FACULTY OF INFORMATICS

ASSIGNMENT SPECIFICATION

Assignment 1(12%)
You are to implement a simple "file system" with login authentication and access control. Specifically: Files can be created, read from, written to, but only in accordance with a three–level access control model (0,1,2 - BLP). You do not need to have an actual file system, simply a collection of records at the levels specified. Although in practice we would use a client/server system, here we will simply simulate the transmission process. You can implement the program in C++, C or Java. You need to provide compilation instructions for your code.

The initialisation details

Your program will, initially, need blank files salt.txt and shadow.txt. Run your FileSystem with the instruction FileSystem –i

This program should prompt for a username, something like...
Username: Bob
Check if the username exists already. If it does, terminate the program with an appropriate notification to the user. If it doesn't, request a password, with something like

Password: ....
Confirm Password: .... <Set your own password policy; min 8 characters, 1 symbol, and 1 digit>

Assuming the passwords are the same, we make a final request of the user, something like ...
User clearance (0 or 1 or 2): 1
Once we have this information we can modify the passwd.txt, salt.txt and shadow.txt files to include this user.

To
salt.txt we add a line, with a generic example and a specific one for user Bob given here:
Bob:38475722
Username:Salt
where Salt is a randomly chosen string of 8 digits. It is fine to use a time seeded rand() to generate the Salt, although this is cryptographically unsafe.
We also add a line to *shadow.txt*, with a generic example and a specific one for user Bob given here:
Bob:dd2da44f4437d529a80809932cb3da83:0
Username:PassSaltHash:SecurityClearance
PassSaltHash is generated as the MD5 hash of the concatenation of the user's password with the salt, For example if the Password is "alphabet" and the Salt is "12345678", we would pass "alphabet12345678" to the MD5 function. The file salt.txt is nominally associated with the client, while the file shadow.txt is nominally associated with the server, although here both the client and server are in the same program. *Passwd.txt* file has to be created according to the Ubuntu linux format.

Logging in

Running FileSystem with no arguments will allow a user to try and log in to the file system.

Username: Bob

Password: .... <password must be silent>

The ”client” part checks if the Username is listed in the file salt.txt and passwd.txt. If the Username is in the file, then their salt value is retrieved and the PassSaltHash is generated. A message should be displayed to indicate that the salt has been retrieved.

Bob found in salt.txt and passwd.txt

salt retrieved: Salt

hashing ...

hash value: PassSaltHash

The ”server” part should now compare the PassSaltHash value with that in the file shadow.txt. If salt.txt doesn’t contain an entry for the Username, or if the information in shadow.txt doesn’t match the transmitted information, FileSystem should stop with appropriate error messages. If the shadow.txt information matches, the clearance of the user is reported, and authentication is reported to be complete.

Authentication for user Bob complete.

The clearance for Bob is 1.

Once logged in ...

A list of allowed actions is now displayed

Options: (C)reate, (R)ead, (W)rite, (L)ist, (S)ave or (E)xit. (*****)

The C option will result in a request for filename and classification, from the client.

Filename: alpha

Security level (0 or 1 or 2): 0

The program should maintain a list of ”files” as internal entries. If the passed file doesn’t exist, it’s name and classification should be added to the list. If the passed file does exist an appropriate message should be displayed and the client should re–display the menu (*****).

The R and W choices each result in a request for a filename.

Filename: alpha

Again a check is made as to whether the file exists. If the file doesn’t exist an appropriate error message should be provided and the menu (*****) should be re–displayed. If the file does exist, a message informing success or failure will be displayed. Success or failure is determined by the relative clearance of the user and the classification of the file they are trying to access. Subsequently the menu (*****) should be re–displayed. The L option lists all files in the FileServer records. The S option saves all the data to a file Files.store. This file should always be loaded if it is available when FileServer starts without the -i argument.

The E option should exit the FileServer, after checking with the user:

Shut down the FileServer? (Y)es or (N)o

If FileServer is to be shut down, it should firstly display a list of files within it’s records.

Note:

i. Password should be hidden when you key-in.
ii. Set rules for password
iii. Set access rights for User clearance (0,1,2) and Security level (0,1,2)- Follow BLP
iv. Introduce CAPTCHA in addition to the basic authentication
v. Ensure passwd and shadow format of Linux is followed
vi. Ensure readme.txt is available for instruction

vii. Assigning Group / Role to a user will be give more credits
viii. Perform necessary validation

Due date: 12/9/2018