EITA25 - Lab 2

André Frisk & Hugo Bläckberg

February 17, 2022

Preparatory Questions

1. - **Ensure that you know what do to when we ask "Read the manual pages for login(1)". Read Section B.0.1 if you are unsure.**: This command refers to the file format of the file `/etc/passwd` and not the `passwd` user program.
   - **Read the manual pages for login(1) and passwd(5). Make sure you understand the format of /etc/passwd**: `login(1)`: This is a program that establish a new session of the system. It is normally invoked automatically by responding to the login: prompt of the user's terminal. When called from shell, login should be executed as exec login which will cause the user to exit from the current shell (and thus will prevent the new logged in user to return to the session of the caller). The user is prompted for a password, echoing is disabled to prevent revealing the password and only a small number of password failures are permitted before login exits and the communication link is severed. `passwd(5)`: The passwd directory contains one line for each user account with seven field, these are:
     - login name
     - optical encrypted password
     - numerical user ID
     - numerical group ID
     - user name or comment field
     - user home directory
     - optional user command interpreter

     If the password field is a lower case then the encrypted password is actually stored in `shadows(5)`, if there is not a corresponding line in the shadows directory then the user is invalid. The encrypted password may be empty, in which case no password is required to authenticate as the specific login name. However, some applications which read the /etc/passwd file may decide not to permit any access at all if the password field is blank. A password field which starts with an exclamation mark means that the password is locked.
2. **Look at the contents of the supplied pwfile, and compare it with the format used in the real /etc/passwd. What are the differences? The format of the real /etc/passwd is described in the lecture slides, and also in the man page passwd(5). The format of the pwfile can be found by looking in pwfile, and by checking the definition of struct pwdb passwd in the downloaded file pwdblib.h.**:

- The format of the **pwfile**:

```
donald:0BWGsAnLYCtBU:2001:2001:Donald Duck:/home/donald:/bin/sh:0:0
daisy:.izm4aS5j2Tuo:2002:2002:Daisy Duck:/home/daisy:/bin/bash:0:0
scrooge:IKrnhK9XPi5/Q:2003:2003:Scrooge McDuck:/home/scrooge:/bin/sh:0:0
mickey:Pvxs5lpD.Ls4s:2004:2004:Mickey Mouse:/home/mickey:/bin/sh:0:0
minnie:FQsAWMXsJLDD2:2005:2005:Minnie Mouse:/home/minnie:/bin/bash:0:0
goofy:X2iOq64zClyug:2006:2006:Goofy:/home/goofy:/bin/bash:0:0
```

  The structure of saved passwords where the following:

```
struct pwdb_passwd {
    char *pw_name;   /* users login name */
    char *pw_passwd; /* 2 bytes salt value + encrypted password */
    int pw_uid;      /* user id */
    int pw_gid;      /* group id */
    char *pw_gecos;  /* real name */
    char *pw_dir;    /* home directory */
    char *pw_shell;  /* preferred shell */
    int pw_failed;   /* number of contiguous unsuccessful logins */
    int pw_age;      /* number of successful logins */
};
```

- **/etc/passwd**: The structure of saved passwords where the following:

```
username:password:UID:GID:ID string:home directory:login shell
```

  Examples with this format is:

```
alice:x:1004:100:Alice:/home/alice:/bin/bash
bob:x:1005:100:Bob:/home/bob:/bin/bash
```

3. **Study the downloaded program userinfo.c. Copy it to a new file called mylogin.c. Rewrite it so that it mimics the system login procedure, which for example includes that it should not echo the password as you type it on the screen. The program should check the password entered by the user with the corresponding entry in the pwfilefile. Useful library functions might include pwdb_getpwnam, getpass(3), crypt(3) and strcmp(3). You may ignore warnings that getpass(3) is deprecated. If the passwords match, the program should write something like "User authenticated successfully" and terminate. If the password is wrong, or the username invalid, it should respond "Unknown user or incorrect password." and start over with the "login:" prompt.**: See file mylogin.c

4. **Make a copy of your original mylogin.c-file, and implement the following new features in the new file. For an example of how you update the pwfile, look at the provided example in update user.c**: See the mylogin2.c file for these features,

   - **The field pw_failed should be used to count the number of unsuccessful logins. Each time an unsuccessful login is encountered this counter should increase by one. When the right password is entered, pw_failed should be reset to zero**
   - **The field pw_age should be used to count the number of successful logins.**

When the age is greater than some value, e.g. 10, the user should be reminded to change his or her password. This field would be zeroed by a program corresponding to passwd(1), which you don't have to write.

- **Use the pw_failed counter to lock out a user account which has entered the wrong password more than say 5 times in a row. The account locking can be made in several ways, implement it as you like but remember that it should be easy for the administrator to enable the account again. So erasing the entry in pwfile is not a great idea...**

5. - **Study the program openshell_demo.c. Make sure you understand how the program works. Remember that when you call fork(2), both the parent and the child process continues execution in the program, but fork returns different values to the parent and the child, so that you can separate the line of execution.**: Creates a new child process which starts a xterm window. The parent just waits for the child to complete, and then exits. fork() creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process. The child process and the parent process run in separate memory spaces. At the time of fork() both memory spaces have the same content. _exit() terminates the calling process "immediately". pid==0, child runs a terminal emulator and immediately terminates if the child returns. pid<0, couldn't create a new process. pid>0, parent wait's for child to complete.
   - **Read about the functions setuid(2), seteuid(2), setgid(2) and setegid(2)**:
     - **setuid(2)**: sets the effective user ID of the calling process. sets the effective user ID of the calling process. Unprivileged processes may only set the effective user ID to the real user ID, the effective user ID or the saved set-user-ID.
     - **seteuid(2)**: sets the effective user ID of the calling process. Unprivileged processes may only set the effective user ID to the real user ID, the effective user ID or the saved set-user-ID.
     - **setgid(2)**: sets the effective user ID of the calling process. Unprivileged processes may only set the effective user ID to the real user ID, the effective user ID or the saved set-user-ID.
     - **setegid(2)**: sets the effective user ID of the calling process. Unprivileged processes mayonly set the effective user ID to the real user ID, the effective user ID or the saved set-user-ID.
6. **Which commands are used to change an owner and permissions of a file?**: The `chmod` command enables you to change the permissions on a file. You must be superuser or the owner of a file or directory to change its permissions. The command input can be in the form -rwx-w-r– which means that the user have read, write and execute permission while group user have only write and anyone beside the user and group have read only.
7. **Read about ACL. Which commands are used to set and read ACL in Linux?**: ACL provides an additional, more flexible permission mechanism for file systems. It is designed to assist with UNIX file permissions. ACL allows you to give permissions for any user or group to any disc resource. The commands used to set and read ACL is **setfacl** & **getfacl**.

Lab Questions

1. **Why is the text not echoed on the screen when the user enters the password?**: To protect the user from revealing the password length
2. **Why do we use the same error message for both "Invalid user" and "Wrong password"? Would it be a problem to print two different error messages instead?**: Security, if you type in wrong username or password the program will print out "Invalid user or password", this is because that the user or attacker trying to login should not know if the username or password is right or wrong, which could potentially risk the account into guessing right username and then password.
3. **Compile and run your program mylogin.c with the functionality we have described so far. Remember to link the crypt library to your program (the -lcrypt option) if you use the crypt(3) function, and to also compile the pwdblib.c-file. Does it work as you expect? See below for some username/passwords to test with**: Yes, but you have infinite amount of tries to login
4. **Why is it important to have a pw_failed counter? Considering an on-line attack, propose some other countermeasure that could easily be implemented.**: To lock-out someone (brute force) if someone tries to access your account. Two-factor authentication
5. **Can you think of any problems with locking a user account automatically as done in this exercise?**: Maybe some kind of warning, before getting locked out. Timed lock-out?
6. **Try to press Control-C at various times in your program. Does the program terminate every time? When does it terminate and when does it not? Why? (You will need to test this on Linux, not Mac OS X!). Why is it bad if we can terminate the login program?**: Terminate in login, can not do it in password input (get_pass is a kernel function). It is bad such as a potential attacker might get access to the computer when terminating the program.
7. **What is the difference between real user and effective user of a program? Describe the circumstances under which they are different/equal?**: The distinction between a real and an effective user id is made because you may have the need to temporarily take another user's identity. The real user id is who you really are (the one who owns the process), and the effective user id is what the operating system looks at to make a decision whether or not you are allowed to do something.
8. **Why doesn't your program switch to the correct user and group?**: They does not exist in the computer passwd file.
9. **Run ls -l and write down the permissions and owner of mylogin**: -rwxrwxr-x linuxuser
10. **What does niceify do? Is it a secure thing to have lying around in a system? Hint: Check the permissions and owner again—compare it to your answer from Problem 9.**: -rwsrwxrwx, root

    niceify changes the access process ID from real to effective, gives all users access. Owner changed to root.
11. **Can you do the following with this permission**:

- **Make the subdirectory the current directory with cd**: Yes
- **List the subdirectory**: No
- **Display the contents of welcome.txt**: Yes (cat, executable?)
- **Create a copy of welcome.txt in the subdirectory**: No

12. **Repeat the experiments from the question above for the following three cases. If you want to, you can fill in the results in the fancy table below**:
    - **The subdirectory has read permission only**: No, Yes, No, No
    - **The subdirectory has both read and execute permissions**: Yes, Yes, Yes, No (write access)
    - **The subdirectory has read, write, and execute permissions**: Yes, Yes, Yes, Yes
13. **Which flags need to be set on a directory in order to create new files?**: write and execute
14. **Which files do you think Alice can read among the other files? Hint: use command id to check user and group names of the current user**: File 13 (+ thingy)
15. **Was your initial guess correct? Explain why you can or can't read specific files. Write down the content of the read file. Hint: pay attention to plus sign in permissions, what does it mean?**: Yes. ACL_0HztU/&. Why you can read file 13 is because of the plus sign which is ACL which gives more permissions. Because you don't have access
16. **Create a file with arbitrary content, make sure that the owner and the group of the file is alice. Set permissions for this file to -rw-rw—-. Using ACL, allow user bob to read this file and write to it.**: -rw-rw—-+ <– which means that there are exclusive access in the ACL, which is to Bob.
17. **Why the program doesn't work properly? Fix permissions so that the program can correctly switch effective user to alice. Ignore hello.c file in the directory**: Something was wrong, SUID not correct. niceify used to change effective user to Alice.