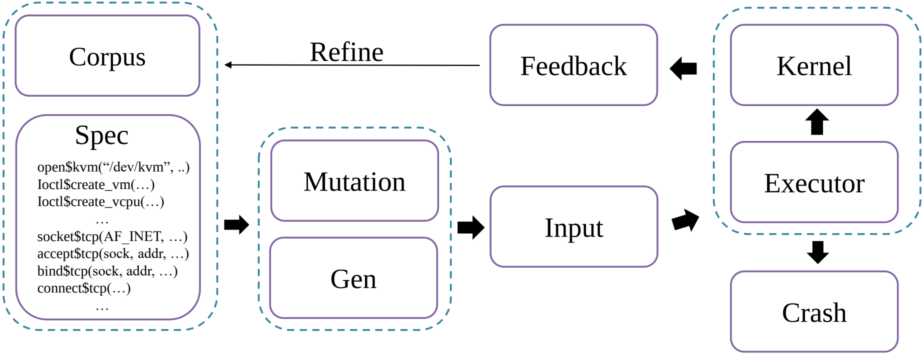
KSG: Augmenting Kernel Fuzzing with System Call Specification Generation

Hao Sun, Yuheng Shen, Jianzhong Liu, Yiru Xu, Yu Jiang

Tsinghua University

Kernel Fuzz Testing



System Call Specification

Socket system call prototype

```
int socket(int domain, int type, int protocol);
int setsockopt(int socket, int level, int option_name,
        const void *val, socklen_t len);
```

Simple program using TCP

```
// setup TCP
sock_tcp = socket(AF_INET, SOCK_STREAM, 0);
...
// setup fields of tcp_repair_window
struct tcp_repair_window window = { .snd_wl1 = ...};
...
// set socket option
setsockopt(sock_tcp, IPPROTO_TCP, TCP_REPAIR_WINDOW,
           &window, sizeof(window));
```

Syzlang specification for TCP

```
resource sock_tcp[sock_in]
tcp_repair_window {
    snd_wl1         int32
    snd_wnd         int32
    …
}
socket$TCP(domain const[AF_INET], type const[SOCK_STREAM],
           protocol const[0]) sock_tcp
setsockopt$TCP(sock sock_tcp, level const[IPPROTO_TCP],
           opt_name const[TCP_REPAIR_WINDOW],
           val ptr[tcp_repair_window], len len[val])
```

- System calls are **hard** to fuzz:
  - abstraction over submodules.
  - accept different types.
- Specifications specialize calls.
- **Bypass** basic validation:
  - input structure.
  - semantics, e.g., length.

Issues

Socket system call prototype

```
int socket(int domain, int type, int protocol);
int setsockopt(int socket, int level, int option_name,
        const void *val, socklen_t len);
```

Simple program using TCP

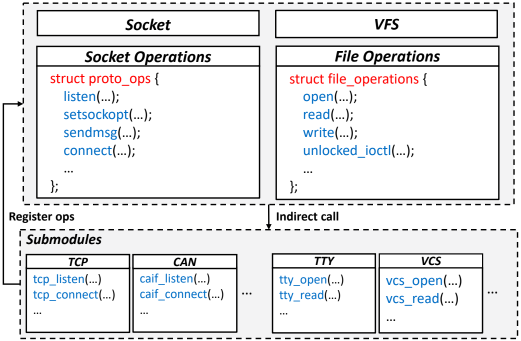
```
// setup TCP
sock_tcp = socket(AF_INET, SOCK_STREAM, 0);
...
// setup fields of tcp_repair_window
struct tcp_repair_window window = { .snd_wl1 = ...};
...
// set socket option
setsockopt(sock_tcp, IPPROTO_TCP, TCP_REPAIR_WINDOW,
          &window, sizeof(window));
```

Syzlang specification for TCP

```
resource sock_tcp[sock_in]
tcp_repair_window {
    snd_wl1         int32
    snd_wnd         int32
    …
}
socket$TCP(domain const[AF_INET], type const[SOCK_STREAM],
          protocol const[0]) sock_tcp
setsockopt$TCP(sock sock_tcp, level const[IPPROTO_TCP],
          opt_name const[TCP_REPAIR_WINDOW],
          val ptr[tcp_repair_window], len len[val])
```

- Encode specifications is extremely **time-consuming**.
- Require knowledge of **submodules**:
  - input types.
  - semantics of each field.
- Require knowledge of **domain lang**:
  - syntax mapping.
  - encode semantics.

Ch1: Extracting Entries of Submodules



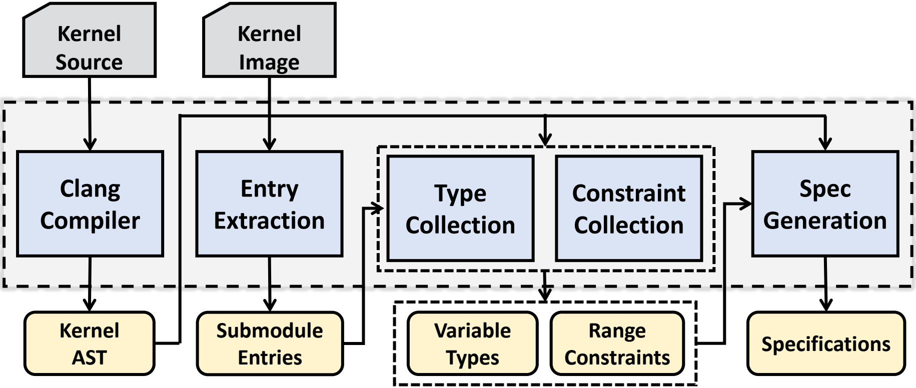
- System calls **dispatch** input to submodules' entries.
- **Submodules' entries** are the target.
- Entries are registered during **different times**:
  - kernel booting.
  - module loading.
- Registered via **various approaches**.

Ch2: Identifying Input Types of Entries

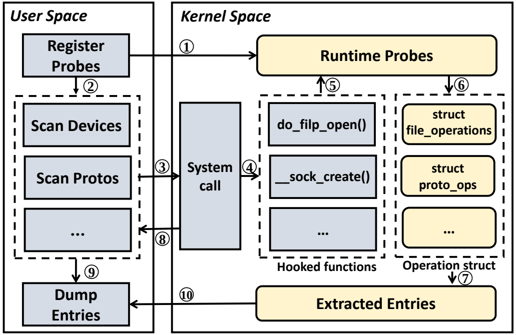
```
static int do_tcp_setsockopt(struct sock *sk, int level,
           int optname, sockptr_t optval, unsigned int optlen)
{
    struct tcp_sock *tp = tcp_sk(sk);
    ...
    switch (optname) {
        case TCP_CONGESTION: {
            char name[TCP_CA_NAME_MAX];
Path1: ➡    // type of `optval` is char[TCP_CA_NAME_MAX]
            strncpy_from_sockptr(name, optval,  …);
        }
        case TCP_MAXSEG:
            int val;
Path2: ➡    // type of `optval` is int*
            copy_from_sockptr(&val, optval, sizeof(val));
            tp->rx_opt.user_mss = val;
        case TCP_REPAIR_WINDOW:
            struct tcp_repair_window opt;
Path3: ➡    // type of `optval` is tcp_repair_window*
            if (copy_from_sockptr(&opt, optval, sizeof(opt)))
                return -EFAULT;
        }
    return err;
}
```

- Input types differ in different paths.
- Some input control the execution path, e.g., *optname*.
- Others may be cast to different types, e.g., *optval*.
- Hard to identify the **precise** type for each field, and corresponding range constraint.

Overview

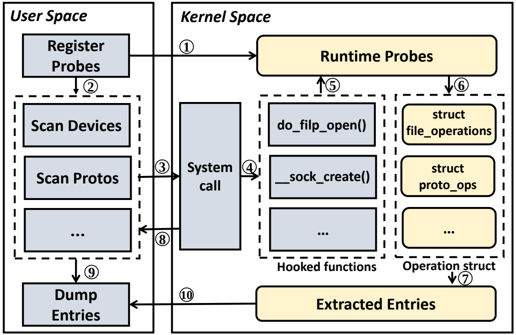

Kernel Source
Kernel Image
Clang Compiler
Entry Extraction
Type Collection
Constraint Collection
Spec Generation
Kernel AST
Submodule Entries
Variable Types
Range Constraints
Specifications

Entry Extraction



- How entries are registered really **doesn't matter**.
- They are eventually stored into the **specific fields**:
  - *file_operations: file->f_ops*.
  - *proto_ops: socket->ops*.
- Extract entries from these fields.

Entry Extraction



- Hook **probes** before kernel functions that create these entries via eBPF and kprobe:
  - *do_filp_open() -> file_operations*.
  - *__sock_create() -> proto_ops*.
- Trigger probes from userspace via **scanning** corresponding resources, e.g., iterate *devs* and *sockets*.
- **Symbolize** extracted entries in userspace with */proc/kallsyms*.

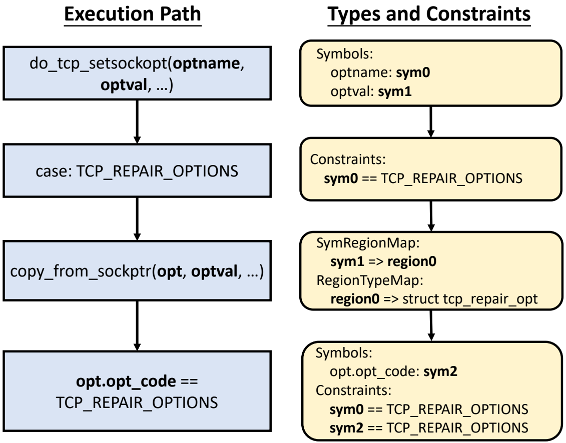
Types and Constraints Collection

Algorithm 1: Collecting Types

```
SymRegionMap := ∅
RegionTypeMap := ∅
RegionMap := ∅
for CastExpr ∈ Entry do
    S := SourceSym(CastExpr)
    T := TargetSym(CastExpr)
    if IsIntegerToPtr(CastExpr) then
        R := Region(T)
        SymRegionMap[S] := R
        continue
    if !IsPtrToPtr(CastExpr) then
        continue
    R0 := Region(S)
    R1 := Region(T)
    Record(R0, R1, RegionMap)
    STy := KnownType(R0, RegionTypeMap)
    TTy := KnownType(R1, RegionTypeMap)
    if IsMorePrecise(STy, TTy) then
        updateRegionType(R1, STy)
    else
        updateRegionType(R0, TTy)
```

- Based on <u>**C**</u>lang <u>**S**</u>tatic <u>**A**</u>nalyzer.
- Collect range constraints with CSA.
- Identify the **most precise** type from each type-related operation.
- Record relationships between symbolic value and memory region.
- Associate type information with memory region.
- Record connections between regions.

Running Example



Specification Generation

```
resource sock_X25_SeqPacket[sock]

socket$X25_SeqPacket(domain const[0x9], type const[0x5],
          proto const[0x0]) sock_X25_SeqPacket

bind$X25_SeqPacket_0(sock sock_X25_SeqPacket, addr
          ptr[in, sockaddr_x25], len bytesize[addr])

setsockopt$X25_SeqPacket_0(sock sock_X25_SeqPacket,
          level const[0x106],  opt_name const[0x1], …)

ioctl$X25_SeqPacket_6(fd sock_X25_SeqPacket, cmd
          const[0x89e5], arg ptr[in, x25_calluserdata])
...

sockaddr_x25{
      sx25_family const[0x9, int16]
      sx25_addr x25_address
}
…
```

- For each execution path, generate specs with two steps.
- Step1 generates system calls that create resources:
  - *open()* for *devs* with corresponding file paths.
  - *socket()* with correct *(domain, type, proto)*.

Evaluation: Specification Generation

```
resource sock_X25_SeqPacket[sock]

socket$X25_SeqPacket(domain const[0x9], type const[0x5],
        proto const[0x0]) sock_X25_SeqPacket

bind$X25_SeqPacket_0(sock sock_X25_SeqPacket, addr
        ptr[in, sockaddr_x25], len bytesize[addr])

setsockopt$X25_SeqPacket_0(sock sock_X25_SeqPacket,
        level const[0x106], opt_name const[0x1], ...)

ioctl$X25_SeqPacket_6(fd sock_X25_SeqPacket, cmd
        const[0x89e5], arg ptr[in, x25_calluserdata])
...

sockaddr_x25{
    sx25_family const[0x9, int16]
    sx25_addr x25_address
}
...
```

- Step2 generates the rest of calls:
  - translate C type to Syzlang type.
  - encode collected range constraints.
  - mark data-flow direction for pointer, e.g., *in* or *out*.
- Take generated specs as input for kernel fuzzers, e.g., Syzkaller.

Evaluation

Specification Generation

KSG extracted 792 entries from 78 sockets and 1098 device files, and the generated specs contain 2433 specialized calls, and 1460 are new to existing specs.

	Scanned	Entries	Specs	New Specs
Socket	78	222	923	+586
Driver	1098	572	1510	+874
Overall	1176	794	2433	**+1460**

Evaluation

Coverage Improvement

With 1460 new specs, Syzkaller achieved 22% coverage improvement on average.

Version	min-impr	max-impr	Average
5.15	+18%	+24%	+21%
5.10	+19%	+25%	+22%
5.4	+20%	+28%	+24%
Overall	+19%	+25%	**+22%**

Evaluation

Bug Finding

KSG assisted fuzzers to discover **26** previously unknown vulnerabilities. All have been confirmed by maintainers; 13 and 6 have been fixed and assigned with CVEs.

Operation	**Risk**	**CVE**
__init_work	use after free	***CVE-2021-4150***
kvm_arch_vcpu_create	logic bug	***CVE-2021-4032***
io_wq_submit_work	logic bug	***CVE-2021-4023***
__btrfs_tree_lock	deadlock	***CVE-2021-4149***
block_invalidatepage	dereference null	***CVE-2021-4148***
rdma_listen	use after free	***CVE-2021-4028***

Summary

- Utilize probe-based tracing to extract entries.
- Collect types and constraints based on CSA.
- Generated specifications can improve performance of fuzzers.
- In future, we will extend KSG to other submodules and implement checkers to collect more semantics.

Thanks for your attention!

Q & A