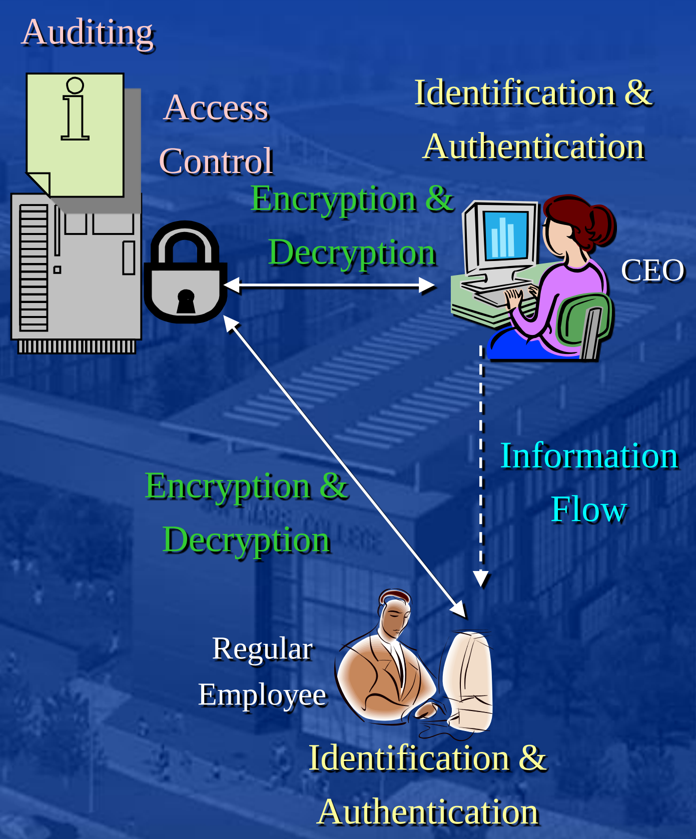
Security and Privacy

Prof. Jingsha He
School of Software Engineering
Beijing University of Technology

Project 2 Demo

Place: Software Building 505-506
Time: 8:30-11:30, Friday, Oct. 18
Other times: by appointment

Access Control


Auditing
Access Control
Identification & Authentication
Encryption & Decryption
CEO
Information Flow
Encryption & Decryption
Regular Employee
Identification & Authentication

Reading Material

- Matt Bishop
  - Chapter 13
  - Chapter 15

Design Principles

- Objectives
  - Simplicity
    - Easy to understand
    - Lower probability of making mistakes and incurring errors in the design and implementation
    - Lower chance of introducing inconsistencies within a security policy or across a set of security policies
  - Restriction
    - Limit the set of access privileges granted to an entity to just what are absolutely necessary

Principle 1: Least Privilege

- Statement
  - A subject should be given only those privileges that are absolutely necessary for the subject to complete its tasks or functions, the so-called "need-to-know" rule
- Requirements
  - Spatial: grant the privileges that are absolutely needed
  - Temporal: grant the privileges only when it is absolutely needed
- Challenges
  - Less granularity of privileges, e.g. write vs. append
  - Super user roles are a common practice, i.e., convenience vs. security
- Reality
  - Violations are not uncommon

Principle 2: Fail-Safe

- Statement
  - A subject should be denied access to an object unless access is explicitly granted
- Requirements
  - The default access right to an object should be "no access"
  - Access privileges should be checked prior to each and every access
- Challenge
  - Access to objects takes place in too many different ways and forms to enforce explicit security checks
    - Example: read an object vs. read the existence of an object
- Reality
  - Total compliance is very difficult, if not impossible

Principle 3: Economy of Mechanism

- Statement
  - Security mechanisms should be as simple as possible
- Requirements
  - Devise and implement simple and effective security control
- Challenges
  - The lack of coherence and trust among system components often results in more security checks than it is necessary, many of which are boundary checks
  - The lack of security assurance in the underlying system often results in exhaustive security checks to be carried out to ensure effectiveness
- Reality
  - Lack of coherence in system/network design and implementation makes it difficult to achieve simplicity in security mechanisms

Principle 4: Complete Mediation

- Statement
  - Access to objects should be explicitly checked to make sure that it is allowed
- Requirements
  - Subsequent accesses to an object should be explicitly checked again regardless of the result of precious access checks
  - Caching of data should not be allowed because it often leads to omission of subsequent security checks
- Challenges
  - Caching is commonly used as an effective way of improving system performance and resource utilization
  - Objects that contain the same information are not treated uniformly with regard to security checks
- Reality
  - Performance is usually given a higher priority than security in commercial systems

Principle 5: Open Design

- Statement
  - The security of mechanisms should not depend on the secrecy of its design or implementation
- Requirements
  - The algorithm of a security mechanism or cryptographic method should be open
  - Security of a mechanism or cryptographic method should not assume that insufficient knowledge is available to the general public
- Challenge
  - Develop security mechanisms with their effectiveness totally independent of the design and implementation
- Reality
  - Details of system design and implementation are often prohibited from being disclosed to the general public for commercial reasons
  - Few except the most highly secure systems maintain the secrecy of the design and implementation as an additional measure to ensure security

Principle 6: Separation of Privileges

- Statement
  - A security system should avoid authorizing access to an object based only on a single condition
- Requirements
  - Multiple, separate conditions should be established as the basis of granting access to an object
  - The multiple conditions should be independent from one another, i.e., the truth of one condition should not automatically lead to that of another
- Challenge
  - Enforcing separation of privilege may result in unnecessary complexity that degrades performance and contradicts to the “principle of economy of mechanism”
- Reality
  - Little attention has been paid to enforcing this principle except when it is naturally the case in the design and implementation

Principle 7: Least Common Mechanism

- Statement
  - Mechanisms used to access system resources should not be shared
- Requirements
  - Sharing of system resources should be prohibited or highly restricted
  - The status of the availability of system resources should be carefully guarded
- Challenge
  - Sharing of system resources is the most common method of improving the utilization and performance of system resources in computer systems and networks
- Reality
  - Performance and utilization often take a higher priority than security in most commercial systems
  - Security goals are often achieved through some other means to minimize negative impact on system performance and resource utilization goals

Principle 8: Psychological Acceptability

- Statement
  - Security mechanisms should not make access to system resources more difficult than if these mechanisms were not in place
- Requirements
  - Security checks should be made as transparent to the user as possible
  - Security checks should return sufficient information to the user, especially when failure occurs
- Challenge
  - Security system design and implementation should actively employ the results from user psychological studies
- Reality
  - Users often prefer convenience over security except when their personal interest is at stake
  - Users would be very likely to take the steps to avoid or disable excessive security for ensuring system security

Access Control Matrix Model

- Row index: subject
- Column index: object

Objects

Subjects		$File_1$	$File_2$	$File_3$	$File_4$
	Bob	own	read write		exe
	Alice	write	own		
	John	read write		write	own

Access Control Implementation

- Straightforward method
  - Access control matrix
- Implementation challenges and problems
  - Matrix is often too large
    - Due to the large number of subjects and objects
  - Most matrix entries are either blank or the same
    - Due to system-provided default access privileges
  - Matrix management could be very complex
    - Due to frequent additions and deletions of subjects and objects
- Conclusion
  - Optimization is highly desirable

Access Control Methods

- Access control lists
  - Access control matrix broken by the column
  - Each column records the access rights of each and every of the subjects to a single object
  - Access control is located on the side of the objects
  - The most popular method
- Capabilities
  - Access control matrix broken by the row
  - Each row records the access rights of a single subject to each and every of the objects
  - Access control is located on the side of the subjects
- Locks and keys
  - The sharing of secret

Access Control Lists (ACLs)

- Method
  - Access control matrix broken by the column
  - Each object is associated with an access control list
  - Subjects are the targets of examination in the access control lists during access control
- Definition of ACL
  - S: a set of subjects
  - O: a set of objects
  - R: a set of access rights
  - acl(o) = {(s, r), where $s \in S$ and $r \subseteq R$}
    - The access control list (ACL) that is associated with object $o \in O$
- Default permissions
  - Access request from a subject to access an object is denied if the subject doesn’t appear in the ACL for the object

An Example

- ACLs
  - acl($File_1$) = {(Bob, {own}), (Alice, {write}), (John, {read, write})
  - acl($File_2$) = {(Bob, {read, write}), (Alice, {own})}
  - acl($File_3$) = {(John, {write})}
  - acl($File_4$) = {(Bob, {exe}), (John, {own})}
- Default permissions
  - Bob has no access right to $File_3$
  - Alice has no access right to $File_3$ and $File_4$
  - John has no access right to $File_2$

Objects

Subjects	$File_1$	$File_2$	$File_3$	$File_4$
Bob	own	read write		exe
Alice	write	own		
John	read write		write	own

ACM

Abbreviation of Access Control Lists

- Motivation
  - For performance and simplicity in implementation and in performing access control
- Methods
  - Subjects may be grouped together
  - Subjects may be classified into different categories
- Consequence
  - Less granularity in access control
    - A typical example of favoring performance and simplicity over security in commercial systems

Access Control Lists in UNIX

- Subjects
  - owner, group, other
- Access rights
  - r (read), w (write), x (execute)
- Access control list: 9 bits
  - $\underbrace{\text{rwx}}_{\text{owner}}\underbrace{\text{rwx}}_{\text{group}}\underbrace{\text{rwx}}_{\text{other}}$
- Example: rw-r-----

Access Control Lists in AIX

- Method
  - An abbreviated ACL as the default access control
  - An explicit ACL to override the default as needed
- Examples
  - Bob: rw-
  - Alice: rw-
  - Alice/adm: r--
  - John/sys: rw-

```
attributes:
base permissions
    owner(Bob):   rw-
    group(sys):   r--
    others:       ---
extended permissions enabled
    specify       rw-     u:Alice
    permit        -w-     u:John,  g=sys
    deny          -w-     u:Alice, g=adm
```

Access Control Lists in Windows

- Objects
  - Files and directories
- Access rights
  - read, write, execute, delete
  - change the permissions of
  - take the ownership of
- Generic rights for files
  - *no access*: cannot access
  - *read*: read or execute
  - *change*: read, execute, write, or delete
  - *full control*: all rights
  - *special access*: assignment of any of the rights

Access Control Lists in Windows

- Generic rights for directories
  - *no access*: cannot access the directory
  - *read*: read or execute files in the directory
  - *list*: list contents of the directory and make change to a subdirectory in that directory
  - *add*: create files or subdirectories in the directory
  - *add and read*: combine *read* and *add*
  - *change*: create, read, execute, or write files and delete subdirectories in the directory
  - *full control*: all rights over files and subdirectories in the directory
  - *special access*: assignment of any combinations of the rights

Access Control Lists in Windows

- Rules of granting access permissions
  - User name and group not in ACL → denial
  - Any entry that explicitly denies access → denial
  - User name in the ACL and/or belonging to one or more groups → union of all the rights assigned to the user name and to the groups
- Example
  - students = {Bob, Alice}
  - staff = {Alice, John, Peter}
  - Directory c:\staff has the following access control list
    - ACL(c:\staff) = {(staff, {add}), (Peter, {change}), (students, {no access})}
  - Access control results
    - Bob: no access to c:\staff
    - Alice: no access to c:\staff
    - John: create subdirectories or files in c:\staff
    - Peter: create, read, execute, or write files in c:\staff; also create and delete subdirectories in c:\staff

Issues for Access Control Lists

- Modification of an ACL
  - The owner who created the object
- Rights of the privileged (super) user
  - *root* in UNIX and *administrator* in Windows
  - ACLs are usually ignored
- Support for groups and wildcards
  - To limit the size of ACLs
- Conflicts
  - Explicit denial approach: any entry that denies access, e.g., Windows ACLs
  - First match approach: access rights in the first matching entry, e.g., Cisco routers
- Explicit ACL entries and default permissions
- Revocation of access rights
  - Cascading of the revocation of access rights
  - Multiple instances of granting the same access rights

Capabilities

- Method
  - Access control matrix broken by the rows
  - Each subject is associated with a capability list
  - Objects are the targets of examination in the capabilities during access control
- Definition
  - S: a set of subjects
  - O: a set of objects
  - R: a set of access rights
  - $c = \{(o, r), o \in O \text{ and } r \subseteq R\}$: a capability list
  - cap: a function that determines the capability list c associated with a subject
    - $c = cap(s) = \{(o_i, r_i), 1 \leq i \leq n\}$: subject s may access o_i using right r_i
- Default permissions
  - Access request from a subject to access an object is denied if the object doesn't appear in the capability list for the subject

An Example

- Capabilities
  - cap(Bob) = {($File_1$, {own}), ($File_2$, {read, write}), ($File_4$, {exe})}
  - cap(Alice) = {($File_1$, {write}), ($File_2$, {own})}
  - cap(John) = {($File_1$, {read, write}), ($File_3$, {write}), ($File_4$, {own})}
- Default permissions
  - Bob has no access right to $File_3$
  - Alice has no access right to $File_3$ and $File_4$
  - John has no access right to $File_2$

Objects

Subjects	$File_1$	$File_2$	$File_3$	$File_4$
Bob	own	read write		exe
Alice	write	own		
John	read write		write	own

ACM

Implementation Issues for Capabilities

- Determination of capabilities
  - Rely on the operating system or a security kernel to return the capability list for a subject
- Protection of capabilities
  - Rely on the operating system or a security kernel to protect the memory areas that store capabilities
  - Rely on cryptographic techniques to protect the integrity of capabilities
- Copying of capabilities
  - A copy flag is usually used that is under the control of the operating system or a security kernel
- Revocation of rights
  - Invalidate the table entries that store the capabilities to objects

Locks and Keys

- The concept
  - A piece of information called "the lock" is associated with each object
  - A piece of information called "the key" is held by the subjects
  - Access request from a subject to an object is granted if the subject holds the set of keys that correspond to the set of locks associated with the object
- One implementation is based on cryptography
  - Lock: encryption with a cryptographic key
  - Key: decryption with the same or a different cryptographic key
  - Or-access: $o' = (E_1(o), E_2(o), \ldots, E_n(o))$ (1 out of n)
  - And-access: $o' = E_n(\ldots(E_2(E_1(o)))\ldots)$ (n out of n)

General Secret Sharing Question

- Definition
  - A (t, n)-threshold scheme is a cryptographic scheme
    - A data item is divided into n parts
    - The availability of any t out of the n parts is sufficient to recover the original data item
    - The n parts are called the shadows of the data item
- Application: protecting cryptographic keys
  - The decryption key is divided into n parts (shadows) and assigned to n different parties
  - The availability of any t parts from the n parties can restore the decryption key

A Secret Sharing Method

- Based on a Lagrange interpolating polynomial of degree t-1
  - ◆ $P(x) = (a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \ldots + a_1x + a_0) \bmod p$
- Constants and numbers in the polynomial
  - Set the constant a_0 to be the shared secret S
    - ◆ $a_0 = S$; consequently, $P(0)=S$
  - Choose p to be greater than both S and n
    - ◆ $p > S$ and $p > n$
  - Choose $a_1, a_2, \ldots, a_{t-2}, a_{t-1}$ arbitrarily
- P(1), P(2), …, P(n) are the n shadows

An Example

- Problem and requirement
  - S = 5: the secret key to be shared
  - t = 3, n = 5: a (3, 5)-threshold scheme
- Constants and numbers
  - Let p=6 (p > S and p > n)
  - Let $a_2=1$, $a_1=2$, $a_0=S=5$
  - Therefore, $P(x) = (x^2 + 2x + 5) \bmod 6$
- The shadows
  - P(1) = (1 + 2 + 5) mod 6 = 8 mod 6 = 2
  - P(2) = (4 + 4 + 5) mod 6 = 13 mod 6 = 1
  - P(3) = (9 + 6 + 5) mod 6 = 20 mod 6 = 2
  - P(4) = (16 + 8 + 5) mod 6 = 29 mod 6 = 5
  - P(5) = (25 + 10 + 5) mod 6 = 40 mod 6 = 4
- The 5 shadows are assigned to 5 parties that are guarded independently
- Availability of any 3 out of the 5 shadows should recover the original shared secret key, i.e., P(0) = S = 5

Recovery of the Shared Secret

- The recovering polynomial
  - Let $k_r = P(x_r)$
  - $P(x) = \sum^{t}_{i=1} k_i \prod^{t}_{j=1, j\neq i}((x-x_j)/(x_i-x_j)) \bmod p$
- One instance
  - 3 shadows P(1), P(2) and P(3) are used to recover the secret
    - $x_1=1$, $x_2=2$, $x_3=3$
    - $k_1=2$, $k_2=1$, $k_3=2$
  - P(x)

$$= (2(x-2)(x-3)/(1-2)(1-3)+1(x-1)(x-3)/(2-1)(2-3)+2(x-1)(x-2)/(3-1)(3-2)) \bmod 6$$
$$= ((x-2)(x-3)-(x-1)(x-3)+(x-1)(x-2)) \bmod 6$$
$$= ((x^2-5x+6)-(x^2-4x+3)+(x^2-3x+2)) \bmod 6 = (x^2-4x+5) \bmod 6$$

- The recovered secret key
  - $S = P(0) = 5 \bmod 6 = 5$

Summary

- Design principles
  - 8 principles
  - Far from being strictly followed in reality
- Access control mechanisms
  - Access control lists
  - Capabilities
  - Locks and keys
    - The (t, n)-threshold scheme for secret sharing

Thought of the Lecture

- In procedural programming, there are local and global variables; in the object-oriented paradigm, they are called private and public objects. Does this have anything to do with access control? How is access granted? Is there any authentication?

Q & A