Solutions, exercises, set 2

Computer Security

Exercise 2.1
a) We store one access control list (ACL) with each object:

ACL for x:	Alice: read, write; Bob: read;
ACL for y:	Alice: read; Bob: read, write;
ACL for z:	Alice: execute;

b) Each user is associated with their own capability list:

Alice's capability:	x: read, write; y: read; z: execute;
Bob's capability:	x: read; y: read, write;

c) With each of the two types of lists, one operation is "easy" and the other is "cumbersome". E.g., removing all of Alice's access rights means in one case scanning through all ACL's for any entry of hers, and in the other just removing her capability list.

Exercise 2.2
a) There are 26 letters so in all, we have $(2 \cdot 26 + 10) = 62$ characters to choose from and we can create 62^6 different passwords. On average, we will need to try half of that and with ten checks per second, we will need

$$\frac{1}{2} \cdot \frac{1}{10} \cdot 62^6 \approx 2.8 \cdot 10^9 \text{ seconds} \approx 124 \text{ years}$$

which is a lot.
b) The second factor in the product above becomes 10^{-6} yielding just below eight hours. This could be doable.
c) The third factor in our previous calculations becomes 62^8 so we can multiply our old answers with $62^2 \approx 4000$. Thus, with six characters we need almost half a million years and in the case with eight characters we need about three and a half year.

Exercise 2.3
We have the set of classifications

$$H = \{\text{public}, \text{confidential}, \text{strictly confidential}\}$$

where

$$\text{public} \leq_H \text{confidential} \leq_H \text{strictly confidential}.$$

We form the set of categories

$$C = \{\text{ADMIN}, \text{LECTURERS}, \text{STUDENTS}\}.$$

Now, by definition, $(h, c) \leq (\text{confidential}, \{\text{STUDENTS}\})$ if and only if $h \leq_H \text{confidential}$ and $c \subseteq \{\text{STUDENTS}\}$. This in turn is equivalent to $h \in \{\text{public}, \text{confidential}\}$ and $c \in \{\{\text{STUDENTS}\}, \emptyset\}$. The various combinations of these give four cases, all in all.

The second question is readily answered by realizing that n different security levels mean we can choose h in n ways and that a subset c of m different categories can be formed in 2^m different ways (for each category, we either include it or we don't; we construct a bit vector of length m). We can thus choose (h, c) in $n2^m$ different ways. $(n, m) = (16, 64)$ gives $n2^m = 16 \cdot 2^{64} = 2^{68} \approx 3 \cdot 10^{20}$.

Exercise 2.4
a) We note that this exercise is heavily related to the previous one where the major difference is that we reverse a definition. For example, a subject with label {STUDENTS} can (only) access objects with one of the labels

- {STUDENTS},
- {ADMIN, STUDENTS},
- {ADMIN, LECTURERS, STUDENTS} and
- {LECTURERS, STUDENTS}.

b) Similarily, a subject with label {ADMIN, STUDENTS} can (only) access objects with one of the labels

- {ADMIN, STUDENTS} and
- {ADMIN, LECTURERS, STUDENTS}.

In other words, there are more restrictions on an object in order to be accessible.
c) A subject with label $\{\emptyset\}$ can access all objects. An object with label $\{\emptyset\}$ can only be accessed by objects with label $\{\emptyset\}$. Thus, $\{\emptyset\}$ would be the level called *system high*. For the label {ADMIN, LECTURERS, STUDENTS} we have the opposite situation. This level is called *system low*.

Exercise 2.5
a) Since other users than root will need to be able to read `/etc/passwd`, the sensitive information normally found in that file (i. e., the hashed passwords) is moved into a file readable by root only. That file is `/etc/shadow`.
b) A salt is a value added to the password before it is hashed. It serves three purposes. First, an attack that aims to recover all passwords in a list needs to test each password with each salt. Second, if two users have the same password, they will have a different hash provided that they have different salts. Third, it protects against time-memory tradeoff attacks (or rainbow attacks).

Exercise 2.6 The error is the conclusion made in the last sentence. It is indeed correct that the rainbow table needs approximately a factor t fewer table lookups. However, this does not mean that it is a factor t faster. Instead, this attack will require more computations before a table lookup can be made. In the first try we apply the reduciton function R_{t-1} before searching the endpoints. In the next try we have to apply R_{t-2}, h, R_{t-1}, the next time $R_{t-3}, h, R_{t-2}, h, R_{t-1}$

and so on. For the Hellman case, we only need to apply h followed by R to get a new candidate endpoint.

Exercise 2.7 The salt is used as an input to the hash function. When the tables are created the reduction function needs to map the hash value to a new input. If the salt is a long random number number it will not be feasible to cover all possible values in the table. In this case, the salt will be seen as a part of the password even though it is considered known in the realtime phase. If the salt is known during the preprocessing phase, the tables can be created according to the known salt. However, if the salt is unique for different users, the tables can not be used to break other accounts. Since the precomputation time is asymptotically the same as a brute force, it would be just as easy to just do a brute force on the hash value with known salt.

Exercise 2.8 The effective user ID for the program will be the same as the owner of the program, i.e., the program will run with root privileges. In this case, the setuid bit is not useful at all since only the root can run the program anyway.

Exercise 2.9 For files, group will not be allowed to write and others will not be allowed to read or write. The permissions will be given by 640. For programs, group will not be allowed to write and others will not be allowed to read, write or execute. The permissions will be given by 750.

Exercise 2.10 In Unix Alice will be allowed to read the file but not write to it. Members of the group students will be allowed to write to the file but not read it. In Windows Alice will be allowed to both read and write to the file and other members of the group Students will only be given write access.

Exercise 2.11 In Unix/Linux each process receives the UID of the user and the GID of the user's group. Both real IDs and effective IDs are inherited by the process. The effective IDs are used to determine if the process have access to objects or not. In Windows, each user has an access token which is created when the user logs in. When a process is started a copy of the access token is created and tied to the process. The access token contains the SID of the user and the SIDs of the groups the user is a member of. In addition, a token has a list of privileges which can be used to gain access to tasks.