CSE 410/565 Computer Security
Spring 2021

Lecture 8: Access Control

Department of Computer Science and Engineering
University at Buffalo

Outline

- Access control principles
  - access control matrices
  - access control lists
  - capability tickets
- Types of access control
  - discretionary access control
  - mandatory access control
  - role-based access control
  - attribute-based access control

Access Control Basics

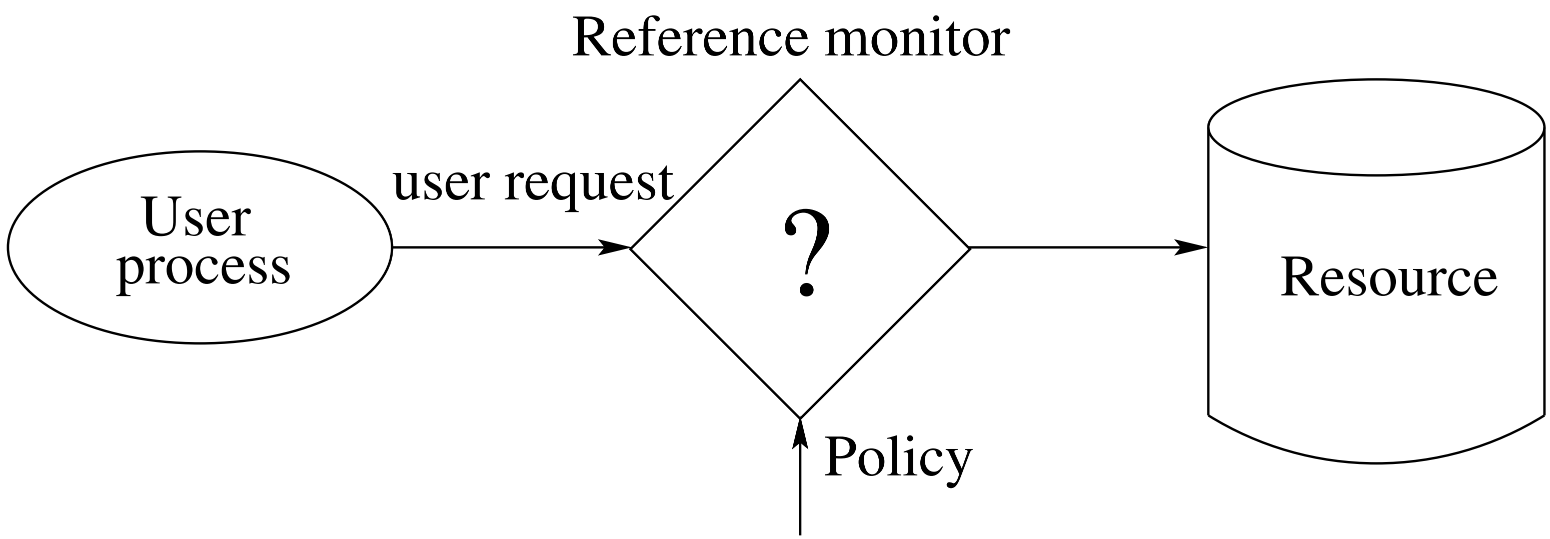
- What is access control?
  - prevention of an unauthorized use of a resource or use in an unauthorized manner
- In some sense, all of security is concerned with access control
- We look at a more specific notion of access control model
- An access control model specifies who is allowed to access what resource and what type of access is permitted
  - it may also specify when access is permitted
- What makes it hard?
  - interaction between different types of access

Related Security Concepts

- In a broader context, access control is related to the following concepts
  - authentication, identity and credential management
    - creation, maintenance, and verification of user or entity identity and/or credentials
  - authorization and information flow
    - granting rights or privileges based on established trust assumptions and imposing controls on information flow
  - audit and integrity protection
    - system monitoring to ensure proper use of resources and compliance with policies
    - detection of breaches in security and taking corresponding actions and/or making recommendations

Access Control Model Basics

- Reference monitor mediates access to resources
  - complete mediation means controlling all accesses to resources



Access Control Principles

- Least privilege
  - each entity is granted the minimum privileges necessary to perform its work
  - limits the damage caused by error or intentional unintended behavior
- Separation of duty
  - practice of dividing privileges associated with one task among several individuals
  - limits the damage a single individual can do
  - example:

Access Control Model Basics

- There is a set of resources or objects, O, to be protected
  - directories, files, devices, periferals, even facilities
- There is a set of subjects, S, that may obtain access to the resources
  - each subject can have a number of attributes (name, role, groups)
  - each subject is normally accountable for its actions
- Access right or privilege describes the type of access
  - read, write, execute, delete, search
- Access control requirements form rules
  - subject s has $read$ access to object o

Access Control Matrix

- The rules can be represented as an access control matrix
- Example

	Internal	Local	Long distance	International
Public	CRT			
Students	CRT	CRT	R	R
Staff	CRT	CRT	CRT	R
Administration	CRT	CRT	CRT	CRT

C = call, R = receive, T = transfer

- Often access control matrices are sparse and can instead be represented as access control lists (ACLs)

Access Control Lists

- In ACLs each object has a list of subjects authorized to access it and their types of access
  - for each object, a column of the access control matrix is stored
- Example of ACLs for previous system

Internal: $\boxed{\text{Public/CRT, Students/CRT, Staff/CRT, Administration/CRT}}$

Local: $\boxed{\text{Students/CRT, Staff/CRT, Administration/CRT}}$

Long distance: $\boxed{\text{Students/R, Staff/CRT, Administration/CRT}}$

International: $\boxed{\text{Students/R, Staff/R, Administration/CRT}}$

- Do Unix permission bits constitute ACLs?

Capability Lists

- With ACLs, it is hard to determine what privileges a subject has
- We can gather information about subject privileges in so-called capability lists
  - for each subject, store a row of the access control matrix
- Example

Public: $\boxed{\text{Internal/CRT}}$

Students: $\boxed{\text{Internal/CRT, Local/CRT, Long dist/R, International/R}}$

Staff: $\boxed{\text{Internal/CRT, Local/CRT, Long dist/CRT, International/R}}$

Administration: $\boxed{\text{Internal/CRT, Local/CRT, Long dist/CRT, Intl/CRT}}$

- Each user has a number of capability tickets and might be allowed to loan or give them to others

Access Control Triples

- To address drawbacks of all previous representations, we can have a table with (s, o, a) triples
  - is not sparse like access control matrices
  - sort by objects to obtain ACLs
  - sort by subjects to obtain capability lists

Subject	Access	Object
Public	C	Internal
Public	R	Internal
Public	T	Internal
Students	C	Internal
...	...	...
Administration	T	International

- This data structure is commonly used in relational DBMSs

ACLs vs. Capability Lists

- The choice of ACLs vs capability lists affects many aspects of the system
  - ACL systems need a namespace for both objects and subjects, while a capability ticket can serve both to designate a resource and to provide authority
  - procedures such as access review and revocation are superior on a per-object basis in ACL systems and on per-subject basis in capability systems
  - ACL systems require authentication of subjects, while capability systems require unforgeability and control of propagation of capabilities
- Most real-world OSs use ACLs

Discretionary Access Control

- In mandatory access control (MAC) users are granted privileges, which they cannot control or change

- Discretionary access control (DAC) has provisions for allowing subjects to grant privileges to other subjects
  - as a result, the access control matrix A can change

- Let triple (s, o, a) represent an access right

- At time i, the state X_i of the system is characterized by (S_i, O_i, A_i)

- Transition t_i takes the system from state X_i to X_{i+1}
  - a single transition $X_i \vdash_{t_i} X_{i+1}$
  - series of transitions $X \vdash^* Y$

Discretionary Access Control

- The access control matrix can be extended to include different types of objects
  - the subjects themselves can also be objects
  - different types of objects can have different access operations defined for them
    - e.g., stop and wakeup rights for processes, read and write access to memory, seek access to disk drives

	s_1	$\cdots$	s_n	o_1	$\cdots$	o_m	p_1	$\cdots$	p_ℓ
s_1									
$\cdots$									
s_n									

- For simplicity assume that we are dealing with one type of objects

Discretionary Access Control

- Suppose we have the following access rights
  - basic read and write
  - own: possessor can change their own privileges
  - copy or grant: possessor can extend its privileges to another subject
    - this is modeled by setting a copy flag on the access right
    - for example, right r cannot be copied, but r^* can
- Grant right gives rise to the principle of attenuation of privilege:
  - a subject may not give rights it does not possess
- Each particular model has a set of rules that define acceptable modifications to the access control matrix

Discretionary Access Control

- Primitive commands
  - create object o (with no access)
    - $S_{i+1} = S_i,\ O_{i+1} = O_i \cup \{o\},\ \forall x \in S_{i+1}, A_{i+1}[x, o] = \emptyset$, $\forall x \in S_{i+1}, \forall y \in O_i, A_{i+1}[x, y] = A_i[x, y]$
  - create subject s (with no access)
    - add s to the set of subjects and objects, set relevant access to $\emptyset$
  - add right r to object o for subject s
    - $A_{i+1}[s, o] = A_i[s, o] \cup \{r\}$, everything else stays the same
  - delete right r from $A_i[s, o]$
  - destroy subject s
  - destroy object o

Discretionary Access Control

- Building more useful commands
  - s creates object o
    - create object o with no access
    - add right own to object o for subject s
  - s adds right r to object o for subject s'
    - if ($r^* \in A_i[s, o]$ or $own \in A_i[s, o]$), then $A_{i+1}[s', o] = A_i[s', o] \cup \{r\}$
    - leave the rest unchanged
  - s deletes object o
    - if ($own \in A_i[s, o]$), then remove all access rights $\forall x \in S_i$ from $A[x, o]$ and destroy o

Discretionary Access Control

- Example: suppose we initially have

	s_1	s_2	o_1	o_2	o_3
s_1	own		$own, read^*$	$write$	$read, write$
s_2		own		$own, write$	own

  - subject s_1 creates s_3
  - s_1 grants to s_3 $read^*$ on o_1
  - s_3 grants to s_2 $read$ on o_1
  - can s_1 revoke s_2's right on o_1?

- Attenuation of privilege principle is usually ignored for the owner
  - why?

DAC in Unix File System

- Access control is enforced by the operating system
- Files
  - how is a file identified?
  - where are permissions stored?
  - is directory a file?
- Users
  - each user has a unique ID
  - each user is a member of a primary group (and possibly other groups)

DAC in Unix File System

- Subjects are processes acting on behalf of users
  - each process is associated with a uid/gid pair
- Objects are files and processes
- Each file has information about: owner, group, and 12 permission bits
  - read/write/execute for owner, group, and others
  - suid, sgid, and sticky
- Example

`rw-`	`r--`	`---`

```
user::rw-
group::r--
other:---
```

DAC in Unix File System

- DAC is implemented by using commands `chmod` and `chown`
- A special user "superuser" or "root" is exempt from regular access control constraints
- Many Unix systems support additional ACLs
  - owner (or administrator) can add to a file users or groups with specific access privileges
  - the permissions are specified per user or group as regular three permission bits
  - `setfacl` and `getfacl` commands change and list ACLs
- This is called extended ACL, while the traditional permission bits are called minimal ACL

Security of Discretionary Access Control

- What is secure in the context of DAC?
  - a secure system doesn't allow violations of policy
  - how can we use this definition?
- Alternative definition based on rights
  - start with access control matrix A that already includes all rights we want to have
  - a leak occurs if commands can add right r to an element of A not containing r
  - a system is safe with respect to r if r cannot be leaked

Safety of DAC Models

- Assume we have an access control matrix

	f_a	f_b	f_c
s_a	own, r, w	r	r
s_b	r	own, r, w	r
s_c	r	r	own, r, w

  - is it safe with respect to r?
  - is it safe with respect to w?
  - what if we disallow granting rights? object deletion?

- Safety of many useful models is undecidable
  - safety of certain models is tractable, but they tend not to apply to real world

Decidability of DAC Models

- Decidable
  - we are given a system, where each command consists of a single primitive command
  - there exists an algorithm that will determine if the system with initial state X_0 is safe with respect to right r
- Undecidable
  - we are now given a system that has non-primitive commands
  - given a system state, it is undecidable if the system is safe for a given generic right
  - the safety problem can be reduced to the halting problem by simulating a Turing machine
- Some other special DAC models can be decidable

Does Safety Mean Security?

- Does “safe” really mean secure?
- Example: Unix file system
  - root has access to all files
  - owner has access to their own files
  - is it safe with respect to file access right?
    - have to disallow chmod and chown commands
    - only “root” can get root privileges
    - only user can authenticate as themselves
- Safety doesn’t distinguish a leak from authorized transfer of rights
  - is this definition useful?

Security in DAC

- Solution is trust
  - subjects authorized to receive transfer of rights are considered "trusted"
  - trusted subjects are eliminated from the access control matrix
- Also, safety only works if maximum rights are known in advance
  - policy must specify all rights someone could get, not just what they have
  - how applicable is this?
- And safety is still undecidable for practical models

Mandatory Access Control

- In mandatory access control (MAC) users are granted privileges, which they cannot control or change
  - useful for military applications
  - useful for regular operating systems
- DAC does not protect against
  - malware
  - software bugs
  - malicious local users
- DAC cannot control information flow

MAC in Operating Systems

- The need for MAC
  - host compromise by network-based attacks is the root cause of many serious security problems
    - worm, botnet, DDoS, phishing, spamming
  - hosts can be easily compromised
    - programs contain exploitable bugs
    - DAC mechanisms in OSs were not designed to take buggy software in mind
  - adding MAC to OSs is essential to deal with host compromise
    - last line of defense when everything else fails
- In MAC a system-wide security policy restricts access rights of subjects

Combining MAC and DAC

- It is common to combine mandatory and discretionary access control in complex systems
  - modern operating systems is one significant example
- MAC and DAC are also combined in older models that implement multilevel security (for military-style security classes)
  - Bell-Lapadula confidentiality model (1973)
  - Biba integrity model (1977)
- Related models for commercial applications include
  - Clark-Wilson model
  - Chinese Wall model

Summary

- Access control is central in providing an adequate level of security
- Access control rights can be specified in the form of
  - access control matrix
  - access control lists
  - capability tickets
  - access control tables
- Types of access control
  - already covered DAC and MAC
  - will look at role-based access control (RBAC) and attribute-based access control