TA: George Iordache
CC all reviews to: georgei+509r@cs.sunysb.edu

ACM
- undecidable
- ACL lists, capabilities

Bell-Lapadula
- read
- write
- MAC vs DAC

Bell-Lapadula
ex. (Top Secret, E Area 513)
(Confidential, E 3)
- ordering on labels
"Read down, write up"
For read:
Process with $l$ can read process with $l'$ if $l' \leq l$
For write:
Process with $l$ can read process with $l'$ if $l \leq l'$

Basic Security Theorem
If $E$ starts in secure state and we use read down/write up, then the system can never reach an insecure state
Two strategies:
1. Fixed process label
2. Process label = $\text{lab}(\text{labels of files read})$

files

If the output is to the screen, and the user doesn't have clearance?

**MAC** - Mandatory Access Control (system tells you)
**DAC** - Discretionary Access Control (you tell system)

**Declassification**
- Special, trusted declassification process that is allowed to break the read down/write up rule
- Combine policy and privilege
  - $\text{secret} \rightarrow \text{declassifies} \rightarrow \text{public}$
  - $\text{ex: Salaries} \rightarrow \text{average} \rightarrow \text{average salary}$
  - $\text{ex: separation of privilege}$
  - Declassifier shouldn't have bugs
  - No theorems :(  

**Biba** (Bell-Lapadula upside down) - Read up, write down
  - Double checked (DC)
  - Reliable Witness
  - Anonymous Tips
  - Internet (I)

Over time, info is
  - Less reliable
  - More secret
Role-Based Access Control (RBAC)
- Rights with roles in organization
  - Each user is assigned set of roles

<table>
<thead>
<tr>
<th>University</th>
<th>role</th>
<th>rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>student</td>
<td>submit hw, view grade</td>
<td></td>
</tr>
<tr>
<td>professor</td>
<td>read hw, write grade</td>
<td></td>
</tr>
<tr>
<td>dean</td>
<td>change grade</td>
<td></td>
</tr>
</tbody>
</table>

Hierarchical role:

\[ \text{prof} \subseteq \text{dean} \]

<table>
<thead>
<tr>
<th></th>
<th>submit hw</th>
<th>read hw</th>
<th>write grade</th>
<th>change grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>student</td>
<td>student</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>professor</td>
<td></td>
<td>professor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dean</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Static separation of duties
- No user is both professor and student

Capabilities & The Confused Deputy
- Process possesses a set of rights
- OS automatically allows access if any one of the process' rights allows it
- Process can’t decide which write is used for access

Ex: Compiler that stores billing info in special file /var/cc/txns

```bash
#!/usr/bin/cc

is setgid cc

#!/usr/bin/cc hello.c -o hello.exe ← should only write to the file if user can

#!/usr/bin/cc hello.c -o /var/cc/txns
```
Fixes:

- Check args for "/var/cc/txn"
  - doesn't scale as number of "special" files grows
  - can get out & sync
  - symlink to canonicalization
  - /var/.

- Separate privilege management from privilege enforcement.
  - App manages privileges
  - OS checks them
    - App must provide proof that access is ok to the OS

Compiles source:

```c
// write billing info
open("/var/cc/txn", CCtoken)
...

open(outfile, user_token)
fails if outfile = /var/cc/txn
```

- Compiler holds
  - two tokens: CC token, user token

Capability Implementations

- Cryptography
- Unforgeable Pointers - unix file handles