Linux kernel Fuzzing

Andrey Konovalov <andreyknvl@gmail.com>

ZeroNights 2016, Hardware Village
November 17th 2016

Agenda

- Userspace Sanitizers (ASan, TSan, MSan, UBSan)
- Kernel Sanitizers (KASAN, KTSAN, KMSAN)
- Kernel syscall fuzzers (Trinity, syzkaller)
- USB fuzzing: FaceDancer21, vUSBf

Userspace Sanitizers

- AddressSanitizer (ASan)
  - detects use-after-free and out-of-bounds

- ThreadSanitizer (TSan)
  - detects data races and deadlocks

- MemorySanitizer (MSan)
  - detects uninitialized memory uses

- UndefinedBehaviorSanitizer (UBSan)
  - detects undefined behaviors in C/C++

Kernel Sanitizers

- KASAN (use-after-free and out-of-bounds)
  - CONFIG_KASAN available upstream since 4.0
- KTSAN (data-races and deadlocks)
  - prototype available at https://github.com/google/ktsan
- KMSAN (uninitialized-memory-use)
  - in early prototype stage

KernelAddressSanitizer (KASAN)

Two parts

- Compiler module
  - Instruments memory accesses

- Runtime part
  - Bug detection algorithm

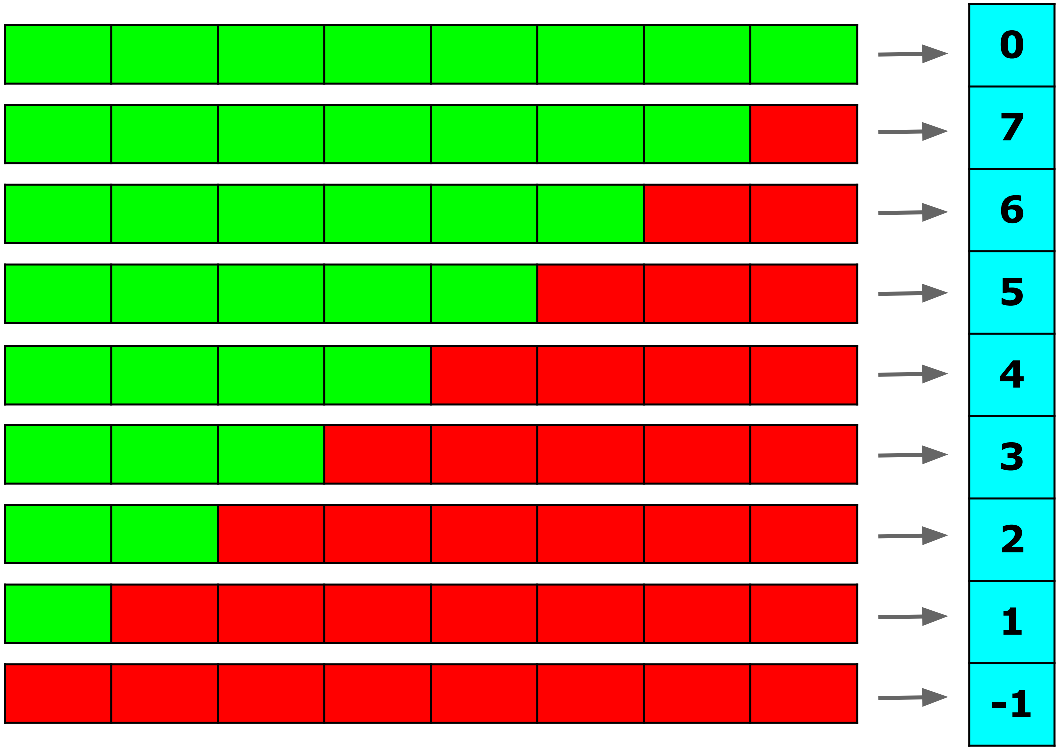
Shadow byte

Any aligned 8 bytes may have 9 states:
N good bytes and 8 - N bad (0 <= N <= 8)

🟩 Good byte

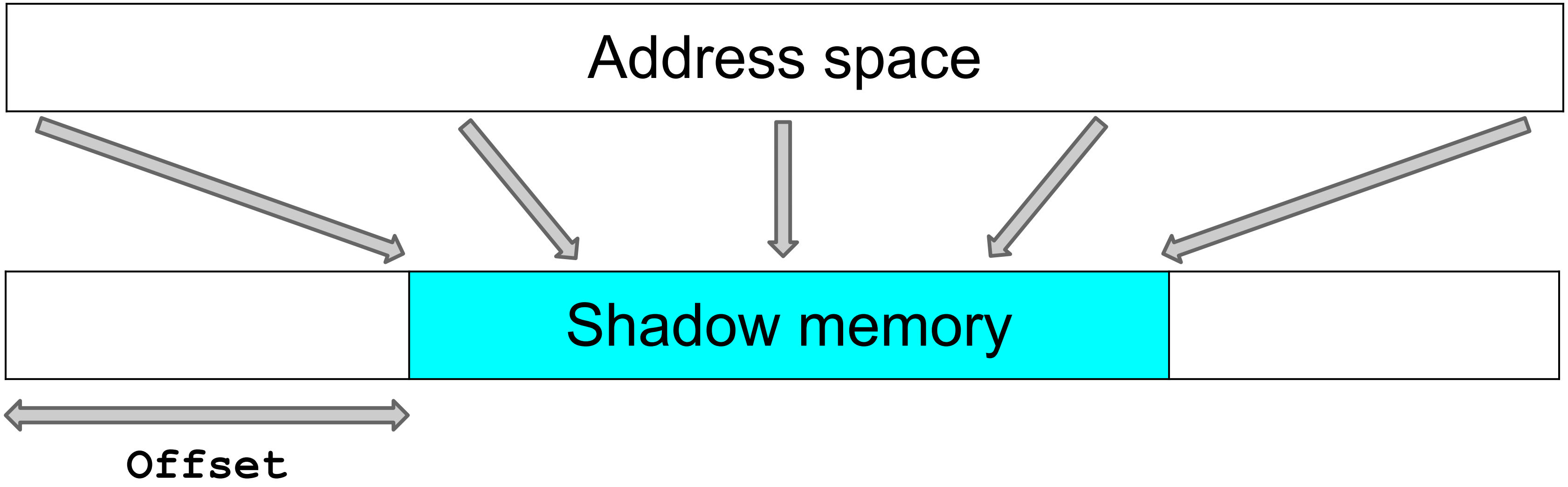
🟥 Bad byte

🟦 Shadow value



Memory mapping

```
Shadow = (Addr >> 3) + Offset
```



x86-64 memory layout

0000000000000000 - 00007fffffffffff (=47 bits) user space, different per mm

hole caused by [48:63] sign extension

ffff800000000000 - ffff87ffffffffff (=43 bits) guard hole, reserved for hypervisor

ffff880000000000 - ffffc7ffffffffff (=64 TB) direct mapping of all phys. memory

ffffc80000000000 - ffffc8ffffffffff (=40 bits) ho le

ffffc90000000000 - ffffe8ffffffffff (=45 bits) vmalloc/ioremap space

ffffe90000000000 - ffffe9ffffffffff (=40 bits) hole

ffffea0000000000 - ffffeaffffffffff (=40 bits) virtual memory map (1TB)

... unused hole ...

ffffec0000000000 - fffffc0000000000 (=44 bits) kasan shadow memory (16TB)

... unused hole ...

ffffff0000000000 - ffffff7fffffffff (=39 bits) %esp fixup stacks

... unused hole ...

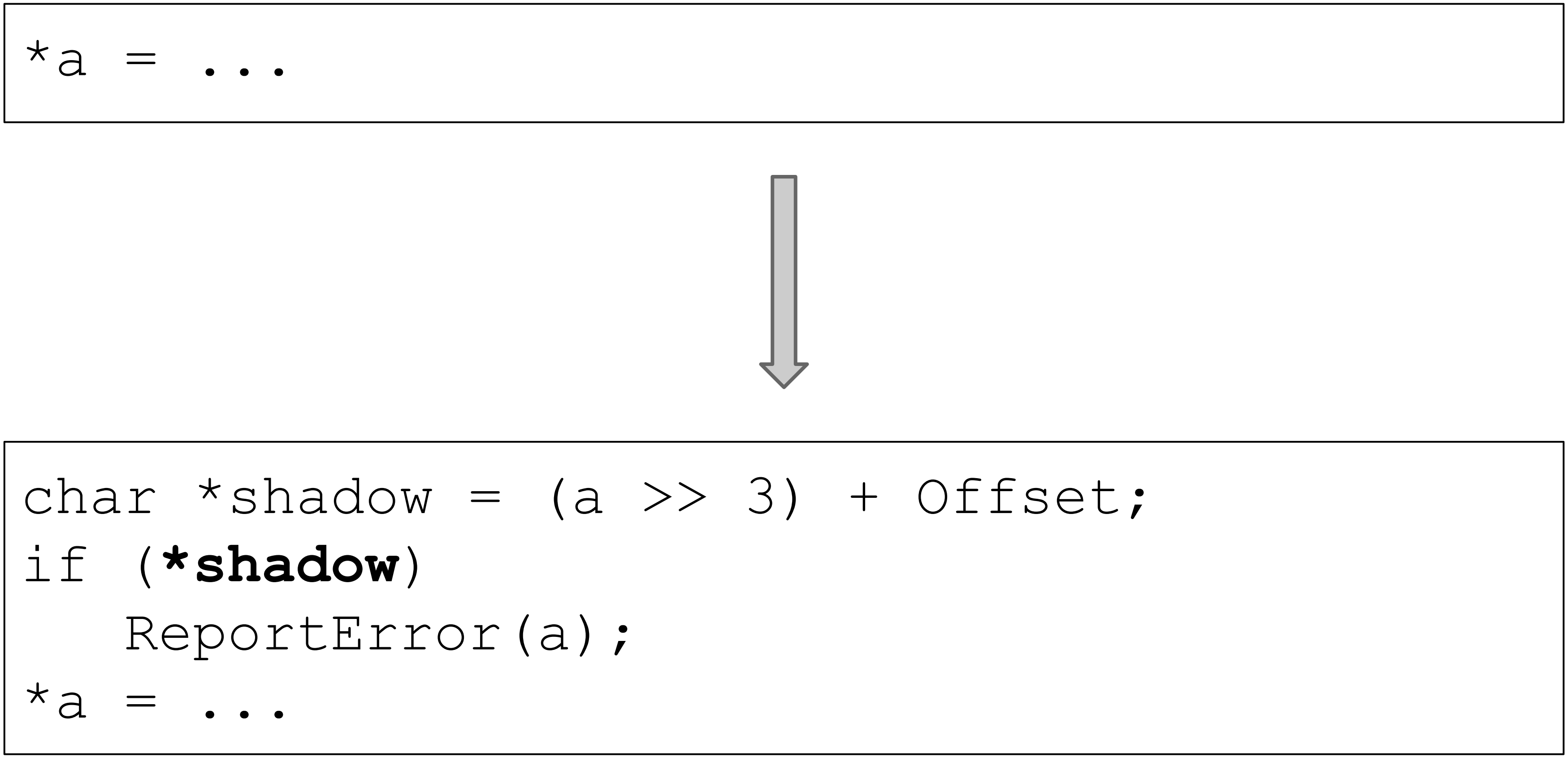
ffffffff80000000 - ffffffffa0000000 (=512 MB) kernel text mapping, from phys 0

ffffffffa0000000 - ffffffffff5fffff (=1525 MB) module mapping space

ffffffffff600000 - ffffffffffdfffff (=8 MB) vsyscalls

ffffffffffe00000 - ffffffffffffffff (=2 MB) unused hole

Compiler instrumentation: 8 byte access



```
*a = ...
```

```
char *shadow = (a >> 3) + Offset;
if (*shadow)
   ReportError(a);
*a = ...
```

Instrumentation: N byte access (N = 1, 2, 4)

```
*a = ...
```

```
char *shadow = (a >> 3) + Offset;
if (*shadow && *shadow < (a & 7) + N)
   ReportError(a);
*a = ...
```

Run-time module

- Maps shadow memory
- Adds redzones around slab objects
- Poisons/unpoisons shadow on kfree/kmalloc
- Ensures delayed reuse of slab objects
- Poisons global redzones on startup
- Collects stack traces for kmalloc/kfree
- Prints error reports

Fuzzing the kernel

- Kernel inputs:
  - system calls
  - network
  - USB
  - bluetooth
  - ....

Kernel system call fuzzers

- Trinity (https://github.com/kernelslacker/trinity)

- syzkaller (https://github.com/google/syzkaller)

syzkaller

Existing system call fuzzers

Trinity in essence:

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

Knows argument types, so more like:

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

- Tend to find shallow bugs
- Frequently no reproducers

Coverage-guided fuzzing

(Think AFL or libFuzzer)

```
void TestOneInput(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

Coverage for the Linux kernel

- Available upstream with CONFIG_KCOV
- GCC pass that inserts a function call into every basic block
- kernel debugfs extension that collects and exposes coverage per-thread

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

⇓

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

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
    hdr virtio_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

1. Start with empty corpus of programs

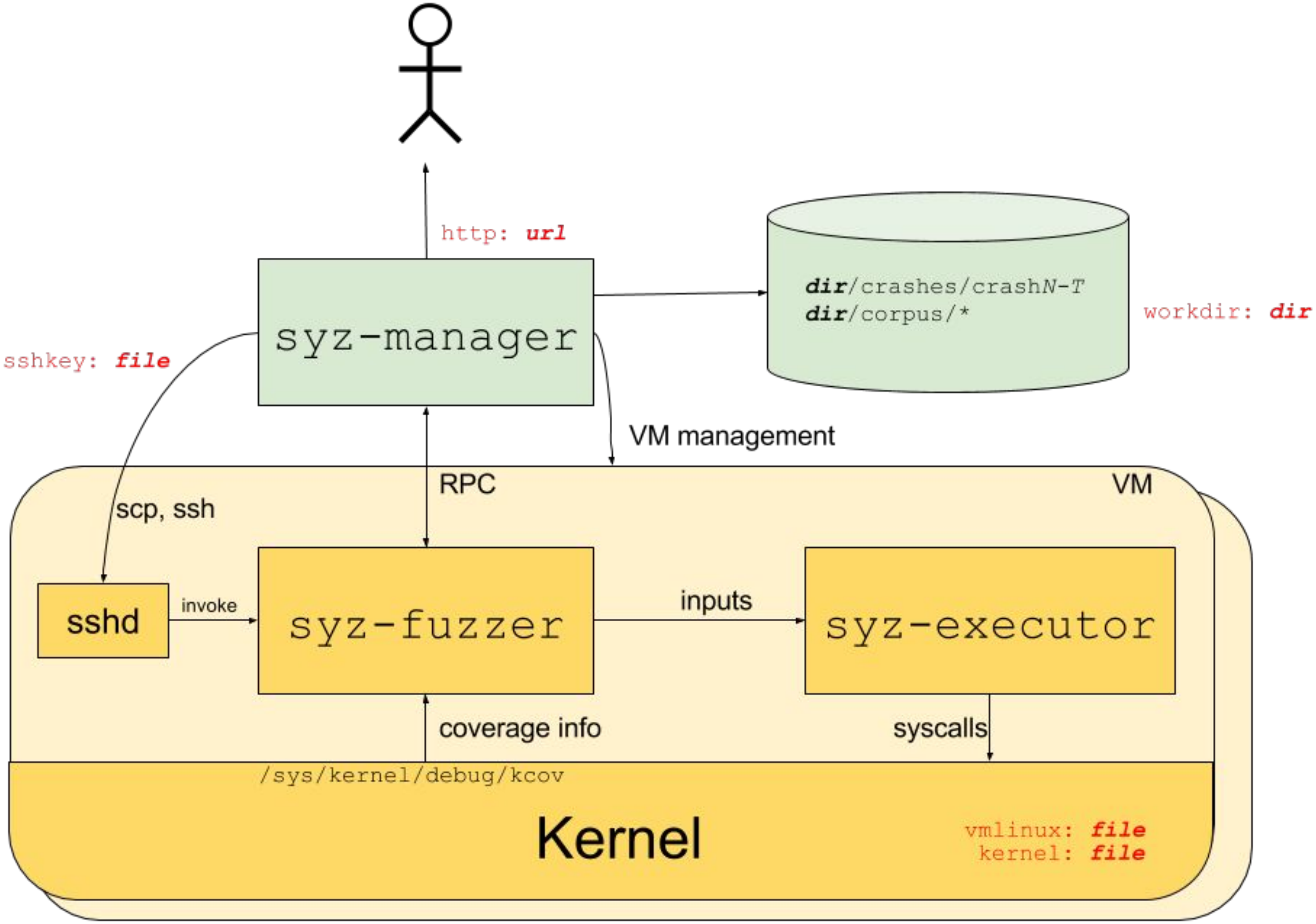
2. Generate a new program, or choose an existing program from corpus and mutate it (know argument types!)

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

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

5. Goto 1

Overview


http: url
syz-manager
dir/crashes/crashN-T
dir/corpus/*
workdir: dir
sshkey: file
VM management
RPC
VM
scp, ssh
sshd
invoke
syz-fuzzer
inputs
syz-executor
coverage info
syscalls
/sys/kernel/debug/kcov
Kernel
vmlinux: file
kernel: file

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)
```

Work in progress; also applicable to USB, ...

Other Linux kernel fuzzers

- https://github.com/oracle/kernel-fuzzing
- https://github.com/nccgroup/TriforceLinuxSyscallFuzzer
- http://web.eece.maine.edu/~vweaver/projects/perf_events/fuzzer/
- https://github.com/schumilo/vUSBf

USB fuzzing

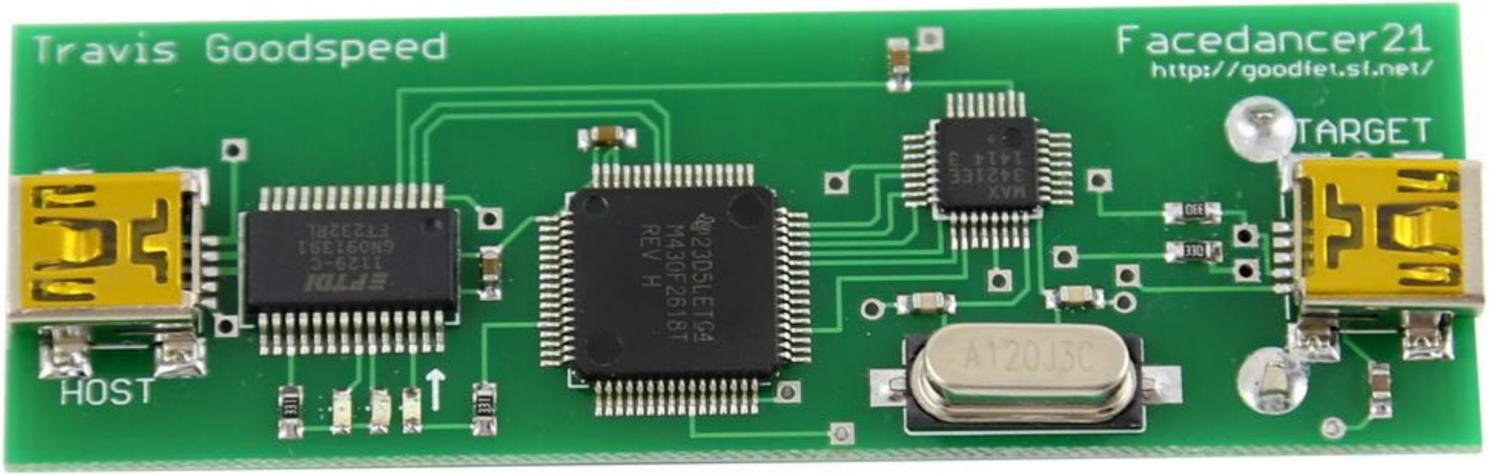
- Hardware: FaceDancer21

- In VM: vUSBf

(not BadUSB)

FaceDancer21

- “The purpose of this board is to allow USB devices to be written in host-side Python, so that one workstation can fuzz-test the USB device drivers of another host”
- http://goodfet.sourceforge.net/hardware/facedancer21/



FaceDancer21

- https://github.com/travisgoodspeed/goodfet
- https://github.com/nccgroup/umap
- https://github.com/nccgroup/umap2

vUSBf

- Virtual USB fuzzer

- QEMU + usbredir

- https://github.com/schumilo/vUSBf

CVE-2016-2384

- Double-free in usb-midi driver
- Found with vUSBf
- Confirmed and exploited with FaceDancer21
- https://xairy.github.io/blog/2016/cve-2016-2384

Questions?

https://github.com/google/kasan/wiki

https://github.com/google/ktsan/wiki

https://github.com/google/syzkaller

Andrey Konovalov, andreyknvl@gmail.com

kasan-dev@googlegroups.com

syzkaller@googlegroups.com