Linux Kernel Fuzzing with SyzKaller

90 minutes introduction

hacktive security

Agenda

Linux Kernel Fuzzing Introduction

Syzkaller Introduction

Extend Syzkaller descriptions with Syzlang

Conclusion

whoami

- Alessandro Groppo (*@kiks*)
  - @Hacktive Security since 2018
    - 2018 – 2023 – Penetration Tester (web, mobile, infrastructure, IoT , ..)
    - 2023 – Vulnerability Researcher / Penetration Tester
  - I love hacking Android, Linux, Kernel & Userland and low-level stuff
  - Personal blog: 1day.dev

Linux Kernel Fuzzing

 # Introduction to Fuzzing

 # Kernel Obstacles and solutions

 # Code coverage (KCOV)

 # Memory Sanitizers (KASAN, KMSAN, UBSAN, ..)

---

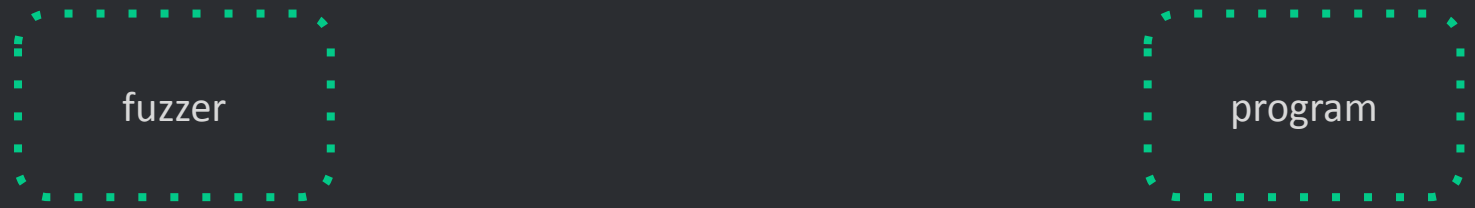
hacktive security

Fuzzing

- «Fuzzing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a program.» *Wikipedia*

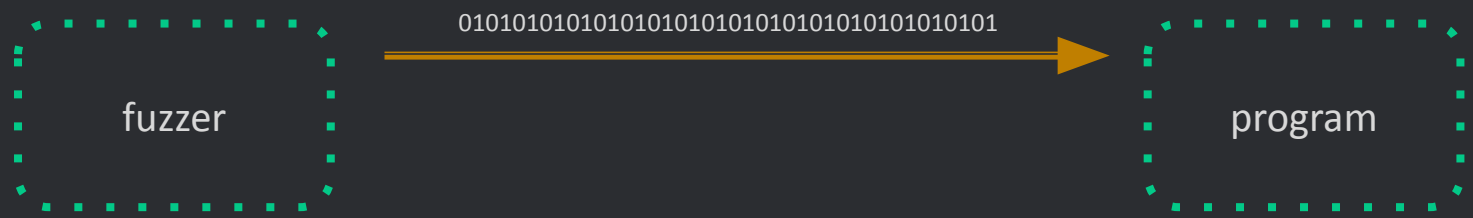
Fuzzing

- «Fuzzing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a program.» *Wikipedia*



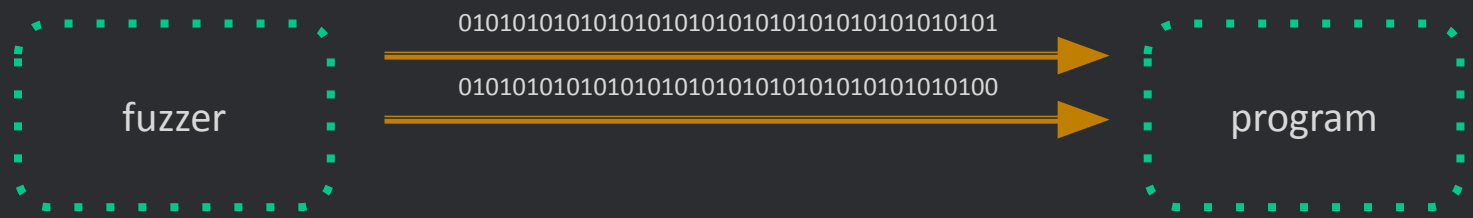
Fuzzing

- «Fuzzing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a program.» *Wikipedia*



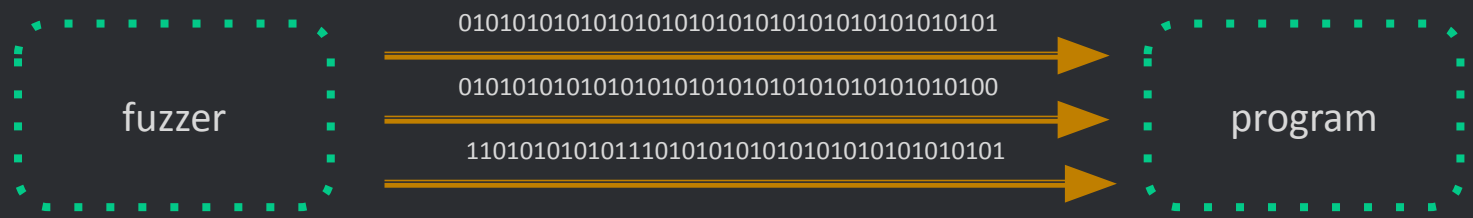
Fuzzing

- «Fuzzing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a program.» *Wikipedia*



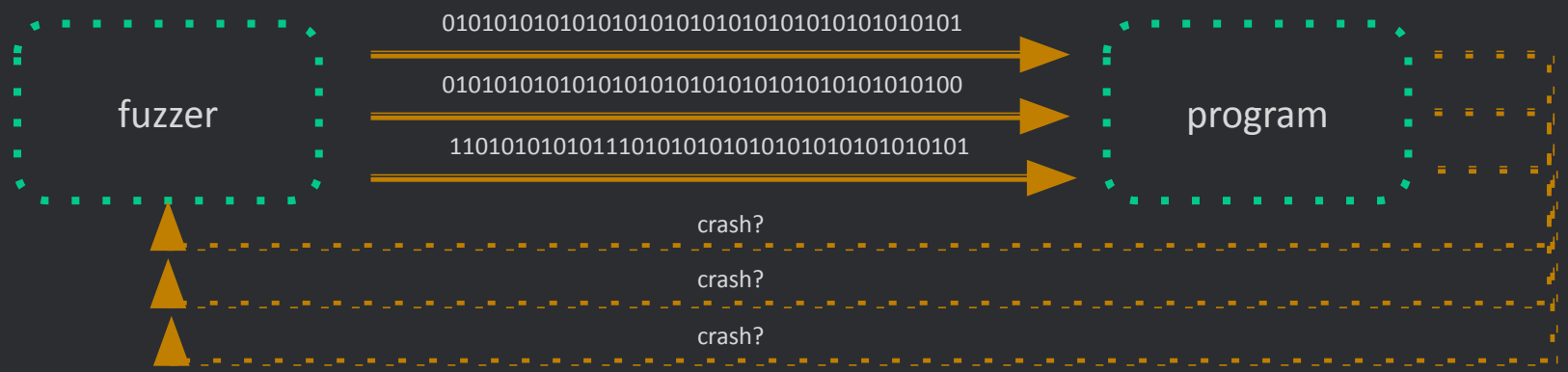
Fuzzing

- «Fuzzing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a program.» *Wikipedia*



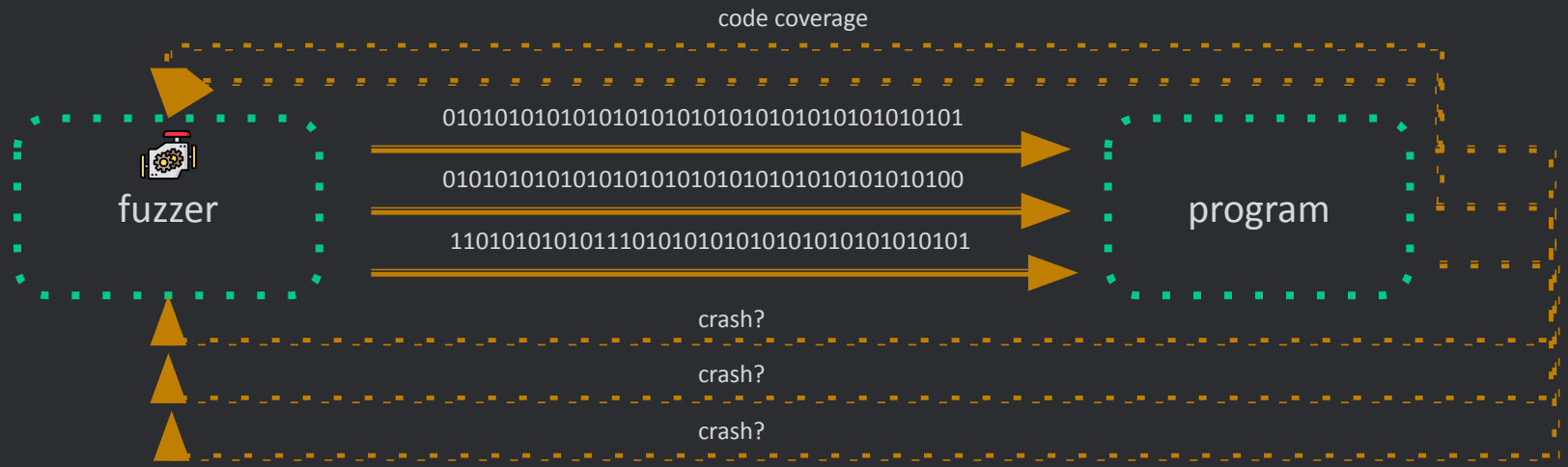
Fuzzing

- «Fuzzing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a program.» *Wikipedia*
- «program is then monitored for exceptions such as crashes, ..»



Fuzzing

- «Fuzzing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a program.» *Wikipedia*
- «program is then monitored for exceptions such as crashes, ..»
- Collect code coverage feedback
  - mutation engine



Fuzzing with code coverage

- Effective to cover the majority of the codebase
- Transform input based feedback
  1. start from an initial input (aka corpus)
  2. execute crash? collect coverage mutate based on feedback repeat
- Different solutions
  - compile-time: gcov (gcc), LLVM SanitizerCoverage, ..
  - black-box: DynamoRIO, TinyInst, ..
- Different levels
  - function, basic blocks, edge levels

```
if(input[0] == 1)
        if(input[1] == 2)
        if(input[2] == 3)
                if(input[3] == 4)
                        // BUG
```

gcc

```
endbr64
push   rbp
mov    rbp,rsp
sub    rsp,0x30
mov    DWORD PTR [rbp-0x24],edi
mov    QWORD PTR [rbp-0x30],rsi
mov    rax,QWORD PTR fs:0x28

mov    QWORD PTR [rbp-0x8],rax
xor    eax,eax
lea    rax,[rbp-0x12]
mov    edx,0xa
mov    rsi,rax
mov    edi,0x0
call   1090 <read@plt>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x1
jne    11f5 <main+0x6c>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x2
jne    11f5 <main+0x6c>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x3
jne    11f5 <main+0x6c>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x4
jne    11f5 <main+0x6c>
lea    rax,[rip+0xe1c]        # 2004 <_IO_stdin_used+0x4>
mov    rdi,rax
mov    eax,0x0
call   1080 <printf@plt>
mov    eax,0x0
mov    rdx,QWORD PTR [rbp-0x8]
sub    rdx,QWORD PTR fs:0x28

je     120e <main+0x85>
call   1070 <__stack_chk_fail@plt>
leave
ret
```

-ftest-coverage -fPIC -fprofile-arcs

```
endbr64
push   rbp
mov    rbp,rsp
sub    rsp,0x30
mov    DWORD PTR [rbp-0x24],edi
mov    QWORD PTR [rbp-0x30],rsi
mov    rax,QWORD PTR fs:0x28

mov    QWORD PTR [rbp-0x8],rax
xor    eax,eax
mov    rax,QWORD PTR [rip+0x4bbe]        # 6100 <__gcov0.main>
add    rax,0x1
mov    QWORD PTR [rip+0x4bb3],rax        # 6100 <__gcov0.main>
lea    rax,[rbp-0x12]
mov    edx,0xa
mov    rsi,rax
mov    edi,0x0
call   1330 <read@plt>
mov    rax,QWORD PTR [rip+0x4b9e]        # 6108 <__gcov0.main+0x8>
add    rax,0x1
mov    QWORD PTR [rip+0x4b93],rax        # 6108 <__gcov0.main+0x8>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x1
jne    15f1 <main+0xd8>
mov    rax,QWORD PTR [rip+0x4b8c]        # 6110 <__gcov0.main+0x10>
add    rax,0x1
mov    QWORD PTR [rip+0x4b81],rax        # 6110 <__gcov0.main+0x10>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x2
jne    15f1 <main+0xd8>
mov    rax,QWORD PTR [rip+0x4b7a]        # 6118 <__gcov0.main+0x18>
add    rax,0x1
mov    QWORD PTR [rip+0x4b6f],rax        # 6118 <__gcov0.main+0x18>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x3
jne    15f1 <main+0xd8>
mov    rax,QWORD PTR [rip+0x4b68]        # 6120 <__gcov0.main+0x20>
add    rax,0x1
mov    QWORD PTR [rip+0x4b5d],rax        # 6120 <__gcov0.main+0x20>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x4
jne    15f1 <main+0xd8>
mov    rax,QWORD PTR [rip+0x4b56]        # 6128 <__gcov0.main+0x28>
add    rax,0x1
mov    QWORD PTR [rip+0x4b4b],rax        # 6128 <__gcov0.main+0x28>
lea    rax,[rip+0x2a24]        # 4008 <_IO_stdin_used+0x8>
mov    rdi,rax
mov    eax,0x0
```

gcc

```
endbr64
push   rbp
mov    rbp,rsp
sub    rsp,0x30
mov    DWORD PTR [rbp-0x24],edi
mov    QWORD PTR [rbp-0x30],rsi
mov    rax,QWORD PTR fs:0x28

mov    QWORD PTR [rbp-0x8],rax
xor    eax,eax
lea    rax,[rbp-0x12]
mov    edx,0xa
mov    rsi,rax
mov    edi,0x0
call   1090 <read@plt>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x1
jne    11f5 <main+0x6c>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x2
jne    11f5 <main+0x6c>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x3
jne    11f5 <main+0x6c>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x4
jne    11f5 <main+0x6c>
lea    rax,[rip+0xe1c]        # 2004 <_IO_stdin_used+0x4>
mov    rdi,rax
mov    eax,0x0
call   1080 <printf@plt>
mov    eax,0x0
mov    rdx,QWORD PTR [rbp-0x8]
sub    rdx,QWORD PTR fs:0x28

je     120e <main+0x85>
call   1070 <__stack_chk_fail@plt>
leave
ret
```

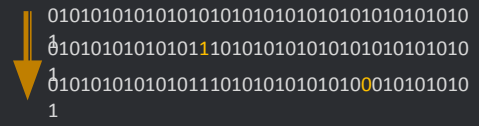
-ftest-coverage -fPIC -fprofile-arcs

```
endbr64
push   rbp
mov    rbp,rsp
sub    rsp,0x30
mov    DWORD PTR [rbp-0x24],edi
mov    QWORD PTR [rbp-0x30],rsi
mov    rax,QWORD PTR fs:0x28

mov    QWORD PTR [rbp-0x8],rax
xor    eax,eax
mov    rax,QWORD PTR [rip+0x4bbe]        # 6100 <__gcov0.main>
add    rax,0x1
mov    QWORD PTR [rip+0x4bb3],rax        # 6100 <__gcov0.main>
lea    rax,[rbp-0x12]
mov    edx,0xa
mov    rsi,rax
mov    edi,0x0
call   1330 <read@plt>
mov    rax,QWORD PTR [rip+0x4b9e]        # 6108 <__gcov0.main+0x8>
add    rax,0x1
mov    QWORD PTR [rip+0x4b93],rax        # 6108 <__gcov0.main+0x8>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x1
jne    15f1 <main+0xd8>
mov    rax,QWORD PTR [rip+0x4b8c]        # 6110 <__gcov0.main+0x10>
add    rax,0x1
mov    QWORD PTR [rip+0x4b81],rax        # 6110 <__gcov0.main+0x10>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x2
jne    15f1 <main+0xd8>
mov    rax,QWORD PTR [rip+0x4b7a]        # 6118 <__gcov0.main+0x18>
add    rax,0x1
mov    QWORD PTR [rip+0x4b6f],rax        # 6118 <__gcov0.main+0x18>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x3
jne    15f1 <main+0xd8>
mov    rax,QWORD PTR [rip+0x4b68]        # 6120 <__gcov0.main+0x20>
add    rax,0x1
mov    QWORD PTR [rip+0x4b5d],rax        # 6120 <__gcov0.main+0x20>
movzx  eax,BYTE PTR [rbp-0x12]
cmp    al,0x4
jne    15f1 <main+0xd8>
mov    rax,QWORD PTR [rip+0x4b56]        # 6128 <__gcov0.main+0x28>
add    rax,0x1
mov    QWORD PTR [rip+0x4b4b],rax        # 6128 <__gcov0.main+0x28>
lea    rax,[rip+0x2a24]        # 4008 <_IO_stdin_used+0x8>
mov    rdi,rax
mov    eax,0x0
```

Fuzzing approaches

Mutation approach

- mutate an initial corpus based on collected feedbacks (e.g. bit flipping, add/delete bytes, ..)
- appropriate for: file parsing (e.g. images), network packets, blob of data, ..



Generation approach

- generate input from an input model or grammar specification
- appropriate for: Javascript engines, Linux kernel syscalls, ..

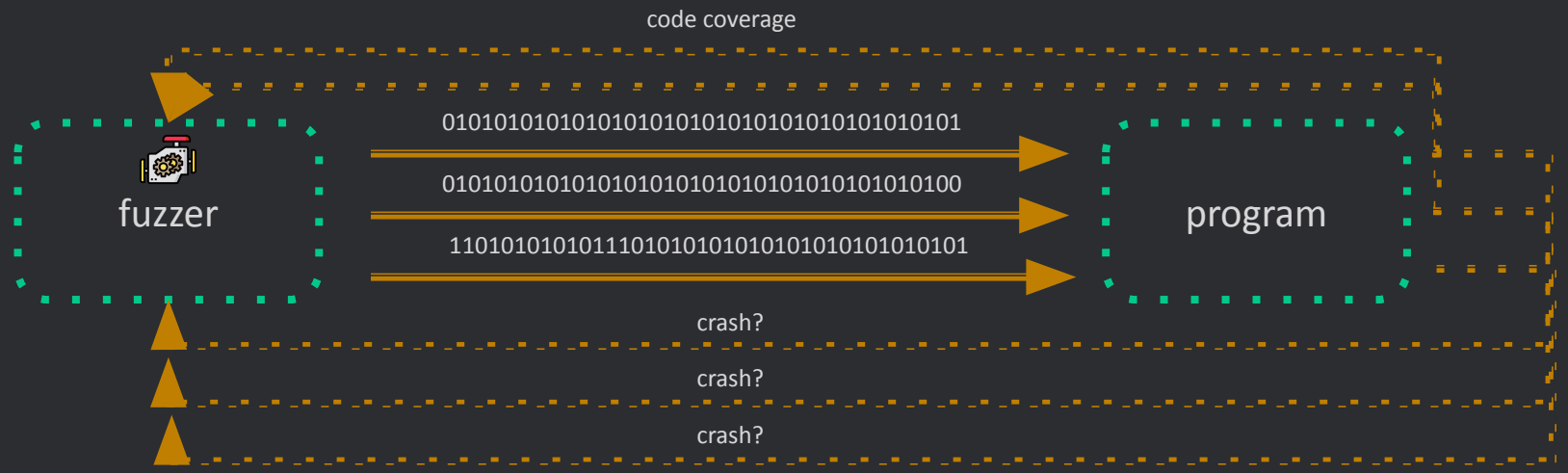
```
a = 1337
a.hasOwnProperty()
a = null
```

```
a = 1337
a = null
a.hasOwnProperty()
```

```
a.hasOwnProperty()
a = 1337
a = null
```

Fuzzing Introduction

- Fuzzing theory is a lot more than this!
- Key concepts
  - general idea
  - code coverage
  - approaches



Linux Kernel Fuzzing

Kernel fuzzing != userland fuzzing

- kernel crash == system crash
- driver and hardware support
- context switches
- different input sources

Multiple public solutions

- syzkaller
  - unsupervised coverage-guided fuzzer
  - grammar generation
- kAFL
  - x86 hardware assisted feedback fuzzer
- trinity
  - semi-intelligent argument generation (fd, value ranges, bit flip flags, ..)
- ..

Linux Kernel Fuzzing

Kernel fuzzing != userland fuzzing

- kernel crash == system crash
  - external host
- driver and hardware support
  - emulation or *physical device*
- context switches
  - per-task coverage (KCOV)
- different input sources
  - *(next slides)*

Multiple public solutions

- syzkaller
  - unsupervised coverage-guided fuzzer
  - grammar generation
- kAFL
  - x86 hardware assisted feedback fuzzer
- trinity
  - semi-intelligent argument generation (fd, value ranges, bit flip flags, ..)
- ..

Linux Kernel Fuzzing

Kernel fuzzing != userland fuzzing

- kernel crash == system crash
  - external host
- driver and hardware support
  - emulation or *physical device*
- context switches
  - per-task coverage (KCOV)
- different input sources
  - *(next slides)*

Multiple public solutions

- syzkaller
  - unsupervised coverage-guided fuzzer
  - grammar generation
- kAFL
  - x86 hardware assisted feedback fuzzer
- trinity
  - semi-intelligent argument generation (fd, value ranges, bit flip flags, ..)
- ..

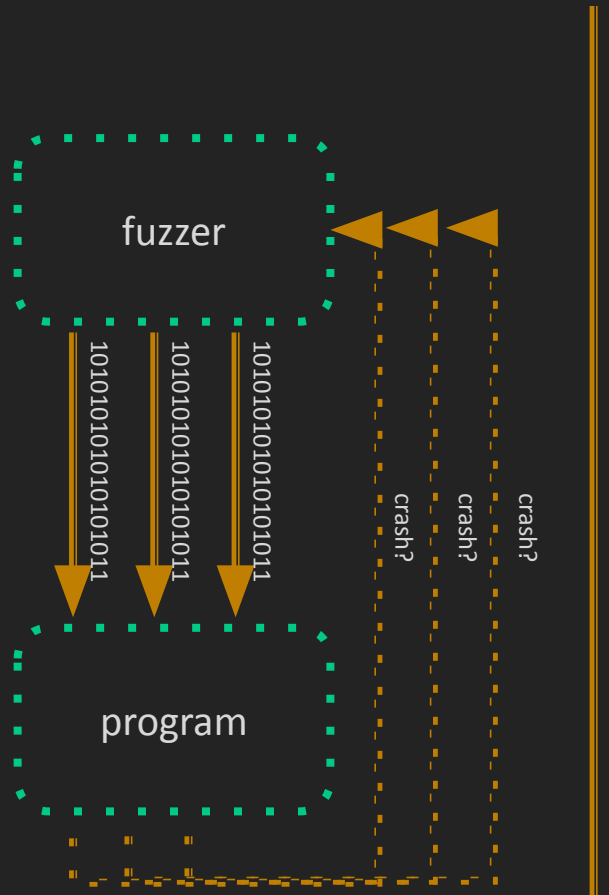
«It's not about the tool (*almost*), it's on how you use it»

Linux Kernel Attack Surface

- Linux Kernel high level Attack Surface
  - internal (syscalls)
    - from user-land to kernel-land (e.g. syscalls)
    - e.g. open, read, write, IPC , filesystem, ..
  - external
    - from hardware or firmware
    - e.g. external chipsets (bluetooth, WiFi, 4G, ..), USB, ..

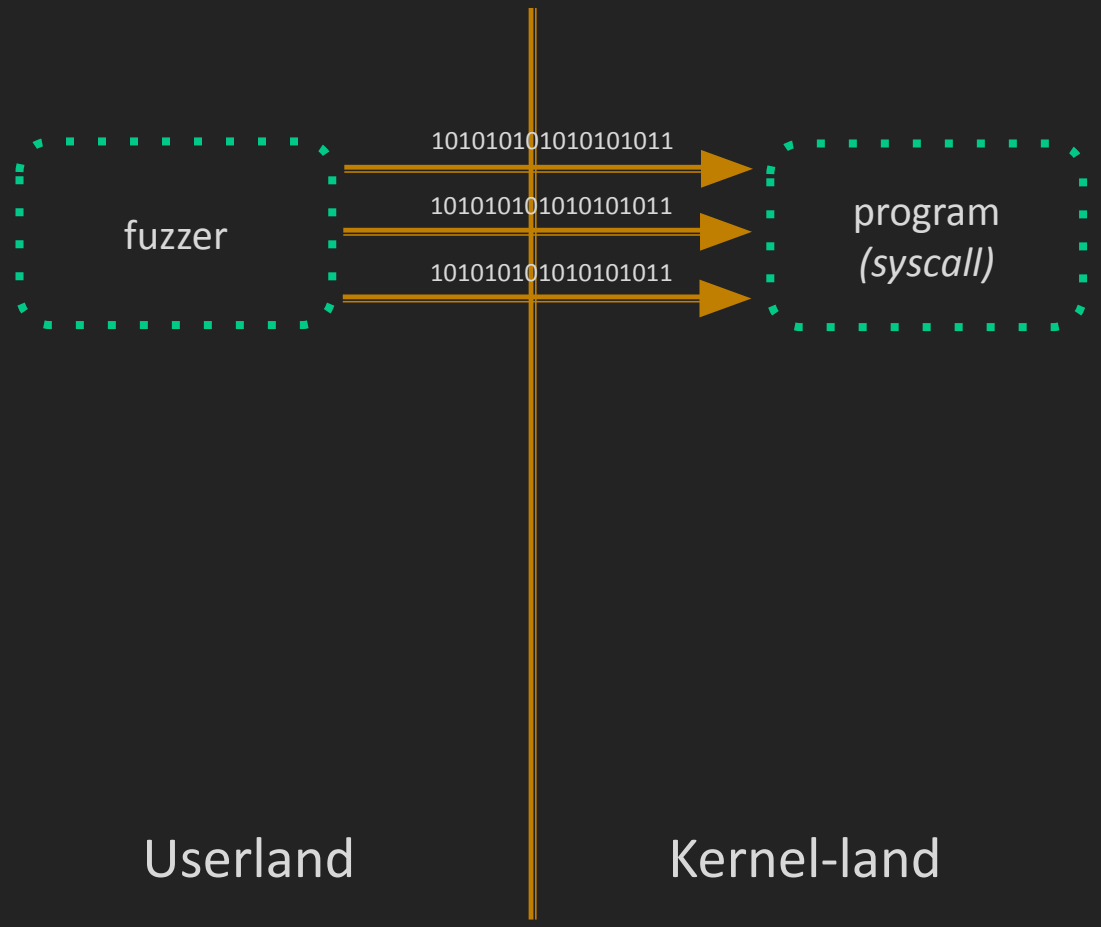
Linux Kernel Attack Surface

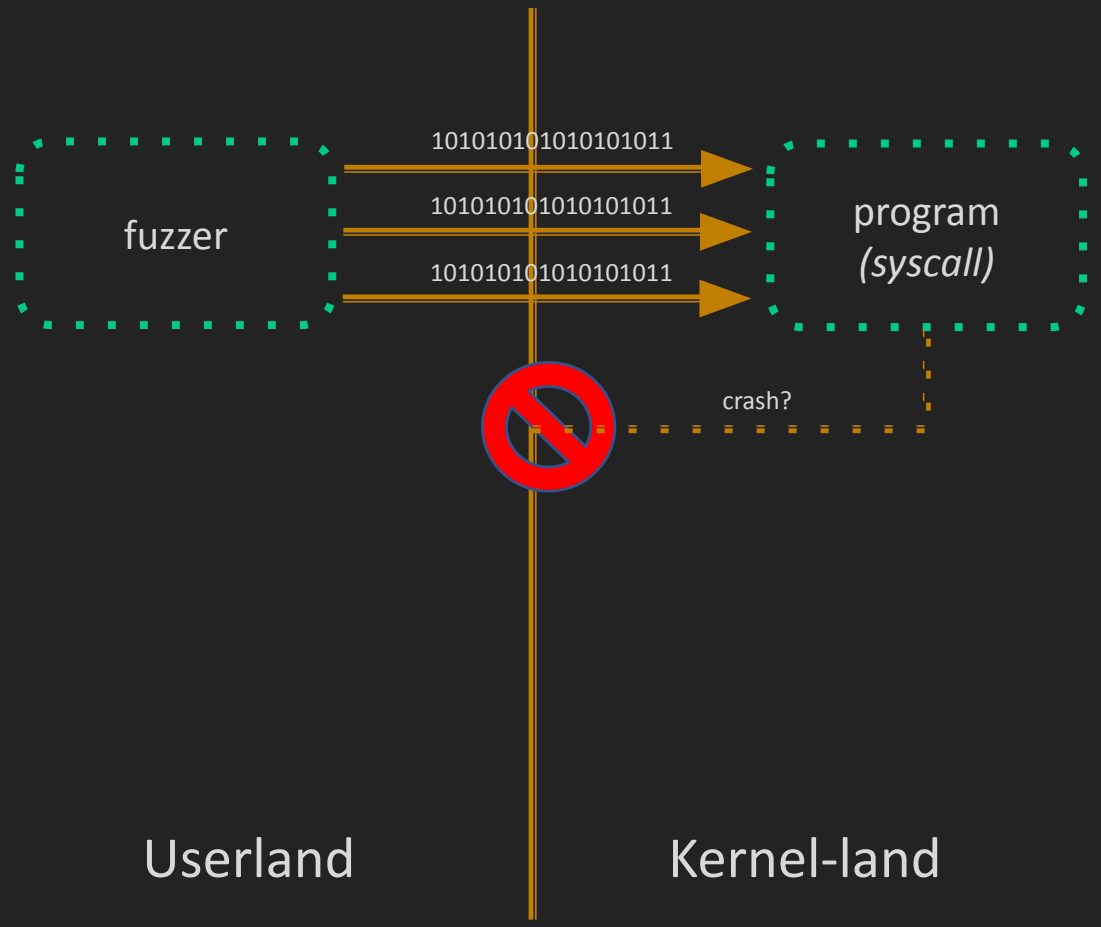
- Linux Kernel high level Attack Surface
  - internal (syscalls)
    - from user-land to kernel-land (e.g. syscalls)
    - e.g. open, read, write, IPC , filesystem, ..
    - interactions from userland
  - external
    - from hardware or firmware
    - e.g. external chipsets (bluetooth, WiFi, 4G, ..), USB, ..
    - requires more effort
      - e.g. chip emulation (hypervisor/emulator), custom userland syscalls, ..

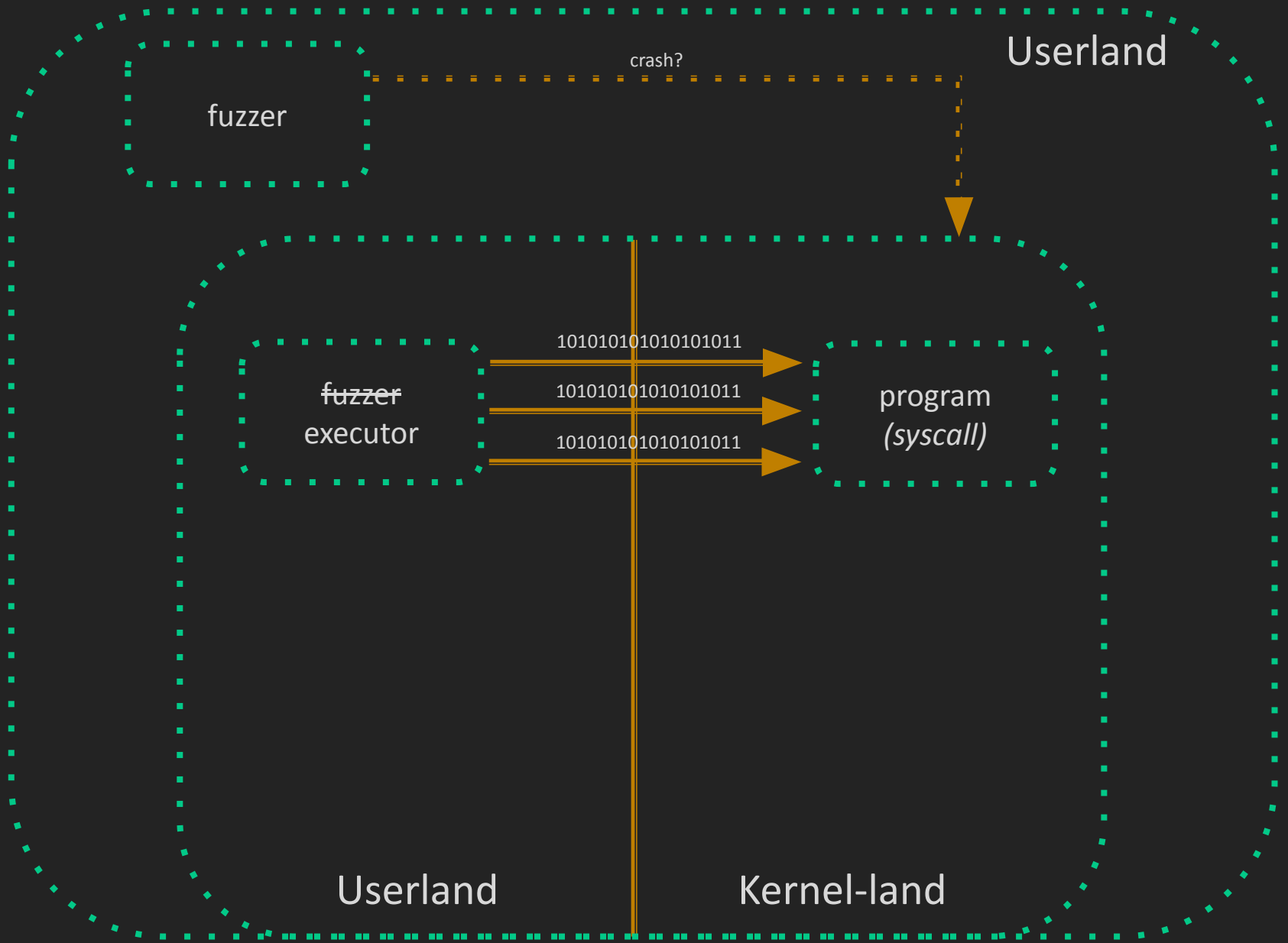


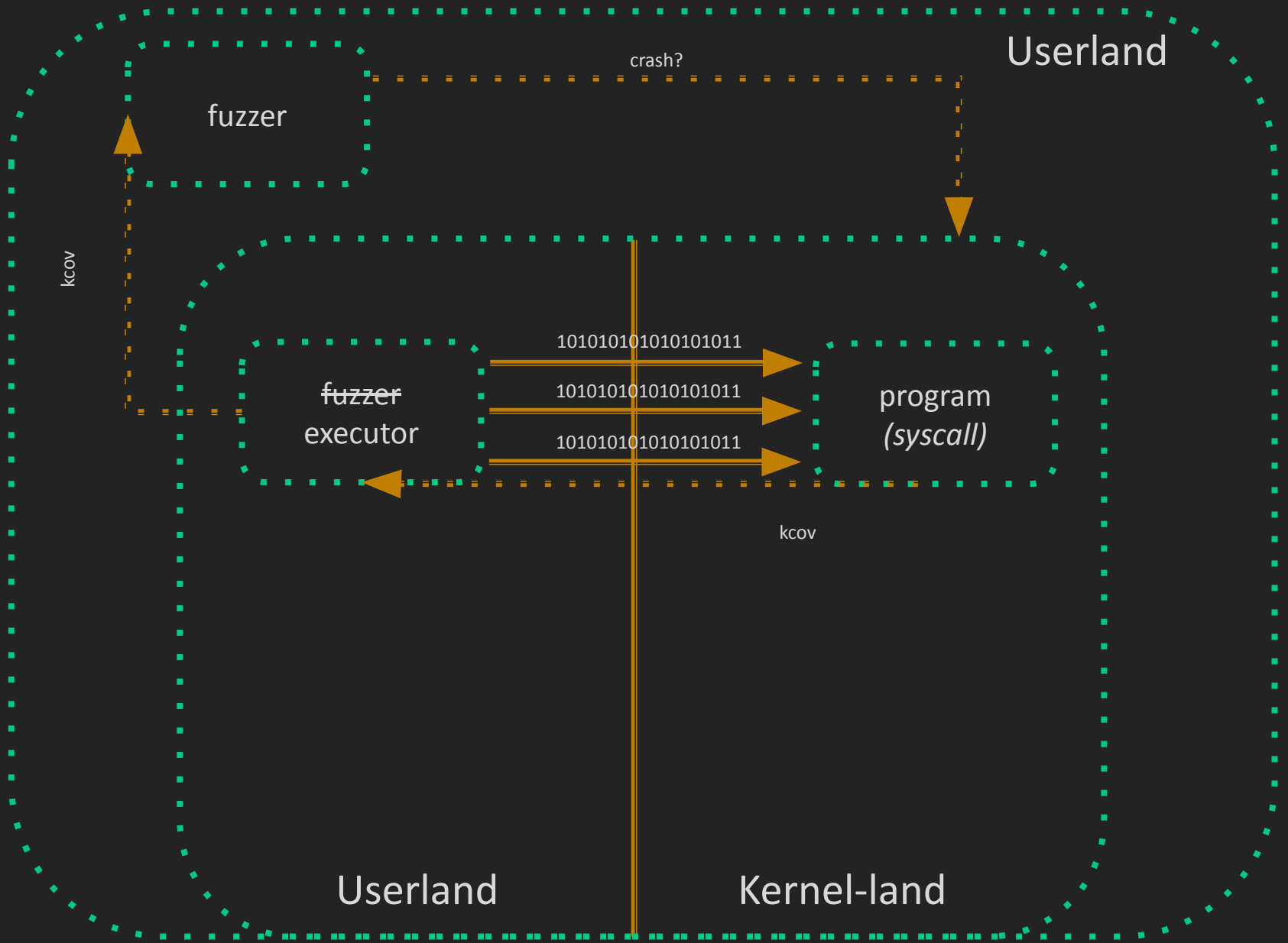
Userland

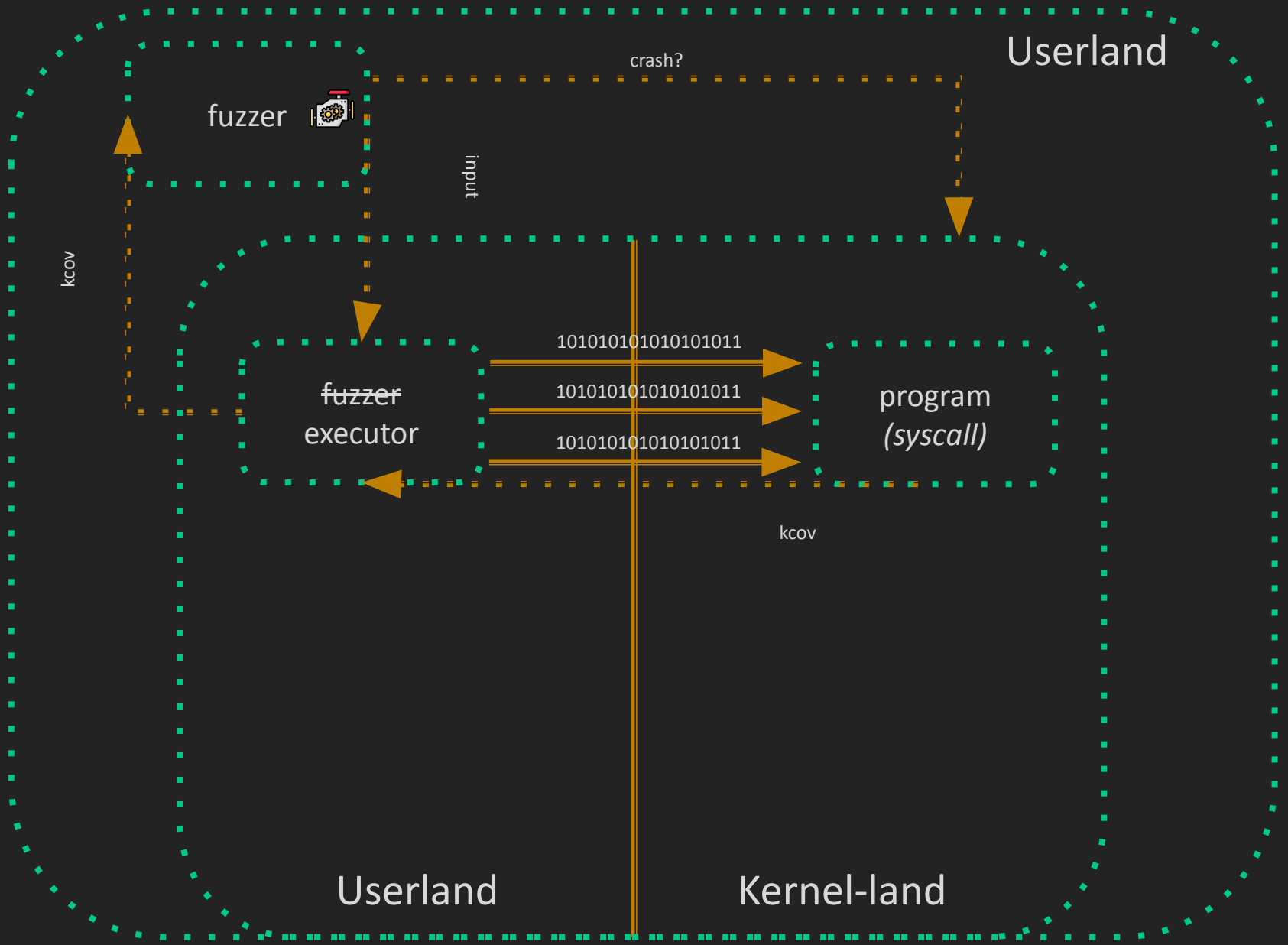
Kernel-land


fuzzer
1010101010101011
1010101010101011
1010101010101011
program
(syscall)
Userland
Kernel-land


1010101010101011
1010101010101011
1010101010101011
fuzzer
program
(syscall)
crash?
Userland
Kernel-land


Userland
crash?
fuzzer
1010101010101011
1010101010101011
1010101010101011
~~fuzzer~~
executor
program
(syscall)
Userland
Kernel-land


Userland
crash?
fuzzer
kcov
1010101010101011
1010101010101011
1010101010101011
~~fuzzer~~
executor
program
(syscall)
kcov
Userland
Kernel-land

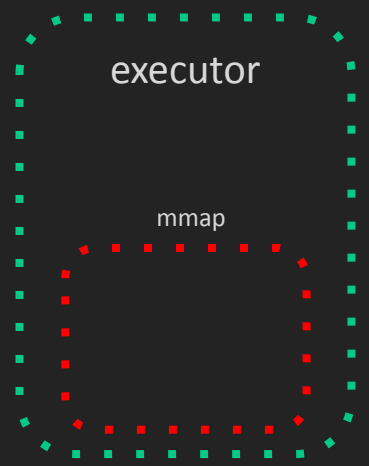

Userland
crash?
fuzzer
input
kcov
1010101010101011
1010101010101011
1010101010101011
~~fuzzer~~
executor
program
(syscall)
kcov
Userland
Kernel-land

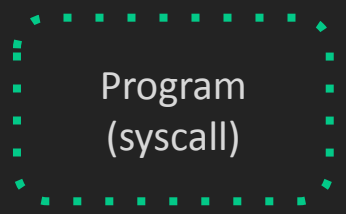
Kernel Code Coverage

- KCOV permits to collect per-task code coverage
  - Good for single syscalls
  - *GCOV collects global level coverage*
- Needs to be enabled at compile-time
  - CONFIG_KCOV / CONFIG_KCOV_INSTRUMENT_ALL
- Performance overhead

```
open("/sys/kernel/debug/kcov", ...);
mmap(NULL, ..., fd, 0);
ioctl(fd, KCOV_INIT_TRACE, ..);
```

```
0x0: copy_from_user(...)
0x8: if(input)
0x10:       return
0x18:       if(input[0] == 1)
0x20:       return
```

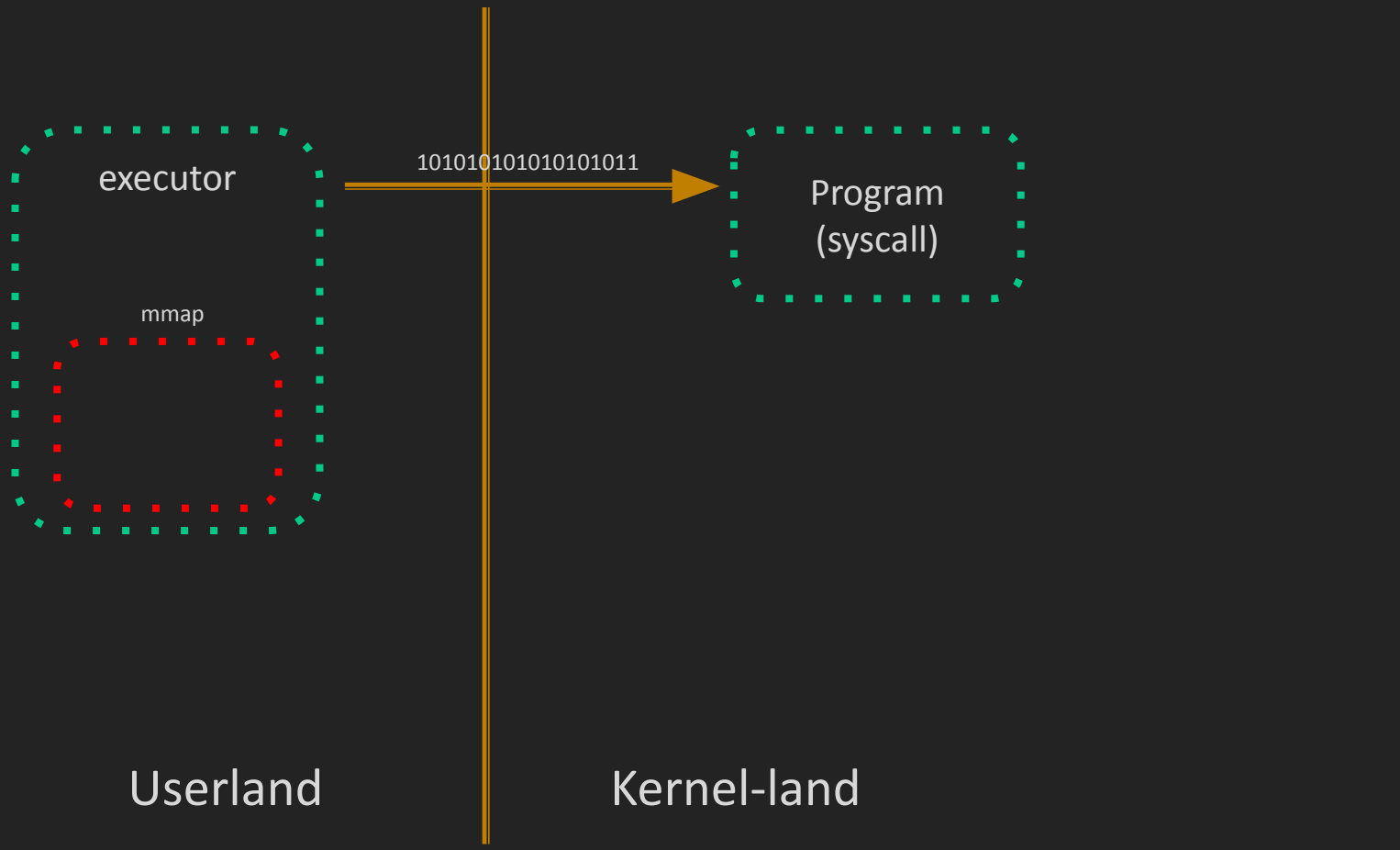




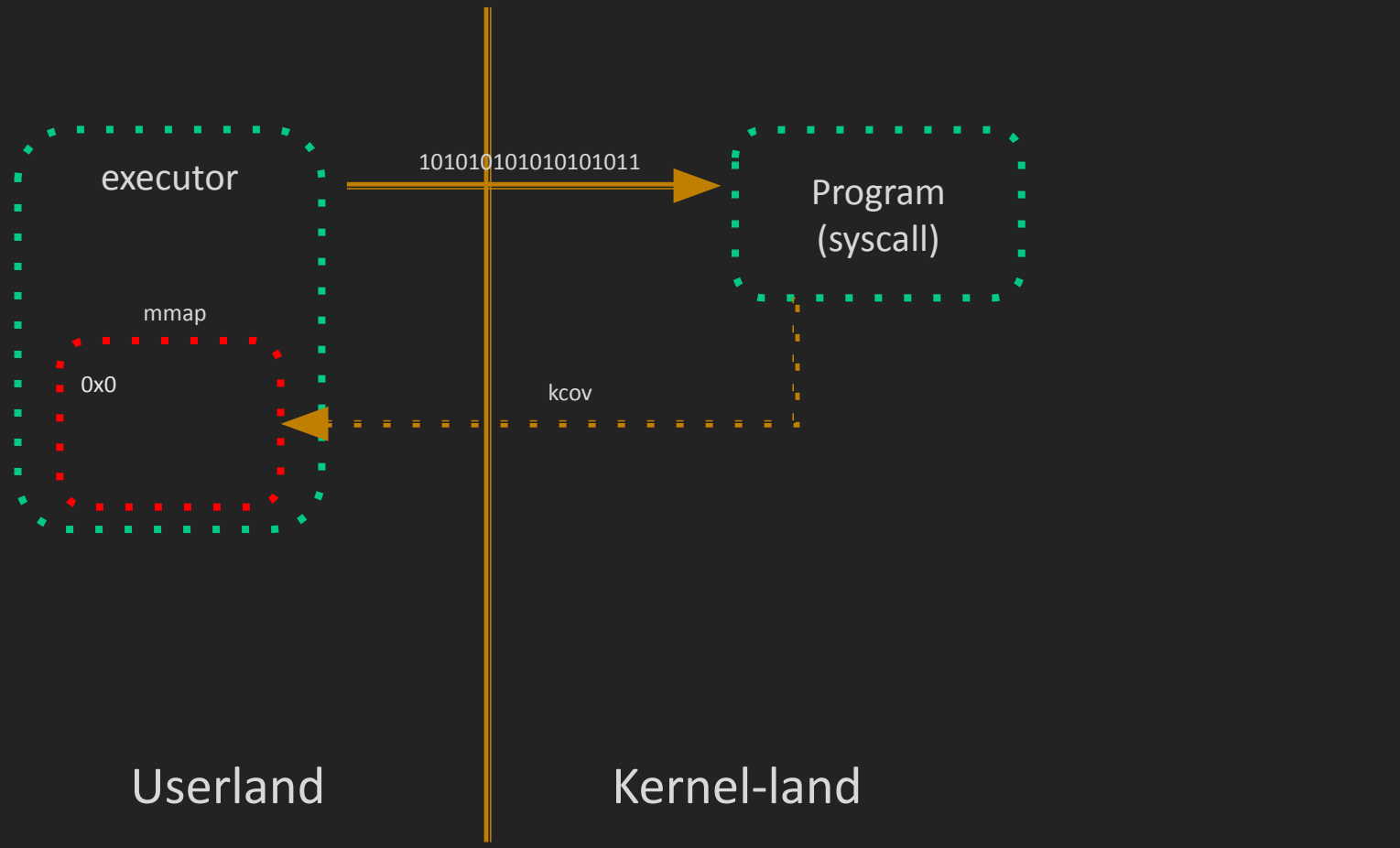
Userland

Kernel-land

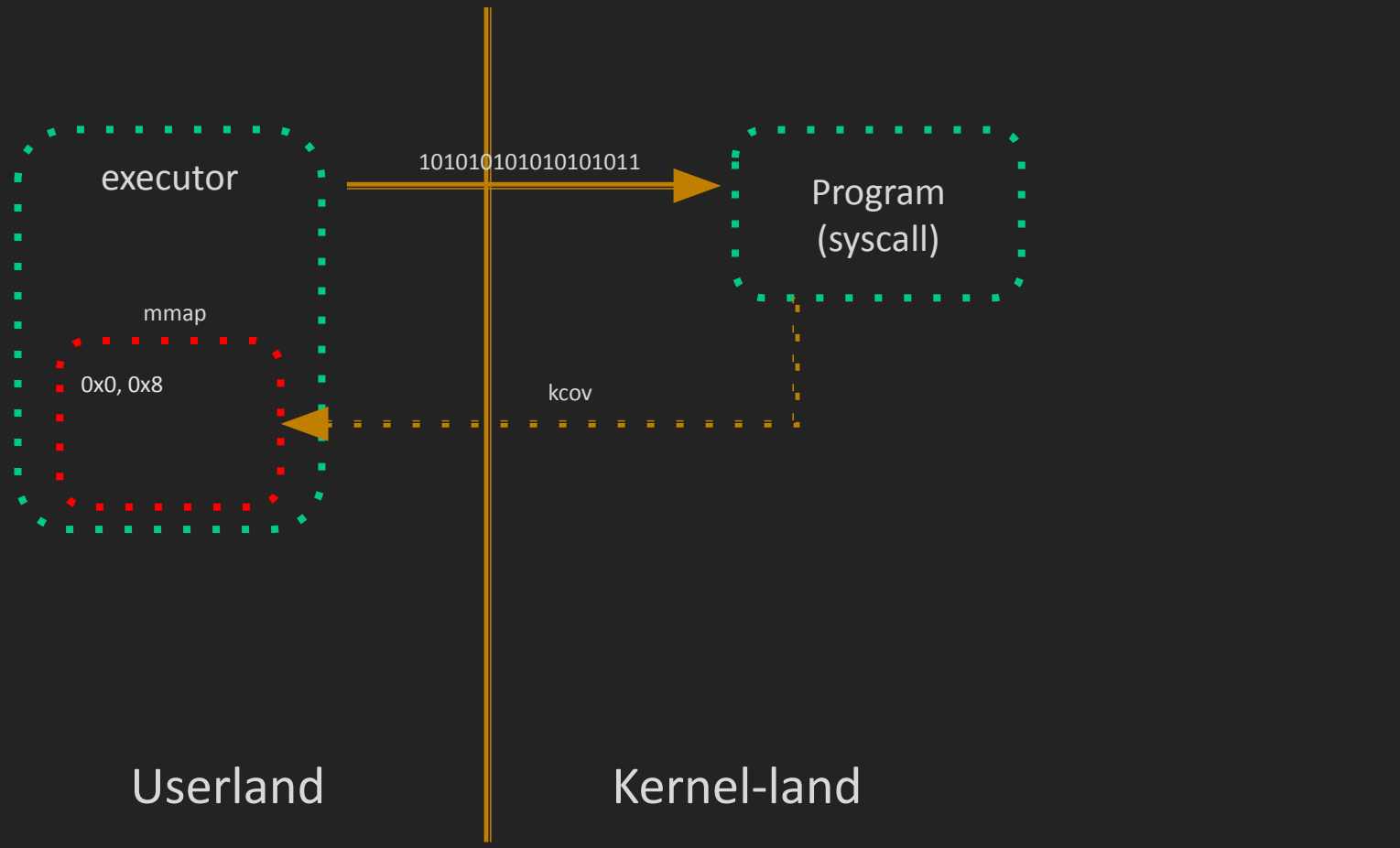
```
0x0: copy_from_user(...)
0x8: if(input)
0x10:       return
0x18:       if(input[0] == 1)
0x20:       return
```



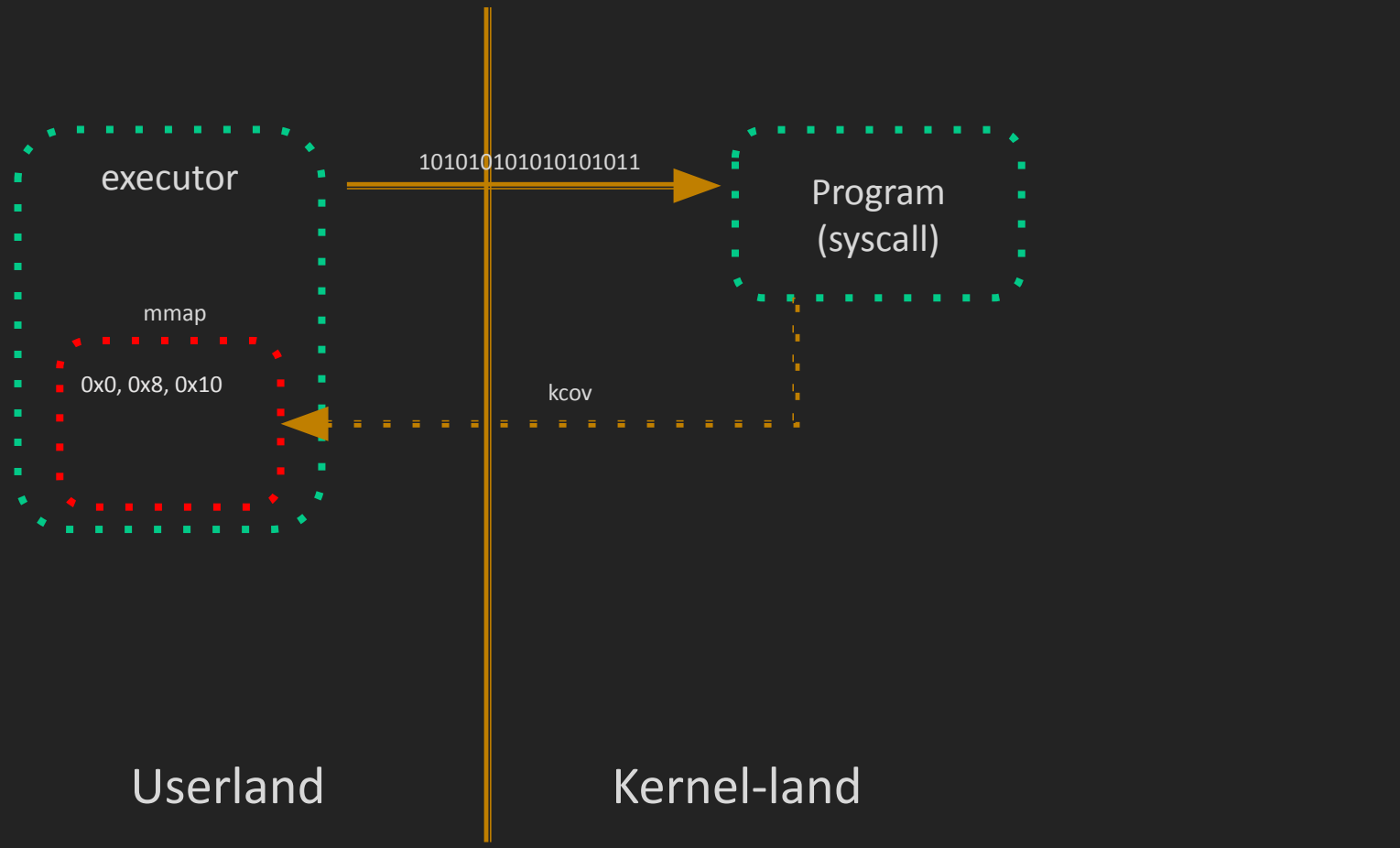
```
0x0:        copy_from_user(...)
0x8:  if(input)
0x10:       return
0x18:       if(input[0] == 1)
0x20:       return
```



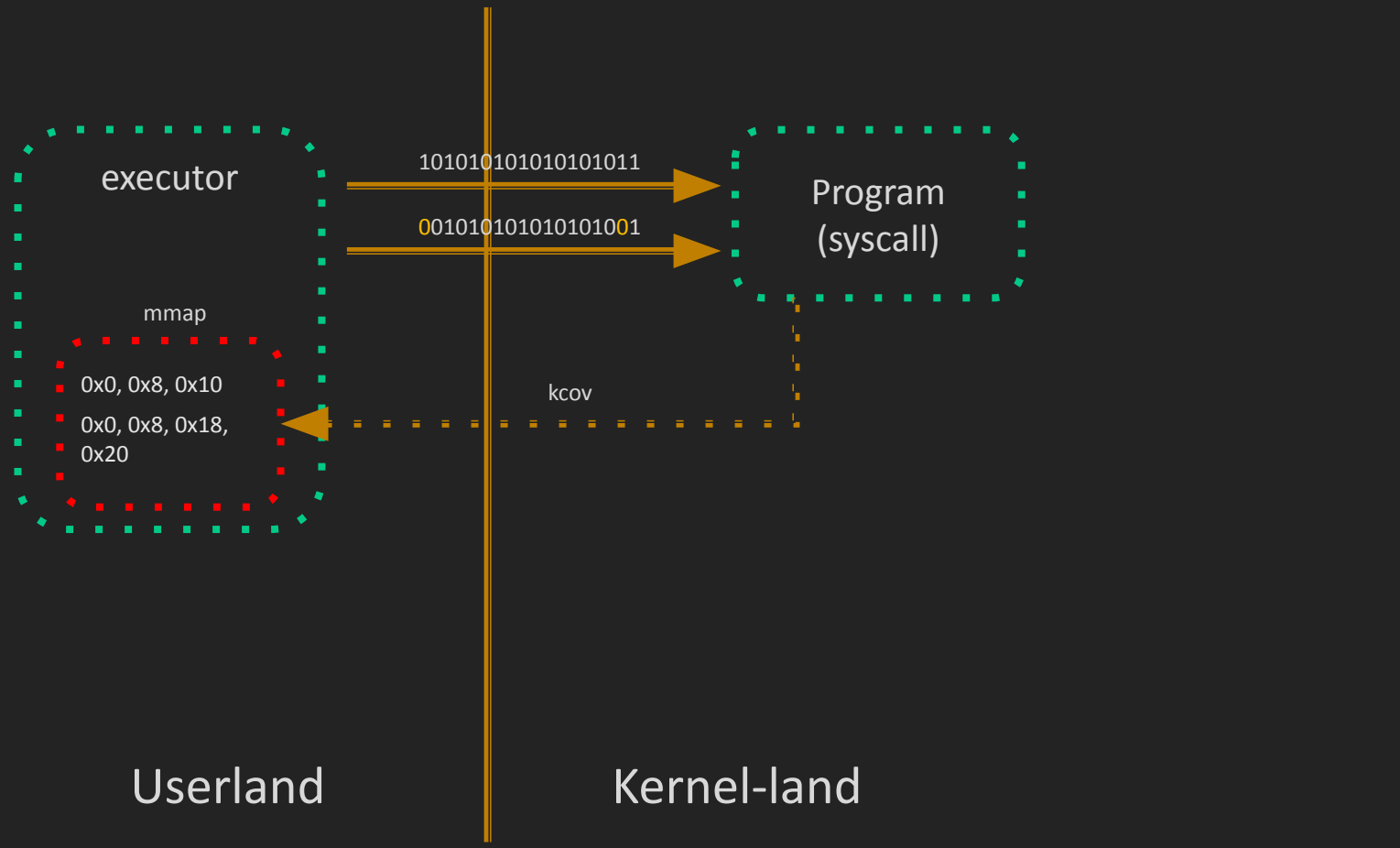
```
0x0: copy_from_user(...)
0x8:      if(input)
0x10:     return
0x18:     if(input[0] == 1)
0x20:     return
```



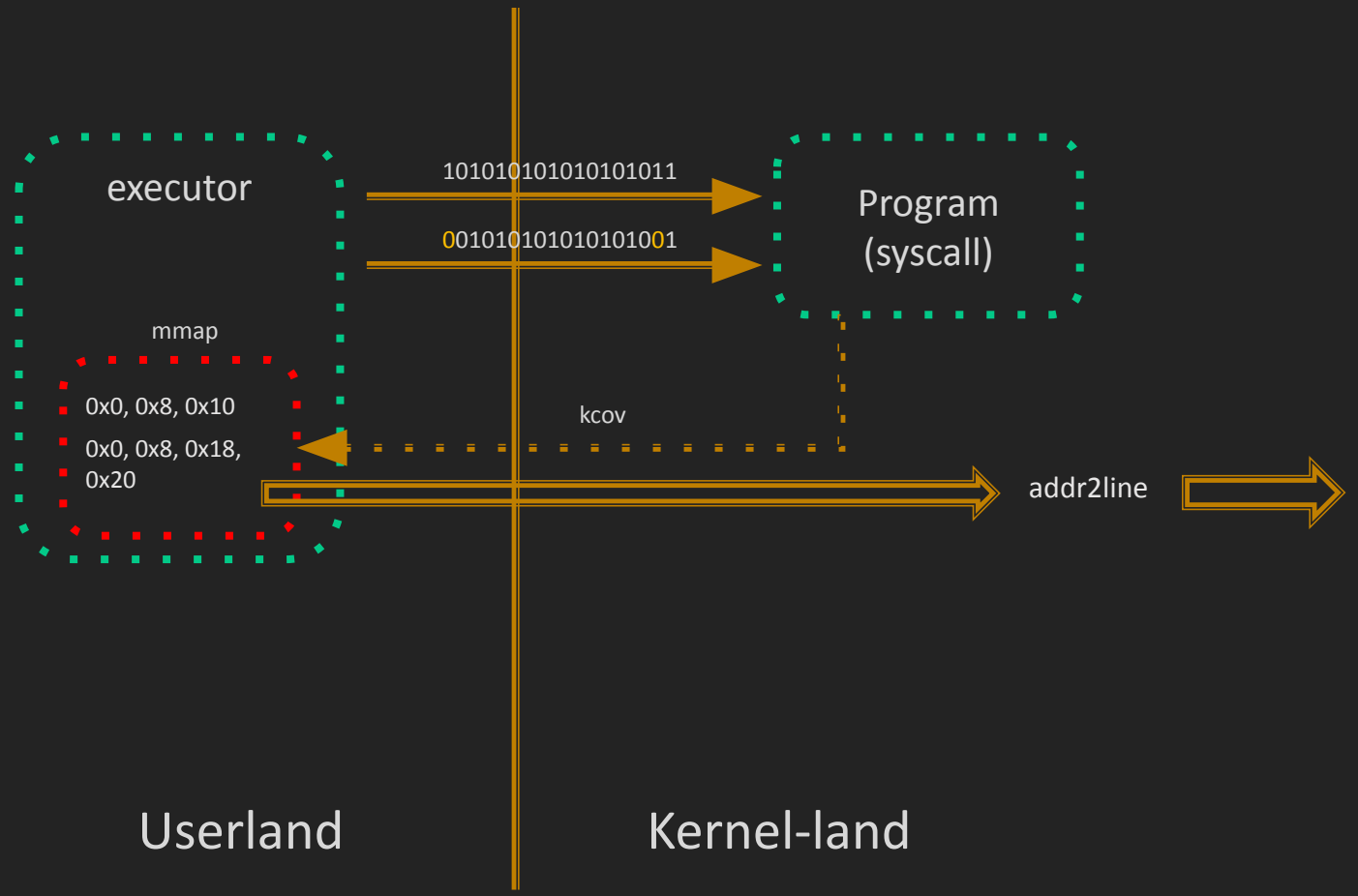
```
0x0: copy_from_user(...)
0x8: if(input)
0x10:       return
0x18:       if(input[0] == 1)
0x20:       return
```



```
0x0:      copy_from_user(...)
0x8:      if(input)
0x10:     return
0x18:     if(input[0] == 1)
0x20:     return
```



```
0x0:    copy_from_user(...)
0x8:    if(input)
0x10:   return
0x18:   if(input[0] == 1)
0x20:   return
```



```
/* init */
kcov_fd = open("/sys/kernel/debug/kcov", O_RDWR);
ioctl(kcov_fd, KCOV_INIT_TRACE, COVER_SIZE)
unsigned long *cover = mmap(NULL, COVER_SIZE * sizeof(unsigned long),
                            PROT_READ | PROT_WRITE, MAP_SHARED, kcov_fd, 0);

/* execute syscalls .. */

/* read */
n = __atomic_load_n(&cover[0], __ATOMIC_RELAXED);
for (i = 0; i < n; i++)
    printf("0x%lx\n", cover[i + 1]);

/* end */
if (ioctl(fd, KCOV_DISABLE, 0))
```

Complete example

Kernel ??

- A bug does not always crash
  - or it can crash in a different moment
    - harder to identify the root cause

How to solve this?

Kernel Sanitizers

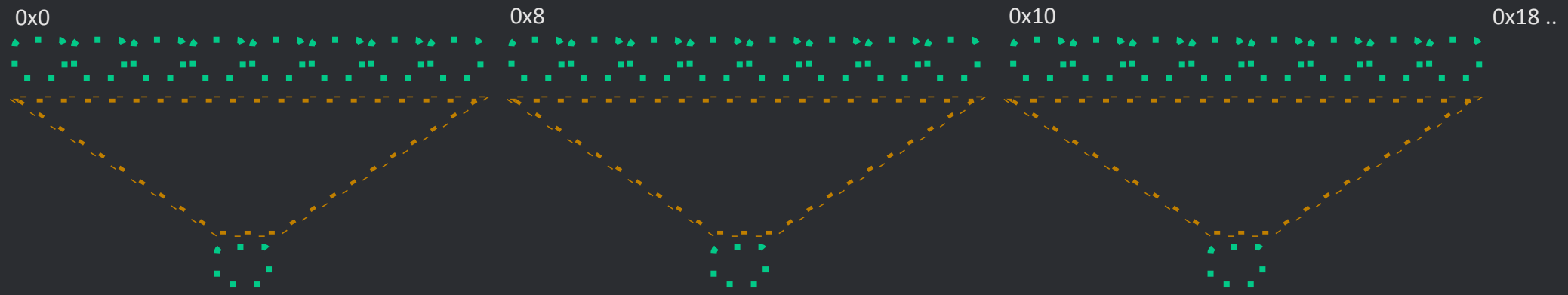
- A bug does not always crash
  - or it can crash in a different moment
    - harder to identify the root cause
- Sanitizers catch vulnerabilities when they happen!
  - KASAN – Kernel Address SANitizer
    - dynamic memory error detector (uaf, oob, double free, ..)
  - KCSAN - Kernel Concurrency SANitizer
    - dynamic race detector (race conditions)
  - KMSAN - Kernel Memory SANitizer
    - dynamic error detector (uninitialized memory access and userspace information leak)
  - UBSAN - UndefinedBehaviorSANitizer
    - undefined behavior detector (array oob, shift oob, ..)
  - ..

KASAN

- KASAN verifies every memory access and report invalid access
- Identify multiple bug classes
  - use-after free, double/invalid free, out of bounds, invalid memory access
- Enabled at compile time – CONFIG_KASAN
  - insert redzones for stack and global variables
  - verify each memory access, with compile time instrumentation, against a known memory state (shadow memory)
    - kmalloc(..) □ kmalloc_trace(..) □ kasan_kmalloc(..)
    - kfree(..) □ .. □ kasan_slab_free(..)
- Memory (1/8) and performance overhead

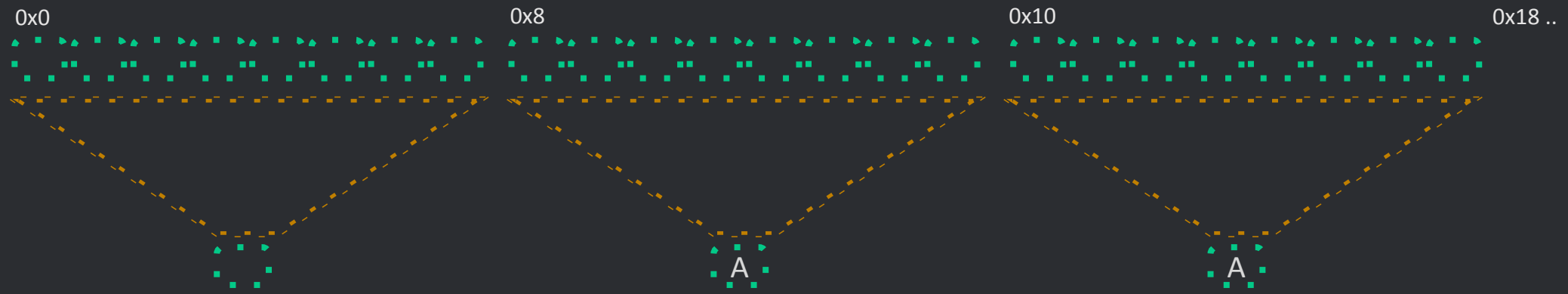
KASAN Shadow Memory

The state of each 8 aligned bytes of memory is encoded in one shadow byte.



KASAN Shadow Memory

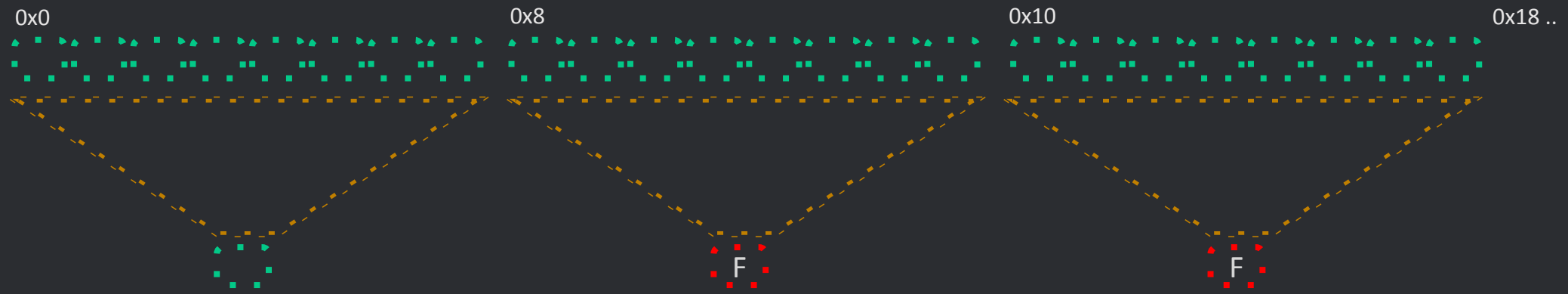
The state of each 8 aligned bytes of memory is encoded in one shadow byte.



kmalloc(0x10) => 0x8

KASAN Shadow Memory

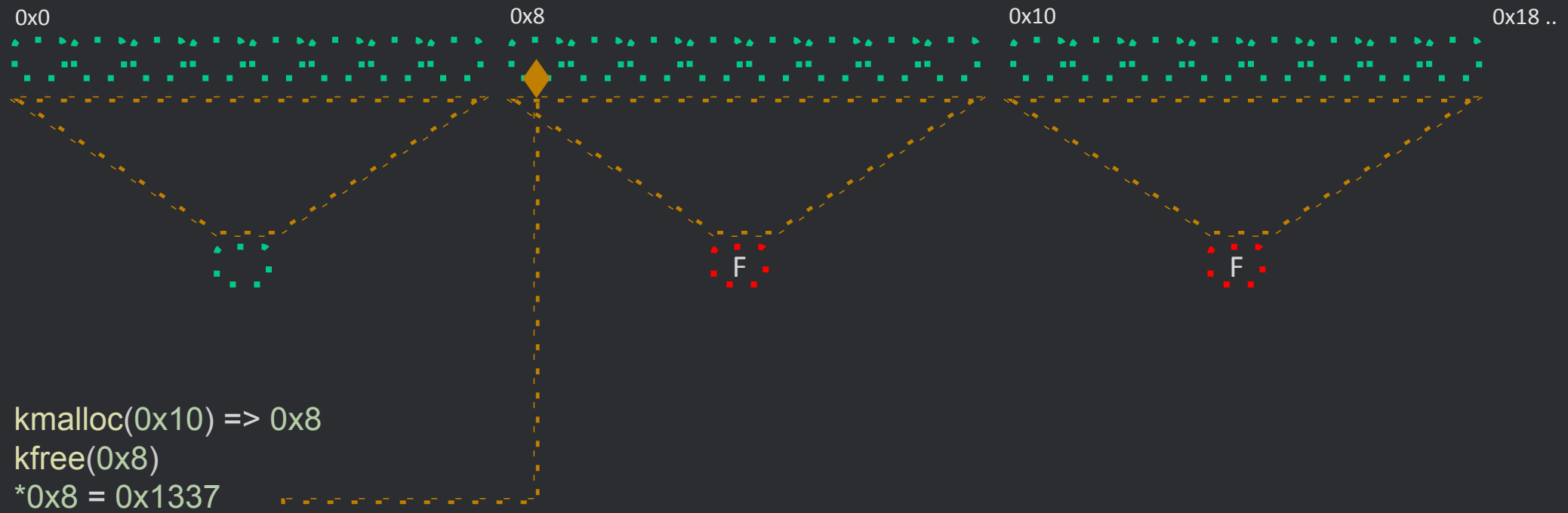
The state of each 8 aligned bytes of memory is encoded in one shadow byte.



kmalloc(0x10) => 0x8
kfree(0x8)

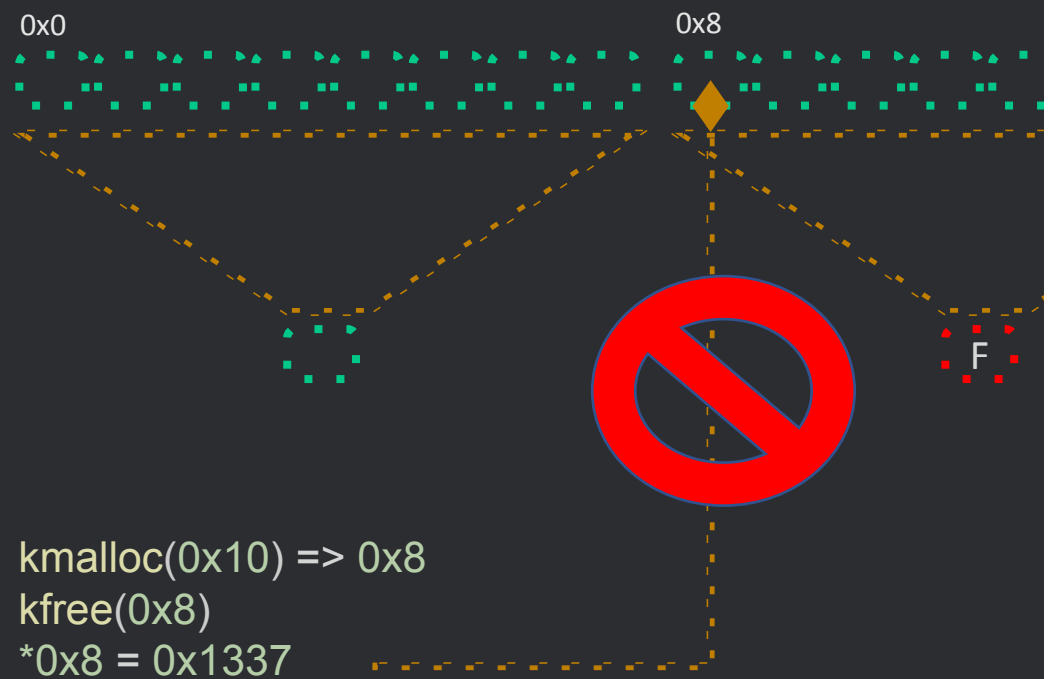
KASAN Shadow Memory

The state of each 8 aligned bytes of memory is encoded in one shadow byte.



KASAN Shadow Memory

The state of each 8 aligned bytes of memory

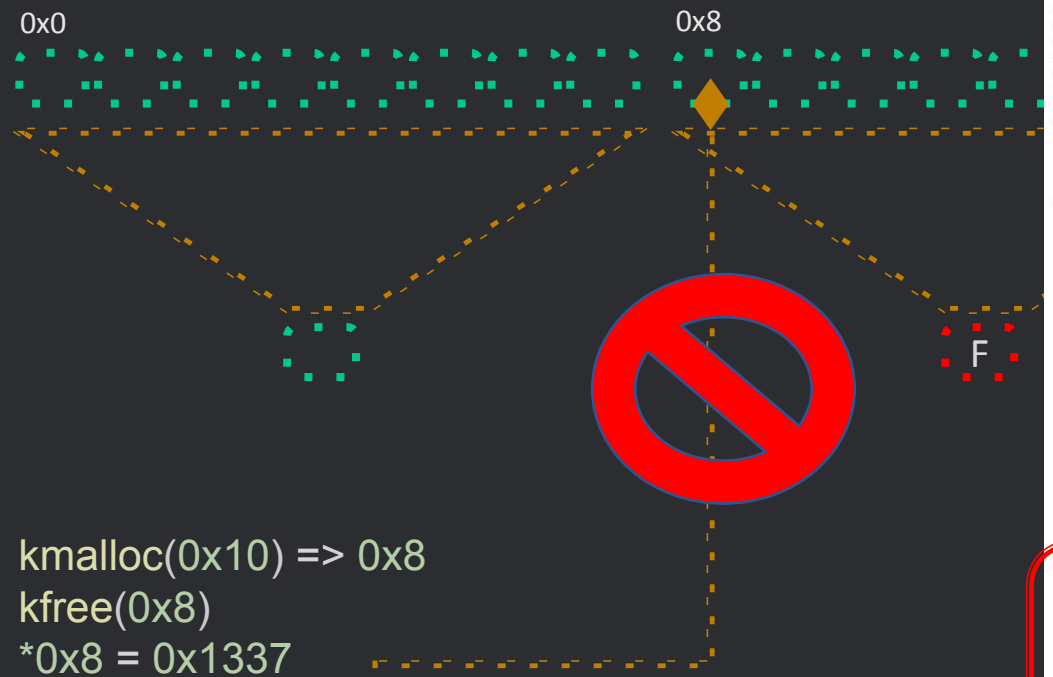


kmalloc(0x10) => 0x8
kfree(0x8)
*0x8 = 0x1337

```
==================================================================
BUG: AddressSanitizer: out of bounds access in kmalloc_oob_right+0x65/0x75 [test_kasan] at addr ffff8800693bc5d3
Write of size 1 by task modprobe/1689
==================================================================================
BUG kmalloc-128 (Not tainted): kasan error
----------------------------------------------------------------------------------

Disabling lock debugging due to kernel taint
INFO: Allocated in kmalloc_oob_right+0x3d/0x75 [test_kasan] age=0 cpu=0 pid=1689
 __slab_alloc+0x4b4/0x4f0
 kmem_cache_alloc_trace+0x10b/0x190
 kmalloc_oob_right+0x3d/0x75 [test_kasan]
 init_module+0x9/0x47 [test_kasan]
 do_one_initcall+0x99/0x200
 load_module+0x2cb3/0x3b20
 SyS_finit_module+0x76/0x80
 system_call_fastpath+0x12/0x17
INFO: Slab 0xffffea0001a4ef00 objects=17 used=7 fp=0xffff8800693bd728 flags=0x100000000004080
INFO: Object 0xffff8800693bc558 @offset=1368 fp=0xffff8800693bc720

Bytes b4 ffff8800693bc548: 00 00 00 00 00 00 00 00 5a 5a 5a 5a 5a 5a 5a 5a  ........ZZZZZZZZ
Object ffff8800693bc558: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc568: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc578: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc588: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc598: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5a8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5b8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5c8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b a5  kkkkkkkkkkkkkkk.
Redzone ffff8800693bc5d8: cc cc cc cc cc cc cc cc                          ........
Padding ffff8800693bc718: 5a 5a 5a 5a 5a 5a 5a 5a                          ZZZZZZZZ
CPU: 0 PID: 1689 Comm: modprobe Tainted: G    B          3.18.0-rc1-mm1+ #98
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.7.5-0-ge51488c-20140602_164612-nilsson.home.kraxel.org 04/01/2014
 ffff8800693bc000 0000000000000000 ffff8800693bc558 ffff88006923bb78
 ffffffff81cc68ae 0000000000000f3 ffff88006d407600 ffff88006923bba8
 ffffffff811fd848 ffff88006d407600 ffffea0001a4ef00 ffff8800693bc558
Call Trace:
 [<ffffffff81cc68ae>] dump_stack+0x46/0x58
 [<ffffffff811fd848>] print_trailer+0xf8/0x160
 [<ffffffffa00026a7>] ? kmem_cache_oob+0xc3/0xc3 [test_kasan]
 [<ffffffff811ff0f5>] object_err+0x35/0x40
 [<ffffffffa0002065>] ? kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffff8120b9fa>] kasan_report_error+0x38a/0x3f0
 [<ffffffff8120a79f>] ? kasan_poison_shadow+0x2f/0x40
 [<ffffffff8120b344>] ? kasan_unpoison_shadow+0x14/0x40
 [<ffffffff8120a79f>] ? kasan_poison_shadow+0x2f/0x40
 [<ffffffffa00026a7>] ? kmem_cache_oob+0xc3/0xc3 [test_kasan]
 [<ffffffff8120a995>] __asan_store1+0x75/0xb0
 [<ffffffffa0002601>] ? kmem_cache_oob+0x1d/0xc3 [test_kasan]
 [<ffffffffa0002065>] ? kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffffa0002065>] kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffffa00026b0>] init_module+0x9/0x47 [test_kasan]
 [<ffffffff810002d9>] do_one_initcall+0x99/0x200
 [<ffffffff811e4e5c>] ? __vunmap+0xec/0x160
 [<ffffffff81114f63>] load_module+0x2cb3/0x3b20
 [<ffffffff8110fd70>] ? m_show+0x240/0x240
 [<ffffffff81115f06>] SyS_finit_module+0x76/0x80
 [<ffffffff81cd3129>] system_call_fastpath+0x12/0x17
Memory state around the buggy address:
 ffff8800693bc300: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc380: fc fc 00 00 00 00 00 00 00 00 00 00 00 00 00 fc
 ffff8800693bc400: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc480: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc500: fc fc fc fc fc fc fc fc fc fc fc 00 00 00 00 00
>ffff8800693bc580: 00 00 00 00 00 00 00 00 00 00 03 fc fc fc fc fc
                                                 ^
 ffff8800693bc600: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc680: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc700: fc fc fc fc fb fb fb fb fb fb fb fb fb fb fb fb
 ffff8800693bc780: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
 ffff8800693bc800: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
==================================================================
```

KASAN Shadow Memory

The state of each 8 aligned bytes of memory



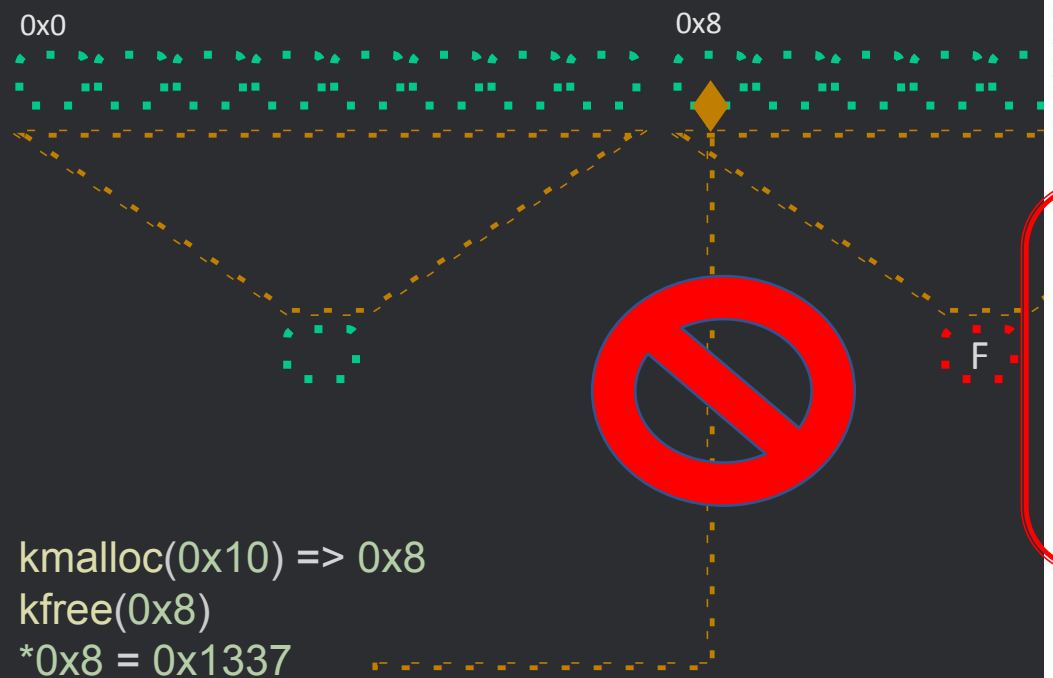
kmalloc(0x10) => 0x8
kfree(0x8)
*0x8 = 0x1337

```
==================================================================
BUG: AddressSanitizer: out of bounds access in kmalloc_oob_right+0x65/0x75 [test_kasan] at addr ffff8800693bc5d3
Write of size 1 by task modprobe/1689
==================================================================================
BUG kmalloc-128 (Not tainted): kasan error
----------------------------------------------------------------------------------

Disabling lock debugging due to kernel taint
INFO: Allocated in kmalloc_oob_right+0x3d/0x75 [test_kasan] age=0 cpu=0 pid=1689
 __slab_alloc+0x4b4/0x4f0
 kmem_cache_alloc_trace+0x10b/0x190
 kmalloc_oob_right+0x3d/0x75 [test_kasan]
 init_module+0x9/0x47 [test_kasan]
 do_one_initcall+0x99/0x200
 load_module+0x2cb3/0x3b20
 SyS_finit_module+0x76/0x80
 system_call_fastpath+0x12/0x17
INFO: Slab 0xffffea0001a4ef00 objects=17 used=7 fp=0xffff8800693bd728 flags=0x100000000004080
INFO: Object 0xffff8800693bc558 @offset=1368 fp=0xffff8800693bc720

Bytes b4 ffff8800693bc548: 00 00 00 00 00 00 00 00 5a 5a 5a 5a 5a 5a 5a 5a  ........ZZZZZZZZ
Object ffff8800693bc558: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc568: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc578: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc588: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc598: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5a8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5b8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5c8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b a5  kkkkkkkkkkkkkkk.
Redzone ffff8800693bc5d8: cc cc cc cc cc cc cc cc                          ........
Padding ffff8800693bc718: 5a 5a 5a 5a 5a 5a 5a 5a                          ZZZZZZZZ
CPU: 0 PID: 1689 Comm: modprobe Tainted: G    B          3.18.0-rc1-mm1+ #98
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.7.5-0-ge51488c-20140602_164612-nilsson.home.kraxel.org 04/01/2014
 ffff8800693bc000 0000000000000000 ffff8800693bc558 ffff88006923bb78
 ffffffff81cc68ae 00000000000000f3 ffff88006d407600 ffff88006923bba8
 ffffffff811fd848 ffff88006d407600 ffffea0001a4ef00 ffff8800693bc558
Call Trace:
 [<ffffffff81cc68ae>] dump_stack+0x46/0x58
 [<ffffffff811fd848>] print_trailer+0xf8/0x160
 [<ffffffffa00026a7>] ? kmem_cache_oob+0xc3/0xc3 [test_kasan]
 [<ffffffff811ff0f5>] object_err+0x35/0x40
 [<ffffffffa0002065>] ? kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffff8120b9fa>] kasan_report_error+0x38a/0x3f0
 [<ffffffff8120a79f>] ? kasan_poison_shadow+0x2f/0x40
 [<ffffffff8120b344>] ? kasan_unpoison_shadow+0x14/0x40
 [<ffffffff8120a79f>] ? kasan_poison_shadow+0x2f/0x40
 [<ffffffffa00026a7>] ? kmem_cache_oob+0xc3/0xc3 [test_kasan]
 [<ffffffff8120a995>] __asan_store1+0x75/0xb0
 [<ffffffffa0002601>] ? kmem_cache_oob+0x1d/0xc3 [test_kasan]
 [<ffffffffa0002065>] ? kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffffa0002065>] kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffffa00026b0>] init_module+0x9/0x47 [test_kasan]
 [<ffffffff810002d9>] do_one_initcall+0x99/0x200
 [<ffffffff811e4e5c>] ? __vunmap+0xec/0x160
 [<ffffffff81114f63>] load_module+0x2cb3/0x3b20
 [<ffffffff8110fd70>] ? m_show+0x240/0x240
 [<ffffffff81115f06>] SyS_finit_module+0x76/0x80
 [<ffffffff81cd2129>] system_call_fastpath+0x12/0x17
Memory state around the buggy address:
 ffff8800693bc300: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc380: fc fc 00 00 00 00 00 00 00 00 00 00 00 00 00 fc
 ffff8800693bc400: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc480: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc500: fc fc fc fc fc fc fc fc fc fc fc 00 00 00 00 00
>ffff8800693bc580: 00 00 00 00 00 00 00 00 00 00 03 fc fc fc fc fc
                                                 ^
 ffff8800693bc600: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc680: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc700: fc fc fc fc fc fb fb fb fb fb fb fb fb fb fb fb
 ffff8800693bc780: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
 ffff8800693bc800: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
==================================================================
```

hacktive security

KASAN Shadow Memory

The state of each 8 aligned bytes of memory

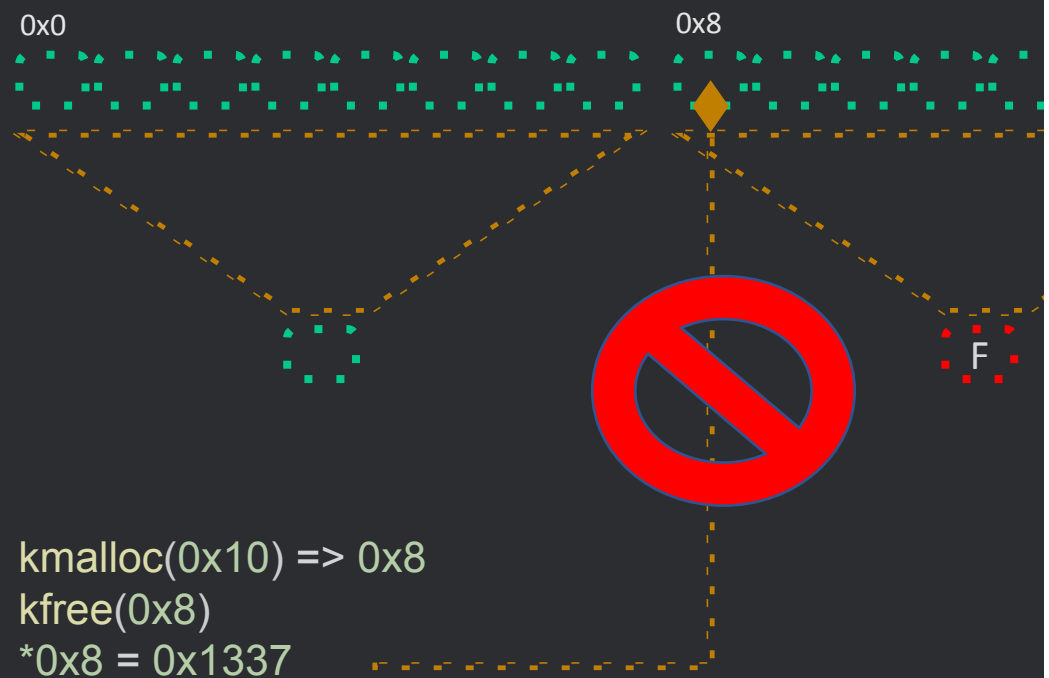


kmalloc(0x10) => 0x8
kfree(0x8)
*0x8 = 0x1337

```
==================================================================
BUG: AddressSanitizer: out of bounds access in kmalloc_oob_right+0x65/0x75 [test_kasan] at addr ffff8800693bc5d3
Write of size 1 by task modprobe/1689
==================================================================
BUG kmalloc-128 (Not tainted): kasan error
------------------------------------------------------------------------------

Disabling lock debugging due to kernel taint
INFO: Allocated in kmalloc_oob_right+0x3d/0x75 [test_kasan] age=0 cpu=0 pid=1689
 __slab_alloc+0x4b4/0x4f0
 kmem_cache_alloc_trace+0x10b/0x190
 kmalloc_oob_right+0x3d/0x75 [test_kasan]
 init_module+0x9/0x47 [test_kasan]
 do_one_initcall+0x99/0x200
 load_module+0x2cb3/0x3b20
 SyS_finit_module+0x76/0x80
 system_call_fastpath+0x12/0x17
INFO: Slab 0xffffea0001a4ef00 objects=17 used=7 fp=0xffff8800693bd728 flags=0x100000000004080
INFO: Object 0xffff8800693bc558 @offset=1368 fp=0xffff8800693bc720

Bytes b4 ffff8800693bc548: 00 00 00 00 00 00 00 00 5a 5a 5a 5a 5a 5a 5a 5a  ........ZZZZZZZZ
Object ffff8800693bc558: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc568: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc578: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc588: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc598: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5a8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5b8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5c8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b a5  kkkkkkkkkkkkkkk.
Redzone ffff8800693bc5d8: cc cc cc cc cc cc cc cc                          ........
Padding ffff8800693bc718: 5a 5a 5a 5a 5a 5a 5a 5a                          ZZZZZZZZ
CPU: 0 PID: 1689 Comm: modprobe Tainted: G    B          3.18.0-rc1-mm1+ #98
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.7.5-0-ge51488c-20140602_164612-nilsson.home.kraxel.org 04/01/2014
 ffff8800693bc000 0000000000000000 ffff8800693bc558 ffff88006923bb78
 ffffffff81cc68ae 00000000000000f3 ffff88006d407600 ffff88006923bba8
 ffffffff811fd848 ffff88006d407600 ffffea0001a4ef00 ffff8800693bc5d3
Call Trace:
 [<ffffffff81cc68ae>] dump_stack+0x46/0x58
 [<ffffffff811fd848>] print_trailer+0xf8/0x160
 [<ffffffffa00026a7>] ? kmem_cache_oob+0xc3/0xc3 [test_kasan]
 [<ffffffff811ff0f5>] object_err+0x35/0x40
 [<ffffffffa0002065>] ? kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffff8120b9fa>] kasan_report_error+0x38a/0x3f0
 [<ffffffff8120a79f>] ? kasan_poison_shadow+0x2f/0x40
 [<ffffffff8120b344>] ? kasan_unpoison_shadow+0x14/0x40
 [<ffffffff8120a79f>] ? kasan_poison_shadow+0x2f/0x40
 [<ffffffffa00026a7>] ? kmem_cache_oob+0xc3/0xc3 [test_kasan]
 [<ffffffff8120a995>] __asan_store1+0x75/0xb0
 [<ffffffffa0002601>] ? kmem_cache_oob+0x1d/0xc3 [test_kasan]
 [<ffffffffa0002065>] ? kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffffa0002065>] kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffffa00026b0>] init_module+0x9/0x47 [test_kasan]
 [<ffffffff810002d9>] do_one_initcall+0x99/0x200
 [<ffffffff811e4e5c>] ? __vunmap+0xec/0x160
 [<ffffffff81114f63>] load_module+0x2cb3/0x3b20
 [<ffffffff8110fd70>] ? m_show+0x240/0x240
 [<ffffffff81115f06>] SyS_finit_module+0x76/0x80
 [<ffffffff81cd3129>] system_call_fastpath+0x12/0x17
Memory state around the buggy address:
 ffff8800693bc300: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc380: fc fc 00 00 00 00 00 00 00 00 00 00 00 00 00 fc
 ffff8800693bc400: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc480: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc500: fc fc fc fc fc fc fc fc fc fc fc 00 00 00 00 00
>ffff8800693bc580: 00 00 00 00 00 00 00 00 00 00 03 fc fc fc fc fc
                                                 ^
 ffff8800693bc600: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc680: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc700: fc fc fc fc fb fb fb fb fb fb fb fb fb fb fb fb
 ffff8800693bc780: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
 ffff8800693bc800: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
==================================================================
```

KASAN Shadow Memory

The state of each 8 aligned bytes of memory

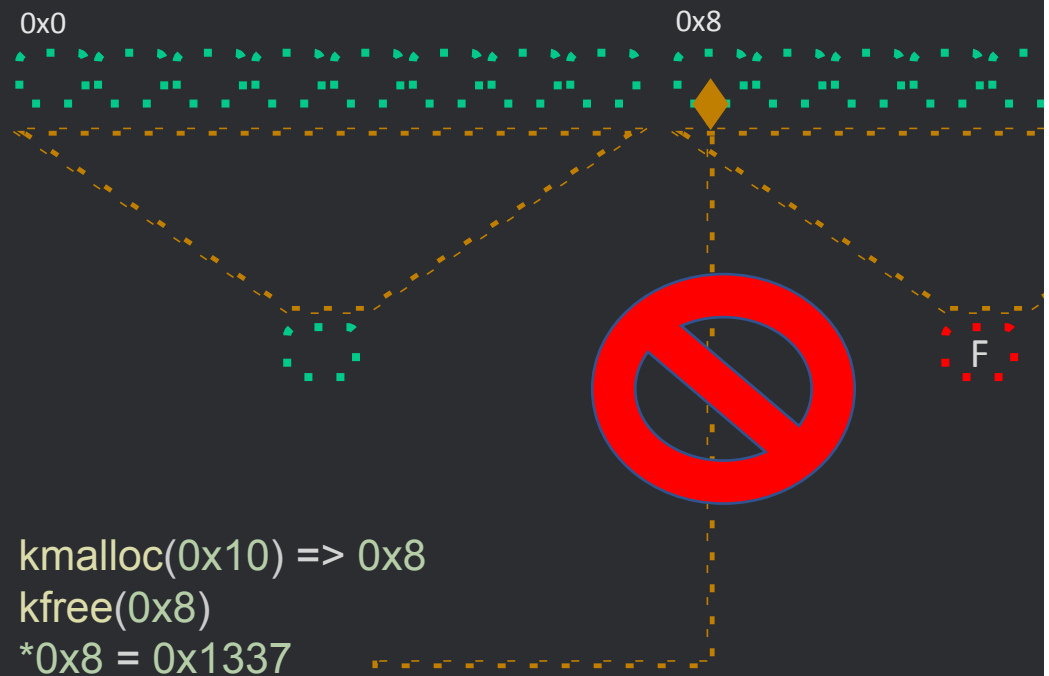


kmalloc(0x10) => 0x8
kfree(0x8)
*0x8 = 0x1337

```
==================================================================
BUG: AddressSanitizer: out of bounds access in kmalloc_oob_right+0x65/0x75 [test_kasan] at addr ffff8800693bc5d3
Write of size 1 by task modprobe/1689
==================================================================================
BUG kmalloc-128 (Not tainted): kasan error
----------------------------------------------------------------------------------

Disabling lock debugging due to kernel taint
INFO: Allocated in kmalloc_oob_right+0x3d/0x75 [test_kasan] age=0 cpu=0 pid=1689
 __slab_alloc+0x4b4/0x4f0
 kmem_cache_alloc_trace+0x10b/0x190
 kmalloc_oob_right+0x3d/0x75 [test_kasan]
 init_module+0x9/0x47 [test_kasan]
 do_one_initcall+0x99/0x200
 load_module+0x2cb3/0x3b20
 SyS_finit_module+0x76/0x80
 system_call_fastpath+0x12/0x17
INFO: Slab 0xffffea0001a4ef00 objects=17 used=7 fp=0xffff8800693bd728 flags=0x100000000004080
INFO: Object 0xffff8800693bc558 @offset=1368 fp=0xffff8800693bc720

Bytes b4 ffff8800693bc548: 00 00 00 00 00 00 00 00 5a 5a 5a 5a 5a 5a 5a 5a  ........ZZZZZZZZ
Object ffff8800693bc558: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc568: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc578: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc588: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc598: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5a8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5b8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5c8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b a5  kkkkkkkkkkkkkkk.
Redzone ffff8800693bc5d8: cc cc cc cc cc cc cc cc                          ........
Padding ffff8800693bc718: 5a 5a 5a 5a 5a 5a 5a 5a                          ZZZZZZZZ
CPU: 0 PID: 1689 Comm: modprobe Tainted: G    B          3.18.0-rc1-mm1+ #98
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.7.5-0-ge51488c-20140602_164612-nilsson.home.kraxel.org 04/01/2014
 ffff8800693bc000 0000000000000000 ffff8800693bc558 ffff88006923bb78
 ffffffff81cc68ae 00000000000000f3 ffff88006d407600 ffff88006923bba8
 ffffffff811fd848 ffff88006d407600 ffffea0001a4ef00 ffff8800693bc558
Call Trace:
 [<ffffffff81cc68ae>] dump_stack+0x46/0x58
 [<ffffffff811fd848>] print_trailer+0xf8/0x160
 [<ffffffffa00026a7>] ? kmem_cache_oob+0xc3/0xc3 [test_kasan]
 [<ffffffff811ff0f5>] object_err+0x35/0x40
 [<ffffffffa0002065>] ? kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffff8120b9fa>] kasan_report_error+0x38a/0x3f0
 [<ffffffff8120a79f>] ? kasan_poison_shadow+0x2f/0x40
 [<ffffffff8120b344>] ? kasan_unpoison_shadow+0x14/0x40
 [<ffffffff8120a79f>] ? kasan_poison_shadow+0x2f/0x40
 [<ffffffffa00026a7>] ? kmem_cache_oob+0xc3/0xc3 [test_kasan]
 [<ffffffff8120a995>] __asan_store1+0x75/0xb0
 [<ffffffffa0002601>] ? kmem_cache_oob+0x1d/0xc3 [test_kasan]
 [<ffffffffa0002065>] ? kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffffa0002065>] kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffffa00026b0>] init_module+0x9/0x47 [test_kasan]
 [<ffffffff810002d9>] do_one_initcall+0x99/0x200
 [<ffffffff811e4e5c>] ? __vunmap+0xec/0x160
 [<ffffffff81114f63>] load_module+0x2cb3/0x3b20
 [<ffffffff8110fd70>] ? m_show+0x240/0x240
 [<ffffffff81115f06>] SyS_finit_module+0x76/0x80
 [<ffffffff81cd3129>] system_call_fastpath+0x12/0x17
Memory state around the buggy address:
 ffff8800693bc300: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc380: fc fc 00 00 00 00 00 00 00 00 00 00 00 00 00 fc
 ffff8800693bc400: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc480: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc500: fc fc fc fc fc fc fc fc fc fc fc 00 00 00 00 00
>ffff8800693bc580: 00 00 00 00 00 00 00 00 00 00 03 fc fc fc fc fc
                                                 ^
 ffff8800693bc600: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc680: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc700: fc fc fc fc fc fb fb fb fb fb fb fb fb fb fb fb
 ffff8800693bc780: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
 ffff8800693bc800: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
==================================================================
```

KASAN Shadow Memory

The state of each 8 aligned bytes of memory



kmalloc(0x10) => 0x8
kfree(0x8)
*0x8 = 0x1337

```
==================================================================
BUG: AddressSanitizer: out of bounds access in kmalloc_oob_right+0x65/0x75 [test_kasan] at addr ffff8800693bc5d3
Write of size 1 by task modprobe/1689
==================================================================================
BUG kmalloc-128 (Not tainted): kasan error
----------------------------------------------------------------------------------


Disabling lock debugging due to kernel taint
INFO: Allocated in kmalloc_oob_right+0x3d/0x75 [test_kasan] age=0 cpu=0 pid=1689
 __slab_alloc+0x4b4/0x4f0
 kmem_cache_alloc_trace+0x10b/0x190
 kmalloc_oob_right+0x3d/0x75 [test_kasan]
 init_module+0x9/0x47 [test_kasan]
 do_one_initcall+0x99/0x200
 load_module+0x2cb3/0x3b20
 SyS_finit_module+0x76/0x80
 system_call_fastpath+0x12/0x17
INFO: Slab 0xffffea0001a4ef00 objects=17 used=7 fp=0xffff8800693bd728 flags=0x100000000004080
INFO: Object 0xffff8800693bc558 @offset=1368 fp=0xffff8800693bc720

Bytes b4 ffff8800693bc548: 00 00 00 00 00 00 00 00 5a 5a 5a 5a 5a 5a 5a 5a  ........ZZZZZZZZ
Object ffff8800693bc558: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc568: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc578: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc588: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc598: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5a8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5b8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  kkkkkkkkkkkkkkkk
Object ffff8800693bc5c8: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b a5  kkkkkkkkkkkkkkk.
Redzone ffff8800693bc5d8: cc cc cc cc cc cc cc cc                          ........
Padding ffff8800693bc718: 5a 5a 5a 5a 5a 5a 5a 5a                          ZZZZZZZZ
CPU: 0 PID: 1689 Comm: modprobe Tainted: G    B          3.18.0-rc1-mm1+ #98
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.7.5-0-ge51488c-20140602_164612-nilsson.home.kraxel.org 04/01/2014
 ffff8800693bc000 0000000000000000 ffff8800693bc558 ffff88006923bb78
 ffffffff81cc68ae 00000000000000f3 ffff88006d407600 ffff88006923bba8
 ffffffff811fd848 ffff88006d407600 ffffea0001a4ef00 ffff8800693bc558
Call Trace:
 [<ffffffff81cc68ae>] dump_stack+0x46/0x58
 [<ffffffff811fd848>] print_trailer+0xf8/0x160
 [<ffffffffa00026a7>] ? kmem_cache_oob+0xc3/0xc3 [test_kasan]
 [<ffffffff811ff0f5>] object_err+0x35/0x40
 [<ffffffffa0002065>] ? kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffff8120b9fa>] kasan_report_error+0x38a/0x3f0
 [<ffffffff8120a79f>] ? kasan_poison_shadow+0x2f/0x40
 [<ffffffff8120b344>] ? kasan_unpoison_shadow+0x14/0x40
 [<ffffffff8120a79f>] ? kasan_poison_shadow+0x2f/0x40
 [<ffffffffa00026a7>] ? kmem_cache_oob+0xc3/0xc3 [test_kasan]
 [<ffffffff8120a995>] __asan_store1+0x75/0xb0
 [<ffffffffa0002601>] ? kmem_cache_oob+0x1d/0xc3 [test_kasan]
 [<ffffffffa0002065>] ? kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffffa0002065>] kmalloc_oob_right+0x65/0x75 [test_kasan]
 [<ffffffffa00026b0>] init_module+0x9/0x47 [test_kasan]
 [<ffffffff810002d9>] do_one_initcall+0x99/0x200
 [<ffffffff811e4e5c>] ? __vunmap+0xec/0x160
 [<ffffffff81114f63>] load_module+0x2cb3/0x3b20
 [<ffffffff8110fd70>] ? m_show+0x240/0x240
 [<ffffffff81115f06>] SyS_finit_module+0x76/0x80
 [<ffffffff81cd3129>] system_call_fastpath+0x12/0x17
Memory state around the buggy address:
 ffff8800693bc300: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc380: fc fc 00 00 00 00 00 00 00 00 00 00 00 00 00 fc
 ffff8800693bc400: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc480: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc500: fc fc fc fc fc fc fc fc fc fc fc 00 00 00 00 00
>ffff8800693bc580: 00 00 00 00 00 00 00 00 00 00 03 fc fc fc fc fc
                                                 ^
 ffff8800693bc600: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc680: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffff8800693bc700: fc fc fc fc fb fb fb fb fb fb fb fb fb fb fb fb
 ffff8800693bc780: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
 ffff8800693bc800: fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb fb
==================================================================
```

Syzkaller

 # Introduction & Architecture

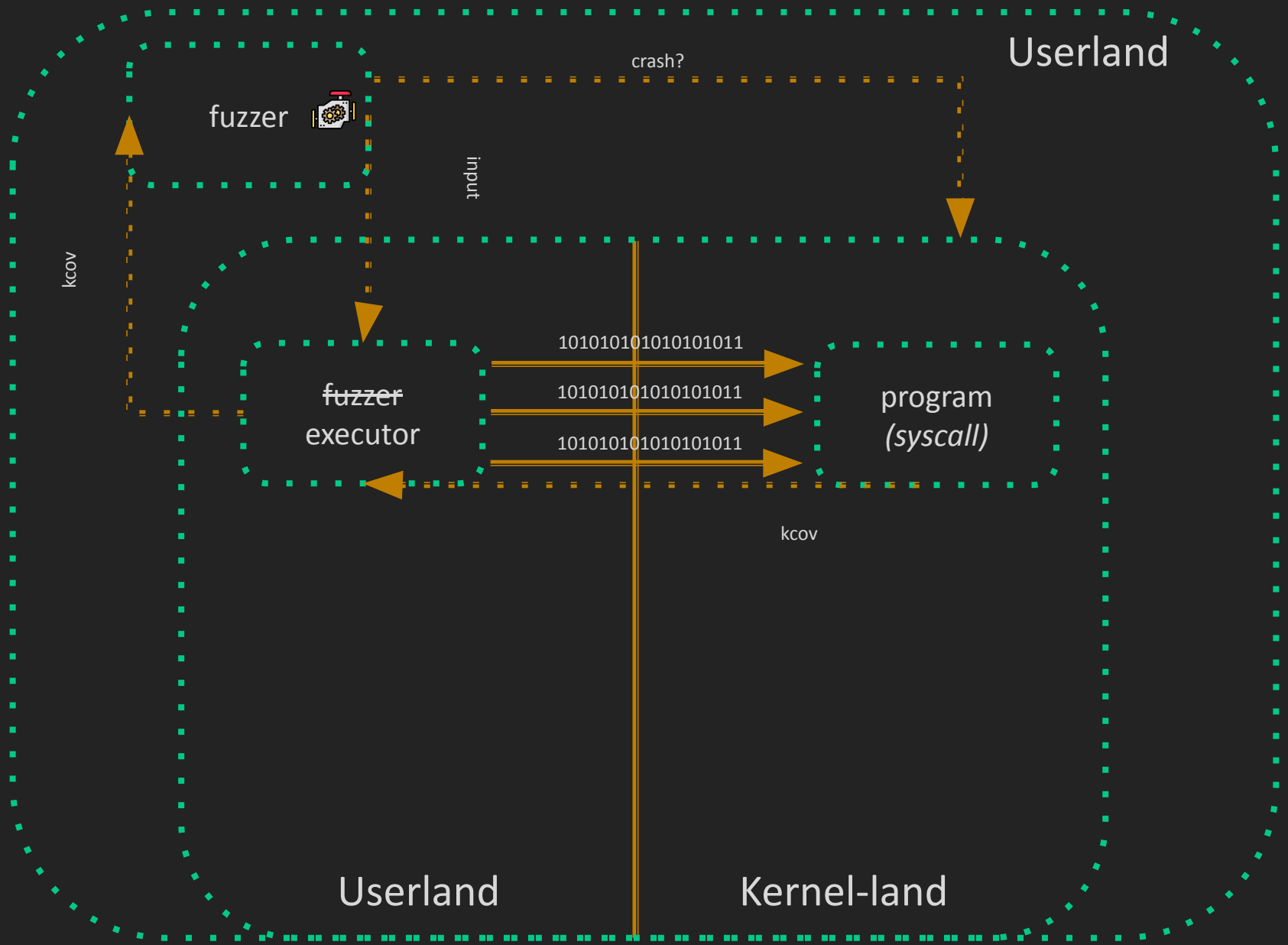
 # Syzbot

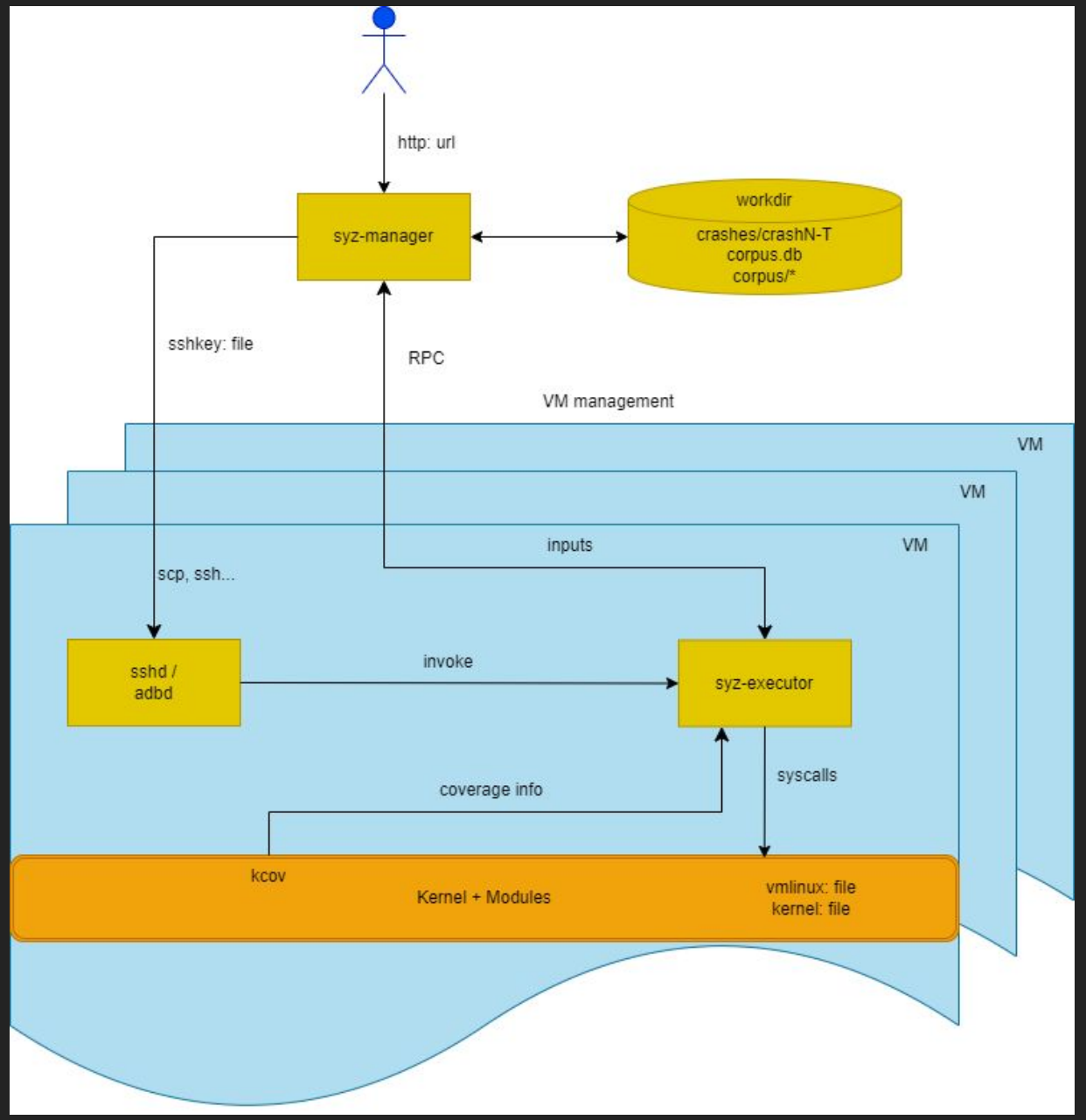
 # First dry-run

hacktive security

Syzkaller Introduction

- Syzkaller is an unsupervised coverage-guided kernel fuzzer
  - written in golang/C++/C
- A google open-source project since 2015
  - github repository
- Syzlang syntax language
  - grammar description of syscalls
- Support multiple OSes
  - Linux, XNU, Windows, FreeBSD, NetBSD, OpenBSD

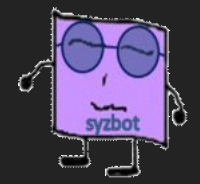

Userland
crash?
fuzzer
input
kcov
101010101010101011
101010101010101011
101010101010101011
~~fuzzer~~
executor
program
(syscall)
kcov
Userland
Kernel-land


http: url
syz-manager
workdir
crashes/crashN-T
corpus.db
corpus/*
sshkey: file
RPC
VM management
VM
VM
VM
inputs
scp, ssh...
sshd /
adbd
invoke
syz-executor
syscalls
coverage info
kcov
Kernel + Modules
vmlinux: file
kernel: file

Syzkaller targets

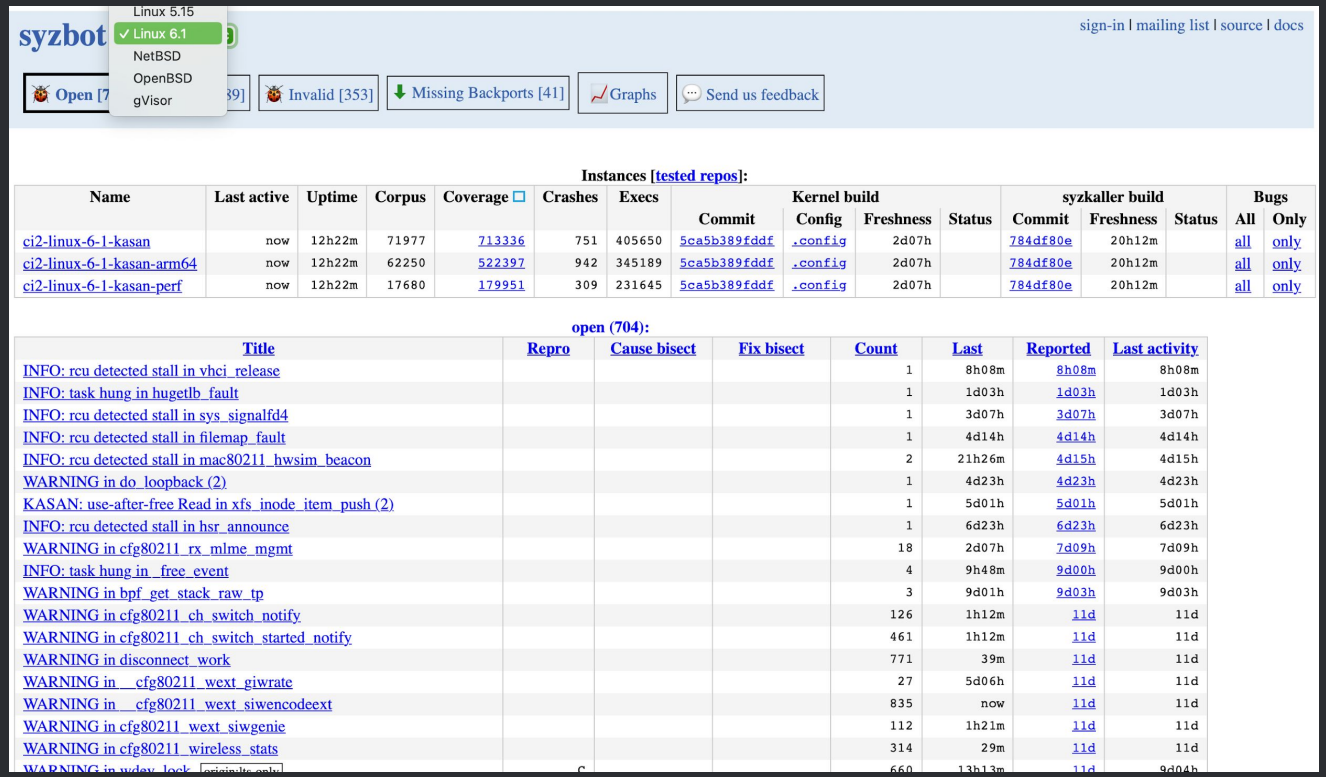
- Support multiple targets (just need SSH/adb/console access)
  - virtualization/emulation (qemu, vmware, ..)
  - physical device (android device, laptop, ..)
  - *(optional) serial access*

Syzbot



- Continuously fuzz the linux kernel since 2017
  - automatic reports
  - thousands of bugs found 

  - a lot of running instances
  - it’s public



syzbot — Linux 5.15 / Linux 6.1 / NetBSD / OpenBSD / gVisor — sign-in | mailing list | source | docs

Open [7 … 89] | Invalid [353] | Missing Backports [41] | Graphs | Send us feedback

Instances [tested repos]:

Name	Last active	Uptime	Corpus	Coverage	Crashes	Execs	Kernel build				syzkaller build			Bugs	
							Commit	Config	Freshness	Status	Commit	Freshness	Status	All	Only
ci2-linux-6-1-kasan	now	12h22m	71977	713336	751	405650	5ca5b389fddf	.config	2d07h		784df80e	20h12m		all	only
ci2-linux-6-1-kasan-arm64	now	12h22m	62250	522397	942	345189	5ca5b389fddf	.config	2d07h		784df80e	20h12m		all	only
ci2-linux-6-1-kasan-perf	now	12h22m	17680	179951	309	231645	5ca5b389fddf	.config	2d07h		784df80e	20h12m		all	only

open (704):

Title	Repro	Cause bisect	Fix bisect	Count	Last	Reported	Last activity
INFO: rcu detected stall in vhci_release				1	8h08m	8h08m	8h08m
INFO: task hung in hugetlb_fault				1	1d03h	1d03h	1d03h
INFO: rcu detected stall in sys_signalfd4				1	3d07h	3d07h	3d07h
INFO: rcu detected stall in filemap_fault				1	4d14h	4d14h	4d14h
INFO: rcu detected stall in mac80211_hwsim_beacon				2	21h26m	4d15h	4d15h
WARNING in do_loopback (2)				1	4d23h	4d23h	4d23h
KASAN: use-after-free Read in xfs_inode_item_push (2)				1	5d01h	5d01h	5d01h
INFO: rcu detected stall in hsr_announce				1	6d23h	6d23h	6d23h
WARNING in cfg80211_rx_mlme_mgmt				18	2d07h	7d09h	7d09h
INFO: task hung in __free_event				4	9h48m	9d00h	9d00h
WARNING in bpf_get_stack_raw_tp				3	9d01h	9d03h	9d03h
WARNING in cfg80211_ch_switch_notify				126	1h12m	11d	11d
WARNING in cfg80211_ch_switch_started_notify				461	1h12m	11d	11d
WARNING in disconnect_work				771	39m	11d	11d
WARNING in __cfg80211_wext_giwrate				27	5d06h	11d	11d
WARNING in __cfg80211_wext_siwencodeext				835	now	11d	11d
WARNING in cfg80211_wext_siwgenie				112	1h21m	11d	11d
WARNING in cfg80211_wireless_stats				314	29m	11d	11d
WARNING in wdev_lock [originals only]		C		660	13h13m	11d	9d04h

Syzkaller setup #0

- «How to setup syzkaller» official doc

How to set up syzkaller

Generic instructions on how to set up Linux kernel fuzzing with syzkaller are below.

Instructions for a particular VM type or kernel architecture can be found on these pages:

- Setup: Ubuntu host, QEMU vm, x86-64 kernel
- Setup: Linux host, QEMU vm, arm64 kernel
- Setup: Linux host, QEMU vm, arm kernel
- Setup: Linux host, QEMU vm, riscv64 kernel
- Setup: Linux host, QEMU vm, s390x kernel
- Setup: Linux host, Android device, arm32/64 kernel
- Setup: Linux isolated host
- Setup: Ubuntu host, VMware vm, x86-64 kernel

Syzkaller setup #0

```
# kernel 6.10, image and syzkaller
git clone https://github.com/hacktivesec/beginner-kernel-exploitation-setup.git
cd beginner-kernel-exploitation-setup
./setup_syzkaller_demo.sh

# everything in ./working/
```

Syzkaller setup #1

```
# go
sudo apt update
sudo apt install golang-go

# syzkaller
git clone https://github.com/google/syzkaller
cd syzkaller
make install_prerequisites
make
## OR make TARGETARCH=amd64 TARGETOS=linux
```

Syzkaller setup #2

```
# manual kernel compilation
git clone --branch v6.2-rc8 git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git linux-main-6.2-rc8
sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev flex bison bc
cd linux-main-6.2-rc8
make CC=clang defconfig
make CC=clang kvm_guest.config
cat << EOF >> .config
CONFIG_KCOV=y
...
EOF
make CC=clang olddefconfig
make CC=clang -j $(nproc)
```

```
# Coverage collection.
CONFIG_KCOV=y

# Debug info for symbolization.
CONFIG_DEBUG_INFO=y
CONFIG_DEBUG_INFO_DWARF4=y

# Memory bug detector
CONFIG_KASAN=y
CONFIG_KASAN_INLINE=y

# Required for Debian image
CONFIG_CONFIGFS_FS=y
CONFIG_SECURITYFS=y

# Sometimes not enabled with defconfig
CONFIG_VIRTIO_NET=y
CONFIG_E1000=y
```

Syzkaller setup #3

```
# manual image creation
mkdir image
cd image
wget https://raw.githubusercontent.com/google/syzkaller/master/tools/create-image.sh -O create-image.sh
chmod +x create-image.sh
sudo apt install debootstrap
./create-image.sh
```

```
export KERNEL=./kernel/linux-6.10/
export IMAGE=./image/

qemu-system-x86_64 \
        -m 2G \
        -smp 2 \
        -kernel $KERNEL/arch/x86/boot/bzImage \
        -append "console=ttyS0 root=/dev/sda earlyprintk=serial net.ifnames=0" \
        -drive file=$IMAGE/bullseye.img,format=raw \
        -net user,host=10.0.2.10,hostfwd=tcp:127.0.0.1:10021-:22 \
        -net nic,model=e1000 \
        -enable-kvm \
        -snapshot \
        -nographic \
        -pidfile vm.pid \
        2>&1 | tee vm.log
```

```
ssh -p 10021 root@127.0.0.1 -i ./image/bullseye.id_rsa
```

Syzkaller setup #3

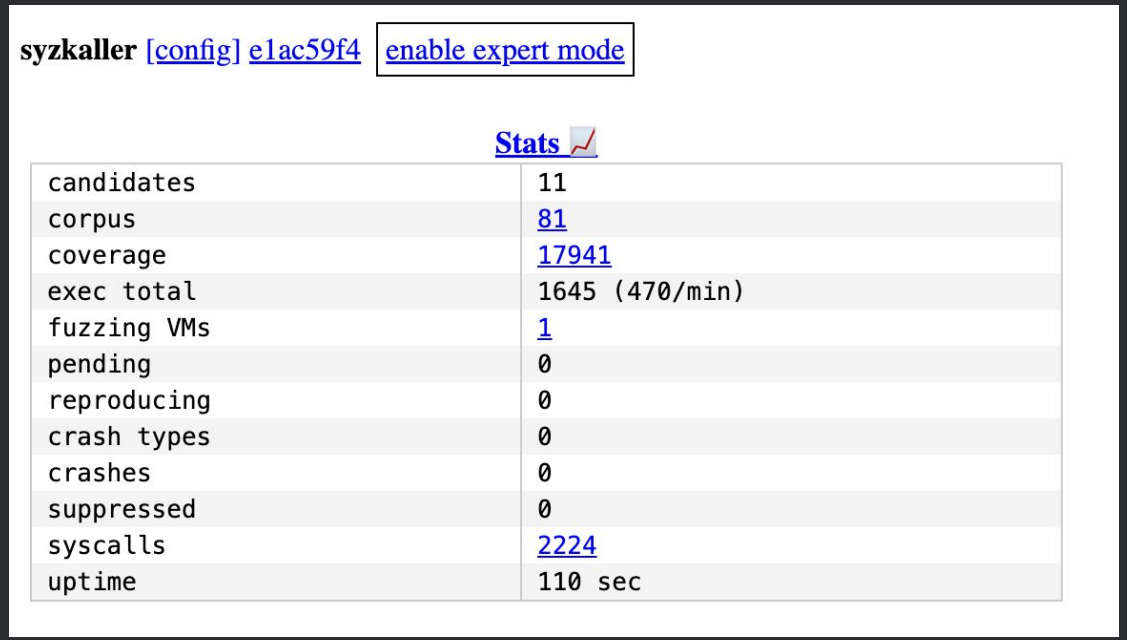
```
mkdir workdir
# config.json content
{
        "target": "linux/amd64",
        "http": "0.0.0.0:56741",
        "syzkaller": "./",
        "workdir": "./workdir",
        "kernel_obj": "./linux-6.10.10",
        "image": "./image/bullseye.img",
        "sshkey": "./bullseye.id_rsa",
        "procs": 4,
        "type": "qemu",
        "vm": {
                "qemu_args": "-enable-kvm",
                "count": 2,
                "cpu": 2,
                "cmdline": "net.ifnames=0",
                "mem": 2048,
                "kernel": "./linux-6.10.10/arch/x86/boot/bzImage"
        }
}
# execute
./bin/syz-manager -config=config.json # --debug
# https://github.com/google/syzkaller/blob/master/pkg/mgrconfig/config.go
```

Syzkaller setup #3

```
mkdir workdir
# config.json content
{
        "target": "linux/amd64",
        "http": "0.0.0.0:56741",
        "syzkaller": "./",
        "workdir": "./workdir",
        "kernel_obj": "./linux-6.10.10",
        "image": "./image/bullseye.img",
        "sshkey": "./bullseye.id_rsa",
        "procs": 4,
        "type": "qemu",
        "vm": {
                "qemu_args": "-enable-kvm",
                "count": 2,
                "cpu": 2,
                "cmdline": "net.ifnames=0",
                "mem": 2048,
                "kernel": "./linux-6.10.10/arch/x86/boot/bzImage"
        }
}
# execute
./bin/syz-manager -config=config.json # --debug
# https://github.com/google/syzkaller/blob/master/pkg/mgrconfig/config.go
```

Syzkaller setup

- Reach the dashboard at http://0.0.0.0:56741
  - If you see some coverage and corpus, you're good to go ☺️
- Some quick troubleshooting
  - -debug & patient (*qemu can be slow*)
  - ps –ef | grep qemu □ SSH (ps)
  - KCOV enabled
  - If your system crashes, balance given resources



syzkaller [config] e1ac59f4 enable expert mode

Stats

candidates	11
corpus	81
coverage	17941
exec total	1645 (470/min)
fuzzing VMs	1
pending	0
reproducing	0
crash types	0
crashes	0
suppressed	0
syscalls	2224
uptime	110 sec

Extend Syzkaller descriptions with Syzlang

 # Syzlang grammar basics

 # How to describe syscall interfaces

 # Implement a syzlang description from scratch

hacktive security

Syzlang Introduction

- *«syzkaller uses declarative description of syscall interfaces to manipulate programs (sequences of syscalls)»*
  - *«The translated descriptions are then used to generate, mutate, execute, minimize, serialize and deserialize programs.»*
- Syzkaller ultimately generates an executable file
  - descriptions => syz programs => C programs => binary

```
syscallname "(" [arg ["," arg]*] ")" [type] ["(" attribute* ")"]
arg = argname type
argname = identifier
type = typename [ "[" type-options "]" ]
typename = "const" | "intN" | "intptr" | "flags" | "array" | "ptr" |
                "string" | "filename" | "glob" | "len" |
                "bytesize" | "bytesizeN" | "bitsize" | "vma" | "proc" |
                "compressed_image"
type-options = [type-opt ["," type-opt]]
```

Syzlang basics

syscallname(argname typename[type_options], ..)

- argname is an identifier
  - it can be referenced elsewhere
- typename describes parameter type
  - integer, array, ptr, flags, struct, ..
- type_options further describes parameter type
  - array[int64], ptr[in, int64], ..

C

```c
char buf[256];
int fd = open("./file", O_RDWR | O_CREAT, S_IRWXU);
read(fd, buffer, 10);
close(fd)
```

C

```c
char buf[256];
int fd = open("./file", O_RDWR | O_CREAT, S_IRWXU);
read(fd, buffer, 10);
close(fd)
```

```c
int open(const char *pathname, int flags, ...
                /* mode_t mode */ );
```

```c
ssize_t read(int fd, void buf[.count], size_t count);
```

```c
int close(int fd);
```

Syzlang basics

syscallname(argname typename[type_options], ..)

```
open(file filename, flags flags[open_flags], mode flags[open_mode]) fd
read(fd fd, buf buffer[out], count len[buf])
open_flags = O_RDWR, O_CREAT, ..
```

- argname is an identifier
  - it can be referenced elsewhere
- typename describes parameter type
  - integer, array, ptr, flags, struct, ..
- type_options further describes parameter type
  - array[int64], ptr[in, int64], ..

C

```
char buf[256];
int fd = open("./file", O_RDWR | O_CREAT, S_IRWXU);
read(fd, buffer, 10);
close(fd)
```

SyzLang

```
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_RDWR, O_CREAT, ..
open_mode = S_IRWXU, S_IWUSR, ..
```

filename generates file names
flags is a set of values
buffer is equivalent of char*
len is the size of buf

Syzlang typenames & typeoptions

Arrays

- array[typename, size]
- array[int8], array[int8, 10]
- ..(.., list array[int16], sz len[list])

Integers

- intN
- int8, int16, int32, int64, intptr
- int32[0:100] – range of values
- int32be – big endian

Constants

- const[value, typename]
- const[1337, int16]

Strings

- string["strings", size]
- string["4x0r"]
- string["4x0r", 256] – 0 padding
- stringnoz["4x0r"]

Pointers – ptr/ptr64

- ptr[direction, typename]
- ptr[in, array[int8, 10]]

flags, filename, len, buffer, ..

- complete list

SyzLang

```
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_RDWR, O_CREAT, ..
open_mode = S_IRWXU, S_IWUSR, ..
```

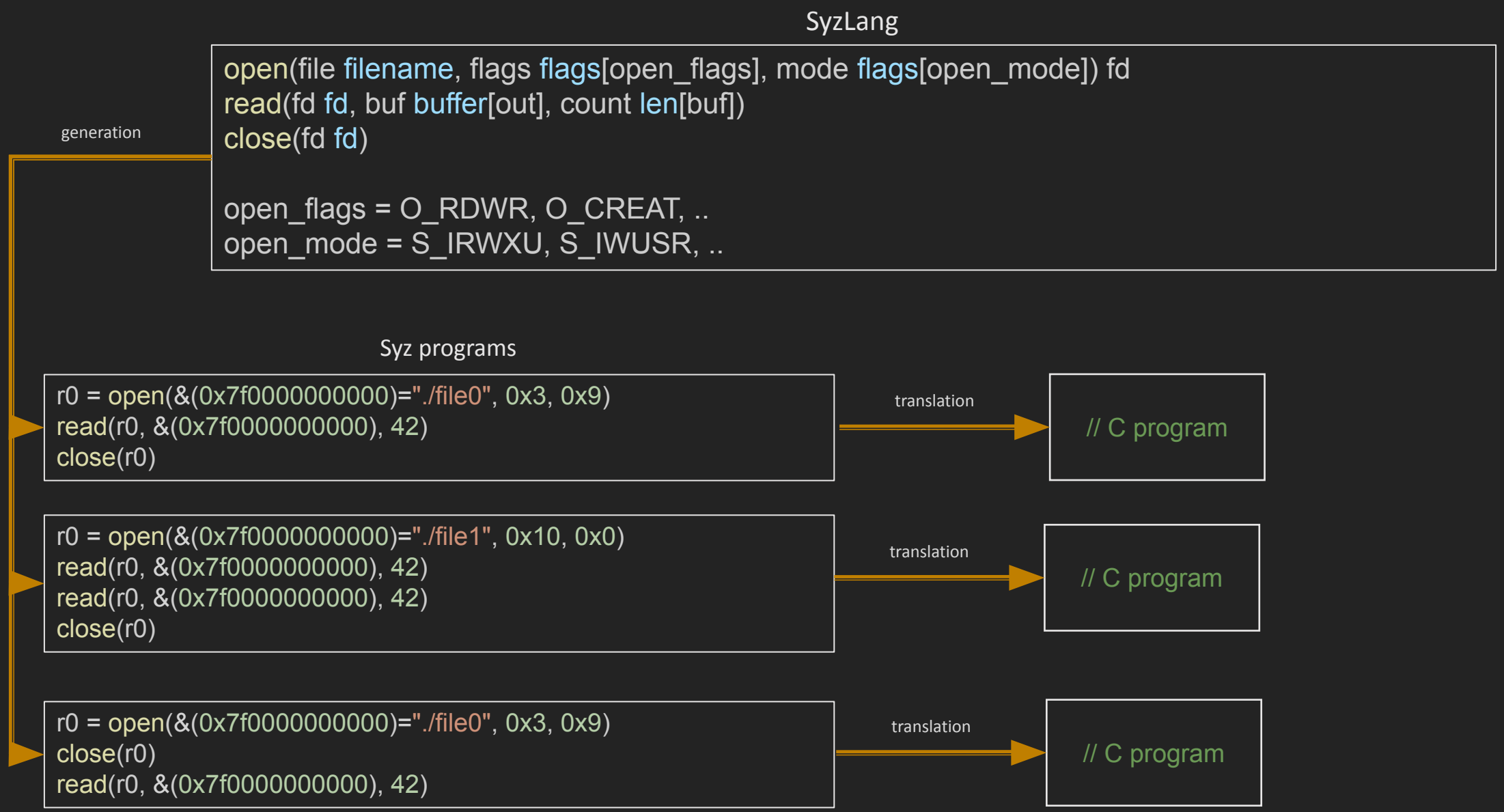
generation

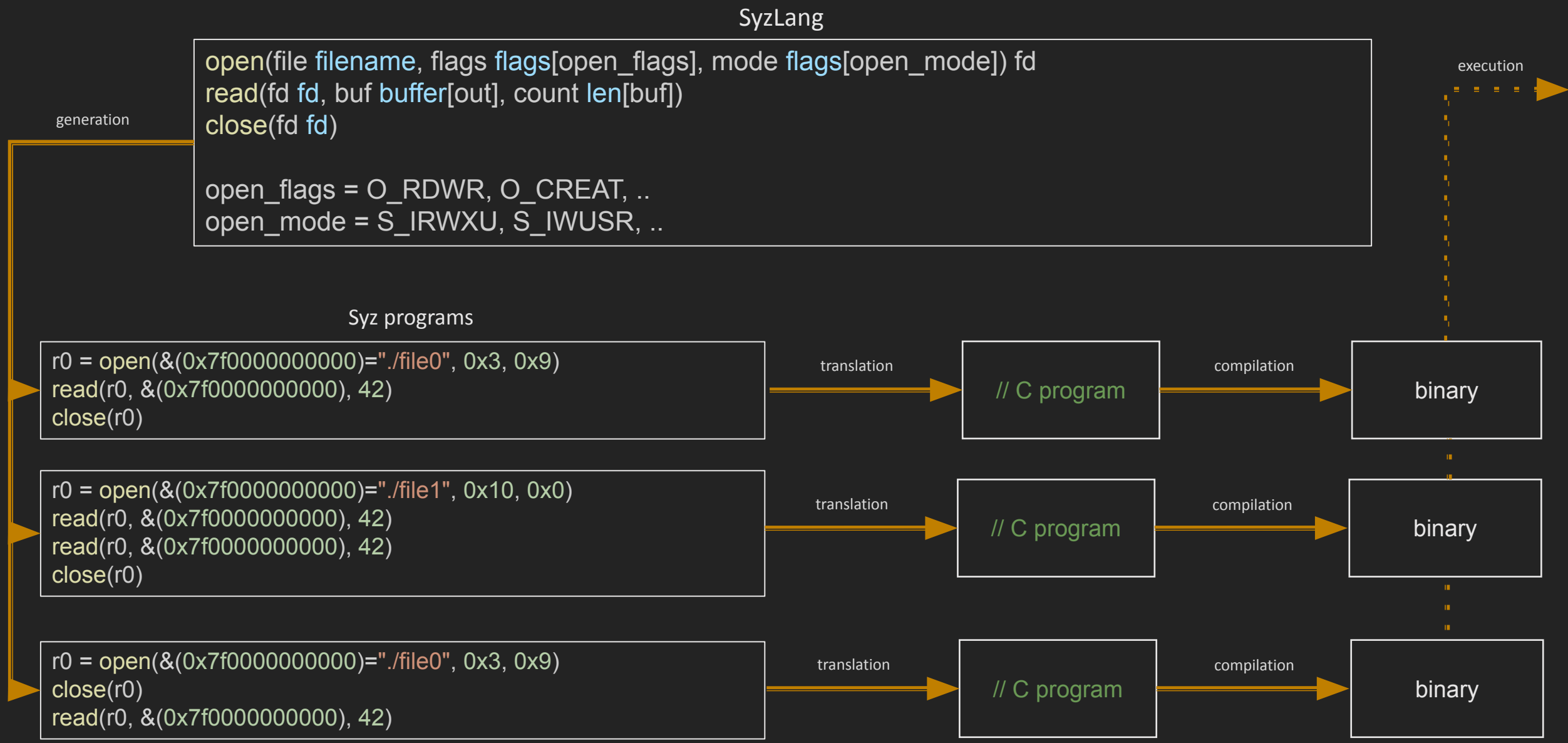
Syz programs

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

```
r0 = open(&(0x7f0000000000)="./file1", 0x10, 0x0)
read(r0, &(0x7f0000000000), 42)
read(r0, &(0x7f0000000000), 42)
close(r0)
```

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


SyzLang
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_RDWR, O_CREAT, ..
open_mode = S_IRWXU, S_IWUSR, ..
generation
Syz programs
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
read(r0, &(0x7f0000000000), 42)
close(r0)
translation
// C program
r0 = open(&(0x7f0000000000)="./file1", 0x10, 0x0)
read(r0, &(0x7f0000000000), 42)
read(r0, &(0x7f0000000000), 42)
close(r0)
translation
// C program
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
close(r0)
read(r0, &(0x7f0000000000), 42)
translation
// C program


SyzLang
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_RDWR, O_CREAT, ..
open_mode = S_IRWXU, S_IWUSR, ..
generation
execution
Syz programs
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
read(r0, &(0x7f0000000000), 42)
close(r0)
r0 = open(&(0x7f0000000000)="./file1", 0x10, 0x0)
read(r0, &(0x7f0000000000), 42)
read(r0, &(0x7f0000000000), 42)
close(r0)
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
close(r0)
read(r0, &(0x7f0000000000), 42)
translation
translation
translation
// C program
// C program
// C program
compilation
compilation
compilation
binary
binary
binary

Syzlang structs

```
ssize_t sendmsg(int socket, const struct msghdr *message, int flags);
```

Syzlang structs

```
ssize_t sendmsg(int socket, const struct msghdr *message, int flags);
```

C language

```
struct msghdr {
        void*           msg_name;
        int             msg_namelen;
        struct iovec*   msg_iov;
        __kernel_size_t msg_iovlen;
        void*           msg_control;
        __kernel_size_t msg_controllen;
        unsigned int    msg_flags;
};
```

Syzlang structs

```
ssize_t sendmsg(int socket, const struct msghdr *message, int flags);
```

C language

```
struct msghdr {
        void*           msg_name;
        int                     msg_namelen;
        struct iovec*           msg_iov;
        __kernel_size_t msg_iovlen;
        void*           msg_control;
        __kernel_size_t msg_controllen;
        unsigned int            msg_flags;
};
```

Syzlang

```
send_msghdr {
        msg_name            ptr[in, sockaddr_storage, opt]
        msg_namelen         len[msg_name, int32]
        msg_iov     ptr[in, array[iovec_in]]
        msg_iovlen          len[msg_iov, intptr]
        msg_control         ptr[in, array[cmsghdr], opt]
        msg_controllen bytesize[msg_control, intptr]
        msg_flags  const[0, int32]
}
```

Syzlang structs

```
structname {
        argname typename[type_options] (direction)
        argname typename[type_options] (direction)
        ...
}
```

- structname is the structure identifier
- argname is the member identifier
- typename describes parameter type
  - integer, filename, ptr, flags, struct, string, ..
- typeoptions further describes parameter type
  - array[int64], ptr[in, int64], ..
- direction can be in, out or inout

C language

```
struct struct_1{
        int m1;
        int m2;
}
```

```
struct struct_2{
        struct struct_1* s1;
        int m2;
}
```

```
struct struct_3{
        int flags;
        char[256] name;
        int name_len;
}
```

C language

```
struct struct_1{
        int m1;
        int m2;
}
```

Syzlang

```
struct_1{
        m1 int32
        m2 int32
}
```

```
struct struct_2{
        struct struct_1* s1;
        int m2;
}
```

```
struct struct_3{
        int flags;
        char[256] name;
        int name_len;
}
```

C language

```
struct struct_1{
        int m1;
        int m2;
}
```

Syzlang

```
struct_1{
        m1 int32
        m2 int32
}
```

```
struct struct_2{
        struct struct_1* s1;
        int m2;
}
```

```
struct_2{
        s1 ptr[in, struct_1]
        m2 int32
}
```

```
struct struct_3{
        int flags;
        char[256] name;
        int name_len;
}
```

C language

```
struct struct_1{
        int m1;
        int m2;
}
```

Syzlang

```
struct_1{
        m1 int32
        m2 int32
}
```

```
struct struct_2{
        struct struct_1* s1;
        int m2;
}
```

```
struct_2{
        s1 ptr[in, struct_1]
        m2 int32
}
```

```
struct struct_3{
        int flags;
        char[256] name;
        int name_len;
}
```

```
struct_3{
        flags flags[struct_3_flags, int32]
        name array[int8, 256]
        name_len len[name, int32]
}
struct_3_flags = ONE, TWO, ..
```

Syzlang name variants

- Problem
  - a single syscall can be used for totally different things
  - e.g. open can be used to open a standard file or a character driver
    - fd = open("./file", ..) => read(fd, ..)
    - fd = open("/dev/audio", ..) => ioctl(fd, AUDIO_CMD, ..)
    - fd = open("/dev/tty", ..) => ioctl(fd, TTY_CMD, ..)
  - a generic description is not enough
    - open(file filename, flags flags[open_flags], mode flags[open_mode]) fd

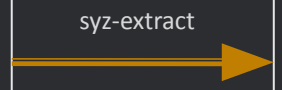
Syzlang name variants

- Problem
  - a single syscall can be used for totally different things
  - e.g. open can be used to open a standard file or a character driver
    - fd = open("./file", ..) => read(fd, ..)
    - fd = open("/dev/audio", ..) => ioctl(fd, AUDIO_CMD, ..)
    - fd = open("/dev/tty", ..) => ioctl(fd, TTY_CMD, ..)
  - a generic description is not enough
    - open(file filename, flags flags[open_flags], mode flags[open_mode]) fd
- Solution: variant names
  - describes each scenario individually
    - open$file(..) fd_file
    - open$audio_driver(..) fd_audio
    - open$tty(..) fd_tty

Syzlang ./sys/OS/

- Descriptions defined in: ./sys/OS/subsystem.txt
  - ./sys/linux/socket.txt
  - ./sys/linux/socket_inet6.txt
  - ./sys/linux/io_uring.txt
  - ./sys/linux/sys.txt
- Constants at ./sys/OS/subsystem.txt.const
  - Retrieved from syz-extract with the kernel source code (./include/)
    - Default .const files are sporadically updated with the mainline kernel

```
// linux-src/include/uapi/asm-generic/fcntl.h
#define O_CREAT 00000100
#define O_EXCL 00000200
#define O_DIRECT 00040000
#define O_DIRECTORY 00200000
```




```
// syzkaller-src/sys/linux/sys.txt.const
O_CREAT = 64, mips64le:256
O_EXCL = 128, mips64le:1024
O_DIRECT = 16384, arm:arm64:65536, mips64le:32768, ppc64le:131072
O_DIRECTORY = 65536, arm:arm64:ppc64le:16384
```

Syzlang compile descriptions

- Recompile all descriptions
  - make extract TARGETOS=linux SOURCEDIR=$KSRC
    - update sys/OS/*.const files (*syz-extract*) – requires kernel recompilation
  - make generate
    - translate descriptions in Go (*syz-sysgen*) – sys/OS/gen/
  - make
    - rebuild binaries
- Compile specific descriptions
  - tools/syz-env make bin/syz-extract
  - bin/syz-extract -os linux -arch $ARCH -buildir $BSRC -sourcedir $KSRC <new>.txt
  - tools/syz-env make generate
  - tools/syz-env make

Syzlang demo

- Live demo
  - Write description for syzdemo driver
  - Trigger BUG() and KASAN
- On your own
  - Write description for syzdiy driver
  - Trigger BUG() and KASAN

Coverage report

- Different color represent different statuses (coverage.md)
  - black
    - All PCs values associated with the line are covered
  - orange
    - Multiple PCs values associated with the line and not all of them are executed
  - crimson red
    - The function this line is in is not executed at all
  - red
    - Uncovered line but function executed.
  - grey
    - Line not instrumented

Conclusion

Syzlang ./sys/OS/

- Sometimes the documentation is not 100% complete
  - grep 'typename' sys/linux/*.txt (e.g. grep 'buffer\[in' sys/linux/*.txt)
- SyzLang sytnax highlighting with a VSCode plugin

- Time is limited (90m), you should have the basics to go on your own:
  - pseudo syscalls
  - type aliases
  - type templates
  - conditional fields
  - real-life descriptions (sys/OS/*.txt)
  - experiment!

exit(exit_flags)

- if(need_support || question || just_chat || flex)
  - if(!at(around_conference) || !at(hacktive_security_stand) || too_late)
    - linkedin = Alessandro Groppo;
    - twitter = @kiks7_7;
    - email = alessandro@hacktivesecurity.com; BUG();
- else if(job_opportunity)
  - email = jobs@hacktivesecurity.com;
- else
  - questions? || exit(0);

hacktive security

Q&A

Thanks for the time

alessandro [at] hacktivesecurity [dot] com

info [at] hacktivesecurity [dot] com