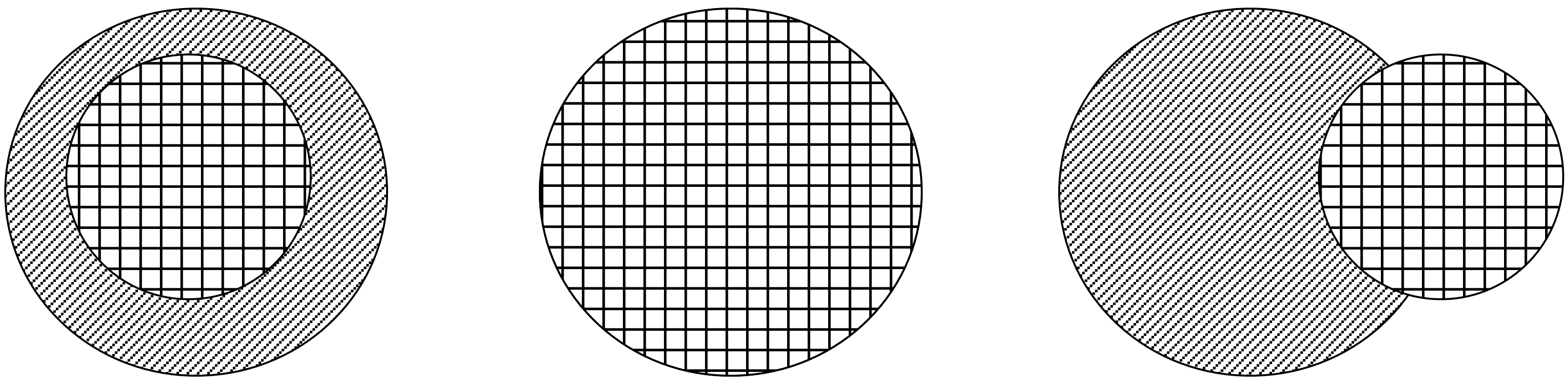
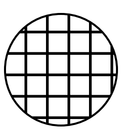
April 17: Policy

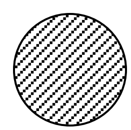
- Limits on secure and precise mechanisms
- Bell-LaPadula confidentiality model
- Tranquility
- Declassification
- McLean's criticism and System Z

Types of Mechanisms



secure precise broad

 set of reachable states

 set of secure states

Secure, Precise Mechanisms

- Can one devise a procedure for developing a mechanism that is both secure *and* precise?
  - Consider confidentiality policies only here
  - Integrity policies produce same result
- Program a function with multiple inputs and one output
  - Let p be a function $p: I_1 \times ... \times I_n \rightarrow R$. Then p is a program with n inputs $i_k \in I_k$, $1 \le k \le n$, and one output $r \rightarrow R$

Programs and Postulates

- Observability Postulate: the output of a function encodes all available information about its inputs
  – Covert channels considered part of the output
- Example: authentication function
  – Inputs name, password; output Good or Bad
  – If name invalid, immediately print Bad; else access database
  – Problem: time output of Bad, can determine if name valid
  – This means timing is part of output

Protection Mechanism

- Let p be a function $p: I_1 \times \ldots \times I_n \rightarrow R$. A *protection mechanism* m is a function

$$m: I_1 \times \ldots \times I_n \rightarrow R \cup E$$

  for which, when $i_k \in I_k$, $1 \leq k \leq n$, either
  - $m(i_1, \ldots, i_n) = p(i_1, \ldots, i_n)$ or
  - $m(i_1, \ldots, i_n) \in E$.
- E is set of error outputs
  - In above example, E = { "Password Database Missing", "Password Database Locked" }

Confidentiality Policy

- Confidentiality policy for program *p* says which inputs can be revealed
  - Formally, for p: $I_1 \times \ldots \times I_n \rightarrow R$, it is a function c: $I_1 \times \ldots \times I_n \rightarrow A$, where $A \subseteq I_1 \times \ldots \times I_n$
  - A is set of inputs available to observer
- Security mechanism is function

$$m: I_1 \times \ldots \times I_n \rightarrow R \cup E$$

  - m is *secure* if and only if $\exists\, m'$: $A \rightarrow R \cup E$ such that, $\forall i_k \in I_k$, $1 \le k \le n$, $m(i_1, \ldots, i_n) = m'(c(i_1, \ldots, i_n))$
  - m returns values consistent with c

Examples

- $c(i_1, ..., i_n) = C$, a constant
  – Deny observer any information (output does not vary with inputs)
- $c(i_1, ..., i_n) = (i_1, ..., i_n)$, and $m' = m$
  – Allow observer full access to information
- $c(i_1, ..., i_n) = i_1$
  – Allow observer information about first input but no information about other inputs.

Precision

- Security policy may be over-restrictive
  - Precision measures how over-restrictive
- m_1, m_2 distinct protection mechanisms for program p under policy c
  - m_1 *as precise as* m_2 ($m_1 \approx m_2$) if, for all inputs $i_1, \ldots, i_n$, $m_2(i_1, \ldots, i_n) = p(i_1, \ldots, i_n) \Rightarrow m_1(i_1, \ldots, i_n) = p(i_1, \ldots, i_n)$
  - m_1 *more precise than* m_2 ($m_1 \sim m_2$) if there is an input $(i_1', \ldots, i_n')$ such that $m_1(i_1', \ldots, i_n') = p(i_1', \ldots, i_n')$ and $m_2(i_1', \ldots, i_n') \neq p(i_1', \ldots, i_n')$.

Combining Mechanisms

- m_1, m_2 protection mechanisms
- $m_3 = m_1 \cup m_2$
  - For inputs on which m_1 and m_2 return same value as p, m_3 does also; otherwise, m_3 returns same value as m_1
- Theorem: if m_1, m_2 secure, then m_3 secure
  - Also, $m_3 \approx m_1$ and $m_3 \approx m_2$
  - Follows from definitions of secure, precise, and m_3

Existence Theorem

- For any program p and security policy c, there exists a precise, secure mechanism m^* such that, for all secure mechanisms m associated with p and c, $m^* \approx m$
  - Maximally precise mechanism
  - Ensures security
  - Minimizes number of denials of legitimate actions

Lack of Effective Procedure

- There is no effective procedure that determines a maximally precise, secure mechanism for any policy and program.
  - Sketch of proof: let policy c be constant function, and p compute function $T(x)$. Assume $T(x) = 0$. Consider program q, where

```
p;
if z = 0 then y := 1 else y := 2;
halt;
```

Rest of Sketch

- m associated with q, y value of m, z output of p corresponding to $T(x)$
- $\forall x[T(x) = 0] \rightarrow m(x) = 1$
- $\exists x' \, [T(x') \neq 0] \rightarrow m(x) = 2$ or $m(x)\uparrow$
- If you can determine m, you can determine whether $T(x) = 0$ for all x
- Determines some information about input (is it 0?)
- Contradicts constancy of c.
- Therefore no such procedure exists

Key Points

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

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
  - $lub(L) = (max(A), C)$
  - $glb(L) = (min(A), \varnothing)$

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

Formal Model

- Allows us to reason precisely about the model
- Provides a formalism to validate systems against

Formal Model Definitions

- S subjects, O objects, P rights
  - Defined rights: r read, a write, w read/write, e empty
- M set of possible access control matrices
- C set of clearances/classifications, K set of categories, $L = C \times K$ set of security levels
- $F = \{ (f_s, f_o, f_c) \}$
  - $f_s(s)$ maximum security level of subject s
  - $f_c(s)$ current security level of subject s
  - $f_o(o)$ security level of object o

More Definitions

- Hierarchy functions $H: O \rightarrow P(O)$
- Requirements
  1. $o_i \neq o_j \Rightarrow h(o_i) \cap h(o_j) = \varnothing$
  2. There is no set $\{ o_1, \ldots, o_k \} \subseteq O$ such that, for $i = 1, \ldots, k$, $o_{i+1} \in h(o_i)$ and $o_{k+1} = o_1$.
- Example
  - Tree hierarchy; take $h(o)$ to be the set of children of o
  - No two objects have any common children (#1)
  - There are no loops in the tree (#2)