Security Engineering, Assignment 0

Due date: Feb 10, 2016

Total points: 30

As a part of the assignment you would need to create a rudimentary shell that allows multiple users to login simultaneously (hint: Multi-threaded TCP listening server). For now, you could log in merely with usernames (no passwords needed for now). You could log in with usernames like – u1, u2, u3…uN. Your system must have a rudimentary filesystem which could have a directory dedicate to storing separate home directories, corresponding to each user. Let's call it 'simple_home'. Inside 'simple_home' you could have the directories corresponding to the users 'simple_home/u1', 'simple_home/u2', etc. The 'simple_home' should further be inside a 'simple_slash' directory, that represents the mounted file system that stores the 'simple_home' directory and all its subdirectories.

A group (like UNIX/Linux systems) would be associated with each of the users. The username and the corresponding group name (which would be same as the user name) could be stored in a file within the system.

Once logged in, each user is presented with a shell wherein he could execute the following commands – 'ls', 'fput','fget', 'create_dir' and 'cd'.

'ls': Takes argument as directory name and prints out the contents of the directory (other files). The argument could be an absolute or a relative pathnames. The only difference from regular 'ls' command would be that it must not display the underlying file permissions. It should display the username and group name associated with the file (corresponding to the user who created it).
'fput': Takes argument as filename (with absolute or relative path) and populates it with characters that are input from 'stdin'. Additionally, the 'fput' command can be used to append to existing files. First time the file is created, the 'fput' program must also prompt the user to input owner and group name.
'fget': Takes argument as filename (with absolute or relative path) and prints its contents to 'stdout'. There are however some conditions regarding "who" can access which file. The owner of file can fully access (both READ and WRITE) the file. If the user is a group member, corresponding to the group of the file, then he/she can only READ the contents of the file, and not read them.
'create_dir': creates a directory (within the home directory) or recursive inside another directory. Sub-directories or files in a certain directory must inherit the user and group associations from the parent directories. Here again, the users must input the owner and group for the file.
'cd': The 'cd' command can be used to switch between multiple directories, much like the usual 'cd' command present in most OS platforms.

As mentioned above, a user can read and write files/directories created by him/her, while he/she can only read files/directories that he hasn't created but is a member of the group corresponding to the file.

You would also need a client program to connect and communicate to the multi-user shell (running as a server).

Your program should typically check all possible corner cases related to basic permissions and access control, e.g. issues and bugs related to multiple user login, possible attempts to brute force the login attempt, crash the system etc.

Feel free to consider all possible assumptions. DO NOT forget to list the assumptions in the system description that you are required to deliver.

What you are supposed to submit:

1. C source code for the aforementioned shell server and client programs.

2. Makefile through which one could compile these programs. (see: http://mrbook.org/tutorials/make/ for a simple tutorial on how to generate Makefiles).

3. A write up of what your system does, what all assumptions you made, the inputs that you used to test your program and all the errors that you handled, the Threat Model that you considered and how your system defends against those.

How you would be graded:

1. Successful compilation using Makefile – 10 points
2. Use of system calls like – fork(), exec() (and its "siblings"), opendir(), readdir() or libraries like the pthread library – 3 points
3. Successful execution/error handling for each of the shell commands – 5 points for each command, total 20 points

4. Successfully defend against atleast 3 attacks/bugs/errors – 10 points (List the bugs/errors/attacks that you defend against)

5. Description of the systems, commands to execute and test the program and the assumptions that you made. – 7 points