Problem Set 3

This problem set is due on *Thursday, October 7, 1999* at the end of class. Late homeworks will *not* be accepted. You are to work on this problem set in groups of three to four people. (Be sure that you all understand the solutions turned in to each problem.)

Mark the top of each sheet with your names, 6.857, the problem set number and question, and the date. **Type up your solutions**[1], and be clear. Points may be deducted if your TA has problems understanding your solution.

Problem 3-1. Take/Grant groups

Suppose you desire a computer security system in which you can define *groups* of subjects, and grant everyone in the group the ability to read certain files, write certain files, or perform other operations. You need to be to able to create a group, add a new subject to the group, remove a subject from the group, and add or remove rights from the group.

Explain, as best you can, how to model such a scheme using the "take/grant" model. If you find the take/grant model inadequate in any respect, sketch an extended take/grant model that can accommodate your concerns. Try to think through all the possibilities and implications. (For example, what about groups having other groups as members?)

Be concise, but complete; two pages max should suffice.

Problem 3-2. Access-Matrix Versus Stack-Inspection

The Java stack-inspection model determines whether to honor a request for a protected object by examining the method-call stack from top to bottom. Once a method is found that has explicit permission or explicit denial of permission to access that object, that permission or denial is operative.

Briefly compare the access-matrix model studied in class with this stack-inspection model. Do you feel that one model is more expressive than another, or are they just different? Try to give example scenarios that illustrate what each scheme can do that the other can't. Explicitly discuss the issue of what a method call might look like in an access-matrix model.

Problem 3-3. Tripwire in the Matrix

Alice wants to implement an integrity verification system within the framework of the Graham/Denning security model presented in class. She creates a subject S_{sm} which she designates as the 'security monitor'. For each subject S_i in the system, the security monitor creates an H_i which is a hash ("snapshot") via some public hash function of S_i.

(a) Argue whether the hashes need to be separate objects, or whether they can be 'bundled' with either their respective S_i or with S_{sm} itself. What rights does S_i need over H_i? If necessary, modify the rules in Table 1 of the paper to implement Alice's idea.

[1] If you latex your solutions, include a URL or AFS path where I can find the latex file(s); I will distribute my favorite solution for each problem to the class as the 'official' solution – this is your chance to become famous! Note that if I choose your solution, then you get an extra point on that problem.

(b) How does S_{sm} maintain a secure version of H_{sm}? Can you add rules to make it more difficult for a compromised S_{sm} to modify its H_{sm} and thus hide the compromise? Would using multiple security monitors be more helpful? Are there other solutions that would be effective?

(c) Taking a hash of a data object is easy, but a given subject might consist of the program (eg executable) plus state (register, memory, etc). How does this affect Alice's idea? Can you change the scheme (maintaining equivalent security) so the hash is formed from some subset of each subject? What keeps track of which subset of each subject should be hashed? Does this introduce new vulnerabilities?