School of Electrical Engineering and Computing

SENG2250/6250
SYSTEM AND NETWORK SECURITY
(S2, 2020)

Operating System Security

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

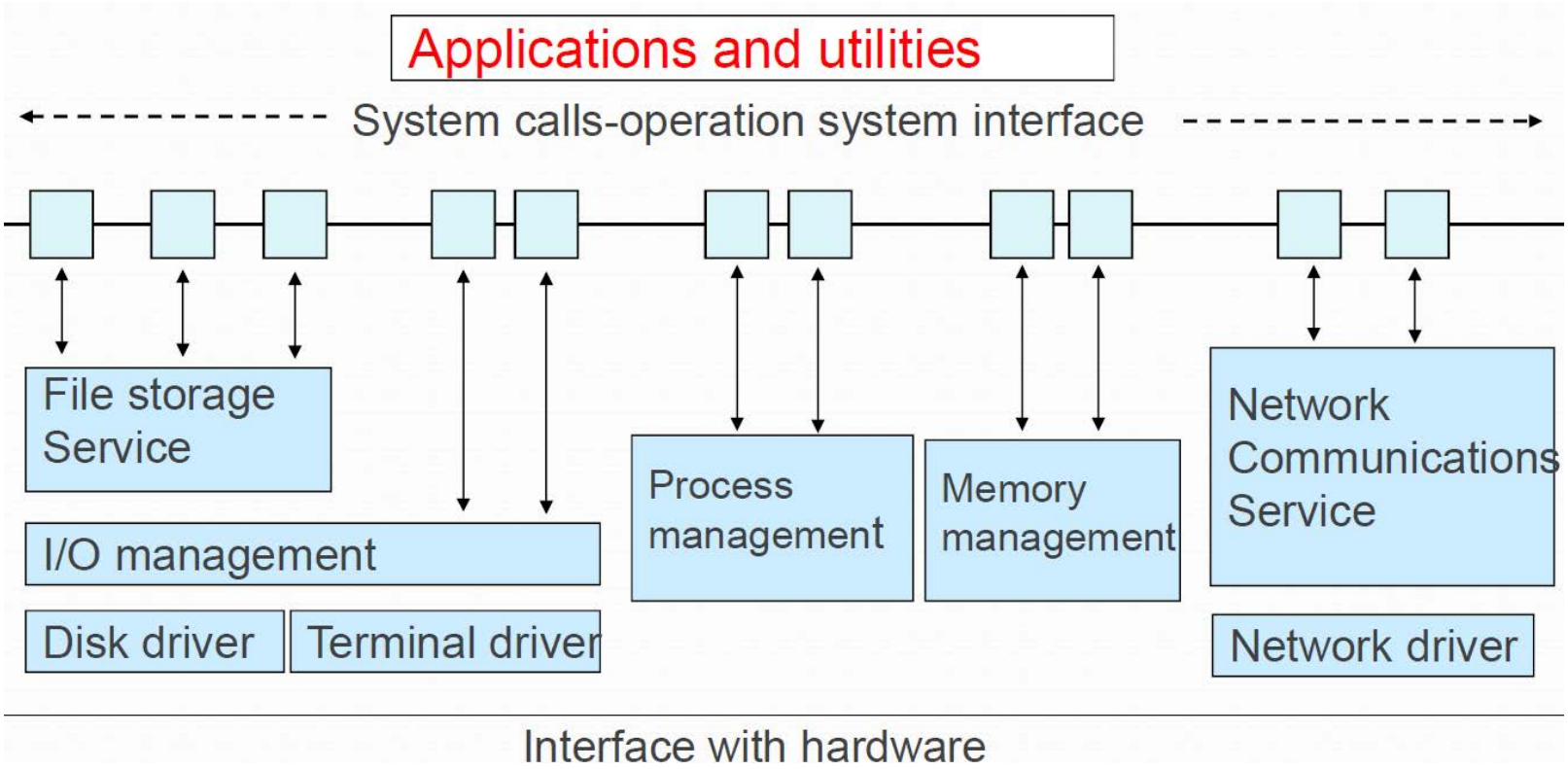
Outline

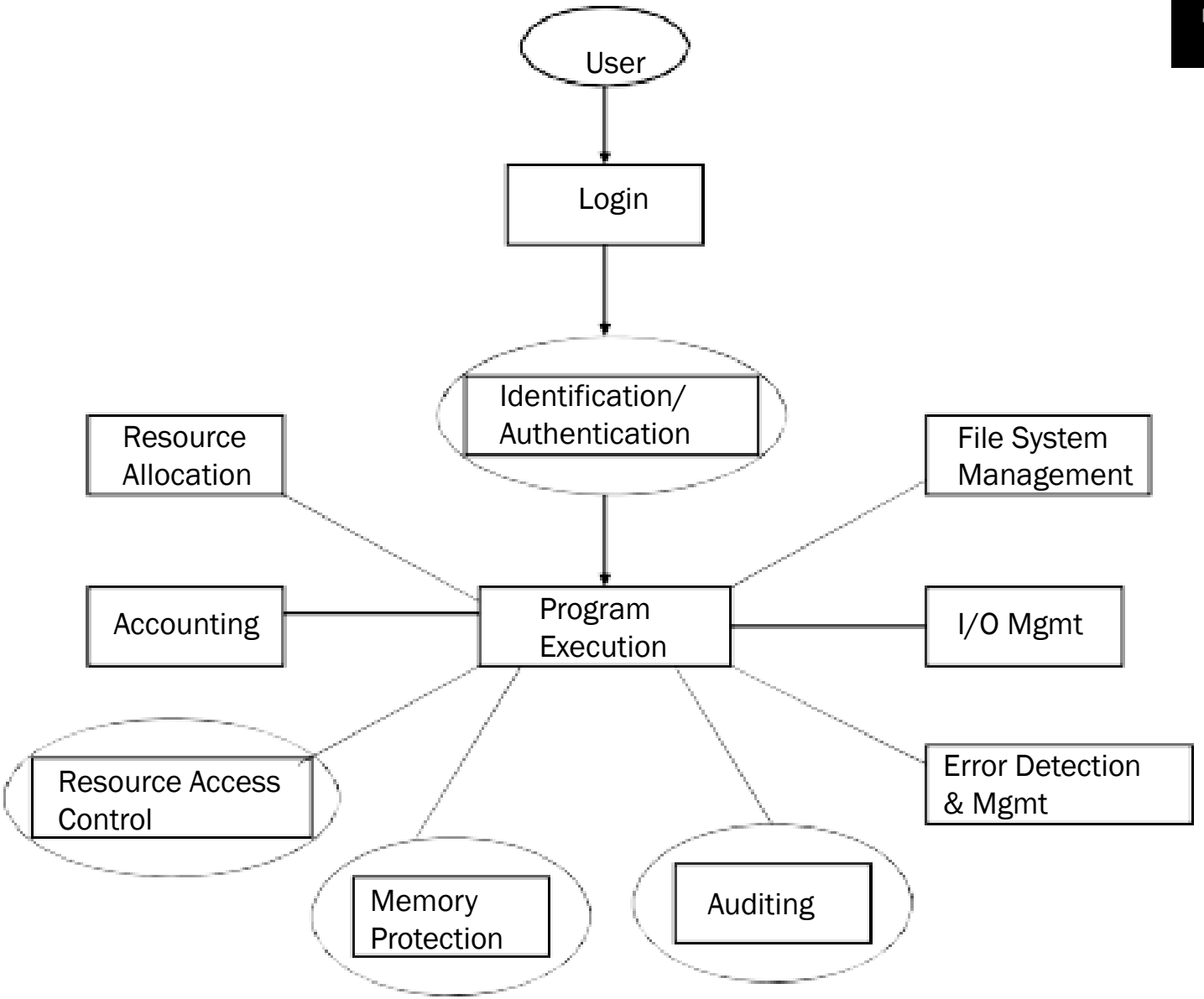
- Operating System Security
- Unix Basics
- Unix Security
  - *Access Control*
  - *File Access*
  - *Unix Access Security*

Operating Systems

- Process and Processor Management
  - *Support of Concurrent Processes*
- Resource Management
  - *Allocation of Memory, Files, I/O Devices etc to Applications*
- Supervision
  - *Interfaces with Application Programs*
  - *Supports implementation of Application Languages*
  - *Scheduling of Processes and Controls running programs*

Closed Operating System Structure




User
Login
Identification/
Authentication
Resource
Allocation
File System
Management
Accounting
Program
Execution
I/O Mgmt
Resource Access
Control
Error Detection
& Mgmt
Memory
Protection
Auditing

Operating System Security

- Authentication of Users
  - *E.g. secure login using passwords*
- Protection of Memory
  - *Each user's program must run in a portion of memory protected against unauthorized accesses*
  - *Memory protection usually involves hardware access mechanisms such as paging and segmentation*
- Protection of Files and I/O Devices
  - *Only authorised users/processes (Subjects) accessing resources containing data (Objects) → Access Matrix*

Operating System Security

- Concurrency and Synchronization Mechanisms
  - *Use of these constructs must be controlled so that one user does not have negative effect on other users*
- Guarantee of Fair Service
  - *E.g. all users expect CPU usage and other services to be provided so that no user is indefinitely starved from receiving service*
- Inter-process Communication
  - *E.g. Operating system provides services that act as bridge between processes, that are needed for synchronization or responding to processes for asynchronous communication*
  - *Inter process communications mediated by access control*

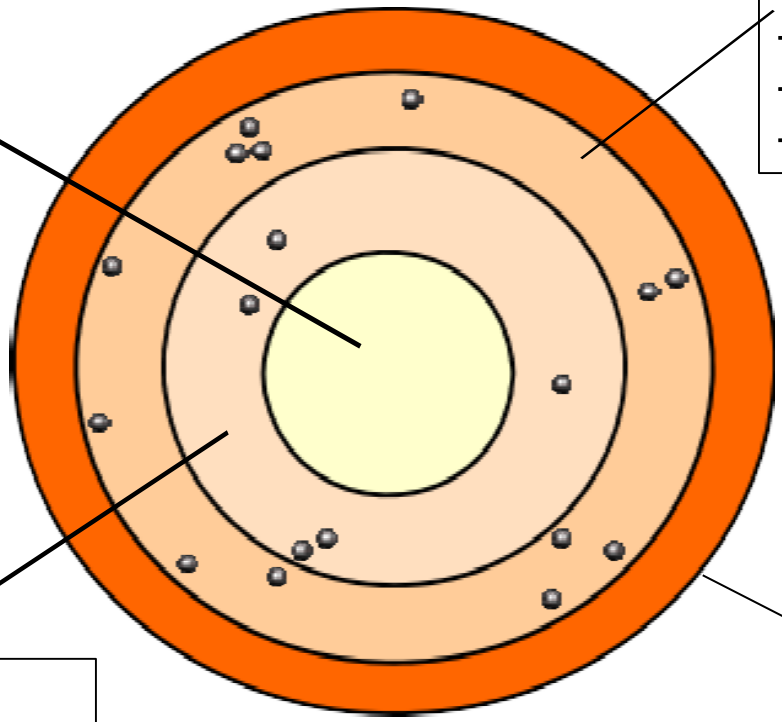
Operating System Security

- Kernel is part of an operating system that performs the lowest level functions
  - *Inter process communication, message passing, interrupt handling, synchronization etc.*
- A Security Kernel is responsible for enforcing the security mechanisms of the entire operating system
- Reference Monitor is an important portion of a Security Kernel.
  - *Controls all accesses to objects by subjects*
  - *It is not a single piece of code, but a collection of access controls for devices, files, memory, inter process communication and other objects*
- Other parts of Security Kernel include mechanisms for identification, authentication, auditing etc.

Operating System Security

- Security Kernel usually must satisfy the following properties
  - *Completeness: Kernel mediates all accesses to system objects*
  - *Isolation: Kernel must be tamper proof*
  - *Verifiability: Kernel code must be verifiable to prove it implements the security policies described by the security model*

Combined Security Kernel – Operating System



Level 1.
Hardware

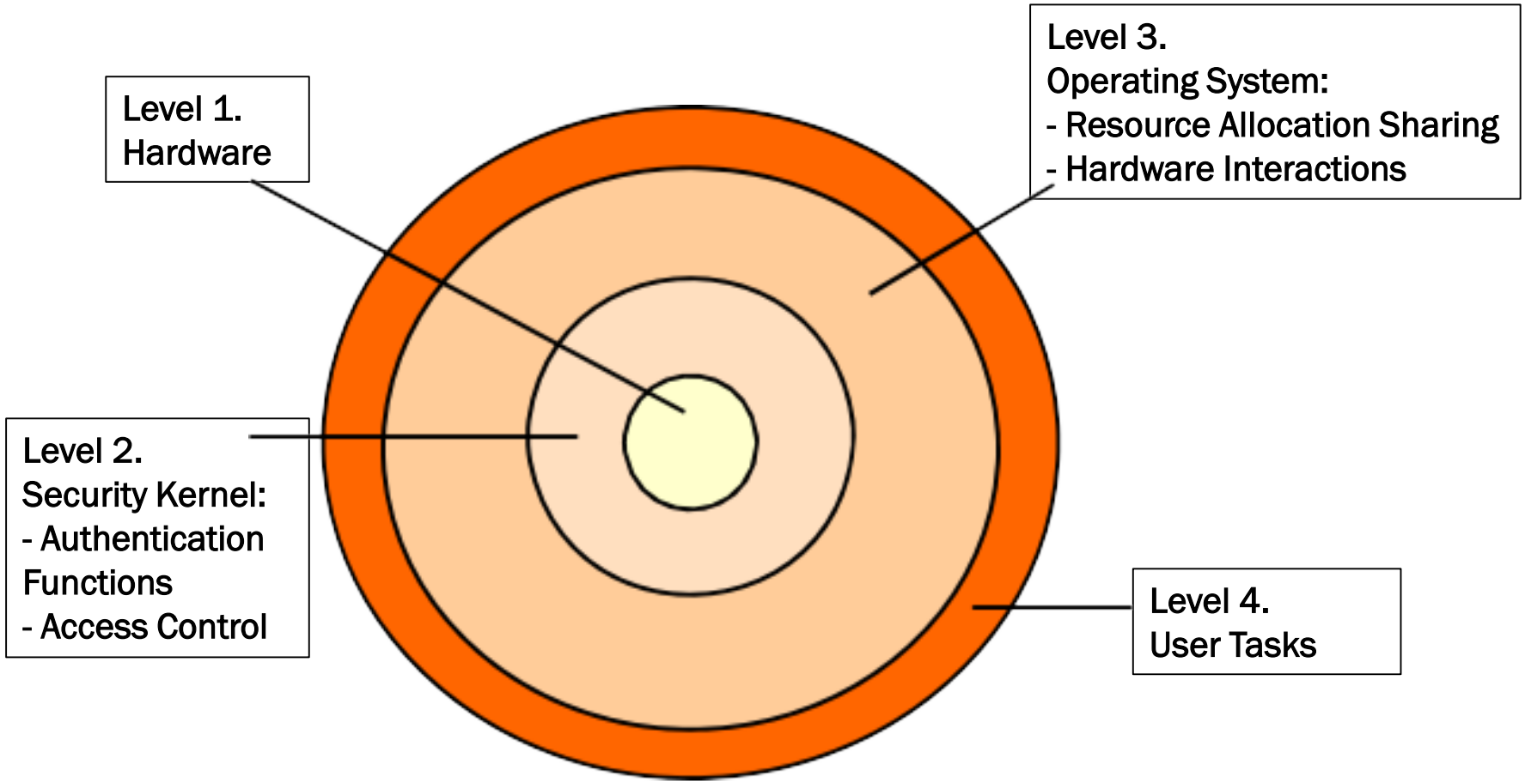
Level 3.
Operating System:
- Resource Allocation Sharing
- Access Control
- Authentication Functions

Level 2.
Operating System Kernel:
- Hardware interactions
- Access Control

Level 4.
User Tasks

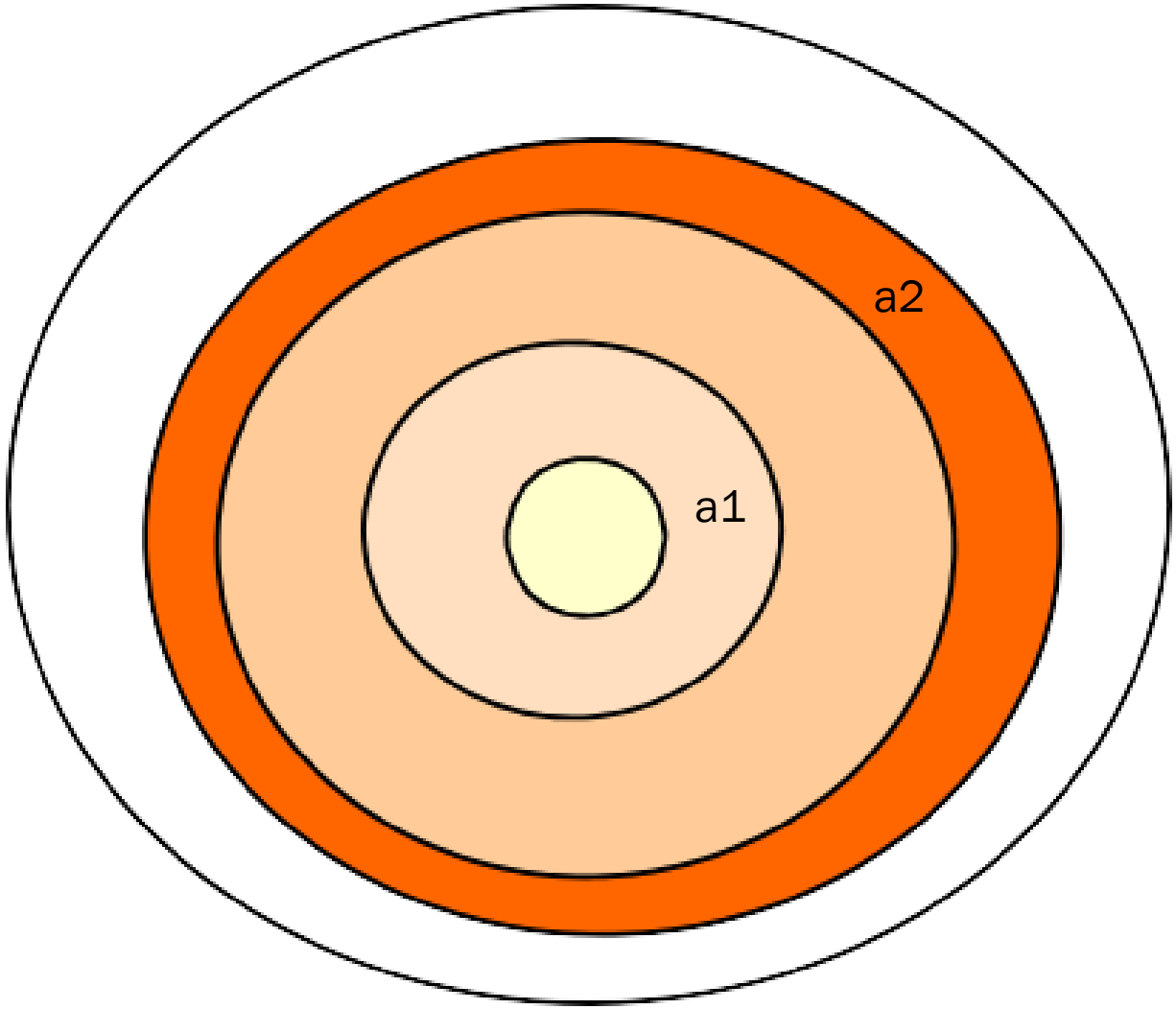
Security Activities

Separate Security Kernel



Ring Based Access Control

- Kernel resides at the lowest level, ring 0 (zero)
- Higher the ring lower the privileges the programs in that ring have.
- Lower the ring, more privileges that programs in that ring have


a2
a1

Ring Based Access Control

- Data Segment in Ring
  - *The data segment has associated with a pair of ring numbers (a1, a2) with a1 ≤ a2*
- A program running in a ring r can access this data if
  - *r ≤ a1 : access permitted*
  - *a2 < r : all accesses denied*
  - *a1 < r ≤ a2 : read and execute accesses permitted; write and append accesses denied*

Secure System Design Principles

- Least Privilege
  - *A subject (user or process) should be given only those privileges that it needs in order to complete the task*
- Default
  - *Unless given explicit access to an object, default condition is denial of access*
- Economy of Mechanism
  - *Design of secure system should be simple and small*
- Principle of Open Design
  - *Should not rely on secrecy of the design or implementation as a core aspect*

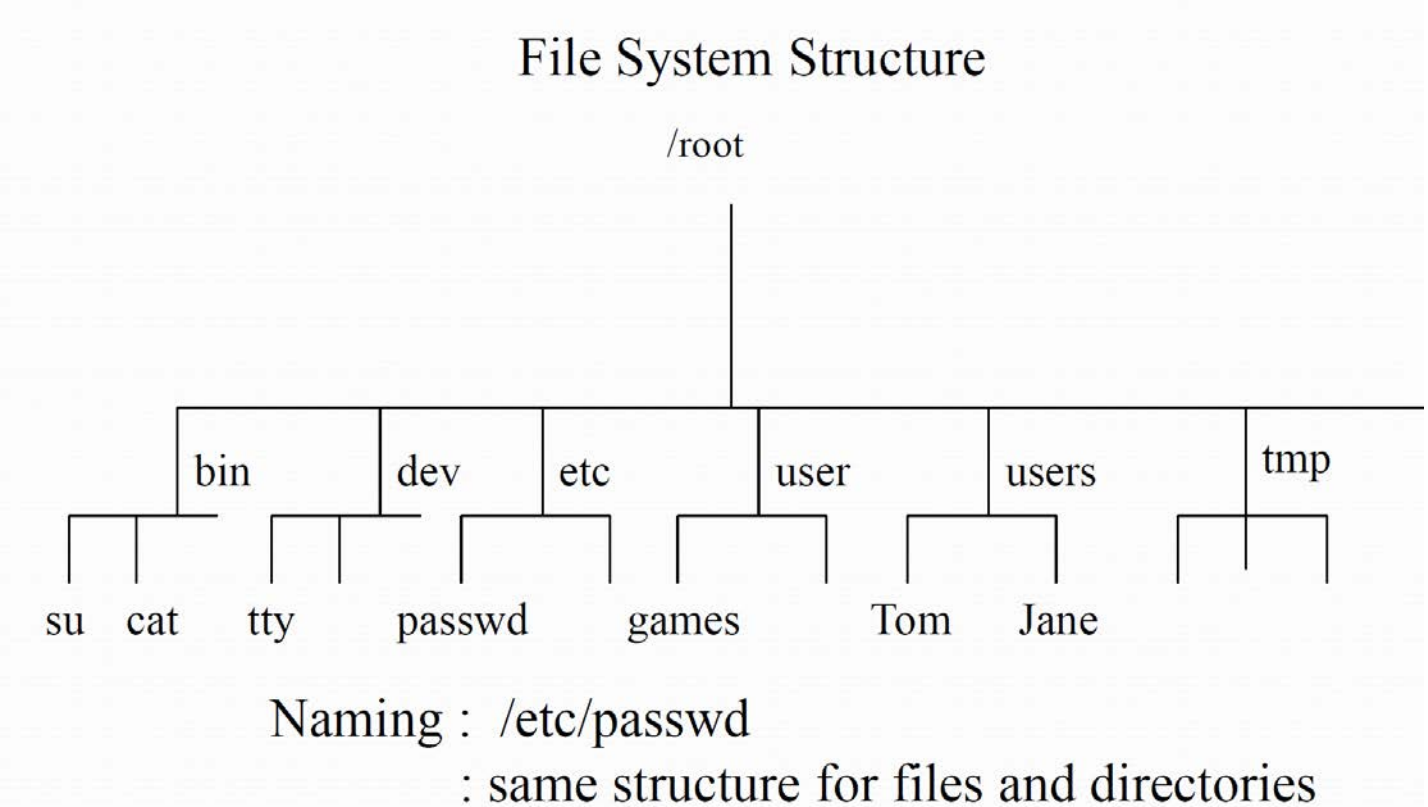
Secure System Design Principles

- Complete Mediation
  - *All accesses to objects should be checked*
- Separation of Privilege
  - *Have multiple conditions to grant permission*
  - *E.g. separation of duty*
- Least Common Mechanism
  - *Shared objects provide potential channels for information flow → Minimize such sharing*
- Ease of Use and Acceptability
  - *Otherwise security mechanisms will not be used in practice*

Unix Operating System

- Kernel : Controls input/output, allows multiple programs to run, allocates system time and memory etc.
- Utility Programs : Variety of programs
  - *E.g. List Files (/bin/ls), Copy Files (/bin/cp)…*
  - *E.g. Shells (/bin/sh, /bin/csh) (programs that users use to type commands (Unix : command language and scripting language)*
- System Database Files
  - *E.g. /etc/passwd: passwords of every user on the system*
  - *E.g. /etc/group: groups of users with similar access rights*

Unix Basics


File System Structure
/root
bin
dev
etc
user
users
tmp
su
cat
tty
passwd
games
Tom
Jane
Naming : /etc/passwd
: same structure for files and directories

Unix Security

- Security Relevant Elements
  - *Users*
  - *Groups*
  - *Processes*
  - *Files*

Unix Security

- Users
  - *Users : Login Name and Password*
  - *Authentication : Password*
  - *Some Constraints on password selection*
    - Choose mixed-case, digits, punctuation, etc
    - Change passwords frequently
  - *Checks for insecure passwords*
    - Run Password Cracker Programs
  - *Protect Password Files : One-Way Functions*
- User Identifiers (UID)
  - *Is the number that the OS uses to identify user*
  - *UIDs are 16-bit numbers*
  - *UIDs 0 to 9 are typically used for system functions*
  - *Translation between usernames and UIDs kept in /etc/passwd*

Unix Security

- Users
  - *root: super user (uid = 0) -- Unlimited set of rights*
  - *daemon: handles networks*
  - *nobody: owns no files, used as a default user for unprivileged operations*
- Users login with password
  - *Unix Password Entry Format*
    <username, encrypted passwd, UID, GID, Comments, homedir, shell>
  - *E.g.*

    root: JM0XY7tUB:0:2:Admin:/:

    *sys : * :3 :3 :Admin:/usr/admin:*

    nan : dfg87DASinqXM : 14022 : 1022 : Li,Nan: /users/nan :/usr/bin/csh

Unix Security

- Now we have shadow password files
- /etc/passwd

  nan : * : 14022 : 1022 : Li, Nan : /users/nan : /usr/bin/csh

- /etc/shadow is readable only by root

  nan : $Hash_algorithm$Salt$Password : last changed : min : max: warning: inactive : expire

  - *last changed: days since password changed*
  - *min: minimum no of days required before password changes*
  - *max: maximum no of days password is valid*
  - *warning: no of days before password expires user is warned*
  - *inactive: no of days after password expired, account is disabled*
  - *expire: no of days since account disabled, login not allowed*

Unix Security

- Groups
  - *Unix user also belongs to one or more groups*
  - *Groups have group names and group identification numbers (GIDs)*
  - *Each user belongs to a primary group that is stored in the /etc/passwd file*
  - *etc/group file contains all groups and its corresponding GID*
    - groupname : optional group password: groupid : user1, user2,..
    - Sales: : 1009:tom:alice:jack
  - *Files and Directories that need to be shared*
    - owned by Group
  - ***groups** uid (displays the groups that uid belongs to)*

Unix Security

- Rules concerning multiple group membership are different on different Unix.
  - *User can reside in only a single group at a time. (e.g. Unix System V)*
    - Use **newgrp** command to change current group
    - Change to groups that have username in /etc/group
  - *User can reside in more than one group at a time (Berkeley Unix)*
    - /bin/login scans the entire /etc/group and places the user into all the groups to which that user belongs

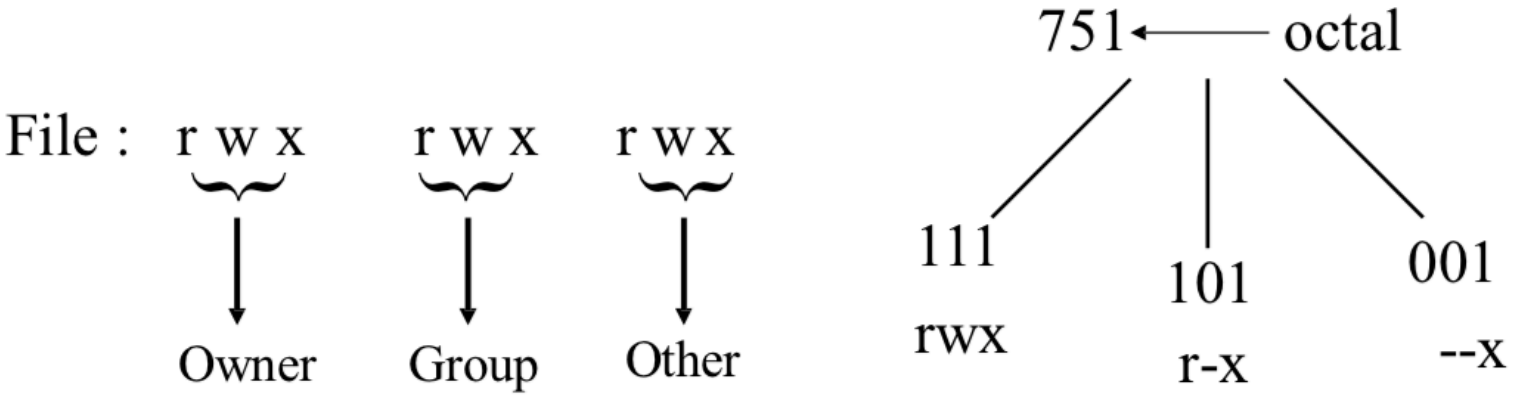
Unix Access Control

- Files
  - *File => i-node (16 bits)*
    - 12 bits : protection information
    - 4 bits: file type
- Protection bits
  - *3 bits: setuid, setgid, sticky bit*
  - *9 bits : rights (owner/group/other users "world")*

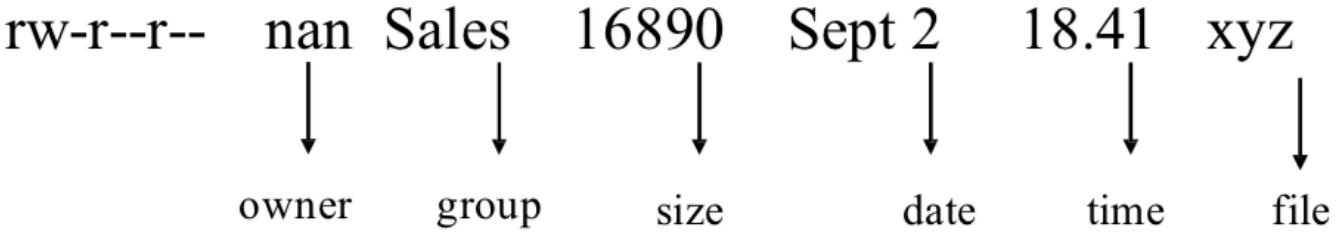
File Access Permissions

- Owner (u) : Access rights associated with the owner (typically the creator) of the file
- Group (g) : Access rights associated with the owning group (typically that of the creator) of the file
- Others (o) : Anyone else
- Permissions: read (r) , write (w), execute (x)

File Access



ls –l file



Some File Access Commands

- Change Permission (chmod), Change Owner (chown), Change Group (chgrp)

$ chmod 660 file

==> rw_rw_ _ _

$ chmod o+r file

==> rw_rw_r_ _

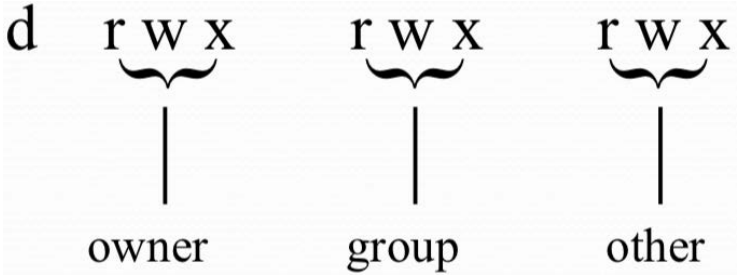
$ chown maria file $ chgrp mgr file

==> rw_rw_r_ _ maria mgr 16890 Sep 2 18:41 file

Directory Access

- r: read a directory's contents ==> ls command
  w: add/delete a directory entry ==> add/delete
  a file with w permission on dir

  ==> no w on file required
- x: search through a directory ==> change to directory/
  open files within dir/ determine owner and length of file



Unix Access Security

- Processes
  - *Process Owners → Identifiers*
    - Real User Identifier : Process Owner
    - Effective User Identifier : User whose discretionary rights are currently available for the process.
    - Real Group Identifier : A Group that the user belongs.
    - Effective Group Identifier : Group whose discretionary rights are currently available for the process.

Unix Access Security

- Program executes → Process created
- Process : Real and Effective UID, Real and Effective GID
- Process access to Files determined by Effective UID and GID
- Usually: Effective UID and GID → User's UID and GID

Unix Access Security

- setuid -- set user id
  - *When set and program is executed, process's effective UID is that of the owner of the program (rather than the user who is executing)*
  - *Process access determined by its effective UID and not real UID*
  - *setuid program has same irrespective of who executes.*

Unix Access Security

- setgid -- set group id
  - *When set and program is executed, process runs with the group access rights of the group associated with the program*

- setuid and setgid
  - *Process runs with the effective UID and GID of the owner and group owner*

Unix Access Security

- setuid/setgid
  - *Allow superuser to propagate some rights*
  - *Allows a privileged program to give users extra rights*
  - *E.g. Password program /bin/passwd owned by root with setuidbitset=>updates/etc/passwd (/etc/shadow) file when user changes password*
  - *E.g. Mail program with setuid option to root enables writing to users private mailboxes*

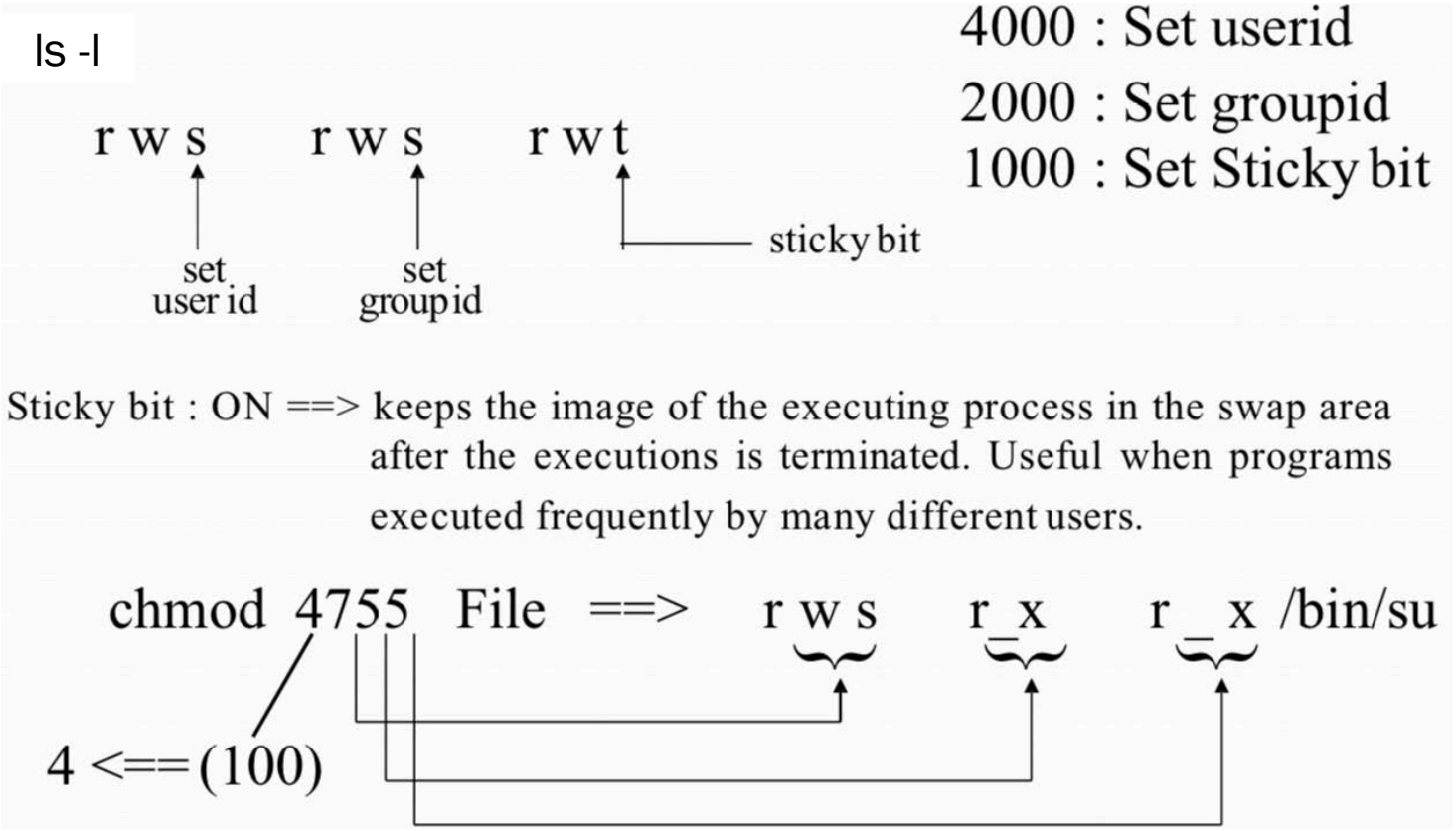
Another Example

- The GNU *ping* utility
  - *Its permissions are -rwsr-xr-x*
  - *Its owner is root*

- When you run it
  - *You become root to perform the executable code*
  - *You need this to access low level system interfaces to preform some socket operations (e.g. SOCK_RAW)*

Special Access Permissions

ls -l



Unix Access Security

- Warning
  - *If file owned by root is setuid*

    ---> during execution ---> all users get superuser privileges

  - *Files belonging to owners of setuid/setgid program must be protected against this same program.*

Unix Access Security

- setuid/setgid : some precautions
  - *A setuid/setgid program must be realised as a secure program*
    - Should only be able to perform functions that were compiled into it
- setuid/setgid commands : via secure channel to the kernel
- setuid/setgid disabled when writing a file
- System administrator verifies setuid/setgid bits of desired programs
- Avoid shell scripts with setuid

Principle of Least Privilege

- Every program and every user of the system should operate using the least set of privileges necessary to complete the job
  - *Reduce the damage that can occur if the code be exploited by a malicious user.*
- Issues to consider when writing a privileged program
  - *Does the program need special privileges?*
  - *Does the program need all the privileges?*
  - *Does the program need the privileges now?*
  - *Does the program need the privileges in the future?*