Lecture #7

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

Policy Languages

- Express security policies in a precise way
- High-level languages
  - Policy constraints expressed abstractly
- Low-level languages
  - Policy constraints expressed in terms of program options, input, or specific characteristics of entities on system

High-Level Policy Languages

- Constraints expressed independent of enforcement mechanism
- Constraints restrict entities, actions
- Constraints expressed unambiguously
  - Requires a precise language, usually a mathematical, logical, or programming-like language

Example: Web Browser

- Goal: restrict actions of Java programs that are downloaded and executed under control of web browser
- Language specific to Java programs
- Expresses constraints as conditions restricting invocation of entities

Expressing Constraints

- Entities are classes, methods
  - Class: set of objects that an access constraint constrains
  - Method: set of ways an operation can be invoked
- Operations
  - Instantiation: *s* creates instance of class *c*: $s \dashv c$
  - Invocation: s_1 executes object s_2: $s_1 \mapsto s_2$
- Access constraints
  - **deny**(*s op x*) **when** *b*
  - While *b* is true, subject *s* cannot perform *op* on (subject or class) *x*; empty *s* means all subjects

Sample Constraints

- Downloaded program cannot access password database file on UNIX system
- Program's class and methods for files:

```
class File {
  public file(String name);
  public String getfilename();
  public char read();
```

- Constraint:

```
deny( |-> file.read) when
   (file.getfilename() == "/etc/passwd")
```

Another Sample Constraint

- At most 100 network connections open
- *Socket* class defines network interface
  - *Network.numconns* method giving number of active network connections
- Constraint

```
deny( -| Socket) when
         (Network.numconns >= 100)
```

Low-Level Policy Languages

- Set of inputs or arguments to commands
  - Check or set constraints on system
- Low level of abstraction
  - Need details of system, commands

Example: tripwire

- File scanner that reports changes to file system and file attributes
  – *tw.config* describes what may change

    `/usr/mab/tripwire +gimnpsu012345678-a`

    - Check everything but time of last access ("-a")

  – Database holds previous values of attributes

Example Database Record

```
/usr/mab/tripwire/README 0 ..../. 100600 45763
1 917 10 33242 .gtPvf .gtPvY .gtPvY
0 .ZD4cc0Wr8i21ZKaI..LUOr3 .
0fwo5:hf4e4.8TAqd0V4ubv ?...... ...9b3
1M4GX01xbGIX0oVuGo1h15z3 ?:Y9jfa04rdzM1q:eqt1AP
gHk ?.Eb9yo.2zkEh1XKovX1:d0wF0kfAvC ?
1M4GX01xbGIX2947jdyrior38h15z3 0
```

- file name, version, bitmask for attributes, mode, inode number, number of links, UID, GID, size, times of creation, last modification, last access, cryptographic checksums

Comments

- System administrators not expected to edit database to set attributes properly
- Checking for changes with tripwire is easy
  - Just run once to create the database, run again to check
- Checking for conformance to policy is harder
  - Need to either edit database file, or (better) set system up to conform to policy, then run tripwire to construct database

Example English Policy

- Computer security policy for academic institution
  - Institution has multiple campuses, administered from central office
  - Each campus has its own administration, and unique aspects and needs
- Authorized Use Policy
- Electronic Mail Policy

Authorized Use Policy

- Intended for one campus (Davis) only
- Goals of campus computing
  - Underlying intent
- Procedural enforcement mechanisms
  - Warnings
  - Denial of computer access
  - Disciplinary action up to and including expulsion
- Written informally, aimed at user community

Electronic Mail Policy

- Systemwide, not just one campus
- Three parts
  - Summary
  - Full policy
  - Interpretation at the campus

Summary

- Warns that electronic mail not private
  - Can be read during normal system administration
  - Can be forged, altered, and forwarded
- Unusual because the policy alerts users to the threats
  - Usually, policies say how to prevent problems, but do not define the threats

Summary

- What users should and should not do
  - Think before you send
  - Be courteous, respectful of others
  - Don't interfere with others' use of email
- Personal use okay, provided overhead minimal
- Who it applies to
  - Problem is UC is quasi-governmental, so is bound by rules that private companies may not be
  - Educational mission also affects application

Full Policy

- Context
  - Does not apply to Dept. of Energy labs run by the university
  - Does not apply to printed copies of email
    - Other policies apply here
- E-mail, infrastructure are university property
  - Principles of academic freedom, freedom of speech apply
  - Access without user's permission requires approval of vice chancellor of campus or vice president of UC
  - If infeasible, must get permission retroactively

Uses of E-mail

- Anonymity allowed
  - Exception: if it violates laws or other policies
- Can’t interfere with others’ use of e-mail
  - No spam, letter bombs, e-mailed worms, *etc*.
- Personal e-mail allowed within limits
  - Cannot interfere with university business
  - Such e-mail may be a “university record” subject to disclosure

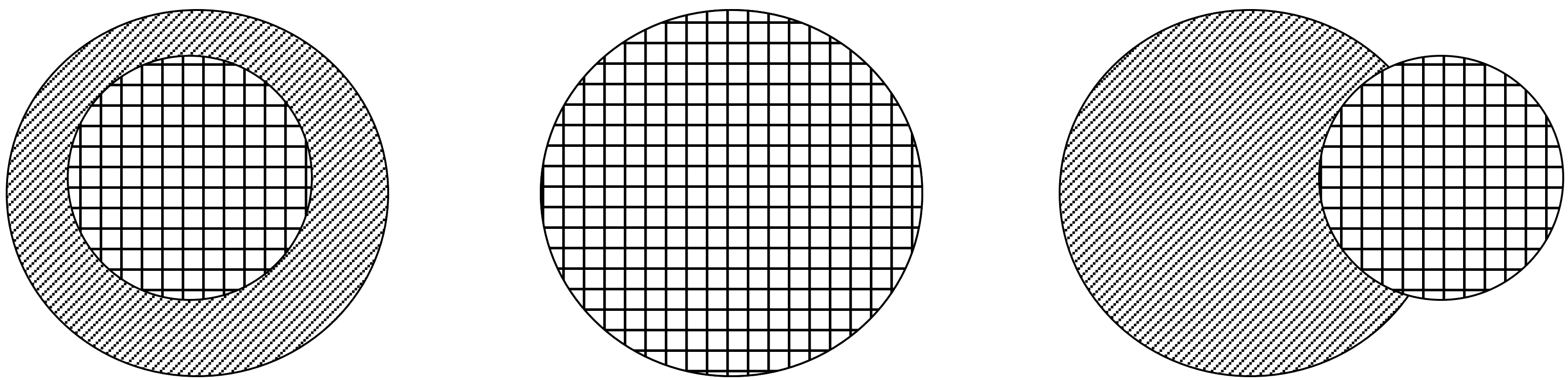
Security of E-mail

- University can read e-mail
  - Won't go out of its way to do so
  - Allowed for legitimate business purposes
  - Allowed to keep e-mail robust, reliable
- Archiving and retention allowed
  - May be able to recover e-mail from end system (backed up, for example)

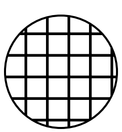
Implementation

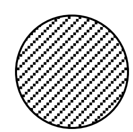
- Adds campus-specific requirements and procedures
  - Example: “incidental personal use” not allowed if it benefits a non-university organization
  - Allows implementation to take into account differences between campuses, such as self-governance by Academic Senate
- Procedures for inspecting, monitoring,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

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 $i_k \in I_k$, $1 \leq k \leq n$, 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 function $p: I_1 \times \ldots \times I_n \rightarrow R$. Protection mechanism m is a function $m: I_1 \times \ldots \times I_n \rightarrow R \cup E$ for which, when $i_k \in I_k$, $1 \le k \le n$, 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
    $$c: I_1 \times \ldots \times I_n \rightarrow A, \text{ where } A \subseteq I_1 \times \ldots \times I_n$$
  - 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 $i_k \in I_k$, $1 \le k \le n$, $m(i_1, \ldots, i_n) = m'(c(i_1, \ldots, i_n))$
  - 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 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

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, $L(o) \leq L(s)$ 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 $L(s) \le L(o)$ 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}) ¬*dom* (Confidential, {EUR})
- Let C be set of classifications, K set of categories. Set of security levels $L = C \times K$, *dom* form lattice
  - $lub(L) = (max(A), C)$
  - $glb(L) = (min(A), \emptyset)$

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*) *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