Seminar

Haonan Lu

COMPASS

Feb 25, 2021

- Capture syscalls
  - *Deploy sysdig*
  - Write a kernel module manually to replace sysdig
- Make more experiments
  - Figure out the overhead
  - Judge whether the information captured is enough
  - Make a demo to finish original schedule

- Capturing syscall: Github
  - Finish a basic version (*print log*, *condition filter*)
  - Investigate different types (i.e., passing different parameters) of syscalls
  - Set this project as my *graduation project*

For example, we have a syscall `getrandom`:

```
ssize_t getrandom (void *__buffer, size_t __length, unsigned int __flags)
```

And there is the schema of syscall in Arm64:

Table 1: Syscall Calling Convention for Arm64

arch	syscall NR	**return**	**arg0**	arg1	arg2	arg3	arg4	arg5
arm64	x8	x0	x0	x1	x2	x3	x4	x5

Then we call it in our code:

```
int main(){
    int a;
    getrandom(&a,sizeof a,GRND_RANDOM);
}
```

The capturing snippet show as follow: *but `regs[0]` was replaced by `ret`*

```
[my_sysdig:] syscall 0x116, with pid=0x178e, name=a.out
[my_sysdig:] exit syscall, regs[0]=0x4, with pid=0x178e, ret=0x4
```

There are 26 syscalls (arm64) similar to `getrandom`, and we need to save their `regs[0]` when entering the syscall.

- Set a global variable at `entering` and get its value at `exit`?
  - No, multiple system calls can exist *at the same time*
  - Considering multithread: When the program is waiting for a syscall (such as `read`), other threads can still run normally or call syscalls.
- Maintain a data struct to save `regs[0]`
  - Syscall will block the thread, so we could save a value for each thread number.