



## Multics History Computer Security Principles

 CS165, B. D. Fleisch 1


## The Protection of Information in Computer Systems

- Canonical paper in computer system security
- Starts with a glossary
- Narrows in on system software issues

 CS165, B. D. Fleisch 2


## Definitions

- Privacy – “The ability of an individual to decide whether, when, and to whom personal information is released.”
- Security – ability to prevent unauthorized:
  - ◆ Reading (confidentiality)
  - ◆ Writing (integrity)
  - ◆ Availability
- Protection: security (access control) for programs running on your machine
- Authentication: verify the identity of a person or agent making a request

 CS165, B. D. Fleisch 3

## Multiple Use Systems

- Computer with more than one purpose
  - ◆ 1975: mainframes, time-sharing
  - ◆ 2000: networked PCs
- In both cases: you share the environment with people and programs who don't trust each other completely.

 CS165, B. D. Fleisch 4

## Levels of Information Protection

- Unprotected Systems
- All-or-Nothing Systems
- Controlled (Static) Sharing
  - ◆ Different people can access each item
  - ◆ UNIX: user/group IDs, mode bits
  - ◆ NFS: access control lists
- Programmable Sharing Controls
  - ◆ Two users must agree to modification
  - ◆ Access only between 2am and 6am
  - ◆ Implementation techniques:
    - Reference monitors
    - Capabilities
- Tagged Information



CS165, B. D. Fleisch

5

## Unprotected Systems

- No protection at all; any running program can access the entire system
  - ◆ 1975: IBM's Disk Operating System (pronounced "System 370")
  - ◆ 1995: Microsoft's Disk Operating System (pronounced "Windows 95")
- May have protection against "mistakes"
- Doesn't count for security
  - ◆ Attacker just looks for the holes



CS165, B. D. Fleisch

6

## All or Nothing Systems

- Users are completely isolated
  - ◆ Network equivalent: disconnect
- May also provide a "library"
- May allow users to contribute to library
  - ◆ All users have equal access
  - ◆ No authentication on contributions to library
- Everything public or private



CS165, B. D. Fleisch

7

## Controlled Sharing

- Goal: control who may access *each* item stored in the system, e.g.
  - ◆ UNIX user and group IDs, mode bits
  - ◆ Access Control Lists
- Problems:
  - ◆ implementation is complex
  - ◆ access specification is complex, so setting it *correctly* is hard



CS165, B. D. Fleisch

8

## User-programmable Sharing Controls

- Allow weird access restrictions, e.g:
  - ◆ only between 9 am & 4 PM
  - ◆ modify only if two users agree
- Technique: build a protected subsystem
  - ◆ only subsystem can access protected data
  - ◆ subsystem can do arbitrary checks before granting access request
- Example: UNIX suid facility

University of California Berkeley
CS165, B. D. Fleisch
9

## Tagged Information

- Previous controls do checks prior to releasing information
- May also want to control information *after* it is released
- Very hard to do:
  - ◆ don't actually release the info
  - ◆ rather, release a reference to the info
  - ◆ check access on each reference to it
  - ◆ what if a client *memorizes* accessed info?

University of California Berkeley
CS165, B. D. Fleisch
10

## Isolated Virtual Machine: Hardware Protection

- Isolate memory regions
  - ◆ Pre 70s: fence registers
  - ◆ 1970's: base & bound registers
  - ◆ 1990's: virtual address spaces
- Who controls the address regions?
  - ◆ *Privileged* status bit in processor
  - ◆ Privilege bit must be set to manipulate any address region
  - ◆ Only supervisor program has the supervisor bit set

University of California Berkeley
CS165, B. D. Fleisch
11

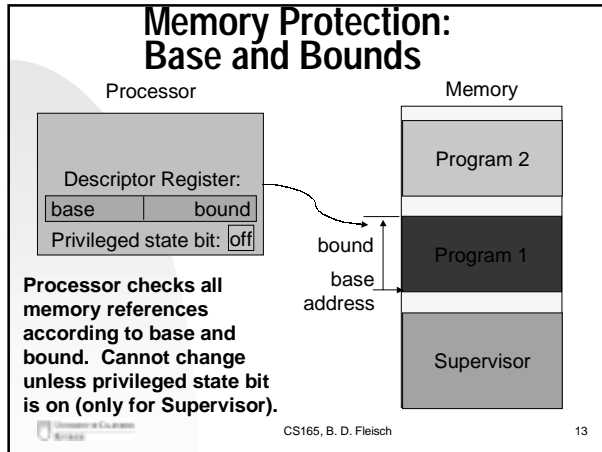
## Memory Protection using a Fence

Processor checks all memory references to see if reference is beyond fence address

Later implementations loaded a register with fence address so that os boundary could grow or shrink

Restriction: can not protect one user from another

University of California Berkeley
CS165, B. D. Fleisch
12



- ### Memory Protection: Base and Bounds
- Limitation: text versus data segment within same process not protected
  - Solution: two sets of base/bound registers
    - ◆ One for data
    - ◆ One for text
- CS165, B. D. Fleisch 14

- ### Virtual Memory and Limitations
- Virtual Memory: separate virtual address spaces
  - Cannot share memory between programs
    - ◆ Can add additional descriptors to set up shared memory, add read/write bits, etc.
  - Requires special hardware
  - Performance cost for every memory access
- CS165, B. D. Fleisch 15

### Tagged Architecture

- Every word of machine memory has one or more extra bits to identify the access rights to that word
- Bits can only be set by privilege mode instructions
- Bits examined and tested every time that location is accessed

Tag Memory Word	
R	0001
RW	0137
R	0098
X	xxxx
X	xxxx
X	xxxx
X	xxxx
X	xxxx
X	xxxx
X	xxxx
X	xxxx
X	xxxx
X	xxxx
R	4091
RW	0002

Code:  
 R = Read Only  
 RW = Read/Write  
 X = Execute-Only

CS165, B. D. Fleisch 16

## Supervisor Program as a Protected Subsystem

- Special instruction:
  - ◆ switches on privileged bit
  - ◆ *and* transfers control to the supervisor
- Innards of supervisor *not* accessible to users
- Facilities of supervisor *are* accessible to users
- Correctness of supervisor is *critical*



CS165, B. D. Fleisch

17

## Information Protection

- Divide data that warrants separate protection into *partitions*
- Put a wall around each partition
- Put a door in each wall
- Guard decides who gets through the door, based on
  - ◆ *identity*
  - ◆ *authorization*



CS165, B. D. Fleisch

18

## Descriptor-Based Protection

- *Not* relevant to memory systems
  - ◆ All modern systems use a flat VM scheme
- *However* Java and DCOM use it:
  - ◆ Descriptor == Unique Identifier == Object Reference



CS165, B. D. Fleisch

19

## Descriptor-based Protection

### 1975:

- ◆ Descriptor is a value in a hardware register
- ◆ Points to a segment table, guards actual segment
- ◆ Set of segment values is your domain
- ◆ Need privilege bit to load a segment register

### 1997:

- ◆ Descriptor is an object reference
- ◆ *Compiler* enforces unforgability of object references
- ◆ Can only invoke methods of the object



CS165, B. D. Fleisch

20

## Segmentation

- Each program is subdivided into blocks of non-equal size called segments
- When a process gets loaded into main memory, its different segments can be located anywhere
- Each segment is fully packed with instructions/data: no internal fragmentation
- There is external fragmentation; it is reduced when using small segments

Figure 6-4 Logical and Physical Representation of Segments

CS165, B. D. Fleisch 21

## Capabilities and Tagged Architectures

- **Capability:** an unforgeable pointer to a controlled system resource
  - ◆ Usually 1) tagged or 2) stored in special segments
- Lets you place descriptor values in memory *safely*
- A reference with the tag bit on is called a **capability**
  - ◆ Allows for arbitrary, controlled memory sharing
  - ◆ Capabilities can refer to hardware devices also (they are just memory addresses)
- Need to make capabilities unforgeable
  - ◆ Hardware-protected tags used
  - ◆ (~ Java – type-checking + bytecode verification)

CS165, B. D. Fleisch 22

## Capability Use

- To use a capability, a program must have been first explicitly given that capability, either as part of its initialization or as the result of calling another capability.
- Once a capability has been given to a program, the program may then use the capability as much as it wishes
- It may pass the capability to other programs.
- This leads to a basic property of capabilities: any program which *has* a capability *must* have been permitted to use it.

CS165, B. D. Fleisch 23

## Principals

- **Principal:** entity accountable for actions of a virtual processor
- Principals don't map directly to people:
  - ◆ System administrator doesn't do everything as *root*
  - ◆ Web server has its own user-ID, but several people have access to that ID

CS165, B. D. Fleisch 24

## Domains

- **Domain**: the set of objects a principal may access
- A form of local name space
- **Tickets** are *small* principals
  - ◆ Groups of tickets (roughly) form aggregate principals by defining a domain



CS165, B. D. Fleisch

25

## Capabilities Generalize Sharing

- Supervisor creates segments containing sets of capabilities
- Capabilities may be associated with a string thus naming them
  - ◆ This makes the segment an ORB
- Supervisor invoked to create new segments



CS165, B. D. Fleisch

26

## Authentication, Capabilities and Domains

- User Doe authenticates to login program
- login
  - ◆ Creates virtual processor
  - ◆ Loads Doe's segment registers
  - ◆ Doe can then access any capability stored in those registers
- User authenticated to be the principal  
Doe now connected to Doe's domain



CS165, B. D. Fleisch

27

## Dynamic Authorization of Sharing

- Doe wants to share *x* with Smith
- Give Doe write access to Smith's catalog?
  - ◆ Then Doe can destroy Smith's catalog
- Give Smith read access to Doe's catalog?
  - ◆ Then Smith gets *all* of Doe's capabilities
- Need more control over sharing



CS165, B. D. Fleisch

28

## Controlled Sharing

- **Anticipate the need for sharing**
  - ◆ Create segment that both can read & write
  - ◆ Sender marks msg with ID of sender and receiver
  - ◆ Receiver checks msg to ensure identity of sender
- **Only works for *a priori* communication**
- **To scale up: Privileged “post office” program dynamically creates channel**



CS165, B. D. Fleisch

29

## Revocation and Control Propagation

- **Capability is *unlimited* delegated authority.**  
Problems:
  - ◆ Changed your mind
  - ◆ Worried that `smith` passed it out to others
  - ◆ Can't know who has capability to your resources
- **Once someone has a capability, how can you deny access?**
  - ◆ Must destroy original object
- **Java: once someone has an open `FileOutputStream`, can't revoke it!**
- **No cheap solution:**
  - ◆ Store capabilities somewhere special and disallow copying
  - ◆ Require an indirection step through something object owner controls



CS165, B. D. Fleisch

30

## Revocation and Control Propagation

- **Copy bit:**
  - ◆ Without it, can't copy the capability
- **Capability-holding segments:**
  - ◆ Only special segments can hold capabilities
  - ◆ Reduces work for auditing and revocation
- **Capabilities not persistent:**
  - ◆ Can't store capabilities to disk



CS165, B. D. Fleisch

31

## Revocation and Control Propagation

- **Double indirection:**
  - ◆ Capability points to a pointer to the data
  - ◆ Owner of capability can destroy the pointer
- **Basic problem:**
  - ◆ Authorization happens when capability is copied
  - ◆ Restrictions on copy-ability restrict general usefulness of capabilities
  - ◆ For finer control, use Access Control Lists




CS165, B. D. Fleisch

32


### Capabilities Problems: Propagation

- How can you control who capability is passed to? (You can't.)
- Java: can't control which applet that FileOutputStream is passed to
- Possible solutions:
  - Store somewhere special and disallow copying (passing as parameters)
  - Associate capability with principal (need to check call stack)


CS165, B. D. Fleisch 33


### Access Control Lists

- Maintain a list of principals and access permissions
- Delay check until last possible moment (can "revoke" until then)
- Fixes the authorization problem:
  - Check on every reference
- Implementation:
  - Associate ACL with each segment
  - Capabilities -> simple pointers, because the pointee does the check
  - Critically depends on authentication to do the access check


CS165, B. D. Fleisch 34


### Illustration of ACLs

FILE	ACCESS LIST POINTER	USER	ACCESS RIGHTS
BIBLIOG	→	USER_A	ORW
TEMP	→	USER_B	R
F	→	USER_S	RW
HELP.TXT	→	USER_A	ORW
		USER_A	ORW
		USER_S	R
		USER_A	R
		USER_B	R
		USER_S	R
		USER_T	R
		YSMGR	RW
		USER_SVCS	O


CS165, B. D. Fleisch 35

### Differences from Pure Capability Systems

- Authorization has known consequences
  - Authorize *only* Smith
- Revocation is easy: edit list
- Auditing is easy: look at list
- Authorization dissociated from data organization:
  - Objects stored together can have different ACLs


CS165, B. D. Fleisch 36

## Protection Groups

- ACLs can get tediously long
- Protection groups:
  - ◆ Principals that can be used by more than one user
  - ◆ Protection group in an ACL -> all members of the protection group have access



CS165, B. D. Fleisch

37

## Protecting Objects Other Than Segments

- Define operations on the object
- Define permissions for those operations
  - ◆ E.g. treat a segment as a FIFO queue
- Problem: a rogue virtual processor can treat *any* segment as a FIFO, and can do arbitrary operations to a segment declared to be a FIFO



CS165, B. D. Fleisch

38

## Protecting Objects Other Than Segments: *Types*

- Add a *type* field to each segment
  - ◆ 1975: hardware knows the types
  - ◆ 1998: *compiler* knows the types
  - ◆ Only permitted operations for that type are allowed
- To support dynamic types without compiler support:
  - ◆ Use a *protected subsystem* to mediate access to the object



CS165, B. D. Fleisch

39

## Protected Subsystems

- Want to define *our own* protected operations on objects
  - ◆ This is OO abstraction with *enforcement*
- Protected Subsystem:
  - ◆ Collection of program & data segments *encapsulated* so that other programs cannot manipulate them directly
  - ◆ Encapsulated objects are *protected objects*
  - ◆ External programs can only invoke explicit methods



CS165, B. D. Fleisch

40

## Protected Subsystems

- Access methods define their own restrictions
- Any decidable restriction can be implemented
- Can plug together *untrusted* borrowed programs (e.g. “applet” :-)
- ◆ Borrowed program runs in a domain different from the user’s normal domain
- ◆ Limits damage to the “sandbox” domain



CS165, B. D. Fleisch

41

## Changing Domains

- Need mechanism to change domains when calling methods, i.e. need to associate multiple domains with a single computation
- ◆ Use multiple virtual machines, or
- ◆ Associate multiple domains with a single virtual machine
- ◆ In ACL terms, changing domains is changing principal identifiers



CS165, B. D. Fleisch

42

## Changing Domains: ACL

- Add an *enter* right to a domain
- ◆ Each entry is a domain ID that may call this domain
- ◆ Processor changes domain ID when call is made, e.g. system call trap
- ◆ Once inside, the domain’s protection methods can safely determine the rest of access control



CS165, B. D. Fleisch

43

## Changing Domains: Capabilities

- Add an *enter* right to a capability
- ◆ If you have the *enter* bit on, you can call the capability
- ◆ Users need *enter* capability for sending mail, but not read or write



CS165, B. D. Fleisch

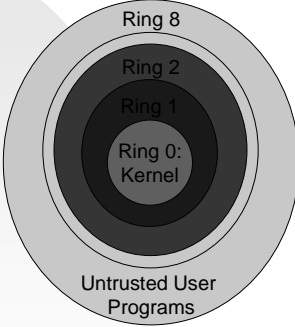
44

### Changing Domains *Safely*

- Switching domains must careful:
  - ◆ Change of domain and change of program counter must be *atomic*
- Passing arguments must be careful:
  - ◆ Called domain must have access to arguments without violating it's own domain restrictions
  - ◆ *Huge* example: stack smashing attacks

CS165, B. D. Fleisch 45

### Multics Rings



Lower rings have more privileges.

Memory segments have descriptors that indicate highest ring number that may read/write segment.

Special instructions for switching between rings (e.g., making a system call).

CS165, B. D. Fleisch 46

### Relationship between Capabilities and ACLs

	BIBLIOG	TEMP	F	HELPTXT	C_COMP	LINKER	SYS_CLOCK	PRINTER
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	—	—	R	X	X	R	W
USER S	RW	—	R	R	X	X	R	W
USER T	—	—	—	R	X	X	R	W
SYS_MGR	—	—	—	RW	OX	OX	ORW	O
USER_SVCS	—	—	—	O	X	X	R	W

Figure 6-13 Access Control Matrix

**Rows: Capabilities**  
**Columns: ACLs**

CS165, B. D. Fleisch 47


### Design Principles for Protection Mechanisms

- Least Privilege
- Economy of Mechanism
- Complete Mediation
- Open Design
- Separation of Privilege
- Least Common Mechanism
- Fail Safe Defaults
- Psychological Acceptability

CS165, B. D. Fleisch 48


### Least Privilege

- Should only have rights needed to complete the task required
- Default should be lack of access
- Allow as little access as possible for job

 CS165, B. D. Fleisch 49


### Economy of Mechanism

- Sufficiently small and simple to be verified and implemented
- Keep it small and stupid
- Example: security kernel

 CS165, B. D. Fleisch 50


### Complete Mediation

- Every access to every object is checked
- Must be efficient
- In addition to normal runtime, must be done at:
  - ◆ Initialization
  - ◆ Shutdown
  - ◆ Restart

 CS165, B. D. Fleisch 51


### Open Design

- Doesn't depend on secrecy of the implementation or Don't depend on security by obscurity
- Scrutiny of the community valuable
- Better to have flaw found by colleague rather than a foe

 CS165, B. D. Fleisch 52


### Separation of Privilege

- Access to objects should depend on more than one condition being satisfied
- Require two separate checks/keys for permission

 CS165, B. D. Fleisch 53


### Least Common Mechanism

- Minimize the amount of mechanism common to more than one user and depended on by all users
- Don't share mechanisms between users
- Every shared mechanism is a potential information path

 CS165, B. D. Fleisch 54


### Fail Safe Defaults

- The default is lack of access
- Those than need access should argue that they need access rather than there being an argument that access should be taken away

 CS165, B. D. Fleisch 55

### Psychological Acceptability

- User interface should be easy to use so that users apply the mechanisms easily or they can be automatically applied easily
- Don't confuse users or drive them crazy
- Otherwise the mechanisms will be bypassed

 CS165, B. D. Fleisch 56

### Principles for a Secure Design

- Design security in from the start
- Allow for future security enhancements
- Minimize and Isolate security controls
- Employ least privilege
- Structure the security relevant features
- Make security friendly
- Don't depend on secrecy for security

University of California Berkeley CS165, B. D. Fleisch 57

### Making Security Friendly

- Assure security does not impact users who play by the rules
- Establish reasonable defaults
- Make giving access friendly
- Make restricting access friendly

University of California Berkeley CS165, B. D. Fleisch 58

### S&S Principles: Conflicts?

- Economy of Mechanism vs. Least Common Mechanism
- Fail-safe defaults vs. Psychological Acceptability
- Separation of Privilege vs. Economy, Psychological Acceptability
- Least privilege vs. Psychological Acceptability

University of California Berkeley CS165, B. D. Fleisch 59