# Access Control Matrix

- List all proceses and files in a matrix
- Each row is a process (“subject”)
- Each column is a file (“object”)
- Each matrix entry is the access rights that subject has for that object

# Sample Access Control Matrix

Subjects *p* and *q*

Objects *f*, *g*, *p*, *q*

Access rights `r` (read), `w` (write), `x` (execute), `o` (owner)

| | *f* | *g* | *p* | *q* |
|---|---|---|---|---|
| *p* | rwo | r | rwx | w |
| *q* | - | r | r | rwxo |

# Other Permissions

- Append
- Delete file
- Owner (can change ACL)

# Access Control Matrix Operations

- System can transition from one ACM state to another
- Primitive operations: create subject, create object; destroy subject, destroy object; add access right; delete access right
- Transitions are, of course, conditional

# Conditional ACM Changes

Process $p$ wishes to give process $q$ read access to a file $f$ owned by $p$.

**command** *grant_read_file(p, f, q)*
  **if** $o$ **in** $a[p, f]$
  **then**
    **enter** $r$ **into** $a[q, f]$
  **fi**
**end**

# Safety versus Security

- *Safety* is a property of the abstract system
- *Security* is a property of the implementation
- To be secure, a system must be safe *and* not have any access control bugs

# Undecidable Question

- Query: given an ACM and a set of transition rules, will some access right ever end up in some cell of the matrix?
- Model ACM and transition rules as Turing machine
- Machine will halt if that access right shows up in that cell
- Will it ever halt?
- Clearly undecidable
- Conclusion: We can never tell if an access control system is safe (Harrison-Ruzzo-Ullman (HRU) result)

# Complex Access Control

- Simple user/group/other or simple ACLs don’t always suffice
- Some situations need more complex mechanisms

# Temporal Access Control

- Permit access only at certain times
- Model: time-locks on bank vaults

# Implementing Temporal Access Control

- Obvious way: add extra fields to ACL
- Work-around: timer-based automatic job that changes ACLs dynamically

# Problems and Attacks

# Problems and Attacks

- Is your syntax powerful enough for concepts like holidays? On what calendar?
- What if the clock is wrong?
- Can the enemy change the clock?
- How is the clock set? By whom or what?

# Time Protocols

```
berkshire.ntp > time.nist.gov.ntp: NTPv4 client, strat 0
time.nist.gov.ntp > berkshire.ntp: NTPv4 server, strat 1
berkshire.ntp > meow.febo.com.ntp: NTPv4 client, strat 0
meow.febo.com.ntp > berkshire.ntp: NTPv4 server, strat 2
```

# Changing the ACL

- Who changes it?
- What are the permissions on the clock daemon's tables?
- Is there a race condition at permission change time?
- What if the daemon's tables get out of sync with reality? Suppose a new file or directory is added?
- We have introduced new failure modes!

# Role-Based Access Control

- Permissions are granted to *roles*, not users
- Map users to roles
- "Any software problem can be solved by adding another layer of indirection"
- Mapping can change; should be reasonably dynamic
- Example: substitute worker; replacement worker

# Using RBAC

- RBAC is the mechanism of choice for complex situations
- Often, it isn't used where it should be, because it's more complex to set up.
- Example: giving your secretary your email password

# Using RBAC

- RBAC is the mechanism of choice for complex situations
- Often, it isn't used where it should be, because it's more complex to set up.
- Example: giving your secretary your email password
- New attack: corrupt the mapping mechanism between users and roles

# Program-Based Control

- Sometimes, there's no general enough model
- There are constraints that cannot be expressed in any table
- Common example: some forms of digital rights management (DRM), which may include forcing a user to scroll through a license agreement and then click "yes"
- It requires a program

# All Bets are Off

- Is the program correct?
- Is it secure?
- Who wrote it?
- Who can change it?
- Does it do what you want?

# Military Classification Model

- Documents are classified at a certain level
- People have certain clearances
- You’re only allowed to see documents that you’re cleared for

# Classifications

- Levels: Confidential, Secret, Top Secret
- Compartments: Crypto, Subs, NoForn
- (“NoForn” is ”No foreign nationals”)
- To read a document, you must have at least as high a clearance level *and* you must be cleared for each compartment
- Systems that support this are known as multi-level security systems

# Examples

Pat is cleared for **Secret**, *Subs*

Chris is cleared for **Top Secret**, *Planes*

We have the following files:

| | | |
|---|---|---|
| warplan | **Top Secret** | *Troops, Subs, Planes* |
| runway | **Confidential** | *Planes* |
| sonar | **Top Secret** | *Subs* |
| torpedo | **Secret** | *Subs* |

Who can read which file?

# Examples

- Pat cannot read `warplan`; she isn't cleared high enough and she doesn't have *Troops* or *Planes* clearance
- Chris can't read it, either; he doesn't have *Subs* or *Planes* clearance
- Chris can read `runway`; Pat can't
- Pat can't read `sonar`; she has *Subs* clearance but only at the **Secret** level
- She can, however, read `torpedo`

# Comparing Clearances

- Who has a higher clearance, Chris or Pat?
- Which is higher, ⟨**Secret**, *Subs*⟩ or ⟨**Top Secret**, *Planes*⟩
- Neither — they aren't comparable

# Formally Comparing Labels

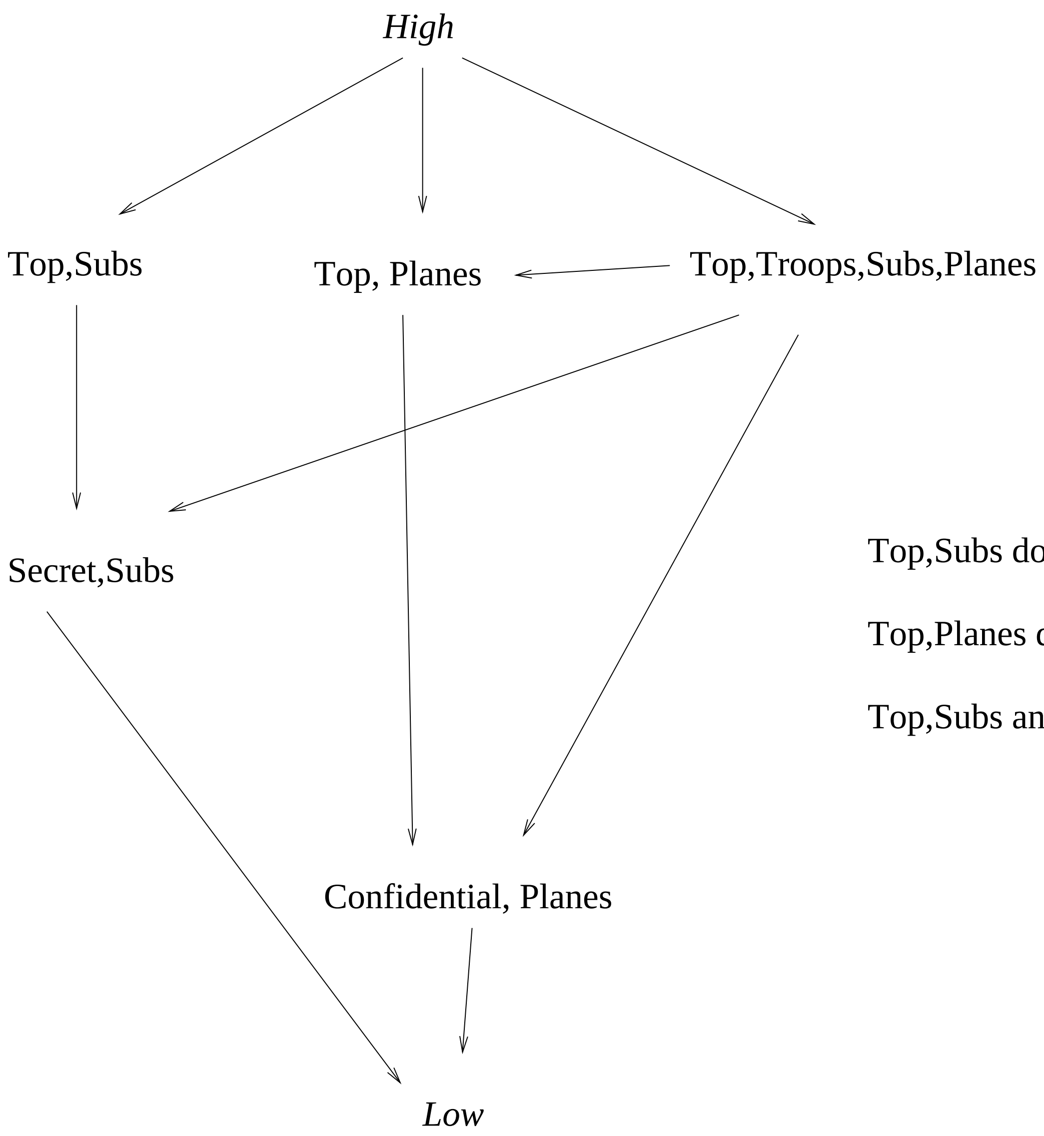
- A label is the tuple $\langle L, C \rangle$, where $L$ is the hierarchical level and $C$ is the set of compartments
- $S \geq O$ if and only if $L_S \geq L_O$ and $C_S \supseteq C_O$

# Lattices

- Clearances here are represented in a *lattice*
- A lattice is a directed graph
- We say that label $A$ *dominates* label $B$ if there is a valid path down from $A$ to $B$
- Expressed differently, if $A$ dominates $B$, information is allowed to flow from $B$ to $A$. We write $B \leq A$.
- Known as the Bell-LaPadula model

# Properties of Lattices

- Lattices are a *partial ordering*
- Lattice domination is transitive, reflexive, anti-symmetric:
  If $C \leq B$ and $B \leq A$, then $C \leq A$
  $A \leq A$
  $B \leq A$ and $A \leq B$ implies $A = B$




Top,Subs dominates Secret,Subs

Top,Planes dominates Confidential,Planes

Top,Subs and Top,Planes are not comparable

# Using this Scheme

- Processes are *subjects*
- Files are *objects*
- A process can read a file if its label dominates the file's label
- Known as "no read up"
- File labels are typically subject to mandatory access control (MAC)

# Writing Files

- Suppose there are three labels, $A$, $B$, and $C$, such that $A$ dominates $B$ and $B$ dominates $C$
- A process with label $A$ can read a file with label $B$ or label $C$ A process with label $C$ can read a file labled $C$ but not $B$
- Suppose that a process with label $A$ reads $B$ and then writes the contents to a file labeled $C$.
- Can a $C$-labeled process now read this?
- No — a process can only write to a file if the file's label dominates it
- Known as "no write down"

# That Isn't Right, Either

- Should a process at **Confidential** be able to overwrite a **Top Secret** file?
- The usual rule is that a process can only write to a file whose label is an exact match

# Formal Version

**Simple Security Condition** $S$ can read $O$ if and only if $l_o \leq l_s$

***-property** $S$ can write $O$ if and only if $l_s \leq l_o$

**Basic Security Theorem** If $\Sigma$ is a system with secure inital state $\sigma_0$ and $T$ is a set of state transitions that preserve the simple security condition, every state $\sigma_i, i \geq 0$ is secure

# Combining MAC and DAC

- The Bell-LaPadula model includes DAC as well as MAC
- Users control DAC settings; the site security officer controls the MAC values
- To read or write a file, both MAC and DAC conditions must be satisfied

# Confidentiality versus Integrity

- This scheme is geared towards confidentiality
- We can use it for integrity, too
- Make sure that all system files are labeled **Low**
- All labels dominate **Low**
- Thus, no process can write to it (“no write down”)
- Overwriting a system file appears to the access control mechanism as a confidentiality violation!
- Known as Biba integrity

# Floating Labels

- Instead of “no read up/no write down”, labels can float
- A process that reads a file acquires a label that dominates its original label and the file’s label
- When a process writes to a file, the file’s label changes as well
- Subjects and objects can have limits; if the label can’t float high enough, the output can’t take place

# Thinking Semantically

- Simpler permission schemes protect *objects*
- Bell-LaPadula schemes protect *information*
- Information flow is a dynamic concept

# Implementing Bell-LaPadula

- Does anyone actually use this stuff?
- First implemented in Multics
- Available today in Trusted Solaris
- Part of many DoD-certified systems
- But — such systems are rarely used outside of DoD, and not often within it
- The assurance process is too slow and expensive

# Exporting Labels

- Labels have to stay with the data
- Transmitted in network packets
- Printed on output
- Recorded on CDs, etc.
- What happens if a labeled CD is physically carried to — and from — a non-MLS (or otherwise untrusted) machine?

# The Commercial Uselessness of Bell-LaPadula

- Most commercial data isn't as rigidly classified as is military data
- Few commercial operating systems support it
- It's hard to transfer labels across networks, among heterogeneous systems
- *Downgrading* is hard

# Downgrading Information

- Suppose we have a web server as a front end for a sensitive database
- We can label the database **Top Secret**
- To read it, the web server needs to have **Top Secret** privileges
- But the end user — the web client — isn't trusted to that level
- Where does the downgrade operation take place?
- Downgrade is a *very* sensitive operation and can only be done by a trusted module. Is your web server that trusted?