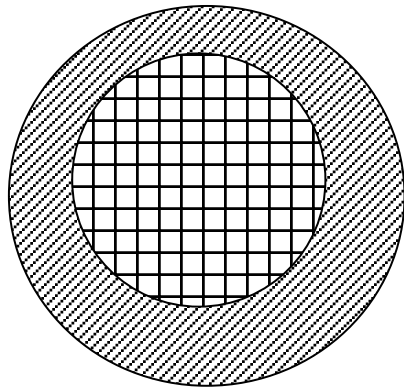


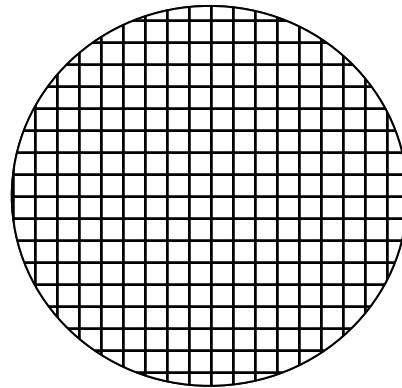
Lecture #9

- Secure and precise mechanisms
 - Can we do both?
- Bell-LaPadula model
 - Informal: lattice version
 - Formal: more mathematical one (but still a lattice!)

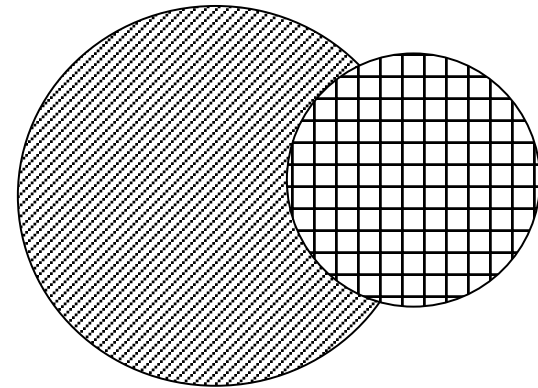
Types of Mechanisms



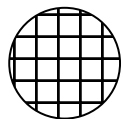
secure



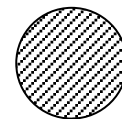
precise



broad



set of reachable states



set of secure states

Implementation

- Adds campus-specific requirements and procedures
 - Example: “incidental personal use” not allowed if it benefits a non-university organization
 - Allows implementation to take into account differences between campuses, such as self-governance by Academic Senate
- Procedures for inspecting, monitoring, disclosing e-mail contents
- Backups

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 $p: I_1 \times \dots \times I_n \rightarrow R$. Then p is a program with n inputs $i_k \in I_k$, $1 \leq k \leq n$, 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 function $p: I_1 \times \dots \times I_n \rightarrow R$. Protection mechanism m is a function $m: I_1 \times \dots \times I_n \rightarrow R \cup E$ for which, when $i_k \in I_k$, $1 \leq k \leq n$, either
 - $m(i_1, \dots, i_n) = p(i_1, \dots, i_n)$ or
 - $m(i_1, \dots, i_n) \in E$.
- E is set of error outputs
 - In above example, $E = \{ \text{“Password Database Missing”}, \text{“Password Database Locked”} \}$

Confidentiality Policy

- Confidentiality policy for program p says which inputs can be revealed
 - Formally, for $p: I_1 \times \dots \times I_n \rightarrow R$, it is a function $c: I_1 \times \dots \times I_n \rightarrow A$, where $A \subseteq I_1 \times \dots \times I_n$
 - A is set of inputs available to observer
- Security mechanism is function $m: I_1 \times \dots \times I_n \rightarrow R \cup E$
 - m secure iff $\exists m': A \rightarrow R \cup E$ such that,
for all $i_k \in I_k, 1 \leq k \leq n, m(i_1, \dots, i_n) = m'(c(i_1, \dots, i_n))$
 - m returns values consistent with c

Examples

- $c(i_1, \dots, i_n) = C$, a constant
 - Deny observer any information (output does not vary with inputs)
- $c(i_1, \dots, i_n) = (i_1, \dots, i_n)$, and $m' = m$
 - Allow observer full access to information
- $c(i_1, \dots, i_n) = i_1$
 - 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, \dots, i_n ,
 $m_2(i_1, \dots, i_n) = p(i_1, \dots, i_n) \Rightarrow m_1(i_1, \dots, i_n) = p(i_1, \dots, i_n)$
 - m_1 more precise than m_2 ($m_1 \sim m_2$) if there is an input
 (i_1', \dots, i_n') such that $m_1(i_1', \dots, i_n') = p(i_1', \dots, i_n')$ and
 $m_2(i_1', \dots, i_n') \neq p(i_1', \dots, i_n')$.

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 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

Overview

- Bell-LaPadula
 - Informally
 - Formally
 - Example Instantiation
- Tranquility
- Controversy
 - System Z

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

<i>security level</i>	<i>subject</i>	<i>object</i>
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, $L(o) \leq L(s)$ 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 $L(s) \leq L(o)$ 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

- $(A, C) \text{ dom } (A', C')$ iff $A' \leq A$ and $C' \subseteq C$
- Examples
 - $(\text{Top Secret}, \{\text{NUC}, \text{ASI}\}) \text{ dom } (\text{Secret}, \{\text{NUC}\})$
 - $(\text{Secret}, \{\text{NUC}, \text{EUR}\}) \text{ dom } (\text{Confidential}, \{\text{NUC}, \text{EUR}\})$
 - $(\text{Top Secret}, \{\text{NUC}\}) \not\text{dom } (\text{Confidential}, \{\text{EUR}\})$
- Let C be set of classifications, K set of categories. Set of security levels $L = C \times K$, dom form lattice
 - $\text{lub}(L) = (\max(A), C)$
 - $\text{glb}(L) = (\min(A), \emptyset)$

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 $L(s) \text{ dom } L(o)$ 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 $L(o) \text{ dom } L(s)$ 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
 - $maxlevel(s) \text{ dom } curlevel(s)$
- 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 $L(\text{Major}) \text{ dom } curlevel(\text{Colonel})$
 - 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 Definitions

- S subjects, O objects, P rights
 - Defined rights: \underline{r} read, \underline{a} write, \underline{w} read/write, \underline{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. $o_i \neq o_j \Rightarrow h(o_i) \cap h(o_j) = \emptyset$
 2. There is no set $\{o_1, \dots, o_k\} \subseteq O$ such that, for $i = 1, \dots, k$, $o_{i+1} \in h(o_i)$ 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)

States and Requests

- V set of states
 - Each state is (b, m, f, h)
 - b is like m , but excludes rights not allowed by f
- R set of requests for access
- D set of outcomes
 - \underline{y} allowed, \underline{n} not allowed, \underline{i} illegal, \underline{o} error
- W set of actions of the system
 - $W \subseteq R \times D \times V \times V$

History

- $X = R^N$ set of sequences of requests
- $Y = D^N$ set of sequences of decisions
- $Z = V^N$ set of sequences of states
- Interpretation
 - At time $t \in N$, system is in state $z_{t-1} \in V$; request $x_t \in R$ causes system to make decision $y_t \in D$, transitioning the system into a (possibly new) state $z_t \in V$
- System representation: $\Sigma(R, D, W, z_0) \in X \times Y \times Z$
 - $(x, y, z) \in \Sigma(R, D, W, z_0)$ iff $(x_t, y_t, z_{t-1}, z_t) \in W$ for all t
 - (x, y, z) called an *appearance* of $\Sigma(R, D, W, z_0)$

Example

- $S = \{ s \}, O = \{ o \}, P = \{ \underline{r}, \underline{w} \}$
- $C = \{ \text{High}, \text{Low} \}, K = \{ \text{All} \}$
- For every $f \in F$, either $f_c(s) = (\text{High}, \{ \text{All} \})$ or $f_c(s) = (\text{Low}, \{ \text{All} \})$
- Initial State:
 - $b_1 = \{ (s, o, \underline{r}) \}, m_1 \in M$ gives s read access over o , and for $f_1 \in F, f_{c,1}(s) = (\text{High}, \{ \text{All} \}), f_{o,1}(o) = (\text{Low}, \{ \text{All} \})$
 - Call this state $v_0 = (b_1, m_1, f_1, h_1) \in V$.

First Transition

- Now suppose in state v_0 : $S = \{ s, s' \}$
- Suppose $f_{c,1}(s) = (\text{Low}, \{\text{All}\})$
- $m_1 \in M$ gives s and s' read access over o
- As s' not written to o , $b_1 = \{ (s, o, \underline{r}) \}$
- $z_0 = v_0$; if s' requests r_1 to write to o :
 - System decides $d_1 = \underline{y}$
 - New state $v_1 = (b_2, m_1, f_1, h_1) \in V$
 - $b_2 = \{ (s, o, \underline{r}), (s', o, \underline{w}) \}$
 - Here, $x = (r_1)$, $y = (\underline{y})$, $z = (v_0, v_1)$

Second Transition

- Current state $v_1 = (b_2, m_1, f_1, h_1) \in V$
 - $b_2 = \{ (s, o, \underline{r}), (s', o, \underline{w}) \}$
 - $f_{c,1}(s) = (\text{High}, \{ \text{All} \}), f_{o,1}(o) = (\text{Low}, \{ \text{All} \})$
- s' requests r_2 to write to o :
 - System decides $d_2 = \underline{n}$ (as $f_{c,1}(s) \text{ dom } f_{o,1}(o)$)
 - New state $v_2 = (b_2, m_1, f_1, h_1) \in V$
 - $b_2 = \{ (s, o, \underline{r}), (s', o, \underline{w}) \}$
 - So, $x = (r_1, r_2), y = (\underline{y}, \underline{n}), z = (v_0, v_1, v_2)$, where $v_2 = v_1$

Basic Security Theorem

- Define action, secure formally
 - Using a bit of foreshadowing for “secure”
- Restate properties formally
 - Simple security condition
 - *-property
 - Discretionary security property
- State conditions for properties to hold
- State Basic Security Theorem

Action

- A request and decision that causes the system to move from one state to another
 - Final state may be the same as initial state
- $(r, d, v, v') \in R \times D \times V \times V$ is an *action* of $\Sigma(R, D, W, z_0)$ iff there is an $(x, y, z) \in \Sigma(R, D, W, z_0)$ and a $t \in N$ such that $(r, d, v, v') = (x_t, y_t, z_t, z_{t-1})$
 - Request r made when system in state v ; decision d moves system into (possibly the same) state v'
 - Correspondence with (x_t, y_t, z_t, z_{t-1}) makes states, requests, part of a sequence

Simple Security Condition

- $(s, o, p) \in S \times O \times P$ satisfies the simple security condition relative to f (written *ssc rel f*) iff one of the following holds:
 1. $p = \underline{e}$ or $p = \underline{a}$
 2. $p = \underline{r}$ or $p = \underline{w}$ and $f_s(s) \text{ dom } f_o(o)$
- Holds vacuously if rights do not involve reading
- If all elements of b satisfy *ssc rel f*, then state satisfies simple security condition
- If all states satisfy simple security condition, system satisfies simple security condition

Necessary and Sufficient

- $\Sigma(R, D, W, z_0)$ satisfies the simple security condition for any secure state z_0 iff for every action $(r, d, (b, m, f, h), (b', m', f', h'))$, W satisfies
 - Every $(s, o, p) \in b - b'$ satisfies *ssc rel f*
 - Every $(s, o, p) \in b'$ that does not satisfy *ssc rel f* is not in b
- Note: “secure” means z_0 satisfies *ssc rel f*
- First says every (s, o, p) added satisfies *ssc rel f*; second says any (s, o, p) in b' that does not satisfy *ssc rel f* is deleted

*-Property

- $b(s: p_1, \dots, p_n)$ set of all objects that s has p_1, \dots, p_n access to
- State (b, m, f, h) satisfies the *-property iff for each $s \in S$ the following hold:
 1. $b(s: \underline{a}) \neq \emptyset \Rightarrow [\forall o \in b(s: \underline{a}) [f_o(o) \text{ dom } f_c(s)]]$
 2. $b(s: \underline{w}) \neq \emptyset \Rightarrow [\forall o \in b(s: \underline{w}) [f_o(o) = f_c(s)]]$
 3. $b(s: \underline{r}) \neq \emptyset \Rightarrow [\forall o \in b(s: \underline{r}) [f_c(s) \text{ dom } f_o(o)]]$
- Idea: for writing, object dominates subject; for reading, subject dominates object

*-Property

- If all states satisfy simple security condition, system satisfies simple security condition
- If a subset S' of subjects satisfy *-property, then *-property satisfied relative to $S' \subseteq S$
- Note: tempting to conclude that *-property includes simple security condition, but this is false
 - See condition placed on w right for each

Necessary and Sufficient

- $\Sigma(R, D, W, z_0)$ satisfies the *-property relative to $S' \subseteq S$ for any secure state z_0 iff for every action $(r, d, (b, m, f, h), (b', m', f', h'))$, W satisfies the following for every $s \in S'$
 - Every $(s, o, p) \in b - b'$ satisfies the *-property relative to S'
 - Every $(s, o, p) \in b'$ that does not satisfy the *-property relative to S' is not in b
- Note: “secure” means z_0 satisfies *-property relative to S'
- First says every (s, o, p) added satisfies the *-property relative to S' ; second says any (s, o, p) in b' that does not satisfy the *-property relative to S' is deleted

Discretionary Security Property

- State (b, m, f, h) satisfies the discretionary security property iff, for each $(s, o, p) \in b$, then $p \in m[s, o]$
- Idea: if s can read o , then it must have rights to do so in the access control matrix m
- This is the discretionary access control part of the model
 - The other two properties are the mandatory access control parts of the model

Necessary and Sufficient

- $\Sigma(R, D, W, z_0)$ satisfies the ds-property for any secure state z_0 iff, for every action $(r, d, (b, m, f, h), (b', m', f', h'))$, W satisfies:
 - Every $(s, o, p) \in b - b'$ satisfies the ds-property
 - Every $(s, o, p) \in b'$ that does not satisfy the ds-property is not in b
- Note: “secure” means z_0 satisfies ds-property
- First says every (s, o, p) added satisfies the ds-property; second says any (s, o, p) in b' that does not satisfy the *-property is deleted

Secure

- A system is secure iff it satisfies:
 - Simple security condition
 - *-property
 - Discretionary security property
- A state meeting these three properties is also said to be secure

Basic Security Theorem

- $\Sigma(R, D, W, z_0)$ is a secure system if z_0 is a secure state and W satisfies the conditions for the preceding three theorems
 - The theorems are on the slides titled “Necessary and Sufficient”