DES332: Computer & Network Security
Midterm Summary

Paphana Yiwsiw - @waterthatfrozen

Lecture # 1 — Introduction

- **Computer Security**: the protection afforded to an automated information system in order to attain the applicable objectives of preserving confidentiality / integrity / availability of information system resources. (NIST, 1995)
  - deals w/ the prevention & detection of unauthorized actions of users of computer systems.
  - about access control / the protection of information assets
- **Security Measures**
  - Classification of measure:
    - **Prevention**: measures that prevent your assets from being damaged.
    - **Detection**: measures that detect when/whom/how an asset has been damaged.
    - **Reaction**: measures that recover your assets / from a damage done.

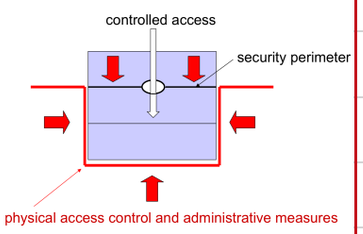
    ↻ the more you invest in prevention, the more you invest in detection (to make sure prevention is working)
  - Security Objectives:
    - **Confidentiality**: Prevention of unauthorized disclosure of information (unauthorized readings)
    - **Integrity**: Prevention of unauthorized modification of information (unauthorized writings).
    - **Availability**: Prevention of unauthorized withholding of information
    - **Authenticity**: "know whom you are talking to"
    - **Accountability**: prove that entity was involved in some event.
- **Confidentiality**
  - **Secrecy**: Protection of an organizational data. (security & secrecy are closely related)
    - Do we want to hide the info. existence? (Traffic analysis / unlinkability / anonymity)
- **Privacy**: Protection of personal data. (the right to be left alone / not bothered by spam)
  - put user in control of their. / more seriously considered by company.
- **Integrity**: "make sure that everything is as it is supposed to be"
  - : is a prerequisite for many security service (e.g. OS)
  - **Data Integrity**: computerized data = source document / not exposed to accidental / alteration / destruction
- **Availability**: accessible & usable upon demand
  - **DoS** (Denial of Service): prevention of authorized access or delaying time-critical operations.
    - **DDoS** (Distributed DoS) : attackers send ICMP requests to broadcast address & spoofed as victim address. then, victim is flooded w/ many incoming messages.
- **Accountability**: proof entity's involvement in some event.; audit logs security-related event & identify associated user.

- Non-Repudiation services: provide unforgettable evidence that specific action occured.
  - of origin: protects against sender (denying sending of data)
  - of delivery: protects against receiver (denying receiving of data)
  - of bitcoin transaction: protects against sender denying that. coins was transferred.
- Fundamental Dilemma: Security-unaware users have specific requirements but no security expertise
  - General Principle of designing a secure IS
    - 1st Design Decision: Should protection mechanism focus on data/operation/users?
      ex. integrity may refer rules on format & content (internal consistency) / ops: to perform/authorize user on data item
    - 2nd Design Decision: Which layers that mechanism should be placed?
      - generic & more controlling access of lower layer; higher layer address about individual requirements.

      IT system layer: [highest] Application / services / OS / kernel / hardware [lowest]
    - 3rd Design Decision: Simplicity & Higher assurance to a feature-rich environment?
      - simple may not match specific requirements / choose right features from a rich menu, be security expert.
    - 4th Design Decision: centralize or decentralize control?
      - single entity is easy to achieve uniformity but can be performance bottleneck.
      - distributed may be more efficient but difficult to manage (guarantee that enforce a consistent policy).
    - 5th Design Decision: Prevent an attacker from access layers below mechanism?
      - attackers will try to bypass protection mechanism.
      - How to stop from getting access to layers below protection mechanism?

      ex: - recovery tool.: restore data by directly read memory & restore file structure.
      to circumvent logic access control, not care for logical memory structure.
      - Unix treats devices (I/O & memory) like files: if access given to disk, it is easy to reconstruct protected files
      - Object reuse: avoid storage residues (data left behind in the memory allocated to new process)
      (mostly in single processor systems).

* w/in same domain of policy, same control should be enforced.



Lecture #2 Authentication

- secure system might track user identities requesting its services.

- log on computer scenario. (w/ username & password)
  - 2 steps - 1: identification : announce who you are
    - 2: authentication: prove who you claim to be.
  - Possible attacks on password guessing.
    - Brute Force/Exhaustive Search : try all different combination ; dictionary attack.
    - Intelligence Search : search through restricted space (popular password / associated w/ user)
      - cannot prevent from accidentally guessing, but can reduce probability of compromising password.
      - counter measure
        - avoid guessable password / aging / limit login attempts / inform user.
- Password file
  - OS maintains file w/ username & password, which attackers could try to compromise the integrity / confidentiality of this file
  - Protection Option : cryptography / OS access control / combination of both.
- Cryptographic function / One-way function
  - often uses one-way function f(x) (easy-to-compute but hard-to-reverse)
  - the value of f(x) is stored instead of password x. Thus, when user enters password x', it compares the value of f(x') with f(x)
    - ↳ encrypted password a.k.a. hash of x
  - dictionary attack can still be used with hashing,
    salt is appended to the password before encryption and stored w/ encrypted password → 2 users w/ same pw. now have diff. hash.
- Access Control Settings
  - only privileged users can have write access to the password file. otherwise, attacker could get access.
  - In theory, if read access given only privileged user, password can stored unencrypted.
  - If password file contains unprivileged users data, password must be encrypted.
    - In UNIX, /etc/passwd is password file, /etc/shadow stored encrypted password file, readable by root only.
      - ↳ entries format: user name : encrypted password : user ID : group ID : ID string : home directory : login shell.

More on authentication

- we can do authentication on basis of what you know / what you hold / who you are / what you do / where you are.

Something you know

- user must know secret to be authenticated: secret is only know to you. e.g. PIN / Password / Personal Info

* a password doesn't authenticate a person * successful auth. means only implies that user knew a secret.
no way of telling difference b/w legit user & intruder.

Something you hold

- user presents a token to be authenticated, but physical token can be lost or stolen.
- who have the token will have same rights as legit owner. Thus, physical token is used with combination w/ sth. you. know.

Who you are

- Biometric schemes use physical characteristics, offers the most secure solution. ex. fingerprint

What you do

- Perform mechanical tasks in a repeatable & specific way. ex. sign on pad. etc.

Where you are.

- Some OS only allow access if log on from a certain terminal; GPS can be used during authen.

Lecture 3 • Access Control

- Process of authorization: Subject → Access Request → Reference Monitor → Object

access request described by (s, o, a): subject s is performing access/operation a on object o [subject is active, object is passive]
↳ process/user ↳ files/devices/ports

Reference Monitor is an abstract machine mediates all access of subjects to objects

- UID in Unix is identity used in authentication

Authorization: RM decides access is granted or denied according to access control policy (what subject is allowed/object can be done?)

- RM find & evaluate rules relevant for the given request.

Access Operation

- some basic policies can be expresed w/ basic access mode.

- **Access Mode**: observe & alter define most elementary level operations.
  - observe: look at content
  - alter: change the content.

BLP Access Right

- 4 access rights
  - execute (e)
  - read (r)
  - append (a)
  - write (w)

mode \ rights	execute	append	read	write
observe			X	X
alter		X		X

- In multi-user OS, it can avoid conflicts when multiple users writing into the same file simutaneously. (when open, file is to read or to write)

Multics Access Attributes

- distinguish between access attributes for files (data segment) and for directories (links to files)

model \ attr.	files				directories			
multics	r	e	r,w	w	status	status & modify	append	search
BLP	r	e,r	w	a	r	w	a	e

UNIX/Linux Access Rights

- 3 access rights on files (r, w, e) & 3 on directories (r: ls, w: mkdir or mv, e: cd)
- can map to access mode as follows: r → observe, w → observe & alter, e → observe.

RDB Access Operation

- Relational Database; access ops. include SELECT, UPDATE, DELETE, INSERT
- DBMS = reference monitor (decide to permit requested operation of user)
- privilege: (grantor, grantee, object, operation, grantable)

Access Control Structure.

- consists of 3 components: S (subject set) / O (object set) / A (access ops. set).
- Access Rights are defined by access control matrix. M.

$$M = (M_{so})_{s \in S, o \in O} \quad \text{with } M_{so} \in A$$

(access of object o for subject s is within access rights A)

Capability and Access Control Lists

- M is normally huge, so we want to store it as separated rows or column.

- **Capabilities**: rows in access control matrix (show what user capable to do) / kept & associated w/ subject
  - used in database application for fine-grained access / useful in distributed system
  - hard to see who has access to given object in overview. / hard to revoke subject access.
- **Access Control List**: column in access control matrix (show what access user have on that object) / associated w/ object.
  - used in OS, when request is made, kernel search ACL of O and search for S, and look for access right.

Intermediate Level

- Subject are grouped together to simplify policy definition. ex. in Unix has 3 groups: owner, group, others.
- **Protection rings**: assign security level to subject & objects, so that lower level cannot access to higher level.
- **Role-based access control**: "Inheritance", higher level can do more things than lower level roles

Lattice of security

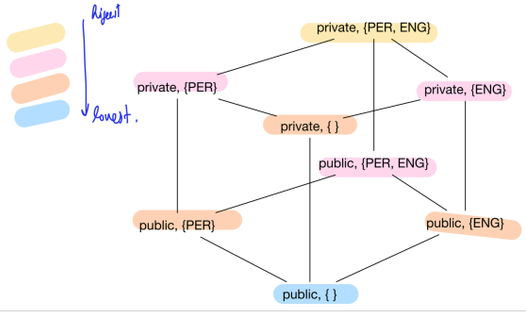
: each object & subject is associated w/ security level a.k.a. lattice.

Typical Policy

- **No-read-up**: S can read O of less or equal security level : $f_O(o) \leq f_S(s)$
  - ensure information flow from low to high (S reads O, info. flow from O to S)
- **No-write-down**: S can write O of higher or equal security level : $f_S(s) \leq f_O(o)$
  - ensure info flow from low to high (S writes O, info. flow from S to O) / prevent abuse of power.

Lattice

- $H = \{public, private\}$
- $C = \{PER, ENG\}$ (personnel and engineering)



*usually in this class

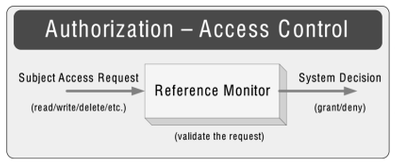
H is set of classifications w/ linear ordering

C is set of categories

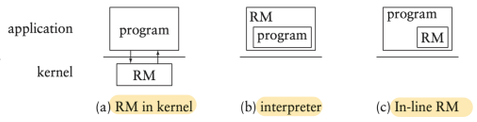
- security level is pair of (h, c) where $h \in H$ and $c \subseteq C$
- partial ordering is $(h_1, c_1) \leq (h_2, c_2)$ if $h_1 \leq_H h_2$ and $c_1 \subseteq c_2$

* partial ordering used to tell which level dominates which.

Lecture 4 • References

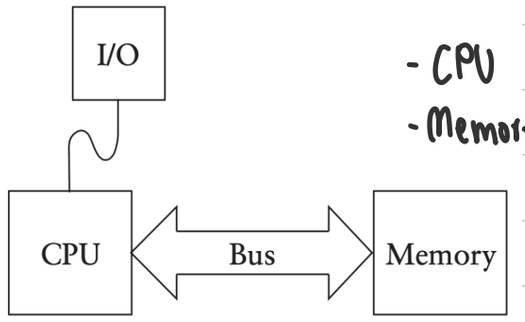


- RM is abstract concept. / kernel (security) = firmware + TCB that implement the RM
  - ↳ Trusted Computer Base: totality of protection mechanism
- RM can place anywhere in layered systems.



- Hardware : access control in microprocessors
- OS : security kernel in OS
- Service : RMs for DBs / JVM etc.
- Application : Security checks for codes.

• Overview of Computer Architecture

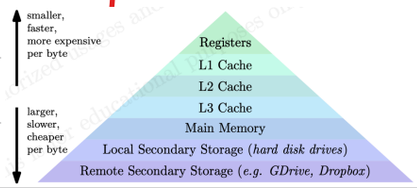


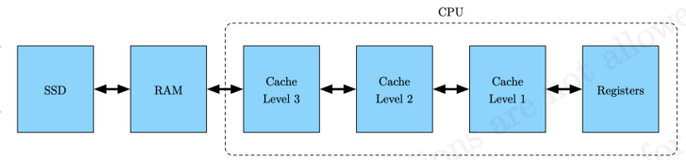
- CPU - I/O
- Memory - Bus

Core CPU Components

- Registers : both general purpose & dedicated (program counter / stack pointer / status reg.)
- ALU : executes instructions given in a machine language

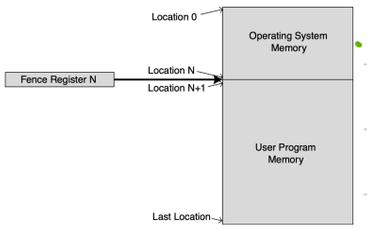
Memory Structure



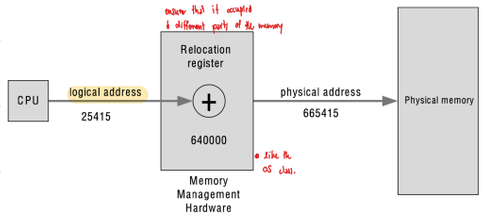


• Memory Protection

- OS has control access to objects; access control to data object rely on access control to memory segments (aka. memory management mechanism)
- Memory Management Mechanism built into hardware include



- Fence : for Single-user OS: to prevent user program to modify OS code.
  - -fence register is used to contain the address of the end of OS code. The memory access request will compare with fence.
- Relocation
- Base & Bound
- Segmentation
- Paging



→ relocation register ensure that it is occupied the different part of main memory, not where OS locates.

• Processes & Threads

- ↳ Processes : program in execution ; Logical Separation of processes as a security basis.
- ↳ Threads : strands of execution w/in a process.

Interrupts

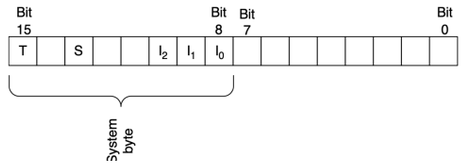
generic terms.

- CPU deals w/ interrupts created by errors etc. through execptions / interrupts / traps.
- **Trap**: is a special input to CPU; includes addr in interrupt vector table giving location of the program to deal w/ specified cond.
  - when trap occurs, CPU saves its current state, then executes the interrupt handler.
    Later, interrupt handler has to restore CPU to proper state before return control to the user.
- A further interrupt may arrive while CPU deals with current interrupt; CPU may interrupt current one.
  * Improper handling of interrupt can cause **security failure.**

Processor's status flag

- support for access control at ML based processor's status flag
- now CPU has **processor status register**, controls how the remaining CPU hardware works.

ex. 16-bit Motorola 68000 : 16-bits for security controls occupy the top section



13 : superior bit (0 = user, 1 = superior) → only interrupt & exception can switch S=0 to S=1
10-8 : Current interrupt level ($I_2I_1I_0$) → the former is interrupted if $I_2I_1I_0$ < latter level

Memory Protection

- OS controls access to data objects in memory, represented by collection of bits in certain memory location.
- Access to logical object is translated to **access operation at machine language level**

3 options for controlling access to memory

- OS **Modify** address from user process (so it can't directly access)
- OS **construct** effective address from relative address it
- OS **check** address received whether it is in **given bound** or not.

- OS manage access to data & resources;
  - 2 Competing Requirements : User can **invoke** OS but not **misuse** it
  - 2 commonly important tools:
    - **Modes of operation** : User mode (not-critical for security) / Supervisor (privileged); Status flag to record current mode.
    - **Controlled Invocation**: OS need to toggle b/w modes. System should perform **predefined set of operation** in supervisor then return back to user mode (normally, changing flag would give sudo priv. to user w/o control)

Lecture 5 • Operating System

- Unix/Linux supports DAC (Discretionary Access Control) policy : owner of files is freely to give permission to any user.
  - /etc/passwd is password file , /etc/shadow stored encrypted password file, readable by root only.
    - entries format: user name : encrypted password : user ID : group ID : ID string : home directory : login shell.
    - password is encrypted using DES (earlier), MD5 (later version)

• Users & Superuser

- each user is represented by UID : root UID = 0, daemon = 1, bin = 2
- superuser is special user (normally = root), superuser can create & delete user account; can assume role of anyone using su.

• Group

- each user can be associated w/ one or more group; GID = group ID

/etc/group contains all group. → entries format: groupname : password : GID : list of users.

- principal: is either UID or GID /- put user in group is easier to manage access control decisions.
- id command: can be used to display the UID & list of GID and its group name (if user is belong to any group).

• Subjects

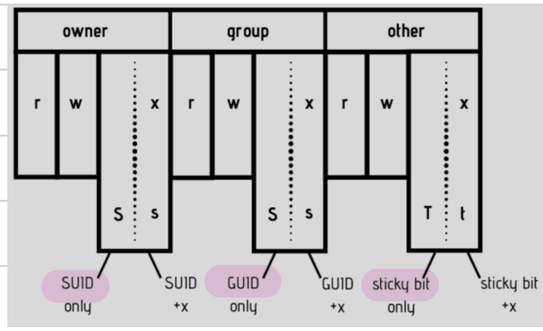
- running program aka. processes. (each has PID); after a successful login, shell forked to authenticated user
- each process is associated by several principals.
  - Real UID/GID : user who created/executed the process.
  - Effective UID/GID : if process is SUID program : effective UID = program owner's UID. If not, effective UID = user UID.

• Objects

- treat all resources in uniform manner., each file entry in the directory is a pointer to data structure called inode.
- inode fields: mode / uid / gid / time / mtime / block count
- Access rights: represents in character / decimal

chmod s g t r w x r w x r w x

- r w x → owner
- r w x → group
- r w x → public
- t → sticky bit
- g → set GID: if g=1 => SGID program.
- s → set UID: if s=1 => SUID program.



permission

0765 : rwxrw-r-x

4765 : rwsrw-r-x

6765 : rwsrwS r-x

Changing Permissions

- chmod : altering access right command
- chown : altering ownership command
- chgrp : altering group owner command (change group)

Access Control

- when a process accesses a file, access control uses the effective UID/GID of a process.
  It allows user A runs programs own by user B and can use user B privileges during execution.

Order of Checking

- when subject/process requests an access to a file,
  - if subject is not SUID program → if UID = owner ; use owner permission
    → if GID = owner's group ; use group permission
    → otherwise, use public/others permission
  - if subject is SUID/SGID program. → use as same as above but use EUID instead.
    * use effective UID(EUID)
    - use setfacl to give permission to specific user (can done by owner only)

Directory permission

- You need execute permission in directory to access your own files.
- w/o read permission, you can open files, but cannot list the contents in directory.
- w/ write & execute permission, you can delete a file (no permission on file is needed.)
- use sticky bit to only allow the owner to delete a file.

SUID/SGID Program

- Unix/Linux provides a SUID & SGID program collection owned by root, such gives temporary or restrict access to files normally not accessible by other user ("controlled invocation)
- Important SUID programs (/bin/passwd, /bin/login, /bin/at, /bin/su)

SUID to root

- login program calls for setuid (login is SUID to root) → run with EUID=0 → when login satisfied, invoke setuid & setegid to the logged in user. → execute shell with real UID & EUID set for the appropriate user.

Applying Control Invocation

→ Integrity of SUID programs must be monitored. & take user input processed w/ extreme care

→ sensitive resources can be protected by combination of ownership, permission bits & SUID programs.

↳ create new UID that owns resources & program, only owner gets access, define all programs to be SUID

* Beware for over-protection: you may need to provide indrect access through SUID programs if you deny direct acces too much.
* flawed SUID program gives more access opportunities than wisely chosen permission bit. (if owner of SUID program is privileged user)

TCP Wrappers

- host-based networking ACL system., used to restrict access to TCP services
- 2 reference files: /etc/host.allow & /etc/host.deny
- if client requested, if found matching rules in host.allow → granted; applied in host.deny & found match → deny; otherwise, granted.

Iptables

- host-based networking ACL on network layer.
- if host is blocked by IP tables → denied access (no matter what TCP wrappers says).

Audit logs & Intrusion Detection

↳ auditing: recording of security events for later analysis.

↳ intrusion detection: detection of suspicious event. & alert system admin

↳ intrusion response: react immediately to security alarm & taking appropriate actions

Paphana Yiwsiw - 6 water that frozen

DES 332 : Homework 5

This question concerns the Bell-LaPadula (BLP) model.

Consider a state $\delta = (b, M, f)$ where $b = \{(Ekawit, f1, r), (Gun, f2, r)\}$; $M_{s,o} = \{r, w, a\}$ for any subject s and object o; f_S and f_O are given by the following table; and $f_C(Ekawit) = (2, \{cpe\})$ and for other subject s, $f_C(s) = f_S(s)$ (i.e. the current level of s coincides with its maximal level).

f_S f_O

Subject	Subject Label	Object	Object Label
Ekawit	(3,{cpe,de})	$f1$	(2,{cpe,de})
Gun	(3,{cpe})	$f2$	(2,{cpe})
Nan	(2,{cpe})	$f3$	(1,{de})
Student	(1,{de})	$f4$	(2,{cpe,de})

where a label (h_2, c_2) dominates another label (h_1, c_1) (i.e. $(h_1, c_1) \leq (h_2, c_2)$ if $h_1 \leq h_2$ and $c_1 \subseteq c_2$ (For example, $(2, \{de\}) \leq (3, \{cpe, de\})$).

From this δ, what accesses (r, w, a, e) should the subjects be allowed to do on the objects?

State $\delta = (b, M, f)$

- $f = (f_S, f_C, f_O)$ * for each s: $f_C(s) \leq f_S(s)$
  - f_S: security level of each object (a.k.a classification of o)
  - f_C: current security level of each subject
  - f_O: maximal security level of each subject (a.k.a. clearance of s)
- M: access control matrix ($M_{s,o}$)
- b: triples of (s, o, a) — subject s is currently performing action a on object o

In this Homework:

$b = \{(Ekawit, f1, r), (Gun, f2, r)\}$

$M_{s,o} = \{r, w, a\}$ for any subject s and object o

$f_S(Ekawit) = (3, \{cpe, de\})$ $f_C(Ekawit) = (2, \{cpe\})$

$f_S(Gun) = (3, \{cpe\})$ $f_C(Gun) = (3, \{cpe\})$

$f_S(Nan) = (2, \{cpe\})$ $f_C(Nan) = (2, \{cpe\})$

$f_S(Student) = (1, \{de\})$ $f_C(Student) = (1, \{de\})$

$f_O(f1) = (2, \{cpe, de\})$

$f_O(f2) = (2, \{cpe\})$

$f_O(f3) = (1, \{de\})$

$f_O(f4) = (2, \{cpe, de\})$

Partial Ordering (Dominance)

$(h_1, c_1) \leq (h_2, c_2)$ if 1) $h_1 \leq h_2$ 2) $c_1 \subseteq c_2$ (subset)

BLP Access Right.

	execute	append	read	write
observe			X	X
alter		X		X

access rights that requires observe access mode = read (r) / write (w)

access rights that requires alter access mode = append (a) / write (w)

Question : ask for $N_{s,o}$ (access that s is allowed to do on o)

Noted that for $N_{s,o}$ the state δ must be secure (according to BLP model)
↳ $= \{ a \in \{r, w, a, e\} \mid (b \cup \{(s,o,a)\}, M, f)$ is secure$\}$

So, what "secure" means?
↳ all 3 security properties are satisfied

① ss-property (no-observe-up)
for each $(s,o,a) \in b$
where a is read(r) or write(w) (a.k.a requires observe mode)
* the security level of subject s dominates object o ($f_o(o) \leq f_s(s)$)
→ ensure that doesn't create info. flow from higher to lower level

② * - property (no-alter-down)
both following conditions must be true.
2.1) for each $(s,o,a) \in b$
where a is append(a) or write(w) (a.k.a. requires alter mode)
* the current security level of subject S is lower than security level of object O
($f_c(s) \leq f_o(o)$)
2.2) if there exists element $(s,o,a) \in b$ where a is append(a) or write(w) * same subject performing many a on multiple O
then for any object o' with $(s,o',a') \in b$ and a' is read(r) or write(w)
(s can transfer information to o, and can copy information from o')
* we must have $f_o(o') \leq f_o(o)$
→ ensure that possible info. flow from o' to o (via s) is low to high

③ ds-property
state (b, M, f) satisfies the ds-property
if for each element $(s,o,a) \in b$
we have $a \in M_{s,o}$
(currently performing action must be in the access control matrix)

Paphana Yiwsiw - @waterthatfrozen

DES332 : Homework 6 Explaination

Settings: *Chinese wall model*

- subject S has read fK1 and fD2 (in the past)

4 Companies: Bangkok Bank (B), Kasikorn (K), AIS (A), DTAC (D) — Bank, Telecom class.

Each Company has 2 files
1: cannot access by company in same class.
2: public files

Thus,

Owner : y	Restricted Company : x
y(fK1) = Kasikorn	x(fK1) = {Bangkok Bank}
y(fK2) = Kasikorn	x(fK2) = ∅
y(fB1) = Bangkok Bank	x(fB1) = {Kasikorn}
y(fB2) = Bangkok Bank	x(fB2) = ∅
y(fA1) = AIS	x(fA1) = {DTAC}
y(fA2) = AIS	x(fA2) = ∅
y(fD1) = DTAC	x(fD1) = {AIS}
y(fD2) = DTAC	x(fD2) = ∅

$N_{S,fK1}$ = true $N_{S,fD2}$ = true , anything else = false

Questions: S do the task: (read/write) to (files)
(check ss-, weak *-, Strong *- property)

- C : set of companies (note: 2^C = power set)
- O : set of objects
- y : (O → C) owner's document
- x : (O → 2^C) set of companies that are not allowed to learn about it.
- security label of object o is (x(o) , y(o))
- $N = (N_{so})_{s \in S, o \in O}$ that N_{so} = true iff s has access to o (in the past)

- **ss-property** (read/write)
  subject s is allowed to observe object o
  iff for all object o' with $N_{s,o'}$ = true (s has accessed o' sometime in the past)
  then ① y(o) ∉ x(o') (not in restrict list) or ② y(o) = y(o') (same owner)
- **weak *-property**
  subject s is allowed to write object o
  if for all object o' read by s (sometimes in the past before)
  then ① y(o) = y(o') (same owner) or ② x(o') = ∅ (o' is a public file)
- **Strong *-property**
  subject s is allowed to write object o
  if for all object o' read by s (sometimes in the past before)
  then
  ① y(o) = y(o') and (if x(o') ≠ ∅ then x(o) ≠ ∅) (same owner and if o' is sensitive, o is sensitive too,
  or
  ② x(o') = ∅ (o' is public file)

  ↳ this guarantee one doesn't write public obj. after read sensitive obj. o'