Exercises with solutions, Set 2

EITF55 Security, 2021

Dept. of Electrical and Information Technology, Lund University, Sweden

Instructions

These exercises are for self-assessment so you can check your knowledge during the course. You do not need to submit the answers.

- These exercises are based on material in Chapter 3-4, 12, 26 (ACL,ACE) and course slides.
- Some exercises may require you to look for information outside the textbook as well.
- Exam questions can be based on some, part of, or none of the exercises.
- The teacher(s) is/are happy to help you out if you run into problems.

Exercise 2.1

Alice can read and write to the file x, read the file y and execute the file z. Bob can read x, read and write to y and has no access to z.

a) Write access control lists for this situation.

b) Write capability lists for this situation.

c) What is the difference between access control lists and capability lists in terms of revoking all access rights to a specific file and revoking all access rights for a specific person?

Exercise 2.2

Assuming that passwords have length 6 and all the (English) alphanumerical characters, upper and lower case, can be used in their construction. How long will a brute force attack take on average if

a) it takes one tenth of a second to check a password?

b) You use a GPU engine. Using a modern graphics card GPU engine one can implement a rather effective password search engine (see G Tech Case Study). Suppose your search engine can make 500 million passwords test per second.

c) Compare your answers to the case when the passwords have length 8.

Exercise 2.3

a) In Unix, why are shadow password files used and how do they differ from the normal password files?

b) What is a salt and why is it used when a password is encrypted?

Exercise 2.4

Consider a hash function h which is known to have good cryptographic properties and whose output word length (that is the size of the hash) is 64 bits.

Compute the length k of two (equally long) lists with random character strings/words of length 256 so that the probability that these two lists have at least a word in common whose hashes are equal is 0.5. Check by using the Birthday Paradox if the number of collisions inside the same list (that is words belonging to the same list that have identical hash values) can be ignored (that is the number of collisions in the list is small compared to k). State your answer as a power of 2.

Hint: Show first that for each value we compute in the second table, we have a probability $(N-k)/N$ that it does not coincide with any value in the first table. So, the probability $P(k)$ that NO value in the second table coincides with any value in the first table is $[(N-k)/N]^k$, where $N = 2^{64}$ is the number of possible different hash values. You will have use for the following approximation $\ln(1-k/N) \approx -k/N$ if $k << N$.

You may use the following notation when answering: K(ilo) words $= 2^{10}$ words, M(ega) words $= 2^{20}$ words, G(iga) words $= 2^{30}$ words.

Exercise 2.5

Explain why a random salt protects against the time-memory tradeoff (or rainbow) attack.

Exercise 2.6

Explain the access permissions given to the program `run` when it is exe- cuted. To what extent are the permissions useful?

```
-rwsr--r-- 1 root admins 504 2010-02-01 05:43 run
```

Exercise 2.7

In a Unix system what effect does the command `umask 027` has on newly created files and programs?

Exercise 2.8

Assume that the file `Bill.txt` is owned by user Alice and the group Students. The file permissions in Unix give the owner read access and the group write access (420). In Windows there is one ACE giving Alice read access and one ACE giving the group Students write access. How is this seemingly similar situation handled in Unix and Windows respectively.

ACE=An access control list (ACL) is a list of access control entries (ACE). Each ACE in an ACL identifies a trustee and specifies the access rights.

Exercise 2.9

When a user starts a process, the permissions of the user have to be transferred to the process which will be acting on behalf of the user. How is this implemented in Unix/Linux and Windows respectively?

SOLUTIONS EXERCISE SET 2

Solution 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.

Solution 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

$$\frac{1}{2} \cdot \frac{1}{10} \cdot 62^6 \approx 2.8 \cdot 10^9 \text{seconds} \approx 90 \text{ years}$$

which is a lot.

b) The second factor in the product above becomes $2 \cdot 10^{-9}$ yielding about 57 seconds. This is fine.

c) The third factor in our previous calculations becomes 62^8 so we can multiply our old answers with $62^2 = 3844 \approx 4000$. Thus, in case with eight characters we get for case a) a bit above a quarter of a million years and in case b) when using the GPU, a bit more than 2.5 days. The latter may still be fast enough.

Solution Exercise 2.3

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).

Solution Exercise 2.4

We assume that k is small enough so we do not get collisions in the list itself. We later will check if this assumption is ok. Now if the first list has k different elements then the probability that an element in the second list does not occur in the first list must be given by the fraction $(N-k)/N$ of N elements that differ from the k elements in the first list. Thus the probability that this happens for k elements in the second list is

$$\left(\frac{N-k}{N}\right)^k$$

Thus we want to find k such that $\left(\frac{N-k}{N}\right)^k = 0.5$. Taking natural logarithms at both sides of the equation gives $k\ln(1-k/N) = \ln(1/2) = -ln2$. Using the approximation we conclude that $k^2/N = \ln 2$ and thus $k = \sqrt{N\ln 2} = 0.83 \cdot \sqrt{N} = 0.83 \cdot 2^{32} = 3,3$ G words. We observe that for this value of k the Birthday paradox predicts the occurrence of collisions within the first list. On the other hand the value of k is not dramatically higher than the threshold where we expect collisions to appear. Hence we might still assume that of list of k elements only contains relatively few collisions and we still ignore this effect.

Solution Exercise 2.5

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 pre-computation 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.

Solution Exercise 2.6

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.

Solution Exercise 2.7

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.

Solution Exercise 2.8

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.

Solution Exercise 2.9

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.