Homework 1 Wet

Due Date: 24/04/2018 23:30

Teaching assistant in charge:

- Shalev Kuba

Important: the Q&A for the exercise will take place at a public forum Piazza only. Critical updates about the HW will be published in pinned notes in the piazza forum. These notes are mandatory and it is your responsibility to be updated. A number of guidelines to use the forum:

- Read previous Q&A carefully before asking the question; repeated questions will probably go without answers
- Be polite, remember that course staff does this as a service for the students
- You're not allowed to post any kind of solution and/or source code in the forum as a hint for other students; In case you feel that you have to discuss such a matter, please come to the reception hour
- When posting questions regarding hw1 , put them in the hw1 folder

Only the TA in charge, can authorize postponements. In case you need a postponement, contact him directly at 234123cs@gmail.com .

Introduction

Your goal in this assignment will be to add new system calls to the kernel's interface and to change some existing system calls. While doing so you will gain extra knowledge in compiling the kernel. Furthermore, in this exercise, we will use VMware to simulate a virtual machine on which we will compile and run our "modified" Linux. You will submit only changed source files of the Linux kernel.

General Description

We want you to restrict certain processes from calling specific system calls. To do so, you will add a new features to the Linux kernel in which every process is associated with a specific privilege level. Then, you will modify some existing system calls that you have learned about, such that only a process with an appropriate privilege level will be able to invoke them. Finally, you will create a logging service that records forbidden activities of processes in the OS.

The policy will be defined as follows:

- **Turning On The Policy Feature**: By default, all processes will not be affected by this feature. You will define a system call which get the PID of a process and turn on the feature for this process. If the feature is turned off for a process, then the process executes as normal and is not affected by anything in this homework exercise, regardless of its privilege level.
- **Privilege Level**: Each process should have a privilege level in the range of [0,2]. Processes by default starts with privilege level 2 when the policy feature is set on for them.
- **Setting the Privilege Level**: Every process may set the privilege level of any other process (including itself) through special system calls. These system calls require an administrator password which will be sent as a parameter.
- **Effects of the Privilege Level**:
    - Below is a list of some system calls that you have seen in class. For each of these, we define a minimum privilege level requirement.
    - For each of these system calls, if the calling process has a privilege level which is <u>strictly lower</u> than the threshold of that system call and the policy feature was set on for this process, then the call should fail and return -EINVAL. We define such behaviour as a **forbidden activity**.
      Clarification: A system call with threshold *x* can only be executed by a process which has a privilege level greater than or equal to *x.*
    - You should not enforce the policy in any other system call.
- **Recording Forbidden Activities**: You must define a separate log for each process. If a process attempts to perform a forbidden activity, then this activity should be recorded in the log of that process. Specific details on the implementation of the log are discussed below.

The system calls that interest us are:

System call	Minimum threshold
fork()	2
every kind of wait (wait() , waitpid(), etc.) **hint**- think which function is invoked by every kind of wait.	1
sched_yield()	1
every other system call	0

Detailed Description

You required to implement code wrappers and the corresponding system calls. For example: enable_policy is a code wrapper and sys_enable_policy is a system call (see tutorial 1, from slide 36 onwards).

The log records description

As mentioned before, we would like to record the forbidden activities of a given process.
For this purpose, we define the following struct that represents a forbidden activity log record:

```
struct forbidden_activity_info{
        int syscall_req_level;
        int proc_level;
        int time;
}
```

- **syscall_req_level -** stores the privilege level threshold of the involved system call.
- **proc_level** - stores the involved process's privilege level **at the time** the forbidden activity has been occurred. Note that process's privilege level might changed through time via ad-hoc system call, as we will discuss below.
- **time** - stores the clock ticks (in jiffies) in which the forbidden activity occurred.

For simplicity, the order of the forbidden activities in the log will be by time of occurring.

What happens when fork is invoked

When fork is invoked, the new process's privilege level should be set to the default privilege level (=2), and the policy feature should be set off (need to enable manually for each process). In addition, the new process's forbidden activity log should be empty.

Notes:

- We record only **forbidden** activities, so when we turn off the policy feature to a process, its log should be removed.
- You can assume that we will not enforce system processes.
- For the sake of the home assignment, please ignore handling of processes containing multiple threads. That is, assume tgid=pid for all processes, and ignore memory sharing issues.

Code Wrappers

system call number 243

int enable_policy (pid_t pid ,int size, int password)

Description

Enable the enforcement of the policy for process with PID=pid **only** if the password is indeed the administrator password. The privileged level of this process should be set to the default level. The forbidden activities log of the process will last up to size forbidden_activity_info objects. You can assume that this process will not make violations that will lead to more than size forbidden_activity_info objects.
Note: the administrator password is 234123.

Return values

- On success: return 0.
- On failure: return -1
    - If pid<0 errno should contains *ESRCH*
    - If no such process exists errno should contains *ESRCH*
    - If password is incorrect errno should contains *EINVAL*
    - If the policy feature already set on for this process, errno should contains *EINVAL*.
    - If size<0 errno should contains *EINVAL*
    - On memory allocation failure errno should contains *ENOMEM*
    - On any other failure errno should contains *EINVAL*

system call number 244

int disable_policy (pid_t pid ,int password)

Description

Disable the enforcement of the policy for process with PID=pid if the password is indeed the administrator password. As mentioned before, after invoking this system call, the log of process with PID=pid should be removed.

Note: the password is "234123" as mentioned above.

Return values

- On success: return 0.
- On failure: return -1
    - If pid<0 errno should contains *ESRCH*
    - If no such process exists errno should contains *ESRCH*
    - If the policy feature already set off for this process, errno should contains *EINVAL*.
    - If password is incorrect errno should contains *EINVAL*
    - On any other failure errno should contains *EINVAL*

system call number 245

int set_process_capabilities(pid_t pid,int new_level,int password)

Description

Set the privilege level of the process with PID=pid to new_level, if the password is correct. The log of process with PID=pid should not be affected at all.

Note: the password is "234123" as mentioned above.

Return values

- On success: return 0.
- On failure: return -1
    - If pid<0 errno should contains *ESRCH*
    - If no such process exists errno should contains *ESRCH*
    - If new_level is not 0,1,2 errno should contains *EINVAL*
    - If the password is incorrect errno should contains *EINVAL*
    - If the policy feature is set off for this process, errno should contains *EINVAL*.
    - On memory allocation failure errno should contains *ENOMEM*
    - On any other failure errno should contains *EINVAL*

system call number 246

int get_process_log(pid_t pid,int size,struct forbidden_activity_info* user_mem)

Description

Returns in user_mem the firsts $size_{th}$ records of the forbidden activities of the process with PID=pid. In addition, those $size_{th}$ records should be removed from the process's forbidden activities log.

Note: You can assume that user_mem have enough space to contain the answer.

Return values

- On success: return 0
- On failure: return -1
    - If pid<0 errno should contains *ESRCH*
    - If no such process exists errno should contains *ESRCH*
    - If size>number of records in the log, errno should contains *EINVAL* and the log shouldn't be affected at all.
    - If size<0 errno should contains *EINVAL*
    - If the policy feature is set off for this process, errno should contains *EINVAL*.
    - On any other failure errno should contains *EINVAL*

What should you do?

Use VMware, like you learned in the preliminary assignment, in order to make the following changes in the Linux kernel:

1. Update entry.S (add system call numbers and references in the syscall table).
2. Make any necessary changes in the kernel code so the new system calls can be used like any other existing Linux system call. Your changes can include modifying any .c, .h or .S (assembly) file that you find necessary.
3. Make necessary changes in file **fork.c**, **sched.h, sched.c and exit.c.**
4. Update more files if needed.
5. Recompile and run the new kernel like you did in the preliminary assignment.
6. Put the wrappers functions in **hw1_syscalls.h,** note that **hw1_syscalls.h** is not part of the kernel, the user should include it when using your system calls This also means you don't need to recompile the entire kernel when modifying the header.
7. Boot with your new Linux, and try to compile and run the test program to make sure the new system calls work as expected.
8. Submit **kernel.tar.gz**, **submitters.txt** and **hw1_syscalls.h** (see below)

Did it all? Good work, Submit your assignment.

Important Notes and Tips

- First, try to understand exactly what your goal is.
- Figure out which data structures will serve you in the easiest and simplest way
- Figure out which new states you have to save and add them to the task_struct (defined in **sched.h**).
- Figure out in which exact source files you need to place your code and where exactly in each file.
- Do not reinvent the wheel, try to change only what you really understand, and those are probably things related to the subjects you have seen in the tutorials.
- **Debugging the kernel** is not a simple task, use ***printk*** to print messages from within the kernel. Whenever possible - write and compile short and simple segments of code and make sure they work before expanding on them (For example, writing a preliminary system call that simply returns a number, and making sure that works, before adding functionality to it)
- The linux developers wrote comments in the code, read them, they might help you understand what's happening.
- You are not allowed to use syscall functions to implement code wrappers, or to write the code wrappers for your system calls using the macro _syscall1. You should write the code wrappers according to the example of the code wrapper given above.
- If there is more than one error when calling a syscall you should return the first one by the order in the function description.
- Write your own tests. We will check your assignment with our test program

- We are going to check for kernel oops (errors that don't prevent the kernel from continue running such as NULL dereference in syscall implementation). You should not have any.
  If there was kernel oops, you can see it in dmesg (dmesg it's the command that prints the kernel messages, e.g. printk, to the screen).
  To read it more conveniently use: dmesg | less -S
- Linux is case-sensitive. entry.S means entry.S, not Entry.s, Entry.S or entry.s.
- You can assume that the system is with a single CPU.
- You should use kmalloc and kfree in the kernel in order to allocate and release memory. If kmalloc fails you should return ENOMEM. For the kmalloc function use flag GFP_KERNEL for the memory for kernel use.
- Pay attention that the process descriptor size is limited. Do not add to many new fields. Also, add your fields at the **end** of the struct because the kernel sometimes uses the offsets of the fields.
- Start working on the assignment **as soon as possible**. The deadline is final, NO postponements will be given, and a high load on the VMWare machines will not be accepted as an excuse for late submissions

Submission

You should create a zip file (use zip only, not gzip, tar, rar, 7z or anything else) containing the following files:

a. A tarball named kernel.tar.gz containing all the files in the kernel that you created or modified (including any source, assembly or makefile).

   To create the tarball, run (inside VMWare):

```
cd /usr/src/linux-2.4.18-14custom
tar -czf kernel.tar.gz <list of modified or added files>
```

   Make sure you don't forget any file and that you use relative paths in the tar command. For example, use kernel/sched.c and not /usr/src/linux-2.4.18- 14custom/kernel/sched.c

   Test your tarball on a "clean" version of the kernel – to make sure you didn't forget any file.

   If you missed a file and because of this, the exercise is not working, you will get 0 and resubmission will cost 10 points. In case you missed an important file (such as the file with all your logic) we may not accept it at all. In order to prevent it you should open the tar on your host machine and see that the files are structured as they supposed to be in the source directory. It is highly recommended to create another clean copy of the guest machine and open the tar there and see it behave as you expected.

To open the tar:

```
cd /usr/src/linux-2.4.18-14custom
tar -xzf <path to tarball>/kernel.tar.gz
```

b. A file named **submitters.txt** which includes the ID, name and email of the participating students. The following format should be used:

```
Linus Torvalds linus@gmail.com 234567890
Ken Thompson ken@belllabs.com 345678901
```

c. Additional files requirements go here

Important Note: Make the outlined zip structure exactly. In particular, the zip should contain only the X files, without directories.

You can create the zip by running (inside VMware):

```
zip final.zip kernel.tar.gz submitters.txt hw1_syscalls.h
```

The zip should look as follows:

```
zipfile -+
        |
        +- kernel.tar.gz
        |
        +- submitters.txt
        |
        +- hw1_syscalls.h
```

Important Note: when you submit, **retain your confirmation code and a copy of the file(s)**, in case of technical failure. Your confirmation code is **the only valid proof** that you submitted your assignment when you did.

Have a Successful Journey,
The course staff