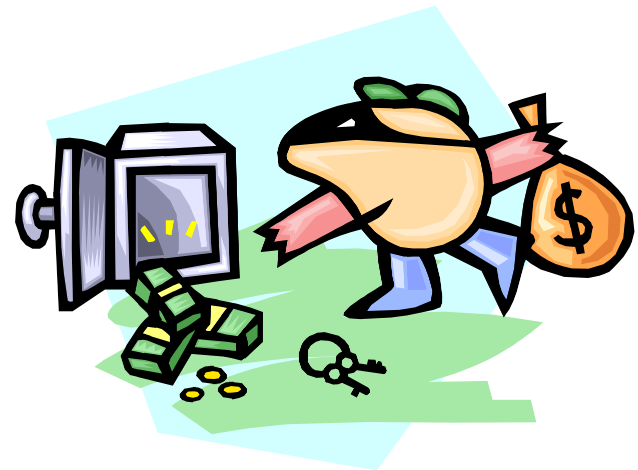
Computer Security

What is Security?

- Example: Is DOS (a single user system with no access control) secure?
  - What if the machine has no data?
  - What if it has the companies financial data?
  - What if it is in the foyer?
  - What if it is in a locked room?
  - What if it is on the Internet?
  - What if it is behind a firewall?

What is Security?

- Another Example: Department store's weekly takings are banked, is it secure to:
  - Ask a random customer to do it?
  - Ask many random customers to do it?
  - Ask a staff member?
  - Ask several staff members?
  - Hire a security firm?
  - Hire several security firms?



Secure System

- Given a security policy
  - ⇒ specification of allowed and disallowed states of the system
- A goal of a secure system is to ensure the system remains in allowed state

Aspects of Computer Security

- Confidentiality
  - Concealment of data (or resource) from those unauthorized to “know”.
  - Includes knowledge of existence.
- Integrity
  - Trustworthiness of data or a resource.
  - Prevention of unauthorized modification.
  - Includes both data integrity (correctness) and origin integrity (authentication).
- Availability
  - Ability to use data or resource when desired.

Threats

- *Threats* are potential violations of security.
  - Example: The threat of theft
- Threats must be guarded against (even though they may not have occurred)
  - Example: Armored Car
- The act of violating security is called an *attack*, which is performed by *attackers.*

Threats

- Snooping
  - Disclosure
  - Unauthorised interception of data
  - Attack on confidentiality
- Modification or Alteration
  - Unauthorised change of data
  - Attack on integrity
- Masquerading or spoofing
  - Impersonation of one entity by another
  - Attack in authentication integrity
  - Delegation is an issue
- Repudiation of origin
  - False denial of being the source
  - Attack on Integrity
- Denial of receipt
  - False denial of receiving something
  - Availability and Integrity
- Delay
  - Temporary inhibition of service
  - Attack on Availability
- Denial of Service
  - Long-term inhibition of service
  - Attack on availability

Policy and Mechanism

It is important to distinguish between policy and mechanism

- A *security policy* is a statement of what is and what is not allowed.
- A *security mechanism* is a method, tool, procedure for enforcing security policy.

Policy

- Ideally, a security policy unambiguously partitions the system into a set of allowed and disallowed states.
  - Preferably sound mathematical models
  - English descriptions
    - Can be imprecise, ambiguous, conflicting, unenforceable
    - Example: Bank tellers are authorised to transfer up to $100,000 between accounts without branch manager approval
      - Transferring $10,000 to is own account does not violate this policy.

Mechanisms

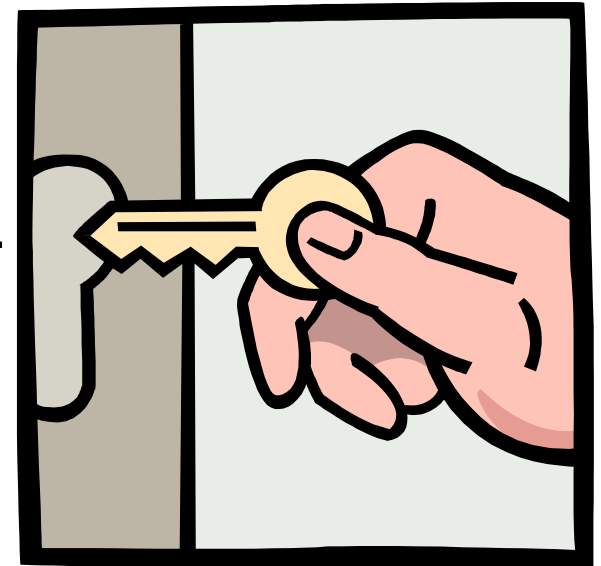
- Used to enforce security policy
- May be computer access control methods, file access control, procedures, tools, etc.
- Example:
  - Policy: Only the accountant can access the financial computer
  - Mechanism: Computer in locked room, only accountant has the key.

Revisiting - Secure System

- Given a security policy
  - ⇒ unambiguous specification of allowed and disallowed states of the system
- A goal of a secure system is to ensure the occurrence of disallowed states are either:
  - Prevented
  - Detected
  - Recoverable

  using the mechanisms available.

Assumptions and Trust



Example:

- Opening a locked door requires the key.
- Assumption:
  - The lock secure against lock picking
    - Assumption appears correct for most people
  - A skilled lock picker will not violate security
    - The lock picker is trustworthy
- Assumption not true in presence of untrustworthy lock picker
  - $\Rightarrow$ opening locked door does not require the key
- Invalid assumptions or misplaced trust results in no security.

Assumptions and Trust

- Implicit (and also explicit) assumptions can result in loss of security

$\Rightarrow$ Assumptions need to be
  - clearly identified
  - evaluated for validity
- *Trusted entities are those entities that can violate security*
  - They are not defined as the entities known to behave correctly according to security policy
  - Ideally, trusted entities also behave correctly
    - Need procedures to assure trustworthiness (correctness)
    - Example: Locksmiths are registered after background checks to reduce likelihood of incorrect behaviour.

Potentially Invalid General Assumptions

- The security policy unambiguously and correctly divides the system into safe and unsafe states.
- Each mechanism is designed to correctly implement one of more parts of the security policy
- The union of all mechanisms covers the security policy
- The mechanisms are implemented correctly
- The mechanisms are installed and administered correctly

Assurance

- Assurance is a process or system for bolstering (substantiating or specifying) trust in an entity.
- Example: Medication uses
    - Certification to ensure the utility and safety of drug
    - Manufacturing quality control ensure what is made is what was certified
    - Safety seals on packages to ensure what was manufactured is what the customer eventually receives
  - Together, the system provide a high degree of assurance to customers that they are getting what they expect

Software Assurance

- Specifications
  - Unambiguous description of system behaviour
  - Formal or informal
- Design
  - Justification that it does not violate the specification
  - Mathematical translation of specification
  - Compelling argument
- Implementation
  - Justification that it actually satisfies the design
  - By mathematical proof, or rigorous testing
  - By transitivity, must also satisfy the spec
- Operation and Maintenance
  - Justification that the system is used and maintained as per original assumptions in the specification
- Assurance does not guarantee correctness or security
  - It provides a basis for determining what must be trusted
  - Conveys the rigor used to construct the system
  - Specification and analysis required improves chances of finding errors.

Summary

- Computer security is dependent of many aspects of a computer system.
- Policy defines security, mechanisms enforces security.
- Important factors are:
  - the assumptions made about what is true or trustworthy
  - Misplaced trust or invalid assumptions provide no security
- Security is relative (not absolute)
  - Given enough resources, an attacker can defeat mechanisms in place.
- Human factors play a part

Protection Mechanisms

- Protection state of system
  - Describes current settings, values of system relevant to protection
- Access control matrix
  - Describes protection state precisely
  - Matrix describing rights of subjects
  - State transitions change elements of matrix

Description

objects (entities)

	o_1	…	o_m	s_1	…	s_n
s_1						
s_2						
…						
s_n						

subjects

- Subjects $S = \{ s_1,\ldots,s_n \}$
- Objects $O = \{ o_1,\ldots,o_m \}$
- Rights $R = \{ r_1,\ldots,r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \ldots, r_y \}$ means subject s_i has rights $r_x, \ldots, r_y$ over object o_j

Example 1

- Processes *p*, *q*
- Files *f*, *g*
- Rights *r*, *w*, *x*, *a*, *o*

	f	*g*	*p*	*q*
p	*rwo*	*r*	*rwxo*	*w*
q	*a*	*ro*	*r*	*rwxo*

Example 2

- Procedures *inc_ctr*, *dec_ctr*, *manage*
- Variable *counter*
- Rights +, –, *call*

	counter	*inc_ctr*	*dec_ctr*	*manage*
inc_ctr	+			
dec_ctr	–			
manage		*call*	*call*	*call*

State Transitions

- Change the protection state of system
- |– represents transition
  - X_i |– $_\tau$ X_{i+1}: command τ moves system from state X_i to X_{i+1}
  - X_i |– * X_{i+1}: a sequence of commands moves system from state X_i to X_{i+1}
- Commands often called *transformation procedures*

Primitive Operations

- **create subject** *s*; **create object** *o*
  - Creates new row, column in ACM; creates new column in ACM
- **destroy subject** *s*; **destroy object** *o*
  - Deletes row, column from ACM; deletes column from ACM
- **enter** *r* **into** *A*[*s*, *o*]
  - Adds *r* rights for subject *s* over object *o*
- **delete** *r* **from** *A*[*s*, *o*]
  - Removes *r* rights from subject *s* over object *o*

Creating File

- Process *p* creates file *f* with *r* and *w* permission

```
command create•file(p, f)
    create object f;
    enter own into A[p, f];
    enter r into A[p, f];
    enter w into A[p, f];
end
```

Mono-Operational Commands

- Make process *p* the owner of file *g*

```
command make•owner(p, g)
    enter own into A[p, g];
end
```

- Mono-operational command
  – Single primitive operation in this command

Conditional Commands

- Let *p* give *q* *r* rights over *f*, if *p* owns *f*

```
command grant•read•file•1(p, f, q)
    if own in A[p, f]
    then
        enter r into A[q, f];
end
```

- Mono-conditional command
  – Single condition in this command

Multiple Conditions

- Let *p* give *q* *r* and *w* rights over *f*, if *p* owns *f* and *p* has *c* rights over *q*

```
command grant•read•file•2(p, f, q)
    if own in A[p, f] and c in A[p, q]
    then
            enter r into A[q, f];
            enter w into A[q, f];
end
```

Copy Right

- Allows possessor to give rights to another
- Often attached to a right, so only applies to that right
  - *r* is read right that cannot be copied
  - *rc* is read right that can be copied
- Is copy flag copied when giving *r* rights?
  - Depends on model, instantiation of model

Own Right

- Usually allows possessor to change entries in ACM column
  - So owner of object can add, delete rights for others
  - May depend on what system allows
    - Can’t give rights to specific (set of) users
    - Can’t pass copy flag to specific (set of) users

Attenuation of Privilege

- Principle says you can't give rights you do not possess
  - Restricts addition of rights within a system
  - Usually *ignored* for owner
    - Why? Owner gives herself rights, gives them to others, deletes her rights.

Key Points

- Access control matrix simplest abstraction mechanism for representing protection state
- Transitions alter protection state
- 6 primitive operations alter matrix
  - Transitions can be expressed as commands composed of these operations and, possibly, conditions

What Is “Secure”?

- Adding a generic right *r* where there was not one is “leaking”
- If a system *S*, beginning in initial state s_0, cannot leak right *r*, it is *safe with respect to the right r*.

Safety Question

- Does there exist an algorithm for determining whether a protection system *S* with initial state s_0 is safe with respect to a generic right *r*?
  – Here, “safe” = “secure” for an abstract model

Mono-Operational Commands

- Answer: *yes*
- Sketch of proof:

  Consider minimal sequence of commands $c_1, \ldots, c_k$ to leak the right.

  – Can omit **delete**, **destroy**

  – Can merge all **create**s into one

  Worst case: insert every right into every entry; with s subjects and o objects initially, and n rights, upper bound is $k \leq n(s+1)(o+1)$

General Case

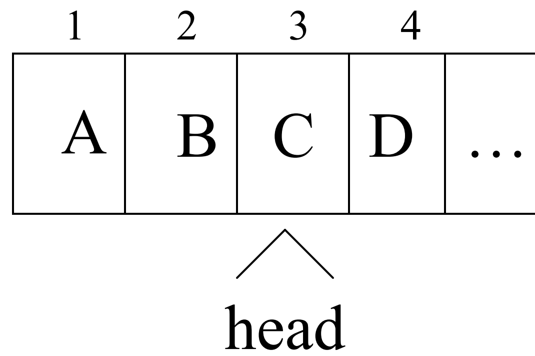
- Answer: *no*
- Sketch of proof:

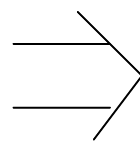
  Reduce halting problem to safety problem

  Turing Machine review:

  – Infinite tape in one direction
  – States K, symbols M; distinguished blank b
  – Transition function $\delta(k, m) = (k', m', L)$ means in state k, symbol m on tape location replaced by symbol m', head moves to left one square, and enters state k'
  – Halting state is q_f; TM halts when it enters this state

Mapping

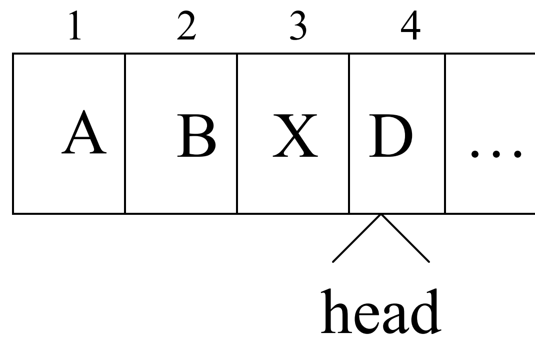


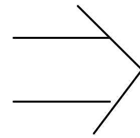


	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			C k	*own*	
s_4				D end	

Current state is k

Mapping





	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				D k_1 end	

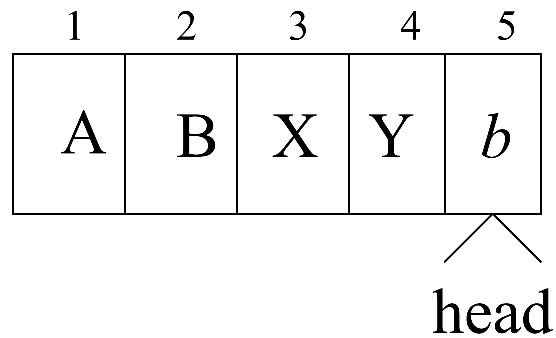
After $\delta(k, \text{C}) = (k_1, \text{X}, \text{R})$ where k is the current state and k_1 the next state

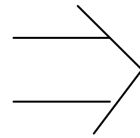
Command Mapping

$\delta(k, \mathrm{C}) = (k_1, \mathrm{X}, \mathrm{R})$ at intermediate becomes

```
command c_{k,C}(s_3, s_4)
if own in A[s_3, s_4] and k in A[s_3, s_3]
      and C in A[s_3, s_3]
then
  delete k from A[s_3, s_3];
  delete C from A[s_3, s_3];
  enter X into A[s_3, s_3];
  enter k_1 into A[s_4, s_4];
end
```

Mapping





	s_1	s_2	s_3	s_4	s_5
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				Y	*own*
s_5					b k_2 end

After $\delta(k_1, \mathrm{D}) = (k_2, \mathrm{Y}, \mathrm{R})$ where k_1 is the current state and k_2 the next state

Command Mapping

$\delta(k_1, \mathrm{D}) = (k_2, \mathrm{Y}, \mathrm{R})$ at end becomes

```
command crightmost_{k,C}(s_4, s_5)
if end in A[s_4, s_4] and k_1 in A[s_4, s_4]
        and D in A[s_4, s_4]
then
   delete end from A[s_4, s_4];
   create subject s_5;
   enter own into A[s_4, s_5];
   enter end into A[s_5, s_5];
   delete k_1 from A[s_4, s_4];
   delete D from A[s_4, s_4];
   enter Y into A[s_4, s_4];
   enter k_2 into A[s_5, s_5];
end
```

Rest of Proof

- Protection system exactly simulates a TM
  - Exactly 1 *end* right in ACM
  - 1 right in entries corresponds to state
  - Thus, at most 1 applicable command
- If TM enters state q_f, then right has leaked
- If safety question decidable, then represent TM as above and determine if q_f leaks
  - Implies halting problem decidable
- Conclusion: safety question undecidable in general

Other Results

- Delete **create** primitive; then safety question is complete in **P-SPACE**
- Delete **destroy**, **delete** primitives; then safety question is undecidable
  - Systems are monotonic
- Safety question for monoconditional, monotonic protection systems is decidable
- Safety question for monoconditional protection systems with **create**, **enter**, **delete** (and no **destroy**) is decidable.

Take-Grant Protection Model

- A specific (not generic) system
  - Set of rules for state transitions
- Safety decidable, and in time linear with the size of the system

Key Points

- Safety problem undecidable
- Limiting scope of systems can make problem decidable
- The set of protection commands that model a particular security policy affects whether safety of that model is decidable.

Security Policy

- Policy partitions system states into:
  - Authorized (secure)
    - These are states the system can enter
  - Unauthorized (nonsecure)
    - If the system enters any of these states, it's a security violation
- Secure system
  - Starts in authorized state
  - Never enters unauthorized state

Confidentiality

- X set of entities, I information
- I has *confidentiality* property with respect to X if no $x \in X$ can obtain information from I
- I can be disclosed to others
- Example:
  - X set of students
  - I final exam answer key
  - I is confidential with respect to X if students cannot obtain final exam answer key

Integrity

- *X* set of entities, *I* information
- *I* has *integrity* property with respect to *X* if all $x \in X$ trust information in *I*
- Types of integrity:
  - trust *I*, its conveyance and protection (data integrity)
  - *I* information about origin of something or an identity (origin integrity, authentication)
  - *I* resource: means resource functions as it should (assurance)

Availability

- X set of entities, I resource
- I has *availability* property with respect to X if all $x \in X$ can access I
- Types of availability:
  - traditional: x gets access or not
  - quality of service: promised a level of access (for example, a specific level of bandwidth) and not meet it, even though some access is achieved

Policy Models

- Abstract description of a policy or class of policies
- Focus on points of interest in policies
  - Security levels in multilevel security models
  - Separation of duty in Clark-Wilson model
  - Conflict of interest in Chinese Wall model

Key Points

- Policies describe *what* is allowed
- Mechanisms control *how* policies are enforced

Confidentiality Policy

- Goal: prevent the unauthorized disclosure of information
  - Deals with information flow
  - Integrity incidental
- Multi-level security models are best-known examples
  - Bell-LaPadula Model basis for many, or most, of these

Bell-LaPadula Model, Step 1

- Security levels arranged in linear ordering
  - Top Secret: highest
  - Secret
  - Confidential
  - Unclassified: lowest
- Levels consist of *security clearance* $L(s)$
  - Objects have *security classification* $L(o)$

Example

security level	*subject*	*object*
Top Secret	Tamara	Personnel Files
Secret	Samuel	E-Mail Files
Confidential	Claire	Activity Logs
Unclassified	Ulaley	Telephone Lists

- Tamara can read all files
- Claire cannot read Personnel or E-Mail Files
- Ulaley can only read Telephone Lists

Reading Information

- Information flows *up*, not *down*
  - "Reads up" disallowed, "reads down" allowed
- Simple Security Condition (Step 1)
  - Subject *s* can read object *o* iff, $L(o) \le L(s)$ and *s* has permission to read *o*
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no reads up" rule

Writing Information

- Information flows up, not down
  - "Writes up" allowed, "writes down" disallowed
- *-Property (Step 1)
  - Subject *s* can write object *o* iff $L(s) \le L(o)$ and *s* has permission to write *o*
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no writes down" rule

Basic Security Theorem, Step 1

- If a system is initially in a secure state, and every transition of the system satisfies the simple security condition, step 1, and the *-property, step 1, then every state of the system is secure
  - Proof: induct on the number of transitions

Bell-LaPadula Model, Step 2

- Expand notion of security level to include categories
- Security level is (*clearance*, *category set*)
- Examples
  - (Top Secret, { NUC, EUR, ASI })
  - (Confidential, { EUR, ASI })
  - (Secret, { NUC, ASI })

Levels and Lattices

- (A, C) *dom* (A', C') iff $A' \leq A$ and $C' \subseteq C$
- Examples
  - (Top Secret, {NUC, ASI}) *dom* (Secret, {NUC})
  - (Secret, {NUC, EUR}) *dom* (Confidential,{NUC, EUR})
  - (Top Secret, {NUC}) $\neg$*dom* (Confidential, {EUR})
- Let C be set of classifications, K set of categories. Set of security levels $L = C \times K$, *dom* form lattice

Levels and Ordering

- Security levels partially ordered
  - Any pair of security levels may (or may not) be related by *dom*
- "dominates" serves the role of "greater than" in step 1
  - "greater than" is a total ordering, though

Reading Information

- Information flows *up*, not *down*
  - "Reads up" disallowed, "reads down" allowed
- Simple Security Condition (Step 2)
  - Subject *s* can read object *o* iff *L*(*s*) *dom* *L*(*o*) and *s* has permission to read *o*
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no reads up" rule

Writing Information

- Information flows up, not down
  - "Writes up" allowed, "writes down" disallowed
- *-Property (Step 2)
  - Subject *s* can write object *o* iff *L*(*o*) *dom* *L*(*s*) and *s* has permission to write *o*
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no writes down" rule

Basic Security Theorem, Step 2

- If a system is initially in a secure state, and every transition of the system satisfies the simple security condition, step 2, and the *-property, step 2, then every state of the system is secure
  - Proof: induct on the number of transitions
  - In actual Basic Security Theorem, discretionary access control treated as third property, and simple security property and *-property phrased to eliminate discretionary part of the definitions — but simpler to express the way done here.

Problem

- Colonel has (Secret, {NUC, EUR}) clearance
- Major has (Secret, {EUR}) clearance
  - Major can talk to colonel (“write up” or “read down”)
  - Colonel cannot talk to major (“read up” or “write down”)
- Clearly absurd!

Solution

- Define maximum, current levels for subjects
  - *maxlevel*(*s*) *dom curlevel*(*s*)
- Example
  - Treat Major as an object (Colonel is writing to him/her)
  - Colonel has *maxlevel* (Secret, { NUC, EUR })
  - Colonel sets *curlevel* to (Secret, { EUR })
  - Now *L*(Major) *dom curlevel*(Colonel)
    - Colonel can write to Major without violating “no writes down”
  - Does *L*(*s*) mean *curlevel*(*s*) or *maxlevel*(*s*)?
    - Formally, we need a more precise notation

Key Points

- Confidentiality models restrict flow of information
- Bell-LaPadula models multilevel security
  - Cornerstone of much work in computer security

Chinese Wall Model

Problem:

- Tony advises American Bank about investments
- He is asked to advise Toyland Bank about investments

- Conflict of interest to accept, because his advice for either bank would affect his advice to the other bank

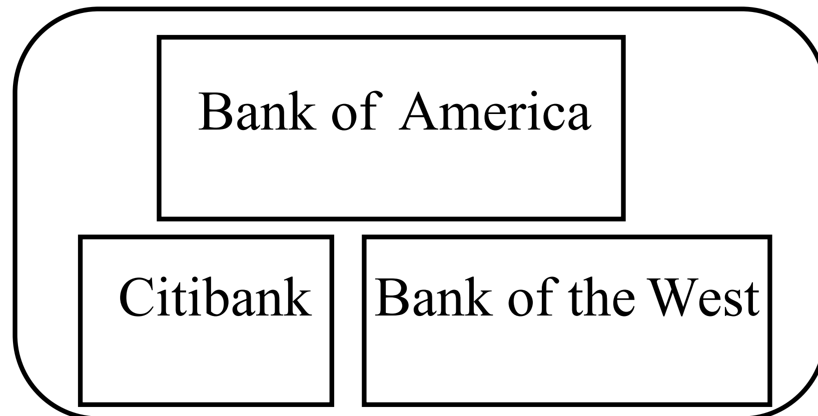
Organization

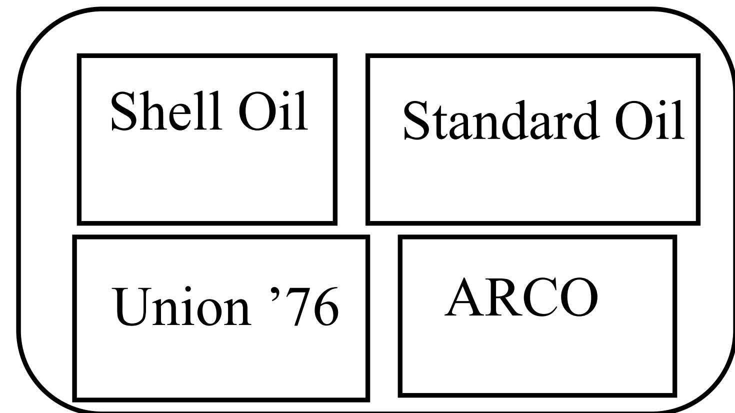
- Organize entities into “conflict of interest” classes
- Control subject accesses to each class
- Control writing to all classes to ensure information is not passed along in violation of rules
- Allow sanitized data to be viewed by everyone

Definitions

- *Objects*: items of information related to a company
- *Company dataset* (CD): contains objects related to a single company
  - Written *CD*(*O*)
- *Conflict of interest class* (COI): contains datasets of companies in competition
  - Written *COI*(*O*)
  - Assume: each object belongs to exactly one *COI* class

Example





Temporal Element

- If Anthony reads any CD in a COI, he can *never* read another CD in that COI
  - Possible that information learned earlier may allow him to make decisions later
  - Let *PR*(*S*) be set of objects that *S* has already read

CW-Simple Security Condition

- s can read o iff either condition holds:
  1. There is an o' such that s has accessed o' and $CD(o') = CD(o)$
     – Meaning s has read something in o's dataset
  2. For all $o' \in O$, $o' \in PR(s) \Rightarrow COI(o') \neq COI(o)$
     – Meaning s has not read any objects in o's conflict of interest class
- Ignores sanitized data (see below)
- Initially, $PR(s) = \varnothing$, so initial read request granted

Sanitization

- Public information may belong to a CD
  - As is publicly available, no conflicts of interest arise
  - So, should not affect ability of analysts to read
  - Typically, all sensitive data removed from such information before it is released publicly (called *sanitization*)
- Add third condition to CW-Simple Security Condition:
  3. *o* is a sanitized object

Writing

- Anthony, Susan work in same trading house
- Anthony can read Bank 1's CD, Gas' CD
- Susan can read Bank 2's CD, Gas' CD
- If Anthony could write to Gas' CD, Susan can read it
  - Hence, indirectly, she can read information from Bank 1's CD, a clear conflict of interest

CW-*-Property

- *s* can write to *o* iff both of the following hold:
  1. The CW-simple security condition permits *s* to read *o*; and
  2. For all *unsanitized* objects *o′*, if *s* can read *o′*, then *CD*(*o′*) = *CD*(*o*)
- Says that s can write to an object if all the (unsanitized) objects it can read are in the same dataset

Mechanisms

- Entity or procedure that enforces some part of the security policy
  – Access controls (like bits to prevent someone from reading a homework file)
  – Disallowing people from bringing CDs and floppy disks into a computer facility to control what is placed on systems

Types of Access Control

- Discretionary Access Control (DAC, IBAC)
  - individual user sets access control mechanism to allow or deny access to an object
- Mandatory Access Control (MAC)
  - system mechanism controls access to object, and individual cannot alter that access

Issues For Access Control Mechanisms

- Propagation of rights:
  - Can agent grant access to another?
- Restriction of rights:
  - Can agent propagate restricted rights?
- Revocation of rights:
  - Can access, once granted, be revoked?
- Amplification of rights:
  - Can unprivileged agent perform restrict operations
- Determination of object accessibility:
  - Which agents have access?
- Determination of agent's protection domain:
  - Which objects are accessible?

Access Control Mechanisms

- Access Control Lists
- Capabilities

Access Control Lists

- Columns of access control matrix

	file1	*file2*	*file3*
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

ACLs:

- file1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
- file2: { (Andy, r) (Betty, r) (Charlie, rwo) }
- file3: { (Andy, rwo) (Charlie, w) }

Abbreviations

- ACLs can be long … so combine users
  - UNIX: 3 classes of users: owner, group, rest
  - rwx rwx rwx

    rest

    group

    owner
  - Ownership assigned based on creating process
    - Some systems: if directory has setgid permission, file group owned by group of directory (SunOS, Solaris)

ACL Modification

- Who can do this?
  - Creator is given *own* right that allows this
  - System R provides a *grant* modifier (like a copy flag) allowing a right to be transferred, so ownership not needed
    - Transferring right to another modifies ACL

Privileged Users

- Do ACLs apply to privileged users (*root*)?
  - Solaris: abbreviated lists do not, but full-blown ACL entries do
  - Other vendors: varies

Access Control Lists

- Propagation of rights:
  - Meta-right (e.g. *own*, chmod)
- Restriction of rights:
  - Meta-right
- Revocation of rights:
  - Meta-right (Owner deletes subject's entries from ACL, or rights from subject's entry in ACL)
- Amplification of rights:
  - Protected invocation right (e.g. setuid)
- Determination of object accessibility:
  - explicit in ACL
- Determination of agent's protection domain:
  - hard (if not impossible)

Capability Lists

- Rows of access control matrix

	file1	*file2*	*file3*
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

C-Lists:

- Andy: { (file1, rx) (file2, r) (file3, rwo) }
- Betty: { (file1, rwxo) (file2, r) }
- Charlie: { (file1, rx) (file2, rwo) (file3, w) }

Semantics

- Like a bus ticket
  - Mere possession indicates rights that subject has over object
  - Object identified by capability (as part of the token)
    - Name may be a reference, location, or something else
- Must prevent process from altering capabilities
  - Otherwise subject could change rights encoded in capability or object to which they refer
- Implemented as
  - Tagged (protected by hardware)
  - Partitioned (protected by software)
  - Sparse (protected by obscurity)

Tagged Capabilities

- Tag bit(s) with every (group of) memory word(s):
  - Tags identify capabilities
  - Capabilities used like “normal” pointers.
  - Hardware checks permissions on dereferencing capability
  - Users can copy capabilities
  - Modifications turn tags off.
  - Only privileged instructions (kernel) can turn them on.
- Examples:
  - B5700: tag was 3 bits and indicated how word was to be treated (pointer, type, descriptor, *etc*.)
  - IBM System/38, AS/400, i-series; many historical systems.

Tagged Capabilities

- Propagation of rights:
  - Easy (copy)
- Restriction of rights:
  - Requires kernel intervention
- Revocation of rights:
  - Requires scanning of ALL data (with system stopped)
- Amplification of rights:
  - More later
- Determination of object accessibility:
  - Again, requires impractical scanning
- Determination of agent's protection domain:
  - Difficult, requires scanning subject

Partitioned Capabilities

- Capabilities are segregated from applications in C-List
  - Programs must refer to them by pointers (indexes)
    - Otherwise, program could use a copy of the capability—which it could modify
  - Paging/segmentation protections
    - Like tags, but put capabilities in a read-only segment or page
      - CAP system did this
  - Or, Keep C-list in the PCB (in kernel)
- System validates access via C-list when mapping a page
- Examples: Hydra, Mach, KeyKOS, Grasshopper, Eros, others (maybe future L4)

Partitioned Capabilities

- Validation fast
  - Cap reference points (in)directly to object and rights
- Propagation of rights:
  - Requires kernel intervention
- Restriction of rights:
  - Requires kernel intervention
- Revocation of rights:
  - Requires scanning of all C-Lists
- Amplification of rights:
  - More later
- Determination of object accessibility:
  - Again, requires impractical scanning
- Determination of agent's protection domain:
  - Explicit in C-List
- Reference counting (garbage collection) possible.

Sparse Capabilities

- Basic idea is to add a bit-string to make valid capabilities a very small subset of capability space
  - Cryptography
    - Associate with each capability a cryptographic checksum enciphered using a key known to OS
    - When process presents capability, OS validates checksum
  - Passwords
    - Requires copy of access rights in server
    - No encryption, easy to validate
- Capabilities can be passed around like data
  - No need for kernel intervention
- Good for user-level servers and distributed systems

Partitioned Capabilities

- Validation
  - depends (crypto versus password)
- Propagation of rights:
  - Copy as normal data
- Restriction of rights:
  - Usually requires kernel to make new cap
- Revocation of rights:
  - Done by remove entry in object table the capability refers to.
- Amplification of rights:
  - More later
- Determination of object accessibility:
  - Impossible
- Determination of agent's protection domain:
  - Depends
    - Impossible in general
    - Possible if restrictions place on cap presentation

Amplifying

- Allows *temporary* increase of privileges
- Needed for modular programming
  - Module pushes, pops data onto stack

    ```
    module stack … endmodule.
    ```
  - Variable *x* declared of type stack

    ```
    var x: module;
    ```
  - *Only* stack module can alter, read *x*
    - So process doesn't get capability, but needs it when *x* is referenced—a problem!
  - Solution: give process the required capabilities while it is in module

Examples

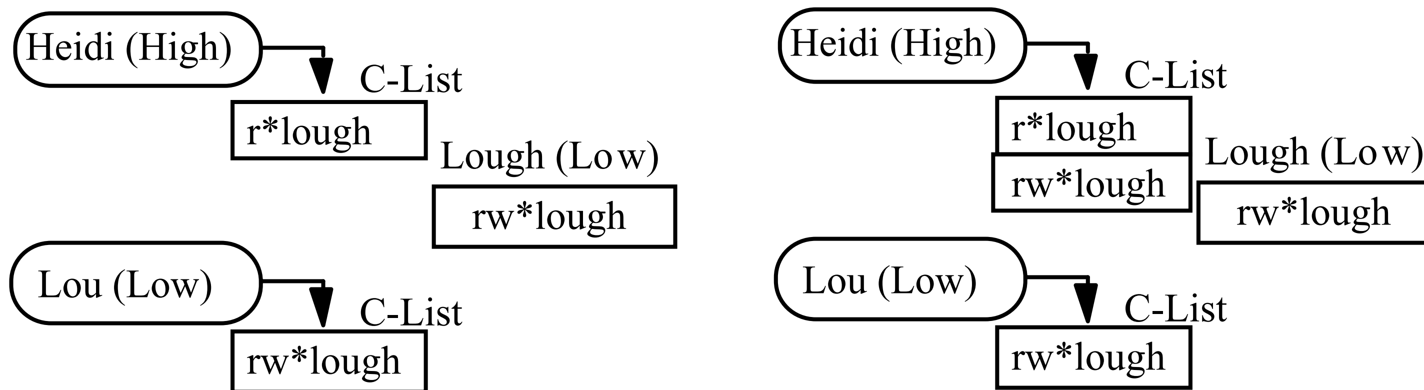
- HYDRA: templates
  - Associated with each procedure, function in module
  - Adds rights to process capability *while the procedure or function is being executed*
  - Rights deleted on exit
- Intel iAPX 432: access descriptors for objects
  - These are really capabilities
  - 1 bit in this controls amplification
  - When ADT constructed, permission bits of type control object set to what procedure needs
  - On call, if amplification bit in this permission is set, the above bits or'ed with rights in access descriptor of object being passed

Revocation

- Scan all C-lists, remove relevant capabilities
  - Far too expensive!
- Use indirection
  - Each object has entry in a global object table
  - Names in capabilities name the entry, not the object
    - To revoke, zap the entry in the table
    - Can have multiple entries for a single object to allow control of different sets of rights and/or groups of users for each object
  - Example: Amoeba: owner requests server change random number in server table
    - All capabilities for that object now invalid

Limits

- Problems if you don't control copying of capabilities



The capability to write file *lough* is Low, and Heidi is High so she reads (copies) the capability; now she can write to a Low file, violating the *-property!

Remedies

- Label capability itself
  - Rights in capability depends on relation between its compartment and that of object to which it refers
    - In example, as as capability copied to High, and High dominates object compartment (Low), write right removed
- Check to see if passing capability violates security properties
  - In example, it does, so copying refused
- Distinguish between “read” and “copy capability”
  - Take-Grant Protection Model does this (“read”, “take”)
- Note: Data (sparse) capabilities are problematic
  - No way to determine if permitted data or disallowed capability is transferred.

ACLs vs. Capabilities

- Both theoretically equivalent; consider 2 questions
  1. Given a subject, what objects can it access, and how?
  2. Given an object, what subjects can access it, and how?
  - ACLs answer second easily; C-Lists, first
- Suggested that the second question, which in the past has been of most interest, is the reason ACL-based systems more common than capability-based systems
  - As first question becomes more important (in incident response, for example), this may change
- Additionally, ACLs usually have *owner* right
  - DAC is what most users expect
  - MAC not possible with additional overriding mechanisms

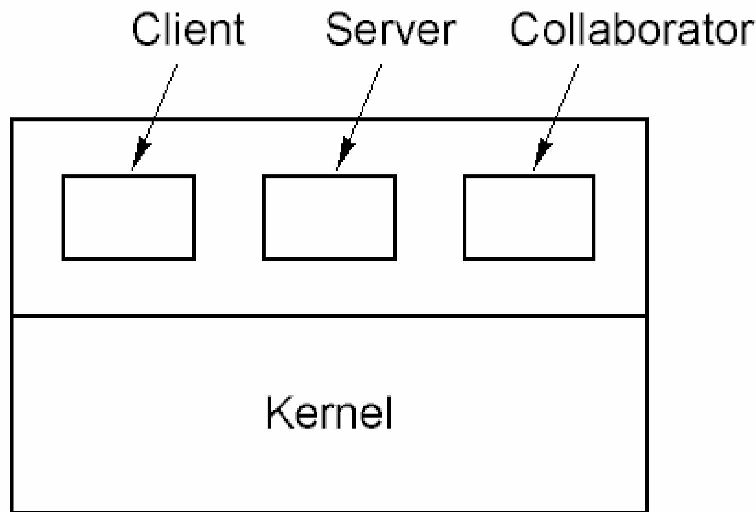
Confinement

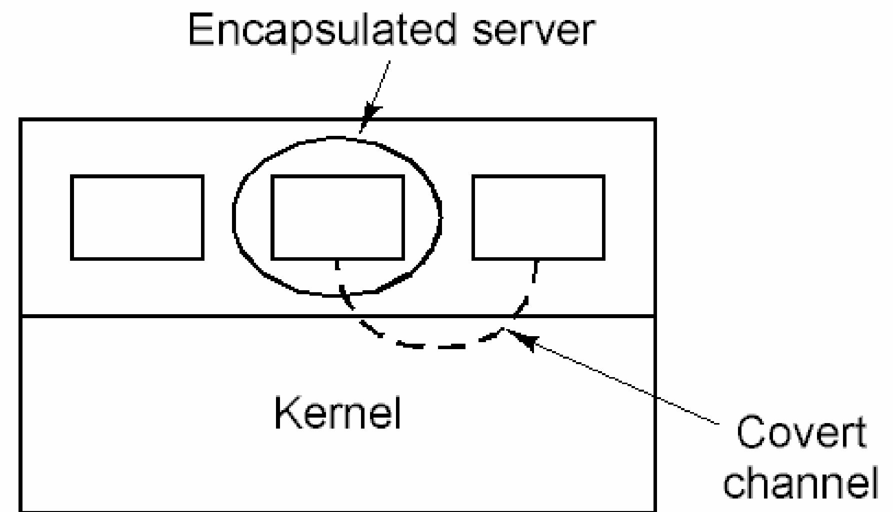
- Confinement is the problem preventing a program leaking information considered confidential
  - Several other formulations (stronger and weaker)
- In practice difficult to achieve due to covert channels
  - Covert channel: A path of communication that was not design for communication

Covert Channels

- A *covert storage channel* uses an attribute of a shared resource.
- A *covert timing channel* uses a temporal or ordering relationship among accesses to a shared resource.

Covert Channels





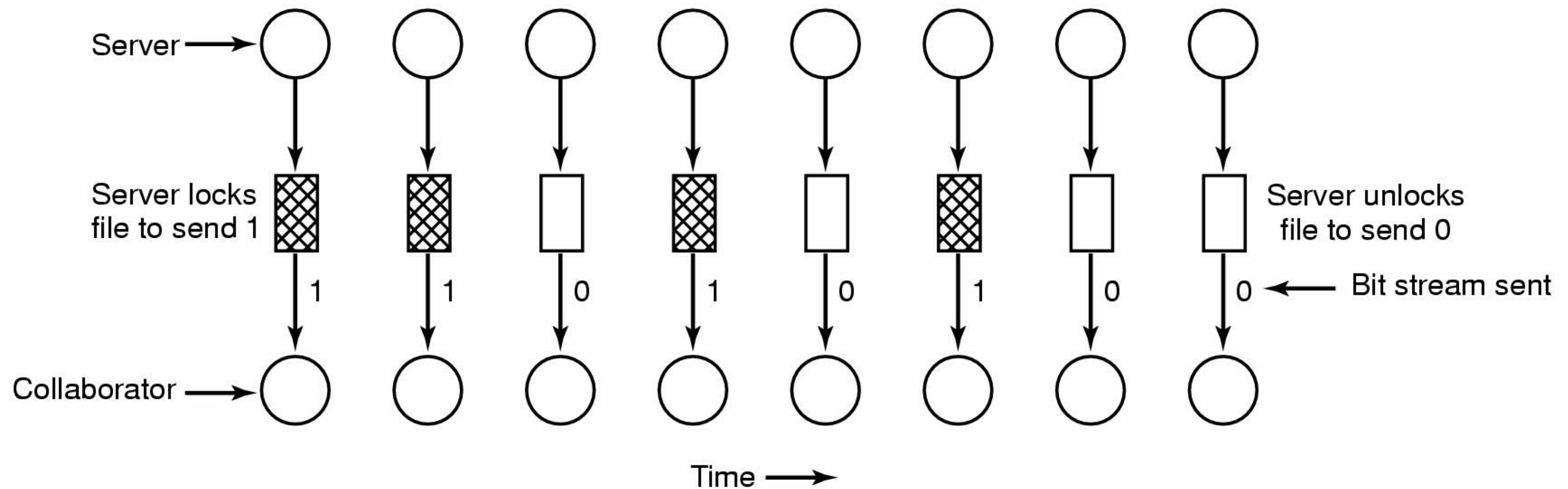
Client, server and collaborator processes

We'd like to confine the server so as to not pass on client's info

Encapsulated server can still leak to collaborator via covert channels

Example: CPU modulation

Covert Channels



A covert channel using file locking
(Assuming a common read-only file)

Covert Channels

- Can be created using a any shared resource whose behaviour can be monitored
  - Network Bandwidth
  - CPU time
  - Disk Response time
  - Disk Bandwidth

Design Principles

- Overview
- Principles
  - Least Privilege
  - Fail-Safe Defaults
  - Economy of Mechanism
  - Complete Mediation
  - Open Design
  - Separation of Privilege
  - Least Common Mechanism
  - Psychological Acceptability

Overview

- Simplicity
  - Less to go wrong
  - Fewer possible inconsistencies
  - Easy to understand
- Restriction
  - Minimize access
  - Inhibit communication

Least Privilege

- A subject should be given only those privileges necessary to complete its task
  - Function, not identity, controls
  - Rights added as needed, discarded after use
  - Minimal protection domain

Fail-Safe Defaults

- Default action is to deny access
- If action fails, system as secure as when action began

Economy of Mechanism

- Keep it as simple as possible
  - KISS Principle
- Simpler means less can go wrong
  - And when errors occur, they are easier to understand and fix
- Interfaces and interactions

Complete Mediation

- Check every access
- Usually done once, on first action
  - UNIX: access checked on open, not checked thereafter
- If permissions change after, may get unauthorized access

Open Design

- Security should not depend on secrecy of design or implementation
  - Popularly misunderstood to mean that source code should be public
  - “Security through obscurity”
  - Does not apply to information such as passwords or cryptographic keys

Separation of Privilege

- Require multiple conditions to grant privilege
  - Separation of duty

Least Common Mechanism

- Mechanisms should not be shared
  - Information can flow along shared channels
  - Covert channels
- Isolation
  - Virtual machines
  - Sandboxes

Psychological Acceptability

- Security mechanisms should not add to difficulty of accessing resource
  - Hide complexity introduced by security mechanisms
  - Ease of installation, configuration, use
  - Human factors critical here

Key Points

- Principles of secure design underlie all security-related mechanisms
- Require:
  - Good understanding of goal of mechanism and environment in which it is to be used
  - Careful analysis and design
  - Careful implementation