

CSci 5271  
Introduction to Computer Security  
Day 11: OS security: higher assurance

Stephen McCamant  
University of Minnesota, Computer Science & Engineering

## Outline

Multilevel and mandatory access control  
Capability-based access control  
Announcements intermission  
OS trust and assurance  
More Unix access control

## Bell-LaPadula, linear case

- State-machine-like model developed for US DoD in 1970s
- 1. A subject at one level may not read a resource at a higher level
  - Simple security property, "no read up"
- 2. A subject at one level may not write a resource at a lower level
  - \* property, "no write down"

## High watermark property

- Dynamic implementation of BLP
- Process has security level equal to highest file read
- Written files inherit this level

## Biba and low watermark

- Inverting a confidentiality policy gives an integrity one
- Biba: no write up, no read down
- Low watermark policy
- $BLP \wedge Biba \Rightarrow$  levels are isolated

## Information-flow perspective

- Confidentiality: secret data should not flow to public sinks
- Integrity: untrusted data should not flow to critical sinks
- Watermark policies are process-level conservative abstractions

## Covert channels

- Problem: conspiring parties can misuse other mechanisms to transmit information
- Storage channel: writable shared state
  - E.g., screen brightness on mobile phone
- Timing channel: speed or ordering of events
  - E.g., deliberately consume CPU time

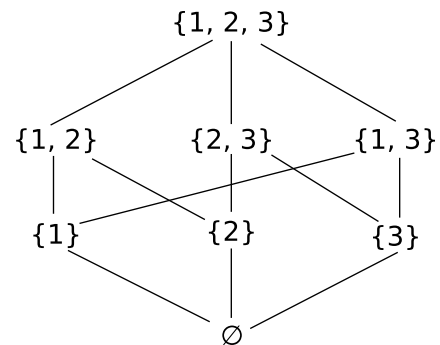
## Multilateral security / compartments

- In classification, want finer divisions based on need-to-know
- Also, selected wider sharing (e.g., with allied nations)
- Many other applications also have this character
  - Anderson's example: medical data
- How to adapt BLP-style MAC?

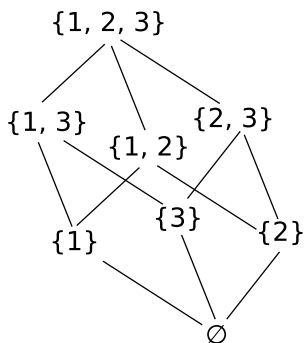
## Partial orders and lattices

- $\leq$  on integers is a *total order*
  - Reflexive, antisymmetric, transitive,  $a \leq b$  or  $b \leq a$
- Dropping last gives a *partial order*
- A *lattice* is a partial order plus operators for:
  - Least upper bound or join  $\sqcup$
  - Greatest lower bound or meet  $\sqcap$
- Example: subsets with  $\subseteq, \cup, \cap$

## Subset lattice example



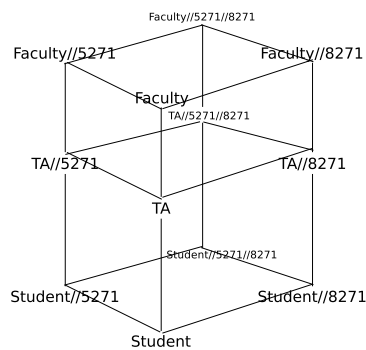
## Subset lattice example



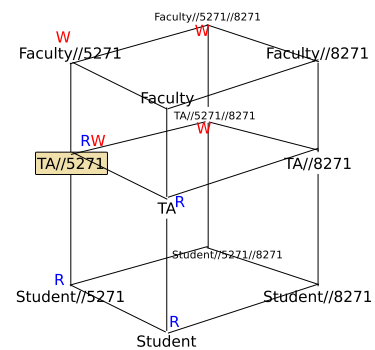
## Lattice model

- Generalize MLS levels to elements in a lattice
- BLP and Biba work analogously with lattice ordering
- No access to incomparable levels
- Potential problem: combinatorial explosion of compartments

## Classification lattice example



## Lattice BLP example



## Another notation

Faculty

→ (Faculty, ∅)

Faculty//5271

→ (Faculty, {5271})

Faculty//5271//8271

→ (Faculty, {5271, 8271})

## MLS operating systems

- 1970s timesharing, including Multics
- “Trusted” versions of commercial Unix (e.g. Solaris)
- SELinux (called “type enforcement”)
- Integrity protections in Windows Vista and later

## Multi-VM systems

- One (e.g., Windows) VM for each security level
- More trustworthy OS underneath provides limited interaction
- E.g., NSA NetTop: VMWare on SELinux
- Downside: administrative overhead

## Air gaps, pumps, and diodes

- The lack of a connection between networks of different levels is called an *air gap*
- A *pump* transfers data securely from one network to another
- A *data diode* allows information flow in only one direction

## Chelsea Manning cables leak

- Manning (née Bradley) was an intelligence analyst deployed to Iraq
- PC in a T-SCIF connected to SIPRNet (Secret), air gapped
- CD-RWs used for backup and software transfer
- Contrary to policy: taking such a CD-RW home in your pocket

<http://www.fas.org/sgp/jud/manning/022813-statement.pdf>

## Outline

Multilevel and mandatory access control

Capability-based access control

Announcements intermission

OS trust and assurance

More Unix access control

## ACLs: no fine-grained subjects

- Subjects are a list of usernames maintained by a sysadmin
- Unusual to have a separate subject for an application
- Cannot easily subset access (sandbox)

## ACLs: ambient authority

- All authority exists by virtue of identity
- Kernel automatically applies all available authority
- Authority applied incorrectly leads to attacks

## Confused deputy problem

- Compiler writes to billing database
- Compiler can produce debug output to user-specified file
- Specify debug output to billing file, disrupt billing

## (Object) capabilities

- A *capability* both designates a resource and provides authority to access it
- Similar to an object reference
  - Unforgeable, but can copy and distribute
- Typically still managed by the kernel

## Capability slogans (Miller et al.)

- No designation without authority
- Dynamic subject creation
- Subject-aggregated authority mgmt.
- No ambient authority
- Composability of authorities
- Access-controlled delegation
- Dynamic resource creation

## Partial example: Unix FDs

- Authority to access a specific file
- Managed by kernel on behalf of process
- Can be passed between processes
  - Though rare other than parent to child
- Unix not designed to use pervasively

## Distinguish: password capabilities

- Bit pattern itself is the capability
  - No centralized management
- Modern example: authorization using cryptographic certificates

## Revocation with capabilities

- Use indirection: give real capability via a pair of middlemen
- $A \rightarrow B$  via  $A \rightarrow F \rightarrow R \rightarrow B$
- Retain capability to tell R to drop capability to B
- Depends on composability

## Confinement with capabilities

- A cannot pass a capability to B if it cannot communicate with A at all
- Disconnected parts of the capability graph cannot be reconnected
- Depends on controlled delegation and data/capability distinction

## OKL4 and seL4

- Commercial and research microkernels
- Recent versions of OKL4 use capability design from seL4
- Used as a hypervisor, e.g. underneath paravirtualized Linux
- Shipped on over 1 billion cell phones

## Joe-E and Caja

- Dialects of Java and JavaScript (resp.) using capabilities for confined execution
- E.g., of JavaScript in an advertisement
- Note reliance on Java and JavaScript type safety

## Outline

Multilevel and mandatory access control  
Capability-based access control  
Announcements intermission  
OS trust and assurance  
More Unix access control

## Deadlines reminder

- Tonight: Project progress reports
- Thursday: Ex. 2
- Friday: HA1 attack(s) 5 (extra credit)
- Monday: midterm

## Midterm exam Monday

- Usual class time and location
- Covers up through today's lecture
- Mix of short-answer and exercise-like questions
- Open books/notes/printouts, no computers or other electronics
- Sample exams w/solutions (2013-2015) posted

## Outline

Multilevel and mandatory access control  
Capability-based access control  
Announcements intermission  
OS trust and assurance  
More Unix access control

## Trusted and trustworthy

- Part of your system is trusted if its failure can break your security
- Thus, OS is almost always trusted
- Real question: is it trustworthy?
- Distinction not universally observed: trusted boot, Trusted Solaris, etc.

## Trusted (I/O) path

- How do you know you're talking to the right software?
- And no one is sniffing the data?
- Example: Trojan login screen
  - Or worse: unlock screensaver with root password
  - Origin of "Press Ctrl-Alt-Del to log in"

## Minimizing trust

- Kernel → microkernel → nanokernel
- Reference monitor concept
- TCB size: measured relative to a policy goal
- Reference monitor  $\subseteq$  TCB
  - But hard to build monitor for all goals

## How to gain assurance

- Use for a long time
- Testing
- Code / design review
- Third-party certification
- Formal methods / proof

## Evaluation / certification

- Testing and review performed by an independent party
- Goal: separate incentives, separate accountability
- Compare with financial auditing
- Watch out for: form over substance, misplaced incentives

## Orange book OS evaluation

- Trusted Computer System Evaluation Criteria
- D. Minimal protection
- C. Discretionary protection
  - C2 adds, e.g., secure audit over C1
- B. Mandatory protection
  - B1 < B2 < B3: stricter classic MLS
- A. Verified protection

## Common Criteria

- International standard and agreement for IT security certification
- Certification against a *protection profile*, and *evaluation assurance level* EAL 1-7
- Evaluation performed by non-government labs
- Up to EAL 4 automatically cross-recognized

## Common Criteria, Anderson's view

- Many profiles don't specify the right things
- OSES evaluated only in unrealistic environments
  - E.g., unpatched Windows XP with no network attacks
- "Corruption, Manipulation, and Inertia"
  - Pernicious innovation: evaluation paid for by vendor
  - Labs beholden to national security apparatus

## Formal methods and proof

- Can math come to the rescue?
- Checking design vs. implementation
- Automation possible only with other tradeoffs
  - E.g., bounded size model
- Starting to become possible: machine-checked proof

## Proof and complexity

- Formal proof is only feasible for programs that are small and elegant
- If you honestly care about assurance, you want your TCB small and elegant anyway
- Should provability further guide design?

## Some hopeful proof results

- seL4 microkernel (SOSP'09 and ongoing)
  - 7.5 kL C, 200 kL proof, 160 bugs fixed, 25 person years
- CompCert C-subset compiler (PLDI'06 and ongoing)
- RockSalt SFI verifier (PLDI'12)

## Outline

Multilevel and mandatory access control

Capability-based access control

Announcements intermission

OS trust and assurance

More Unix access control

## Special case: /tmp

- We'd like to allow anyone to make files in /tmp
- So, everyone should have write permission
- But don't want Alice deleting Bob's files
- Solution: "sticky bit" 01000



## Special case: group inheritance

- When using group to manage permissions, want a whole tree to have a single group
- When 02000 bit set, newly created entries will have the parent's group
  - (Historic BSD behavior)
- Also, directories will themselves inherit 02000

## "POSIX" ACLs

- Based on a withdrawn standardization
- More flexible permissions, still fairly Unix-like
- Multiple user and group entries
  - Decision still based on one entry
- Default ACLs: generalize group inheritance
- Command line: `getfacl`, `setfacl`

## ACL legacy interactions

- Hard problem: don't break security of legacy code
  - Suggests: "fail closed"
- Contrary pressure: don't want to break functionality
  - Suggests: "fail open"
- POSIX ACL design: old group permission bits are a mask on all novel permissions

## "POSIX" "capabilities"

- Divide root privilege into smaller (~35) pieces
- Note: not real capabilities
- First runtime only, then added to FS similar to `setuid`
- Motivating example: `ping`
- Also allows permanent disabling

## Privilege escalation dangers

- Many pieces of the root privilege are enough to regain the whole thing
  - Access to files as UID 0
  - `CAP_DAC_OVERRIDE`
  - `CAP_FOWNER`
  - `CAP_SYS_MODULE`
  - `CAP_MKNOD`
  - `CAP_PTRACE`
  - `CAP_SYS_ADMIN` (`mount`)

## Legacy interaction dangers

- Former bug: take away capability to drop privileges
- Use of temporary files by no-longer `setuid` programs
- For more details: "Exploiting capabilities", Emeric Nasi