CSCE 345 / 3401 – Operating Systems

Prof. Amr El-Kadi

Project II Report

Ahmed A. Agiza

Mohammed R. Anany

Introduction:

The project's goal is to implement a new system call that loops over the current processes' task_struct's, query the open net ports and report the listening processes back to the user space with number of open ports.

Members Roles:

The following are the main responsibilities of each group member, although each group member participated in every aspect of the project.

Ahmed A. Agiza: Querying the running processes and analyzing their resources.

Mohammed R. Anany: Studying the kernel structures, the syscalls and modules interfaces.

Implementation Steps:

1. Compiling the kernel source code and experiencing the configuration techniques.
2. Adding kernel module and syscall interface.
3. Studying the task_struct and how to obtain the required information.
4. Creating a system module that achieves the required task.
5. Turning the system module into a syscall interface.
6. Copying the data from the kernel space to the user space.
7. Final compilation and testing.

Source Code Documentation:

The following is a description of the syscall source code.

Main header files:

linux/printk.h: For prink() function to print data to the log since the tradition printf() does not function in the kernel space.

linux/sched.h: For the tast_struct data structure.

linux/linkage.h: Required for the syscall defintion.

linux/fs.h: For file operations.

linux/slab.h: For kmalloc() and kfree().

linux/net.h: For socket data structure.

net/sock.h:For socket data structure.

asm/uaccess.h:For copying data between kernel and user space.

Reported data struct:

The following is the definition of the struct used to report the data to the caller.

```
struct spy_struct{
   long pid; //ID of the queried process.
   long number_of_ports; //Number of open sockets by the process.
   char process_name[MAX_PROCESS_NAME]; //Process name.
};
```

Syscall API:
asmlinkage long sys_spy(struct spy_struct __user *spy_buf, long num_proc)
spy_buf: the buffer that will be filled with data.
num_proc: the maximum number of processes to query.
The syscall returns back the number of processes queries and negative number on error.

Variables definitions:

```
struct files_struct *task_files; //Files struct to access the files of each process.
struct fdtable *files_table; //File table for the files of the process.
struct task_struct *task; //Current queried process task_struct.
struct socket *sock; //Socket for socket file descriptors.
long i /**Loop counter**/, num_sockets /**Number of open ports by the process**/, p_count = 0;
//Number of queries process.
int socket_error = 0; //Error number when converting file to socket.
struct spy_struct *spys; //Reported data buffer.
```

Main code:

```
if (num_proc > 0) //Check that the maximum number of required processes is larger than zero.
for_each_process(task) //To loop of the current active processes.

task_files = task->files;
files_table = files_fdtable(task_files); //Getting the file descriptors owned by each process.

while(files_table->fd[i] != NULL) //Looping over file descriptors.

if(S_ISSOCK(files_table->fd[i]->f_path.dentry->d_inode->i_mode)) //Checking if the file descriptor
belongs to a socket through its inode.

sock = sock_from_file(files_table->fd[i], &socket_error); //Creating socket struct from the file
descriptor.

if(!socket_error && sock->type == SOCK_STREAM) //To count only valid net sockets.

if(num_sockets > 0 && p_count < num_proc){ //Filling the buffer with acquired data.
  spys[p_count].pid = task->pid;
  strcpy(spys[p_count].process_name, task->comm);
  spys[p_count].number_of_ports = num_sockets;
   p_count++;
}

if(access_ok(VERIFY_WRITE, spy_buf, num_proc * sizeof(struct spy_struct))) //Verifiying the abilitiy
to write the data back to the user space.

copy_to_user(spy_buf, spys, num_proc * sizeof(struct spy_struct)) //Copying the obtained data back to
the caller user space.
```

Userspace test program

```
struct spy_struct buf[10]; // The buffer that will be filled with the data.

long proc_spyed = syscall(314, buf, 10); //Calling the new syscall.

for(i = 0; i < proc_spyed; i++)
printf("Process %s(%ld) is listening to %ld communication(s) ports.\n", buf[i].process_name,
buf[i].pid, buf[i].number_of_ports); //Printing the obtained data.
```