CMPE 279
Assignment 4
Ke He 012561380
Qiyue Zhang 012550096

Assignment:
Using seccomp-bpf, AppArmor, or SELinux, configure a security policy that restricts the system calls
allowed by the client and server program to that minimum set required. You may need to use the 'strace'
tool as we did in class to determine what that minimum set is required for each program. Finally, ensure
that your enforcement technique is working by intentionally calling a prohibited system call in one of the
programs and note the resulting behaviour (don't turn in that code though).

Questions:

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

Seccomp filtering provides a means for a process to specify a filter for incoming system calls. The filter is expressed as a Berkeley Packet Filter (BPF) program, as with socket filters, except that the data operated on is related to the system call being made: system call number and the system call arguments. This allows for expressive filtering of system calls using a filter program language with a long history of being exposed to userland and a straightforward data set.

We use seccomp-bpf, it is easier to use, we can simply whitelist and list out all the system call we need by our program. AppArmor and SELinux are more complex.

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

Firstly, we tried "*strace"* to list out all the system calls by client and server, but we found it isn't the complete system call list by strace, then we realized we need to use *"strace -f"* to list out both parent and child process made system calls. After *"strace -f"* we are be able to find all the system calls that made by our client and server.

3. What system calls are needed by each?

System calls required by **server:**

```
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(execve), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(brk), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(access), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(openat), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(fstat), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(mmap), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(close), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(arch_prctl), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(mprotect), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(munmap), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(socket), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(setsockopt), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(bind), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(listen), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(accept), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(clone), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(wait4), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit_group), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(seccomp), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(prctl), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(getcwd), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(chdir), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(chroot), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(setuid), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(sendto), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(stat), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(dup), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(fcntl), 0);
```

System calls needed by **client:**

```
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(execve), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(brk), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(access), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(openat), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(fstat), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(mmap), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(close), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(arch_prctl), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(mprotect), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(munmap), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(socket), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit_group), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(prctl), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(seccomp), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(connect), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(sendto), 0);
```

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

Seccomp will filter out those calls that are not on the list, when prohibited system call will not proceed if it is not on the list.

When application called prohibited system call, the process will run before the system calls before that prohibited system call, once whe the prohibited system call made, it will stop the program right there and return the message “bad system call”.