Linux Kernel Fuzzing in Practice

Alexander Popov

Positive Technologies

ISPRAS Open Conference, December 5–6, 2019

$ whoami

- Alexander Popov
- Linux kernel developer
- Security researcher at POSITIVE TECHNOLOGIES
- Speaker at: Linux Security Summit, Still Hacking Anyway, Open Source Summit, Positive Hack Days, etc

Agenda

- What is fuzzing
- About syzkaller (my favorite tool)
- Tales from my fuzzing experience
- Pitfalls: what makes your fuzzing efforts fall short

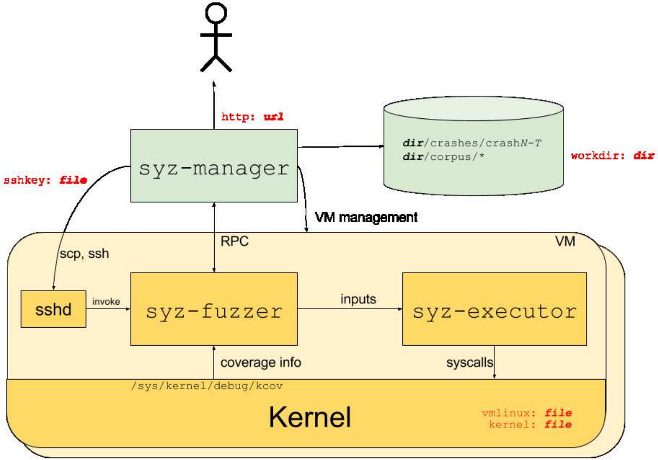
Fuzzing

- **Fuzzing** is aimed at finding bugs by providing random inputs to programs
- Fuzz testing history starts in the 1980s (fuzzing of command-line utilities)
- Earliest syscall fuzzer – Tsys for System V (around 1991)
- Linux kernel fuzzers:
  - **Trinity** syscall fuzzer
  - **perf_fuzzer** for `perf_event_open()`
  - **syzkaller** – a coverage-guided kernel fuzzer (my favorite project)

What Empowers Fuzzers [1]: Code Coverage Feedback

- A fuzzer is more effective if it achieves a higher degree of **code coverage**
- The tested binary should be instrumented to provide coverage information
- Fuzzer uses this info as feedback to choose interesting inputs

Architecture of syzkaller



https://raw.githubusercontent.com/google/syzkaller/master/docs/process_structure.png

What Empowers Fuzzers [2]: Grammar Knowledge

- A good fuzzer should contain **knowledge of the target API**
- syzkaller has the syscall descriptions in /sys/linux/
- Description example:

```
resource fd_rfkill[fd]
openat$rfkill(fd const[AT_FDCWD], file ptr[in, string["/dev/rfkill"]],
              flags flags[open_flags], mode const[0]) fd_rfkill
write$rfkill(fd fd_rfkill, data ptr[in, rfkill_event], len bytesize[data])
read$rfkill(fd fd_rfkill, data ptr[out, rfkill_event], len bytesize[data])
ioctl$RFKILL_IOCTL_NOINPUT(fd fd_rfkill, cmd const[RFKILL_IOCTL_NOINPUT])
rfkill_event {
      idx     int32
      type    int8[0:NUM_RFKILL_TYPES]
      op      int8[0:RFKILL_OP_CHANGE_ALL]
      soft    int8[0:1]
      hard    int8[0:1]
} [packed] dfd
```

What Empowers Fuzzers [3]: Bug Detection Mechanisms

- Additional bug detection and sanitizers spot errors during fuzzing
- Bug detection mechanisms for the Linux kernel:
  - KASAN, UBSAN, KMSAN, KTSAN
  - HARDENED_USERCOPY, REFCOUNT_FULL, DEBUG_LIST
  - lockup detectors
  - etc
- For the mapping to vulnerability types see the Linux Kernel Defence Map: *https://github.com/a13xp0p0v/linux-kernel-defence-map*

Please Credit Researchers

Fuzzing OS kernel does **NOT** give you vulnerabilities or exploits.

It gives you **crashes**, which are:

- **not** always meaningful,
- **not** always security-relevant,
- **not** always reproducible,
- **not** unique if you didn't do any tuning for your fuzzing.

It's a **researcher** who finds, exploits, and fixes the bug!

Tale 1:

CVE-2017-2636

About CVE-2017-2636

- LPE in the Linux kernel introduced in 2009
- Bug type: race condition in `drivers/tty/n_hdlc.c`
- All major distros were affected (`CONFIG_N_HDLC=m`)
- Exploit analysis:

  https://a13xp0p0v.github.io/2017/03/24/CVE-2017-2636.html

Nice! But How?

- Google is fuzzing the Linux kernel very intensively
- But why was it that I found it?

    1. I built the kernel with Ubuntu config
    2. I baked the kernel modules into the rootfs image
    3. The vulnerable module is automatically loaded if the N_HDLC line discipline is set for a pseudoterminal

- Moreover, syzkaller managed to create a C repro for this crash

A Lucky Experiment?

Yes, absolutely!

Tale 2:

Fuzzing works if it doesn't

If the Fuzzer Doesn't Work...

- A lot of soft lockups, RCU stalls, task hangs, and deadlocks in my syzkaller dashboard
- None of them are reproducible
- It looks like the fuzzer is completely broken
- Two days of debugging revealed that...

If the Fuzzer Doesn't Work... Then It Works!

- syzkaller abuses ION allocator and then itself suffers, eh?
- **No!** ION allocator doesn't respect any memory consumption restrictions for a process. **That's bad!**
- Discussion on syzkaller github page: *https://github.com/google/syzkaller/issues/1267*
- Discussion on LKML: *https://lkml.org/lkml/2019/7/17/507*

Tale 3:

Fuzzing works if it doesn't

Part II

If the Fuzzer Doesn't Work Well...

- No interesting crashes for several weeks
- Lost connection to VMs from time to time
- Nothing suspicious for me in syzkaller dashboard
- But one fine morning I...

If the Fuzzer Doesn't Work Well... Then It Works Great!

- But one fine morning I logged in to the fuzzing machine via GUI
- And I saw the alert from gnome-abrt...
- ...that QEMU has crashed. Oh nice!

QEMU Bug

- One week of research and I had a stable reproducer
- One more week of research and I created a fix
- QEMU has a wrong assertion that DMA transfers handled in `ide_dma_cb()` should be a multiple of 512 (the size of a sector)
- So the guest VM can crash QEMU with a weird ATA command :)

Not All Bugs are Treated Well

- I did responsible disclosure to QEMU security team
- But they say that it's not a security issue
- So I posted PoC and fixing patch in the public ML: *https://lists.nongnu.org/archive/html/qemu-devel/2019-07/msg01651.html*
- But maintainers didn't apply my fix because all that code should be redesigned
- No actions for 4 months, so I've started working on it myself: *https://www.mail-archive.com/qemu-devel@nongnu.org/msg662225.html*

Tale 4:

Bug collider

Promising Crash

- I decided to fuzz the Linux kernel compat syscalls
- Later my syzkaller instance got an interesting crash
- It had a stable reproducer, nice!
- It only required access to floppy drives, not root privileges
- I started the investigation

The Bug

Just look at this code snippet in `drivers/block/floppy.c`

```
static int compat_getdrvstat(int drive, bool poll,
                struct compat_floppy_drive_struct __user *arg)
{
    struct compat_floppy_drive_struct v;

    memset(&v, 0, sizeof(struct compat_floppy_drive_struct));
...
    if (copy_from_user(arg, &v, sizeof(struct compat_floppy_drive_struct)))
        return -EFAULT;
...
}
```

The Crash on x86_64

It causes memset() of the userspace memory from the kernelspace:

1. access_ok() for the copy_from_user() source (2nd parameter) fails
2. copy_from_user() then tries to erase the copy destination (1st parameter)
3. But the destination is in the userspace instead of kernelspace :-)
4. So we have a kernel crash:

```
BUG: unable to handle page fault for address: 0000000041414242
#PF: supervisor write access in kernel mode
#PF: error_code(0x0002) - not-present page
```

Bug Collision

- I used static analysis tools Semmle QL and Coccinelle to find similar bugs (it's another story)
- I was ready to send patches to security@kernel.org...
- A friend of mine noticed that he saw similar patches on LKML
- Yes, Jann Horn from P0 reported them in March 2019
- He used sparse tool to find them

A Lost Patch

- Why does fuzzing still hit these bugs?
- Because the patch was lost!
- I've reported that to the maintainers
- Jens Axboe will apply Jann's lost patch for Linux kernel **v5.4**
- The full story: *https://a13xp0p0v.github.io/2019/08/10/cfu.html*

Tale 5:

CVE-2019-18683

CVE-2019-18683

- 5-year old race conditions in the vivid driver (V4L2 subsystem)
- I created a PoC local privilege escalation exploit (LPE)
- Full disclosure:
  https://www.openwall.com/lists/oss-security/2019/11/02/1
- **Fuzzing tricks**:
  - I modified the kernel, not the fuzzer
  - That allowed the fuzzer to get deeper into the kernel code and hit the bug

Closing Thoughts

- Fuzzing is just like **gold mining**:
  - A lot of people are doing it
  - You need good hardware
  - You need to keep an eye on the process all the time
  - You need to invent special tricks to find something unique
  - You have no guarantees of success
- That kind of research is **exhausting**...
- But it is so **exciting** when you finally find something!

Thanks! Questions?

alex.popov@linux.com
@a13xp0p0v

http://blog.ptsecurity.com/
@ptsecurity