Outline

- Information Protection
  - Information Protection in Computer Systems (Cont’d)
- Q&A
Access Control List System

- Use of an Access Controller
  - Components
    - Addressing descriptor
    - Access control list
  - Characteristics
    - Kind of indirect address
    - Possibly replacing protection descriptors with unprotected pointers
  - Data Reference
    - Using the pointer to address the access controller
    - Searching the access control list
    - Searching the permission bits
    - Generating a reference request if permitted
Access Controller Model
Revised Separating Organization
Differences with Capability Systems

- Using the Pointer via the Access Controller
  - Preventing Unauthorized Access

- Using the Access Control List
  - Authorizing Possibly on Every Access

- Updating the Access Control List
  - Making Access Revocation Manageable

- Examining the Access Control List
  - Figuring Out Authorized Users

- Using Different Access Control Lists
  - Differing Based on the Purposes
System with Access Control Lists
Protection Groups

Definition
- Principals Used by More Than One User
  - Members Permitted to Access the Segment

Motivation
- Possibly Very Long List
- Frequently Changing List

Simple Method
- Extending the “Principal Holding” Register
  - One for a personal principal ID
  - Another for each protection group

Protection Group List Must Be Controlled Systematically
Problems & Solutions

- Several Memory References
  - Shadow Capability Register for Each Pointer Register
    - Loaded with a capability consisting of copies of the addressing descriptor and protection bits
    - Whenever a pointer register is first loaded
    - Not affected by changing an access control list (a minor change in revocability properties)

  Clearing Shadow Registers and Triggering Their Reloading
Shadow Capability Registers
Problems & Solutions (Cont’d)

- Allocation and Search of an Access Control List
  - Constraining All Access Control Lists to Contain a Certain Number of Entries
    - E.g., three entries on each access list
      - User
      - Group
      - Universal Group
Authority to Change Access Control Lists

- Self Control
  - Including Permission to Modify the Access Control List
    - Allowing the creator to adjust the list
  - Not Permitting Graceful Changes of Authority
Self Control Scheme

access controller

base

Smith  read  write  ACL-mod
Jones  read  write
Doe    read
...

bound

addressing descriptor for this segment

access control list

principal identifiers

permissions

...
Authority to Change Access Control Lists (Cont’d)

Hierarchical Control

- Including Permission to Modify the Lists of Lower Level Access Controllers
  - Allowing a node to change lists further down the hierarchy
- Permitting Access to Anything in the Entire Subtree
  - Giving too much power to high-level nodes, leading to concentration of authority and possible (unchecked) abuse of higher level authority

Higher Level Checking and Taking Actions Based on the Prescript Field: e.g., Logging the ID and Delaying the Change
Hierarchical Control Scheme
Discretionary and Nondiscretionary Controls

■ Discretionary Control
  □ Creator’s Authorization of Access to What Is Created
    ■ Against the principle of least privilege

■ Nondiscretionary Control
  □ Authorization Not Limited by “Employees”
    ■ Similar to Isolated Compartments and Sensitivity Levels

Confinement: No Sharing with Other Domains
Protecting Objects

Operations On Objects, Working Out Permissions for These Objects

- Type Field Added to a Capability
  - Special instructions requiring different operand capabilities
  - Interpretation of the permission bits different for types
  - Extended “create” operation to permit type specification

Examples of Type Extension in Capability System: Any Data Structure Including Queues, I/O Streams Attached to Terminals, Printers, and the Like

- Type Field Placed in the Access Controller
## Typical Protected Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Typical Separately Permissible Operations</th>
</tr>
</thead>
</table>
| Data segment                                | READ data from the segment  
|                                             | WRITE data into the segment  
|                                             | Use any capability found in the segment  
|                                             | Write a capability into the segment  |
| Access controller                           | Read access control list  
|                                             | Modify names appearing on an access control list  
|                                             | Modify permissions in access control list entries  
|                                             | Destroy objects protected by this access controller  |
| FIFO message queue                          | Enqueue a message  
|                                             | Dequeue a message  
|                                             | Examine queue contents without dequeueing  |
| Input/Output                                | READ data  
|                                             | WRITE data  
|                                             | Issue device-control commands  |
| Remove recording medium (e.g. magnetic tape reel) | READ data  
|                                             | WRITE over data  
|                                             | WRITE data in new area  |

Extending the Range of Protected Objects by Using the Type Field Containing the Additional Unique ID
Protected Objects and Domains

Protected Subsystems

Definition

- Collection of encapsulated program and data segments
  - Preventing other programs from reading/writing them
  - Preventing other programs from disrupting the intended operations
  - Allowing other programs to call the programs by calling the designated entry points

Motivation

- Limitations of using objects independently controlled such as reading, writing, or executing as programs
  - Not being able to define a new type dynamically
  - Danger presented by executing a borrowed program
Illustration: A Protected Subsystem

Executing the Borrowed Program Outside the Protected Subsystem Domain
Implementation of Protected Subsystems

- **Associating Multiple Domains with a Single Computation (with Asynchronous Activities)**
  - Use of a Separate Virtual Processor
    - One with its own domain
    - Another for each protected subsystem

- **Using a Single Virtual Processor**
  - Association of multiple domains with a single virtual processor

Changing Domains: Changing Principal IDs in ACL Systems, and Changing the Set of Capabilities of the New Domain in Capability Systems; e.g., Including the ENTER Permission Possibly with the Domain ID
Other Considerations

- Domain Switching
  - (Restricted) Role of the Supervisor
    - Having only the Enter capability leading to an authentication program
  - Using the Seal Capability (with a Seal in the Type Field)
    - Accessing internal data (Separation of Privilege)
  - Distinction between Dynamic and Static Data
    - Using multiple virtual processors
  - Passing of Arguments
    - Controlling specially the argument indicating how to return
Other Considerations (Cont’d)

- Precise Model of Protection Goals
  - Isolation vs Sharing
- Verifying the Implementation
  - Establishing the Correctness Formally
- Providing Need-Specification Interface
- Providing System S/W with Constraints
  - Establishing a Set of Protected Functions
- Making Protection Not Affected by Failure
- Constraining After-Release Use
- Communicating Keys to Authorized Users