# Questions

by Tong Wu, Lu Yu

**1.Which capabilities API (seccomp-bpf, AppArmor, or SELinux) did you choose? Why did you make that choice?**

We choose to use seccomp-bpf as our capabilities API.

Seccomp is a Linux feature that allows a userspace program to set up syscall filters. The BPF program loaded into the kernel starts with a system call + arguments and results in a filtering decision. Based on the results of the filter, the system call can be allowed, blocked, or the process can be killed. It can be easily implemented with the <seccomp-bpf.h> header file included in the program code.

AppArmor, or SELinux is MAC (Mandatory Access Control) systems/framework in which you add policies to sandboxing. It is used to whitelist or blacklist a subject's (program's) access to an object (file, path, etc.). The policies can also be used to restrict capabilities, or even limit network access. These strive for system-wide enforcement of policies that control the actions and resources that each program on a system can perform. Comparing to AppArmor, SELinux can be more complex to configure.

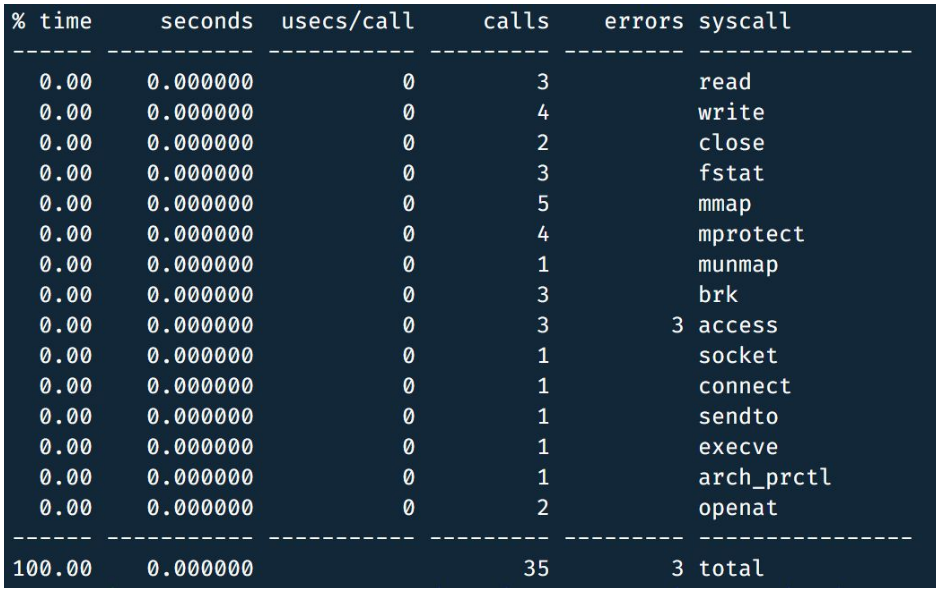
In this assignment, our goal is to restrict system calls in the program. There is no need to add a systemwide security policy. With seccomp-bpf, the requirements can be achieved by just modifying the client and server code. However, if using AppArmor or SELinux, a separate configuration file needs to be added, and a sandbox or VM is required. Otherwise, the policy will be implemented in the entire system. This costs much more effort than seccomp-bpf.

**2.What was the process you used to ascertain the list of system calls required by each program?**

a. use `strace -xfc ./a.out` to get the required system calls for each program
b. `#include <seccomp.h>`
c. Initiate the filter with `scmp_filter_ctx ctx` and `ctx = seccomp_init(SCMP_ACT_TRAP)`
d. Add the list of allowed system calls to the filter with `seccomp_rule_add()`
e. Load the filter with `seccomp_load(ctx)`

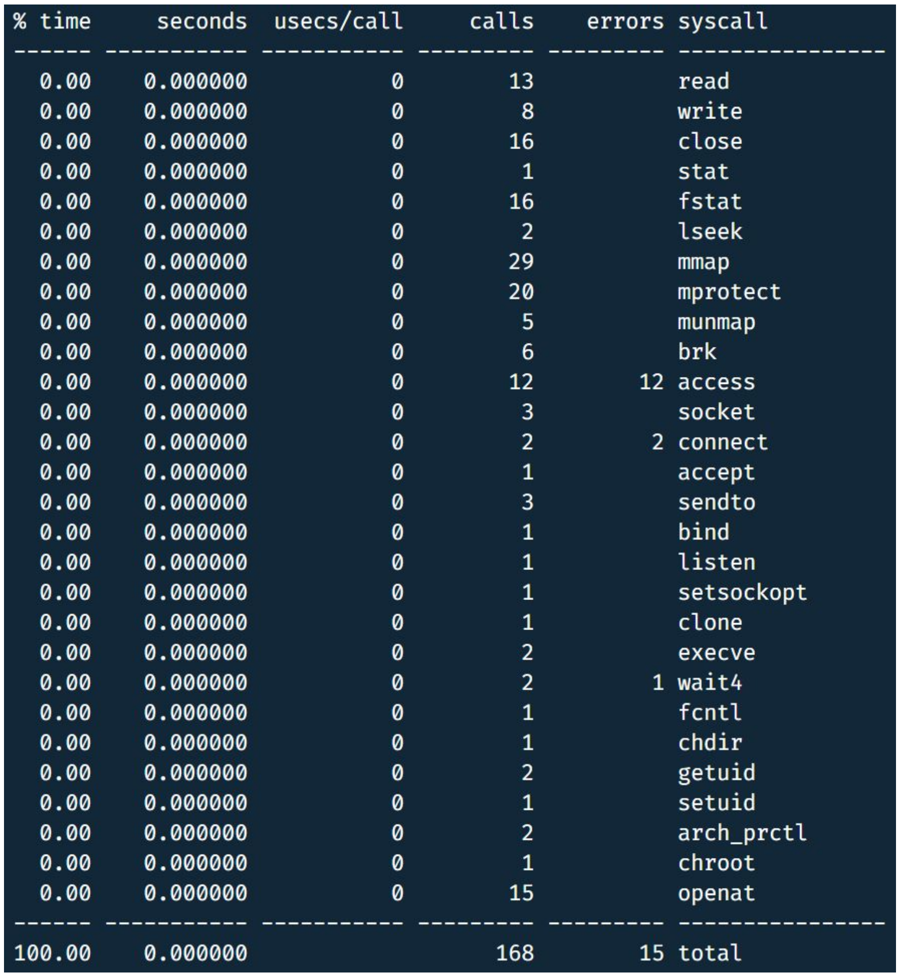
**3.What system calls are needed by each?**

client:



```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         3           read
  0.00    0.000000           0         4           write
  0.00    0.000000           0         2           close
  0.00    0.000000           0         3           fstat
  0.00    0.000000           0         5           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         3         3 access
  0.00    0.000000           0         1           socket
  0.00    0.000000           0         1           connect
  0.00    0.000000           0         1           sendto
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           arch_prctl
  0.00    0.000000           0         2           openat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                    35         3 total
```

server:



```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0        13           read
  0.00    0.000000           0         8           write
  0.00    0.000000           0        16           close
  0.00    0.000000           0         1           stat
  0.00    0.000000           0        16           fstat
  0.00    0.000000           0         2           lseek
  0.00    0.000000           0        29           mmap
  0.00    0.000000           0        20           mprotect
  0.00    0.000000           0         5           munmap
  0.00    0.000000           0         6           brk
  0.00    0.000000           0        12        12 access
  0.00    0.000000           0         3           socket
  0.00    0.000000           0         2         2 connect
  0.00    0.000000           0         1           accept
  0.00    0.000000           0         3           sendto
  0.00    0.000000           0         1           bind
  0.00    0.000000           0         1           listen
  0.00    0.000000           0         1           setsockopt
  0.00    0.000000           0         1           clone
  0.00    0.000000           0         2           execve
  0.00    0.000000           0         2         1 wait4
  0.00    0.000000           0         1           fcntl
  0.00    0.000000           0         1           chdir
  0.00    0.000000           0         2           getuid
  0.00    0.000000           0         1           setuid
  0.00    0.000000           0         2           arch_prctl
  0.00    0.000000           0         1           chroot
  0.00    0.000000           0        15           openat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                   168        15 total
```

**4.What happens when your application calls the prohibited system call? What is the application behaviour that results from the call?**

When calling the prohibited system calls after loading the seccomp rules, the process will be killed with a notification saying “Bad system call (core dumped)”. Besides, an err value was also provided as exit code which could be accessed using “echo $?” command.