ubsan, kasan, syzkaller

Florian Westphal

July 2017

4096R/AD5FF600 fw@strlen.de
1C81 1AD5 EA8F 3047 7555
E8EE 5E2F DA6C F260 502D

syn

1. kernel debugging options
2. testing using compiler extensions, e.g. KASAN or UBSAN
3. kernel-fuzzing

SLUB Debugging

- default allocator as of 2.6.23 (october 2007)
- lot of debugging options, enable with `slub_debug` boot parameter
  - Z: Redzoning
  - P: Poisoning
- `/proc/slabinfo`, `linux/tools/vm/slabinfo.c`
- some options are available at runtime:
  `echo 1 > /sys/kernel/slab/$name/trace`
- only of limited use for production systems (dmesg spew)

Fault-Injection Framework

`CONFIG_FAULT_INJECT=y`

- `FAILSLAB`, `FAIL_MAKE_REQUEST`, ...
- configuration via debugfs:

```
ls /sys/kernel/debug/fail*
/sys/kernel/debug/failslab:
cache-filter ignore-gfp-wait interval probability
space task-filter times verbose
```

- to limit to single process:
  `echo 1 > /proc/$pid/make-it-fail`
- Boot option

Notifier Error Injection

`CONFIG_NOTIFIER_ERROR_INJECT=y`

- notifier: kernel callbacks invoked when certain events occur
  - CPU x is offlined
  - new network interface added/removed (can also be vlan, bridge, etc)
  - mtu, status, feature changes (`ethtool -K`)
- some notifiers can veto the change request – "`return NOTIFY_BAD`"

other compile timer kernel debug options

- atomic-sleep
- lockdep
- soft/hard lockup detector
- linked-list debugging

AddressSanitizer

- memory(address) handling error detection
- gcc 4.8

```
int main(void) {
   char xx[100];
   xx[100] = 0;
   return 0;
}
gcc -fsanitize=address x.c && ./a.out
ERROR: AddressSanitizer: stack-buffer-overflow
WRITE of size 1 at ...
    #0 0x40090d in main (/tmp/a.out+0x40090d)
    #1 0x7fc6c0db162f in __libc_start_main
Address 0x7ffcd7a08d24 is located in stack of thread ...
```

recent gcc: `ASAN_OPTIONS='help=1' ./a.out`

KernelAddressSanitizer

- `CONFIG_KASAN=y`
  - use-after-free
  - out-of-bounds accesses
- $\frac{1}{8}$ kernel RAM used for shadowing
- Shadowbyte 0 $\rightarrow$ Word is allocated
- Shadowbyte 1 $\rightarrow$ 1 Byte only (n: n bytes)
- Shadowbyte < 0: locked (free'd, redzone, etc)
- compiler adds `__asan_load/_store` calls for all memory accesses

KernelAddressSanitizer (cont.)

- faster than kmemcheck
- no detection of uninitialized reads
- slower than SLUB debug, but more powerful
  - OOB-reads
  - OOB-write: instant, slub can only detect it at free time

Undefined Behaviour Sanitizer (UbSan)

gcc: `-fsanitize=undefined`, e.g.:

- signed-integer-overflow
- enum (runtime error if it contains invalid/out-of-enum range value)
- shifts ('1<<32'), etc.

```
int main(void) { return INT_MIN / -1; }
gcc -fsanitize=undefined x.c && ./a.out
x.c:9:17: runtime error: division of -2147483648 by -1
cannot be represented in type 'int'
```

Fuzzer: Trinity, syzkaller

simplie idea: pass random input (to program, function, etc)

- Kernel has numerous places where it processes data
  - syscalls: setsockopt, ioctl, etc.
  - data buffers passed in (netlink, raw sockets ...)
  - network traffic, external media, ...

```
syscall(random(), random(), random(), random());
```

... doesn't really work

Trinity

syscall fuzzer, random syscalls with random arguments, but ...

- not everything is random
- creates various file descriptors (files, pipes, sockets, etc)
- list with syscalls and expected arguments (z.B. `accept(fd, ..)`)
- list with flags for syscalls (`send, sendto, ...`)
- permutation of arguments, order, ...

problem: no information over code coverage (call chain depth, branches taken)

afl

american fuzzy lop - compile-time instrumentation fuzzer

- `afl-{clang,gcc,g++}`
- needs valide inputs (place in `afl/in`)
- `afl-fuzz -i afl/in -o afl/out program [args]`
- assumes program reads stdin, else "-f foo"

syzkaller

more recent syscall fuzzer, more like afl
`https://github.com/google/syzkaller`

- distribued, fuzzing occurs in virtual machines
- syscalls and expected arguments are described/declared
- generates random programs and performs mutations on these

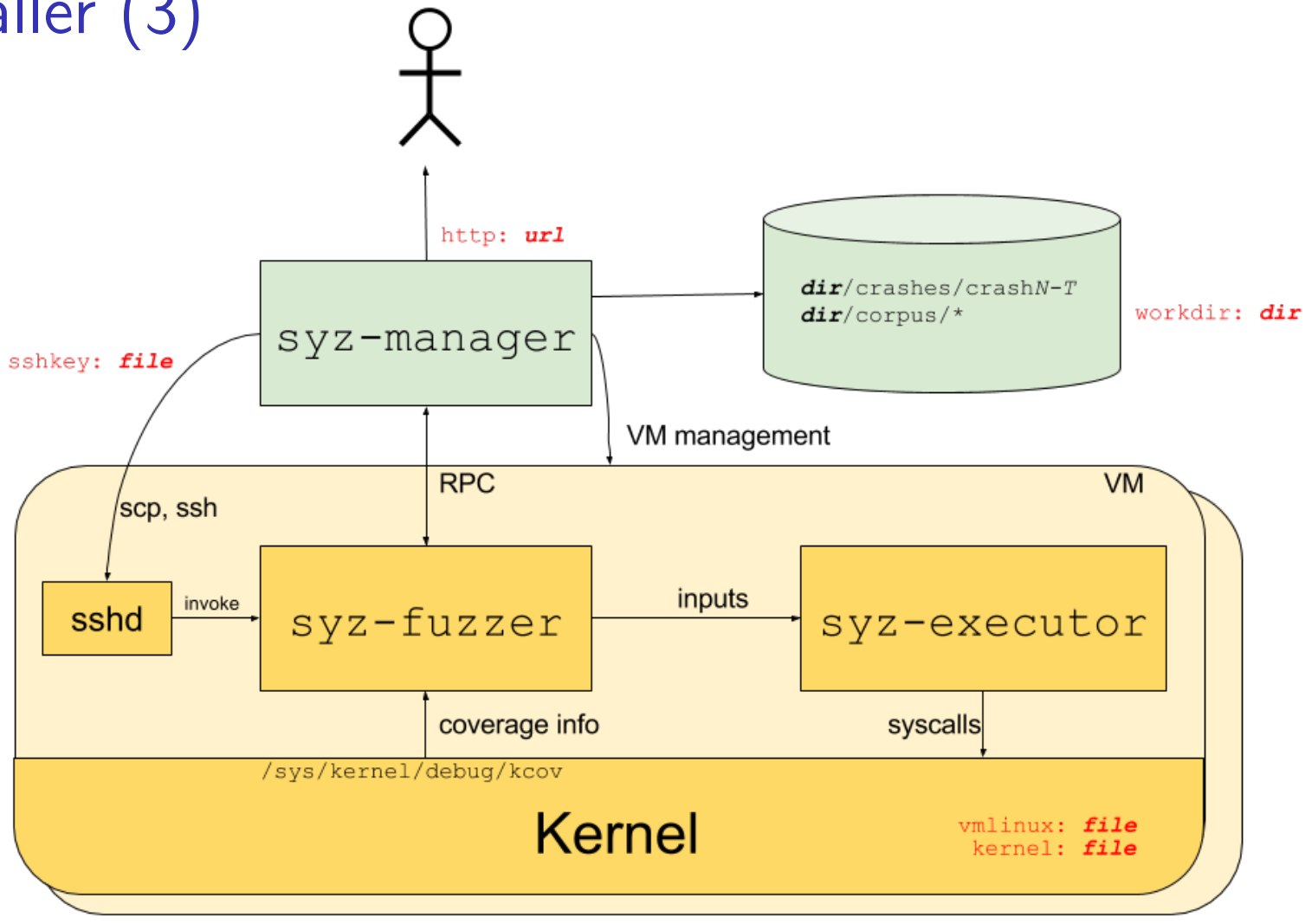
```
open(filename, f flags[open_flags], m flags[open_mode])
close(fd fd)
writev(fd fd, vec ptr[in, array[iovec_in]], vlen len[vec])
open_flags = O_RDONLY, O_WRONLY,
iovec_in {
    addr    buffer[in]
    len     len[addr, intptr]
}
```

syzkaller (2)

prerequisites:

- go and C++ compiler
- C-compiler with coverage support (gcc 6.1 or newer)
- linux kernel starting with 4.6
- QEMU + distro disk image (see `create-image.sh`)
- syzkaller tools
  - `syz-manager`: start/stop/monitoring of VMs
  - `syz-fuzzer`: generates Tests, Mutations, sends inputs that increased coverage to manager
  - `syz-executor`: executes inputs (i.e. syscall sequence)

syzkaller (3)



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

syzkaller (4)

config-file (ex):

```
"http": "localhost:56741",
"workdir": "/home/me/syzkaller/workdir",
"kernel": "/home/me/build/linux/arch/x86/boot/bzImage",
"vmlinux": "/home/me/build/linux/vmlinux",
"image": "/home/me/syzkaller/debian.img",
"sshkey": "/home/me/syzkaller/ssh/id_rsa",
"syzkaller": "/home/me/syzkaller",
"leak": false,
"cmdline": "root=/dev/sda ro console=ttyS0",
"count": 2,
"cpu": 2,
"mem": 2048,
"enable_syscalls": [ .. ]
```

syzkaller (5)

workdir

- autogenerated
- `corpus`:
  - inputs that created new outputs
  - kept around to accelerate new starts
- `crashes`:
  - dmesg oopses/warnings
  - lockups

syzkaller (6)

`corpus/$sha`: autogenerated programs

```
mmap(&(0x7f0000000000)=nil, (0xc000), 0x3, 0x32, 0xffffffff
clock_gettime(0x7, &(0x7f0000003000-0x10)={0x0, 0x0})
mmap(&(0x7f0000002000)=nil, (0x1000), 0x3, 0x32, 0xffffffff
clock_gettime(0x7, &(0x7f0000003000)={0x0, 0x0})
setsockopt(0xffffffffffffffff, 0x0, 0x40, &(0x7f0000004000-
clock_gettime(0x5, &(0x7f0000005000)={0x0, 0x0})
clock_gettime(0x7, &(0x7f0000008000-0xd)={0x0, 0x0})
socket(0x2, 0x3, 0xff)
setsockopt(0xffffffffffffffff, 0x0, 0x81, &(0x7f000000c000-
```

`syz-prog2c` to convert to something more readable

syskaller (7)

1. `bin/syz-manager -config ~/syzkaller/syzkaller.cfg`
2. wait – outputs events to stdout

```
2016/05/04 15:41:10 loaded 3225 programs
2016/05/04 15:41:10 serving http on http://localhost:56741
2016/05/04 15:41:10 serving rpc on tcp://127.0.0.1:41620
2016/05/04 15:44:40 qemu-1: saving crash 'UBSAN: Undefined
2016/05/04 15:49:18 qemu-2: saving crash 'UBSAN: Undefined
2016/05/04 15:50:56 qemu-1: saving crash 'WARNING: CPU: 3 P
2016/05/04 15:51:04 qemu-0: saving crash 'UBSAN: Undefined
2016/05/04 15:52:00 qemu-2: saving crash 'no output' to cra
```

Crashes/Traces in workdir/crashes/

kcov: code coverage for fuzzing

`CONFIG_KCOV=y` (linux.git) gcc flag
`-fsanitize-coverage=trace-pc`

- like Kasan/Ubsan: compiler based
- kernel implements function that gets called for each block
- stores PC in a Log buffer
- calls from irq context are ignored
- userspace needs to turn this on first $\rightarrow$ `ioctl`

kcov: code coverage for fuzzing (cont.)

```
fd = open("/sys/kernel/debug/kcov", O_RDWR);
ioctl(fd, KCOV_INIT_TRACE, COVER_SIZE);
cover = mmap(.., PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
ioctl(fd, KCOV_ENABLE, 0);
read(-1, NULL, 0); /* fuzz */
for (i = 1; i < cover[0]; i++)
    printf("0x%lx\n", cover[i]);
```

→ /sys/kernel/debug/kcov

→ addr2line

```
SyS_read
fs/read_write.c:562
__fdget_pos
...
```

fin

KernelThreadSanitizer (KTSan)

- `fsanitize=thread`:
  - Program execution == sequence of events
  - memory accesses
  - synchronisation (Locks)
  - automata stores events as they occur