Access Control and Security Models

CSC 348·648

WAKE FOREST
UNIVERSITY
Department of Computer Science

Spring 2013

Role-Based Access Control

- **Role** is a collection of jobs
  - Each role r is authorized to perform transactions
  - The set of authorized transactions for r is $trans(r)$

  Assume you are playing dota2 with your favorite system administrator, what is an example of role?

- **Active role of** s, denoted $actr(s)$ is the role that s is performing
- The **predicate** $canexec(s, t)$ is true iff s can execute t

RBAC Axioms

- S is set of subjects and T set of transactions, **the rule of role assignment** is $(\forall s \in S)(\forall t \in T)[canexec(s,t) \rightarrow actr(s) \neq \emptyset]$

  What does the axiom state? What is bound to a role?

- S is set of subjects, **the rule of role authorization** is $(\forall s \in S)[actr(s) \subseteq authr(s)]$

  What does the axiom state? Why is it important?

- S is set of subjects and T set of transactions, **the rule of transaction authorization** is $(\forall s \in S)(\forall t \in T)[canexec(s,t) \rightarrow t \in trans(actr(s))]$

  What does the axiom state?

RBAC Mutual Exclusion

- RBAC can model the *separation of duty*

  What is separation of duty? Why is it important?

  - Prevents subjects in certain roles from entering other roles
  - If roles r_1 and r_2 are bound to separation of duty

$$(\forall s \in S)[r_1 \in authr(s) \rightarrow r_2 \notin authr(s)]$$

- Let r be a role and s be a subject such that $r \in auth(s)$. Then the mutually exclusive authorization predicate, $meauth(r)$, is the set of roles that s cannot assume because of separation of duty

$$(\forall r_1, r_2 \in R)[r_2 \in meauth(r_1) \rightarrow [(\forall s \in S)[r_1 \in authr(s) \rightarrow r_2 \notin authr(s)]]]$$

RBAC in Practice

Is RBAC used in computer systems?

Design Principles of Access Control Mechanisms

Design principles that apply to protection mechanisms, explained by Saltzer and Schroeder in "The Protection of Information in Computer Systems."

- **Economy of mechanism** - Keep design simple and small.
- **Fail-safe defaults** - Base decisions on permission, not exclusion.
- **Complete mediation** - Access must be checked for authority.
- **Open design** - Design should not be secret.
- **Separation of privilege** - Protection mechanism that requires two keys is better than one that allows access based on a single key.
- **Least privilege** - Every program and every user of the system should operate using the least set of privileges necessary to complete the job.
- **Least common mechanism** - Minimize the amount of mechanism common to more than one user and depended on by all users.
- **Psychological acceptability** - Ease of use (*otherwise it won't be*).

Enforcement and Policy

- Up to this point we have been discussing access control mechanisms
  - ACM, ACL, C-list, rings, digital french-fries, and RBAC
- Now let's consider what we would like to enforce with the mechanism
  - Privacy, integrity, etc...

Foundational Results

- *Given a computer system how do we know if it is secure?*
  - Can an algorithm determine if a system is secure
  - Not identifying where it may be (un)secure, just if...
- What do we mean by *secure*
  - Let R be the set of generic (primitive) rights of the system
  - **Definition:** When a generic right r is added to an element of the access control matrix not already containing r, that right is said to be *leaked*
  - **Definition:** If a system can never leak right r, the system is called safe with respect to the right r. If the system can leak right r, the system is called unsafe with respect with the right r

Formal Specification

- Given the following
  - Initial state $X_0 = (S_0, O_0, A_0)$
  - Set of primitive commands c
  - r is not in A_0
- Can we reach a state X_n where
  - $\exists s, o$ such that $A_n[s, o]$ includes right r not in $A_0[s, o]$
  - if the answer is yes, then the system is **not** safe
  - if the answer is no, then the system is safe
- Formally defined by Harrison, Ruzzo, and Ullman
  - Safety decidability model, also called the HRU model

Safety and Security

- Safety refers to the abstract model
- Security refers to the actual implementation
- Secure system corresponds to a model safe with respect to rights
- A model safe with respect to rights doesn't ensure it's secure
- *Back to the original question: Does an algorithm exist for determining whether a given protection system with initial state s_0 is safe with respect to a generic right r?*

  Is there?

Is it Safe?

- Given (S, O, A) is there a sequence of commands that will leak r?
- A simple approach is to
  1. Evaluate commands
  2. Develop different command sequences and test...

  How many sequences should you try? Upper bound?

Mono-Operational

- **Theorem:** There exists an algorithm that will determine whether a given mono-operational protection system with initial state s_0 is safe with respect to a generic right r
  - Mono-operational indicates commands have a single primitive
- **Proof sketch:** Each command is identified by the primitive operation it invokes. Consider minimal sequence of commands needed to leak r from the system with initial state s_0. We can show that the length of this sequence is bounded. Thus, we can enumerate all possible states and determine if safe.
  - It may be computationally infeasible, but it is computable

Generic Rights

- **Theorem:** It is undecidable whether a given state of a given protection system is safe for a given generic right
- **Proof sketch:** We show that an arbitrary Turing machine can be reduced to the safety problem, with the Turing machine entering a final state corresponding to the leaking of a given generic right. Then if the safety problem is decidable, we can determine when the Turing machine halts. Since the halting problem is undecidable, the safety problem cannot be undecidable either.

Implications and Questions

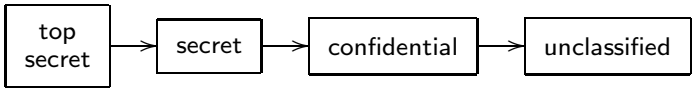
- Safety decidability of some models
  - Decidable given restrictions, but not in a generic application
    Are the restrictions applicable?
- Safety only works if maximum rights known in advance
  - Specify all rights someone can obtain, not just what they have
- Two key questions
  - *Given a particular system with specific rules for transformation, can we show the safety question is decidable?*
  - *What are the weakest restrictions that will make the safety question decidable in that system?*

Types of Access Control

- Discretionary Access Control (DAC)
  - Users (object owners) decide permission assignments, *"a subject with a certain access permission is capable of passing permissions"*
  - An example is the *traditional* Unix permissions
- Mandatory Access Control (MAC) *which is an unfortunate name*
  - Expressed in terms of labels attached to subjects and objects
  - System administrator decides permission assignments
  - SELinux and Windows Vista are examples

Military Security Levels

- Consider four security levels, want to answer questions like...



  - *Given two objects at different security levels, what is the minimal security level a subject must be to read both?*
  - *Given two subjects at different security levels, what is the maximal security level an object can have and still be read?*

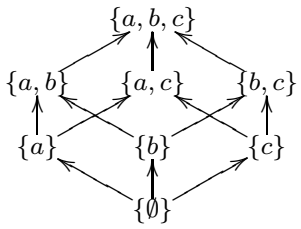
What are the levels?

- Often there are multiple attributes to consider
  - Determine the *partial ordering* of security attributes

More attributes makes the questions more difficult, why?

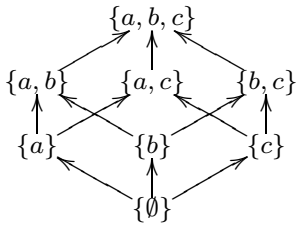
Lattices

- A lattice $(L, \star)$ consists of a set L and a partial ordering $\star$, so that for every two elements $a, b, \in L$ there exists a *system low* $u \in L$ and a *system high* $g \in L$
  - *A partial ordering occurs when a relation orders some, but not all, the elements of a set (often called a poset)*
- Consider the set $L = (\mathcal{P}(\{a, b, c\}), \subseteq)$



  - In this example, the relation $(\star)$ is $\subset$
  - The *system low* is $\emptyset$, while the *system high* is $\{a, b, c\}$

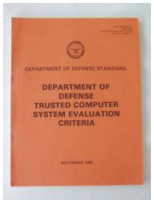
- An arrow is drawn between A and B, $A, B \in \mathcal{P}(\{a, b, c\})$
  - Iff $A \subseteq B$ and $A \neq B$
  - A is a subset of B iff there is an arrow from A to B



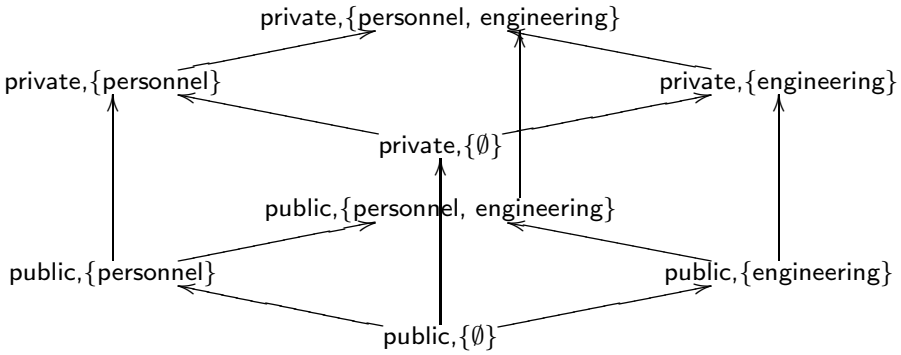
- Some important attributes of lattices
  - Not all pairs need to be directly comparable, for example $\{a\}$ and $\{c\}$
  - However every pair of elements should have a lower bound and upper bound, for example $\emptyset$ and $\{a, b, c\}$ respectively

What does this have to do with security?

Orange Book



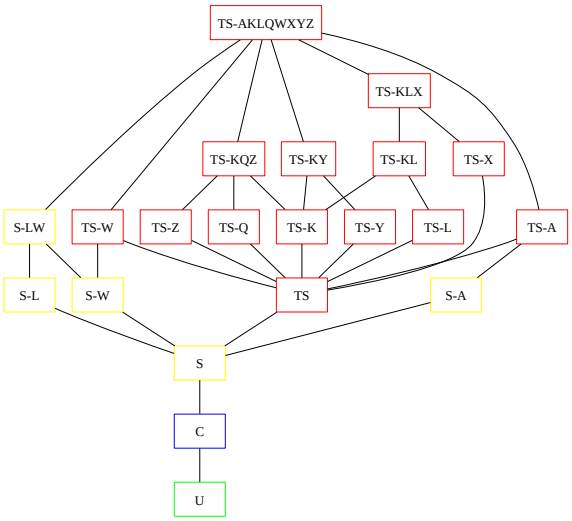
- Consider the following *orange book* example
  - Let H be a set of classifications with hierarchical ordering $\leq$
  - Let C be the set of categories (divisions, departments, ...)
- A security label is a pair (h, c), where $h \in H$ and $c \subset C$
  - The partial ordering is $\leq$ of security labels is defined by $(h_1, c_1) \leq (h_2, c_2)$ iff $h_1 \leq h_2$ and $c_1 \subseteq c_2$
- For example, assume...
  - Two hierarchical levels *public* and *private*
  - Two categories *personnel* and *engineering*



- The lattice can be used to (dis)allow information flow

Is this mechanism or policy? What are we protecting?

Smith's Lattice



- Military levels (TS>S>C>U) and categories {A, K, L, Q, W, X, Y, Z}
- Has 21 out of $4 \times 2^8 = 1024$ possible labels

Bell-LaPadula

- A formal description of allowable paths of information
  - Developed in 1973 by US Air Force, who were concerned about time-share systems

    What is the security question?

- Assume you have the following
  - S is a set of subjects
  - O is a set of objects
- Elements $s \in S$ and $o \in O$ has a fixed security class $c(s)$ and $c(o)$
  - Security classes are ordered by some relation
  - *Kinda smells like a lattice...*

Two Important Information Flow Properties

- **Simple Security Property** - a subject s may have read access to any object o if $c(s) \geq c(o)$
  - *"no process may read data at a higher level"*
  - This is also known as **no read up**
- $\star$-**Property** - a subject s can write to an object o if $c(s) \leq s(o)$

  Tahw? I can write to objects that have a higher security class?
  - *"no process may write data to a lower level"*
  - This is also known as **no write-down** (prevent info leaks)
  - An alternate definition, a subject s who has read access to an object o may have write access to an object p if $c(o) \leq c(p)$

Example Use

- *What does the $\star$-property actually mean?*
  - A person who obtains information at one level may pass that information to only people at a level **no** lower than the level of the information
- Can help defend against malicious code in a computer system
  - User could trick admin to execute malware (`ls` attack)
  - Malware (with admin's permissions) copies secret information
  - Information is made available to any user
- Bell-LaPadula prevents such a situation, so it always works right...

Tranquility

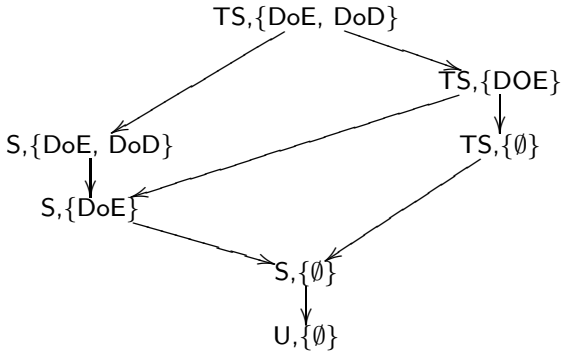
- Bell-LaPadula was very popular
  - Provides a simple set of rules and proves secrecy
- Consider a computer system that has one admin
  - Assume two Bell-LaPadula properties are followed
  - Any user can ask the admin to reclassify an object

Does this violate the two properties? Secrecy maintained?

- Must introduce the **tranquility property**
  - **Strong tranquility** - security labels never change during operation
  - **Weak tranquility** - security labels never change in such a way that violates the security policy

Another Example

- Assume: Top-Secret, Secret, Classified, and Unclassified, and two departments DoE and DoD
  - $c(x) \leq c(y)$ iff $l(x) \leq l(y)$ and $d(x) \subseteq d(y)$



Can a person with (TS,{∅}) read a document (S,{DoE})

Why Have Weak Tranquility

- Often necessary to have *principle of least privilege*
  - User login as unclassified even if they have top secret clearance
  - As the user performs tasks, the level is upgraded
  - This is a *high water mark* principal

Does this break the Bell-LaPadula model?

Bell-LaPadula and ACM

- *What is the relationship between Bell-LaPadula and ACM?*
  - Bell-LaPadula can guide the addition of rights within ACM

Have we seen similar rules for access control?

Biba

- Bell-LaPadula model concerns secrecy
  - Identifies paths that could lead to inappropriate disclosure
- Biba model concerns **integrity** (information contamination)
  - Therefore it is the dual of Bell-LaPadula

    What?
- Assume subjects and objects have *integrity levels*

Two Important Integrity Properties

- **Simple Integrity Property** - a subject s can modify object o if $I(s) \geq I(o)$
  - *"no process can change a document at a higher level"*
  - This is a **no write-up** policy
- **Integrity $\star$-Property** - a subject s has read access to object o with $I(o)$, s can have write access to object p if $I(o) \geq I(p)$
  - Captures the idea that *"an untrusted subject who has write access to an object reduces the integrity of the object"*

Another Take on Biba

- Consider $I(\text{priests}) > I(\text{monks}) > I(\text{congregation})$
  - A monk may write a book that can be read by the congregation, but not write a book to be read by a priest
    What Biba rule is this?
  - A monk may read a book written by the high priest, but may not read a book written by a lowly person in the congregation...
    What Biba rule is this?

LOMAC

- An example Biba-type system is LOMAC, an extension for Linux
  - Attempts to deal with malicious code arriving over the network
- System provides two levels, high and low integrity
  - System files have high integrity
  - Network has low integrity (*that's insulting*)
- As soon as a process receives traffic from the network it is moved to low integrity

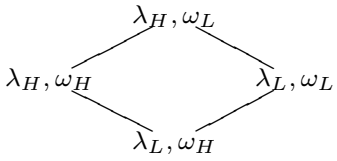
BLP + Biba = BFF

- Often a system requires confidentiality and integrity
  - The system would apply BLP and Biba for a set of labels
  - If one set of labels, there are conflicting constraints
    Tahw?

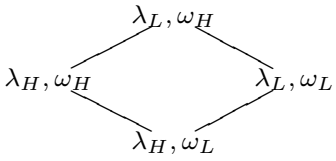
- Better if items have BLP and Biba labels
  - Let $\{\lambda_1, \lambda_2, ..., \lambda_n\}$ be confidentiality labels and $\{\omega_1, \omega_2, ..., \omega_n\}$ be integrity lablels
  - Lattices have high confidentiality and integrity at the top, rules are

    Subject s can read from o iff $\lambda(s) \geq \lambda(o)$ and $\omega(s) \leq \omega(o)$
    Subject s can write to o iff $\lambda(s) \leq \lambda(o)$ and $\omega(s) \geq \omega(o)$
- Simultaneous BLP and Biba application, information flows are opposite: upward for confidentiality and downward for integrity

- For example, $\Lambda = \{\lambda_H, \lambda_L\}, \lambda_H \geq \lambda_L$ and $\Omega = \{\omega_H, \omega_L\}, \omega_H \geq \omega_L$



equivalent BLP
(no read up and no write down)



equivalent Biba
(no write up and no read down)

- Consider the equivalent BLP lattice and a subject with $\{\lambda_H, \omega_H\}$
  - Cannot read objects with $\{\lambda_H, \omega_L\}$, because of the ω
  - Cannot write objects with $\{\lambda_L, \omega_H\}$, because of the λ

Clark-Wilson

- An integrity model developed for the *commercial sector*
  - Integrity is important, prevents fraud and error
  - Lattice model is not sufficient to model integrity
  - Based on idea of *double-entry* accounting (transactions are important, not classifications)
- In a commercial environment, we would want to
  - Make certain data is always in *constant-state*
  - Use only *well-formed-transactions*, a series of operations that transforms the system from one constant state to another
  - These are the ideals, need rules to enforce...

Data Items and Procedures

- Two types of data items
  - **Constrained Data Items** (CDI) have integrity protected
  - **Unconstrained data Items** (UDI) integrity not covered
- Two types of procedures
  - **Transformation Procedure** (TP) procedures allowed to modify CDIs, or take arbitrary user input and create new CDIs, designed to take the system from one valid state to another
  - **Integrity Verification Procedure** (IVP) make certain CDI conform to integrity constraints
- For example, consider a bank
  - Balances are CDI's
  - Check that the accounts are balanced is an IVP
  - Making deposits and withdrawals from accounts are TP

Certification and Enforcement

- Clark-Wilson certification rules
  - **CR1** - When an IVP is executed must ensure all CDI's are valid
  - **CR2** - For associated sets of CDI's, a TP must transform those CDI's for one valid state to another
    - Defines relation *certified*, associates CDIs with particular TP
    - For example in banking: TP balance, CDIs accounts
- Enforcement rules make certain a TP is certified to work on a CDI
  - **ER1** - System must maintain certified relations and ensure only TP's certified to run on a CDI change that CDI
  - **ER2** - System must associate a user with each TP and set of CDI's, the TP may access the CDI on behalf of the user, if the user is not associated with the TP and CDI, then disallow

Want More? No Problem

- **CR3** - Allowed relationships must meet the requirement of *separation of duty*
- **ER3** - System must authenticate each user attempting a TP
  - Type of authentication is undefined...
  - Authentication is not required before use of the system, just before manipulation of CDI's
- **CR4** - All TP's must append enough information to reconstruct the operation
  - Auditors need to be able to determine what happened
- **ER4** - Only the certifier of a TP may change the list of entities associated with that TP (*enforces separation of duty with respect to certified and allowed relations*)

- **CR5** - Any TP that takes a UDI as input may perform only valid transactions for all possible values of the UDI, the transformation either accepts (convert to CDI) or rejects
  - In a bank, numbers entered at keyboard are UDIs, so cannot be input to TPs. TPs must validate numbers (to make them a CDI) before using them; if validation fails, TP rejects UDI

Clark-Wilson

- Provides the following
  - Subjects have to be identified via certain programs
  - Objects only manipulated via certain programs
  - Subjects can only execute certain programs
  - Audit logs maintained
  - System has to be certified to operate properly
- More complex than Biba, most systems implement a form of Clark-Wilson given its flexibility