Andrew Ingson

11/15/19

CMSC 421

Project 2 Initial Design

Kernel Code (line numbers presume an unmodified copy of kernel 5.2.11)

- `Makefile` -
  - Modify line 5 to have the project version string "-cmsc421project2-aings1"
  - Modify line 996 to include the project 2 directory
- `arch/x86/entry/syscalls/syscall_64.tbl` - At line 358, add the table entries of the three syscalls, sbx421_block, sbx421_unblock, sbx421_count. All calls should be common
- `include/linux/syscalls.h` - At line 1400, add the three asmlinkages for the previous three syscalls. All three syscalls should return *long*. All three should pass the parameters *pid_t proc* and *unsigned long nr*
- `proj2/Makefile` - Create a makefile that refers to the kernel folder makefile
- `proj2/kernel/Makefile` - Create a makefile that points to the source code for the three syscalls
- `proj2/kernel/proj2Kernel.c` - Create a C file for the source code for the 3 syscalls. This file will use a skipList data structure of all the pids the syscalls are used upon. Inside each of the skipList nodes, there will be an array of all the syscalls the given pid is allowed to use. Additionally, there will be another array with the count data for each syscall. This data structure will therein act as the

access control list implementation for this project. The reason the decision for a skipList was made because I am most familiar with this data structure (and how to get it working in the kernel) after project one, however, I fear this may not be the most efficient choice for this scenario. The choice to sort by pids instead of syscall numbers was based entirely on efficiency. If I were to have a skipList sorted by syscall ids, I would have up to 436 nodes and possibly very long arrays, instead, sorting the skipList by process id (which have a max of 4194304 in our case) allows us to have more efficient usage.

- `long sbx421_block(pid_t proc, unsigned long nr)` - Checks if skipList is initialized, if not, initializes it. Checks for a lock, then checks permissions. Then traverses the skipList to see if the given pid already exists, if it doesn't, add it to the skipList. Then add syscall id to the node’s syscall array if the id doesn’t already exist. Then updates count variables accordingly
    - Returns EINVAL for bad parameters (e.g. pid or nr is out of range)
    - Returns EACCES if proc is non-zero, the syscall was not called as root, and the pid does not equal the current process id.
    - Returns 0 on success
- `long sbx421_unblock(pid_t proc, unsigned long nr)` - Checks if skipList is initialized, if not, initializes it. Checks for a lock, then checks permissions. Then traverses the skipList to see if the given pid already exists, if it doesn't, throw error. If it does exist, check if the syscall

    is in the block list. If it is, remove it. Then removes the corresponding count variable as necessary
        - Returns EINVAL for bad parameters (e.g. pid or nr is out of range)
        - Returns EACCES if proc is zero or the syscall was not called as root
        - Returns ESRCH if the provided pid is not in the access control list
        - Returns EIDRM if the syscall provided is not on the pid's block list
        - Returns 0 on success
  - `long sbx421_count(pid_t proc, unsigned long nr)` - This syscall will first traverse the skipList to see if the provided pid exists. If it is, it then checks to see if the provided syscall exists. If both these cases are true, it returns the count stored for that syscall.
      - Returns EIDRM if the syscall provided is not on the pid's block list
      - Returns 0 or the positive access count
- `arch/x86/entry/common.c` - Modify line 287 to have an if statement. This statement should check to see if the current pid with a given syscall id is in the access control list (this should be a simple skipList traverse similar to the other functions)
  - Returns EPERM if the pid does not have permission for the syscall

Userspace Code

- `proj2/userland/Makefile` - Create a makefile for the four syscall test programs
- `proj2/userland/sbx421_block.c` - Create a C file that passes the command line arguments to the sbx421_block syscall
- `proj2/userland/sbx421_unblock.c` - Create a C file that passes the command line arguments to the sbx421_unblock syscall
- `proj2/userland/sbx421_count.c` - Create a C file that passes the command line arguments to the sbx421_count syscall
- `proj2/userland/sbx421_run.c` - Create a C file that uses all three syscalls to run any other application (via a text file of provided syscalls) on the system with the credentials of another user.