CS349 Computer Security **Handout # 3**
Prof. Lyn Turbak **September 24, 2004**
Wellesley College

Homework 3
Due: Anytime Tuesday October 5

Reading:

- Bishop, "How to Write a Setuid Program"
- Chen, Wagner, & Dean, "Setuid Demystified"
- Bishop *Computer Security*, Chapters 25 (Intrusion Detection; skip the network parts) and 28 (User Security)
- Hatch, Lee, and Kurtz, *Hacking Linux Exposed*, Chapters 2 (Proactive Measures), 8 (Elevating User Privileges), and 10 (How Hackers Maintain Access).

You may (but are not required to) work on all parts of this problem with your sysadmin partner. If you do work as a pair, turn in only one hardcopy submission for the pair.

Problem 1: Computer Forensics
In this problem, please do not *consult with other teams. Try to figure out everything between the two members of your team.*

a. You've been attacked — I know all of your root passwords! This problem is to figure out how I managed to get them. To help you out, I've left behind a number of clues that a good hacker would not normally leave behind. Put on your detective hats, look for clues, and write up a description of exactly what I did to steal your root passwords. Linux's `find` and `grep` commands are particularly helpful for this task.

b. I could have made it much more difficult to find out what I did by covering my tracks better. Explain all the steps I could have taken to make your detective work tougher.

c. I did clean up *some* of my tracks. Are there clues you expected to find but did not?

d. Repair your system so that I cannot monitor your future password changes. Describe how you cleaned things up.

e. I have left behind other "backdoors" into your system so that I can still gain root access to your machines even after you clean up and change your root password. Can you find them?

Problem 2: Setuid
You should read (or at least skim) the two Setuid papers and the setuid/setuserid sections of Hacking Linux Exposed *before attempting this problem.*

a. What Linux command can you execute to set the setuid bit of a program `foo`? What do the permissions of `foo` look like after you set the command?

b. Describe a situation in which setting the setuid bit of a program is helpful for making the program work properly.

c. Setuid programs are a potential security risk. Why?

d. Linux ignores the setuid bit on shell scripts. Why did the Linux designers make this decision?

e. Write a shell script that finds all of the setuid programs on your machine. Do any look suspicious?

Problem 3: Access Control Design

The goal of this problem is to give you practical experience with Linux permissions and shell scripts.

Professors Foo Bar and Baz Quux are co-teaching introductory course CS1. They and all of their students will be using your (one!) computer in 121B for all of their work. They have asked you to administer all of the accounts for their course. This includes the accounts of the two professors, as well as a course account (`cs1`) and the accounts of all the students. For simplicity, assume that there are only two students (`stud1` and `stud2`). However, your design should scale to a large number of students.

The `cs1` home directory should be in `/home/cs1` and it should have three subdirectories: `public`, `private`, and `drop`. The `public` directory should be visible to everyone in the class. The `private` directory should only be visible to the two professors. All the contents of `drop` directory should be visible to the two professors. As we shall see below, individual students will "own" certain subdirectories within the `drop` directory, but no student should be able to see the directories of any other student.

Professors Bar and Quux desire the following Linux commands for their course:

- `publish` *`dir psname`*: This is a command that the professors use to publish directories of code and instructions for their problem sets. Only the two professors should be able to execute this command. It copies the contents of the directory *`dir`* to the directory `/home/cs1/public/`*`psname`*, creating the latter if it does not already exist. It also creates the directory `/home/cs1/drop/`*`psname`* if it does not already exist. The whole class should be able to read each file within the directory `/home/cs1/public`, and no one (not even the professors) should be able to edit such files. No one outside the course should be able to read any of the files in `/home/cs1/public` (though they might be able to list them). Calling `publish` with a *name* that already exists in `/home/cs1/public` will overwrite any files in `/home/cs1/public/`*`name`* that happen to have the same name as a file in *dir*. The name *dir* should not end in a slash, and the name *name* should not contain any slashes.

- `privish` *`dir name`*: This is a command that the professors use to share directories of problem set solutions with each other. It can only be executed by the professors. It copies the contents of the directory *`dir`* to the directory `/home/cs1/private/`*`name`*. Only the professors should be able to read and change the files within `/home/cs1/private` that have been added by calling `privish`. Calling `privish` on a directory that already exists in `/home/cs1/private` should overwrite the previous directory. The name *dir* should not end in a slash, and the name *name* should not contain any slashes.

- `drop` *`dir psname`*: This is a command that can be executed by all CS1 students, as well as by the professors, but by no one else. It makes a copy of the directory *`dir`* and stores it in the directory whose name is `/home/cs1/drop/`*`psname`*`/`*`username`*, where *`username`* is the name of the user executing the command. The command signals an error if `/home/cs1/drop/`*`psname`* does not already exist. If `/home/cs1/drop/`*`psname`*`/`*`username`* already exists, the old contents are overwritten. A student should be able to read and write any directory that they have submitted via `drop`, but they should not have read or write access any `drop` directory they have not submitted. However, the professors should have read and write access to *all* subdirectories of the `drop` directory. The name *dir* should not end in a slash, and the name *psname* should not contain any slashes.

In this problem, you should create accounts for the professors, students, and course, and then should write `bash` shell scripts (or `perl` programs, if you prefer) for the above three commands. You

will need to think carefully about ownership, groups, permissions, and paths in order to implement the commands. Where should the commands "live"? For `bash`/`perl` programming, consult one of the many books or online resources on the topic. Yes, you are expected to just "pick up" one of these languages on your own. This is what you are expected to do in the real world! But you certainly don't need to learn the whole language — just enough to write the three scripts.

This is a challenging problem, and requires careful thought. Here are some notes you may find helpful (see Linux documentation for more details):

- The "sticky bit" of a directory can be changed via `chmod +t` or `chmod 1`*???* (where each *?* can be any octal number). Any user with write access to such a directory can add, modify, and delete files that she owns, but cannot modify files owned by others. Without the sticky bit, a user with write acces to a directory can modify and delete any files within the directory, regardless of who owns them.

- The "setgid bit" of a directory can be changed via `chmod g+s` or `chmod 2`*???* (where each *?* can be any octal number). When the setgid bit of a directory is set, any files created within the directory will inherit their group from the directory rather than from the user creating the file.

- Many aspects of the problem would be simplified if it were possible use setuid with scripts. But Linux does not permit this because of security issues in using setuid with scripts. It is possible to circumvent this problem by writing a C program that invokes the script, and performing setuid on the executable program produced by this process. You are not expected to use this technique in this problem (but can if you want to).

- Within a shell script, any text delimited by single backquotes indicates a command that should be executed and replaced by its result. For example, if `stud1` executes a script containing the filename `/home/‘whoami‘/stuff`, this filename "expands" to `/home/stud1/stuff`.

Test your commands to make sure that they work as you expect. Turn in your shell scripts and testing transcripts that show your commands work as specified. I.e., the commands have the right behavior and all relevant parties can only execute the commands and read/write files to which they should have access.