Security Engineering, Assignment 2

Due date: Feb 13, 2015

Implementing ACLs using mediated access (Total points 50)

The objective of this assignment is to familiarize you with using setuid() (and related) system call(s). In this assignment you would need to use your existing laptop and desktop and create multiple real users – say call them 'larry', 'bill', 'steve', 'mukesh', 'azim, 'mark' *etc*. (whatever you feel like). You could create them using commands like – 'adduser'. Their home directories should be inside the simple_slash directory that you created earlier. The directories could have names like simple_slash/home/larry (for user larry). You need to accordingly modify the /etc/passwd file to indicate the home directories for each of these users. The directories 'simple_slash' and 'simple_slash/home' should have 'root' as the owner. You could additionally create a group called 'simple_slash' and the directory 'simple_slash' and 'simple_slash/home' could have their groups set to 'simple_slash'. The files and directories in each of the users' directories should be owned by the users themselves (and not by 'root'). *E.g.* simple_slash/home/larry should have the owner 'larry'. You could add some of these users to the group 'simple_slash' to have intra-group permissions (*e.g.* permission for 'steve' to read 'azim's' files and/or vice versa).

To access files in each other's directories, one may either have the regular DAC permissions or may need to do so through 'setuid' root programs which, having access to all files and directories, would mediate the access of the accessing user and the accessed files. These programs could be in a directory called 'simple_slash/bin', and should have the owner as 'root'. These programs, with 'setuid' bit set, should be able to switch the user to the accessed files'/directories' owner. *E.g.* if larry does not have access to joe's files (they may have permissions -rw-------), then larry may access the 'ls' program in the 'simple_slash/bin' directory to access the said files. The special 'ls' program, being a setuid program would execute as 'root' even when a non-root user executes it. This special 'ls' would need to switch to the userid of the accessed program's owner, before accessing it (principle of least privileges). Similar to assignment 1, you would also need programs like 'fput', 'fget', and 'create_dir' which should have the same functionality as earlier, but would now be setuid root programs mediating access to files and directories.

Additionally, you also need to implement ACLs which should work side by side with Linux DAC. These ACLs would be stored with the files (since we are not implementing a FS). One way to do this is to use a C structure / class that has an array of strings representing ACLs, along with a pointer to a character buffer which can be expanded or contracted based on the number of files in the file. *E.g.*

```
struct file_data{
  unsigned char **acl; // ACL strings
 unsigned int acl_len; // Number of acl strings
 unsigned char *data; // File data
 unsigned int data_len; // Number of bytes of 'data'}
```

Access to the ACL would be via programs called ‘setacl’ and ‘getacl’. You need to implement your own semantics for ACLs to prioritize it over DAC. Whatever semantics you may choose, you would need to document them. Only a file/directory owner should be able to modify the ACLs associated to its file or directory.

Each directory should have a ‘.acl’ file that stores the ACLs associated with each directory. You need to have semantics to inherit permissions for each directory and its subdirectory.

Every access needs to a file or a directory must be via these special commands which need to check the DAC permissions (using stat() system call(s)) and the ACLs associated to the files/directories to determine if the accessing user could be allowed to access the requested resource or not.

The system should also print the actual userID/username of the effective user immediately before and after accessing the file or directory (just to make sure that the programs are working correctly). You would need to use the setuid() family of functions to achieve this functionality.

You don’t need a remote login feature for this assignment. However, you still need to check for every corner case with regards to the functionality of access controls.

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.
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 – stat() and setuid() to implement the aforementioned functionality, along with the system calls used previously like readdir(), opendir(), read(), write() etc. – 5 points.
3. Successfully handling various access control scenarios – e.g. users attempting to access a file which the DAC doesn’t allow by the ACLs allow or vice versa -- 20 points
4. Successfully defending 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. – 5 points