IS 2150 / TEL 2810
Introduction to Security

James Joshi

Assistant Professor, SIS

Lecture 3

September 13, 2007

Mathematical Review
Security Policies

Mathematics Review

Propositional logic/calculus

- Atomic, declarative statements (propositions)
  - that can be shown to be either TRUE or FALSE but not both; E.g., “Sky is blue”; “3 is less than 4”
- Propositions can be composed into compound sentences using connectives
  - Negation $\neg p$ (NOT) highest precedence
  - Disjunction $p \vee q$ (OR) second precedence
  - Conjunction $p \wedge q$ (AND) second precedence
  - Implication $p \rightarrow q$ q logical consequence of p
- Exercise: Truth tables?

Propositional logic/calculus

- Contradiction:
  - Formula that is always false : $p \wedge \neg p$
  - What about: $\neg(p \wedge \neg p)$?
- Tautology:
  - Formula that is always True : $p \vee \neg p$
    - What about: $\neg(p \vee \neg p)$?
- Others
  - Exclusive OR: $p \oplus q$; p or q but not both
  - Bi-condition: $p \leftrightarrow q$ [p *if and only if* q (p iff q)]
  - Logical equivalence: $p \Leftrightarrow q$ [p is logically equivalent to q]
- Some exercises...

Some Laws of Logic

- Double negation
- DeMorgan's law
  - $\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$
  - $\neg(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$
- Commutative
  - $(p \vee q) \Leftrightarrow (q \vee p)$
- Associative law
  - $p \vee (q \vee r) \Leftrightarrow (p \vee q) \vee r$
- Distributive law
  - $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$
  - $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$

Predicate/first order logic

- Propositional logic
- Variable, quantifiers, constants and functions
- Consider sentence: *Every directory contains some files*
- Need to capture “every” “some”
  - F(x): x is a file
  - D(y): y is a directory
  - C(x, y): x is a file in directory y

Predicate/first order logic

- Existential quantifiers $\exists$ (There exists)
  - E.g., $\exists$ x is read as There exists x
- Universal quantifiers $\forall$ (For all)
- $\forall y\ D(y) \rightarrow (\exists x\ (F(x) \wedge C(x, y)))$
- read as
  - for every y, *if* y is a directory, then there exists a x such that x *is a file* and *x is in directory y*
- What about $\forall x\ F(x) \rightarrow (\exists y\ (D(y) \wedge C(x, y)))$?

Mathematical Induction

- Proof technique - to prove some mathematical property
  - E.g. want to prove that M(n) holds for all natural numbers
- Base case OR Basis:
  - Prove that M(1) holds
- Induction Hypothesis:
  - Assert that M(n) holds for $n = 1, \ldots, k$
- Induction Step:
  - Prove that if M(k) holds then M(k+1) holds

Mathematical Induction

- Exercise: prove that sum of first n natural numbers is
  - S(n): $1 + \ldots + n = n(n + 1)/2$
  - S(n): $1^2 + \ldots + n^2 = n(n + 1)(2n + 1)/6$

Lattice

- Sets
  - Collection of unique elements
  - Let S, T be sets
    - Cartesian product: S x T = {(a, b) | a ∈ A, b ∈ B}
    - A set of order pairs
- Binary relation *R* from S to T is a subset of S x T
- Binary relation *R* on S is a subset of S x S
- If (a, b) ∈ *R* we write a*R*b
  - Example:
    - *R* is “less than equal to” (≤)
    - For S = {1, 2, 3}
      - Example of *R* on S is {(1, 1), (1, 2), (1, 3), ????)
  - (1, 2) ∈ *R* is another way of writing 1 ≤ 2

Lattice

- Properties of relations
  - Reflexive:
    - if $a\mathcal{R}a$ for all $a \in S$
  - Anti-symmetric:
    - if $a\mathcal{R}b$ and $b\mathcal{R}a$ implies a = b for all $a, b \in S$
  - Transitive:
    - if $a\mathcal{R}b$ and $b\mathcal{R}c$ imply that $a\mathcal{R}c$ for all $a, b, c \in S$
  - Which properties hold for “less than equal to” ($\leq$)?
  - Draw the Hasse diagram
    - Captures all the relations

Lattice

- Total ordering:
  - when the relation orders all elements
  - E.g., "less than equal to" ($\leq$) on natural numbers
- Partial ordering (poset):
  - the relation orders only some elements not all
  - E.g. "less than equal to" ($\leq$) on complex numbers; Consider (2 + 4i) and (3 + 2i)

Lattice

- Upper bound $(u, a, b \in S)$
  - u is an upper bound of a and b means aRu and bRu
  - Least upper bound : $lub(a, b)$ *closest upper bound*
- Lower bound $(l, a, b \in S)$
  - l is a lower bound of a and b means lRa and lRb
  - Greatest lower bound : $glb(a, b)$ *closest lower bound*

Lattice

- A lattice is the combination of a set of elements *S* and a relation *R* meeting the following criteria
  - R is reflexive, antisymmetric, and transitive on the elements of S
  - For every $s, t \in$ S, there exists a greatest lower bound
  - For every $s, t \in$ S, there exists a lowest upper bound
- Some examples
  - S = {1, 2, 3} and R = $\leq$?
  - S = {2+4i; 1+2i; 3+2i, 3+4i} and R = $\leq$?

Overview of Lattice Based Models

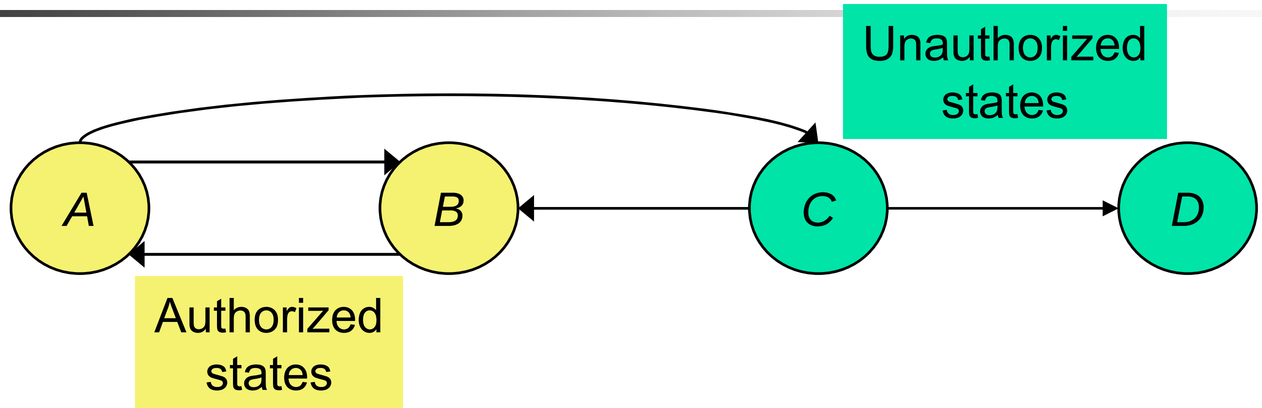
- Confidentiality
  - Bell LaPadula Model
    - First rigorously developed model for high assurance - for military
    - Objects are classified
    - Objects may belong to Compartments
    - Subjects are given clearance
    - Classification/clearance levels form a lattice
    - Two rules
      - No read-up
      - No write-down

Security Policies

Security Policy

- Defines what it means for a system to be secure
- Formally: Partitions a system into
  - Set of secure (authorized) states
  - Set of non-secure (unauthorized) states
- Secure system is one that
  - Starts in authorized state
  - Cannot enter unauthorized state

Secure System - Example



- Is this Finite State Machine Secure?
  - *A* is start state ?
  - *B* is start state ?
  - *C* is start state ?
  - How can this be made secure if not?
  - *Suppose A*, *B*, and *C* are authorized states ?

Additional Definitions:

- Security breach: system enters an unauthorized state
- Let *X* be a set of entities, *I* be information.
  - *I* has **confidentiality** with respect to *X* if no member of *X* can obtain information on *I*
  - *I* has **integrity** with respect to *X* if all members of *X* trust *I*
    - Trust *I*, its conveyance and storage (data integrity)
    - *I* maybe origin information or an identity (authentication)
    - *I* is a resource – its integrity implies it functions as it should (assurance)
  - *I* has **availability** with respect to *X* if all members of *X* can access *I*
    - Time limits (quality of service)

Confidentiality Policy

- Also known as *information flow*
  - Transfer of rights
  - Transfer of information without transfer of rights
  - Temporal context
- Model often depends on trust
  - Parts of system where information *could* flow
  - Trusted entity must participate to enable flow
- Highly developed in Military/Government

Integrity Policy

- Defines how information can be altered
  - Entities allowed to alter data
  - Conditions under which data can be altered
  - Limits to change of data
- Examples:
  - Purchase over $1000 requires signature
  - Check over $10,000 must be approved by one person and cashed by another
    - *Separation of duties :* for preventing fraud
- Highly developed in commercial world

Trust

- Theories and mechanisms rest on some trust assumptions
- 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

- Formal verification provides a 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

Security Mechanism

- Policy describes what is allowed
- Mechanism
  - Is an entity/procedure that enforces (part of) policy
- Example Policy: Students should not copy homework
  - Mechanism: Disallow access to files owned by other users

Security Model

- A model that represents a particular policy or set of policies
  - Abstracts details relevant to analysis
  - Focus on specific characteristics of policies
    - E.g., Multilevel security focuses on information flow control

Security policies

- Military security policy
  - Focuses on confidentiality
- Commercial security policy
  - Primarily Integrity
  - Transaction-oriented
    - 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

Access Control

- Discretionary Access Control (DAC)
  - Owner determines access rights
  - Typically *identity-based access control*: Owner specifies other users who have access
- Mandatory Access Control (MAC)
  - Rules specify granting of access
  - Also called *rule-based access control*

Access Control

- Originator Controlled Access Control (ORCON)
  - Originator controls access
  - *Originator need not be owner!*
- Role Based Access Control (RBAC)
  - Identity governed by role user assumes

Back to ..

Access Control Matrix

Protection System

- State of a system
  - Current values of
    - memory locations, registers, secondary storage, etc.
    - other system components
- Protection state (P)
  - A system state that is considered secure
- A protection system
  - Captures the conditions for state transition
  - Consists of two parts:
    - A set of generic rights
    - A set of commands

Protection System

- Subject (S: set of all subjects)
  - Active entities that carry out an action/operation on other entities; Eg.: users, processes, agents, etc.
- Object (O: set of all objects)
  - Eg.:Processes, files, devices
- Right (R: set of all rights)
  - An action/operation that a subject is allowed/disallowed on objects
  - Access Matrix A: $a[s, o] \subseteq R$
- Set of Protection States: (S, O, A)

State Transitions

- Let initial state $X_0 = (S_0, O_0, A_0)$
- Notation
  - $X_i \vdash_{\tau_{i+1}} X_{i+1}$: upon transition τ_{i+1}, the system moves from state X_i to X_{i+1}
  - $X \vdash^* Y$: the system moves from state X to Y after a set of transitions
  - $X_i \vdash_{c_{i+1}(p_{i+1,1}, p_{i+1,2}, \ldots, p_{i+1,m})} X_{i+1}$: state transition upon a command
- For every command there is a sequence of state transition operations

Primitive commands (Graham-Denning)

Create subject *s*	Creates new row, column in ACM;
Create object *o*	Creates new column in ACM
Destroy subject *s*	Deletes row, column from ACM;
Destroy object *o*	Deletes column from ACM
Read access right of s on o	Copy a[s, o] to x
Delete access right r of s on o	Removes *r* right from subject *s* over object *o*
Grant access right r of s on o	Adds *r* right for subject *s* over object *o*
Transfer access right r or r* to s on o	Adds *r* right for subject *s* over object *o*

Primitive commands (HRU)

Create subject *s*	Creates new row, column in ACM;
Create object *o*	Creates new column in ACM
Enter *r* into *a*[*s*, *o*]	Adds *r* right for subject *s* over object *o*
Delete *r* from *a*[*s*, *o*]	Removes *r* right from subject *s* over object *o*
Destroy subject *s*	Deletes row, column from ACM;
Destroy object *o*	Deletes column from ACM

System commands

- [Unix] process *p* creates file *f* with owner *read* and *write* (*r*, *w*) will be represented by the following:

  Command *create_file*(*p*, *f*)
    Create object *f*
    Enter *own* into *a*[*p*,*f*]
    Enter *r* into *a*[*p*,*f*]
    Enter *w* into *a*[*p*,*f*]
  End

System commands

- Process p creates a new process q

Command $spawn\_process(p, q)$

Create object q;

Enter own into $a[p,q]$

Enter r into $a[p,q]$

Enter w into $a[p,q]$

Enter r into $a[q,r]$

Enter w into $a[q,r]$

End

System commands

- Defined commands can be used to update ACM

  Command *make_owner*(*p*, *f*)

  Enter *own* into *a*[*p*,*f*]

  End

- Mono-operational: the command invokes only one primitive

Conditional Commands

- Mono-operational + mono-conditional

Command *grant_read_file*(*p, f, q*)

If *own* in *a*[*p,f*]

Then

Enter *r* into *a*[*q,f*]

End

Conditional Commands

- Mono-operational + biconditional

Command *grant_read_file*(*p, f, q*)

If *r* in *a*[*p,f*] and *c* in *a*[*p,f*]

Then

Enter *r* into *a*[*q,f*]

End

- Why not “OR”??

Fundamental questions

- How can we determine that a system is secure?
  - Need to define what we mean by a system being "secure"
- Is there a generic algorithm that allows us to determine whether a computer system is secure?

- We will wait till next time .....