

ITEE



What's New:

You should aim to catch up on previous tutorials; this is the last tutorial, so there will be no further new material; the last week of lectures will include practice questions leading to the exam.

Learning objectives for this week:

- Understand the major concepts we've covered, including:
 - different types of protection
 - different aspects of the security problem
- Apply these concepts to reasoning about protection and security
- Apply these concepts to evaluating alternatives
- Understand issues in improving security

I've left a few minor gaps as exercises but the answers are reasonably complete. As before, try to derive new questions from these.

1. Concepts

- a. When you use a UNIX file system, to what extent can you control access to files?
You can control access at the level of owner, group and others. Users can be in more than one group, which adds some flexibility.
- b. How does an access control list relate to or differ from a capability?
An ACL is a list of who has what permission for a specific file (or other resource) whereas a capability is the rights of a specific user (or other access domain).
- c. What is the difference between symmetrical and asymmetric encryption?
Symmetrical encryption requires the same key on both sides.
- d. What are the advantages of public-key encryption?
Public-key encryption allows a key to be exchanged over an insecure channel, relying on the fact that the users on either side have incorporated their own private secret (only known to them, not even to the other party in the communication) into the key. If the secret known only to one side (and the similar secret only known to the other side) is sufficiently hard to deduce from the key, it is a secure approach when the two parties are not in a position to initiate communication through a secure channel.

2. Protection

- a. Capabilities have been widely explored in distributed systems as a way of transferring rights to remote users. What specific features are important if capabilities are to be used in this way, without creating security holes?
Some examples:
 - *forging should not be possible*
 - *it should not be possible to use a capability after it has expired*
 - *it should be possible to revoke a capability*
 - *a capability should be impossible to use without authenticating the intended user*
- b. Consider each of the following scenarios, and describe how you could use access control lists, capabilities or UNIX file permissions to achieve what you are aiming for:
 - i. your header files and given source files for an assignment should be accessible to the whole class, but your personal implementation should only be visible to you, your tutor (and not others), and the lecturer. You should be able to read and modify your files (including compiling and running your finished code), and the others who have access to them should be able to read them, and run executables, but not modify files. Does it make a difference in any case if the differences in access rights are organized around directories or files?
ACLs or capabilities could work here. Each file's ACL would have to have the appropriate list of users and their permissions. Each user could be given a capability with the required permissions listed for each file. Making this work with UNIX permissions may be possible with creative use of groups. The "private" files should be group-readable, and the group they are in should be one which includes your tutor and lecturer. The owner would have execute, read and write permission, the group read and execute permission and others no permission. A different group would have to be created for every tutor, in a strict interpretation that other tutors should not be able to read your files. Organizing access rights around files would be easier because it wouldn't be necessary to put all files with a given level of access in their own directory.
 - ii. you are doing a group project, and the same conditions as in (i) apply, except other members of the group can read your files, but not modify them. There is a central directory accessible to the whole group where final linking takes place, but compilation of individual components happens in your directory, where others should not be able to compile, so only you can make a compiled version of a component you are responsible for available to the group as a whole.
With ACLs or capabilities, a simple change would work here. For UNIX permissions, you would have to make a UNIX group containing other members of your group, your tutor and lecturer – and you would need 1 such group for each student. Clearly a better solution would be to check files into a common shared directory which all members of your group (including you) could only read. In this variant, only one UNIX group would be needed per project group: still not great but doable.
 - iii. you are the administrator of a distributed system and would like to allow a specific user access to a remote file system.
You could give the user a capability, or add the user to an ACL. UNIX permissions could work, but are difficult to use in an ad hoc way. Depending on how the existing permissions were set up, there would be different ways in which this could be done. It would be difficult to do in general without giving the user permission to do other unintended things.
- c. Draw an access matrix corresponding to scenarios of 2(b)(i-ii). Would it have been easier to answer the question if you did this first?
This is still a good exercise, so I will leave it out for now: let me know if you would like a solution. I'd like to see some attempts before putting up my own.
- d. Why, in general terms, is it hard to handle failures in distributed systems?
This one sneaked in from Tutorial 11 – check there for notes on how to answer it.

3. Security

- a. Remembering passwords is a huge problem. Which of the following could contribute to the solution, or add to the problem? Discuss in each case.

- i. force users to change passwords regularly
This solves problems other than helping users to remember passwords. While solving those problems, regular password changes increase the probability of insecure behaviour (writing a password down, recording in on a computer in unencrypted form).
 - ii. force users to use long passwords which aren't similar to English words
Same answer as (i).
 - iii. allow users to store all their passwords in one place with a master password
This has some attractions but one security hole (a flaw in the way the master password is stored, processed or used) can reveal all the other passwords.
 - iv. a graphical user interface hides passwords behind pictures meaningful to the user, but unlikely to be guessed by anyone else (e.g. a map in which clicking on a location either reveals a correct password or a fake one to put intruders off track)
Provided the number of combinations is high, this could work. But it must be clear to the user, not anyone else, where the secret is hidden.
 - v. allow an access with an incorrect password without any warning of an error, but redirect the access to a secure environment with tripwires
This approach is more suited to catching crackers than helping legitimate users with bad memories.
 - vi. replace passwords by a "guessing game" in which your guesses are compared with previous times you've played the game: exact matches aren't looked for, but rather a pattern of similar thought processes (some AI could apply)
An interesting idea, but it would have to be very clear to anyone with security concerns that the game couldn't be defeated.
- b. Before World War II, France had massive defences on their German border. Germany attacked France through neutral countries, ignoring French defences. Can you think of a security attack on a computer system with similarities to the German approach?
Any example which effectively "walks around" defences would do. For example, remove the disk containing the password file, copy it off on a less secure machine, put the original disk back and run a password cracker on the copy.
- c. In each of the following, classify the security problem as a Trojan horse, a worm, a virus or a denial of service attack. If more than one applies, either select the most applicable, or a combination – try to justify your answer as the best variation:
- i. a program attached to an email message installs itself by exploiting a feature of your mail client's handling of attachments. It scans your address book and sends itself to all the addresses in the list (it is a standalone program, which has its own implementation of the mail protocols)
Since it's standalone, it isn't a virus. This is more like a worm.
 - ii. a program attached to an email message inserts itself into your mail client (exploiting a similar security hole to that in (i)) so that whenever you send a message, it attaches itself to the message you send.
Since this attaches itself to another program, it's a virus.
 - iii. a program like that of (i) repeatedly resends messages at short intervals, and includes a reply-receipt header (on receiving the message, the receiving mail client will send a reply automatically)
This one is also a worm, but one which aims at a denial of service attack by flooding the network.
- d. Describe how public-key encryption could combine with capabilities to provide a secure way of allowing an outsider *limited* access to distributed computer resources. This question should build on 2(a).
You could use encrypted capabilities, with part of the key including a timestamp. Part of the strategy should be to encrypt information about how the capability was set up, and expiry information.
- e. Discussion: why would anyone develop a virus, worm or other malicious software? What should we do if such a person is caught?
I don't know why people do this sort of thing (or vandalise bus shelters, leave trash on a hike, or otherwise choose to be antisocial). A good outcome would be to make the perpetrator pay back the cost to society. Should senders of spam have to refund recipients' wasted time?