Computer Security and Privacy (COM-301)

Discretionary Access Control

Interactive exercise solving

Least Privilege and Access control

Answer template on the next slide

Access control policies should be implemented in such a way that subjects are never "overprivileged". In other words, subjects should have the minimal access to an object in order to perform a task.

Imagine a simple permission system where one can have the following permissions:

r: read the content of an object

w: write to an object

x: execute an object

Imagine the system has two directories submission and grading.

How would you assign permissions from principals to objects implementing least privilege to:

1- Students that need to submit their report to the directory submission

2 - TAs that need to grade reports and write the result on a file grades in directory grading

3 – Professor that needs to execute a script averaging in folder grading that uses the results in the file grades in directory grading

Least Privilege and Access control

Your solution should be of the form

Principal	Object	Permission
Student		
TA		
Professor		

Think adversarially to decide on least principles.

Remember there is not only one correct solution, it depends on your threat model.

3

Principals:

Student (consider N principals of type student)

TAs
Professor

Objects:

Student's own assignment (submission/own.txt)
Others student's assignment (submission/others.txt)
Grades file (grades/grades.csv)
Grading script (grades/script.sh)
Directories themselves

ACCESS NEEDS:

Students: need to write their own assignment, but not other students' assignments. According to the question, there is no need for the students to read. They only need to submit. [If you would justify that they need to read, e.g., to check the assignment

was uploaded correctly, it would be ok. As long as you state the need and provide the correct permission configuration]

TAs: need to be able to read the assignments to grade them, and they need to be able to write the grade in the grades.csv file. [As before, they do not necessarily need to be able to read grades.csv, but it is possible to justify that need]

Prof: needs to be able to read the grades and execute the script. [One could also justify need to write the script (to update) or read the assignments (to check the grades) and update the grades in grade.csv]

PERMISSIONS:

Student – Submission/their_own_assignment.txt : w
TA – Submission/ : r
TA – Grading /grades.csv : w
Prof – Grading/script: e
Prof – Grading/grades.csv: r

Building a messenger

Alice and Bob want to build a messenger. When Bob wants to send a message to Alice, he will run the executable `msg`, which will write to `msgfile`. Alice can later read the messages from `msgfile`.

Consider these two propositions:

```
            -rwx--x---  Alice Alice+Bob msg
(1)         -rwx------  Alice Alice+Bob msgfile

            -rwx--x---  Alice Alice+Bob msg
(2)         -rwx-w----  Alice Alice+Bob msgfile
```

What is wrong with each proposition?

Propose a solution.

Proposition 1: Bob cannot write in Alice's file!
Proposition 2: Bob can overwrite Alice's messages!

Solution: "setuid" mechanism:
The solution is to let Bob execute `msg` as if he was Alice. For this we can set the **suid** bit **s**

```
-rws--x---          Alice Alice+Bob msg
-rwx------          Alice Alice+Bob msgfile
```

When the suid bit is set, when the program is executed it runs with the permissions of the owner. When the program ends, the permissions are returned to normal.

Now, when Bob executes `msg`, it will run with Alice's permission. Thus, it can open `msgfile` and write the message. When the message end, Bob cannot write on the file so he cannot overwrite Alice's messages.

Like with any other program, it is very hard to know if the program is doing only what it is meant to do. Thus, setting suid on root programs is very dangerous, as they run on the TCB! If something goes wrong there are no more protections