

Operating System Structures to Support Security and Reliable Software

Theodore A. Linden
National Bureau of Standards
Technical Note 919
August 1976

Presentation Outline

- Motivation, Introduction and Scope of Linden's work
- Reliable Software
- Small Protection Domains
- Capability-Based Addressing
- Flexible Sharing
- Extended-Type Objects
- Conclusions

Motivation

- 1974: > 339 computer-related crimes, \$2M, most involving insider fraud, 85% of these not reported to police
 - Claim: Most fraud possible because of oversight in applications systems
 - Goal: Eliminate fraud by automating concepts of segregated duties, independent checking, and accountability

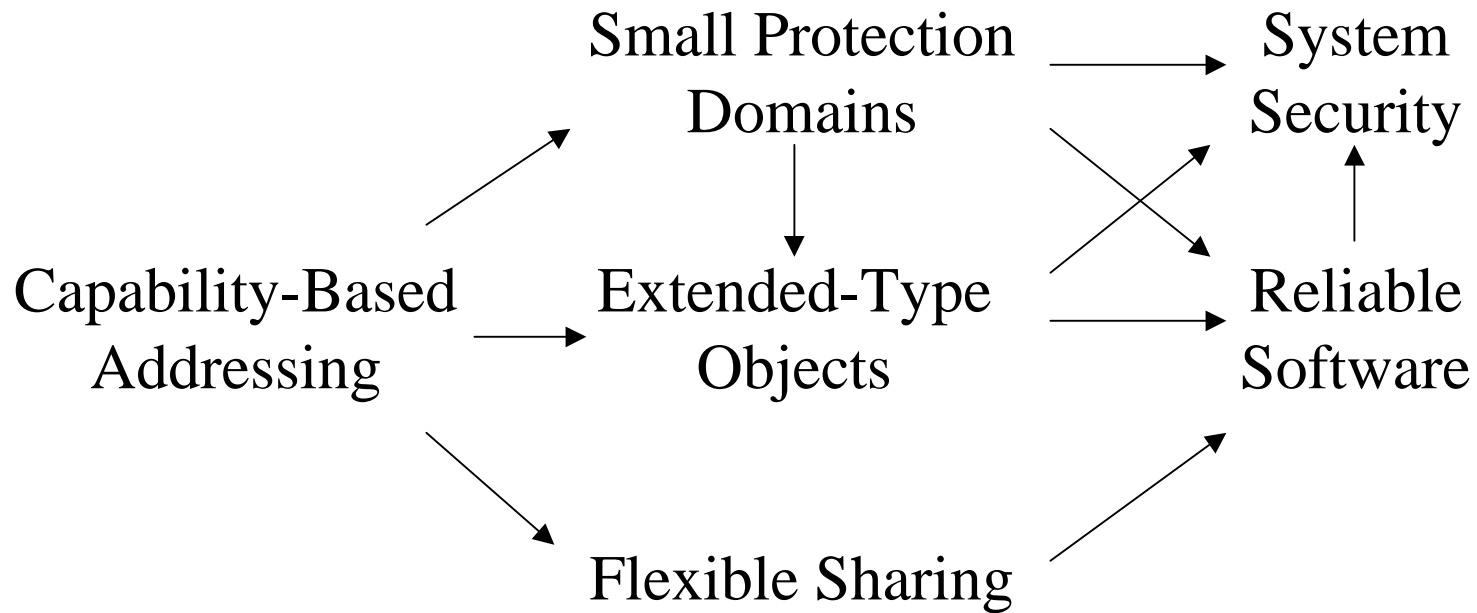
Motivation II

- Most OSs complex, disorganized
 - Goal: Guarantee sys. security over lifetime
- Protection mechanisms can enforce software modularity
 - Claim: Improves reliability through ease in Test, Debug phases
- System crashes are security problem
 - Goal: Eliminate software-induced crashes, repair inadequate fault-recovery mechanisms

Linden's recommendation

- Small Protection Domains
 - each module/subunit of a program run in a restricted environment
- Extended-Type Objects
 - protection inheritance in derived data types

Support Structure



Definitions

- Security
 - protection of resources from accidental or malicious modification, destruction or disclosure
- Reliable Software
 - Provides usable, correct, trustworthy, and available services

Definitions II

- Small Protection Domain
 - Environment/context that restricts subjects to access rights for only the objects needed to accomplish the current task
- Extended-Type Objects
 - Application-created objects of newly-defined Types, where two objects are of different Type if the operations allowed on them differ

Limitations of Scope

- Linden does not address
 - Communication security
 - User identification
 - Physical security

Reliable Software

- “A means to security, and it is an end in itself.”
- Usable
 - services rendered are effective for application
- Correct
 - meets functional specifications
- Trustworthy
 - minimum, verifiable level of services correct

Small Protection Domains

- Design assumption: Program subunits only need access to a small number of objects
 - Result: large, sparse protection matrix
 - Need: easy and efficient domain switching
- Access rights passed to invoked subjects
 - new protection domain created, union of subject's default and instance-granted rights
 - least privilege: need ability to remove rights

Small Protection Domains II

- To enforce small domains, need small modules
 - can be less efficient
 - may interact in unexpected ways
 - Need: limits on interaction, possibly at compile-time
 - Defensive Programming cushions error propagation
- Small OS protection domains require almost complete OS redesign and language support

Small Protection Domains III

- Benefits:
 - Debugging - programming errors easier to find, correction less likely to cause other modules to malfunction because of their isolation
 - Testing - easier due to limited execution environment
 - Fault Handling - easier to contain effects, permitting recovery and retry efforts

Small Protection Domains IV

- Benefits:
 - Maintenance - protection information points to all other modules possibly effected by change
 - Program Property Proofs - simplified due to modular constraints
 - Trojan Horses - cannot access unauthorized information by default

Flexibility vs. Security

- Systems attacked at weakest point
 - Need: no back doors
- Acceptability to the users
 - Better to have a weaker, but always enforced policy than a strong, circumvented one
- Complete and correct auditing
 - Facilitated by protection domains

Flexibility vs. Security II

- Efficient Domain Switching
 - Makes redundant or data-dependent checks more feasible
- Sparse rule set
 - Claim: access rules easier to understand

Capability-Based Addressing

- Access to an object granted iff in possession of a capability for that object
- Determines access rights to named object
- Created only by system
- Not modifiable, except to remove rights
- Moving, copying, passing capabilities allowed

Capability-Based Addressing II

- Must control user access to capabilities
- Capability-related computation ~1ms
- Domain switching implemented by stacks
 - Can only access capabilities in current frame
 - New frame pushed onto stack upon Call, old frame popped off stack upon Return
- Long term storage in a directory structure
 - Access to which maintained by capabilities

Flexible Sharing

- In general, less sharing = greater security, however sharing is often a requirement
- Reliable software more difficult with increased sharing (interactivity), however if trusted, this saves development effort
- “A building block approach to reliable software may soon become feasible.”

Extended-Type Objects

- Problem: Process invoking an object may not have rights to other instances of that type, e.g., directories
 - Solution 1: Amplification (Hydra system) - provided an “add” capability to a specific directory, the Add operation can obtain read and write permissions.

Extended-Type Objects II

- Solution 2: Indirection (Plessy System 250) - associated with each directory is an entry pointer. Indirecting an “entry” capability to access the directory, control transferred to the entry procedure pointed to.
- Solution 3: Extended-Type Manager (Neumann et al.) - access to the type representation provided only to authorized operations within that type.

Modularity

- Problem decomposition to distinct tasks = Horizontal Modularization
 - modules often still quite large
- Global data structures thwart modularity goals
- Problem decomposition to different levels of abstraction = Vertical Modularization
 - efficient modules often call multiple layers

Modularity Support in Languages

- 1970's - Procedures most common modularity unit
- “Class” (Dahl 1968), updated with protection (Palme 1974)
- “Function Clusters” (Liskov) - similar to classes, but object representation compiler-enforced

Support in Languages II

- “Form” (Wulf) - more general than Clusters
- “Modules” (Parnas) - Operations and Value Functions. The latter return a value but do not change a module’s state

Requirements for Language Support

- Modules must be able to
 - have multiple entry points accessible to other modules
 - retain state between successive operations
 - rigorously control module interactions

Extended-Type Objects to enforce Security

- Although a user is permitted to distribute capabilities, we may want to restrict this
 - Proposal: encompass such data or resources within an extended-type object which performs the mandatory access checks desired (best used in a level/compartment lattice such as the military model)

Conclusions

- Not enough economic incentive
 - most proposed changes will cost a lot more
 - only cost-saving is reused code, but no hard figures exist as to its dollar amount
- Backward compatibility
 - achievable only through emulation
 - emulation achievable only after sunk cost in OS, hardware design and implementation