Syzkaller

Future development

Linux Plumbers, Nov 4, 2016, Santa Fe

Dmitry Vyukov, dvyukov@, Google

Agenda

- Why
- How
- Work in progress, ideas, problems

Existing Fuzzers

trinity/iknowthis in essence:

```
syscall(rand(), rand(), rand());
```

Do know argument types, so more like:

```
syscall(rand(), rand_fd(), rand_addr());
```

- tend to find shallow bugs
- frequently no reproducers
- poorly suitable for regression testing

Coverage-guided Fuzzing

void test(const char *data, int size) { ... do something with data ... }

Fuzzer invokes the function with different inputs.

Code coverage guiding:

- corpus of "interesting" inputs
- mutate and execute inputs from corpus
- if inputs gives new coverage, add it to corpus

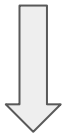
Turns exponential problem into linear.

But how to apply it to kernel?

Coverage (CONFIG_KCOV)

GCC pass that inserts a function call into every basic block (piece of code without branches):

```
if (...) {
     ...
}
```



```
__fuzz_coverage();
if (...) {
     __fuzz_coverage();
     ...
}
__fuzz_coverage();
```

\+ kernel debugfs extension that collects and exposes coverage per-thread.

Syscall Description

Declarative description of all syscalls:

```
open(file filename, flags flags[open_flags],
      mode flags[open_mode]) fd
read(fd fd, buf buffer[out], count len[buf])
close(fd fd)

open_flags = O_RDONLY, O_WRONLY, O_RDWR, O_APPEND ...
```

Rich Syscall Description

```
# Knows discriminated syscalls:
fcntl$dupfd(fd fd, cmd const[F_DUPFD], arg fd) fd
fcntl$getownex(fd fd, cmd const[F_GETOWN_EX],
               arg ptr[out, f_owner_ex])

# Knows layout of structs:
f_owner_ex {
   type     flags[f_owner_type, int32]
   pid      pid
}
# Has unions:
tun_buffer [
   pi tun_pi
   hdrvirtio_net_hdr
] [varlen]
```

Resources

```
resource fd_bpf_map[fd]
resource fd_bpf_prog[fd]

bpf$MAP_CREATE(cmd const[BPF_MAP_CREATE], ...) fd_bpf_map

bpf_map_lookup_arg {
    map    fd_bpf_map
    key    buffer[in]
    val    buffer[out]
}
```

Programs

The description allows to generate and mutate "programs" in the following form:

```
mmap(&(0x7f0000000000), (0x1000), 0x3, 0x32, -1, 0)
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
read(r0, &(0x7f0000000000), 42)
close(r0)
```

Algorithm

0. Start with empty corpus of programs.

1. Generate a new program, or choose an existing program from corpus and mutate it (know argument types!).

2. Interpret the program, collect coverage from every syscall independently.

3. If a syscall covers code that wasn't covered by this syscall previously, minimize program and add to corpus.

4. Goto 1.

Threaded Execution Mode

Can "skip" over blocking calls:

```
pipe(&(0x7f0000000000)={<r0=>0x0, <r1=>0x0})

read(r0, …)

write(r1, …)
```

Collider: clash adjacent syscalls. Finds lots of data races.

What exactly to collide? How hard?

External Stimulus

Systems calls and external stimulus in the same program:

```
listen(r0)
emit_ethernet(syn)
emit_ethernet(ack)
r1 = accept(r0)
emit_ethernet(data)
read(r1)
emit_ethernet(rst)
```

Problems:

- where to hook? (need coverage)
- reproducible, non-interfering programs
- dynamic cookies

Smarter Program Mutation

Have some prioritization:

program works with tcp sockets -> should add more call that work with tcp sockets

Can we do it smarter having:

- knowledge about resources
- coverage
- syscall return values

?

Upstream Syscall Descriptions

Currently 1200+ syscalls: AF_*, kvm, bpf, tty, sound, video, ...

Problems:

- Small group of people can't describe them all
- Sometime requires domain knowledge
- New syscalls are being added

Two locations proposed:

- include/uapi/
- Documentation/

Thanks!

Q&A

Dmitry Vyukov, dvyukov@