Security Part 6

Live Anonymous Q&A:
https://tinyurl.com/cs1670feedback

Windows Security

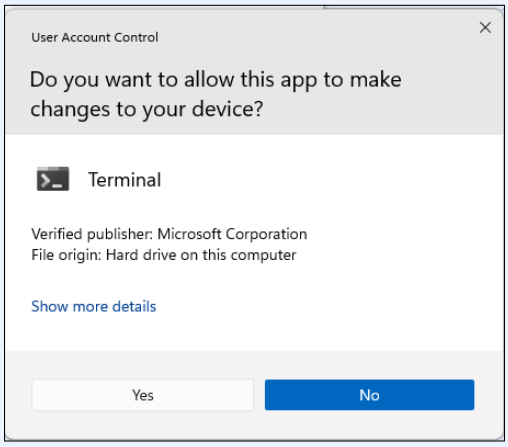
Back to Windows

- Security history
  - DOS and early Windows
    - no concept of logging in
    - no authorization
    - all programs could do everything
  - later Windows
    - good authentication
    - good authorization with ACLs
    - default ACLs are important
      - few understand how ACLs work …
    - many users ran with admin privileges
      - all programs can do everything …

Privileges in Windows

- Properties of accounts
  - administrator ≈ superuser
  - finer breakdown for service applications
- User Account Control (UAC)
  - starting with Vista
  - accounts with administrator privileges have two access tokens
    - one for normal usage
    - another with elevated rights

Windows UAC Example

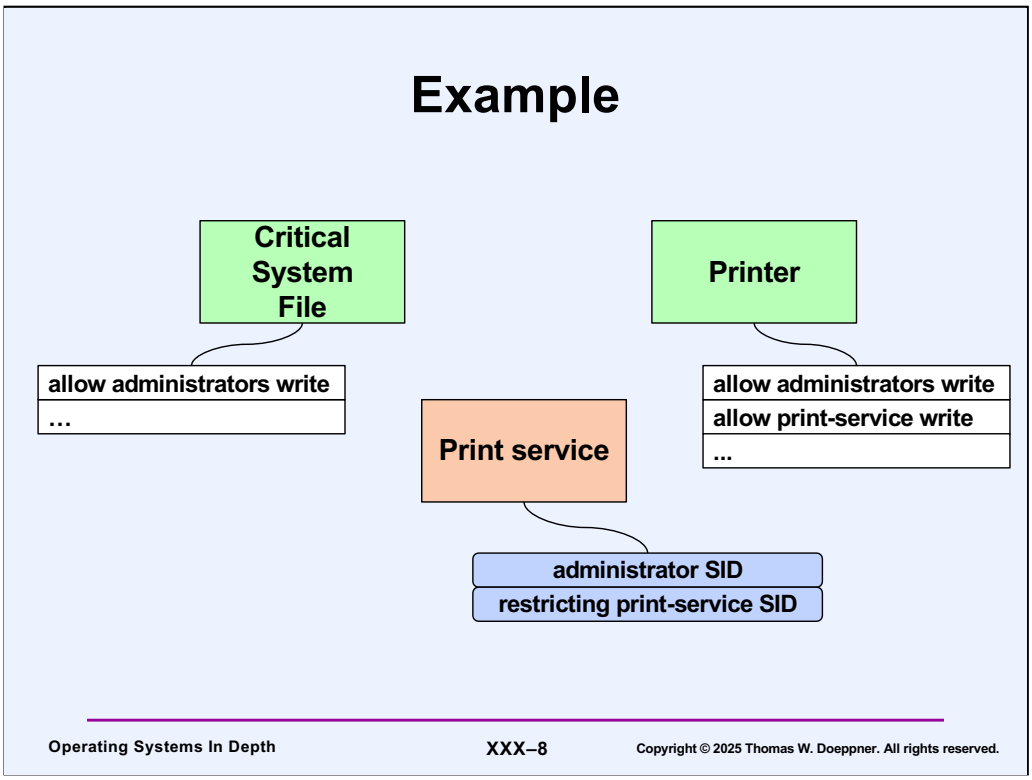


Least Privilege

- Easy answer
  - disable privileges
  - works only if the process has any …
- Another answer
  - restricting SIDs
    - limit what a server can do
    - two passes over ACL for access check
      - first: as previously specified
      - second: using only restricting SIDs

Least Privilege for Servers

- Pre-Vista:
  - services ran in local system account
    - all possible privileges
    - successful attackers “owned” system
    - too complicated to give special account to each service
- Vista and beyond
  - services still run in system account
  - per-service SIDs created
    - used in DACLs to indicate just what service needs
    - marked *restricting* in service token



Here the print service runs as the system administrator and hence would normally be able to access all files. In particular, it has write access to some critical system file, as well as to the printer. By adding a print-service SID to its access token as a restricting SID and adding an ACL entry to the printer (but not to the critical system file), we allow the print service to access the printer, but not any files that lack ACL entries allowing the print service access.

The print service must be granted access both as an administrator and as the print service.

Not a Quiz

- Why are there two passes made over the ACL?
- Answer: a restricting SID is not an additional access right, but it diminishes what can be done with existing rights
  - one must first show that one has an access right, then check if it has been diminished

Least Privilege for Clients

- Pre Vista
  - no
- Vista and beyond
  - windows integrity mechanism
    - a form of MAC

We cover this later in this lecture, when we discuss MAC.

Print Server

- Client sends request to server
  - print contents of file X
- Server acts on request
  - does client have read permission?
    - server may have (on its own) read access, but client does not
    - server might not have read access, but client does

Unix Solution

- **Client execs print-server, passing it file name**
  - **set-uid-root program**
  - **it (without races!) checks that client has access to file, then prints it**

Note that this is not *the* Unix solution, but *a* Unix solution.

Windows Solution

- Server process started when system is booted
- Clients send it print requests
  - how does client prove to server it has access?
  - how does server prove to OS that client has said ok?

Again, this is not *the* Windows solution, but *a* Windows solution.

Impersonation

- Client sends server *impersonation token*
  - subset of its access token
- Server temporarily uses it in place of its own access token

Quiz 1

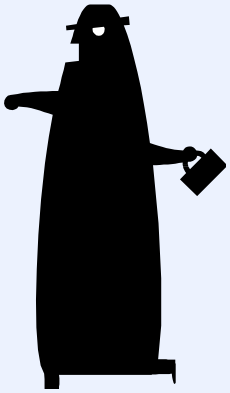
I've written a print server. You would like to use it to print a file. However, you don't trust me — you're concerned that my print server software might read some of your files that you don't want me to read. My print server uses either the Unix approach (setuid-to-twd) or the Windows approach (you send it an impersonation token) to deal with access control.

a) You have nothing to worry about
b) You have nothing to worry about if it uses the Unix approach
c) You have nothing to worry about if it uses the Windows approach
d) You have a lot to worry about with both

Security Models

Serious Security

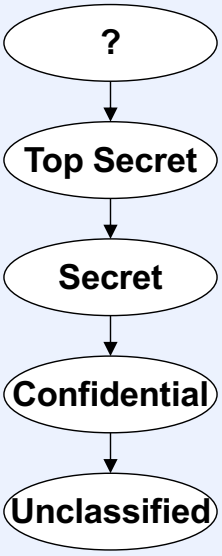
- National defense
- Proprietary information
- Personal privacy



Mandatory vs. Discretionary Access Control

- Discretionary
  - ACLs, capabilities, etc.
    - access is at the discretion of the owner
- Mandatory
  - government/corporate security, etc.
    - access is governed by strict policies

Mandatory Access Control (1)



Mandatory Access Control (2)

- Privacy/confidentiality policies
  - compartmentalization

registrar

dean of the faculty

University-affiliated hospitals

Another use of MAC is to enforce compartmentalization. For example, it might be Brown's policy that, for example, people working in the registrar's office have access to student records, but do not have access to faculty salaries. People working in the dean of the faculty's office do have access to faculty salaries, but do not have access to student records. This should continue to be the case even if someone switches jobs (but not computer IDs), moving from the registrar's office to the dean of the faculty's office. Note that this requires a notion of "role": one's role might change from being a registrar person to being a dean-of-the-faculty person.

Mandatory Access Control (3)

- Local computer policy
  - web-server
    - may access only designated web-server data
  - administrators
    - may execute only administrative programs
    - (may not execute code supplied by ordinary users)

Bell-LaPadula Model

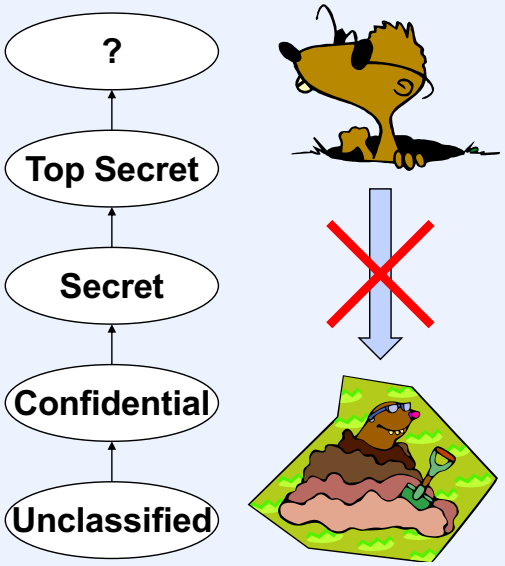
1) Simple security property **no-read-up**
   - no subject may read from an object whose classification is higher than the subject's clearance
2) *-property **no-write-down**
   - no subject may write to an object whose classification is lower than the subject's clearance

Information Black Hole

Managing Confidentiality

- Black-hole avoidance
  - trusted vs. untrusted subjects
  - trusted subjects may write down

Espionage



agent X learns of invasion plans

communication not possible

agent Y can send email to spymaster (but doesn't know what to send)

Covert Channels

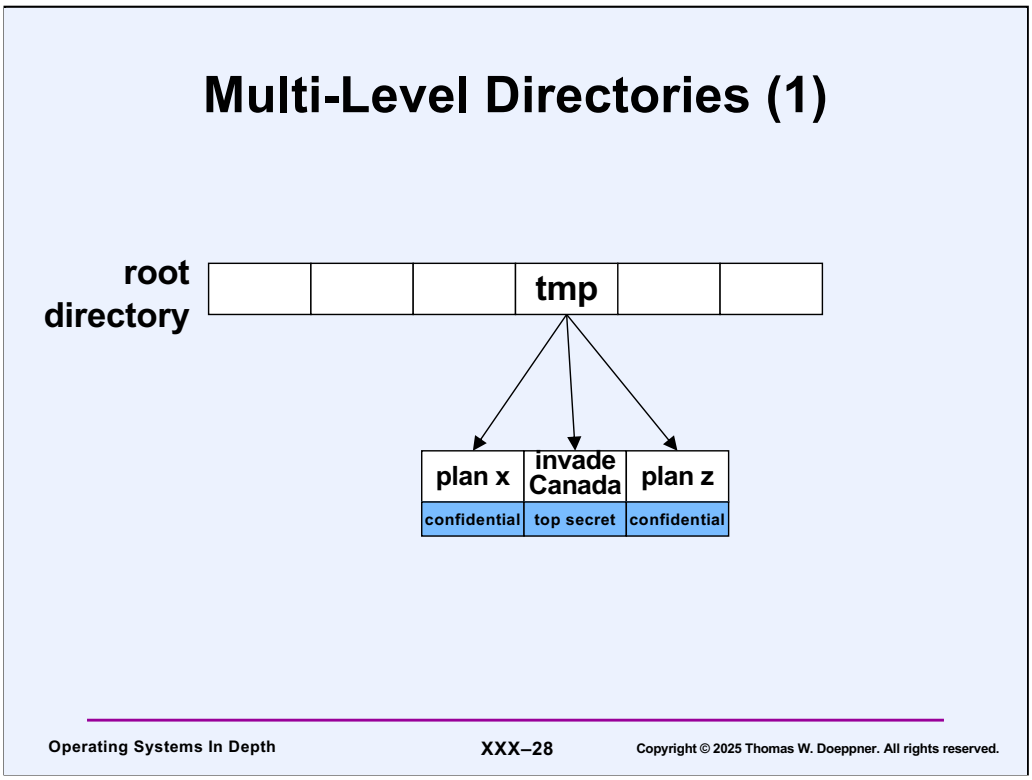
agent X runs resource-intensive program

sneaky communication possible

agent Y monitors load sends email to spymaster

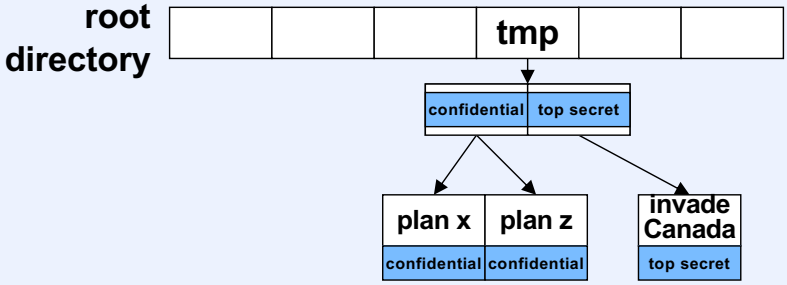
Defense

- Identify all covert channels
  - (good luck …)
- Eliminate them
  - find a suitable scheduler
    - eliminates just one channel



That there is a file named “invade y” might be considered to be information that shouldn’t be made available to just anyone. However, it’s in a directory that accessible to just anyone. We might come up with an access-permission type that prohibits those without the necessary clearance from seeing the name of a directory entry, but what if someone cleared only for confidential tries to create the file “/tmp/invade y”?

Multi-Level Directories (2)



The solution is to create an implicit subdirectory of /tmp with an entry for each classification. Thus if one is in the **top secret** domain, references to /tmp are actually to /tmp/top secret.

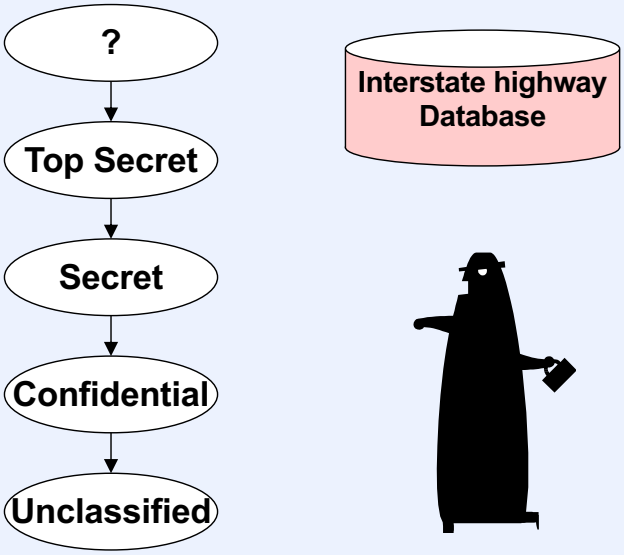
Orange Book

- Evaluation criteria for secure systems
  - D: minimal protection
  - C: discretionary protection
    - C1: discretionary security protection
    - C2: controlled access protection
  - B: mandatory protection
    - B1: labeled security protection
    - B2: structured protection
    - B3: security domains
  - A: verified protection
    - A1: verified design

The "Orange Book" (so-called because of the color of its cover) was released in 1983 and revised in 1985. Its actual title is "Department of Defense Trusted Computer System Evaluation Criteria" and is effectively a government standard on the security for standalone computer systems. Standard Unix and Windows systems, if properly set-up and administered, can achieve C2. SELinux (discussed soon) might be able to achieve B1 or better (it must be officially evaluated first).

The Orange Book requires an implementation of the Bell-LaPadula model.

Integrity



Biba Model

- Integrity is what's important
  - no-write-up
  - no-read-down

Quiz 2

You're concerned about downloading malware to your computer and very much want to prevent it from affecting your computer. Which would be the most appropriate policy to use?

a) no write up

b) no read up

c) no write down

d) no read down

Hint: you don't want code downloaded by your web browser to replace trusted code on your computer.

Windows and MAC

- Concerns
  - viruses
  - spyware
  - etc.
- Installation is an integrity concern
- Solution
  - adapt Biba model

Windows Integrity Control

- No-write-up
- All subjects and objects assigned a level
  - untrusted
  - low integrity
    - Internet Explorer/Edge
  - medium integrity
    - default
  - high integrity
  - system integrity
- Object owners may lower integrity levels
- May set *no-read-up* on an object

The integrity level of an object is stored in its SACL (system access control list).

Industrial-Strength Security

- Target:
  - embezzlers

Clark-Wilson Model

- Integrity and confidentiality aren't enough
  - there must be control over how data is produced and modified
    - well formed transactions

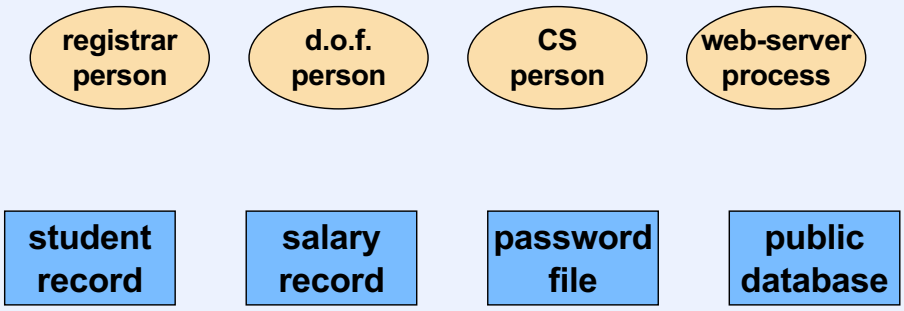
withdrawals here must be matched by entries here

- Separation of duty
  - steps of transaction must involve multiple people

Mandatory Access Control (MAC)

Implementing MAC

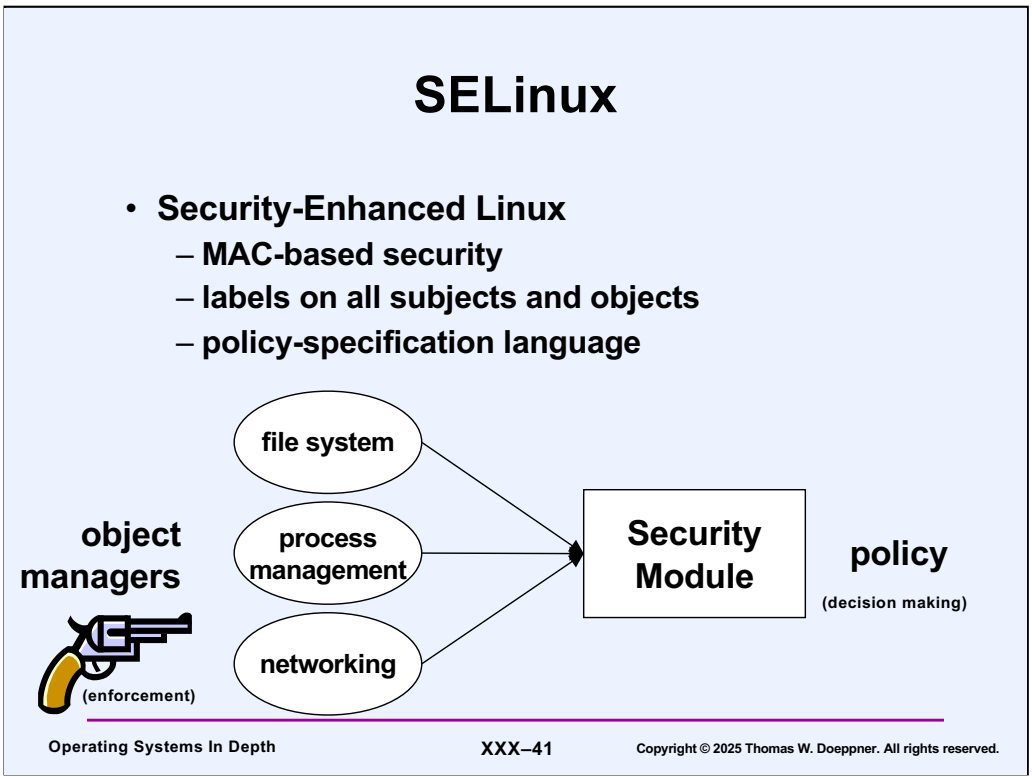
- Label subjects and objects
- Security policy makes decisions based on labels and context



Quiz 3

I have a file that I accidentally set as having rw permission for everyone (0666). You have a process that has opened my file rw. I discover this and immediately change the permissions to 0600 (access only by me). Can your process still read and write the file?

a) It can read and write
b) It can read, but not write
c) It can write, but not read
d) It can do neither



A description of SELinux is given in the paper "Integrating Flexible Support for Security Policies into the Linux Operating System" by Peter Loscocco and Stephen Smalley in the Proceedings of the 2001 USENIX Annual Technical Conference (FREENIX '01), June 2001.

In SELinux, access permissions are checked not just when a file is opened, but on every access to the file.

SELinux Examples (1)

- **Publicly readable files assigned type *public_t***
- **Subjects of normal users run in domain *user_t***
- **/etc/passwd: viewable, but not writable, by all**
- **/etc/shadow: protected**
- **SELinux rules**

```
allow user_t public_t : file read
```

  - **normal users may read public files**

```
allow passwd_t passwd_data_t : file {read write}
```

  - **/etc/shadow is of type *passwd_data_t***
  - **subjects in *passwd_t* domain may read/write /etc/shadow**

This example is covered in the textbook, starting on page 338.

Note that, unless there are additional allow statements for **passwd_data_t**, only subjects in the **passwd_t** domain may access /etc/shadow.

SELinux Examples (2)

- **How does a program get into the *passwd_t* domain?**
  - **assume passwd program is of type *passwd_exec_t***

```
allow passwd_t passwd_exec_t : file entrypoint
allow user_t passwd_exec_t : file execute
allow user_t passwd_t : process transition
type_transition user_t passwd_exec_t : process
  passwd_t
```


The first allow statement says that the domain **passwd_t** can be entered if a program of type **passwd_exec_t** is being run.

The second allow statement says that subjects in the **user_t** domain may execute programs of type **passwd_exec_t**.

The third allow statement says that it's permissible for subjects in the **user_t** domain to transition to the **passwd_t** domain.

The last statement says that when a subject in the **user_t** domain executes a program of type **passwd_exec_t**, it should attempt to switch to the the **passwd_t** domain, assuming it's permitted to do so (which it will be given the prior three statements).

Quiz 4

We've seen how the setuid feature in Unix is used to allow normal users to change their passwords in /etc/shadow.

a) This approach actually isn't secure, which is among the reasons why SELinux exists

b) The approach is secure and thus SELinux doesn't really add any additional protection to /etc/shadow

c) The approach is secure but there are other potential /etc/shadow-related vulnerabilities that SELinux helps deal with

SELinux Examples (3)

- **Accounting example**
  - **one person requests a purchase order; another approves it**
  - **files containing accounting data are of type *account_data_t***
  - **subjects accessing data are in two domains**
    - ***account_req_t***
    - ***account_approv_t***

```
allow account_req_t account_data_t : file {read
  write}
allow account_approv_t account_data_t : file
  {read write}
```


The two allow statements say that subjects in the **account_req_t** and the **account_approv_t** domains have read/write access to files of type **account_data_t**. If no other allow statements are included for **account_data_t**, then no one else has access to files of that type.

SELinux Examples (4)

- **Must specify which programs must be used to manipulate accounting data**
  - **requestPO**
    - **used to request a purchase order**
    - **type *account_req_exec_t***
  - **approvePO**
    - **used to approve purchase order**
    - **type *account_approv_exec_t***

```
allow account_req_t account_req_exec_t : file
  entrypoint
allow account_approv_t account_approv_exec_t :
  file entrypoint
```


These allow statements state that subjects may enter the **account_req_t** domain if the execute a program of type **account_req_exec_t**. Similarly for the **account_approv_t** domain.

SELinux Examples (5)

- **Who may run these programs?**

```
allow user_t account_req_t : process transition
allow user_t account_approv_t : process transition
```


This says that subjects in the **user_t** domain may switch to the **account_req_t** and **account_approv_t** domains.

Not a Quiz

- Our goal is to make sure that only certain people can request purchase orders, and only certain other people can approve purchase orders
- Do we have the machinery yet to achieve this goal?

Not yet! See the next few slides.

SELinux Examples (6)

- **Restrict usage to those users in appropriate roles**

```
role POrequester_r types account_req_t
role POapprover_r types account_approv_t

user mary roles {user_r POrequester_r}
user robert roles {user_r POapprover_r}
allow user_r {POrequester_r POapprover_r}
role_transition user_r account_req_exec_t
  POrequester_r
role_transition user_r account_approv_exec_t
  POapprover_r
```


We'd like to divide users up into the roles they perform. For this example, we have two roles: a PO requester and a PO approver. The two role statements say that the **POrequester_r** role may have users in the **account_req_t** domain, and the **POapprover_r** role may have users in the **account_approv_t** domain.

The two user statements say that mary may be both in the (ordinary) user_r role as well as in the **POrequester_r** role. And robert may be both in the **user_r** role and in the **POapprover** role.

The allow statement says that subjects in the **user_r** role may switch to the **POrequester_r** and the **POapprover_r** roles.

The **role_transition** statement says that if a subject in the **user_r** role executes a file of the **account_req_exec_t** domain, then it automatically switches to the **POrequester_r** role (assuming this is permitted — otherwise it fails). And similarly for the next statement.

SELinux Examples (7)

- **Finally …**

```
allow user_t {account_req_exec_t
  account_approv_exec_t} : file execute
```

  - **allow mary and robert to execute programs they need to run**

This allows subjects in the **user_t** domain to run programs of type **account_req_exec_t** and **account_approv_t**. But, since these programs cause transitions to the **account_req_t** and the **account_approv_t** domains, and only Mary and Robert, respectively, can be in these domains, only Mary may run can run programs of type **account_req_exec_t** and only Robert may run programs of type **account_approv_t**.

Off-the-Shelf SELinux

- Strict policy
  - normal users in *user_r* role
  - users allowed to be administrators are in *staff_r* role
    - but may run admin commands only when in *sysadm_r* role
  - policy requires > 20,000 rules
  - tough to live with
- Targeted policy
  - targets only "network-facing" applications
  - everything else in *unconfined_t* domain
  - ~11,000 rules

Capability-Based Systems

Confused-Deputy Problem

- The system has a pay-per-use compiler
  - keeps billing records in file /u/sys/comp/usage
  - puts output in file you provide
    - /u/you/comp.out
- The concept of a pay-per-use compiler annoys you
  - you send it a program to compile
  - you tell it to put your output in /u/sys/comp/usage
  - it does
    - it's confused
    - you win

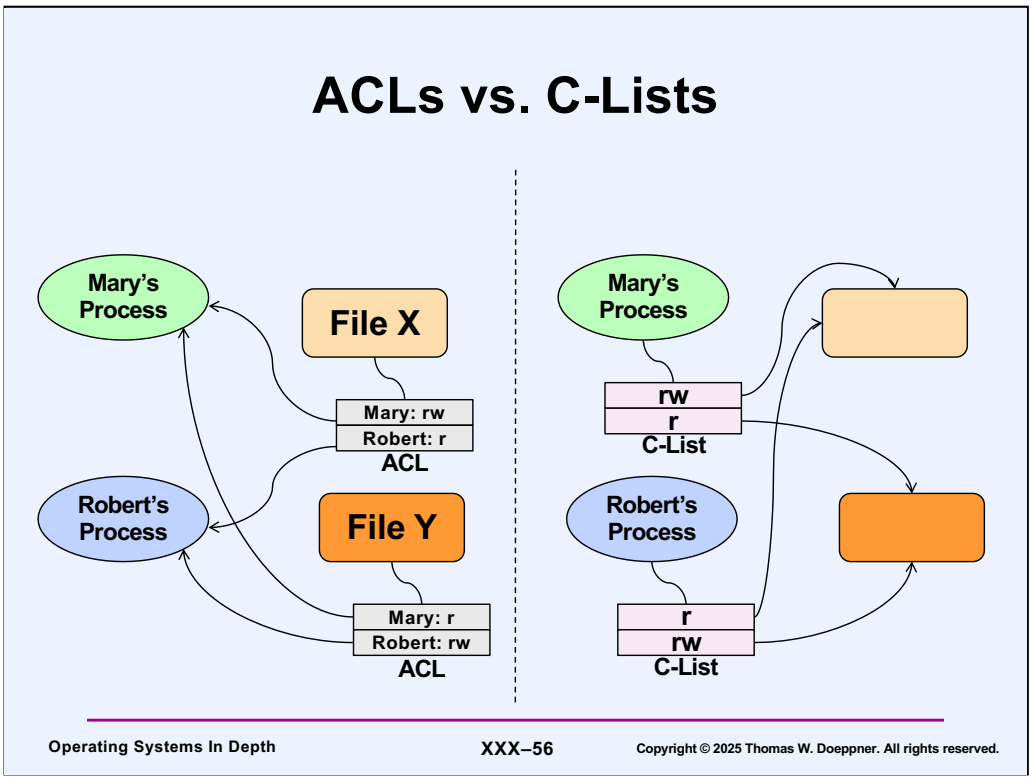
Unix and Windows to the Rescue

- Unix
  - compiler is “su-to-compiler-owner”
- Windows
  - client sends impersonation token to compiler
- Result
  - malicious deputy problem
- Could be solved by passing file descriptors
  - not done
  - should be …

The "malicious deputy" can access all your files.

Authority

- Pure ACL-based systems
  - authority depends on subject's user and group identities
- Pure capability-based systems
  - authority depends upon capabilities possessed by subject

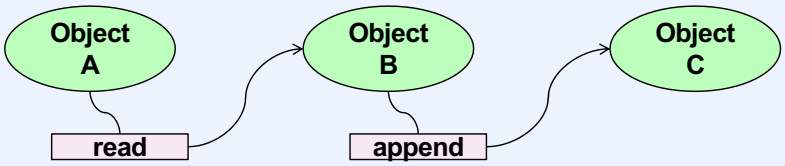


A **capability** is both a reference to a resource and an access right to that resource. Furthermore, it's unforgeable. The set of capabilities possessed by a subject is called its **C-list**. Note that with capabilities, it's not necessary for resources to have names—the capability suffices. Note that the ACLs refer to any process whose user ID is Mary or Robert, whereas a C-list merely indicates that a particular process has a capability for the indicated resource.

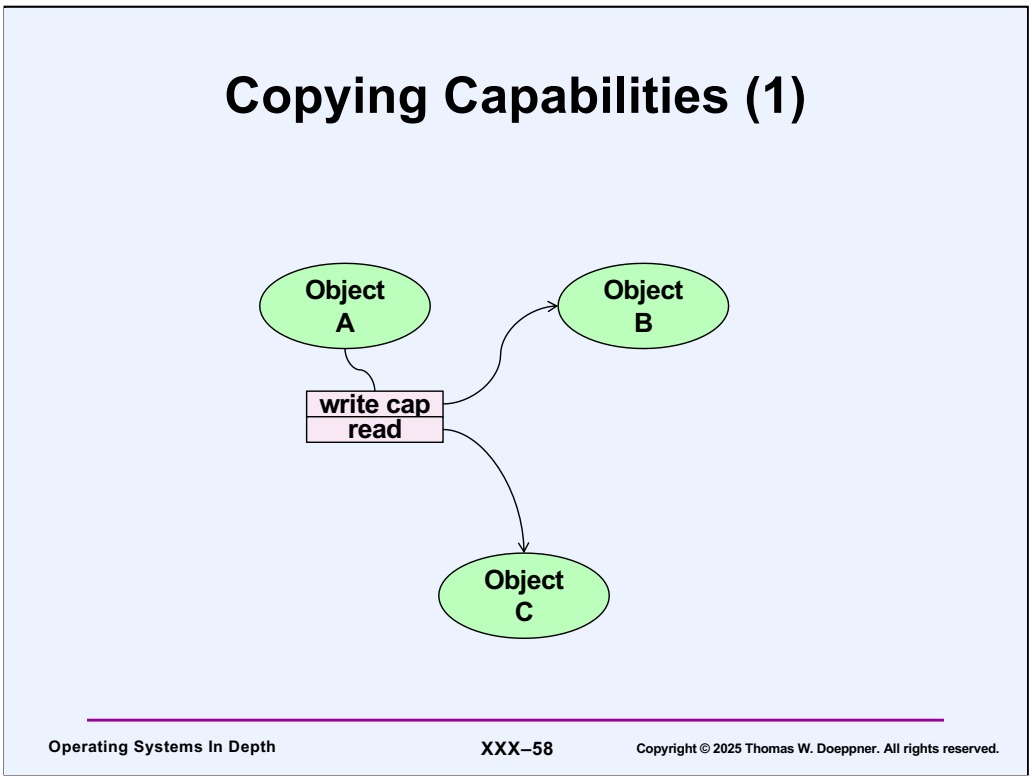
Note that, to avoid confusion with the objects of object-oriented programming, we're using the term "resource" in place of "object."

More General View

- **Subjects and resources are *objects* (in the OO sense)**

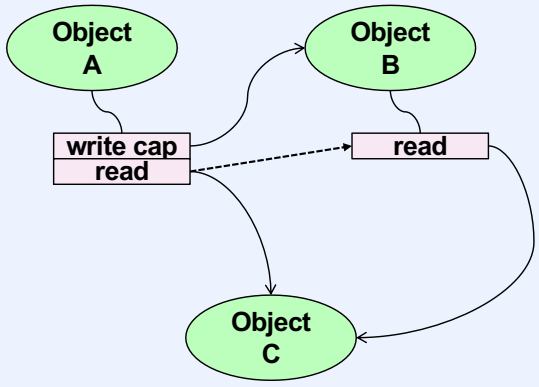


Let's think of both subjects and resources as being general objects (in the object-oriented-programming sense). A capability is a reference to an object that allows the bearer to invoke the indicated operation.

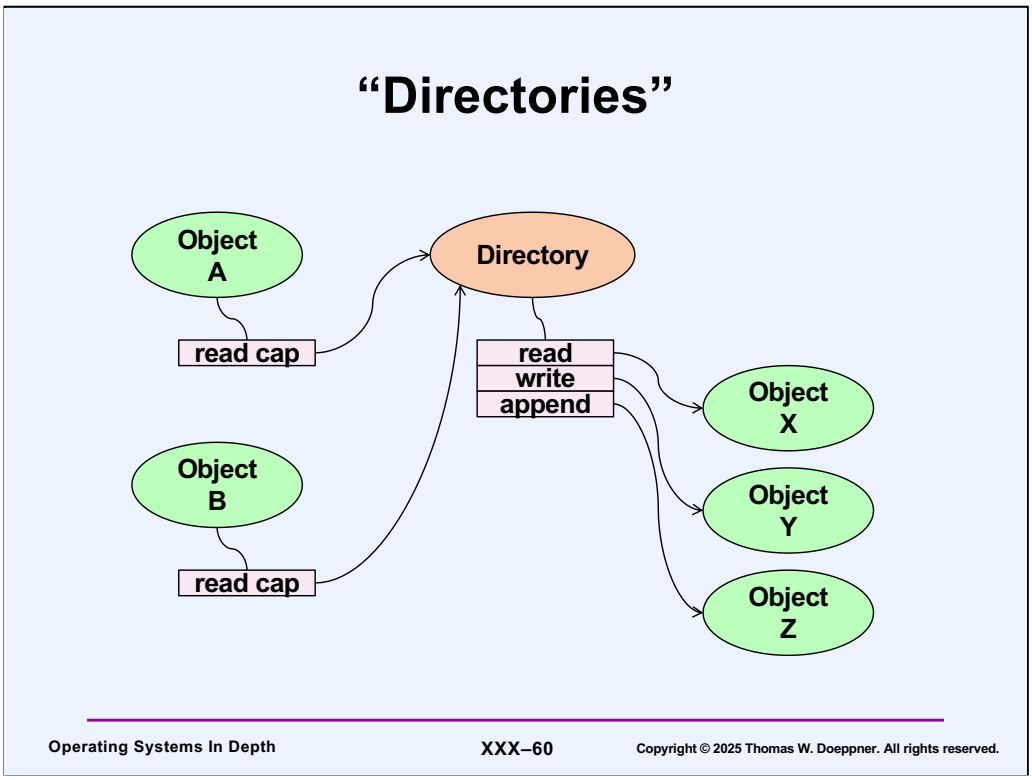


Object A has a “write-capability” capability for object B, which allows A to copy capabilities to B.

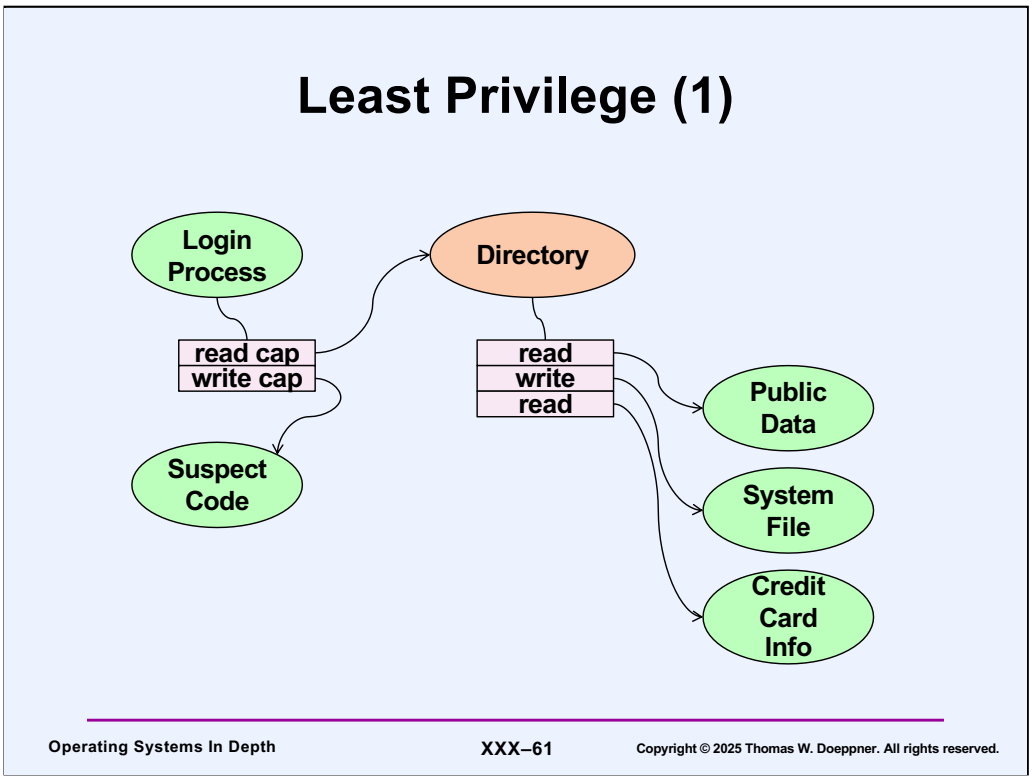
Copying Capabilities (2)



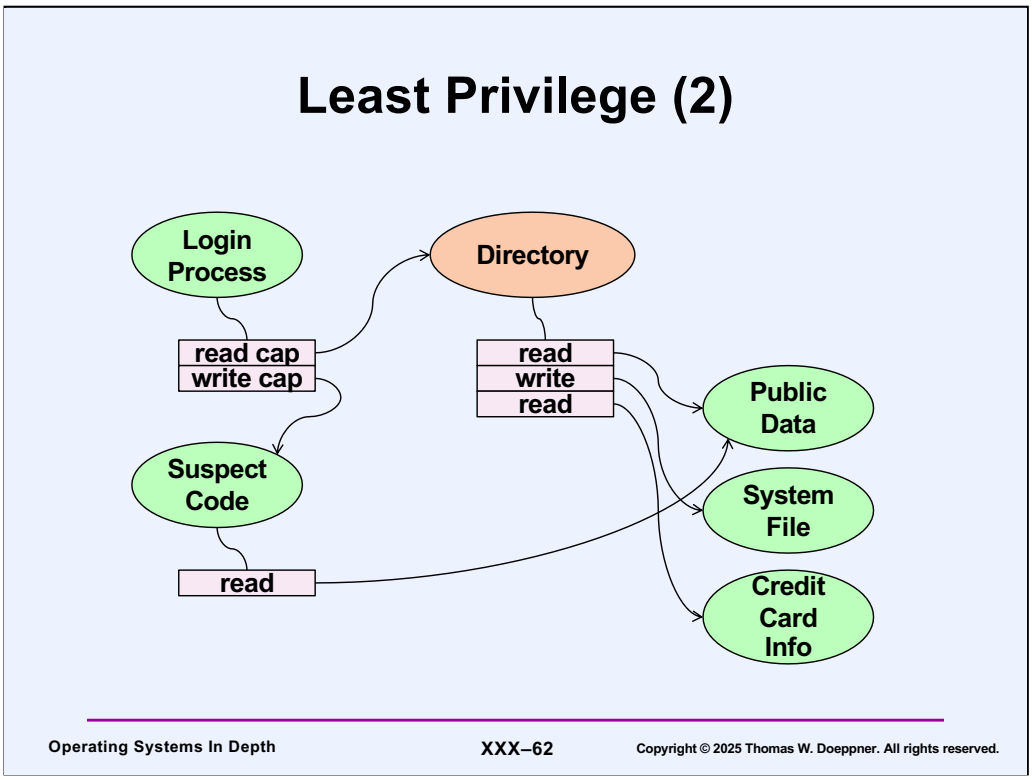
Object A has copied its “read capability for object C” to object B.



Here we have a "directory object" that provides capabilities to other objects. Object A may use its read-capability capability to fetch capabilities from the directory object.



Here we want to run a program that we've recently downloaded from the web. We create a process in which to run it, giving our login process a write-capability capability to it.



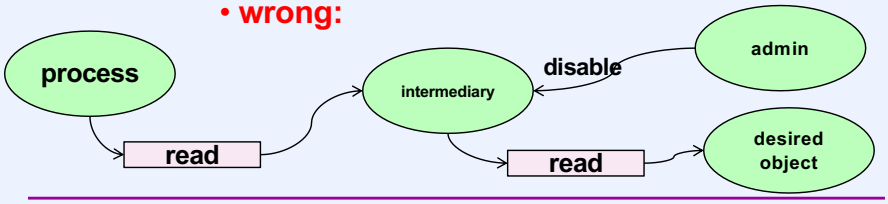
We give the new process a read capability for some public data, but no capability for anything else (and particularly no capability for getting other capabilities from the directory).

Issues

- Files aren't referenced by names. How do your processes get capabilities in the first place?
  - your "account" is your login process
    - created with all capabilities it needs
    - persistent: survives log-offs and crashes

Issues

- Can MAC be implemented on a pure capability system?
  - proven impossible twice
    - capabilities can be transferred to anyone
      - wrong: doesn't account for write-capability and read-capability capabilities
    - capabilities can't be retracted once granted
      - wrong:



Do Pure Capability Systems Exist?

- Yes!
  - long history
    - Cambridge CAP System
    - Plessey 250
    - IBM System/38 and AS/400
    - Intel iAPX 432
    - KeyKOS
    - EROS

Note that this long history doesn't have any recent examples.

A Real Capability System

- KeyKOS
  - commercial system
  - capability-based microkernel
  - used to implement Unix
    - (sort of defeating the purpose of a capability system …)
  - used to implement KeySafe
    - designed to satisfy "high B-level" orange-book requirements
    - probably would have worked
    - company folded before project finished

The KeyKOS implementation of Unix was the system that was defeated in slide XX-26.

KeySafe