Security Part 4
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)

? → Top Secret → Secret → Confidential → Unclassified
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 subject may read from an object whose classification is higher than the subject's clearance

2) *-property
   - no subject may write to an object whose classification is lower than the subject's clearance
Information Black Hole

? → Top Secret → Secret → Confidential → Unclassified

Attack!

Not cleared for top-secret orders
Managing Confidentiality

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

- Top Secret
- Secret
- Confidential
- Unclassified

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

- Top Secret
- Secret
- Confidential
- Unclassified

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 “invasion” 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/invasion”? 
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.
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

- Top Secret
- Secret
- Confidential
- Unclassified

Interstate highway Database
Biba Model

• Integrity is what’s important
  – no-write-up
  – no-read-down
Windows and MAC

- Concerns
  - viruses
  - spyware
  - etc.
- Installation is an integrity concern
- Solution
  - adapt Biba model
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
  
  ![Cash account](image1) ![Accounts-payable account](image2)

  withdrawals here          must be matched by entries here

• Separation of duty
  – steps of transaction must involve multiple people
Implementing MAC

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

- registrar person
- d.o.f. person
- CS person
- web-server process

- student record
- salary record
- password file
- public database
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.
SELinux Examples (2)

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

```plaintext
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
```
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`

```plaintext
allow account_req_t account_data_t : file {read write}
allow account_approv_t account_data_t : file {read write}
```
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
tenrypoint
  allow account_approv_t account_approv_exec_t :
    file entrypoint
SELinux Examples (5)

• Who may run these programs?

  allow user_t account_req_t : process transition
  allow user_t account_approv_t : process transition
    - normal users may, but ...
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 robert roles {user_r POrequester_r}
  user mary 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
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
Off-the-Shelf SELinux

• Strict policy
  – normal users in user_r role
  – users allowed to be administrators 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
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 ...
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.”
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.
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
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
KeySafe

Compartment → Guard
Compartment → Guard
Compartment → Guard

Objects
Security Reference Monitor