Protection

Dave Eckhardt
de0u@andrew.cmu.edu

Outline

- Protection (Chapter 18)
  - Protection vs. Security
  - Domains (Unix, Multics)
  - Access Matrix
    - Concept, Implementation
  - Revocation
- Deferred: Language-based protection (18.7)
- Mentioning EROS

Protection vs. Security

- Textbook's distinction
  - Protection happens inside a computer
    - Which parts may access which other parts (how)?
  - Security considers ***external threats***
    - Is the system's model intact or compromised?

Protection

- Goals
  - Prevent intentional attacks
  - “Prove” *access policies* are always obeyed
  - Detect bugs
    - “Wild pointer” example
- Policy specifications
  - System administrators
  - Users - May want to add new privileges to system

Objects

- Hardware
  - Single-use: printer, serial port, CD writer, ...
  - Aggregates: CPU, memory, disks, screen
- ***Logical*** objects
  - Files
  - Processes
  - TCP port 25
  - Database tables

Operations

- Depend on object
  - CPU: execute(...)
  - CD-ROM: read(...)
  - Disk: read_sector(), write_sector()

Access Control

- Your processes should access only “your stuff”
  - Implemented by many systems
- ***Principle of least privilege***
  - (text: need-to-know)
  - cc -c foo.c
    - should read foo.c, stdio.h, ...
    - should write foo.o
    - ***should not write ~/.cshrc***
  - This is harder

Protection Domains

- process $\rightarrow$ protection domain
- protection domain $\rightarrow$ list of access rights
- access right = (object, operations)

Protection Domain Example

- Domain 1
  - /dev/null, write
  - /usr/davide/.cshrc, read/write
  - /usr/smuckle/.cshrc, read
- Domain 2
  - /dev/null, write
  - /usr/smuckle/.cshrc, read/write
  - /usr/davide/.cshrc, read

Protection Domain Usage

- Least privilege requires domain changes
  - Static “process $\rightarrow$ domain” requires mutable domains
  - Static domains requires ***domain switching***
- Three models
  - Domain = user
  - Domain = process
  - Domain = procedure

Domain = user

- Object permissions depend on ***who you are***
- All processes you are running share privileges
- Domain switch
  - Log off

Domain = process

- Resources managed by special processes
  - Printer daemon, file server process, ...
- Domain switch
  - IPC to resource owner/provider
  - “Please send these bytes to the printer”

Domain = procedure

- Processor limits access at fine grain
  - Individual variables
- Domain switch
  - Procedure call
    - nr = read(fd, buf, sizeof (buf))
    - automatic creation of “the correct domain” during read()

Unix "setuid" concept

- Assume Unix domain = numeric user id
  - Not the whole story!
    - Group id, group vector
    - Process group, controlling terminal
    - Superuser
- Domain switch via ***setuid executable***
  - Special bit: exec() changes uid to file owner
  - Gatekeeper programs
    - Allow user to add file to print queue

Multics Approach

- Trust hierarchy
  - Small “simple” very-trusted ***kernel***
    - Goal: prove it correct
  - Operating system layers
    - Disk device driver
    - Page system
    - File system
    - Print queue
    - User program

Multics Ring Architecture

- Segmented address space
  - Segment = file (persistent segments)
- Segments live in nested ***rings*** (0..7)
  - Ring 0 = kernel, “inside” every other ring
  - Ring 1 = operating system core
  - Ring 2 = operating system services
  - ...
  - Ring 7 = user programs

Multics Domain Switching

- CPU has ***current ring number*** register
- Segments have
  - Ring number
  - Access bits (read, write, execute)
  - Access bracket [min, max]
    - Segment “part of” ring min...ring max
  - Entry limit
  - List of gates (procedure call entry) points
- Every procedure call is a potential domain switch

Multics Domain Switching

- min <= current-ring <= max
  - Standard procedure call
- current-ring < min
  - Calling a less-privileged procedure
  - Trap to ring 0
  - Copy “privileged” procedure call parameters
    - Must be in low-privilege area for callee to access
  - Set current-ring to segment-ring

Multics Domain Switching

- current-ring > max
  - Calling a more-privileged procedure
  - Trap to ring 0
  - Check current-ring < entry-limit
    - User code may be forbidden to call ring 0 directly
  - Check call address is a legal entry point
  - Set current-ring to segment-ring

Multics Ring Architecture

- Does this look familiar?
- Benefits
  - Core security policy small, centralized
  - Damage limited vs. Unix “superuser”' model
- Concerns
  - Hierarchy conflicts with least privilege
  - Requires specific hardware
  - Performance (maybe)

More About Multics

- Back to the future (1969!)
  - Symmetric multiprocessing
  - Hierarchical file system (access control lists)
  - Memory-mapped files
  - Hot-pluggable CPUs, memory, disks
- Significant influence on Unix
  - Ken Thompson was a Multics contributor
- www.multicians.org

Access Matrix Concept

	File1	File2	File3	Printer
D1		rwxd	r	
D2	r		rwxd	w
D3	rwxd	rwxd	rwxd	w
D4	r	r	r	

Access Matrix Details

- OS must still define process $\rightarrow$ domain mapping
- Must control domain switching
  - Add domain ***columns*** (domains are objects)
  - Add switch-to rights to domain objects
- These are both subtle (dangerous)

Updating the Matrix

- Add ***copy rights*** to objects
  - Domain D1 may copy read rights for Object O47
  - So D1 can give D2 the right to read O47
- Add ***owner rights*** to objects
  - D1 has owner rights for O47
  - D1 can modify the O47 column at will

Updating the Matrix

- Add *control rights* to domain objects
  - D1 has control rights for D2
  - D1 can modify D2's rights to any object

Access Matrix Implementation

- Global table
  - Huge, messy, slow
  - Particularly clumsy for...
    - “world readable file”
    - “private file”

Access Matrix Implementation

- Access Control Lists
  - List per matrix column (object)
  - Naively, domain = user
  - AFS ACLs
    - domain = user, user:group, anonymous, IP-list
    - positive rights, negative rights
  - Doesn't really do ***least privilege***

Access Matrix Implementation

- Capability Lists
  - List per matrix row (domain)
  - Naively, domain = user
  - Typically, domain = process
    - Permits ***least privilege***
  - Bootstrapping problem
    - Who gets which rights at boot?
    - Who gets which rights at login?

Mixed approach

- Store ACL for each file
  - Must get ACL from disk
  - May be long, complicated
- open() checks ACL, creates capability
  - Records access rights for this process
  - Quick verification on each read(), write()

Revocation

- Adding rights is easy
  - Make the change
  - Tell the user “ok, try it now”
- Removing rights is harder
  - May be cached, copied, stored

Revocation Taxonomy

- Immediate/delayed
  - How fast? Can we know when it's safe?
- Selective/global
  - Delete ***some domain's*** rights?
- Partial/total
  - Delete particular rights for a domain?
- Temporary/permanent
  - Is there a way to re-add the rights later?

Revocation Approaches

- Access Control List
  - Modify the list
  - “Done”
    - ...if no cached capabilities
- Capability timeouts
  - Must periodically re-acquire (if allowed)

Revocation Approaches

- Capability check-out list
  - Record all holders of a capability
  - Run around and delete the right ones
- Indirection table
  - Domains point to table entry
  - Table entry contains capability
  - Invalidate entry to revoke everybody's access

Revocation Approaches

- Proxy processes
  - Give out ***right to contact*** an ***object manager***
  - Manager applies per-object policy
    - “Capability expired”
    - “No longer trust Joe”

Revocation Approaches

- Keyed capabilities
  - Object maintains list of active keys
  - Give out (key, rights)
  - Check “key still valid” per access
  - Owner can invalidate individual keys
- Special case: #keys = 1
  - Versioned capabilities
  - NFS file handles contain inode generation numbers

Mentioning EROS

- Text mentions Hydra, CAP
  - Late 70's, early 80's
  - Dead
- EROS (“Extremely Reliable Operating System”)
  - UPenn, Johns Hopkins
  - Based on commercial GNOSIS/KeyKOS OS
  - www.eros-os.org

EROS Overview

- “Pure capability” system
  - “ACLs considered harmful”
- “Pure principle system”
  - Don't compromise principle for performance
- Aggressive performance goal
  - Domain switch ~100X procedure call
- Unusual approach to bootstrapping problem
  - ***Persistent processes!***

Persistent Processes

- No such thing as reboot
- Processes last “forever” (until exit)
- OS kernel checkpoints system state to disk
  - Memory & registers defined as ***cache of disk state***
- Restart restores system state into hardware
- “Login” ***reconnects*** you to your processes

EROS Objects

- Disk pages
  - capabilities: read/write, read-only
- Capability nodes
  - Arrays of capabilities
- Numbers
  - Protected capability ranges
    - “Disk pages 0...16384”
- Process – executable node

EROS Revocation Stance

- ***Really*** revoking access is hard
  - The user could have copied the file
- Don't give out real capabilities
  - Give out proxy capabilities
  - Then revoke however you wish

EROS Quick Start

- www.eros-os.org/
  - reliability/paper.html
  - essays/
    - capintro.html
    - wherefrom.html
    - ACLSvCaps.html