ubuCon ASIA 2021

Who ‘kill’ed my processes?

Trace Signals through Linux Kernel hacking

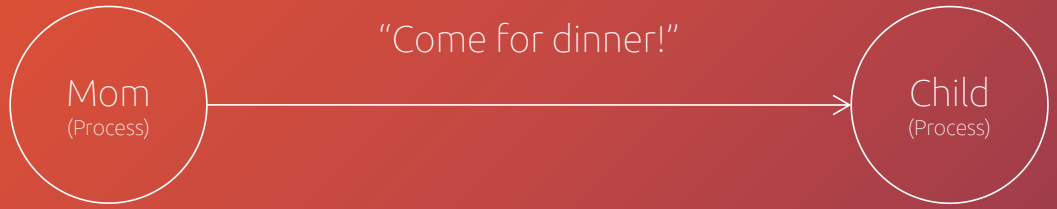
Who am I?

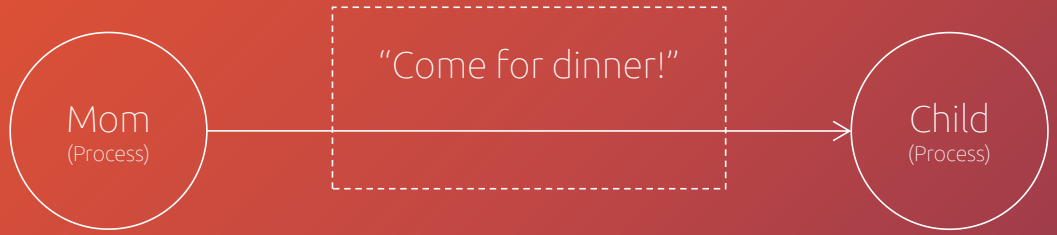
- Leesoo Ahn (lsahn _*at*_ ooseel _*dot*_ net)
- Software Engineer at Wireless AP company
- Contributor at GNOME Desktop (2018 - 2020)
- ARM64 linux kernel core and virtualization

Signal

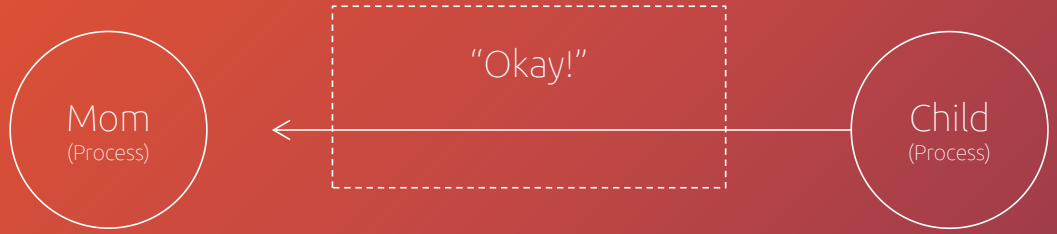

Mom
(Process)


Child
(Process)


Mom
(Process)
"Come for dinner!"
Child
(Process)



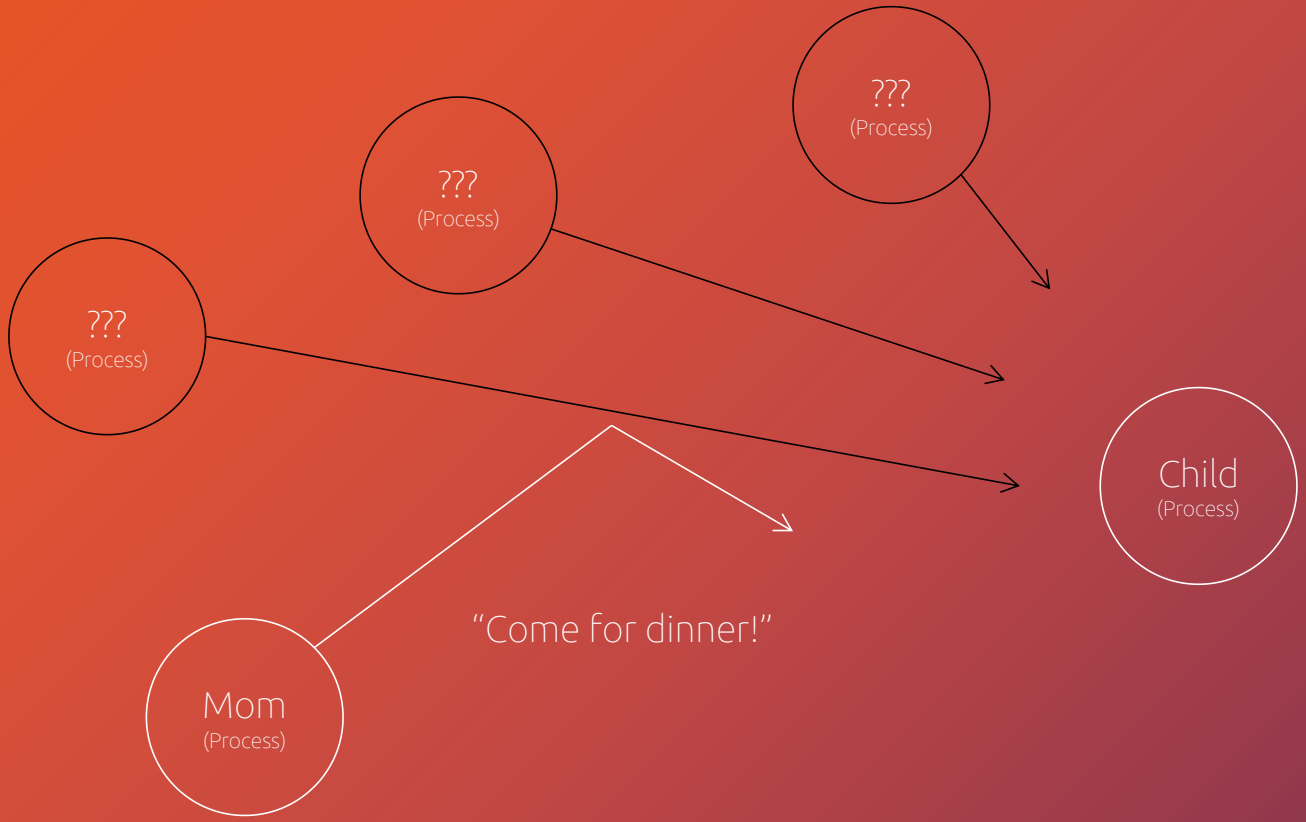
A signal to Child



Response* to Mom

“looks good!”

Real world AIN'T "looks good!"



How real world looks like

We have to survive

We have to survive
but how?

Some helpers

The helpers are

- GDB (line by line debugger)

The helpers are

- GDB (line by line debugger)
- Strace (syscall & signal tracer)

The helpers are

- GDB (line by line debugger)
- Strace (syscall & signal tracer)
- Ftrace (widely kernel tracer)

The helpers are

- GDB (line by line debugger)
- Strace (syscall & signal tracer)
- Ftrace (widely kernel tracer)
- et cetera (something I don’t know)

What if

What if this


???
(Process)
???
(Process)
???
(Process)
Child
(Process)
Mom
(Process)
"Come for dinner!"

What if this happens at booting?

Oops!

- Mount-ing a FS which has helpers ATM!

Oops!

- Mount-ing a FS which has helpers ATM!
- Deal with order of ‘init.d’

Oops!

- Mount-ing a FS which has helpers ATM!
- Deal with order of ‘init.d’
- Shell ain’t allowed

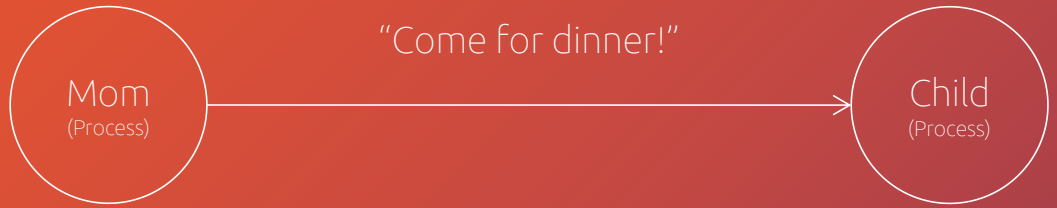
Oops!

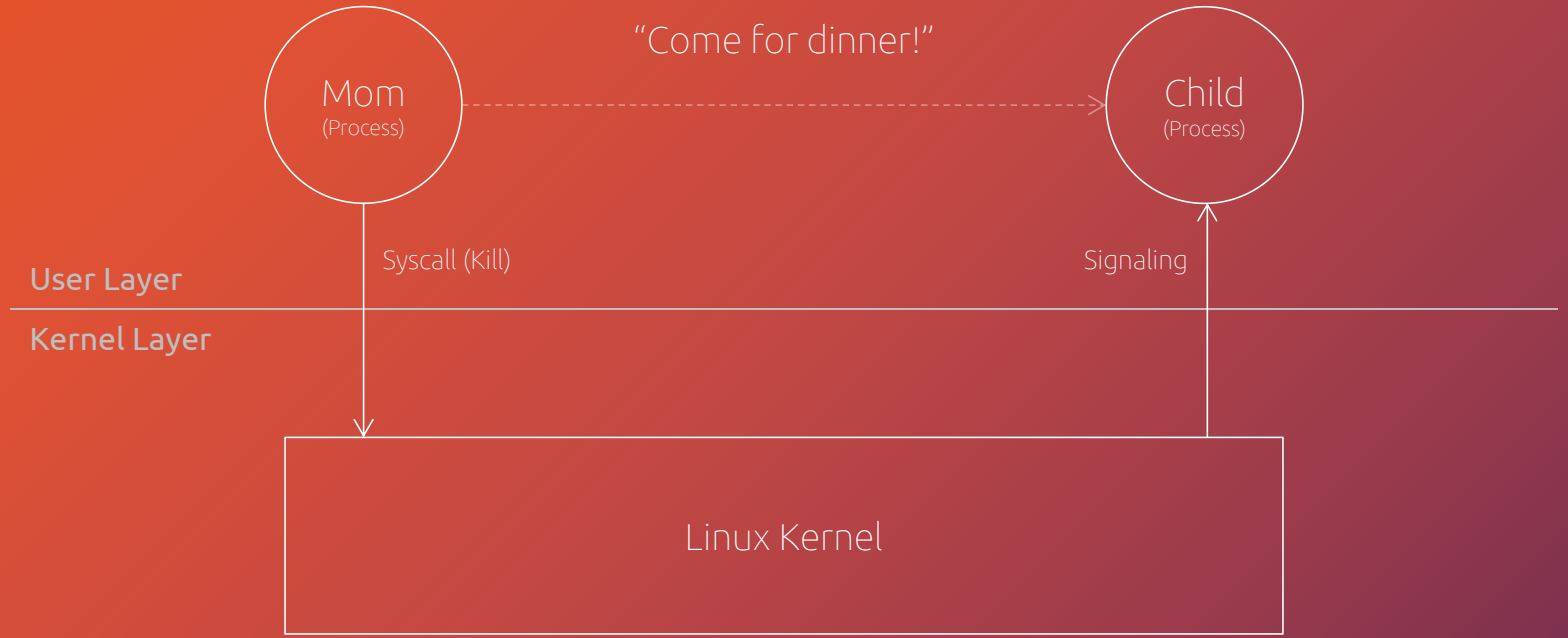
- Mount-ing a FS which has helpers ATM!
- Deal with order of ‘init.d’
- Shell ain’t allowed
- et cetera (yet something else)

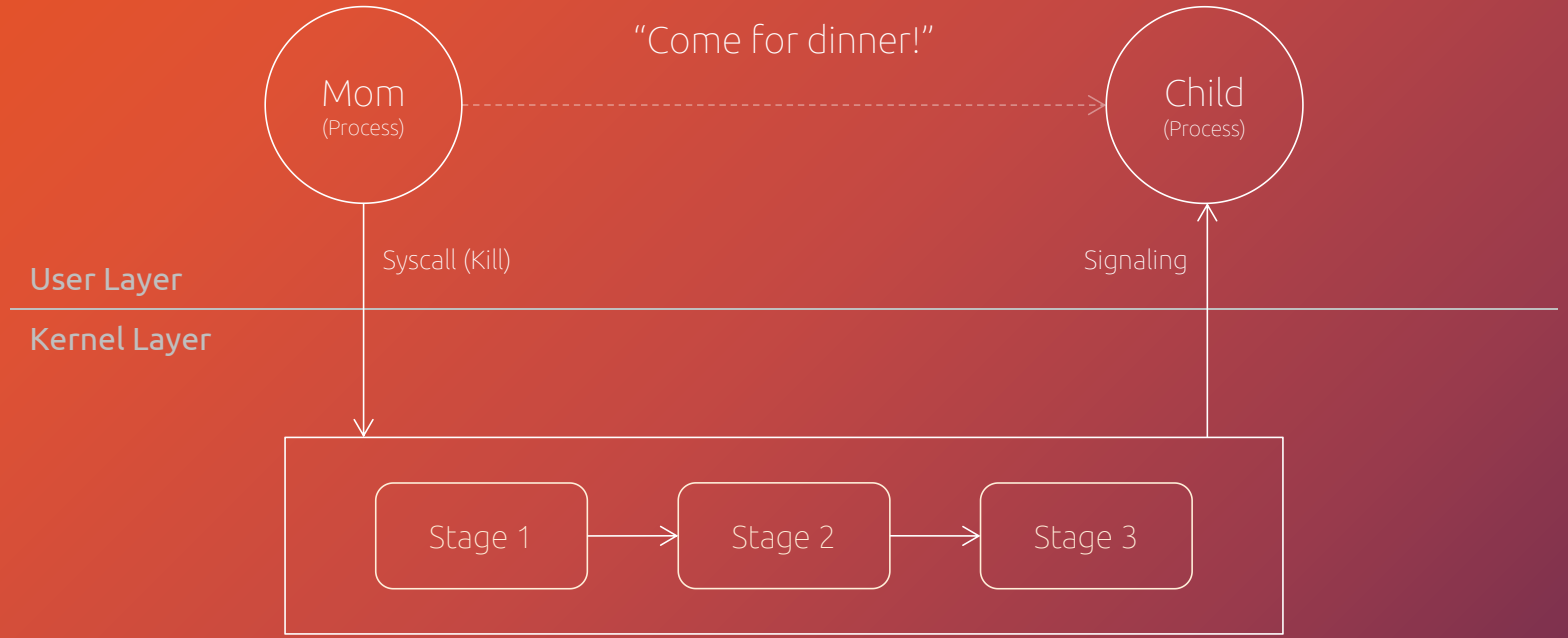
Such a headache!

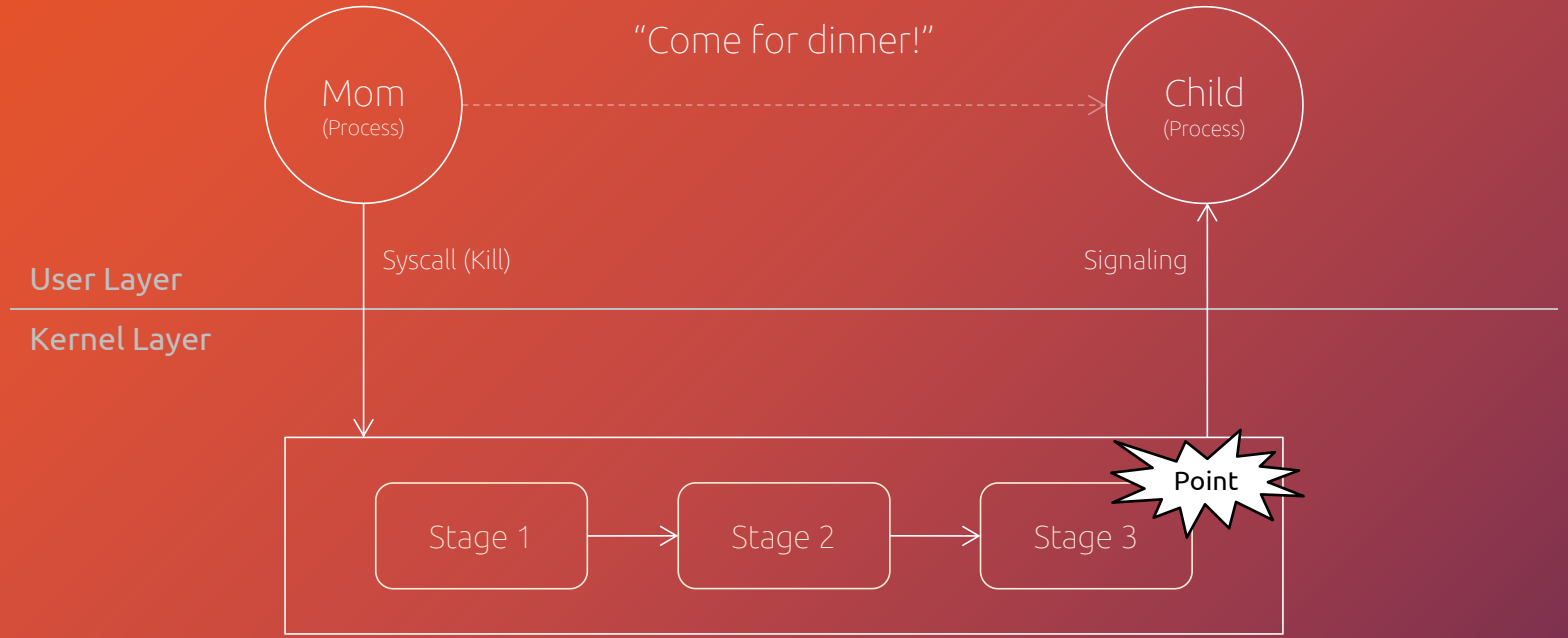
Focus on REAL problem!

What about at ‘Kernel’?


Mom
(Process)
"Come for dinner!"
Child
(Process)


Mom
(Process)
"Come for dinner!"
Child
(Process)
Syscall (Kill)
Signaling
User Layer
Kernel Layer
Linux Kernel


Mom
(Process)
"Come for dinner!"
Child
(Process)
Syscall (Kill)
Signaling
User Layer
Kernel Layer
Stage 1
Stage 2
Stage 3


Mom
(Process)
"Come for dinner!"
Child
(Process)
Syscall (Kill)
Signaling
User Layer
Kernel Layer
Stage 1
Stage 2
Stage 3
Point

Stage 3, Why?

- Signal Check

Stage 3, Why?

- Signal Check
- Process Check

Stage 3, Why?

- Signal Check
- Process Check
- NULL Check

Stage 3, Why?

- Signal Check
- Process Check
- NULL Check
- and a bunch of Checks!

Stage 3, How?

- Filter Signals

Stage 3, How?

- Filter Signals
- Use ‘printk’

Stage 3, How?

- Filter Signals
- Use ‘printk’
- Sender, Receiver and Signal number

Simple Patch

```
    complete_signal(sig, t, type);

    /* code here */
    if (sig != 17 /* SIGCHLD */ &&
        sig != 14 /* SIGALRM */ &&
        info && t)
    {
        int srcpid, dstpid;
        char src[TASK_COMM_LEN] = { 0, };
        char dst[TASK_COMM_LEN] = { 0, };
        struct task_struct *cur_task = NULL;

        if (!force) {
            srcpid = info->si_pid;
            cur_task = find_task_by_vpid(srcpid);
        } else
            srcpid = 0;

        dstpid = t->pid;
        memcpy(src, cur_task ? cur_task->comm : "unknown", TASK_COMM_LEN-1);
        memcpy(dst, t->comm, TASK_COMM_LEN-1);

        printk(KERN_INFO "Signal :: (%s %d) --[%d]--> (%s %d)\n",
            src, srcpid, sig, dst, dstpid);
    }

ret:
    trace_signal_generate(sig, info, t, type != PIDTYPE_PID, result);
    return ret;
}
```

- Filter SIGCHLD & SIGALRM
- Sender, Signal and Receiver

Syscall ’kill’ (Kernel 5.4.x)

- SYSCALL_DEFINE2(kill, ...)
  → kill_something_info(...)
    → kill_pid_info(...)
      → group_send_sig_info(...)
        → do_send_sig_info(...)
          → send_signal(...)
            → __send_signal(...) {
                complete_signal(...)
                /* code here */
              }

Demo

Thank you!

Q&A