KSG: Augmenting Kernel Fuzzing with System Call Specification Generation

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

Tsinghua University
Kernel Fuzz Testing

Challenges

KSG Design

Evaluation

Corpus

Spec

openSKvm("/dev/kvm", ..)
ioctl$create_vm(...)
ioctl$create_vcpu...

Mutation

Gen

Feedback

Input

Kernel

Executor

Crash

Hao Sun

Tsinghua University

KSG

2 / 18
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.
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

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.

```
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];
            // type of `optval` is char[TCP_CA_NAME_MAX]
            strncpy_from_sockptr(name, optval, ...);
        }
        case TCP_MAXSEG: {
            int val;
            // 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;
            if (copy_from_sockptr(&opt, optval, sizeof(opt)))
                return -EFAULT;
        }
    }
    return err;
}
```
Overview

Kernel Source → Clang Compiler → Kernel AST
Kernel Image → Entry Extraction → Submodule Entries

- Type Collection
- Constraint Collection

- Variable Types
- Range Constraints

→ Spec Generation

→ Specifications
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`.

![Diagram of Entry Extraction process](attachment://entry_extraction_diagram.png)
Types and Constraints Collection

**Algorithm 1: Collecting Types**

```
1 SymRegionMap := ∅
2 RegionTypeMap := ∅
3 RegionMap := ∅
4 for CastExpr ∈ Entry do
5     S := SourceSym(CastExpr)
6     T := TargetSym(CastExpr)
7     if IsIntegerToPtr(CastExpr) then
8         R := Region(T)
9         SymRegionMap[S] := R
10        continue
11     if !IsPtrToPtr(CastExpr) then
12        continue
13     R0 := Region(S)
14     R1 := Region(T)
15     Record(R0, R1, RegionMap)
16     STy := KnownType(R0, RegionTypeMap)
17     TTy := KnownType(R1, RegionTypeMap)
18     if IsMorePrecise(STy, TTy) then
19         updateRegionType(R1, STy)
20     else
21         updateRegionType(R0, TTy)
```

- Based on **Clang Static Analyzer**.
- 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

**Execution Path**

```
do_tcp_setsockopt(optname, optval, …)
```

```
case: TCP_REPAIR_OPTIONS
```

```
copy_from_sockptr(opt, optval, …)
```

```
opt.opt_code == TCP_REPAIR_OPTIONS
```

**Types and Constraints**

**Symbols:**
- optname: `sym0`
- optval: `sym1`

**Constraints:**
- `sym0 == TCP_REPAIR_OPTIONS`

**SymRegionMap:**
- `sym1` => `region0`

**RegionTypeMap:**
- `region0` => `struct tcp_repair_opt`

**Symbols:**
- `opt.opt_code`: `sym2`

**Constraints:**
- `sym0 == TCP_REPAIR_OPTIONS`
- `sym2 == TCP_REPAIR_OPTIONS`
For each execution path, generate specs with two steps.

Step 1 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., \textit{in} or \textit{out}.

- Take generated specs as input for kernel fuzzers, e.g., Syzkaller.
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.

<table>
<thead>
<tr>
<th></th>
<th>Scanned</th>
<th>Entries</th>
<th>Specs</th>
<th>New Specs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>78</td>
<td>222</td>
<td>923</td>
<td>+586</td>
</tr>
<tr>
<td>Driver</td>
<td>1098</td>
<td>572</td>
<td>1510</td>
<td>+874</td>
</tr>
<tr>
<td>Overall</td>
<td>1176</td>
<td>794</td>
<td>2433</td>
<td>+1460</td>
</tr>
</tbody>
</table>
Coverage Improvement

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

<table>
<thead>
<tr>
<th>Version</th>
<th>min-impr</th>
<th>max-impr</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.15</td>
<td>+18%</td>
<td>+24%</td>
<td>+21%</td>
</tr>
<tr>
<td>5.10</td>
<td>+19%</td>
<td>+25%</td>
<td>+22%</td>
</tr>
<tr>
<td>5.4</td>
<td>+20%</td>
<td>+28%</td>
<td>+24%</td>
</tr>
<tr>
<td>Overall</td>
<td>+19%</td>
<td>+25%</td>
<td>+22%</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Risk</th>
<th>CVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>__init_work</td>
<td>use after free</td>
<td>CVE-2021-4150</td>
</tr>
<tr>
<td>kvm_arch_vcpu_create</td>
<td>logic bug</td>
<td>CVE-2021-4032</td>
</tr>
<tr>
<td>io_wq_submit_work</td>
<td>logic bug</td>
<td>CVE-2021-4023</td>
</tr>
<tr>
<td>__btrfs_tree_lock</td>
<td>deadlock</td>
<td>CVE-2021-4149</td>
</tr>
<tr>
<td>block_invalidatepage</td>
<td>dereference null</td>
<td>CVE-2021-4148</td>
</tr>
<tr>
<td>rdma_listen</td>
<td>use after free</td>
<td>CVE-2021-4028</td>
</tr>
</tbody>
</table>
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