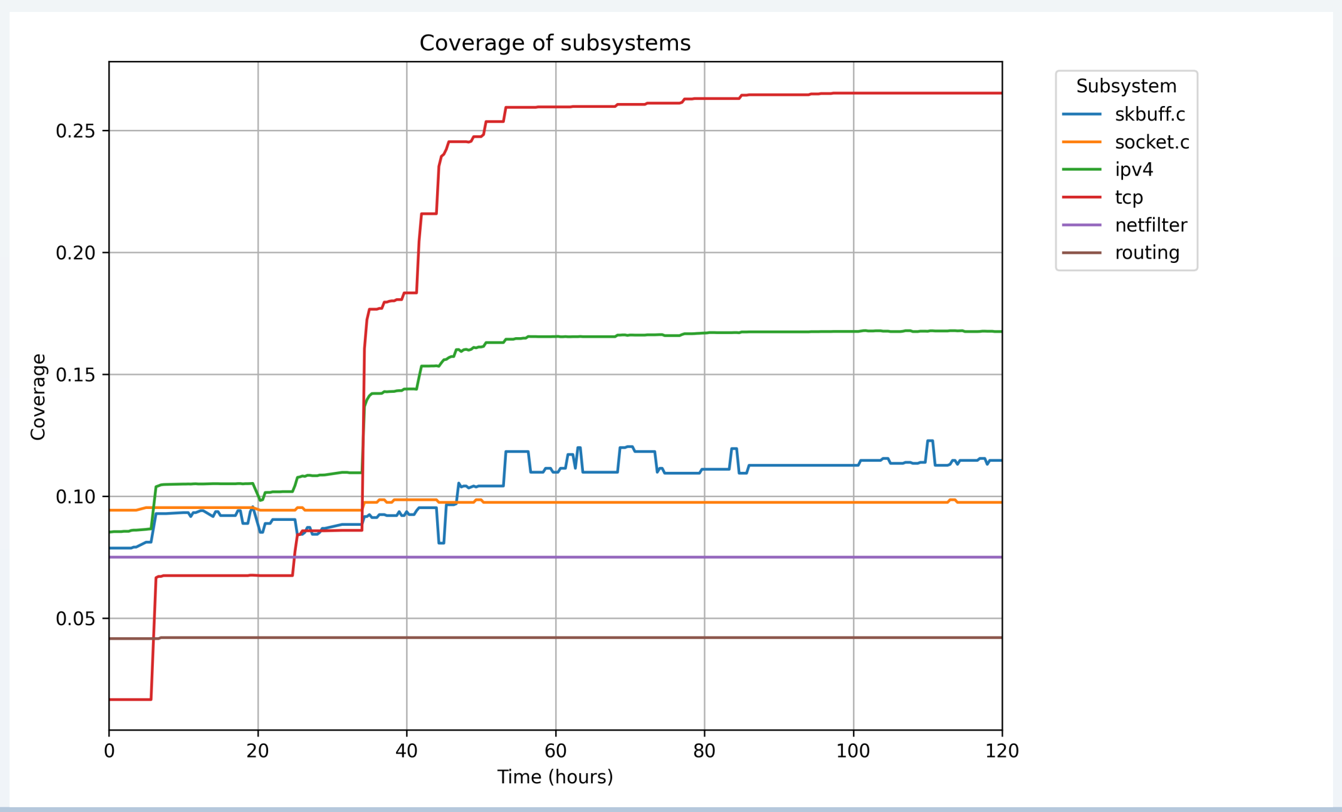
- Virtual network devices in Linux kernel that allow packet reception and transmission for user space programs
- By writing to a TUN/TAP device, a user space program can send a packet to the kernel as if it came from the outside
- By reading from a TUN/TAP device, a user space program can read what the kernel has output to the network interface

Experiment

Setup

- **Host** : Ubuntu 22.04 on Intel® Core™ i5-7500@3.40GHz and 8GB memory
- **Guest** : Three x86-64 QEMU VMs with 2 CPUs and 2GiB memory each, Debian Bullseye (Linux kernel v6.6.1)
- Syzkaller Configs
  - Coverage Filter : `net/ipv4/, net/core/, net/socket.c`
  - Enabled Syscalls : `socket(), bind(), listen(), accept(), syz_emit_ethernet(), syz_extract_tcp_res()` (Syzkaller pseudo-syscalls)
- Fuzzed for **120 hours**

Result



Discussion

Packet Descriptions in Syzkaller

- Simply using randomly generated bytes as inputs cannot help achieve good coverage
- An input should follow the basic packet structure of each protocol, otherwise it will be dropped
  - *e.g.*, Proper packet length, correct header fields & checksum
- In Syzkaller, such constraints are manually documented as **descriptions**

Statefulness of TCP

- Coverage in TCP & IP subsystem rapidly increases after about 35 hours of fuzzing
  - This means that **TCP handshake** was done and the fuzzer entered the TCP subsystem
- As Syzkaller programs are **stateless**, it is difficult to pick up the required event sequence for handshake

Lack of Different Routing Scenarios

- VMs in Syzkaller communicate only with the host OS for fuzzing process control, and don't communicate with other machines
- Therefore, the scenario of receiving an IP packet and forwarding it to another host is not considered
  - There are no routes to other machines in the guest kernels' routing tables
  - Important functions in the packet forwarding scenario(*e.g.*, `__mkroute_input()` in `route.c`, `ip_forward()` in `ip_forward.c`) have **zero coverage**

Future Work

Fuzz without Packet Descriptions

- It is cumbersome to manually write packet descriptions whenever a new network protocol is added to the kernel
- We can instead capture **real network traffic** and use them as a basis for mutation
- Or, we can simply use sockets to write data and let the kernel generate the actual packet (output path)
  - Then, use the TAP interface to mirror the packet back to input path, after optional mutation
- Checksums and Cryptographic fields in packet headers still need to be addressed

Add More Routing Scenarios

- Allowing VMs to communicate with each other may result in **false positive** crashes
- Crashes should only occur from fuzzing, not external traffic; *i.e.*, Each VM must be isolated
- We can simulate multiple hosts in a single executor process using the TAP interface, as it is done in Syzkaller for server and client for TCP handshake
- Corresponding route should also be added to the routing table
- This will increase coverage in IPV4 subsystem and Routing subsystem