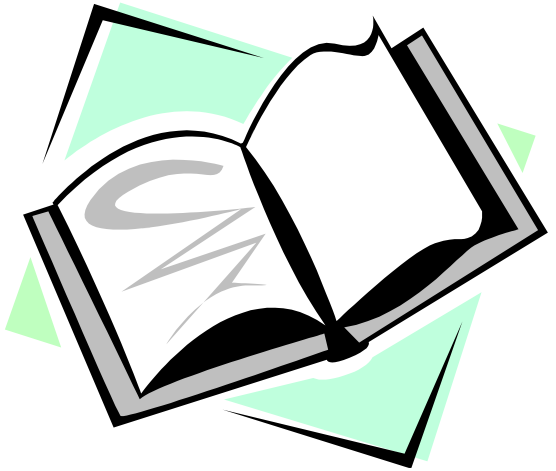
DATA SECURITY AND PRIVACY

Non-Interference, Non-Deducability, and Role Based Access Control

PURDUE UNIVERSITY®

Optional Readings for This Lecture

- Security Policies and Security Models. J.A.Goguen and J.Meseguer. Oakland'1982
- Non-deducibility is from the paper "A Model of Information" by David Sutherland



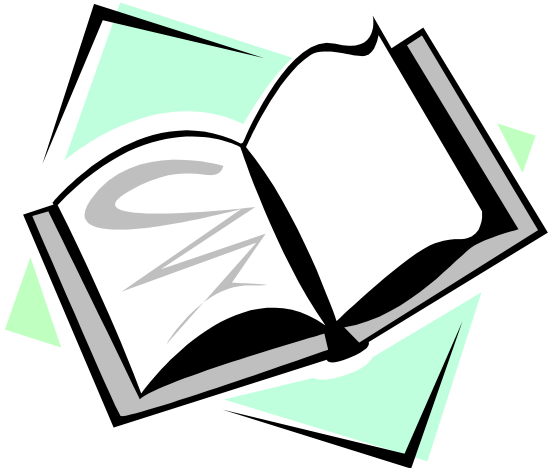
PURDUE UNIVERSITY

Outline

- Non-Interference Model
- Non-deducibility
- The RBAC96 Family of Role Based Access Control Models
- The NIST RBAC Standard and Our Critique
- Attribute Based Access Control and XACML

Source for Non-Interference

- Security Policies and Security Models.
  J.A.Goguen and J.Meseguer.
  Oakland'1982



PURDUE UNIVERSITY

Motivations

- Multi-level security is about information flow
  - Information in high level objects should not flow into low-level subjects
- The BLP model describes access control mechanisms that prevent illegal information flow, but not the meaning of no illegal information flow
  - BLP describes “how”, not “what” for information flow protection
    - Analogy: define secure encryption by giving a particular encryption algorithm and say this is secure encryption
  - As a result, BLP does not prevent information flow through covert channels
  - Also, it doesn’t say whether other mechanisms can be used do information flow protection

Non-interference in Programs

Consider the following functions, is there information flow between x and output of the functions? In non-interference, this means whether changing the value of x could affect the output.

```
add(int x, int z) {
  return x+z;
}

check_pw(char *s) {
    char *x;
    return strcmp(x,s);
}
```

```
f(int x, int z) {
  if x>0 return z+z;
  else return 2*z;
}

g(int x, int z){
  return x*z/x;
}
```

PURDUE UNIVERSITY

Deterministic Non-Interference in Programs

- A set X of inputs is non-interfering with a set Y of outputs if and only if
  - No matter what values X take, the outputs Y remain the same
    - When one changes only values of inputs in X, the output remain unchanged
    - Observing only Y, one learns nothing about any input in X.
  - More formally, let Y=f(X,Z), where f is a deterministic function, and X,Z represents two sets of inputs, ***X is non-interfering with Y***
    - iff $\forall Z_0 \exists Y_0 \; \forall X_0 \; f(X_0, Z_0) = Y_0$
      or equivalently, $\forall Z_0 \; \forall X_0 \; \forall X_1 \; f(X_0, Z_0) = f(X_1, Z_0)$
  - ***X interferes with Y*** iff. $\exists Z_0 \; \exists X_0 \; \exists X_1 \; f(X_0, Z_0) \neq f(X_1, Z_0)$
- For randomized programs, non-interference is harder to define, and we do not cover it in this course

PURDUE UNIVERSITY

More on Non-interference Properties

- Two classes of techniques to ensure that security properties are satisfied by programs
  - Monitor execution of a program and deny illegal actions or terminate the program if illegal action is detected.
    - Can enforce BLP property.
    - Cannot enforce non-interference.
      - Why? Because non-interference is not defined on one execution of a program; it is a property on a program's behaviors on different inputs.
  - Statically verifying that certain non-interference relation holds by analyzing the program
    - Can be used only with access to source code

The Non-Interference Model in the Goguen-Meseguer paper

- A state-transition model, where state changes occur by subjects executing commands
  - S: set of states
  - U: set of subjects
  - SC: set of state commands
  - Out: set of all possible outputs
  - do: $S \times U \times SC \rightarrow S$
    - do(s,u,c)=s' means that at state s, when u performs command c, the resulting state is s'
  - out: $S \times U \rightarrow Out$
    - out(s,u) gives the output that u sees at state s
  - $s_0 \in S$ initial state

Model focuses on interfaces (inputs/outputs) of a system, rather than internal aspects (e.g., subjects, objects, and accesses)

PURDUE UNIVERSITY

Security Policies in the Non-interefence Model

- A security policy is a set of noninterference assertions
- Definition of noninterference: Given two group of users G and G', we say G does not interfere with G' if for any sequence of commands w,
  - View_G'(w) = View_G'(P_G(w))
    - P_G(w) is w with commands initiated by users in G removed.
    - ***No matter what users in G do, users in G' will observe the same.***
- Implicit assumptions:
  - Initial state of the system does not contain any sensitive information
  - Information comes into the system by commands
  - Only way to get information is through outputs

Comparisons of BLP & the GM Noninterference Model

- Differences in model
  - BLP models internals of a system (e.g., objects)
  - GM models the interface (input & output)
- Differences in formulating security policies
  - BLP specifies access control requirement, noninterference specifies information flow goal
- Noninterference could address covert channels concerns
  - Provided that one defines observable behavior to include those in covert channels; doesn't make stopping covert channel easier
- Under noninterference, a low user is allowed to copy one high-level file to another high-level file
  - In general not allowed by BLP

Evaluation of The Non-Interference Policy

- The notion of noninterference is elegant and natural
  - Focuses on policy objective, rather than mechanism, such as BLP
  - Could be useful in other settings
- Mostly concerned with deterministic systems
  - For randomized or otherwise non-deterministic systems, definition is more complicated
- May be too restrictive
  - e.g., consider encrypt and then communicate

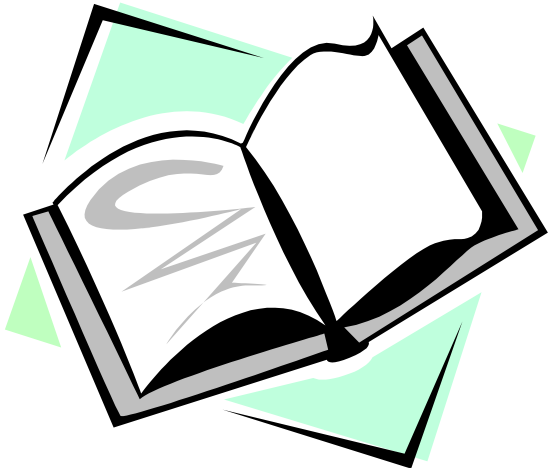
Outline

- Non-Interference Model
- Non-deducibility
- The RBAC96 Family of Role Based Access Control Models
- The NIST RBAC Standard and Our Critique
- Attribute Based Access Control and XACML

PURDUE UNIVERSITY

Source for Non-Deducibility

- “A Model of Information” by David Sutherland Oakland’1986



PURDUE UNIVERSITY

Non-deducibility

- Attempt to define information flow in non-deterministic as well as deterministic systems
- Intuition: there is no information flow between X and Y, iff., when observing only Y, one can never eliminate any value from the domain in X as a possible value
- Definition: let Y=f(X,Z), where f is not necessarily deterministic, there is information flow between X and Y in the non-deducibility sense iff.

  $\exists Y_0 \in \{ f(X,Z) \}\ \exists X_0$ **s.t.** $Y_0 \notin \{ f(X_0, Z) \}$

  - When one observes the value of Y is Y_0, one learns that $X \neq X_0$.
  - There ***is no information flow*** between X and Y in the non-deducibility sense when

    $\forall Y_0 \in \{ f(X,Z) \}\ \forall X_0\ \exists Z_0$ s.t. $Y_0 \in \{ f(X_0, Z_0) \}$

PURDUE UNIVERSITY

Examples: Output is z

High int x = …;
High int y = …;
Low int z;
if x>0 z = y+y;
else z = 3*y;

- x interferes with z
- y interferes with z
- x and z are not non-deducible secure
- y and z are not non-deducible secure

High int x = …;
High int y = …;
Low int z;
if x>0 z= y+y;
else z=2*y;

- x does not interfere with z
- y interferes with z
- x and z are non-deducible secure
- y and z are not non-deducible secure

PURDUE UNIVERSITY®

Examples

```
High int x = …;  // in {0,1,…,p-1}
High int y = …;  // in {0,1,…,p-1}
Low int z1 = (x + y) % p;  // p is large prime
Low int z2 = (x – y) % p;
```

- x interferes with z1
- x interferes with z2
- x and z1 are non-deducible secure
- x and {z1,z2} are not non-deducible secure

```
High char * x = …;
Low char * entered_pw = …;
Low boolean z;
z = strcmp(entered_pw,x);
```

- x interferes with z
- x and {z, entered_pw} are not non-deducible secure

PURDUE UNIVERSITY®

An Example Illustrating that Non-deducibility is Too Weak

- A high user and a low user
  - the high user can write to a file
    - one letter at a time
  - the low user can try to read the n'th character in a file
    - if file is shorter than n, or if the the n'th character is blank, returns a random letter
    - otherwise, with 99.9% probability return the letter, and with 0.1% probalility return a random letter
- The system is nondeducible secure
- The system is intuitively insecure
- Non-deducibility can often be too weak. It deals with possibilistic inference, not probabilistic inference

PURDUE UNIVERSITY

Relationships Between Nondeducibility & Noninterference

- For deterministic systems with just one high input variable (and possibly many other low input variables) and one low output, a system is noninterference secure if and only if it is nondeducibility secure.

- For deterministic systems with more than one high input vars, non-interference is stronger than non-deducibility

Proof.

- Theorem: For deterministic programs with just one high input variable x, let Z be the set of all low variables, x does not interfere with the set Z if and only if x and Z are nondeducible secure.

- Proof. If x does not interfere with Z, no matter what values x takes, the variables in Z are uniquely determined by inputs in Z. Observing values in Z cannot eliminate any value for x.

- If x interferes with Z, then there exist $x_1 \neq x_2$ and $Z_2 \neq Z_1$ such that $Z=Z_1$ when $x=x_1$ and $Z=Z_2 \neq Z_1$ when $x=x_2$. Observing $Z=Z_2$, one knows $x \neq x_1$, making x and X not nondeduciable secure. This is because as x is the only high var and the system is deterministic, when fixing input variables in Z to values in Z_2, the output variables are fixed as well.

PURDUE UNIVERSITY

Outline

- Non-Interference Model
- Non-deducibility
- The RBAC96 Family of Role Based Access Control Models
- The NIST RBAC Standard and Our Critique
- Attribute Based Access Control and XACML

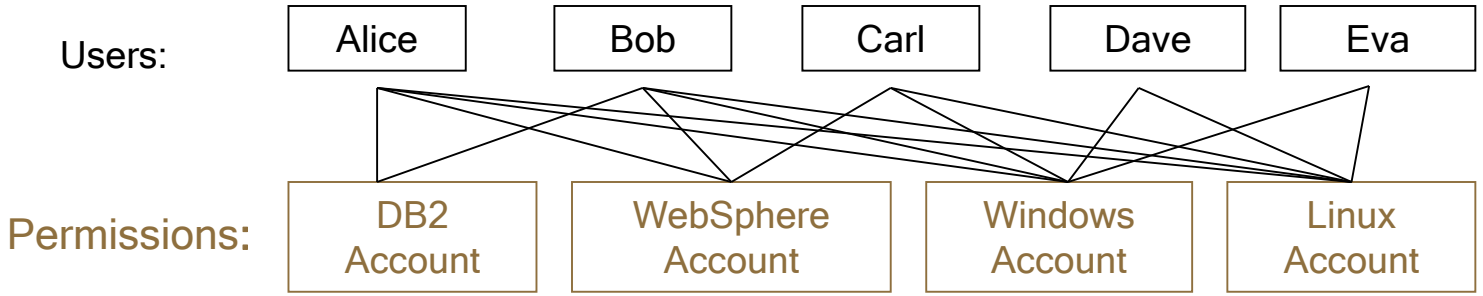
PURDUE UNIVERSITY®

Readings for this segment

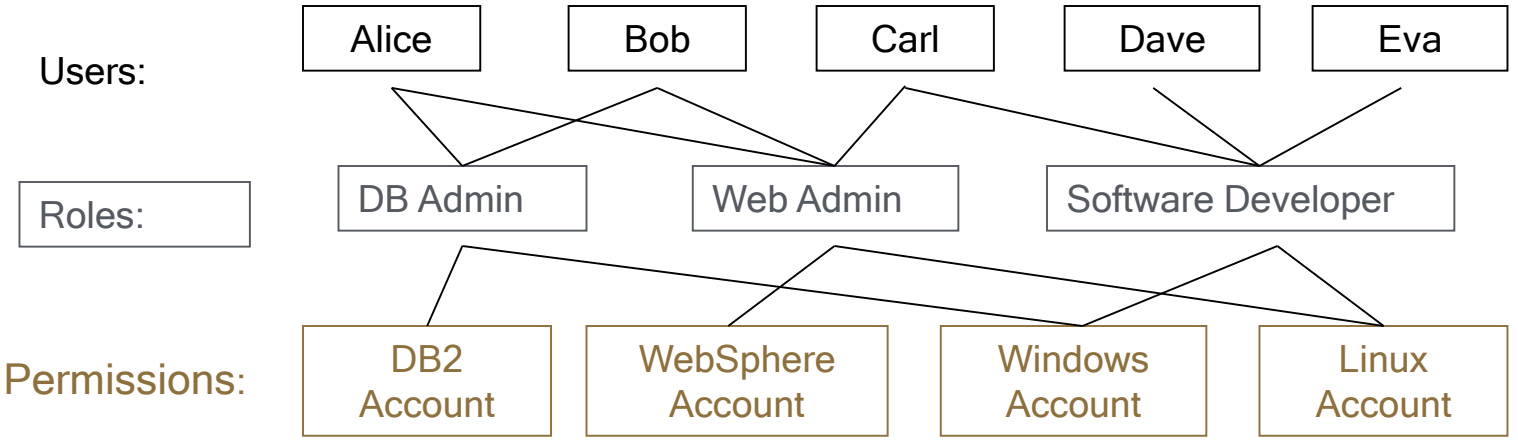
- RBAC96 Family
  - R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. “Role-Based Access Control Models”. *IEEE Computer*, 29(2):38--47, February 1996.

Background: Role Based Access Control

Non-role-based systems



Role-Based Access Control Systems (RBAC)



PURDUE UNIVERSITY

ROLE-BASED ACCESS CONTROL (RBAC)

- Motivation for RBAC: how to administer user-permission relation
  - Different from DAC and MAC, which deal with processes in operating systems
- Roles as a level of indirection
  - Butler Lampson or David Wheeler: "all problems in Computer Science can be solved by another level of indirection"
- RBAC is multi-faceted and open ended
  - Extensions: ARBAC (administrative), CBRAC (constraint), dRBAC (dynamic), ERBAC (enterprise), fRBAC (flexible), GRBAC (generalized), HRBAC (hierarchical), IRBAC (interoperability), JRBAC (Java), LRBAC (Location), MRBAC (Management), PRBAC (privacy), QRBAC (QoS), RRBAC(Rule), SRBAC(Spatial), TRBAC (temporal), V, W, x.

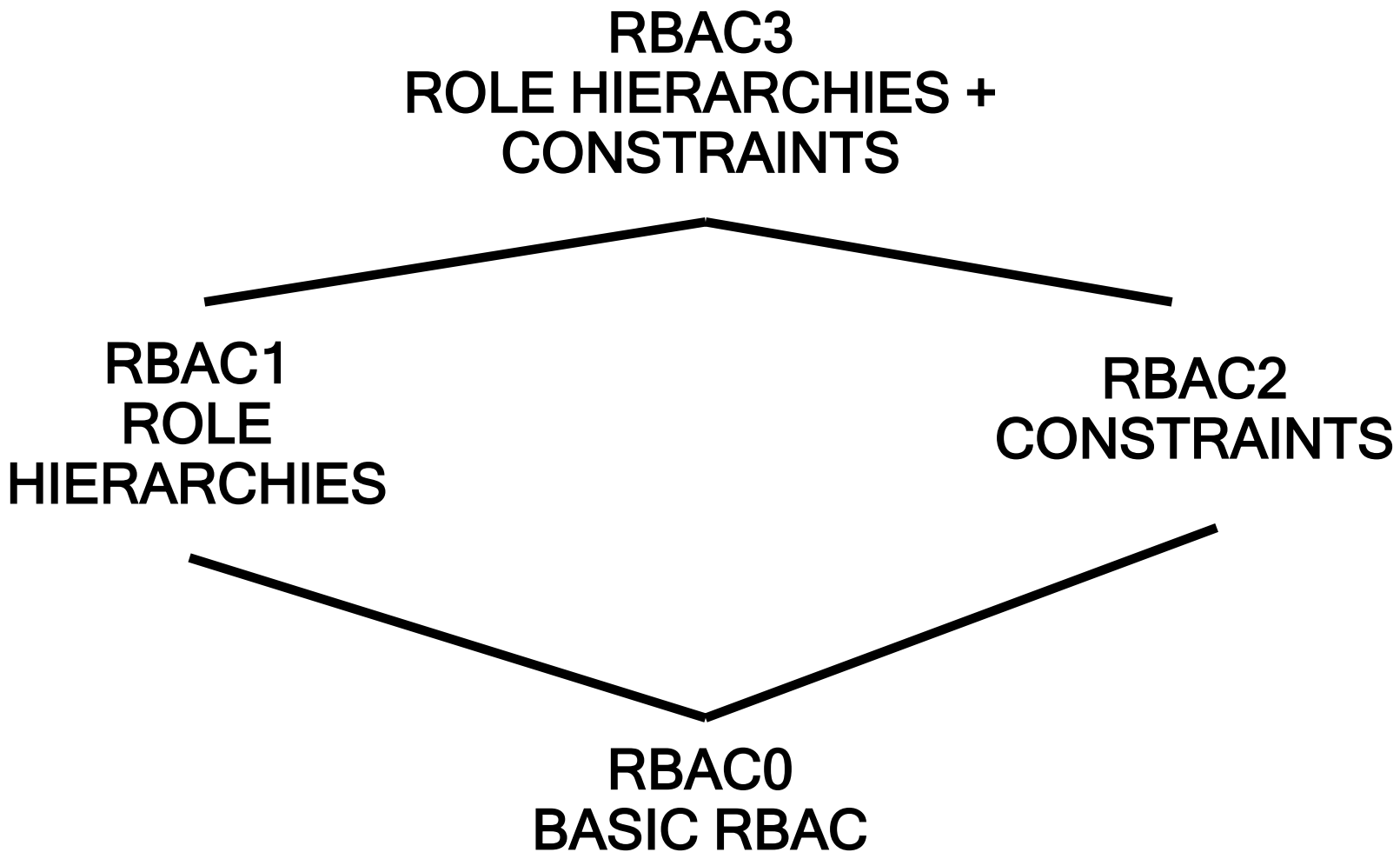
Why Roles?

- Fewer relationships to manage
  - possibly from O(mn) to O(m+n), where m is the number of users and n is the number of permissions
- Roles add a useful level of abstraction
- Organizations operate based on roles
- A role may be more stable than
  - the collection of users and the collection of permissions that are associated with it

Groups vs. Roles

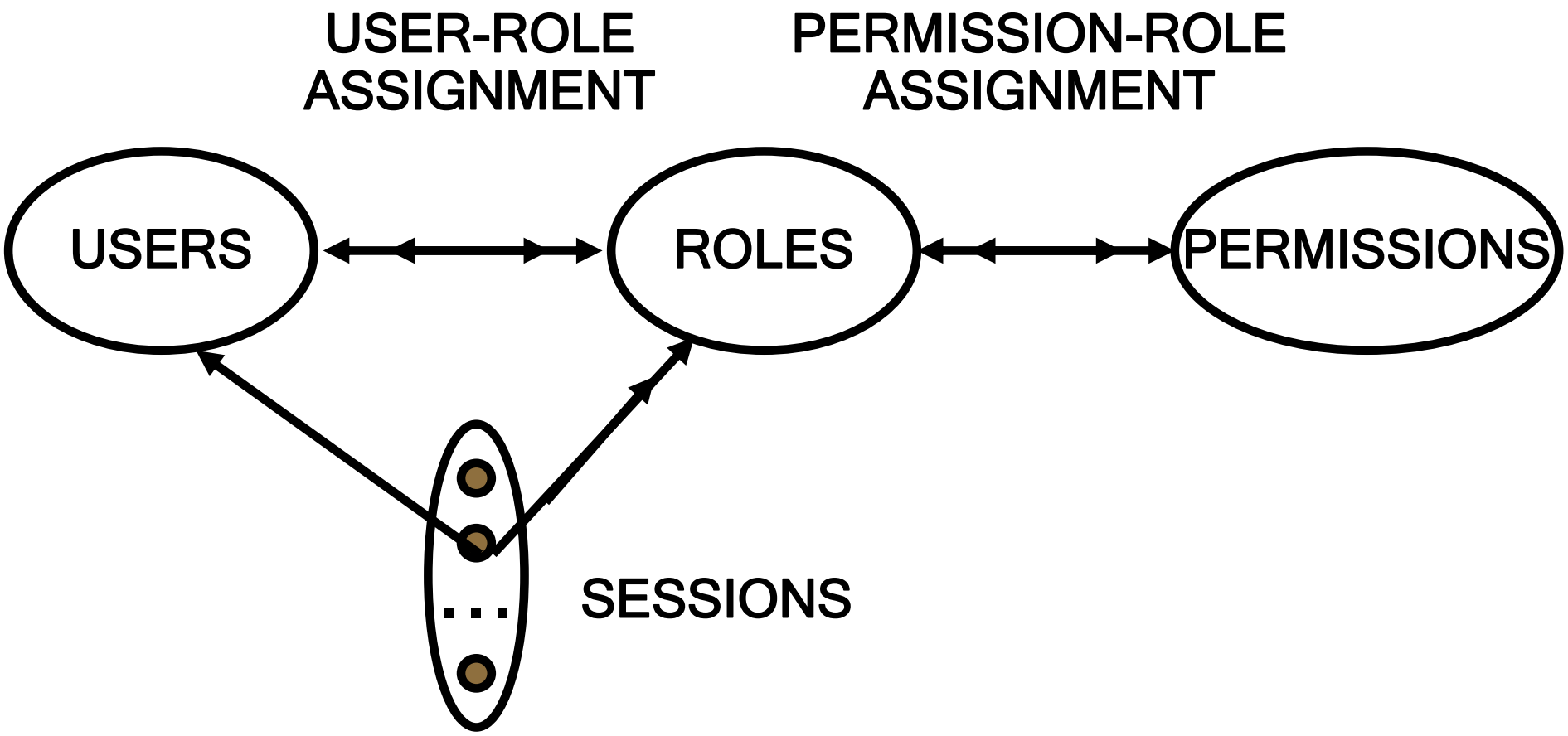
- Depending on the precise definition, can be the same or different.
- Some differences that may or may not be important, depending on the situation
  - Answer 1: sets of users vs. sets of users as well as permissions
  - Answer 2: roles can be activated and deactivated, groups cannot
    - Groups can be used to prevent access with negative authorization.
    - Roles can be deactivated for least privilege
  - Answer 3: can easily enumerate permissions that a role has, but not for groups

RBAC96 FAMILY OF MODELS (Sandhu et al.)



PURDUE UNIVERSITY®

RBACO



PURDUE UNIVERSITY

PERMISSIONS

- Left abstract in the RBAC96 model
- Permissions are positive
- No negative permissions or denials
  - RBAC defines a closed policy, i.e., all accesses are denied unless they are explicitly authorized
- No duties or obligations
  - Example obligation: can access patient document, but must notify patient, or must delete after 30 days

PURDUE UNIVERSITY®

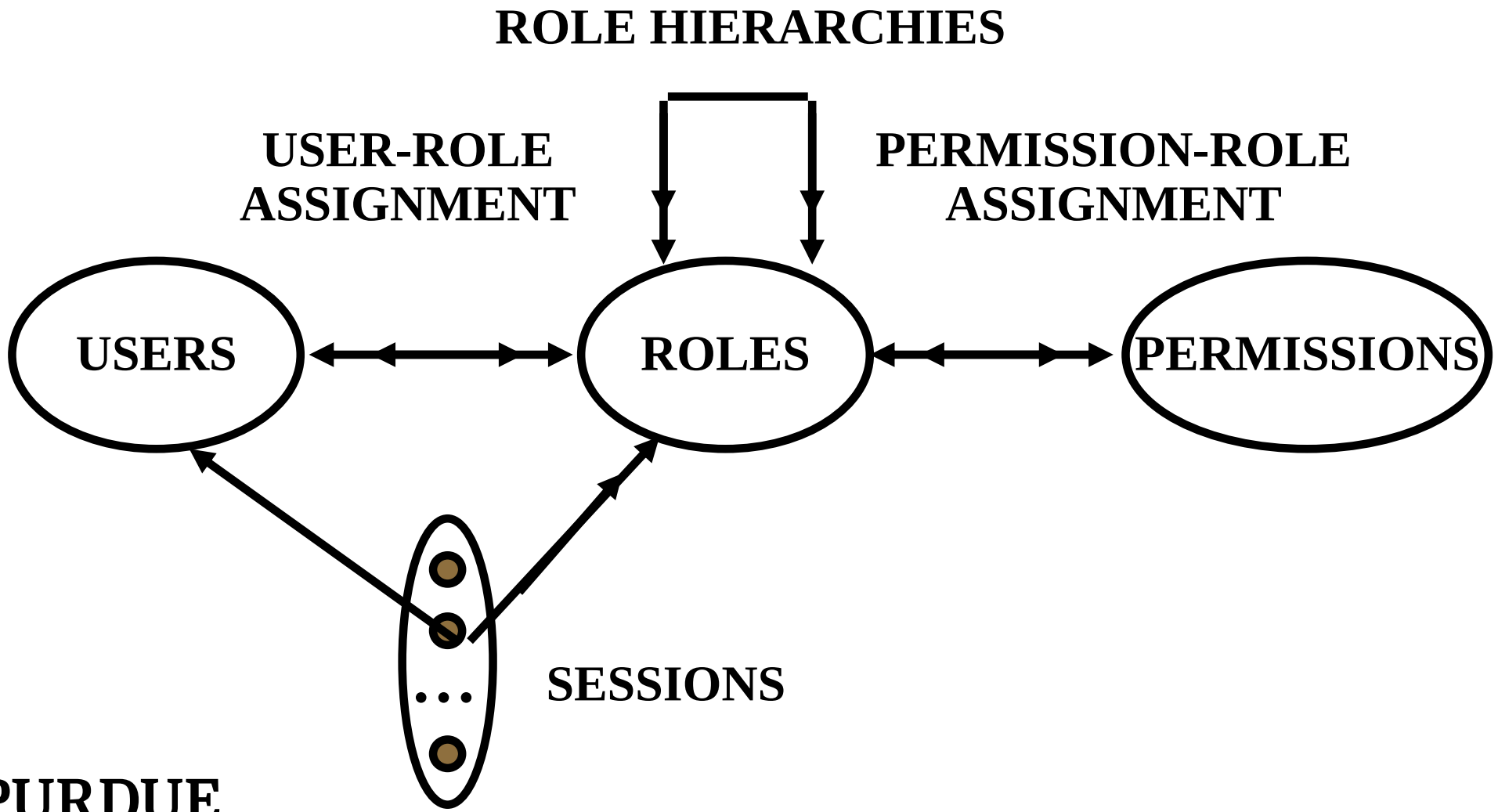
RBAC0: Formal Model

- Vocabulary: U, R, P, S (users, roles, permissions, and sessions)
- Static relations:
  - $PA \subseteq P \times R$ (permission assignment)
  - $UA \subseteq U \times R$ (user assignment)
- Dynamic relations:
- user: $S \rightarrow U$ each session has one user
- roles: $S \rightarrow 2^R$ and some activated roles
  - requires $roles(s) \subseteq \{ r \mid (user(s), r) \in UA \}$

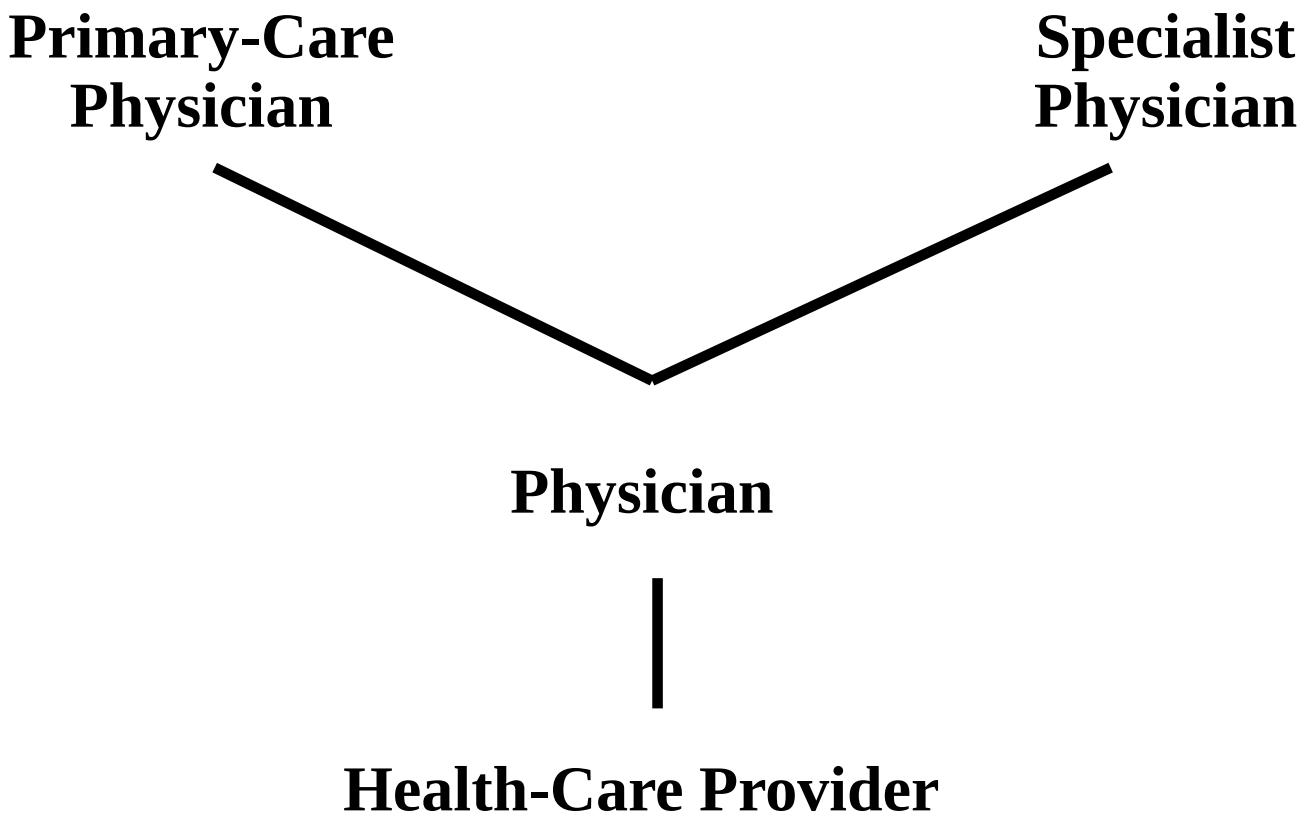
Session s has permissions

$$\bigcup_{r \in roles(s)} \{ p \mid (p, r) \in PA \}$$

PURDUE UNIVERSITY

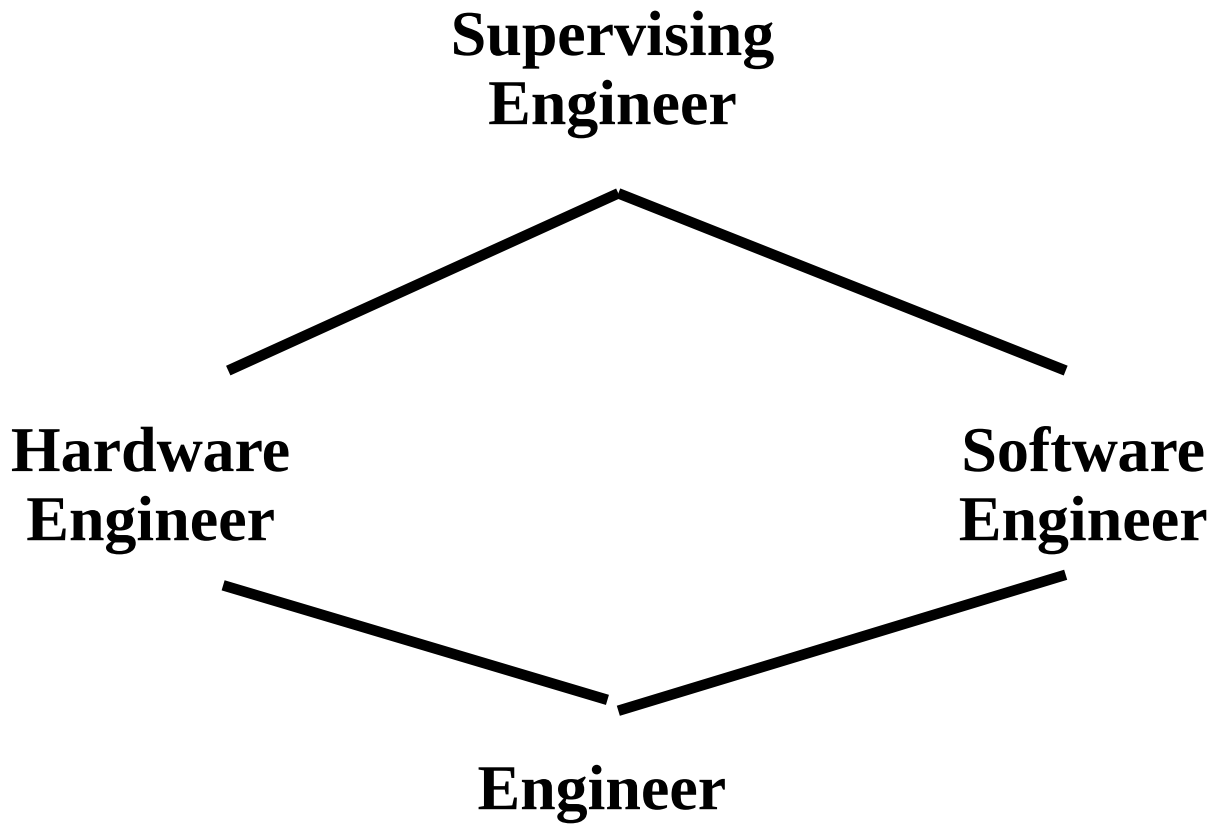

ROLE HIERARCHIES
USER-ROLE ASSIGNMENT
PERMISSION-ROLE ASSIGNMENT
USERS
ROLES
PERMISSIONS
SESSIONS

HIERARCHICAL ROLES (ex 1)



PURDUE UNIVERSITY®

HIERARCHICAL ROLES (ex 2)



PURDUE UNIVERSITY

RBAC1: Formal Model

- U, R, P, S, PA, UA, and user unchanged from RBAC0
- $RH \subseteq R \times R$: a partial order on R, written as $\geq$
  - When $r1 \geq r2$, we say r1 is a senior than r2, and r2 is a junior than r1
- roles: $S \rightarrow 2^R$
  - requires $roles(s) \subseteq \{ r \mid \exists\ r' [(r' \geq r) \ \&\ (user(s), r') \in UA] \}$

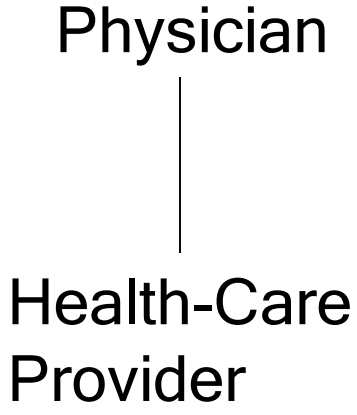
Session s includes permissions

$$\bigcup_{r \in roles(s)} \{ p \mid \exists\ r'' [(r \geq r'') \ \&\ (p, r'') \in PA] \}$$

PURDUE UNIVERSITY

Semantics of Role Hierarchies

- User inheritance
  - r1≥r2 means every user that is a member of r1 is also a member of r2
- Permission inheritance
  - r1≥r2 means every permission that is authorized for r2 is also authorized r1
- Activation inheritance
  - r1≥r2 means that activating r1 will also activate r2



Permission and Activation inheritance have different effect when there are constraints about activation.

PURDUE UNIVERSITY

RBAC2: RBAC0 + Constraints

- No formal model specified for constraints
- Example constraints
  - Mutual exclusion
  - Pre-condition: Must satisfy some condition to be member of some role
    - E.g., a user must be an undergrad student before being assigned the UTA role
  - Cardinality

Mutual Exclusion Constraints

- Mutually Exclusive Roles
  - Static Exclusion: No user can hold both roles
    - often referred to as Static Separation of Duty constraints
    - Preventing a single user from having too much permissions
  - Dynamic Exclusion: No user can activate both roles in one session
    - Often referred to as Dynamic Separation of Duty constraints
    - Interact with role hierarchy interpretation

Cardinality Constraints

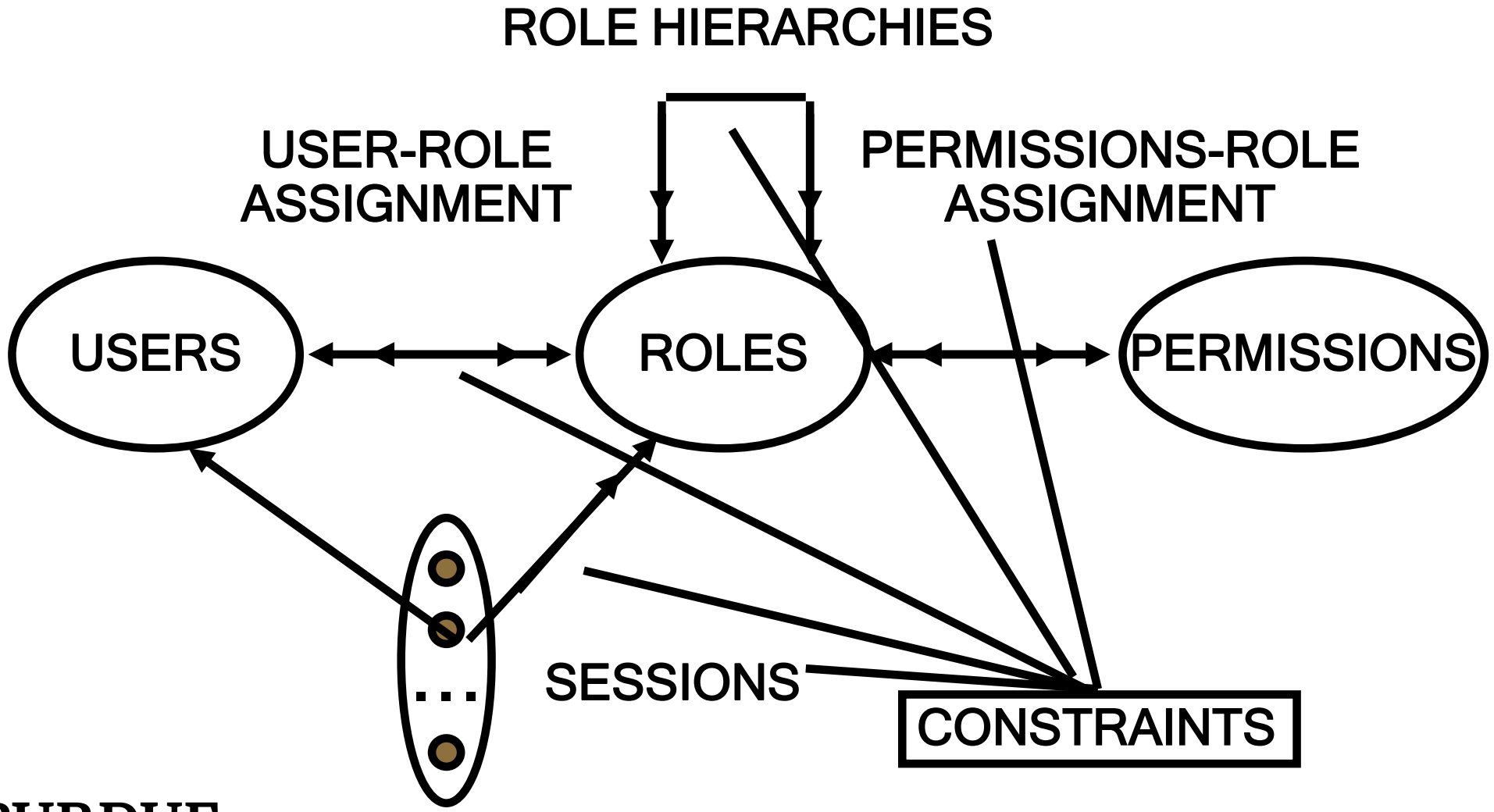
- On User-Role Assignment
  - at most k users can belong to the role
  - at least k users must belong to the role
  - exactly k users must belong to the role
- On activation
  - at most k users can activate a role
  - …

PURDUE UNIVERSITY

Why Using Constraints?

- For laying out higher level organization policy
  - Only a tool for convenience and error checking when admin is centralized
    - Not absolutely necessary if admin is always vigilant, as admin can check all organization policies are met when making any changes to RBAC policies
  - A tool to enforce high-level policies when admin is decentralized

RBAC3



PURDUE UNIVERSITY

Products Using RBAC

- Data Base Management Systems (DBMS)
- Enterprise Security Management
  - IBM Tivoli Identity Manager (central administration and provisioning of accounts, resources, etc)
- Many operating systems claim to use roles

Outline

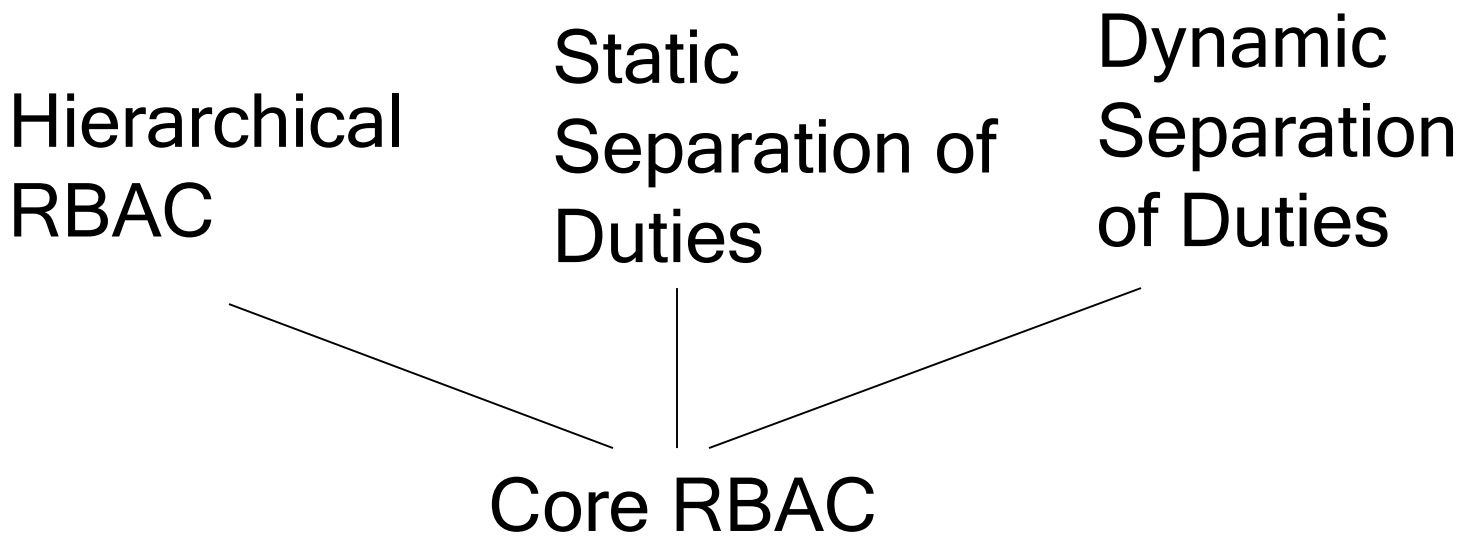
- Non-Interference Model
- Non-deducibility
- The RBAC96 Family of Role Based Access Control Models
- The NIST RBAC Standard and Our Critique
- Attribute Based Access Control and XACML

PURDUE UNIVERSITY

Readings for this segment

- ANSI RBAC standard and its critique
  - Proposed NIST Standard for Role-Based Access Control. David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. TISSEC, August 2001.
  - American National Standards Institute Standard, 2004
  - N. Li, J.-W. Byun, and E. Bertino. "A Critique of the ANSI Standard on Role Based Access Control". *IEEE Security & Privacy, 5(6):41--49, November 2007.*

Overview of the NIST Standard for RBAC



PURDUE UNIVERSITY

Our Critique of the ANSI RBAC Standard

- Many errors
  - Inheritance has been described in terms of permissions; i.e., r1 inherits r2 if all privileges of r2 are also privileges of r1. . . .
    - mistake in cause-effect relationship
  - define permission inheritance as “formally, authorized_permissions(r) = {p $\in$ PRMS | r′ $\geq$ r, (p, r′) $\in$ PA}.”
    - should be r $\geq$ r’
  - The standard defines r1 >> r2 (r1 is immediate parent role of r2) when “there’s no role r3 in the role hierarchy such that r1 $\geq$ r3 $\geq$ r2, where r1 $\neq$ r2 and r2 $\neq$ r3”
    - should be r1 $\neq$ r3
- A number of other limitations and design flaws

PURDUE UNIVERSITY

Our Suggestions for Improving ANSI RBAC Standard

- Remove sessions from core RBAC
- Accommodate single-role sessions as one feature
  - Still allow sessions to activate multiple roles if they want the feature
- Clearly distinguish base and derived relations
- Maintain role-domination relationships explicitly
- Clearly specify role-inheritance semantics

PURDUE UNIVERSITY®

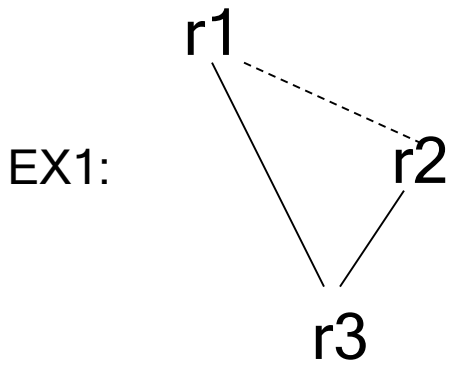
Whether to Allow Multiple Roles to be Activated?

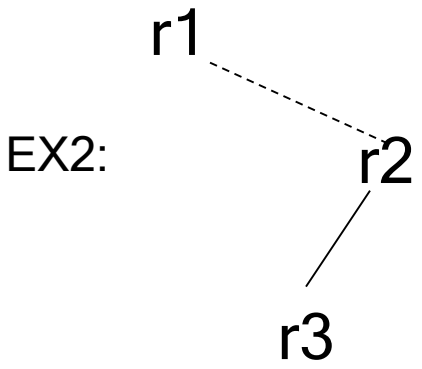
- RBAC96 allows Multi Role Activation
- [Baldwin'90] does not
- Observations:
  - one can define new role to achieve the effect of activating multiple roles
  - dynamic constraints are implicit when only one role can be activated in a session
  - Single-Role Activation is better
    - easier to enforce least privilege
    - better satisfies the fail-safe defaults principle

PURDUE UNIVERSITY

On Modeling Role Hierarchy As A Partial Order

- Modeling RH as a partial order may miss some important information
- Consider the two examples to the right
  - where the dashed edge is added and removed
- Better approach seems to remember the base edges and then compute their transitive and reflexive closure

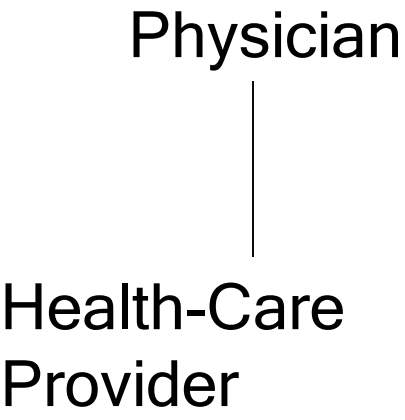




PURDUE UNIVERSITY

Semantics of Role Hierarchies

- User inheritance
  - r1≥r2 means every user that is a member of r1 is also a member of r2
- Permission inheritance
  - r1≥r2 means every permission that is authorized for r2 is also authorized r1
- Activation inheritance
  - r1≥r2 means that activating r1 will also activate r2



They interact with static and dynamic role mutual exclusion constraints.

PURDUE UNIVERSITY

Outline

- Non-Interference Model
- Non-deducibility
- The RBAC96 Family of Role Based Access Control Models
- The NIST RBAC Standard and Our Critique
- Attribute Based Access Control and XACML

PURDUE UNIVERSITY

- ***Attribute-Based Access Control (ABAC)*** is an access control paradigm whereby access rights are granted to users through the use of policies that combine attributes together. The policies can use any type of attributes (user attributes, resource attributes, object, environment attributes, etc.). This model supports Boolean logic, in which rules contain “IF, THEN” statements about who is making the request, the resource, and the action.
  For example: IF the requestor is a manager, THEN allow read/write access to sensitive data.

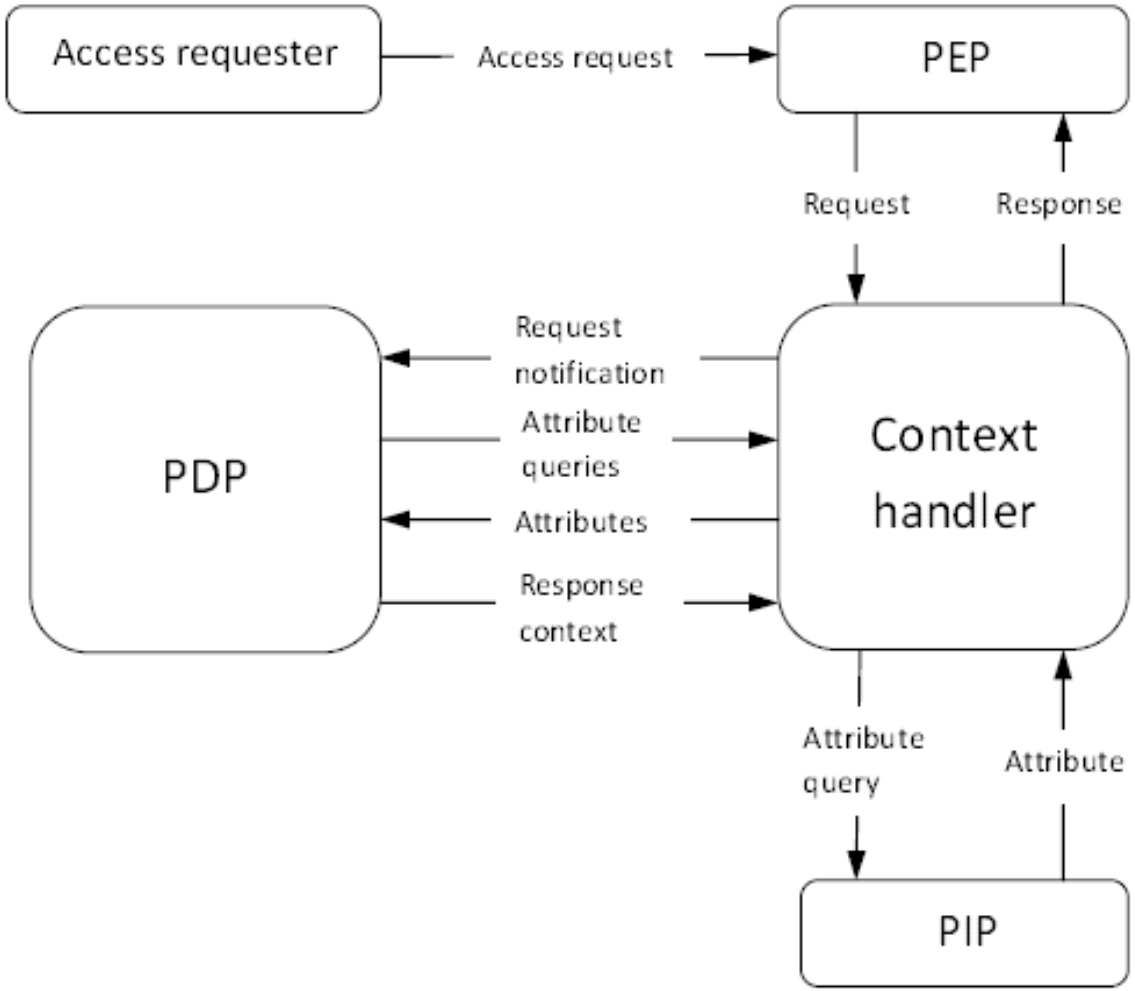
PURDUE UNIVERSITY®

XACML Policy Structure

- Rule
  - Target and condition
  - Effect: Permit, Deny, NotApplicable
- Policy
  - Target and obligations
  - A set of rules and a rule-combining algorithm (RCA)
- Policy Set
  - Target and obligations
  - A set of policies and a policy-combining algorithm (PCA)

XACML Dataflow Model

- Policy Enforcement Point (PEP),
- Policy Decision Point (PDP),
- Policy Information Point (PIP)



PURDUE UNIVERSITY

Rule and Policy Combining Algorithms

- A rule, a policy, or a policy set may return
  - P (Permit), D (Deny), or NA (NotApplicable)
  - Ind (Indeterminate)
    - Overloaded, can mean error or conflict
- Five standard RCAs and six standard PCAs
  - *Deny-overrides* and *ordered-deny-overrides*
  - *Permit-overrides* and *ordered-permit-overrides*
  - *First-applicable* and *only-one-applicable* (PCA only)
- Allow user-defined combining algorithms
  - But does not provide a standard approach or specification language for doing so

Deny-Overrides RCA

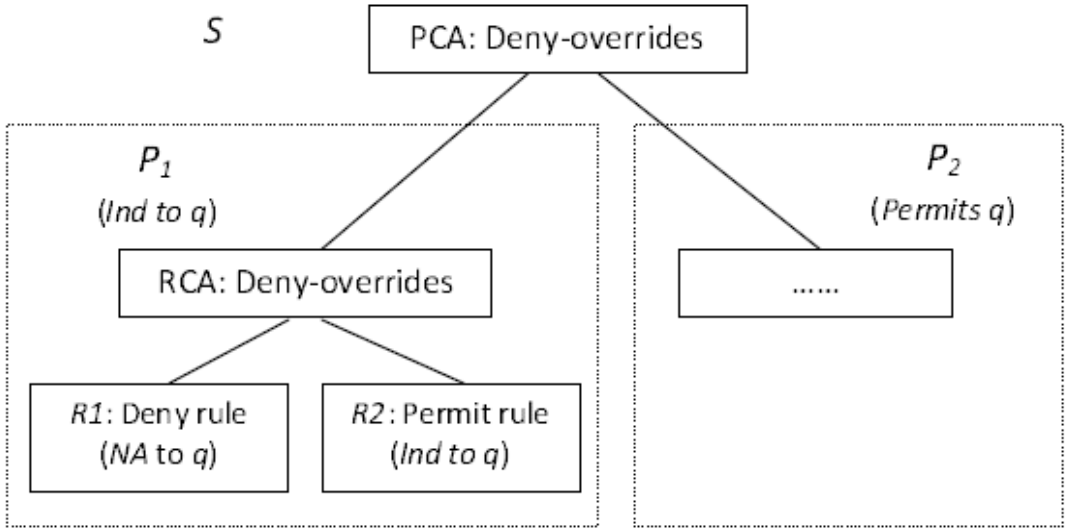
- When no error
  - D > P > NA
- When there is an error
  - D > P > Ind > NA
  - Error in deny-rule: uncertainty over {D, NA}
  - Error in permit-rule: uncertainty over {P, NA}
- Issue 1
  - Ind is treated differently depending on the context it is from. This complicates the specification and understanding of the RCAs

Deny-Overrides PCA

- When no error
  - D > P > NA
- When there is an error
  - Treat Ind as D, and return D immediately when Ind is returned by a policy
  - Conservatively treats Ind as D, due to the overloading of Ind
  - Such a treatment is problematic

Example

S = Deny-overrides(P1 = Deny-overrides(R1, R2), P2)



- S evaluates to D in XACML
- Can argue that it is P that should be returned

- Issue 2: There are plausible ways to treat Ind that are not allowed by XACML

PURDUE UNIVERSITY

Permit-Overrides

- RCA
  - P > D > Ind > NA
- PCA
  - P > D > Ind > NA
- Issue 3
  - Asymmetry between *deny-overrides* and *permit-overrides*

PURDUE UNIVERSITY®

First-Applicable RCA and PCA

- When no error
  - Return the effect of the first rule that applies
- When there is an error
  - Return Ind
- Example
  - S = first-applicable(R1, R2), both R1 and R2 are permit-rules
  - R1 has an error, while R2 applies
  - The standard algorithm returns Ind
  - Can argue that P should be returned

Interaction between PDP and PEP

- Any PEP yields permit (or deny) upon P (or D)
- Base PEP
  - Behavior over NA or Ind is undefined
- Permit-based PEP
  - Yields permit over NA or Ind
- Deny-based PEP
  - Yields deny over NA or Ind
- Issue 4
  - There are plausible interactions between PDP and PEP that cannot be achieved, as too little information is preserved in Ind

PURDUE UNIVERSITY

Principles of Security (Access Control) Mechanisms

- Saltzer and Schroeder: **The Protection of Information in Computer Systems**
  - Fourth ACM Symposium on Operating System Principles (October 1973).
  - Revised version in *Communications of the ACM* 17, 7 (July 1974).

Principles of Security/Access Control

1. Economy of mechanism
   - keep the design as simple and small as possible
2. Fail-safe defaults
   - default is no-access

Principles of Security/Access Control

3. Complete mediation
   - every access must be checked
4. Open design
   - security does not depend on the secrecy of mechanism

Principles of Security/Access Control

5. Separation of privilege
   - a system that requires two keys is more robust than one that requires one
6. Least privilege
   - every program and every user should operate using the least privilege necessary to complete the job

Principles of Security/Access Control

7. Least common mechanism
   - "minimize the amount of mechanism common to more than one user and depended on by all users"
8. Psychological acceptability
   - "human interface should be designed for ease of use"
   - the user's mental image of his protection goals should match the mechanism

Coming Attractions ...

PURDUE UNIVERSITY®