Security Engineering, Assignment 2

Due date: Feb 10, 2016

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 would add these using the actual 'adduser' command. 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'. Additionally you would also require a new user called 'fakeroot' which would be have very similar to the actual 'root' user. The purpose of this would be clear in the next paragraph.

The commands 'fput', 'fget', 'create_dir' and 'ls' would now be individual programs which should have the **setuid** bit enabled. The actual owner of these programs would the 'fakeroot' user which you created above. Whenever a user runs these programs, the program would check the actual user ID of the calling process (using getuid() function). If the actual user is not 'fakeroot', the system would apply the DAC permissions when accessing the files and directories. Otherwise, if the actual user is 'fakeroot' the system actually allows full access to the files and directories (which are passed as arguments to the programs).

You also have to implement a 'chmod' program which a user could use to modify his or permissions. This program would have the setuid bit enabled, having similar properties as the aforementioned commands.

Additionally, you also need to implement ACLs which should work side by side with DAC. These ACLs would be stored with the files. 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 or vice versa. 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 to a file or a directory must be via these special commands which need to check the DAC permissions 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 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 – 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