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
Policies and Mechanisms

• Policy says what is, and is not, allowed
  – This defines “security” for the site/system/etc.
• Mechanisms enforce policies
• Composition of policies
  – If policies conflict, discrepancies may create security vulnerabilities

Types of Mechanisms

- Secure
- Precise

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 \( p: I_1 \times \ldots \times I_n \rightarrow R \). Then \( p \) is a program with \( n \) inputs \( i_k \in I_k \), \( 1 \leq k \leq n \), and one output \( r \in R \)

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, $E = \{ \text{"Password Database Missing"}, \text{"Password Database Locked"} \}$

Confidentiality Policy

• Confidentiality policy for program $p$ says which inputs can be revealed
  – Formally, for $p: I_1 \times \ldots \times I_n \rightarrow R$, 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 $m: I_1 \times \ldots \times I_n \rightarrow R \cup E$
  – $m$ secure iff $\exists m': A \rightarrow R \cup E$ such that, for all $i_k \in I_k$, $1 \leq k \leq n$, $m(i_1, \ldots, i_n) = m'(c(i_1, \ldots, i_n))$
  – $m$ returns values consistent with $c$
Examples

- $c(i_1, \ldots, i_n) = C$, a constant
  - Deny observer any information (output does not vary with inputs)
- $c(i_1, \ldots, i_n) = (i_1, \ldots, i_n)$, and $m' = m$
  - Allow observer full access to information
- $c(i_1, \ldots, i_n) = i_1$
  - 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 $m_2$ ($m_1 \simeq m_2$) 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 $m_2$ ($m_1 \prec m_2$) 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 $m_1$ returns same value as $p$, or $m_2$ returns 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)
• \(\forall x[T(x) = 0] \rightarrow m(x) = 1\)
• \(\exists x' [T(x') \neq 0] \rightarrow m(x) = 2 \text{ or } m(x)\uparrow\)
• If you can determine m, you can determine whether T(x) = 0 for all x
• This is not possible
• Therefore no such procedure exists
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

<table>
<thead>
<tr>
<th>security level</th>
<th>subject</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top Secret</td>
<td>Tamara</td>
<td>Personnel Files</td>
</tr>
<tr>
<td>Secret</td>
<td>Samuel</td>
<td>E-Mail Files</td>
</tr>
<tr>
<td>Confidential</td>
<td>Claire</td>
<td>Activity Logs</td>
</tr>
<tr>
<td>Unclassified</td>
<td>Ulaley</td>
<td>Telephone Lists</td>
</tr>
</tbody>
</table>

- 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})

Lattices

• S set, R: S × S relation
  – If a, b ∈ S, and (a, b) ∈ R, write aRb
• Example
  – I = {1, 2, 3}; R is ≤
  – R = {(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)}
  – So we write 1 ≤ 2 and 3 ≤ 3 but not 3 ≤ 2
Relation Properties

• Reflexive
  – For all \( a \in S \), \( aRa \)
  – On \( I \), \( \leq \) is reflexive as \( 1 \leq 1 \), \( 2 \leq 2 \), \( 3 \leq 3 \)

• Antisymmetric
  – For all \( a, b \in S \), \( aRb \land bRa \Rightarrow a = b \)
  – On \( I \), \( \leq \) is antisymmetric

• Transitive
  – For all \( a, b, c \in S \), \( aRb \land bRc \Rightarrow aRc \)
  – On \( I \), \( \leq \) is transitive as \( 1 \leq 2 \) and \( 2 \leq 3 \) means \( 1 \leq 3 \)

Bigger Example

• \( C \) set of complex numbers
• \( a \in C \Rightarrow a = a_R + a_i \), \( a_R, a_i \) integers
• \( a \leq_C b \) if, and only if, \( a_R \leq b_R \) and \( a_i \leq b_i \)
• \( a \leq_C b \) is reflexive, antisymmetric, transitive
  – As \( \leq \) is over integers, and \( a_R, a_i \) are integers
Partial Ordering

• Relation $R$ orders some members of set $S$
  – If all ordered, it’s total ordering
• Example
  – $\leq$ on integers is total ordering
  – $\leq_C$ is partial ordering on $C$ (because neither $3+5i \leq_C 4+2i$ nor $4+2i \leq_C 3+5i$ holds)

Upper Bounds

• For $a, b \in S$, if $u$ in $S$ with $aRu, bRu$ exists, then $u$ is upper bound
  – Least upper if there is no $t \in S$ such that $aRt, bRt, \text{ and } tRu$
• Example
  – For $1 + 5i, 2 + 4i \in C$, upper bounds include $2 + 5i, 3 + 8i, \text{ and } 9 + 100i$
  – Least upper bound of those is $2 + 5i$
Lower Bounds

• For \( a, b \in S \), if \( l \in S \) with \( lRa, lRb \) exists, then \( l \) is lower bound
  – Greatest lower if there is no \( t \in S \) such that \( tRa, tRb, \) and \( lRt \)

• Example
  – For \( 1 + 5i, 2 + 4i \in C \), lower bounds include \( 0, -1 + 2i, 1 + 1i, \) and \( 1+4i \)
  – Greatest lower bound of those is \( 1 + 4i \)

Lattices

• Set \( S \), relation \( R \)
  – \( R \) is reflexive, antisymmetric, transitive on elements of \( S \)
  – For every \( s, t \in S \), there exists a greatest lower bound under \( R \)
  – For every \( s, t \in S \), there exists a least upper bound under \( R \)
Example

• $S = \{ 0, 1, 2 \}; R = \leq$ is a lattice
  – $R$ is clearly reflexive, antisymmetric, transitive on elements of $S$
  – Least upper bound of any two elements of $S$ is the greater
  – Greatest lower bound of any two elements of $S$ is the lesser