Protection

•••

Comp 417 • John Nickels

Problem and Motivation

- How do you allow multiple programs from different sources to share resources and communicate without interfering with each other?
- How do you prevent one program from making an unauthorized access or modification to another program?
- How do you make sure that a program can still make authorized accesses and modifications?

State of the Art

- 'Protection' was published in 1971, 'A Note on the Confinement Problem' in 1973
  - For context, 'THE' was published in 1968, and 'UNIX' in 1974
- According to Lampson, many techniques already exist:
  - Supervisor/user modes
  - Memory relocation and bounds registers
  - Access control by user to file directories
  - Password identification at logon
- Lampson isn't seeking to replace these techniques
  - Lampson refers to these techniques as "ad hoc mechanisms"
  - These mechanisms are hard to reason about, and are very independent of each other
  - Therefore, Lampson seeks to generalize the strategies of protection with abstract models

Key Idea

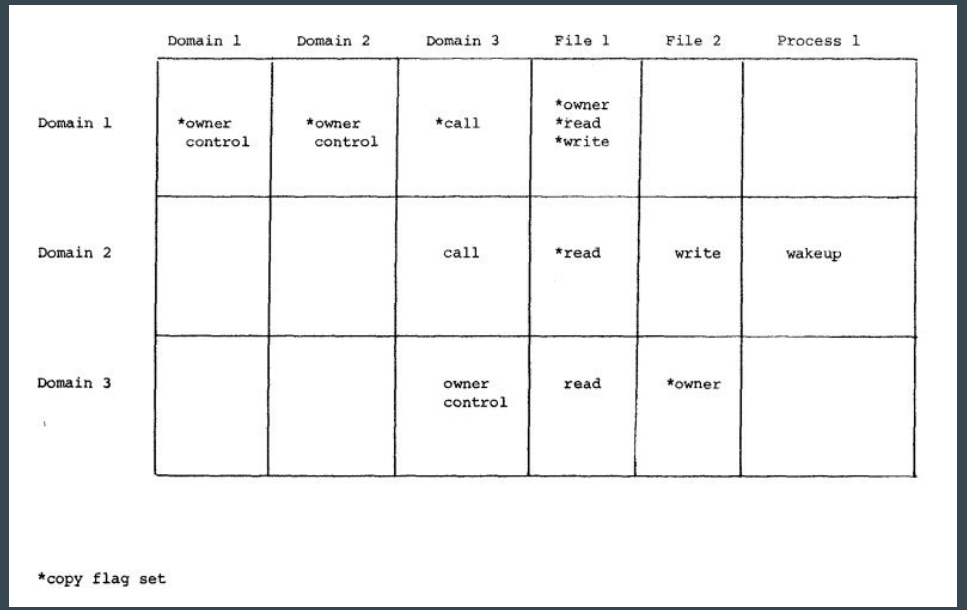
- Lampson introduces 3 models for improving security:
  - Protection Domains
    - Separate the environments of the different processes
  - Access Control Matrix
    - Maps a Domain and an Object to a set of Access Attributes
  - Memory Protection
    - Address space protections allow for enforcement of access control

Protection Domains

- Lampson describes an idealized system, the message system
  - Different processes share nothing, and communicate only through sending messages
    - A message consists of identification (provided by system) and an arbitrary amount of data
  - Everything belongs to some process and cannot be accessed by any other process
    - Can be viewed as physical separation, although that is generally inefficient
  - This allows everyone to protect themselves
    - If you receive a request from an unwanted sender, ignore the request
    - Other processes can't control what you send or what you do
    - Since system supplies identification, it cannot be forged
  - Extremely flexible and general
- However, he also acknowledges two major flaws:
  - No way to check a runaway or crashed process (ie, no kernel or supervisor)
  - Requires outside channels and immense coordination and standardization of protocols

Access Control Matrices

- Maps Domains and Objects to Access Attributes
  - If a certain domain and object map to a 'read' attribute, then that domain can read that object
  - 'Owner' allows attribute management
  - 'Copy' allows read/write management
- Array is inefficient
  - Sparse array, leads to memory waste
  - Instead, use Capabilities:
    - Basically an access key
    - Store all populated items by row
- Group objects into directories
  - Control access to directories, not objects
  - Allows tree-structure
  - Far fewer Access Control Lists



	Domain 1	Domain 2	Domain 3	File 1	File 2	Process 1
Domain 1	*owner control	*owner control	*call	*owner *read *write		
Domain 2			call	*read	write	wakeup
Domain 3			owner control	read	*owner	

*copy flag set

Memory Protection

- Separation of address spaces allow for enforcement of access control
  - Memory that is out of the current address space cannot be named and is therefore protected
  - Pages/segments can have their own protection information (eg PUMP)
- Clever implementations of this allow for dynamic sharing of data

Takeaways

- Since this paper is primarily a discussion of high-level abstractions, no prototype
  - No results to evaluate in context
- However, we can evaluate it based on its impact
  - Protection Domains, although not original, are a fundamental abstraction in modern OS's
  - Access Control Matrices, as well as capabilities, have been adapted into today's research
    - RWX permissions on UNIX
    - We've discussed capabilities before
  - Memory Protection is also not new, but the discussion is accurate and alludes to modern paging