

Homework 4, Question 8

*"I have no special talents.
I am only passionately curious."
— Albert Einstein*

Environment

One way to solve this question is to actually try it out on a Unix system. I know at least three ways of how to set up a testing environment without affecting my main computer:

1. You can download Oracle's VirtualBox

<https://www.virtualbox.org>

There are binaries for Windows and MacOSX (I only tried out MacOSX). In addition, you need to download a Linux distribution. I used a recent iso-file of an Ubuntu distribution. All components are free.

2. If you happen to have a Raspberry Pi laying around (I have two for playing music as well as for all sorts of rainy-afternoon distractions). The cheapest model of a Raspberry Pi costs around £20. You also need an SD memory card of at least 4GB, which can be bought for £5 or less. They come pre-installed with Linux or can be easily loaded with Linux. The good thing about Raspberry Pi's is that despite their miniature size and small cost, they are full-fledged Linux computers...exactly what is needed for such experiments. There are plenty Linux distributions on the Net that are tailored to work with Raspberry Pi's.
3. If you have a spare memory stick laying around, you can try out any of the live USB-versions of Linux.

https://en.wikipedia.org/wiki/Live_USB

The idea is you plug the USB stick into your computer and boot up a Linux system without having to download anything to your computer. A notable live USB version of Linux is called Tails

<https://tails.boum.org>

which comes with Tor pre-installed and is for people who need a maximum of privacy and anonymity (whistleblowers, dissidents). It is supposed to give them anonymity even if their main system is compromised by malicious software, for example installed by the NSA.

However, a live USB Linux will need some support from the computer (BIOS) where you plug in the USB stick. I know Apple computers are a bit “special” with this and would need a 3rd-party boot loader for loading operating systems from an USB memory stick.

For my experiments below, I used option 2. In the past I have used option 1. I have not tried in a while option 3, but know that in the past I had a dedicated bootloader on an Apple computer just for the purpose of running operating systems from external disks.

Setup

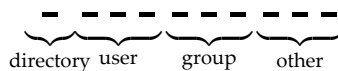
There are a few commands you need to know in order to replicate the ownerships and permissions in the question:

- `useradd` creates a new user
- `groupadd` creates a new group
- `adduser` adds a user to a group
- `chmod` changes the permissions of a file
- `chown`, `chgrp` change the ownership and group of a file

There is also a choice to be made what to use as microedit. I recommended to use `vi`, which is available on pretty much every UNIX system. For a first try out, this is a helpful choice for solving the question.

Permission Basics

The absolute basics is how the permissions are organised in essentially four blocks



Assume a file is owned by Bob with permissions

```
-r--rw-rwx bob students file
```

The access rules imply that Bob will only have read access to this file, even if he is in the group students and the group access rights allow read and write. Similarly every member in the students group who is not Bob, will only have read-write access rights, not read-write-execute.

The question asked whether Ping, Bob and Emma can read or write the given files **using** the file microedit. This means we will call on the command line

> `microedit file`

for all files and for Bob, Ping and Emma. So if you want to find out whether Bob, say, can read or write a file, you need to find out what the access rights with which `microedit` is run. This would be trivial, if `microedit` did not have the `setuid` bit set. Then it would be the rights of the caller (Ping, Bob or Emma). But your friendly lecturer arranged the question so that it has.

Recall that the `setuid` bit gives the program the ability to run with the permissions of the owner `microedit` file, not the permissions of the caller. I wrote in the handout

“The fundamental idea behind the `setuid` attribute is that a file will be able to run not with the callers access rights, but with the rights of the owner of the file.”

Something similar is written in the Wikipedia entry for `setuid`

<http://en.wikipedia.org/wiki/Setuid>

This implies for deciding whether `file` is readable or writable is not determined by the caller, but by the permissions with which `microedit` runs. There is one caveat however: We need to find out first whether the caller (Bob, Ping or Emma) can actually run `microedit`—that is have execute rights for `microedit`. Once `microedit` runs, it will assume the rights of the owner of `microedit`. The question is now whether these permissions are sufficient to read or write the `file`. This should already be useful for answering the first three columns.

For the other two files we have to take into account that they are inside a directory. For directories apply special access rules. In the handout I wrote

“There are already some special rules for directories and links. If the `execute` attribute of a directory is not set, then one cannot change into the directory and one cannot access any file inside it. If the `write` attribute is not set, then one can change existing files (provide they are changeable), but one cannot create new files. If the `read` attribute is not set, one cannot search inside the directory (`ls -la` does not work) but one can access an existing file, provided one knows its name.”

With this also the last two columns can be filled in.

There is one further complication arising from the `setuid` bit. The question asked:

...whether Ping, Bob, or Emma are able to obtain the right to read (R) or replace (W) its contents using the editor `microedit`.

Note the underlined phrase. That means we need to be certain that there is no other way for Bob, Ping and Emma to obtain reading or writing permissions. Actually there is. Any file that has the `setuid` bit set will be called with the

rights of the owner, but once it has done the work, it can lower the permissions again to the callers rights. This is a second “avenue” we have to check whether the files become readable or writable. In the handout I wrote about `setuid` programs:

“As an example consider again the `passwd` program. When started by, say the user `foo`, it has at the beginning the identities:

- real identity: `foo`
effective identity: `foo`
saved identity: `root`

It is then allowed to change the effective identity to the saved identity to have

- real identity: `foo`
effective identity: `root`
saved identity: `root`

It can now read and write the file `/etc/passwd`. After finishing the job it is supposed to drop the effective identity back to `foo`. This is the responsibility of the programmers who wrote `passwd`. Notice that the effective identity is not automatically elevated to `root`, but the program itself must make this change. After it has done the work, the effective identity should go back to the real identity. ”

A Test Program

I suggested above to use a copy of the editors `vm` or `vim` for `microedit`. This works well except in one instance: if a file is not readable, then these editors will not be helpful for checking whether the file is writable. In Figure 1 is a little C program that explicitly checks for readability and writability of files. It is organised into two parts: the first checks readability and writability with the permissions according to the `setuid` bit, and the second when the rights are lowered to the caller.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(int argc, char *argv[])
6 {
7     FILE *f; //file pointer
8
9     printf("Real UID = %d\n", getuid());
10    printf("Effective UID = %d\n", geteuid());
11
12    //read test
13    if ((f = fopen(argv[1], "r")) == NULL) {
14        fprintf(stderr, "%s is not readable\n", argv[1]);
15    } else {
16        fprintf(stderr, "%s is readable\n", argv[1]); fclose(f);
17    }
18
19    //write test
20    if ((f = fopen(argv[1], "w")) == NULL) {
21        fprintf(stderr, "%s is not writable\n", argv[1]);
22    } else {
23        fprintf(stderr, "%s is writable\n", argv[1]); fclose(f);
24    }
25
26    //lowering the access rights to the caller
27    if (setuid(getuid())) {
28        fprintf(stderr, "Could not reset setuid\n"); return 1;
29    }
30
31    printf("Real UID = %d\n", getuid());
32    printf("Effective UID = %d\n", geteuid());
33
34    //read test
35    if ((f = fopen(argv[1], "r")) == NULL) {
36        fprintf(stderr, "%s is not readable\n", argv[1]);
37    } else {
38        fprintf(stderr, "%s is readable\n", argv[1]); fclose(f);
39    }
40
41    //write test
42    if ((f = fopen(argv[1], "w")) == NULL) {
43        fprintf(stderr, "%s is not writable\n", argv[1]);
44    } else {
45        fprintf(stderr, "%s is writable\n", argv[1]); fclose(f);
46    }
47
48    return 0;
49 }

```

Figure 1: A read/write test program in C.