Access Control

Week 2

March 4, 2021

Outline

What is access control?

Its function is to control which principals (persons, processes, machines, etc) have access to which resources in the system

- Which files they can read
- Which programs they can execute
- How they share data with other principals
- ...

Access control is pervasive

- Application
  - Business applications
- Middleware
  - Database management systems
- Operating system
  - Controlling access to files, directories, ports
- Hardware
  - Memory protection, privilege levels

Access control is important

Ross Anderson (Security Engineering) defines

> *Access control is the traditional centre of gravity of computer security. It is where security engineering meets computer science.*

The Orange book evaluates security of computer systems based on access control features and assurance

Access control is interesting

- Has relatively well-developed theories
  - 35+ years history
  - Some theory apparently not useful for other fields
- Many interesting and deep results
- Many misconceptions and debates
- A large percentage of published works contain serious errors
  - Corollary: Be sceptical – don't believe too much what others have written and try form your own opinions

Processing a user request

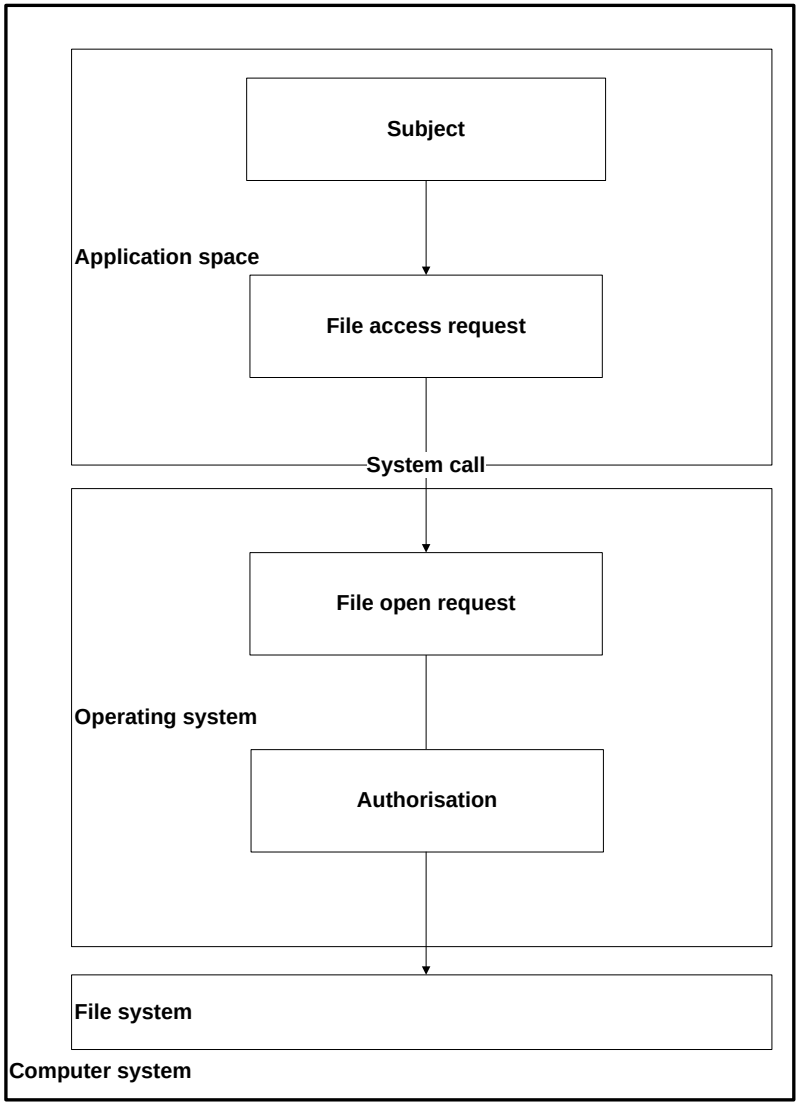
Users want to use resources such as files and printers that are made available to the user and managed by the operating system

- An access request is generated by the system in response to a user interaction with an application program
- A user may enter a program name and a file name parameter from a command line
- A user may select a file from a dialogue box

The request is generated in the course of handling the (file open) system call

- The system call is triggered by the user interaction with the application program
- The request is passed to the authorisation mechanism, which checks that the subject is authorised to access the requested resource

Processing a file access request



Access control policies

An access control policy (or authorisation policy) is defined to encode enterprise security requirements

- The goal of such a policy is to specify the resources for which users are authorised
- A request to access a resource is only permitted if it is authorised by the access control policy

An access control policy specifies which users are authorised to access which resources

- Resource owners may specify the policy
- The policy may be determined by other requirements

Authorisation service

Resources are maintained and protected by the operating system

- The operating system is responsible for enforcing the access control policy

The operating system implements an authorisation service that should intercept all user requests to access resources

- Typically implemented as a set of kernel mode operating system functions that are invoked when processing system calls

Authorisation service

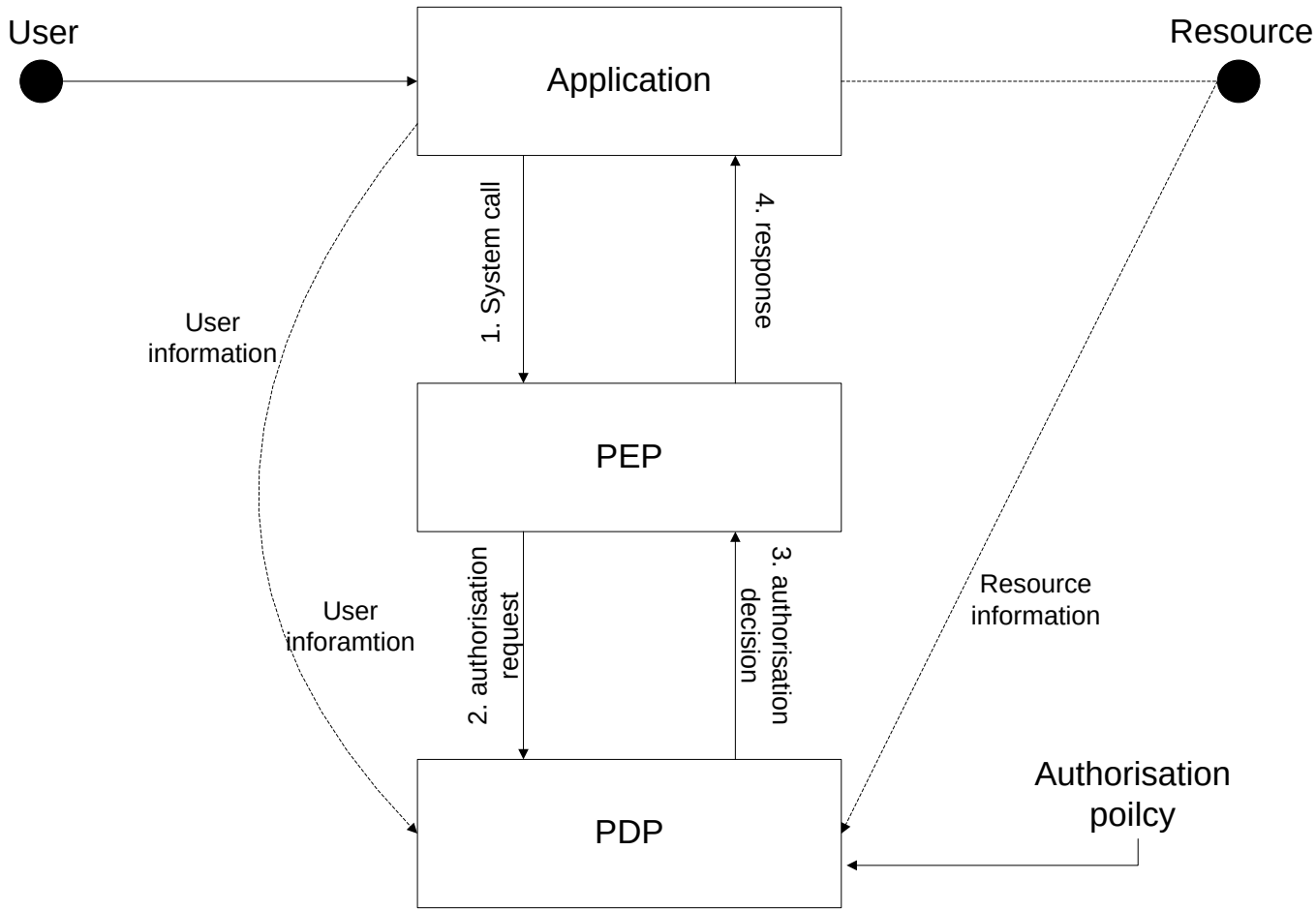
The authorisation service is often implemented as two separate components

- The policy decision point (PDP), or authorisation decision function (ADF), is responsible for determining whether a user request is authorised
- Typically the PDP returns a binary decision (either the request is authorised or it isn't)
- The policy enforcement point (PEP) or authorisation enforcement function, intercepts user requests, forward them to the PDP and enforces the decision of the PDP (allowing the request if it is authorised and denying it otherwise)

The PDP needs to know the following items of information when evaluating a request

- Who is requesting access to what (under what condition)?

Authorisation service architecture



Subjects and objects

Authorisation requires the identification of users by the computer system

- Every authenticated user is associated with information that identifies the user and any other user attributes relevant to authorisation
- We call this information the user's security context

Every process run by the user is associated with her security context

- A process is usually called subject and is the "who" in an access request

The requested resource is usually called an object and is the "what" in an access request

Subjects and objects

Examples of subjects include

- Processes
- Threads

Examples of objects include

- Files
- Directories
- Printers
- Network connections

Principals

A (security) principal is an entity to which authorisation is granted (Saltzer and Schroeder)

- In other words, access control policies specify what principals can (and cannot) do

Typical examples of principals include

- User ID
- Security groups
- Roles
- Cryptographic keys

Principals and subjects

We make distinction between the terms “principal” and ‘subject‘”, although the interpretation of them vary in the literature

- A subject is synonymous with a user, whereas a subject is associated with a collection of principals
- These principals form part of the security context
- The binding of a subject (user) to one or more principals takes place when a user is authenticated

Interacting with objects

Generally the PDP will also need to know how the subject wishes to interact with the object

- We may wish to allow users to read access to a directory but not write

Typical types of interaction include read, write and execute

- The interpretation of these actions vary between systems
- The interpretation of these actions are dependent on the type of the object

Access request

In abstract terms, an authorisation request can be viewed ("modelled") as a triple (s, o, a)

- s is a subject (who)
- o is an object (what)
- a is an action (how)

Access control models

An access control model

- provides a method for encoding an authorisation policy
- defines the conditions that must be satisfied for an authorisation request to be granted

The conditions may be expressed as security properties defined in terms of the (protection) state of the computer system

- The state of a computer system is a snapshot of security relevant information

Access control models

An access control model

- represents the features and behaviour of a security system within a mathematical or logical framework
- provides a means of encoding an authorisation policy
- provides a formal specification of an authorisation service
- provide a blueprint for implementation
- may provide formal assurances of the security of the authorisation service

Outline

Introduction

Very simple model

- Based on the relationship that exists between the subject and object
- A principal is either a user identifier (UID) or a group identifier (GID)
- Each object is associated with an owner (UID) and a GID
- Each subject is associated with a UID and one or more GIDs

Subject-object relationships

There are three different relationships

- The subject owns the object
  - Subject UID matches object's owner UID
- The subject is a member of the group associated with the object
  - Subject has GID that matches object's GID
- The subject is anybody else
  - Subject has no matching UID or GID

This is sometimes referred to as the owner-group-world model

Access rights

Unix supports three generic access rights

- Read (denoted r)
- Write (denoted w)
- Execute (denoted x)

x means different things when applied to different objects

- If the object is a program it means “execute”
- If the object is a directory it means “enter”

To access an object the subject must be able to enter every directory in the pathname of the object

Permission masks

Each object is associated with a access mask

- It defines the actions for which each of owner, group and world are authorised
- Comprises nine bits in three groups of three
- If a bit is set the corresponding access right is granted

Owner read	Owner write	Owner execute	Group read	Group write	Group execute	World read	World write	World execute

Permission masks

The access mark 111 101 101 grants

- r, w and x to the owner
- r and x to the group and world
- Usually written symbolically as rwx r-x r-x

The Unix program ls can be used to list the contents of the current directory

- A "detailed listing" (ls -l) displays the permission mask

Example

```
woody@u    /public_html/Test      ls -l
wl             woody   compstaff   8888   2015-10-05   08:30 project.doc
-rw-rw-r--     woody   compstaff   9999   2015-10-05   08:41   Test
drwxr-x--x
```

- Anyone can read (view) project.doc
- Only woody and members of group compstaff can modify

  project.doc and list the contents of directory Test

Outline

Introduction

A protection matrix is used to represent an authorisation policy

- very simple and intuitive
- It encodes those requests that are authorised

It is a two-dimensional array

- Rows are labelled with principles
- Columns are labelled with objects
- Each entry contains a set of access rights (which may be empty)

An example

Let r denote read, w denote write, and x denote execute

	project.doc	install.exe	exam.html
user_woody	{r, w}	{x}	{r, w}
user_danni	{r}	{x}	{r}

An example

Let r denote read, w denote write, and x denote execute

	project.doc	install.exe	exam.html
user_woody	{r, w}	{x}	{r, w}
user_danni	{r}	{x}	{r}

How do we evaluate the request (user_woody, exam.html, r)?

Groups

- Maintaining a protection matrix for thousands of users and hundreds of thousands of files is an onerous task
- In a large user population many users will share certain characteristics
  - Job description
  - Location
- Create (security) groups
  - Users are associated with groups
  - Used to label rows in the protection matrix
- Groups provide a convenient way of assigning the same rights to many different users
  - Now a subject may be associated with multiple principles

Groups

Assume that users woody and Danni belong to group group is staffand woody and Wei belong to group comp staff

	project.doc	install.exe	exam.html
user_woody	{r, w}	{x}	{r, w}
user_danni		{x}	{r}
user_wei	{r}	{x}	{w}
group_is_staff	{r}	{x}	
group_comp_staff		{x}	{r}

Groups

Assume that users woody and Danni belong to group group is staffand woody and Wei belong to group comp staff

	project.doc	install.exe	exam.html
user_woody	{r, w}	{x}	{r, w}
user_danni		{x}	{r}
user_wei	{r}	{x}	{w}
group_is_staff	{r}	{x}	
group_comp_staff		{x}	{r}

How do we evaluate the request (user_danni, project.doc, r)?

Dealing with multiple principals

How do we evaluate a request from Danni to read the file project.doc

- The authenticated user Danni is associated with two principals user_danni and group_is_staff
- We check the matrix entries for both principals
  - In effect we evaluate two requests (user_danni, project.doc, r) and (group_is_staff, project.doc, r)

Disadvantages of the protection matrix

Authorisation services rarely use a protection matrix

- The memory requirements for such a structure would be considerable
- The performance of access checking algorithm may be poor
- The matrix is likely to be sparse so a lot of storage may be wasted
- Administration is still cumbersome

Access control lists

An access control list (ACL) is associated with an object

- Consists of 0 or more access control entries (ACEs)
- An ACL with 0 entries allows no principle (and hence no subject) access to the object

An ACE specifies a principal and a set of access rights ▶

(user woody, {r, w})

The ACL for project.doc is
[(user_woody, {r, w}), (user wei, {r}), (group is staff, {r})]

Access control lists

An ACL is equivalent to a column of the protection matrix

- ACLs are more efficient (storage and access checking) than a protection matrix because empty entries are ignored

ACLS are used by Windows and modern versions of Unix

Access control lists

How de we evaluate a request from woody to read the fileproject.doc?

- Find the ACL for project.doc
- Check each ACE of the ACL
- If the principal in the ACE is one associated with woody, then

  compare the requested access with that granted in the ACE

ACL implementations

A number of options are available when a subject may be associated with several principals (and hence with several ACEs)

- Length of ACL
  - Fixed length ACLs (owner-group-world model)
  - Variable length ACLs (Windows)
- Access checking
  - Match only one principal (owner-group-world model)
  - Match all principals (Windows)

Fixed length ACLs

Each object is associated with a fixed number of principals

- Unix identifies owner, group and world

The ACL contains an ACE for each principal

- Unix "ACLs" contain 3 entries rwx r-x r-x
- The first entry is for the owner, the second for the group, and the third for everyone else

Varied length ACLs

Fixed length ACLs support only a very limited number of authorisation policy options

- Cannot assign different access rights to two different users or two different groups

Variable length ACLs can include an ACE for each principal identified in the system

- Windows support variable length ACLs

Access control checking

Match the first relevant principal

- Unix stops checking once it finds a relevant principal
- If an object has "ACL" --- --- rwx then neither the owner nor members of the group will be able to access the object

Match all relevant principals

- Windows checks all ACEs in the ACL before denying access
- exam.html has ACL containing these two entries $[(\text{user\_wei}, \text{w}), (\text{group\_comp\_staff}, \text{r}), \ldots]$
- A request from Wei to read exam.html would be granted after examining the second ACE

A difficult policy to implement

Consider the following policy

- All computing staff can read all the files in a particular directory
- Wei is not allowed to read one particular file exam.html in this directory

How do we implement this policy?

- Cannot use the group group_comp_staff to grant access to all files in the directory
- Have to grant access for each member of computing staff to each file in the directory
  - If there are 10 users and 100 files the administrator has to create 999 authorisations

Negative authorisation

Typically used to specify exceptions to a general authorisation policy

- For each file in the directory add (group_comp_staff, r) to the ACL
- Add a negative (access denied) ACE (user_wei, r) to the ACL for exam.html
- The administrator has to create 101 authorisations
- Windows supports negative authorisation

Negative authorisation

Very difficult to reason about overall security policy

- More principals associated with subject does not imply more authorised requests for subject (policy is non-monotonic)

Conflicting authorisations

- Wei is authorised to read exam.html by the ACE for the group group_comp_staff and denied access by the ACE for user_wei

Need to have conflict resolution strategy

- Deny overrides (Windows)
- Permit overrides
- First applicable

Capability lists

ACLs have one disadvantage

- It is difficult to determine the access rights of a particular subject ("before-the-act per-subject review")
- Have to check the ACL of each object to see if it contains an ACE for a principal associated with the subject

The alternative view of the protection matrix is to consider its rows

- Each row generates a capability list
- Each entry specifies an object and a set of access rights
- Associated with a principal
- The capability list for user_woody would be $[(\text{project.doc}, \{r, w\}), (\text{install.exe}, x), (\text{exam.html}, \{r, w\})]$

Capability lists

Capability lists are often used internally by operating system

- A process is associated with a list of objects to which access has been granted
- Windows 2000 maintains a list of (object handle, granted access mask) pairs for each process

Capability lists are rarely used for specifying security policy in operating systems

- Digitally signed capabilities are sometimes used in open systems (X.509 attribute certificates)

Reading

- J. Crampton. Why We Should Take a Second Look at Access Control in Unix. In *Proceedings of the 13th Nordic Workshop on Secure IT Systems*. 27–38, 2008.
- R.J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems* (2nd edition), John Wiley & Sons, 2008 – Chapter 4

Outline

Introduction

One problem with early access control models is that the policy was assumed to be static

- Only trusted users could change the protection matrix

The HRU model considered the effect of changes to the matrix on security

- Commands are allowed to change the matrix
- This allows users to grant access to the objects they own
- This is usual interpretation of a discretionary access control mechanism

Introduction

System designers and system administrators want systems that permit users to manage access to the objects they own

- There is a demand for "feature"
- It is desirable to reduce the administrative burden of system administrators

The main contribution of Harrison, Ruzzo and Ullman was to show that it is extremely difficult to predict the behaviour of such dynamic systems

Components of the model

- The protection matrix model is defined by a matrix M which is defined by S, O and R
- The model is extended to include
  - A set of operations P
  - A set of commands C
- A protection system is defined by an initial matrix M_0, C and R
  - Different choices of M_0, C and R will give rise to different systems
  - The set of operations is fixed

Operations

The operations are

- createSubject — adds a row to the matrix
- createObject — adds a column to the matrix
- enter — adds an access right to a matrix entry
- destroySubject — deletes a row of the matrix
- destroyObject — deletes a column of the matrix
- delete — deletes an access right from a matrix entry

Commands

A command consists of a conditional statement and a body

- The conditional statement has the form "if certain access rights are present in the matrix then"
- The body comprises one or more operations
  - The body is only executed if the conditional statement evaluates to true

In other words a command may modify the state of the protection system

- Whether it does or not is determined by the current state of the protection system (the matrix)
- The modification is determined by the body of the command

A typical command

function GRANTREAD(s,f,o)
  if own $\in [s, o]$ and $r \in [s, o]$ **then** ▷ conditional statement
    enter r in $[f, o]$ ▷ body of command
  end if
end function

- The function takes two subjects s and f and an object o as parameters
- If s owns the object and can read the object then it may grant (add to the matrix) read access to f

A typical command

The execution of GRANTREAD(*woody*, *wei*, *project.doc*) causes thefollowing matrix transition

	project.doc	*index.html*
woody	r, own	
wei		*w*

A typical command

The execution of GRANTREAD(*woody*, *wei*, *project.doc*) causes thefollowing matrix transition

	project.doc	*index.html*
woody	r, own	
wei	r	*w*

The command GRANTREAD(*wei*, *woody*, *project.doc*) does notchange the matrix

A typical command

The execution of GRANTREAD(*woody*, *wei*, *project.doc*) causes thefollowing matrix transition

	project.doc	*index.html*
woody	r, own	
wei	r	*w*

The command GRANTREAD(*wei*, *woody*, *project.doc*) does not change the matrix

- *wei* does not have the own access right for *project.doc* so the conditional statement evaluates to false

The safety problem

- A (protection) system is defined by an initial matrix M_0, C and R
- Executing a sequence of commands $c_0, \ldots, c_{n-1}$ will generate a sequence of matrices $M_1, \ldots, M_n$

$$M_0 \xrightarrow{c_o} M_1 \xrightarrow{c_1} \ldots \xrightarrow{c_{n-1}} M_n$$

- Harrison, Ruzzo and Ullman defined and studied the safety problem
  - Does there exist a sequence of commands that leads to the entry of a particular access right r into the matrix?
  - The safety problem is an important question as it can be used to determined whether a protection system will continue to implement a given security requirement

The safety problem

- A (protection) system is defined by an initial matrix M_0, C and R
- Executing a sequence of commands $c_0, \ldots, c_{n-1}$ will generate a sequence of matrices $M_1, \ldots, M_n$

$$M_0 \xrightarrow{c_o} M_1 \xrightarrow{c_1} \ldots \xrightarrow{c_{n-1}} M_n$$

- Harrison, Ruzzo and Ullman defined and studied the safety problem
  - Does there exist a sequence of commands that leads to the entry of a particular access right r into the matrix?
  - The safety problem is an important question as it can be used to determined whether a protection system will continue to implement a given security requirement
- It is impossible to construct an algorithm that can answer the safety problem for all possible choices of M_0, C and R

Outline

Introduction

Accessing an object can be regarded as initiating an information flow

- Read access causes information to flow from an object to a subject
- Write access causes information to flow from a subject to an object

An information flow policy specifies which information flows are authorised

- Motivated by confidentiality requirements in military systems
- A classified user can read unclassified documents
- An unclassified user can not read classified documents

An information flow policy for read access

In a paper-based system, the security classification is stamped on the document

- A user u is only allowed access to the document d if she can prove that her classification is at least as high as that of d

We formalise this policy using mathematical notation

- There is a set of ordered security labels L and a function λ which associates each user and document with an element in L
- To check whether read access is permitted the security label associated with the user $\lambda(u)$ is compared with the security label associated with the document $\lambda(d)$
- Access is authorised if $\lambda(u) \geqslant \lambda(d)$
- This is sometimes known as a "no-read-up" policy

An example

Let $L = \{\text{unclassified}, \text{classified}, \text{secret}, \text{topsecret}\}$ with

$$\text{unclassified} < \text{classified} < \text{secret} < \text{topsecret}$$

If *woody* has security label classified and *project.doc* has securitylabel unclassified then

- $\lambda(\textit{woody}) = \text{classified}$
- $\lambda(\textit{project.doc}) = \text{unclassified}$
- $\lambda(\textit{woody}) > \lambda(\textit{project.doc})$
- *woody* is authorised to read *project.doc*

An information flow policy for write access

- When a user writes to an object information flows from the user to the object
- Information flow must always flow to an entity with at least as high a security label as the source of the information
- For a user u to write to a document d we require $\lambda(u) \leqslant \lambda(d)$!
  - *woody* would not be permitted to write to *project.doc*
  - This is sometimes known as a "no-write-down" policy

Justifying the no-write-down policy

Consider a user with topsecret clearance who (unwittingly) runs a Trojan horse program

- That program runs as a topsecret subject and can read all documents
- If it could write to an unclassified document the program could "de-classify" any document

Consider a user with topsecret clearance who wishes to print a topsecret document

- The ability to print is usually implemented as the ability to write to a printer object
- If the user could (mistakenly) print the document to an unclassified printer the document could be de-classified

Partially ordered sets of security labels

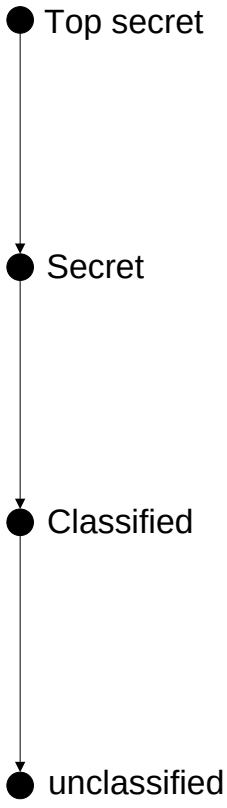
A set is totally ordered if every pair of elements can be compared

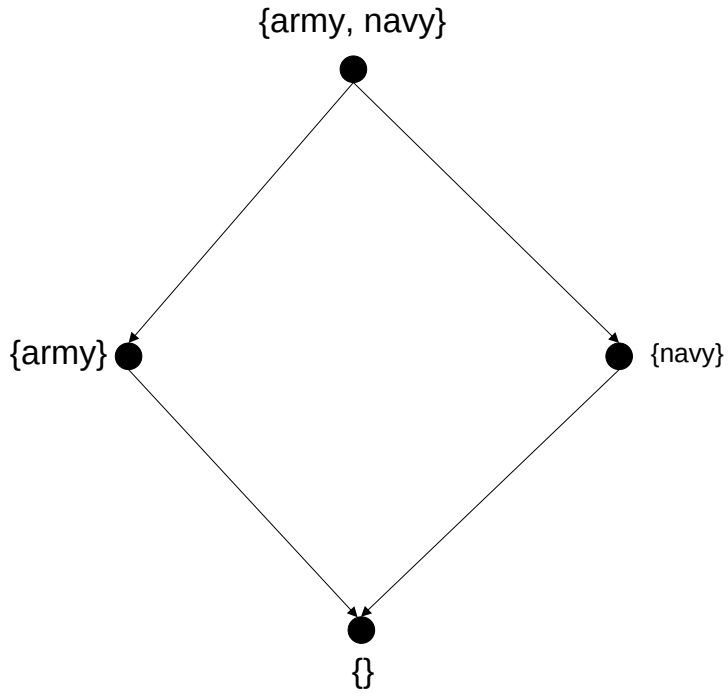
- $\{\text{unclassified}, \text{classified}, \text{secret}, \text{topsecret}\}$ is totally ordered

Some sets are partially ordered

- Consider two different categories: army and navy
- We could label documents and users with 0 or more of these labels
- There are four possibilities: $\{\}$, $\{\text{army}\}$, $\{\text{navy}\}$, and $\{\text{army}, \text{navy}\}$
- Naturally, $\{\} < \{\text{army}\} < \{\text{army}, \text{navy}\}$ and $\{\} < \{\text{navy}\} < \{\text{army}, \text{navy}\}$
- However, $\{\text{navy}\} \not< \{\text{army}\}$ and $\{\text{army}\} \not< \{\text{navy}\}$?

Two examples



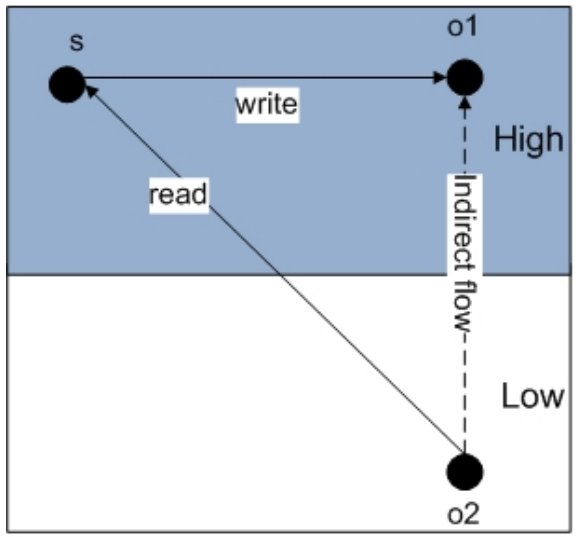


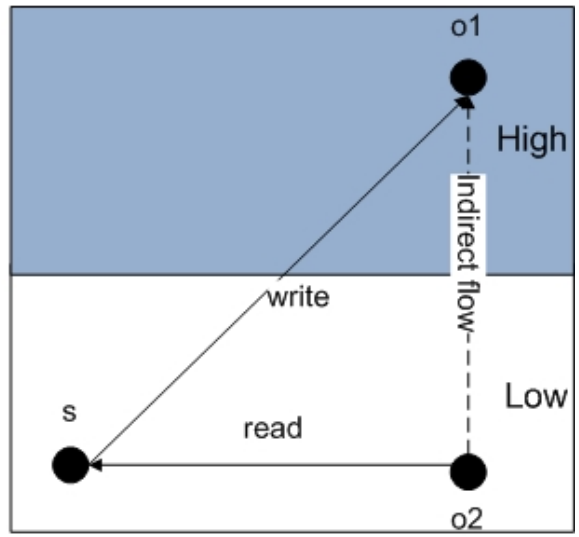
What if $\lambda(woody) = \{\text{army}\}$ and $\lambda(project.doc) = \{\text{navy}\}$?

- Can *woody* read *project.doc*

Indirect information flows

Having read access to one object and write access to another permits indirect information flows between objects





It can be shown that illegal indirect information flows between objects cannot occur if the information flow policy for read and write is enforced

Summary

An information policy specifies

- A partially ordered set of security labels $(L, \leqslant)$
- A security function $\lambda : S \cup O \to L$
- A read request (s, o, r) is authorised if $\lambda(s) > \lambda(o)$
- A write request (s, o, w) is authorised if $\lambda(s) \leqslant \lambda(o)$

Outline

Introduction

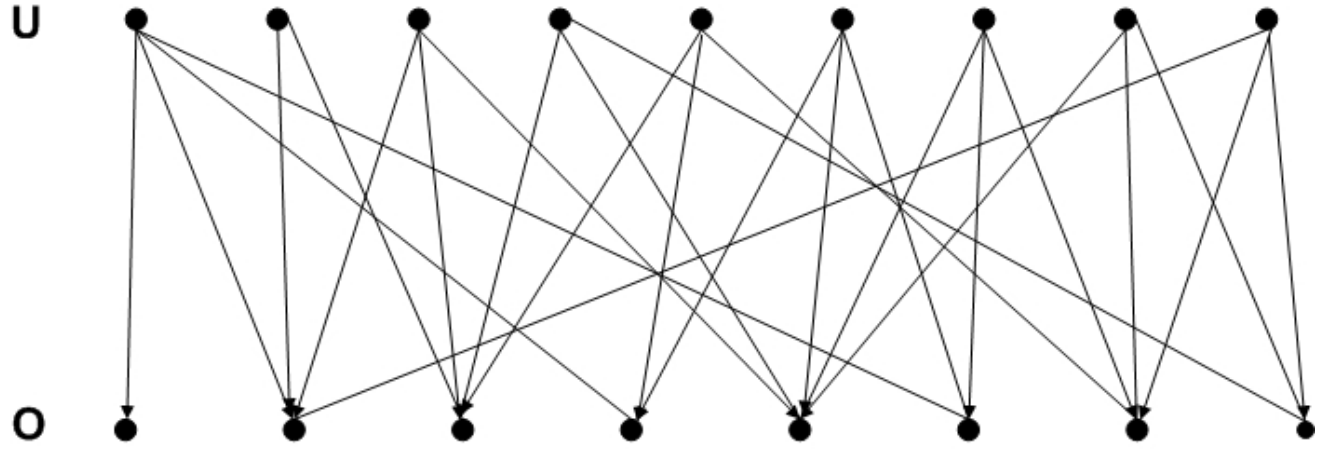
Administration of access control systems can be time-consuming

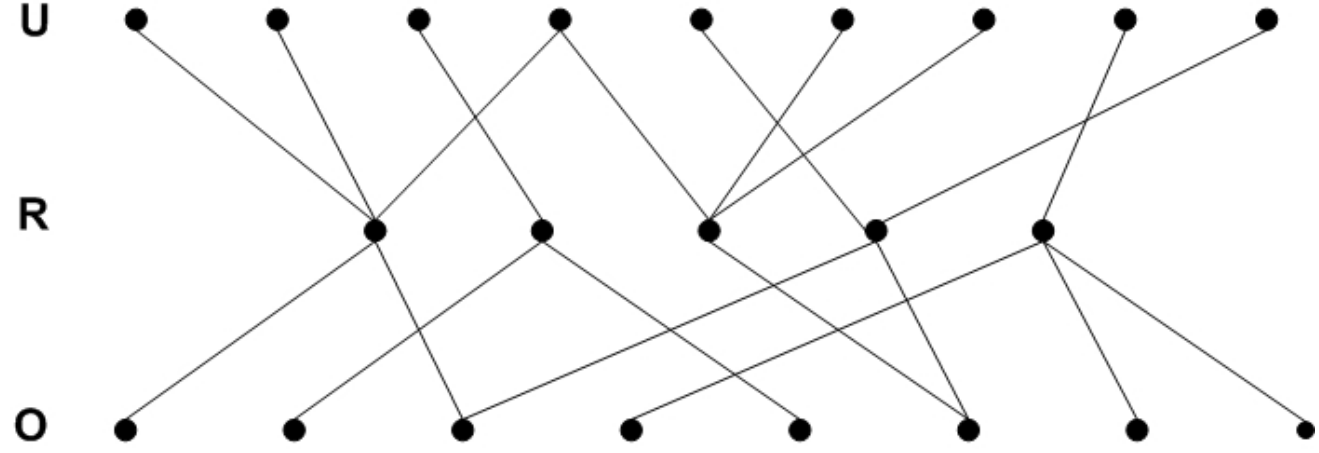
- A system with 1000 users, 100000 objects and 10 access rights contains $1000 * 100000 * 10 = 10^9$ possible authorisation triples
- Default access rights for newly created objects, security groups and ACLs lead to some reduction in the administrative burden
- The potential for mistakes and omissions is large

The central concept in RBAC is that of a role

- Acts as a bridge between users and objects
- Reduces complexity of configuring authorisation policy
- Can be used to automate "user provisioning" by integrating authorisation and human resources databases

Introduction



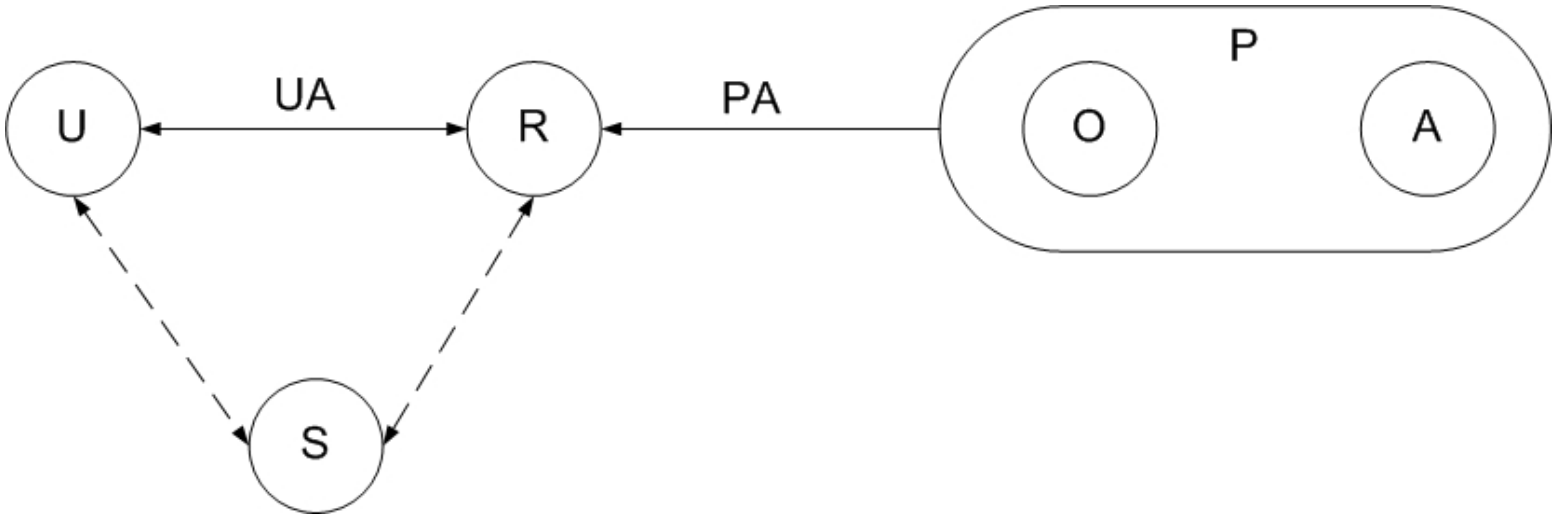


The ANSI RBAC standard: Core component

The standard define the following sets and relations

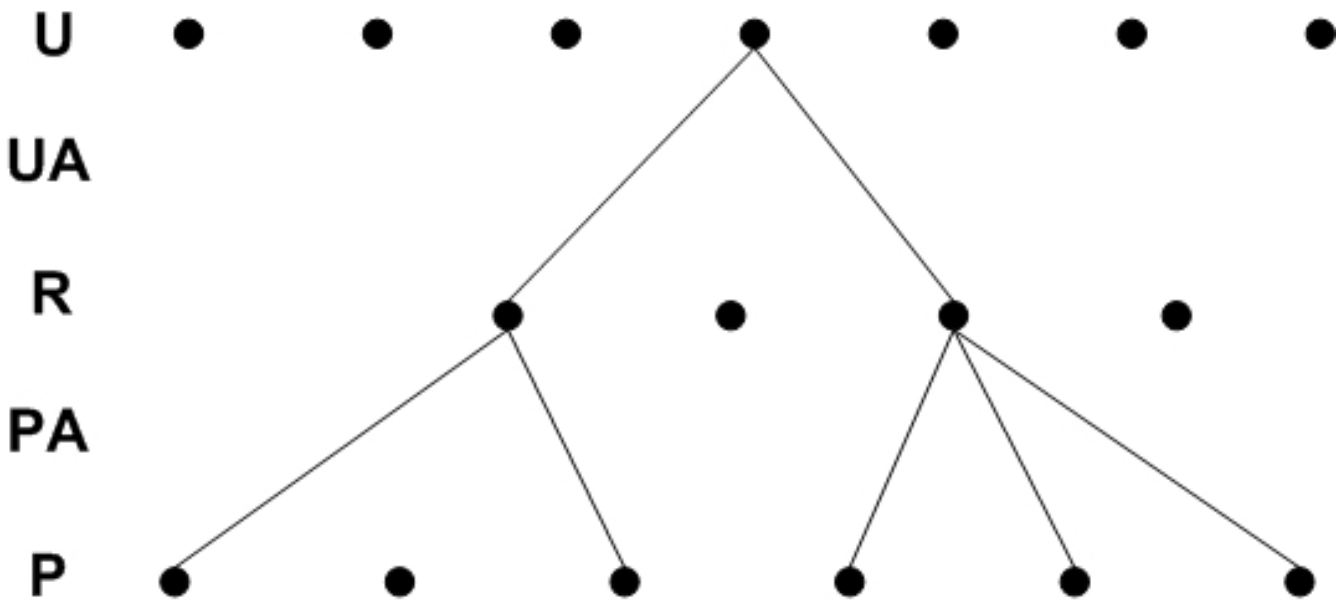
- A set of users U
- A set of roles R
- A set of operations A
- A set of objects O
- A set of permissions $P \subseteq O \times A$
- A user-role assignment relation $UA \subseteq U \times R$
- A permission-role assignment relation $PA \subseteq P \times R$
- A set of sessions $S \subseteq U \times 2^R$

Core RBAC

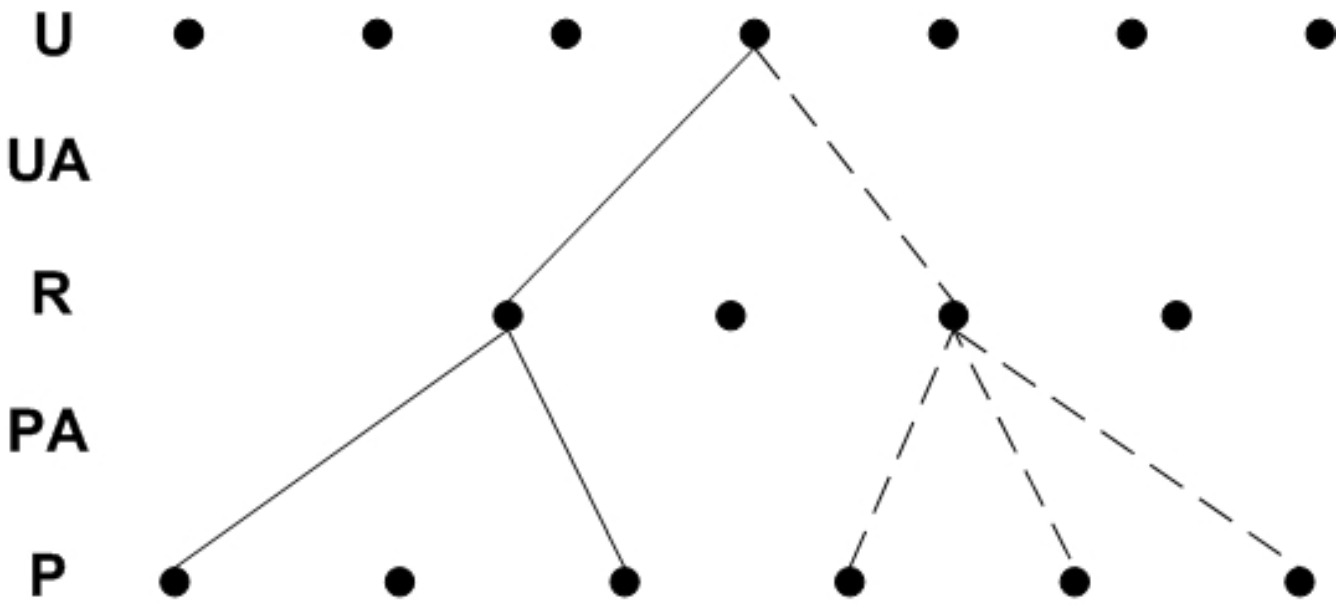


- Users are associated with one or more roles (*UA* relation)
- Permissions are associated with one or more roles (*PA* relation)
- A user activates a session (security context) when authenticating by selecting one or more roles with which she is associated

Core RBAC



Core RBAC



The ANSI RBAC standard: Hierarchical RBAC

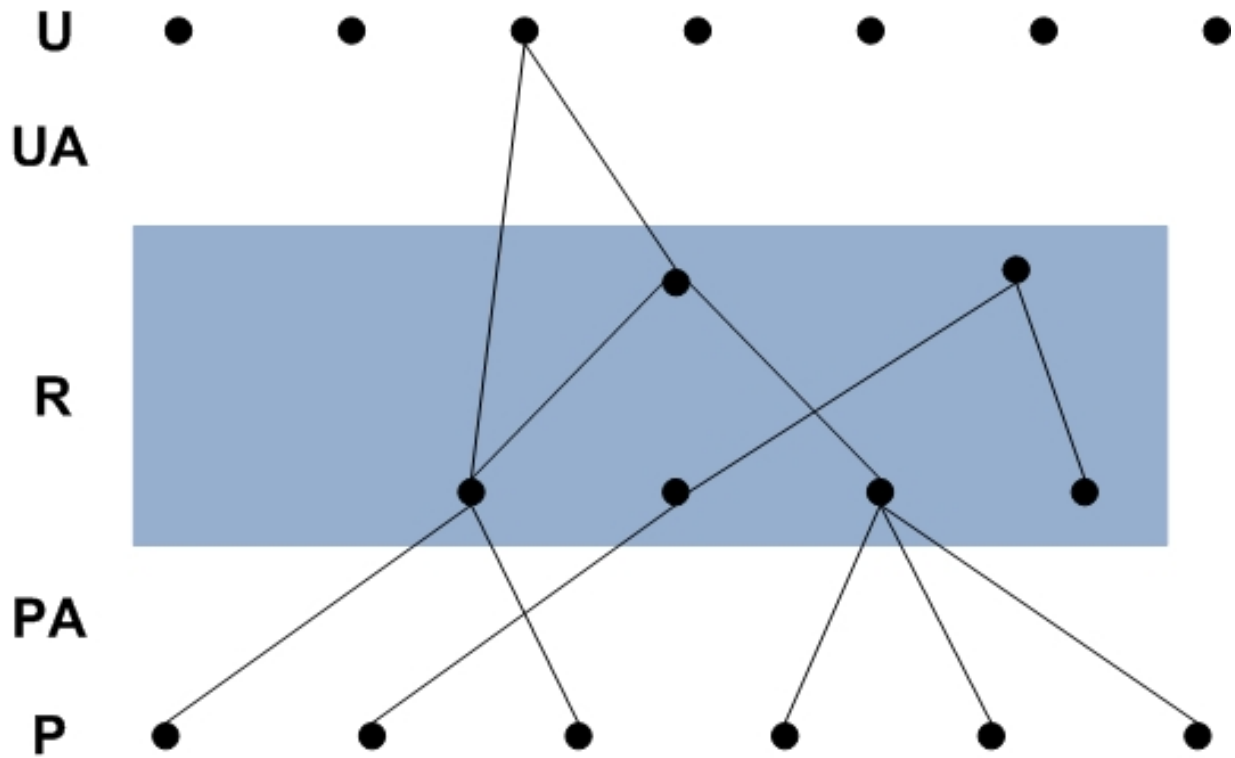
A role hierarchy is used to impose a structure on the set of roles in order to further reduce the administrative burden

- The intuition is that the role hierarchy will encode the organisational structure
- More senior roles will inherit the rights of other roles

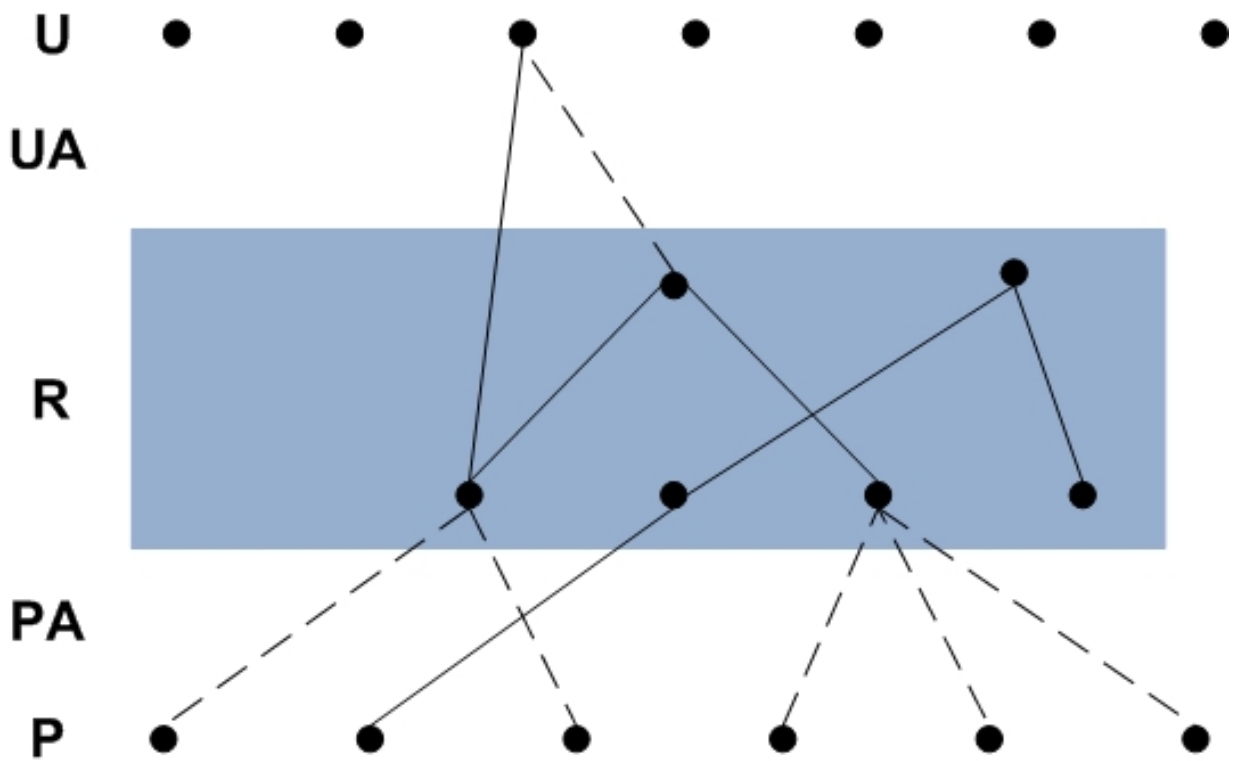
The ANSI standard defines

- A role hierarchy to be a partially ordered set $(R, \leqslant)$
- A user assigned to a role r (via UA) is implicitly assigned to all roles $r' \leqslant r$
- A permission assigned to a role r (via PA) is implicitly assigned to all roles $r' \geqslant r$
- The ANSI standard introduces the notion of a limited role hierarchy which is an inverted tree

Hierarchical RBAC



Hierarchical RBAC



The ANSI RBAC standard: Constrained RBAC

Constraints are used to define separation of duty (SoD) requirements

- The intuition is that certain tasks may require more than one user to complete (“two-man rule”, “four-eyes rule”)
- The relevant permissions for such a task are assigned to different roles
- The membership of those roles is restricted by separation of duty constraints

Separation of duty constraints

Static separation of duty limits the assignment of users to roles

- A static SoD constraint is specified as a pair (R', n) where $R' \subseteq R$ and n is an integer
- No user may be assigned to n or more roles in the set R'
- $(\{\text{finClerk}, \text{poClerk}\}, 2)$ states that no user may be assigned to both the financial clerk and purchasing clerk roles

Separation of duty constraints

Static separation of duty limits the assignment of users to roles

- A static SoD constraint is specified as a pair (R', n) where $R' \subseteq R$ and n is an integer
- No user may be assigned to n or more roles in the set R'
- $(\{\text{finClerk}, \text{poClerk}\}, 2)$ states that no user may be assigned to both the financial clerk and purchasing clerk roles
- Satisfaction of static constraints is enforced by the UA relation

Separation of duty constraints

Dynamic separation of duty limits the activation of roles by users in sessions

- Dynamic SoD constraints are specified in the same way as static SoD constraints
- Satisfaction of dynamic constraints is enforced by the authentication mechanism

What issues arise if constrained RBAC is used with hierarchical RBAC?

RBAC administration

Changes may need to be made to the configuration of an RBAC system

- Assign (revoke) a user to (from) a role
- Assign (revoke) a permission to (from) a role
- Add (delete) a role
- Add (delete) an edge from the role hierarchy

(Compare with HRU)

RBAC administration

Two possible approaches

- Administrative permissions assigned to (administrative) roles
- Issues analogous to the safety problem arise
- Use structure of role hierarchy to limit power of (administrative) roles
  - ARBAC97 (Sandhu et al.) – encapsulated ranges are the basic unit of administration
  - RHA models (Crampton and Loizou) – administrative domains are defined by the structure of the hierarchy

Still an open area of research . . .

Outline

Introduction

A recent standard (eXtensible Access Control Markup Language) defines

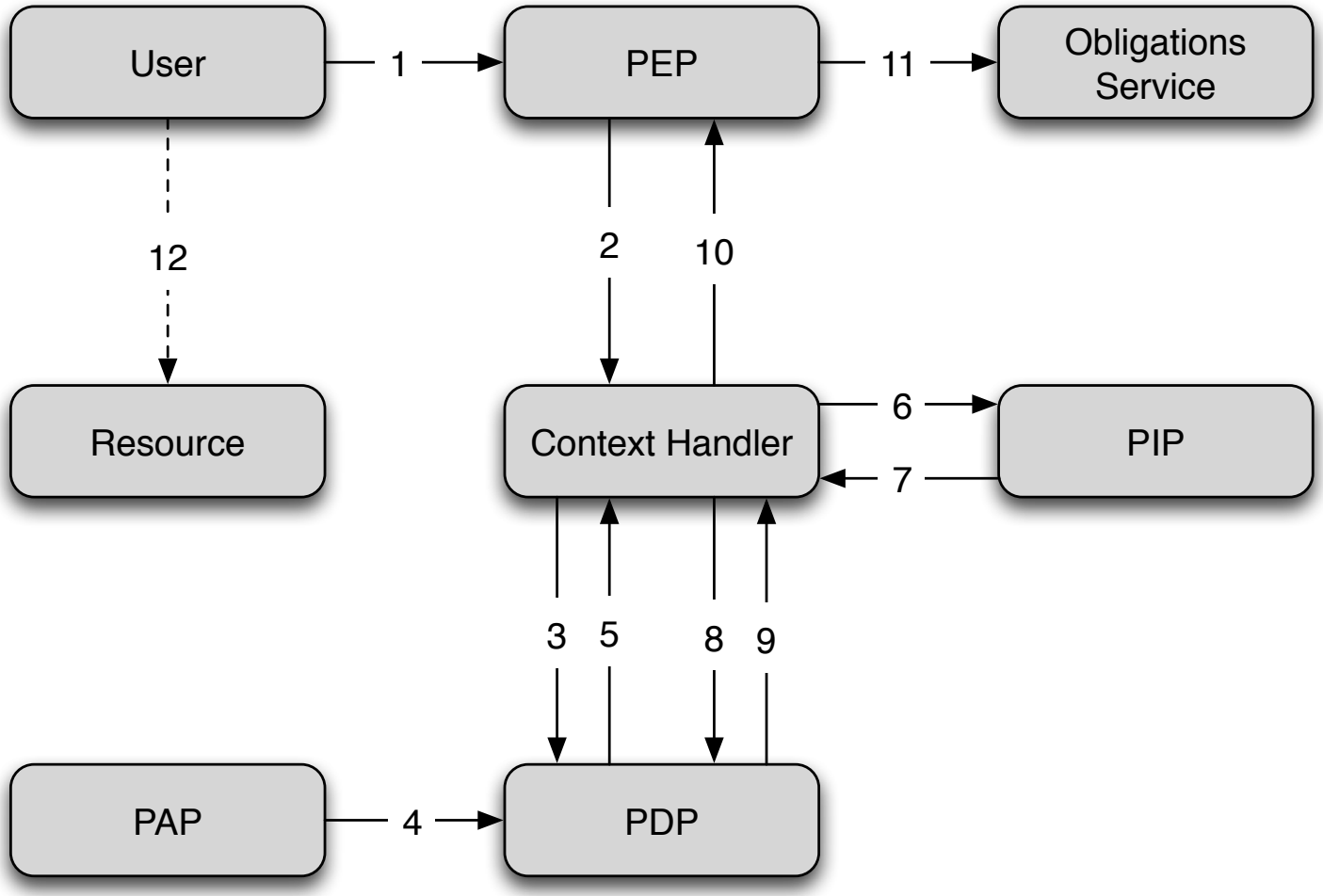
- A XML-based language for specifying access control policies
- A reference architecture for evaluating access requests with respect to the policies

XACML is intended to provide

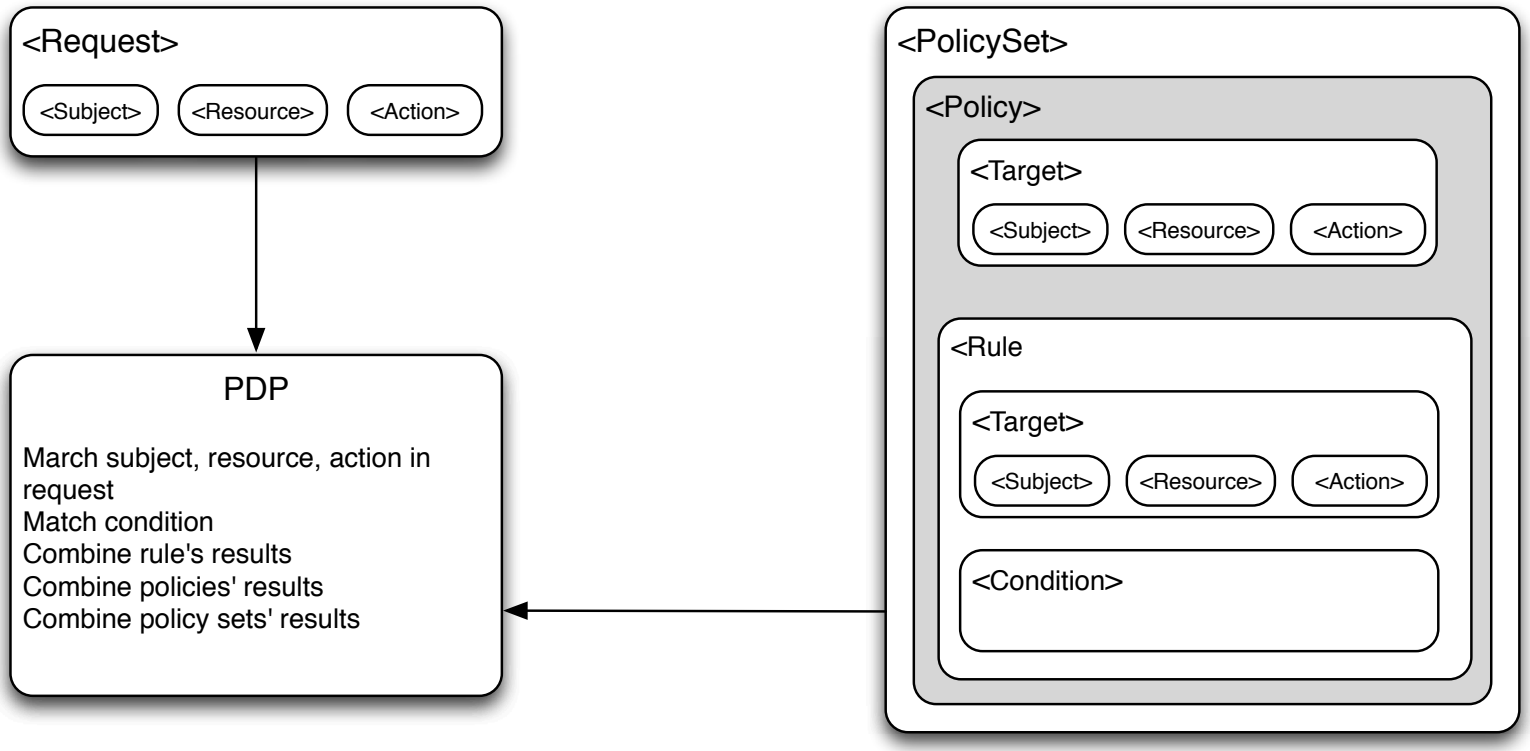
- Interchangeable policy format
- Support for fine- and coarse-grained access control policies
- Policy combination and conflict resolution
- Implementation independent

XACML 3.0 was approved as an OASIS standard in January 2013

XACML architecture



XACML building blocks



XACML rules

An XACML rule specifies an effect and a target

- The target identifies a subject, an object and an action
- The target is similar to an entry in a protection matrix
- The effect determines whether the rule permits or denies access for the target

A rule may specify an optional condition which further limits the applicability of the target

- May specify temporal or other environmental constraints on the use of the target

XACML policies and policy sets

An XACML policy is a set of rules

- The effect of a policy is determined by evaluating the rules and using a rule-combining (conflict resolution) algorithm
- An XACML policy could be used to specify an ACL
  - Each target would reference the same object

An XACML policy set is a set of policies

- An XACML policy set could be used to specify a set of ACLs
- The effect of a policy set is determined by using a policy-combining algorithm

An example

```
<Policy PolicyId="acl:project.doc">
  <Target>
    <Subjects><AnySubject/></Subjects>
    <Resources><Resource>word.exe</Resource></Resources>
    <Actions><AnyAction/></Actions>
  </Target>
  <Rule RuleId="ace:is-staff" Effect="Permit">
    <Target>
      <Subjects>
        <Subject>group-is-staff</Subject>
      </Subjects>
      <AnyResource/>
      <Actions><Action>write</Action></Actions>
    </Target>
  </Rule>
</Policy>
```

Outline

Requirements

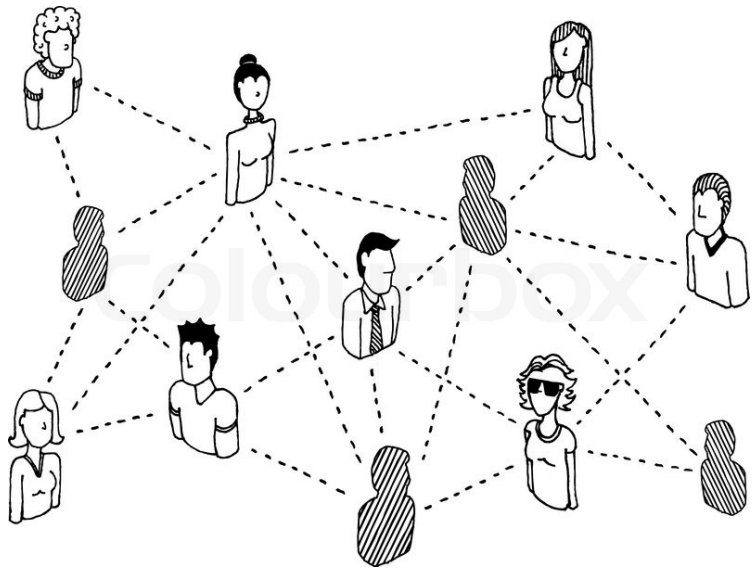
A personal data protection scheme must be

- Fine-grained
  - A user should have full control over how her data is shared
  - Sharing requirements can be as diverse as a person's data collection
- Flexible
  - Apply different sharing criteria for different friends
  - Deal with changing sharing conditions on a data object basis, a well as on a friend-by-friend basis
- Acceptable in terms of the amount of user effort required

Relationship-based access control

Relationship-based access control (ReBAC) is motivated by the protection requirements of social network systems

- The authorisation decision is based on whether the data owner and requester are in a particular kind of relationship (e.g. friend, GP)
- These relationships arise from
  - Subjective assessment of the users
  - The structure of trust that is inherent in the application domain



A simple ReBAC model

A simple ReBAC model contains the following components

- A set of users U
- A set of objects O
- A set of relation identifiers R
- A owner-object assignment function owner : $O \rightarrow U$
- A set of policy expressions E, where e has a grammar as

$$e = r; r \mid r^* \mid r_1 \cup r_2 \mid r_1 \cap r_2$$

- A object-policy assignment function poly : $O \rightarrow E$

A simple ReBAC model

A simple ReBAC model contains the following components

- A set of users U
- A set of objects O
- A set of relation identifiers R
- A owner-object assignment function owner : $O \to U$
- A set of policy expressions E, where e has a grammar as

$$e = r; r \mid r^* \mid r_1 \cup r_2 \mid r_1 \cap r_2$$

- A object-policy assignment function poly : $O \to E$

An access request (u, o) is granted if $(\text{owner}(o), u) \in \text{poly}(o)$

Limitations of ReBAC

- All friends are equal
  - Assume that all friends at a particular hop distance are equal
  - Need the flexibility of differentiating among the friends
- Omniscient users
  - Assume that users are all knowing about their social neighbourhood and able to make the appropriate protection decisions
  - This assumption is not practical in large and dynamic social neighbourhoods
- No impact on friendships
  - Assume that access control decisions do not impact the topology of the social network
- The possibility of inference is ignored

Further Reading

- M.A. Harrison, W.L. Ruzzo and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8), 461–471, 1976.
- R.S. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2), 38–47, 1996.
- OASIS. The XACML 3.0 standard. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`.
- P.W.L. Fong. Relationship-based access control: protection model and policy language. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, 191–202, 2011.