Fuzzing the OpenBSD Kernel

Part 1/N

Anton Lindqvist <anton@openbsd.org>

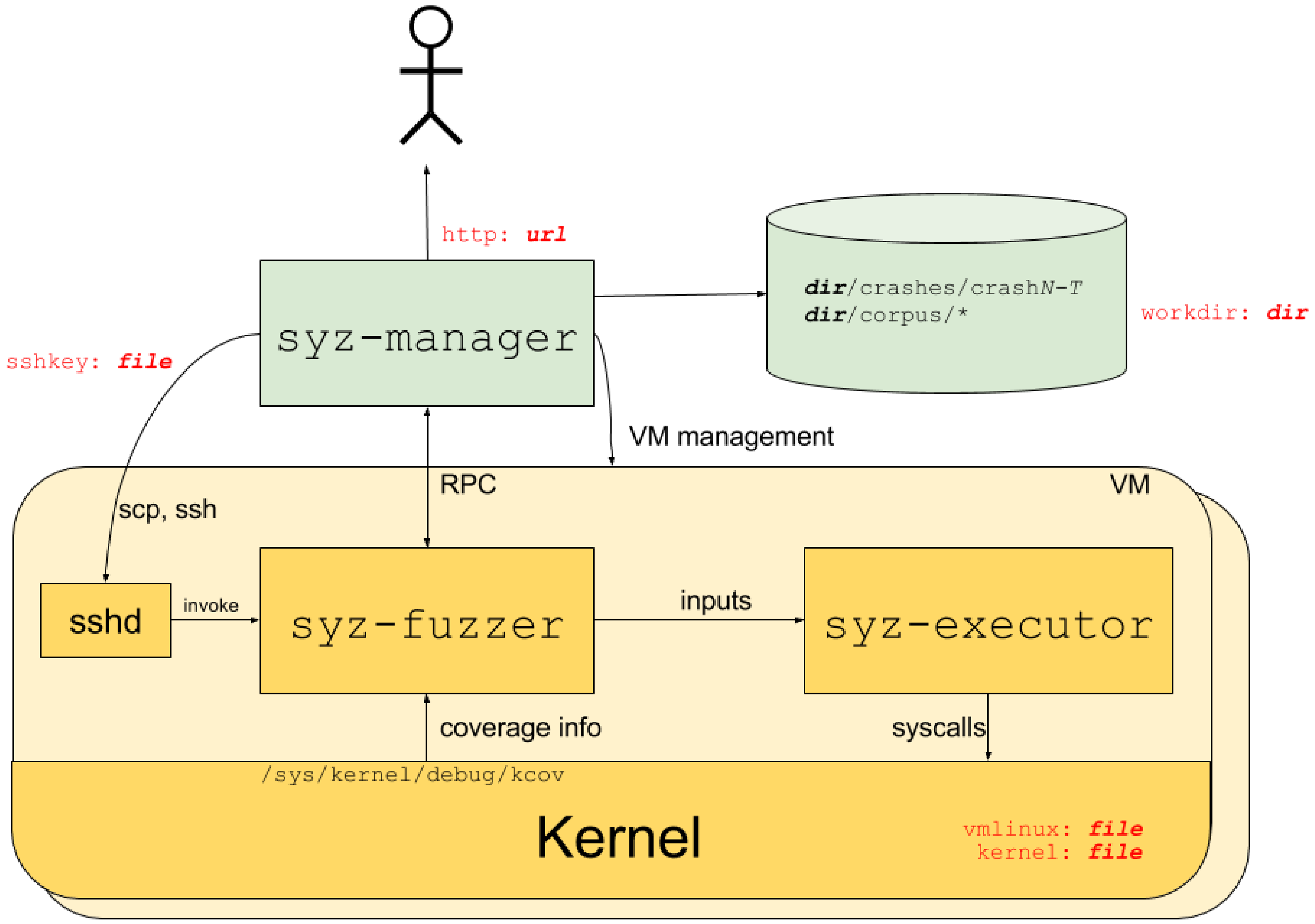
Introduction

- Fuzzing the OpenBSD kernel using the syzkaller kernel fuzzer.
- Heard about first on the BSD Now podcast back in April 2018.
- Ongoing effort, hence 1/N in the title.
- My ambition is to turn this into a recurring topic for future meetups.
- Today, I'll focus on some background and the current state.

syzkaller

- Unsupervised, **coverage-guided** kernel fuzzer.
- Published under Google's account on GitHub but not an official Google product (Apache-2.0 licensed).
- Total of 3200 crashes found in Linux, Android, Chrome OS and other internal kernels.

syzkaller overview



Syscall Descriptions

- Declarative description of syscalls:

```
open(file ptr[in, filename], flags flags[open_flags], mode flags[open_mode]) fd
```

- 225 syscalls supported so far.
- Far from exhaustive since every ioctl(2) command needs a separate description:

```
ioctl$TIOCSETA(fd fd_tty, cmd const[TIOCSETA], arg ptr[in, termios])
```

Syscall Programs

- Descriptions are used to generate and mutate programs:

  ```
  r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
  read(r0 , &(0x7f0000000000), 42)
  close(r0)
  ```

- Novelty arises from the possiblility to test the interaction between different syscalls.
- All generated programs are not equally interesting.
- Programs are categorized based on the heuristic:

  A program is considered interesting if it causes a new code path in the kernel to be executed.

  More on this later...

- An interesting program is further mutated in the hope of continued code path discovery.

syz-prog2c(1)

- Generated programs can be turned into C programs:

```
uint64_t r[1] = {0xffffffffffffffff};

int main(void)
{
    syscall(SYS_mmap, 0x20000000, 0x1000000, 3, 0x1012, -1, 0);
    long res = 0;
    memcpy((void*)0x20000000, "./file0", 8);
    res = syscall(SYS_open, 0x20000000, 0, 0x10);
    if (res != -1)
        r[0] = res;
    syscall(SYS_read, r[0], 0x20000000, 0);
    syscall(SYS_close, r[0]);
    return 0;
}
```

syzbot

- Continous fuzzing of unreleased kernels.
- Can even bisect to find the commit that introduces a regression.
- OpenBSD is not quite there yet...

kcov(4)

- A driver for tracking kernel code coverage.
- Enabled on a per thread basis.
- The kernel program counter is tracked during syscalls made by the same thread.
- Not a strict requirement for syzkaller but improves its ability to generate interesting programs.

kcov(4) - implementation

- Not enabled by default, requires one to compile a custom kernel.
- Limited to architectures using Clang due to usage of the `-fsanitize-coverage=trace-pc` option.
- Newer versions of GCC does support the same option.
- The option will insert calls to a user-supplied function along every line in the original source code (sort of):

```
-fno-sanitize-coverage=trace-pc

int max(int x, int y) {
    if (x > y) {
        return x;
    }
    return y;
}
```

```
-fsanitize-coverage=trace-pc

int max(int x, int y) {
    __sanitizer_cov_trace_pc();
    if (x > y) {
        __sanitizer_cov_trace_pc();
        return x;
    }
    __sanitizer_cov_trace_pc();
    return y;
}
```

Found bugs on OpenBSD

- poll: execution of address 0x0 caused by console redirection
- kqueue: use-after-free in kqueue_close()
- unveil: invalid call to VOP_UNLOCK()
- open: NULL pointer dereference while operating on cloned device
- mprotect: incorrect bounds check in uvm_map_protect()
- fchown: NULL pointer dereference while operating on cloned device
- recvmsg: double free of mbuf
- ftruncate: NULL pointer dereference while operating on cloned device
- kqueue: NULL pointer dereference

What about the other BSDs?

- FreeBSD supported by syzkaller, kcov(4) under development.
- NetBSD supported by syzkaller, kcov(4) under development.

Want to help out?

- Write syscall descriptions (most bang for the buck).
- Know Go? Plenty left todo in syzkaller related supporting continous fuzzing.
- Enable kcov(4) support for remaining Clang architectures. Not as important but could be a fun exercise.

Thanks!

- Martin Pieuchot (OpenBSD) for the help during development of kcov(4).
- Visa Hankala (OpenBSD) for reviewing diffs and fixing found panics.
- Mark Kettenis (OpenBSD) for reviewing diffs and fixing found panics.
- Bob Beck (OpenBSD) for fixing found panics.
- Alexander Bluhm (OpenBSD) for reviewing diffs.
- Philip Guenther (OpenBSD) for reviewing diffs.
- Theo de Raadt (OpenBSD) for reviewing diffs.
- Theo Buehler (OpenBSD) for testing panic fixes.
- Klemens Nanni (OpenBSD) for testing my syzkaller diffs.
- Ingo Schwarze (OpenBSD) for inviting me back in May 2017.
- Dmitry Vyukov (syzkaller) for reviewing diffs.

Questions?