Computer system security

Access control (cont'd)

6 October 2016

Formalizing access control

Recap: access control =
a *mechanism* to allow or deny an entity's access to a resource

We distinguish:
– a set of subjects or principals S
– a set of objects O
– a set of access modes A.

Simplest: $A = \{\mathit{observe}, \mathit{alter}\}$. Usually not enough.

The Bell-LaPadula model refines this to:
$A = \{\mathit{execute}, \mathit{read}, \mathit{append}, \mathit{write}\}$.

Access Control Matrix

The simplest and most general organization of access control

Two dimensions: subjects and objects
– every matrix entry $S \times O$: set of rights/permissions
– a subject may also be an object (e.g. a process):
has the right to read/write(to)/execute another process

	`file1`	`/etc/passwd`	`/bin/rlogin`
Alice	`r, w`	`r`	`x`
Bob	`r`	`-`	`r, x`

Mechanisms: Access Control Lists (ACL)

A representation of the access control matrix where *each object* has associated a list of subjects and their permissions

simple case: Unix permissions

Richer set of permissions: Andrew File System (distributed)

read

list (for directories: content)

insert (new file in directory)

delete (file from directory)

write

loc**k** (may use *lock* in directory)

administer

Rights and their propagation

copy right (grant right): the right to give others permission

own right (admin): the right to give oneself permissions

Principle of attenuation of privilege:
A subject may not give rights it does not possess to another

Q: In Unix, the owner of a file may grant others (group/others) read rights on the file, even if (s)he does not have these rights. Is the above principle violated ?

Fundamental results

Is it possible to design a correct access control system ?

Def: A system is *safe* with respect to a given right (of a subject over an object), if there is no sequence of transitions (operations) by which the right could be added, assuming it does not exist at first.

Fundamental results

Is it possible to design a correct access control system ?

Def: A system is *safe* with respect to a given right (of a subject over an object), if there is no sequence of transitions (operations) by which the right could be added, assuming it does not exist at first.

Theorem: The safety of an arbitrary system in a state, relative to a given access right is *undecidable*.
Proof: a Turing machine can be reduced to (encoded into) such a system
essentially because of the ability to create objects

There are simpler subclasses of systems that are decidable
– if only there are no *create* primitives
– if the systems are *monotonic* (if create, no destroy) and only single conditions are allowed

Types of access control

Discretionary Access Control (DAC)
allows *individual users* (typically: owner) to set mechanisms by which access is granted/forbidden

Mandatory Access Control (MAC)
access is controlled by the system, cannot be changed by the user
usually: based on a set of rules *rule-based access control*

Q: What are advantages and disadvantages of each category?

Role-Based Access Control (RBAC)

system-determined policy, depending on the active *role* of a *subject*

3 (or more) levels: subject $\rightarrow$ *role* $\rightarrow$ object

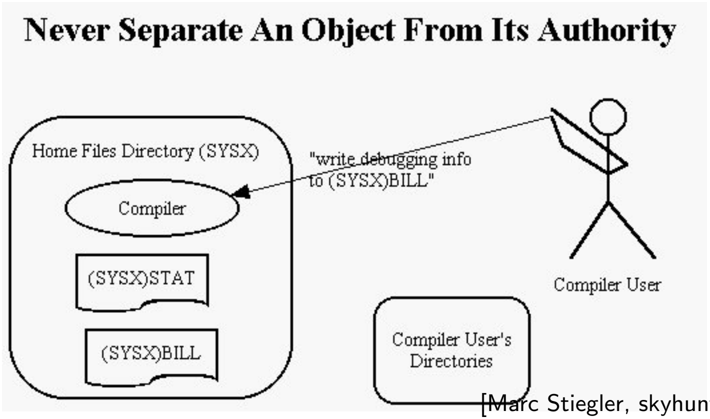
Permissions are defined depending on the role
$\Rightarrow$ a subject may have access when acting in one role, not in another
role *hierarchy* (some may be included in others)
attending physician $\subseteq$ physician $\subseteq$ medical personnel

can model various requirements, e.g. *separation of duty*
ex. a bank loan must be approved by two different bank officers

Security policies must be carefully specified and *checked*
$\Rightarrow$ policy description languages

A classic problem: Confused Deputy

Norman Hardy, The Confused Deputy(or why capabilities might have been invented), ACM SIGOPS Operating Systems Review, 22(4), 1988


Never Separate An Object From Its Authority
Home Files Directory (SYSX)
Compiler
(SYSX)STAT
(SYSX)BILL
"write debugging info to (SYSX)BILL"
Compiler User
Compiler User's Directories

[Marc Stiegler, skyhunter.com]

The Confused Deputy

Who is to blame?

– The code to deposit the debugging output in the file named by the user?
– Must the compiler check to see if the output file name is in another directory?
– Should the compiler check for directory name SYSX?
– Should the compiler check for the name (SYSX)BILL?

Mechanisms: Capabilities

Term with slightly different meanings: in *operating systems*, an *identifier* that denotes an object *and* the rights associated with it
ex. file descriptor/handle (on open, the access mode is also set)

in *security*, a capability is a *list of rights* of a subject
(corresponds to a row in the access control matrix)

E.g.: POSIX/Linux Capabilities `capability.h`
for a new process: `new = forced | (allowed & inheritable)`
Examples: `CAP_CHOWN`, `CAP_KILL`, `CAP_SETUID`, `CAP_SETPCAP`

Multilevel Security (MLS)

Inspired from military domain. Defines *security levels*
e.g. public $\leq$ restricted $\leq$ confidential $\leq$ top secret
also: compartmentalized by domain of interest (*need-to-know basis*)

The set of security attributes is ordered in a *lattice*

Bell-La Padula Model: enforces *confidentiality*. 2 rules:
no read up: a subject may not read above its security clearance level
no write down: may not write (disclose) something under its own level

Biba Model: follows *integrity*. Has dual rules:
prohibits writing above one's own level and reading (using) data found underneath this level: both may corrupt integrity

To attain both, in practice, a subject may voluntarily drop his/her privilege level

Non-interference

a security property for multi-level security systems

In a system with two classes of input/output (observable actions)
high (confidential) și *low* (non-confidential)
it should not be possible to derive something about *high* level actions by observing *low* level actions

If the property is not satisfied, we have an unauthorized information flow (covert channel)

Important to analyze (from hardware to programming languages)

Integrity: from theory to practice

Security principles in developing commercial software (Lipner)

– *users* will not write their own programs, but will use existing production software

– programmers will develop and test on a separate system (not the production system)

– installing a program must constitute a special process

– this process is subject to *control* and *audit*

– managers/auditors will have access to system state and logs

Principles:

– separation of duty

– separation of function

– audit/accountability