HEALER: Relation Learning Guided Kernel Fuzzing

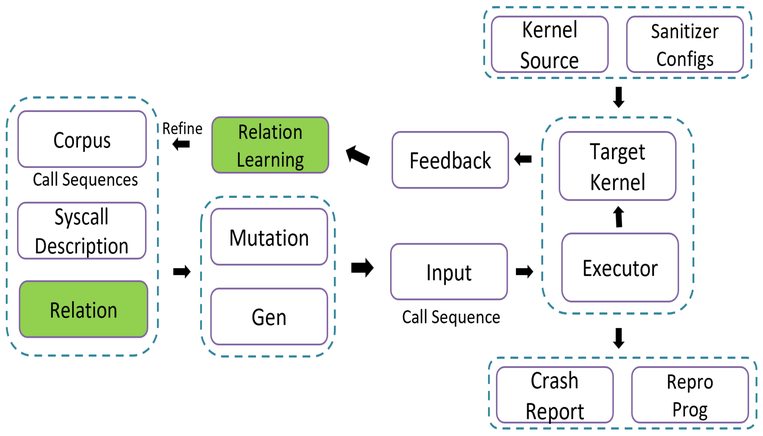
Hao Sun[1], Yuheng Shen[1], Cong Wang[1], Jianzhong Liu[1], Yu Jiang[1], Ting Chen[2], Aiguo Cui[3]
KLISS, BNRist, School of Software, Tsinghua University, Beijing, China[1],
Center for Cybersecurity, University of Electronic Science and Technology of China, Chengdu, China[2]
Huawei Technologies Co., Ltd, China[3]

SOSP'21

Motivation

```
sock_fd = socket(AF_INET, SOCK_STREAM, 0)
                    ↓ Create socket inside the kernel
bind(sock_fd, &addr, sizeof(addr))
                    ↓ Bind address to the socket
          listen(sock_fd, …)
                    ↓ Mark socket
accept(sock_fd, &peer_addr, &size)
```

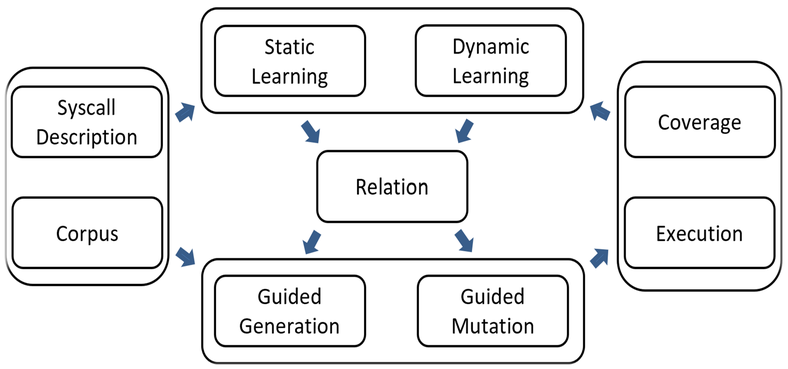
- Former calls setup related kernel states.
- The latter calls can be influenced by those states.
- Influence relations exist between two system calls if a former can alter the latter's execution path.



- Existing work generates and mutates call sequences based on empirical methods, e.g., Syzkaller uses choice table to guide input generation, which may even hinder its fuzzing capabilities.
- We propose to use **relation** to guide the generation and mutation, refine relations with relation learning.

Influence relations exist between syscalls. The quality of generated input can be improved with relations.

Solution



- Learn influence relations between system calls continuously and dynamically.
- Guide call sequence generation and mutation with learned relations to improve the quality of input.

Step 1. Static Learning

```
socket$inet(domain const[AF_INET], type flags[socket_type], proto int32, sock_in
bind$inet(fd sock_in, addr ptr[in, sockaddr_in], addrlen len[addr])

socketpair(domain flags[domain], type flags[socket_type], proto int32, fds ptr[out, sock_pair])
bind$inet(fd sock_in, addr ptr[in, sockaddr_in], addrlen len[addr])

resource sock_in[sock]
socket$inet(domain const[AF_INET], type flags[socket_type], proto int32) sock_in
listen(fd sock, backlog int32)
```

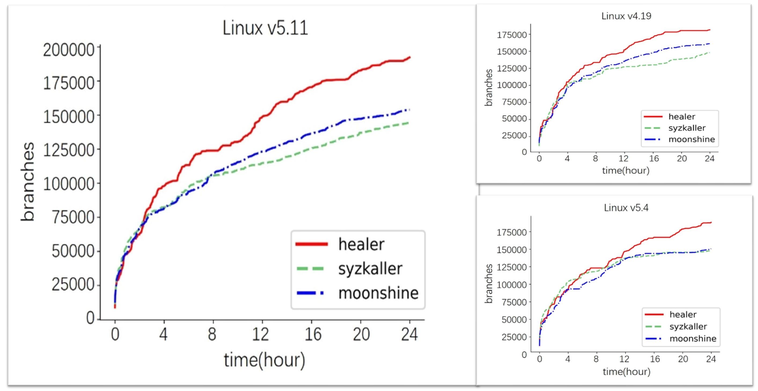
- By analyzing the usage of resource type, obvious relations can be derived.

Step 2. Dynamic Learning

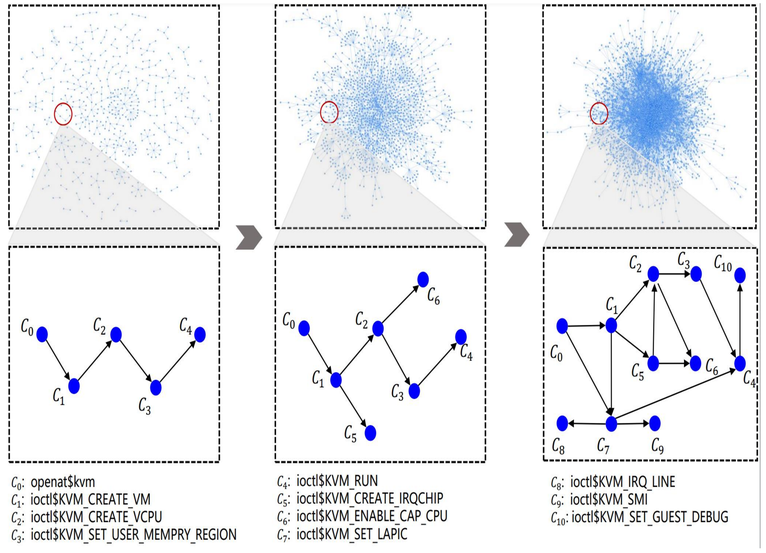
- For adjacent call pair (C_i, C_j) in a minimized sequence:
  - Remove C_i, observe the coverage change of C_j
  - If the coverage of C_j changed, then C_i must have influence relation with C_i, because they're adjacent.
  - Execute whenever an interesting input is discovered.

Initialize with static learning and refine with dynamic learning. Guide synthesis with learned relations.

Evaluation



- Improve coverage by **28%**, **21%**, compared with Syzkaller and Moonshine, with **2.2x**, **1.8x** speedup.
- Healer found **218** bugs during two weeks, **33** are confirmed by maintainers as previously-unknown.



- Algorithm is learning a complex graph, where each node represents a system call and each edge represents influence relation.

Evaluation results demonstrate that relation play a significant role in improving the quality of input.