syzkaller:
adventures in continuous coverage-guided kernel fuzzing

BlueHat IL 2020

Dmitry Vyukov, dvyukov@

Agenda

- Background
- Implementation
- Results
- Future

Dynamic Tools Team

- Bug detection (user-space/kernel):
  - ASAN
  - MSAN
  - TSAN (C++, Go, Java)
  - KCSAN
  - LSAN
  - UBSAN
- Production hardening:
  - CFI
  - SafeStack
  - ShadowCallStack
  - HWASAN
  - Memory tagging (MTE)
  - GWP-ASan
- Bug provocation:
  - LibFuzzer (C++ [Go, Rust])
  - go-fuzz (Go)
  - syzkaller (kernels)
- Misc:
  - OSS-Fuzz
  - syzbot
  - SanitizerCoverage
  - KCOV
  - DFSAN

KASAN

KASAN: KernelAddressSANitizer

- UAF (use-after-free)
- OOB (out-of-bounds)

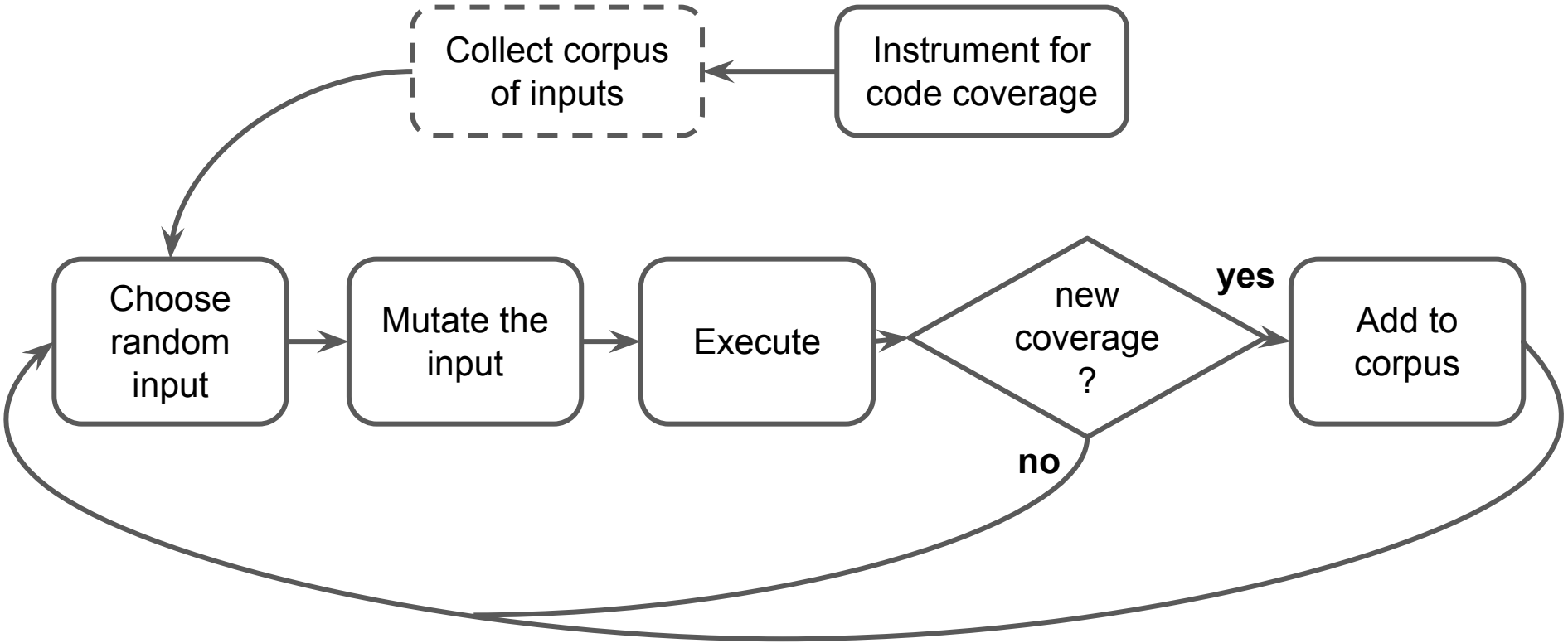
Where are my tests, dude?

Fuzzing

Wikipedia: **Fuzz testing** or fuzzing is a software **testing technique**, often automated or semi-automated, that involves providing **invalid, unexpected, or random data** to the inputs of a computer program.

Code coverage guided fuzzing (AFL, honggfuzz, LibFuzzer, go-fuzz).

Coverage-guided fuzzing



Coverage-guiding in action

```
if (input[0] == 'A')
    if (input[1] == 'B')
        if (input[2] == 'C')
            if (input[3] == 'D')
                input[input[4]] = input[5]; // potential OOB write
```

Requires **"ABCD"** input to crash, **~2^32** guesses to crack when blind.

Coverage-guiding:
Guess **"A"** in ~2^8, add to corpus.
Guess **"AB"** in ~2^8, add to corpus.
Guess **"ABC"** in ~2^8, add to corpus.
Guess **"ABCD"** in ~2^8, add to corpus.

Total: **~2^10** guesses.

See: AFL: Pulling JPEGs out of thin air

Advantages:

- efficient
- reproducers
- regression testing

Existing Kernel Fuzzers

trinity/iknowthis in essence:

```
while (1) syscall(rand(), rand(), rand());
```

Do know argument types, so more like:

```
while (1) syscall(rand(), rand_fd(), rand_addr());
```

Problems:

- shallow bugs
- no reproducers
- no regression testing
- no automation

How do you apply coverage-guided fuzzing to kernel?

Difficulties with kernel

- what is input?
- how do we mutate it?
- isolation/reproducibility
- enormous input space

- blocking syscalls
- provoking races
- parallel fuzzing

- other processes
- background threads
- interrupts
- non-determinism
- flakes from malloc/scheduler

- collapsing kernel
- unsupervised operation
- working-as-intended crashes
- detecting/deduplicating crashes
- work across VMs/boards/phones

Kernel fuzzing is different

1. **Smart** (1000x larger, 1000x slower)

   user-space: 100'000 inputs/sec

   kernel: 1 input/minute

2. **Best-effort** (no definitive answers: new coverage? crashed?)

Inputs

```c
int main() {
    int fd = open("/dev/kvm", O_RDWR);
    ioctl(fd, ...);
    close(fd);
}
```

Structured Fuzzing

Syscall Description Language (syzlang)

```
open(file ptr[in, filename], flags flags[open_flags]) fd
read(fd fd, buf ptr[out, array[int8]], count bytesize[buf])
close(fd fd)

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

Why not C headers?

- FD is not just "`int`"
- What is "`const char*`"? (file system, crypto alg, file name)
- Count of array elems, sizeof things
- struct's with variable-size arrays
- `ioctl(void* ptr)` (¯_(ツ)_/¯)
- `write(void* data)` (thousand lines of C parsing code)
- ...

Resources

Resource - value passed from one syscall to another:

```
resource fd[int32]: -1, AT_FDCWD
open(...) fd
close(fd)

resource fd_cdrom[fd]

open(file ptr[in, string["/dev/cdrom"]], ...) fd_cdrom

ioctl(fd fd_cdrom, cmd const[CDROMPAUSE], ...)
ioctl(fd fd_cdrom, cmd const[CDROMRESUME], ...)
```

Structs

```
foo {
    f1          int32
    f2_len      len[f2, int16]
    f3_len      len[f3, int8]
    f2          array[int8]
    f3          array[bar]
}
```

```
{0x12345678, 0x002, 0x03, [0x0a, 0x0b], [{...},{...},{...}]
```

f1 (4 bytes)

f2_len (2 bytes)

f3_len (1 bytes)

f2 (2*1 bytes)

f3 (3*sizeof(bar) bytes)

Unions

```
foo [
    f1      int32
    f2      array[string]
    f3      ptr[in, array[bar]]
] [varlen]
```

Building mount options

```
mount("uid=123,pid=4567,readahead_blocks=0x200,utf8")
```

```
uid_opt {
     name      stringnoz["uid="]
     val       fmt[dec, uid]
} [packed]

pid_opt {
     name      stringnoz["pid="]
     val       fmt[dec, pid]
} [packed]

blocks_opt {
     name      stringnoz["blocks="]
     val       fmt[hex, flags[nblocks]]
} [packed]

nblocks = 0, 512, 0x8000, 0x4000000
```

```
option [
     uid       uid_opt
     pid       pid_opt
     blocks    blocks_opt
     utf8      stringnoz["utf8"]
] [varlen]

opt_with_comma {
     val       option
     comma     const[',', int8]
} [packed]


mount(opt ptr[in, array[opt_with_comma]])
```

Some other features

- int/const/flags/ranges
- bitfields
- big-endian (`int32be:11[100:200, 3]`)
- fixed-size arrays, range-sized arrays
- structs: packed, aligned, padded
- len/bytesize/bytesize4/bitsize/offsetof
- templates
- void
- ...

[syzlang syntax description]

Programs

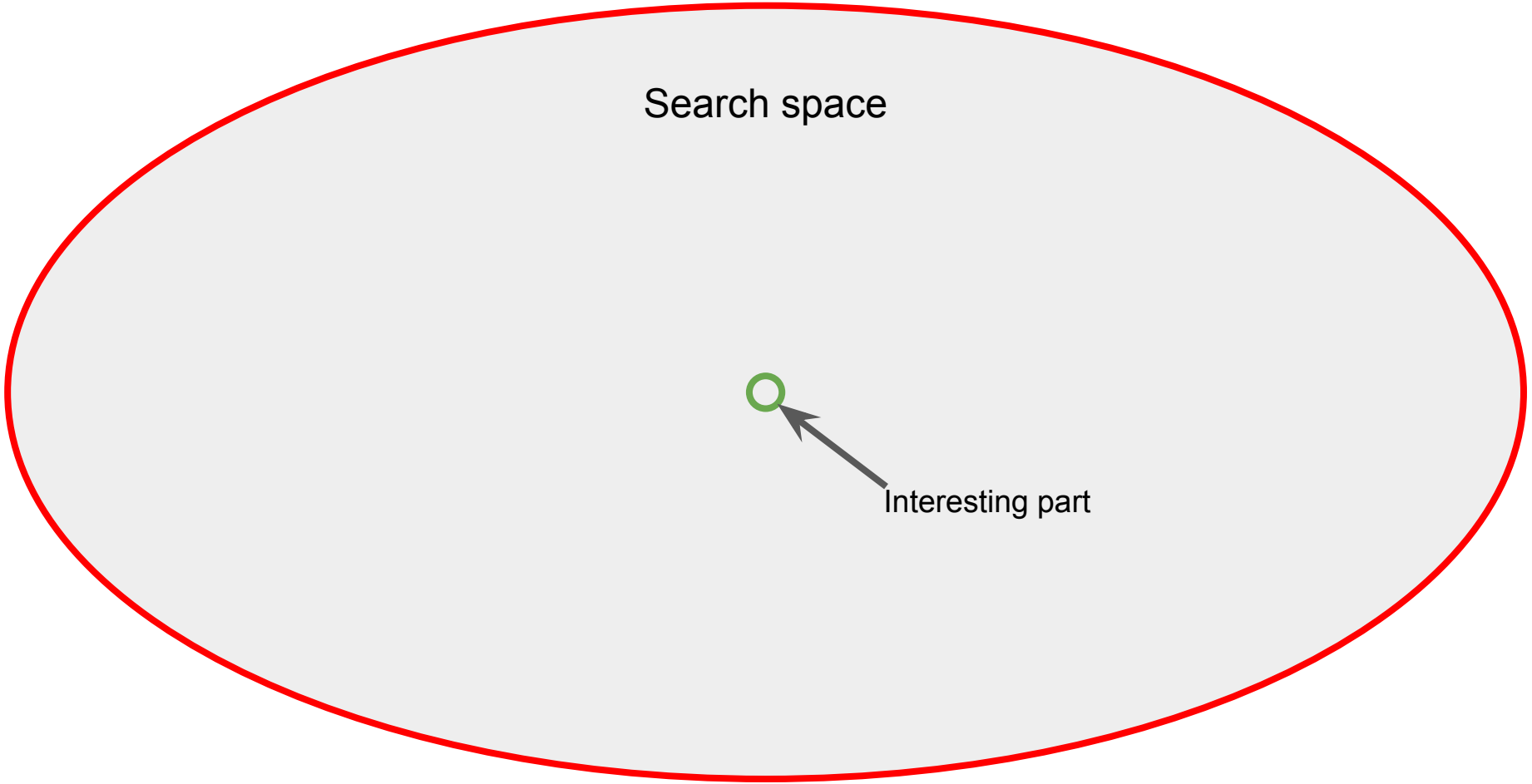
```
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
read(r0, &(0x7f0000000100), 42)
close(r0)
```

- generate
- mutate
- minimize
- analyze
- interpret
- convert to C
- serialize/deserialize

Mutations

- Insert/remove calls
- Change args:
  - resize arrays/buffers
  - change union options
  - flags
  - len/bytesize
  - filename
  - pointers
- Traditional mutation of blobs
  - flip bits, insert/remove bytes, etc
- Splicing of programs
  - based on resources

Prioritization



Prioritization

- Program selection
  - coverage
- Mutation action
  - heuristics
- Syscall selection
  - argument complexity
  - static relation to existing syscalls
  - dynamic relation to existing syscalls
- Argument selection
  - complexity
  - heuristics
- Argument generation
  - resources: frequency of default values
  - flags: how many bits to set
  - ints: frequency of 0

Blocking syscalls

```
pipe(&(0x7f0000000000)={<r0=>0x0, <r1=>0x0})
read(r0, …)       # blocks (pipe is empty)
write(r1, …)      # unblocks read
```

Threaded execution mode:

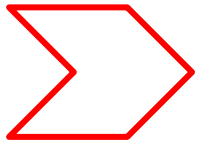
- dispatch syscalls to worker threads
- consider syscall "blocked" after timeout
- continue dispatching other syscalls
- check if previous "blocked" syscalls has returned

Collide mode

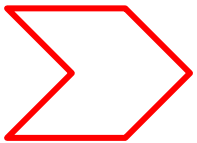
Races!

Consider every second syscall as "blocked" right away.

```
r0 = open("file", O_RDWR)
r1 = open("file", O_RDWR)
read(r0, ...)
write(r1, ...)
```



start at the same time



start at the same time

Fault Injection

kmalloc, page_alloc, futex, IO

Actually meaningful in kernel (must survive!)...

... and there are lots of bugs (use-after-free's, double-free's).

Old fault injection: fail something randomly with X% probability.

Systematic Fault Injection

Primitives:

1. fail N-th site in the current thread
2. check if failure was injected

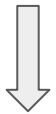
Take syscall with new coverage:

- try to fail **first** site
- if failure was injected, try to fail **second** site
- if failure was injected, try to fail **third** site
- ...
- until failure wasn't injected (tested all error paths)

Coverage (KCOV)

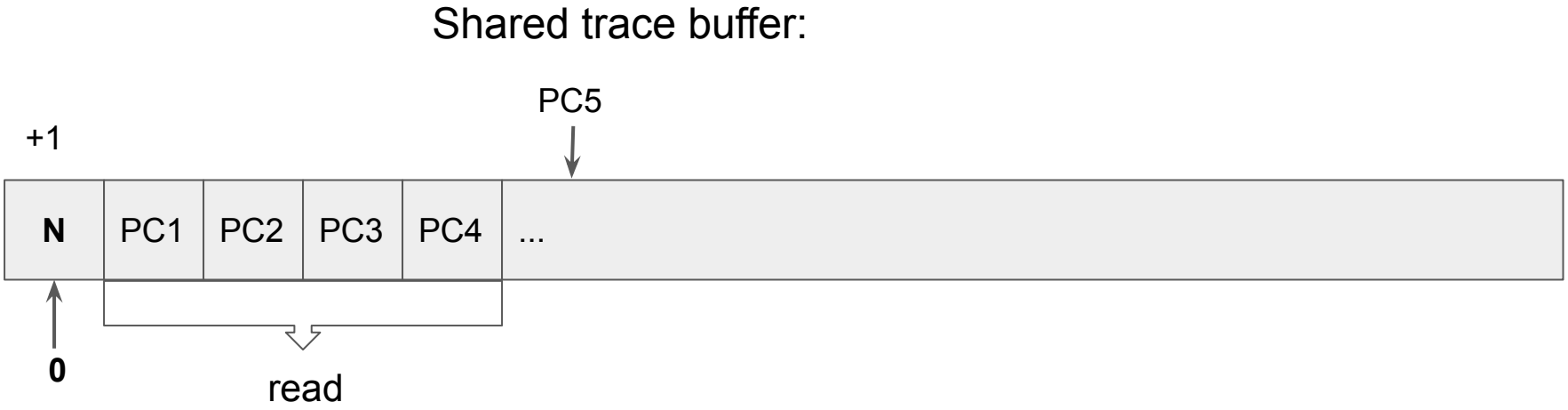
GCC/LLVM pass inserts a function call into every basic block/edge:

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



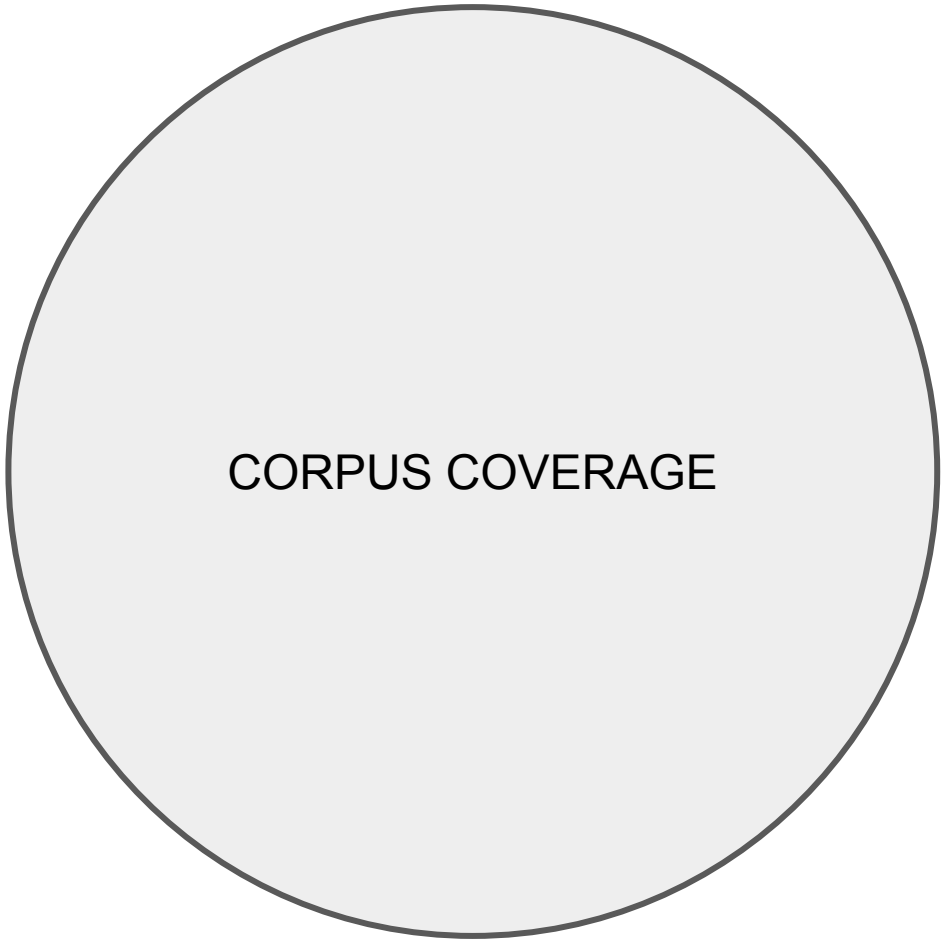
```
__sanitizer_cov_trace_pc();
if (...) {
    __sanitizer_cov_trace_pc();
        ...
}
__sanitizer_cov_trace_pc();
```

KCOV runtime

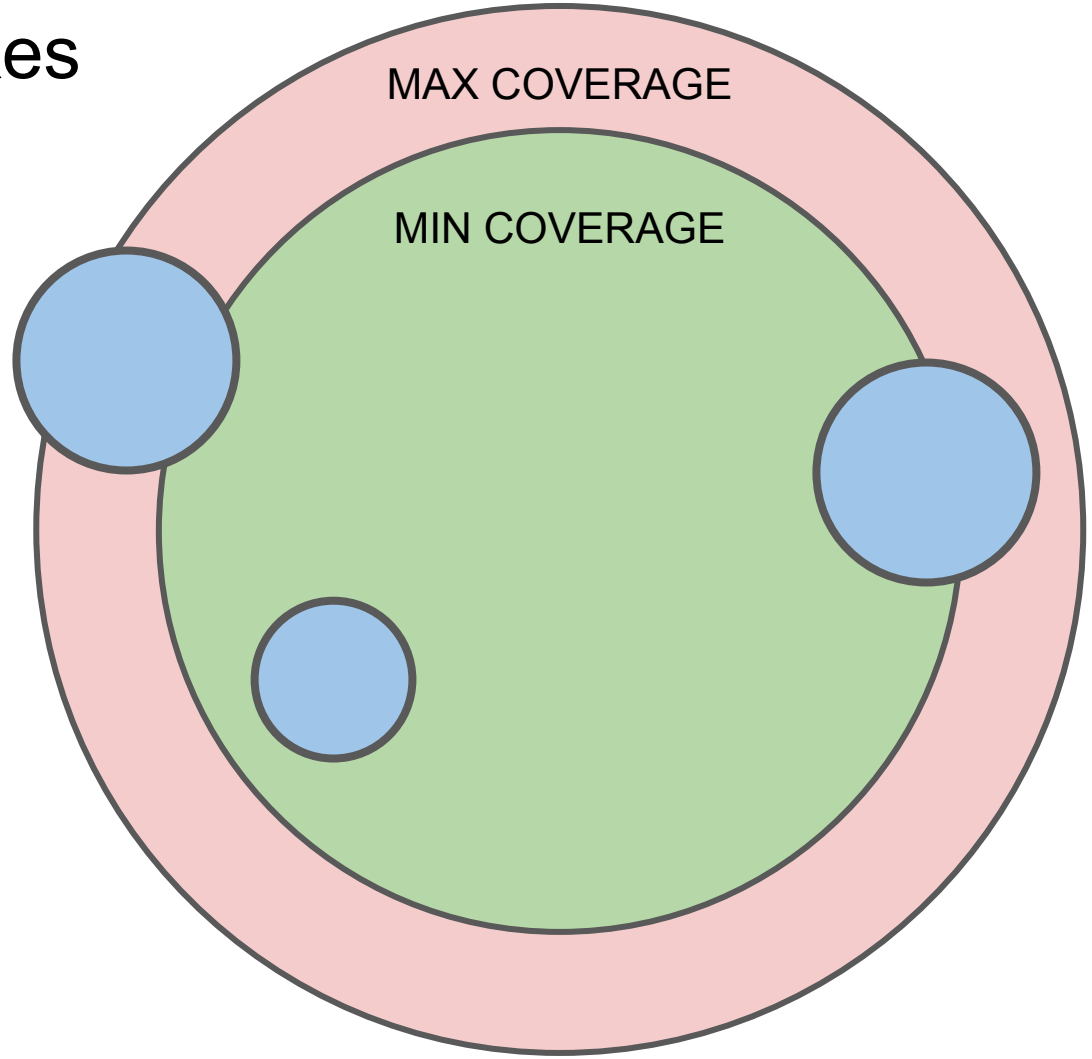


- per thread
- threads can be traced independently
- interrupts are ignored
- some kernel files are not instrumented

Coverage flakes



Coverage flakes



Does it work?

Oh, yes!

KASAN: OOB write in watch_queue_set_filter

```
int main() {
  mmap(0x20000000, 0x1000000, 3, 0x32, -1, 0);
  intptr_t res = 0;
  res = open("/dev/watch_queue", 0, 0);
  if (res != -1)
    r[0] = res;
  *(uint32_t*)0x20000240 = 1;
  *(uint32_t*)0x20000244 = 0;
  *(uint32_t*)0x20000248 = 0x300;
  *(uint32_t*)0x2000024c = 0;
  *(uint32_t*)0x20000250 = 0;
  *(uint32_t*)0x20000254 = 0;
  *(uint32_t*)0x20000258 = 0;
  *(uint32_t*)0x2000025c = 0;
  *(uint32_t*)0x20000260 = 0;
  *(uint32_t*)0x20000264 = 0;
  *(uint32_t*)0x20000268 = 0;
  *(uint32_t*)0x2000026c = 0;
  *(uint32_t*)0x20000270 = 0;
  ioctl(r[0], 0x5761, 0x20000240);
}
```

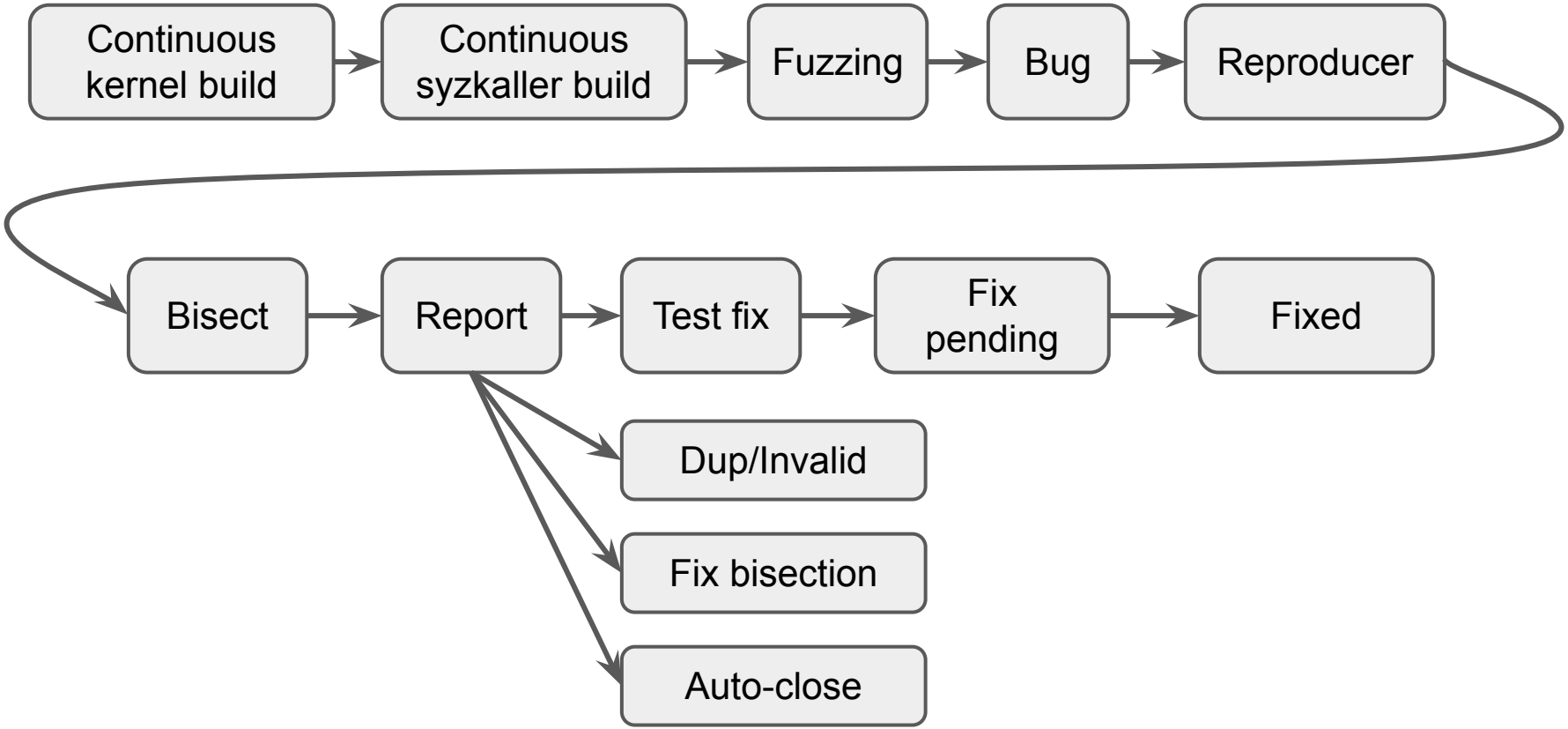
```
BUG: KASAN: slab-out-of-bounds in watch_queue_set_filter
Write of size 4 at addr ffff8880a9b31ddc by task syz-executor545/9

Call Trace:
 __asan_report_store4_noabort+0x17/0x20 generic_report.c:139
 watch_queue_set_filter drivers/misc/watch_queue.c:516 [inline]
 watch_queue_ioctl+0x15ed/0x16e0 drivers/misc/watch_queue.c:555
 do_vfs_ioctl+0x977/0x14e0 fs/ioctl.c:732
 ksys_ioctl+0xab/0xd0 fs/ioctl.c:749

Allocated by task 9097:
 kzalloc include/linux/slab.h:670 [inline]
 watch_queue_ioctl+0xf57/0x16e0 drivers/misc/watch_queue.c:555
 do_vfs_ioctl+0x977/0x14e0 fs/ioctl.c:732
 ksys_ioctl+0xab/0xd0 fs/ioctl.c:749

Freed by task 8821:
 kfree+0x10a/0x2c0 mm/slab.c:3757
 single_release+0x95/0xc0 fs/seq_file.c:609
 __fput+0x2ff/0x890 fs/file_table.c:280
 ____fput+0x16/0x20 fs/file_table.c:313
 task_work_run+0x145/0x1c0 kernel/task_work.c:113
 tracehook_notify_resume include/linux/tracehook.h:188 [inline]
 exit_to_usermode_loop+0x316/0x380 arch/x86/entry/common.c:164
```

syzbot


Continuous kernel build
Continuous syzkaller build
Fuzzing
Bug
Reproducer
Bisect
Report
Test fix
Fix pending
Fixed
Dup/Invalid
Fix bisection
Auto-close

To: coreteam@netfilter.org, netdev@vger.kernel.org, John Doe, ...
Hello,

syzbot found the following crash on:

HEAD commit: 8f8972a3 Merge tag 'mtd/fixes-for-5.5-rc7'
console output: https://syzkaller.appspot.com/x/log.txt?x=1327fa85e00000
kernel config: https://syzkaller.appspot.com/x/.config?x=cfbb8fa33f49f9f3
dashboard link: https://syzkaller.appspot.com/bug?extid=8b5f151de2f35100bbc5
compiler: clang version 10.0
C reproducer: https://syzkaller.appspot.com/x/repro.c?x=16056faee00000

IMPORTANT: if you fix the bug, please add the following tag to the commit:
Reported-by: syzbot+8b5f151de2f35100bbc5@syzkaller.appspotmail.com

BUG: KASAN: use-after-free in bitmap_ip_destroy
Call Trace:
...
Allocated by task 8711:
...

This bug is generated by a bot. It may contain errors.
See https://goo.gl/tpsmEJ for more information about syzbot.
syzbot engineers can be reached at syzkaller@googlegroups.com.

syzbot: Upstream Linux kernel

Reported:	2854+
Fixed:	2046 (71%)
Open:	808

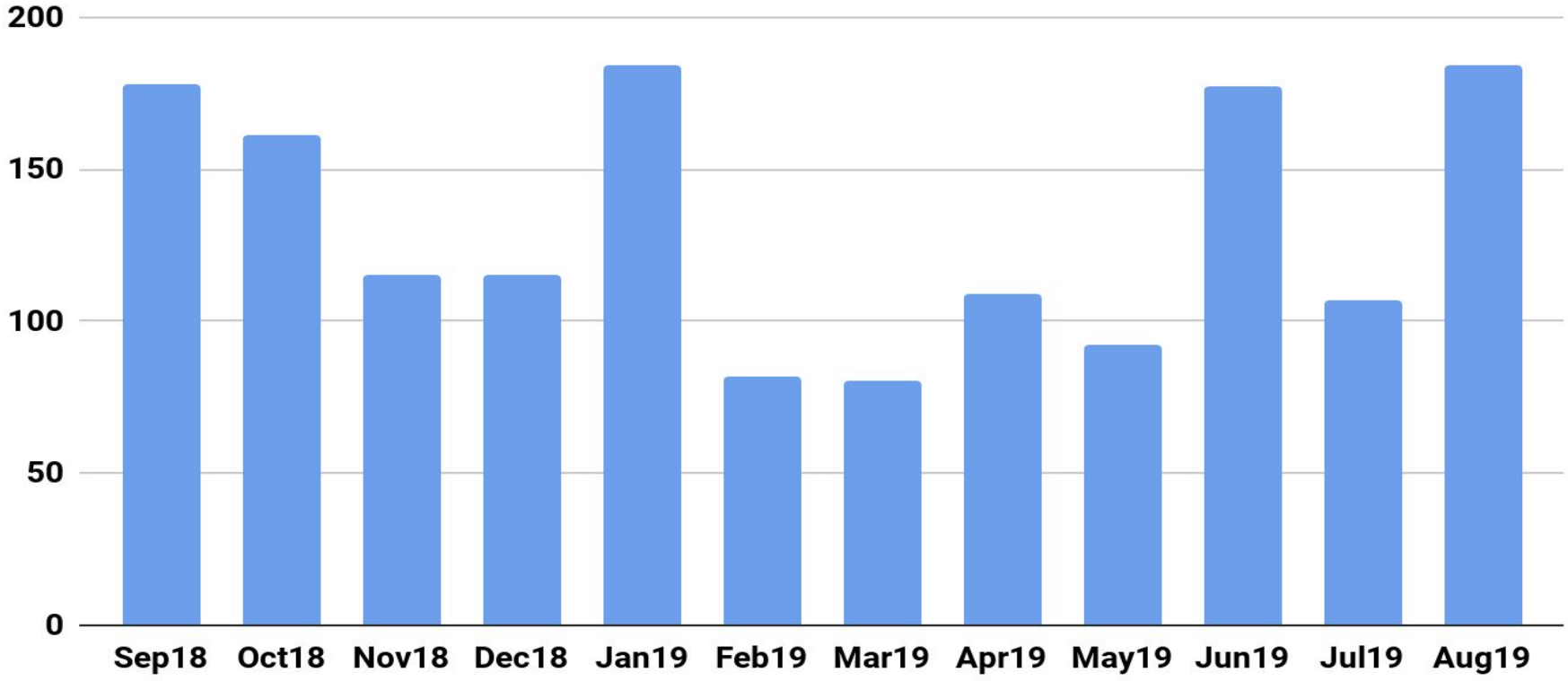
2.5 years: 3 bugs/day, 2 fixed

(+1000 reported manually)

syzkaller.appspot.com: open bugs (671)

Title	Repro	Bisected	Count	Last	Reported
KASAN: invalid-free in iowarrior_disconnect			3	1d02h	1d11h
KASAN: invalid-free in rsi_91x_deinit	C		84	3d08h	119d
KASAN: slab-out-of-bounds Read in bacpy	C	cause	15	4h33m	225d
KASAN: slab-out-of-bounds Read in class_equal	syz	cause	79	13d	87d
KASAN: slab-out-of-bounds Read in hci_event_packet	C	cause	9	30d	225d
KASAN: slab-out-of-bounds Read in hidraw_ioctl	C		48	14h42m	21d
KASAN: slab-out-of-bounds Read in mceusb_dev_recv	C		2	6d00h	8d13h
KASAN: slab-out-of-bounds Write in ax_probe	C		4	2d23h	8d13h
KASAN: slab-out-of-bounds Write in check_noncircular	syz	cause	3	26d	32d
KASAN: slab-out-of-bounds Write in lg4ff_init	C		1	17d	15d
KASAN: use-after-free Read in adu_disconnect	C		236	57m	15d
KASAN: use-after-free Read in blkdev_bio_end_io	C	cause	13	3d16h	15d
KASAN: use-after-free Read in blkdev_direct_IO	C	cause	8	13d	18d
KASAN: use-after-free Read in ccid2_hc_tx_packet_recv	C		77	8d20h	505d
KASAN: use-after-free Read in debugfs_remove (3)	C	cause	77	2d10h	314d
KASAN: use-after-free Read in dvb_usb_device_exit (2)	C		64	2h28m	14d
KASAN: use-after-free Read in hidraw_ioctl	C		1394	1h10m	28d
KASAN: use-after-free Read in iowarrior_disconnect	C		299	7m	1d11h
KASAN: use-after-free Read in iowarrior_release	C		3	3d12h	1d11h
KASAN: use-after-free Read in kfree_skb (3)	C	cause	90	13h55m	105d
KASAN: use-after-free Read in nr_rx_frame (2)	C	cause	3	20d	28d

syzbot bugs/month



Other kernels

Linux Upstream	2854
Linux 4.19, 4.14	811
Android 4.4, 4.9, 4.14, 5.4	614
Internal kernels	1852
gVisor	142
Fuchsia	50
FreeBSD	103
NetBSD	107
OpenBSD	159
AkarOS	40
Total	**6732**

Bug type split

Use-after-free	18.5%
Heap-out-of-bounds	5.2%
Stack-out-of-bound	2.4%
Double-free	0.8%
Wild-access	4.8%
Uninit-memory	4.0%
GPF	20.2%
BUG/panic/div0	10.3%
deadlock/hang/stall	12.5%
WARNING	21.1%

Modest estimation: thousands security bugs (not counting DoS; few have CVEs).

Exploit != use-after-free

- **WARNING** -> inter-VM/process info leaks
  - failure to restore registers
  - WARNING in __switch_to / WARNING in fpu__copy

- **"unresponsive" machine** -> full guest->host escape
  - page ref leak
  - CVE-2017-2596 / kvm: fix page struct leak in handle_vmon

- **stall** -> remote network DoS
  - lockup in udp[v6]_recvmsg
  - anything remotely triggerable is a concern

USB Stack State

Barely scratching the surface yielded 270+ externally triggerable bugs (33 CVEs).

Barely get past handshake (WIP)

USB is not special. Flow of bugs is representative for any subsystem
(kvm, tcp, udp, rdma, sound, 9p, bpf, you name it)

USB drivers

- usb/core: memory corruption due to an out-of-bounds access in usb_destroy_configuration [fix] [CVE-2017-17558]
- usb/net/zd1211rw: possible deadlock in zd_chip_disable_rxtx
- usb/sound: use-after-free in __uac_clock_find_source [fix]
- usb/sound: slab-out-of-bounds in parse_audio_unit [fix]
- usb/media/em28xx: use-after-free in dvb_unregister_frontend [fix]
- usb/media/technisat: slab-out-of-bounds in technisat_usb2_rc_query
- usb/media/tm6000: use-after-free in tm6000_read_write_usb
- usb/net/qmi_wwan: divide error in qmi_wwan_probe/usbnet_probe [fix1, fix2] [CVE-2017-16649, CVE-2017-16650]
- usb/media/uvc: slab-out-of-bounds in uvc_probe
- usb/media/em28xx: use-after-free in em28xx_dvb_fini
- usb/media/em28xx: use-after-free in v4l2_fh_init
- usb/media/pvrusb2: WARNING in pvr2_i2c_core_done/sysfs_remove_group
- usb/sound/usx2y: WARNING in usb_stream_start [fix]
- usb/net/hfa384x: WARNING in submit_rx_urb/usb_submit_urb
- usb/media/dw2102: null-ptr-deref in dvb_usb_adapter_frontend_init/tt_s2_4600_frontend_attach
- usb/net/asix: kernel hang in asix_phy_reset
- usb/media/dtt200u: use-after-free in __dvb_frontend_free [fix] [CVE-2017-16648]
- usb/media/mxl111sf: trying to register non-static key in mxl111sf_ctrl_msg
- usb/media/au0828: use-after-free in au0828_rc_unregister
- usb/input/gtco: slab-out-of-bounds in parse_hid_report_descriptor [fix] [CVE-2017-16643]
- usb/core: slab-out-of-bounds in usb_get_bos_descriptor [fix] [CVE-2017-16535]
- usb/net/asix: null-ptr-deref in asix_suspend [fix] [CVE-2017-16647]
- usb/net/rt2x00: warning in rt2800_eeprom_word_index
- usb/irda: global-out-of-bounds in irda_qos_bits_to_value
- usb/media/imon: global-out-of-bounds in imon_probe/imon_init_intf0
- usb/sound: use-after-free in snd_usb_mixer_interrupt [fix] [CVE-2017-16527]
- usb/net/rtlwifi: trying to register non-static key in rtl_c2hcmd_launcher
- usb/net/prism2usb: warning in hfa384x_usbctlxq_run/usb_submit_urb
- usb/nfs/pn533: use-after-free in pn533_send_complete
- usb/media/imon: null-ptr-deref in imon_probe [fix] [CVE-2017-16537]
- usb/net/prism2usb: warning in hfa384x_drvr_start/usb_submit_urb

Future work

- More descriptions
- More input injection (USB, NFC, WiFi, Bluetooth, ...)
- Auto-generating descriptions
- Extending syzlang
- Smarter fuzzing
- Triggering more races
- More dynamic tools (KUBSAN, KCSAN)
- Making more bugs fixed
- Improving process automation
- Testing hypervisors, VMMs, user-space libraries
- Porting to new OSes

Windows?

- Keeping Windows Secure
- Bugs on the Windshield: Fuzzing the Windows Kernel
- WSL Reloaded
- "...the fork will not be made public for at least a year,still finding too many good issues..."

Some WSL crashers ([1], [2]):

```
void main() {
    write(open("/proc/self/setgroups", O_RDWR), 0xdeadbabe, 6)
}

void main() {
    unshare(CLONE_NEWNS);
    open("/proc/self/ns/mnt", O_RDONLY);
}
```

Linux Vulnerabilities Windows Exploits: Escalating Privileges with WSL

Thank you!

github.com/google/syzkaller

syzkaller.appspot.com

Dmitry Vyukov, dvyukov@

Backup

C reproducers

```
res = syscall(__NR_socket, 0xa, 2, 0);
if (res != -1)
     r[1] = res;

res = syscall(__NR_socket, 0x18, 1, 1);
if (res != -1)
     r[2] = res;

*(uint16_t*)0x20000180 = 0x18;
*(uint32_t*)0x20000182 = 1;
*(uint32_t*)0x20000186 = 0;
*(uint32_t*)0x2000018a = r[2];
*(uint16_t*)0x2000018e = 2;
*(uint16_t*)0x20000190 = htobe16(0);
*(uint32_t*)0x20000192 = htobe32(0xe0000002);
*(uint32_t*)0x2000019e = 4;
*(uint32_t*)0x200001a2 = 0;
*(uint32_t*)0x200001a6 = 0;
*(uint32_t*)0x200001aa = 0;
syscall(__NR_connect, r[1], 0x20000180, 0x26);
```

Discriminated syscalls

~300 syscalls (?)

```
fcntl$F_DUPFD(fd fd, cmd const[F_DUPFD], arg fd) fd
fcntl$F_GETFD(fd fd, cmd const[F_GETFD])
fcntl$F_SETFD(fd fd, cmd const[F_SETFD], flags flags[fcntl_flags])
...
ioctl$FLOPPY_FDEJECT(fd fd, cmd const[FDEJECT])
ioctl$FLOPPY_FDSETPRM(fd fd, cmd const[FDSETPRM], ...)
...
socket$AF_INET(domain const[AF_INET], type const[SOCK_STREAM], ...)
socket$AF_ALG(domain const[AF_ALG], type const[SOCK_SEQPACKET], ...)
```

3200+ "syscalls" (+thousands of union options)

Templates!

```
type dec_opt[NAME, VAL] {
    name    stringnoz[NAME]
    eq      const['=', int8]
    val     fmt[dec, VAL]
} [packed]
```

```
    dec_opt["uid", uid]
    dec_opt["pid", pid]
    dec_opt["blocks", flags[nblocks]]
```

VOID

- some_template[CMD_WITHOUT_ARGS, **void**]
- optional[T]

```
type optional[T] [
    something      T
    nothing        void
] [varlen]
```

Comparison interception