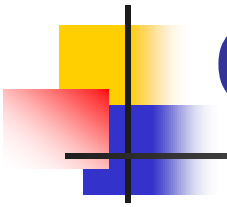
CS590U
Access Control: Theory and Practice

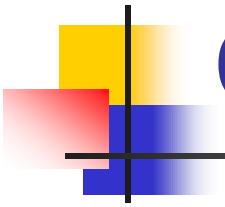
Lecture 17 (March 21)

Capability-Based Systems

Overview of UNIX Access Control



- Based on user/group id of the process
- Child process inherits parent's process
- Setuid
- Confining processes
  - chroot
  - jail
  - DTE
  - system call interception



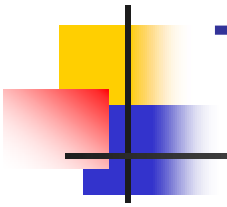
Capability vs. ACL

- ACL
  - an access control list is associated with each object
- Capabilities
  - a list of capabilities is associated with each subject

The Confused Deputy

N. Hardy

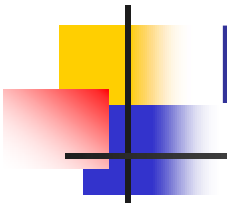
In Operating Systems Review, 1988.



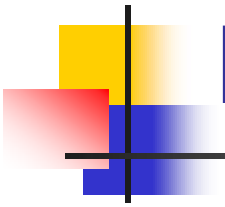
The Confused Deputy Problem

- The compiler program is SYSX/FORT.
- Other files under SYSX include STAT and BILL.
- The compiler program needs to write to files in SYSX directory, so it is given authority to write to files in SYSX.
- A user who runs SYSX/FORT can provide a file name to receive output info.
- A malicious user may use SYSX/BILL as the output name, resulting in billing info being erased.

Analysis of The Confused Deputy Problem

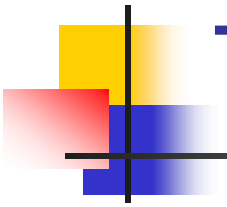


- The compiler runs with authority from two sources
  - the invoker
  - the system admin (who installed the compiler and controls billing and other info)
- It is the deputy of two masters
- There is no way to tell which master the deputy is serving when accessing a piece of resource



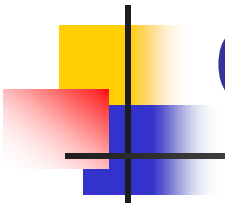
More Analysis

- Compare with setuid in UNIX and the associated security problems
- Compare with the Trojan horse problem
- How can this problem be solved?



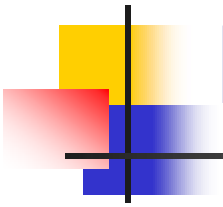
The Capability Approach

- The compiler program is given capabilities to access SYSX/STAT and SYSX/BILL, which are stored in capability slots 1 & 2
- When the invoker runs the compiler program, it gives a capability to write to the output file, which is stored in capability slot 3. The invoker cannot give a capability for SYSX/BILL if it doesn't have the capability.
- When writing billing info, the program uses capability in slot 2. When writing the output, it uses capability in slot 3.



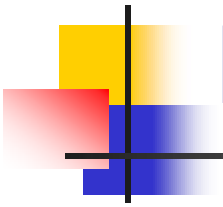
Overview of KeyKOS

- A capability-based microkernel operating system
- A message-based system
  - objects call other objects by sending a key-addressed message



Basic Concepts in KeyKOS

- Domains
  - Similar to processes in UNIX
  - A domain has 16 general slots and several special slots (e.g., address slot)
  - A domain is an object and may be identified in a gate key
- Keys (capabilities)
  - A key designates a specific object and certain authority over the object



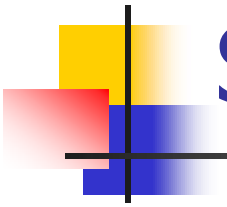
Domains Calling Domains

- When one domain calls another domain
  - The calling domain identifies a general slot and invoke the key in it (should be a gate key)
  - The calling domain may add other keys to be passed to the called domain
  - The called domain receives a message, which include the keys chosen by the calling domain, and in addition, a resume key, implicitly generated by the system

The KeyKOS Microkernel

- It provides
  - several types of primitive objects
  - multiprogramming and scheduling support
  - single-level store. Domains are unaware of the distinction between main storage and disk
  - virtual memories for domains
  - gate keys by which messages are sent between domains
  - an invariant interpretation of keys
  - ……

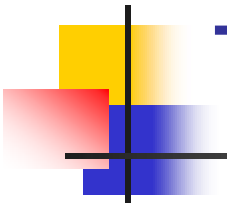
Implications of the Capability System



- The confused deputy problem can be resolved.
- Other problems may arise, however. For example,
  - Roles of programmers and system admins may be mingled?
  - How does one user share files with another user?

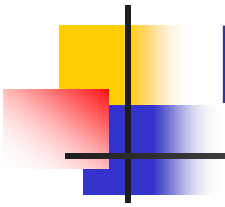
Capability Myths Demolished

Mark S. Miller, Ka-Ping Yee, Jonathan Shapiro



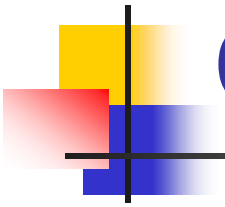
Three Myths

- Equivalence myth: ACL systems and capability systems are equivalent
  - they are just alternative ways of representing access matrices
- Confinement myth: Capability systems cannot enforce confinement
- Irrevocability myth: Capability-based access cannot be revoked



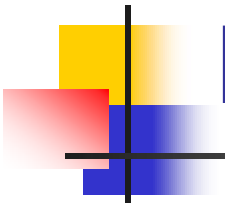
Four Models

- ACLs as columns (of access matrices)
- Capabilities as rows
- Capabilities as keys
- Object capabilities



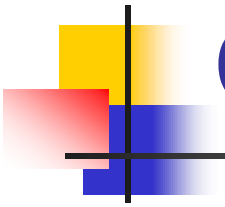
On Equivalence

- While both ACLs and capabilities can represent a static access matrix, state changes are different in ACL systems and capability systems.



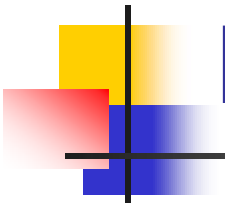
Designation and Authority

- [See the figures comparing ACLs with capabilities]
- ACL systems need a namespace for objects
- In capability systems, a capability can serve both to designate a resource and to provide authority.
- Property A: No designation without authority
  - ACL systems do not have this.
  - [Is this a feature or a bug?]



Granularity of Subjects

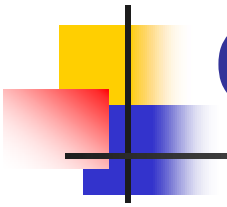
- ACLs also need a namespace for subjects
  - as they need to refer to subjects
- Implications
  - the set of subjects cannot be too many or too dynamic
  - most ACL systems treat users as subjects, and do not support fine-grained subjects
- Property B: Dynamic Subject Creation



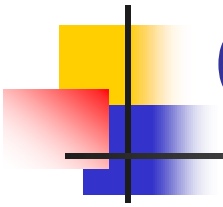
Power to Edit Authorities

- In (almost) all ACL systems, the power to edit authorities is aggregated by resource
  - naturally compatible with DAC model
- In capabilities systems, the power to edit authorities is aggregated by subject
- Property C: Subject-Aggregated Authority Management

ACLs as Columns vs. Capabilities as Rows

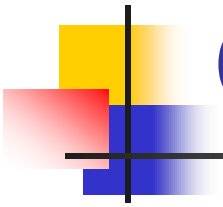


- ACL-based systems do not have the following properties
  - Property A: No designation without authority
  - Property B: Dynamic Subject Creation
  - Property C: Subject-Aggregated Authority Management



On Confinement

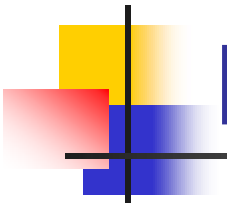
- "The Confinement Myth"
  1. capability systems cannot limit the propagation of authority
  2. capability systems cannot solve the confinement problem
- Observation
  - In object capabilities, for A to give a capability over C to B, A must have a capability over C and a capability over B
  - [addresses 2, but doesn't fully address 1.]



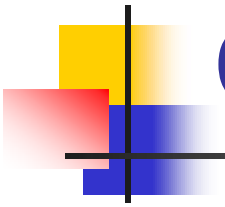
On Irrevocability

- "The irrevocability myth"
  - once a subject holds a capability, no one but the subject can remove the capability
  - delegation is trivial, and revocation is infeasible
- By adding indirection, one can achieve the effect of revocation
  - [See the paper]

On the Ability to Enforce *-property



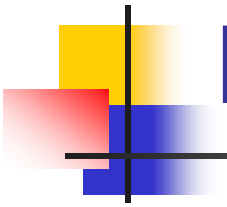
- Boebert claims that "an unmodified capability system cannot enforce the *-property"
  - a low-level user can write the "write low capability" to a place readable by a high-level user
- The authors claim that
  - capabilities cannot be written to data segments; thus the above attack doesn't work
- Unresolved issues
  - What about sending messages from low to high?



Capabilities Are Not Bit Strings

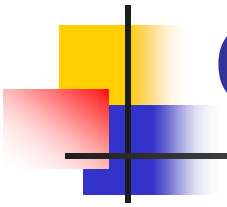
- Gong asserted
  - "Generally a capability is a bit string and can propagate in many ways without detection."
- One category of capability systems, known as password capability system, are like that.

The Capabilities-as-Keys Model



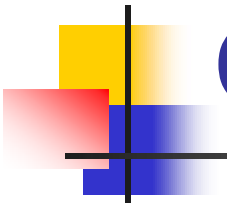
- Capabilities are copyable, unforgeable keys
  - resources are protected by locks
  - accessing a resource requires selecting a key
- Ambient authority means that a user's authority is automatically exercised, but not selected.
  - causes the confused deputy problem
- Property D: No Ambient Authority

Capabilities-as-Keys vs. Object Capabilities



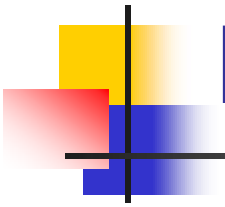
- Property E. Composability of Authorities
  - [Not sure what this property means]
  - access and authorization can be unified
- Property F. Access-Controlled Delegation Channels
  - before A can delegate to B, A must hold a capability over B

Thoughts on OS Access Control and Capabilities



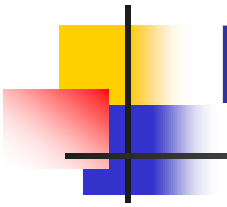
- Static/Dynamic
  - static: resource sharing between users
  - dynamic: access control relationships among processes

- It is unclear whether capability-based systems can handle static resource sharing



Relevant Open Questions

- Are capability-based systems fundamentally better than ACL-based systems such as UNIX?
- Can one add an additional layer of access control to ACL-based systems to improve its access control?
- If so, how the this layer work?



Next Lecture

- Basics of Logic and Logic Programming