Chapter 4: Security Policies

- Overview
- The nature of policies
  - What they cover
  - Policy languages
- The nature of mechanisms
  - Types
  - Secure *vs*. precise
- Underlying both
  - Trust

Overview

- Overview
- Policies
- Trust
- Nature of Security Mechanisms
- Policy Expression Languages
- Limits on Secure and Precise Mechanisms

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

Types of Security Policies

- Military (governmental) security policy
  - Policy primarily protecting confidentiality
- Commercial security policy
  - Policy primarily protecting integrity
- Confidentiality policy
  - Policy protecting only confidentiality
- Integrity policy
  - Policy protecting only integrity

Integrity and Transactions

- Begin in consistent state
  - "Consistent" defined by specification
- Perform series of actions (*transaction*)
  - Actions cannot be interrupted
  - If actions complete, system in consistent state
  - If actions do not complete, system reverts to beginning (consistent) state

Trust

Administrator installs patch

1. Trusts patch came from vendor, not tampered with in transit
2. Trusts vendor tested patch thoroughly
3. Trusts vendor's test environment corresponds to local environment
4. Trusts patch is installed correctly

Trust in Formal Verification

- Gives formal mathematical proof that given input *i*, program *P* produces output *o* as specified
- Suppose a security-related program *S* formally verified to work with operating system *O*
- What are the assumptions?

Trust in Formal Methods

1. Proof has no errors
   - Bugs in automated theorem provers
2. Preconditions hold in environment in which *S* is to be used
3. *S* transformed into executable *S′* whose actions follow source code
   – Compiler bugs, linker/loader/library problems
4. Hardware executes *S′* as intended
   – Hardware bugs (Pentium `f00f` bug, for example)

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
- Originator Controlled Access Control (ORCON)
  - originator (creator) of information controls who can access information

Question

- Policy disallows cheating
  - Includes copying homework, with or without permission
- CS class has students do homework on computer
- Anne forgets to read-protect her homework file
- Bill copies it
- Who cheated?
  - Anne, Bill, or both?

Answer Part 1

- Bill cheated
  - Policy forbids copying homework assignment
  - Bill did it
  - System entered unauthorized state (Bill having a copy of Anne's assignment)
- If not explicit in computer security policy, certainly implicit
  - Not credible that a unit of the university allows something that the university as a whole forbids, unless the unit explicitly says so

Answer Part #2

- Anne didn't protect her homework
  - Not required by security policy
- She didn't breach security
- If policy said students had to read-protect homework files, then Anne did breach security
  - She didn't do this

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

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)
```

DTEL

- Basis: access can be constrained by types
- Combines elements of low-level, high-level policy languages
  - Implementation-level constructs express constraints in terms of language types
  - Constructs do not express arguments or inputs to specific system commands

Example

- Goal: users cannot write to system binaries
- Subjects in administrative domain can
  - User must authenticate to enter that domain
- Subjects belong to domains:
  - *d_user* ordinary users
  - *d_admin* administrative users
  - *d_login* for login
  - *d_daemon* system daemons

Types

- Object types:
  - *t_sysbin* executable system files
  - *t_readable* readable files
  - *t_writable* writable files
  - *t_dte* data used by enforcement mechanisms
  - *t_generic* data generated from user processes
- For example, treat these as partitions
  - In practice, files can be readable and writable; ignore this for the example

Domain Representation

- Sequence
  - First component is list of programs that start in the domain
  - Other components describe rights subject in domain has over objects of a type

    ```
    (crwd->t_writable)
    ```

    means subject can create, read, write, and list (search) any object of type t_writable

d_daemon Domain

```
domain d_daemon = (/sbin/init),
      (crwd->t_writable),
      (rd->t_generic, t_readable, t_dte),
      (rxd->t_sysbin),
      (auto->d_login);
```

- Compromising subject in *d_daemon* domain does not enable attacker to alter system files
  - Subjects here have no write access
- When /sbin/init invokes login program, login program transitions into *d_login* domain

d_admin Domain

```
domain d_admin =
   (/usr/bin/sh, /usr/bin/csh, /usr/bin/ksh),
   (crwxd->t_generic),
   (crwxd->t_readable, t_writable, t_dte,
                                        t_sysbin),
   (sigtstp->d_daemon);
```

- *sigtstp* allows subjects to suspend processes in *d_daemon* domain
- Admin users use a standard command interpreter

d_user Domain

```
domain d_user =
      (/usr/bin/sh, /usr/bin/csh, /usr/bin/ksh),
      (crwxd->t_generic),
      (rxd->t_sysbin),
      (crwd->t_writable),
      (rd->t_readable, t_dte);
```

- No auto component as no user commands transition out of it
- Users cannot write to system binaries

d_login Domain

```
domain d_login =
  (/usr/bin/login),
  (crwd->t_writable),
  (rd->t_readable, t_generic, t_dte),
  setauth,
  (exec->d_user, d_admin);
```

- Cannot execute anything except the transition
  - Only /usr/bin/login in this domain
- *setauth* enables subject to change UID
- *exec* access to *d_user*, *d_admin* domains

Set Up

```
initial_domain = d_daemon;
```

– System starts in *d_daemon* domain

```
assign –r t_generic /;
assign –r t_writable /usr/var, /dev, /tmp;
assign –r t_readable /etc;
assign –r –s dte_t /dte;
assign –r –s t_sysbin /sbin, /bin,
                         /usr/bin, /usr/sbin;
```

– These assign initial types to objects
– –r recursively assigns type
– –s binds type to name of object (delete it, recreate it, still of given type)

Add Log Type

- Goal: users can't modify system logs; only subjects in *d_admin*, new *d_log* domains can

```
type t_readable, t_writable, t_sysbin,
                  t_dte, t_generic, t_log;
```

- New type *t_log*

```
domain d_log =
  (/usr/sbin/syslogd),
  (crwd->t_log),
  (rwd->t_writable),
  (rd->t_generic, t_readable);
```

- New domain *d_log*

Fix Domain and Set-Up

```
domain d_daemon =
    (/sbin/init),
    (crwd->t_writable),
    (rxd->t_readable),
    (rd->t_generic, t_dte, t_sysbin),
    (auto->d_login, d_log);
```

- Subject in *d_daemon* can invoke logging process
- Can log, but not execute anything

```
assign -r t_log /usr/var/log;
assign t_writable /usr/var/log/wtmp,
    /usr/var/log/utmp;
```

- Set type of logs

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: X Window System

- UNIX X11 Windowing System
- Access to X11 display controlled by list
  - List says what hosts allowed, disallowed access

```
xhost +groucho -chico
```

- Connections from host groucho allowed
- Connections from host chico not allowed

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
  .ZD4cc0Wr8i2lZKaI..LUOr3
  .0fwo5:hf4e4.8TAqd0V4ubv ?...... ...9b3
  1M4GX01xbGIX0oVuGo1h15z3
  ?:Y9jfa04rdzM1q:eqtlAPgHk
  ?.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 nterfere 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

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 $r \rightarrow R$

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 $p: I_1 \times ... \times I_n \rightarrow R$. A protection mechanism m is a function $m: I_1 \times ... \times I_n \rightarrow R \cup E$ for which, when $i_k \in I_k$, $1 \le k \le n$, either
  - $m(i_1, ..., i_n) = p(i_1, ..., i_n)$ or
  - $m(i_1, ..., i_n) \in E$.
- 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 $p: I_1 \times ... \times I_n \rightarrow R$, it is a function $c: I_1 \times ... \times I_n \rightarrow A$, where $A \subseteq I_1 \times ... \times I_n$
  - *A* is set of inputs available to observer
- Security mechanism is function $m: I_1 \times ... \times I_n \rightarrow R \cup E$
  - *m secure* iff $\exists\, m': A \rightarrow R \cup E$ such that, for all $i_k \in I_k$, $1 \leq k \leq n$, $m(i_1, ..., i_n) = m'(c(i_1, ..., i_n))$
  - *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

Key Points

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