

Computer Security

Access Control

1

Authorisation

- authentication established who you are
- authorisation establishes what you are allowed to do

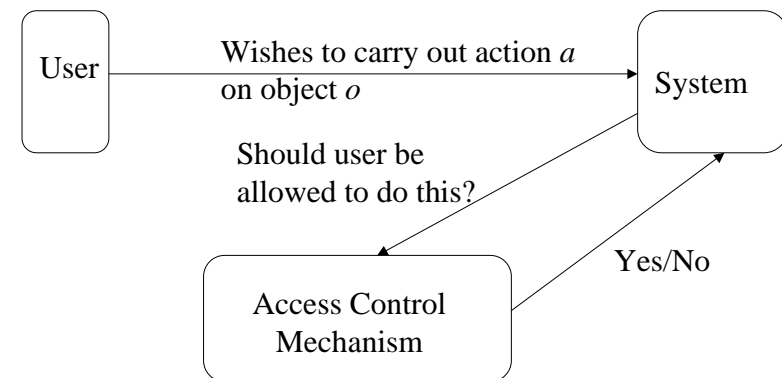
2

Access Control

- system entities attempt operations on other system entities
- access control is, basically, a yes/no question
- is subject s allowed to carry out operation m on object o ?

3

Access Control



4

Policy and Mechanism

- policies are the rules you want obeyed
- mechanism is used to express rules
- most mechanisms will affect what rules can be conveniently expressed

5

Basic Elements

- identity of subjects (eg., users)
- identity of objects (eg., files)
- access rights each subject has for each object

6

Authorisation Information

- Access Control Matrix
- Access Control Lists
- Capabilities
- Label based access control
- Roles

7

Access Control Matrix

- basic model
- most current access control implementations based on models derived from the ACM

8

Access Control Matrix

	bob	judy	jeff	allan
memo	rw		r	r
script		rwX		rw
progl	rwX			
editor	rwX	rwX		
letter	rw	rw	rwX	

9

Access Control Matrix

	bob	judy	jeff	allan
memo	rw			r
script		rwX		rw
progl	rwX			
editor	rwX	rwX		
letter	rw	rw	rwX	
bob	rwX	rw		
judy	rw	rwX		
jeff			rwX	r
allan	rwX			rwX

10

Confinement Problem

- access matrix does not prevent information being propagated
- this is called the confinement problem and is, in general, unsolvable
- essentially, if a process can read information, it can copy it and provide unlimited access to the copy

11

Implementation of the Access Matrix

- store the entire array
- store the rows with subjects
- store the columns with objects

12

Entire Array

- not implemented in practice
- too many entries in matrix
- many will be empty or the same
- more common mechanisms are
 - access control lists
 - capabilities
 - roles
 - lattice based access control

13

Access Control Lists

```

rwx---rwx   file1
  
```

fill it out with real directory contents (you should have seen enough by now)

14

Access Control Lists

- explicitly hold subject identity, object identity, rights
- usually rights drawn from a small generic set (such as r,w,x)
- could be a list with an entry for each subject that actually has some rights over object
- not often implemented this way

15

Groups

- often a number of users will have identical access to one or more objects
- groups are a useful concept
- give groups an entry in ACL
- some systems allow only one group per object

16

Groups

- can have entry for each users in an ACL
- leads to ACL's being too long
- can group users together

Groups

- how to manage groups across machines?
- can centrally manage groups
- can include group information in certificate (or ticket)

Groups

- first approach can be inefficient
- second approach has problems, as number of groups potentially unlimited

Groups

- must ensure that CA is allowed to assert specific group membership
- if individual is removed from group, certificate must be revoked
- group membership usually updated on-line, CA's off-line

Hierarchical Groups

- can allow groups to be members of other groups
- still have problem of how to list all groups of which a user is a member

21

Discussion

- access control lists provide users with the ability to easily specify which domains can access their objects
- determining the complete set of access rights enjoyed by a particular domain is not an easy task
- searching a long access list may not be efficient

22

Delegation

- sometimes you want something to act on your behalf
- For example
 - accessing remote files
 - Or using a printer
- inconvenient to have to explicitly log in at each resource

23

ACL's?

- could add every subject to every ACL for every resource they need to access
- and delete them after their need to access on your behalf is gone
- also inconvenient

24

Delegation

- need a mechanism to allow something to act on your behalf
- known as delegation or authentication forwarding

25

Delegation

- generate a special message
 - signed by you
 - specifies to whom you are delegating the rights
 - the rights being delegated
 - for how long
- last is important
- rarely if ever is delegation unlimited

26

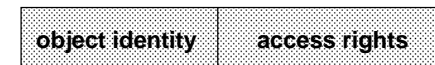
Signing

- with public keys, easy
- with KDC, ask KDC to construct message
- delegated entity can not decrypt, but can give it back to KDC to prove delegation

27

Capabilities

- represent the operations possible for each subject
- are associated with a subject
- are system protected entities
- capability structure



28

Capabilities

- make it easier to provide a user with multiple levels of access to an object
- integrate more easily into the type systems of programming languages
- allow naming and protection mechanisms to be unified

Capability Lists

- set of capabilities associated with subject or object
- usually on process level for subject
- if object then available to any process executing in object

Capabilities

- capability lists are useful for localising information for a subject and for determining the access rights of a subject
- it is not so easy to check the total access to an object
- when a capability is used the system need only check the access rights of the capability to determine if the operation is legal

Leakage

- capabilities may be copied from subject to subject
- difficult to track - as do not contain explicit subject identification
- how to revoke access?
- especially if do not know who has access

Capability Hierarchies

- “root” capability with each object
- all capabilities for an object form a hierarchy
- derived capabilities have equal or less rights
- can access capabilities derived from a capability
- can prune hierarchy

33

Access Control Lists and Capabilities

- some systems use both capabilities and access lists
- when an operation is attempted without a capability, the access list is checked to see if it is allowed
- if it is, the operation proceeds, and a capability is generated to allow future use of that operation

34

Other Access Control Models

- models based directly on ACM do not reflect all real world situations
- for example, military security clearances
- some authorities differentiate between DAC (discretionary access control) and MAC (mandatory access control)

35

Discretionary Access Control

- users can set policies
- most ACM based models fit this description

36

Mandatory Access Control

- user do not set policies
- basic rules are write-up and read-down
- Bell-LaPadula Model
- can be used in addition to DAC

37

Information Flow

- information flows from one “object” to another
- objects identified by their security class

38

Labels

- all subjects and objects are given a security label
- the labels are ordered
- there are high and low values
- objects - security classification
- subjects - security clearances

39

Information Flow

- an information flow policy is a triple $\langle SC, \rightarrow, \oplus \rangle$ where
 - SC is a set of security classes
 - $\rightarrow : SC \times SC$ is a binary can-flow relation on SC
 - $\oplus : SC \times SC \rightarrow SC$ is a binary join operator on SC

40

Information Flow Policies

- all three components of an information flow policy are fixed
- security classes can not be created or destroyed dynamically
- the set of security classes is finite
- objects may be created/destroyed dynamically

41

Can Flow

- $A \rightarrow B$ means information may flow from security class A to security class B
- \rightarrow is a partial order on SC
- ie, \rightarrow is a reflexive, transitive anti-symmetric binary relation
- SC has a lower bound with respect to \rightarrow

42

Join

- $A \oplus B = C$ means that objects that contain information from security classes A and B should be labelled with security class C
- \oplus is a totally defined least upper bound operator
- $A \rightarrow A \oplus B$ and $B \rightarrow A \oplus B$
- if $A \rightarrow C$ and $B \rightarrow C$ then $A \oplus B \rightarrow C$

43

Dominance

- $A \geq B$ (A dominates B) if $B \rightarrow A$ but not $A \rightarrow B$
- A is more sensitive than B

44

Bell-LaPadula Model

- augments DAC with MAC to enforce information flow policies
- there is a discretionary access matrix D
- an operation must be authorised both by entries in the matrix D and by the MAC policy

45

Write Up/Read Down

- subject can only read from object if clearance \geq classification
- subject can only write to object if clearance \leq classification
- write up sometimes referred to as the star property (*-property)
- deal with information flows

46

Flows

- read access implies a flow from object to subject
- write access conversely implies a flow from subject to object

47

Modification

- unclassified subject can write secret object
- may overwrite
- so may require clearance = classification for write

48

Chinese Wall Lattice

- consider a consulting company
- consultants deal with confidential information from a number of clients
- may decide that a consultant should not access information from two banks or two oil companies

49

Intent

- new consultants have no mandatory controls over their access
- a consultant accesses information about bank A
- thereafter consultant is denied access to information about any other bank
- should last long enough to avoid conflict of interest

50

Conflict-of-Interest Classes

- company information categorised into mutually disjoint conflict of interest classes
- each company belongs to one conflict of interest class
- Chinese Wall policy requires a consultant be able to access information on one company per class (at most)

51

Labels

- n conflict of interest classes, $COI_1, COI_2, \dots, COI_N$, each with m_i companies
- So $COI_i = \{1, 2, \dots, m_i\}$
- each object labelled with companies from which it contains information
- eg., [bank A, oil company OC]
- labels such as [bank A, bank B, oil company OC] are contrary to policy

52

Labels

- define labels as an n element vector where each i_k is an element of COI_k or $i_k = \text{null}$ for $k = 1..n$
- So an object labelled $[i_{a1}, i_{b2}, \dots, i_{mn}]$ is regarded as (possibly) containing information from company i_{a1} of COI_1 and so on
- when the element of a label is null then an object so labelled cannot contain information from any object in that COI class
- $l_1 \geq l_2$ if l_1 and l_2 agree wherever l_2 is not null

53

Examples

- $[1, 3, 2] \geq [1, 3, \text{null}]$
- $[1, 3, 1] \geq [\text{null}, \text{null}, 1]$
- $[1, 3, 2]$ and $[1, 2, 3]$ are incomparable

54

High and Low

- label with all null elements corresponds to public information
- no natural system high label
- define special *syshigh* which dominates all others - may be needed for auditing or other administration

55

Join

- labels may be joined (compatible) if wherever they disagree one of them is null
- So $[1, \text{null}, 2]$ joined with $[1, 2, \text{null}]$ gives $[1, 2, 2]$
- comparable labels are compatible but not all compatible labels are comparable

56

Use

- users start with all null label
- consider system with two COI groups
- accessing information about company 1 in COI_1 makes label $[1, \text{null}]$
- if then access company 2 in group 2 label becomes $[1, 2]$
- operations which force a label to *syshigh* are prohibited

57

Use

- note label “floats up”
- this dynamically tracks user’s history
- user can read from anything that their label dominates
- can write to anything dominating their label - including *syshigh*

58

Separation of Duty

- two (or more) operations
- user allowed to perform only one
- static separation of duty
- dynamic separation of duty
 - role based
 - object based
 - operational
 - history based

59

MAC and DAC

- MAC arises from rigid environments, such as military
- classical DAC from co-operative, autonomous environments, such as academia
- neither necessarily satisfies need of commercial enterprises

60

Role Based Access Control

- regulates access based on users' activities
- instead of individual specification of access, specification is based on job descriptions
- role can be defined as a set of actions and responsibilities associated with a particular working activity

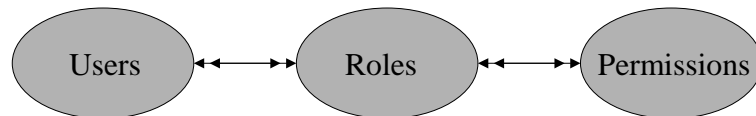
61

Role Based Access Control Model

- classic RBAC model consists of
 - users
 - roles
 - permissions

62

Classic RBAC Model



63

Permissions

- in simple terms identify an object and the allowed access to it
- one permission may be mapped to many roles
- a permission may allow access to more than one object
- permissions in most systems are positive

64

Authorised Roles

- users will not be allowed to be members of all roles in a system
- there will be a subset for which they are *authorised*

65

Session

- when a user logs in they must have at least one role *active* for them to access anything
- the active roles will be a subset of the authorised roles
- some systems allow only one role to be active in a session or at any one time
- others allow any number

66

Role Hierarchies

- often one job description will encompass all the tasks of another job description
- or two job tasks will have a number of tasks in common
- it would be useful if we can derive roles from other roles

67

Role Inheritance

- in simple terms we can consider a hierarchy of roles
- senior roles inherit all the permissions of junior roles and have extra ones as well

68

Role Inheritance vs OO Inheritance

- is role inheritance OO inheritance?
- Classes in OO can inherit structure from other classes - methods, fields
- roles inherit values from each other, ie actual permissions

69

Limited Inheritance

- sometimes may not want all of the permissions of one role to be inherited by another
- we can split the first role into two - the part to be inherited and the part that is not
- some RBAC system allow partial inheritance

70

Constraints

- may not want someone to have two particular roles, either as active or authorised
- many RBAC systems allow *constraints* on what roles may be authorised or active (in a single session or all concurrent sessions)
- constraints can also be applied to role/permission mapping

71

Attributes

- consider the following situation
- a company has a number of warehouses
- clerks at each warehouse are responsible for tracking shipments at that warehouse
- they should not be able to access information for other warehouses
- how to express this using roles?

72

Possibilities

- could have a separate role for each warehouse
- better idea is to be able to identify location of users/objects
- more generally, attach attributes (such as location, security level, etc) and test for them in permissions

73

Separation of Duty in RBAC

- constraints on role membership allows static separation of duty
- common constraint is *mutually exclusive*
- dynamic separation of duty (other than role based) is harder

74

History

- how do I limit someone to accessing my object five times?
- with attributes introduced idea of tests in permissions
- can generalise this to meta-variables in the access control system

75

Roles vs. Groups

- similar concepts
- group users together in some sense
- groups do not directly own permissions
- usually no concept of groups inheriting permissions
- groups often less directly concerned with job description

76

Roles vs. Groups

- most importantly, no concept of sessions for groups
- so no dynamic activation and deactivation of permissions
- however, if you add inheritance and sessions to groups, you effectively get roles

Final Words

- RBAC management
- duties
- negative permissions