

Lecture 5, April 8

ECS 235B, Foundations of Computer and Information Security
Spring Quarter 2026

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 the *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 the *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 information (or a resource)
- I has the *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); x meets it or not, 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

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

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 a consistent state

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, ORGCON)
 - Originator (creator) of information controls who can access information

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: Ponder

- Security and management policy specification language
- Handles many types of policies
 - Authorization policies
 - Delegation policies
 - Information filtering policies
 - Obligation policies
 - Refrain policies

Entities

- Organized into hierarchical domains
- Network administrators
 - *Domain* is /NetAdmins
 - Subdomain for net admin trainees is
 - /NetAdmins/Trainees
- Routers in LAN
 - Domain is /localnet
 - Subdomain that is a testbed for routers is
 - /localnet/testbed/routers

Authorization Policies

- Allowed actions: netadmins can enable, disable, reconfigure, view configuration of routers

```
inst auth+ switchAdmin {  
    subject /NetAdmins;  
    target /localnetwork/routers;  
    action enable(), disable(), reconfig(), dumpconfig();  
}
```

Authorization Policies

- Disallowed actions: trainees cannot test performance between 8AM and 5PM

```
inst auth- testOps {  
    subject /NetEngineers/trainees;  
    target  /localnetwork/routers;  
    action  testperformance();  
    when    Time.between("0800", "1700");  
}
```

Delegation Policies

- Delegated rights: net admins delegate to net engineers the right to enable, disable, reconfigure routers on the router testbed

```
inst deleg+ (switchAdmin) delegSwitchAdmin {  
    grantee    /NetEngineers;  
    target    /localnetwork/testNetwork/routers;  
    action    enable(), disable(), reconfig();  
    valid    Time.duration(8);  
}
```

Information Filtering Policies

- Control information flow: net admins can dump everything from routers between 8PM and 5AM, and config info anytime

```
inst auth+ switchOpsFilter {  
    subject /NetAdmins;  
    target /localnetwork/routers;  
    action dumpconfig(what)  
        { in partial = "config"; }  
    if (Time.between("2000", "0500")) {  
        in partial = "all"; }  
}
```

Refrain Policies

- Like authorization denial policies, but enforced by the *subjects*: net engineers cannot send test results to net developers while testing in progress

```
inst refrain testSwitchOps {  
    subject    s=/NetEngineers;  
    target     /NetDevelopers;  
    action     sendTestResults();  
    when       s.teststate="in progress"  
}
```

Obligation Policies

- Must take actions when events occur: on 3rd login failure, net security admins will disable account and log event

```
inst oblig loginFailure {  
    on          loginfail(userid, 3);  
    subject    s=/NetAdmins/SecAdmins;  
    target     t=/NetAdmins/users ^ (userid);  
    do         t.disable() -> s.log(userid);  
}
```

Example

- Policy: separation of duty requires 2 different members of Accounting approve check

```
inst auth+ separationOfDuty {  
    subject    s=/Accountants;  
    target     t=checks;  
    action     approve(), issue();  
    when       s.id <> t.issuerid;  
}
```

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

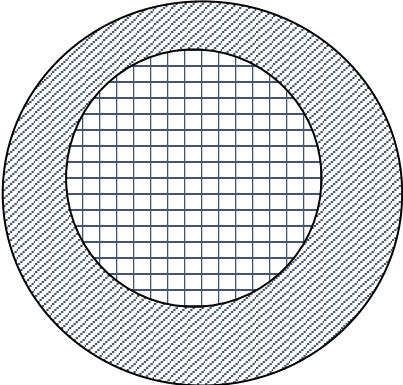
Example: X Window System

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

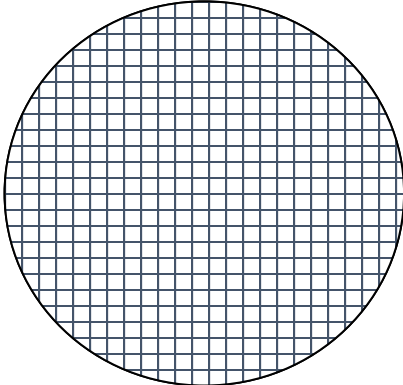
```
xhost +groucho -chico
```

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

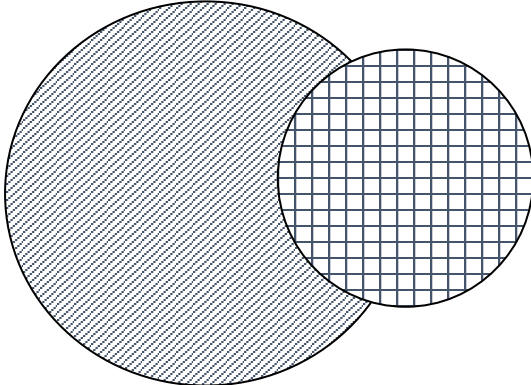
Types of Mechanisms



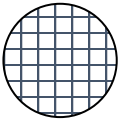
secure



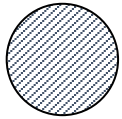
precise



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

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 $p: I_1 \times \dots \times I_n \rightarrow R$. A *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 is *secure* if and only if $\exists m': A \rightarrow R \cup E$ such that,

$$\forall i_k \in I_k, 1 \leq k \leq n, m(i_1, \dots, i_n) = m'(c(i_1, \dots, i_n))$$

- 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 policy c be constant function, and p compute function $T(x)$. Assume $T(x) = 0$. Consider program q , where

```
 $z = p;$   
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 $m(x)$ undefined
- 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

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 are called *security clearance* $L(s)$ for subjects and *security classification* $L(o)$ for objects

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
 - (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
 - $\text{lub}(L) = (\max(A), C)$
 - $\text{glb}(L) = (\min(A), \emptyset)$

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!

Formal Model Definitions

- S subjects, O objects, P rights
 - Defined rights: r read, a write, w read/write, e empty (*not* execute!)
- 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 $H: O \rightarrow \mathbb{P}(O)$
- 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 $\Sigma(R, D, W, z_0)$