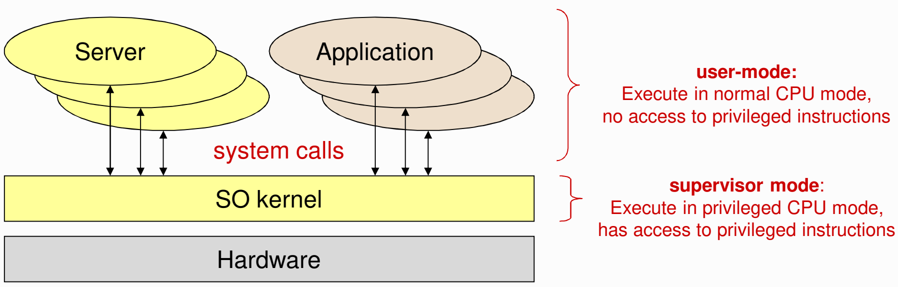
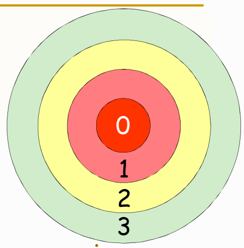
Security in Operating Systems

Operating system



- Kernel mission
  - Virtualize the hardware
    - Computational model
  - Enforce protection policies and provide protection mechanisms
    - Against involuntary mistakes
    - Against non-authorized activities

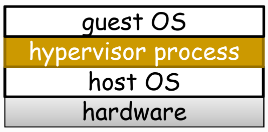
Execution rings



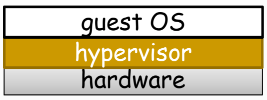
- ▷ Different levels of privilege
  - Forming a set of concentric rings
  - Used by CPU's to prevent non-privileged code from running privileged opcodes
    - e.g. IN/OUT, TLB manipulation
- ▷ Nowadays processors have 4 rings
  - But OS's usually use only 2
    - 0 (supervisor/kernel mode) and 3 (user-mode)
- ▷ Transfer of control between rings requires special gates
  - The ones that are used by syscalls

Virtual machines and hypervisors

- ▷ Emulation of a particular (virtual) hardware with the existing one (real)



- ▷ Hosted virtualization
  - The hypervisor is a process of a given OS (host)
  - The VM runs inside the virtualizer (guest OS)

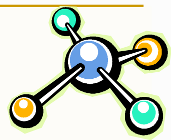


- ▷ Bare-metal virtualization
  - The hypervisor runs on top of the host hardware

Execution of virtual machines

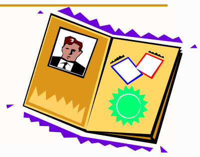
- Common approach for hosted virtualization
  - Software-based virtualization
  - Direct execution of guest user-mode code
  - Binary, on-the-fly translation of privileged code (full virtualization)
    - Guest OS kernels remain unchanged
    - No direct access to the host hardware
- Hardware-assisted virtualization (bare-metal)
  - Full virtualization
  - There is a ring -1 below ring 0
    - Hypervisor (or Virtual Machine Monitor, VMM)
  - It can virtualize hardware for many ring 0 kernels
    - No need of binary translation
    - Guest OS's run faster

Computational model



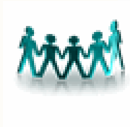
- Set of entities (objects) managed by the OS kernel
  - User identifiers
  - Processes
  - Virtual memory
  - Files and file systems
  - Communication channels
  - Physical devices
    - Storage
      - Magnetic disks, optical disks, silicon disks, tapes
    - Network interfaces
      - Wired, wireless
    - Human-computer interfaces
      - Keyboards, graphical screens, text consoles, mice
    - Serial/parallel I/O interfaces
      - USB, serial ports, parallel ports, infrared, bluetooth

Computational model: User identifiers



- For the OS kernel a user is a number
  - Established during a login operation
  - User ID (UID)
- All activities are executed on a computer on behalf of a UID
  - The UID allows the kernel to assert what is allowed/denied to processes
  - Linux: UID 0 is omnipotent (root)
    - Administration activities are usually executed with UID 0
  - Windows: concept of privileges
    - For administration, system configuration, etc.
    - There is no unique, well-known identifier for and administrator
    - Administration privileges can be bound to several UIDs
      - Usually through administration groups
      - Administrators, Power Users, Backup Operators

Computational model: Group identifiers



- Groups also have an identifier
  - A group is a set of users
  - A group can be defined by including other groups
  - Group ID (GID)
- A user can belong to several groups
  - Rights = UID rights + rights of his groups
- In Linux all activities are executed on behalf of a set of groups
  - Primary group
    - Typically used for setting file protection
  - Secondary groups

Computational model: Processes

- A process defines the context of an activity
  - For taking security-related decisions
  - For other purposes (e.g. scheduling)
- Security-related context
  - Identity (UID and GIDs)
    - Fundamental for enforcing access control
  - Resources being used
    - Open files
      - Including communication channels
    - Reserved virtual memory areas
    - CPU time used

Access control

- The OS kernel is an access control monitor
  - Controls all interactions with the hardware
  - Controls all interactions between entities of the computational model
- Subjects
  - Usually local processes
    - Through the system call API
    - A system call (or syscall) is not an ordinary function call
  - But also messages from other hosts

Mandatory access controls

- OS kernels have plenty mandatory access control policies
  - They are part of the computational model logic
  - They cannot be overruled not even by administrators
    - Unless they change the OS kernel behavior
- Examples:
  - Kernel runs in CPU privileged modes, user applications run in non-privileged modes
  - Separation of virtual memory areas
  - Inter-process signaling
  - Interpretation of files' ACLs

Protection with ACLs

- Each object has an ACL
  - It says which subjects can do what
- An ACL can be discretionary or mandatory
  - When mandatory it cannot be modified
  - When discretionary it can be tailored
- An ACL is checked when an activity, on behalf of a subject, wants to manipulate the object
  - Ifs the manipulation request is not authorized by the ACL, the access is denied
  - The SO kernel is the responsible for enforcing ACL-based protection
    - It acts as a security monitor

Protection with capabilities

- Less common in normal OS kernels
  - Though there are some good examples
- Example: open file descriptors
  - Applications' processes indirectly manipulate open file descriptors through the OS kernel
    - Using integer indexes (also called file descriptors …)
    - The OS kernel has full control over the contents of open file descriptors
  - Open file descriptors can only be granted to other processes through the OS kernel
    - Not really a usual operation, but possible!
  - Changes in the protection of files does not impact existing open file descriptors
    - The access rights are evaluated and memorized when the file is open

Unix file protection ACLs: Fixed-structure, discretionary ACL

- Each file system object has an ACL
  - Binding 3 rights to 3 subjects
  - Only the owner can update the ACL
- Rights: **R W X**
  - Read right / Listing right
  - Write right / create or remove files or subdirectories
  - Execution right / use as process' current working directory
- Subjects:
  - An UID (owner)
  - A GID
  - Others

Windows NTFS file protection: Variable-size, discretionary ACLs

- Each file system object has an ACL and a owner
  - **The ACL grants 14 types of access rights to a variable-size list of subjects**
  - **Owner can be an UID or a GID**
  - **Owner has no special rights over the ACL**
- Subjects:
  - **Users (UIDs)**
  - **Groups (GIDs)**
    - The group "Everyone" stands for anybody
- Rights:
  - **Traverse Folder / Execute File**
  - **List Folder / Read Data**
  - **Read Attributes**
  - **Read Extended Attributes**
  - **Create Files /Write Data**
  - **Create Folders / Append Data**
  - **Write Attributes**
  - **Write Extended Attributes**
  - **Delete Subfolders and Files**
  - **Delete**
  - **Read Permissions**
  - **Change Permissions**
  - **Take Ownership**

Privilege elevation: Set-UID mechanism

- It is used to change the UID of a process running a program stored on a Set-UID file
  - If the program file is owned by UID X and the set-UID ACL bit is set, then it will be executed in a process with UID X, independently of the UID of the subject that executed the program
- It is used to provide privileged programs for running administration task invoked by normal, untrusted users
  - Change the user's password (passwd)
  - Change to super-user mode (su, sudo)
  - Mount devices (mount)

Privilege elevation: Set-UID mechanism (cont.)

- ▷ Effective UID / Real UID
  - Real UID is the UID of the process creator
    - App launcher
  - Effective UID is the UID of the process
    - The one that really matters for defining the rights of the process
- ▷ UID change
  - Ordinary application
    - eUID = rUID = UID of process that executed **exec**
    - eUID cannot be changed (unless = 0)
  - Set-UID application
    - eUID = UID of **exec**'d application file, rUID = initial process UID
    - eUID can revert to rUID
  - rUID cannot change

Privilege elevation: Set-UID/Set-GID decision flowchart

- ▷ exec (path, …)
  - File referred by path has Set-UID?
  - Yes
    - ID = path owner
    - Change the process effective UID to ID
  - No
    - Do nothing
  - File referred by path has Set-GID?
  - Yes
    - ID = path GID
    - Change the process GIDs to ID only
  - No
    - Do nothing

Privilege elevation: sudo mechanism

- Administration by root is not advised
  - One "identity", many people
  - Who did what?
- Preferable approach
  - Administration role (uid = 0), many users assume it
    - Sudoers
    - Defined by a configuration file used by sudo
- sudo is a Set-UID application with UID = 0
  - Appropriate logging can take place on each command run with sudo

Privilege reduction: chroot mechanism (or jail)

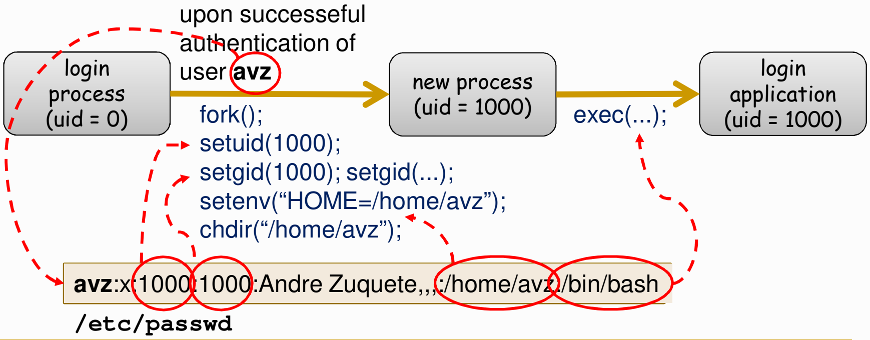
- Used to reduce the visibility of a file system
  - Each process descriptor has a root i-node number
    - From which absolute pathname resolution takes place
  - chroot changes it to an arbitrary directory
    - The process' file system view gets reduced
- Used to protect the file system from potentially problematic applications
  - e.g. public servers, downloaded applications
  - But it is not bullet proof!

Linux login:
Not an OS kernel operation

- A privileged login application presents a login interface for getting users' credentials
  - A username/password pair
  - Biometric data
  - Smartcard and activation PIN
- The login application validates the credentials and fetches the appropriate UID and GIDs for the user
  - And starts an initial user application on a process with those identifiers
    - In a Linux console this application is a shell
  - When this process ends the login application reappears
- Thereafter all processes created by the user have its identifiers
  - Inherited through forks

Linux: from login to session processes

- The login process must be a privileged process
  - Has to create processes with arbitrary UID and GIDs
    - The ones of the entity logging in



Login in Linux: Password validation process

- Username is used to fetch a UID/GID pair from /etc/passwd
  - And a set of additional GIDs in the /etc/group file
- Supplied password is transformed using a digest function
  - Currently configurable, for creating a new user (/etc/login.defs)
  - Its identification is stored along with the transformed password
- The result is checked against a value stored in /etc/shadow
  - Indexed again by the username
  - If they match, the user was correctly authenticated

- File protections
  - /etc/passwd and /etc/group can be read by anyone
    - This is fundamental, for instance, for listing directories (why?)
  - /etc/shadow can only be read by root
    - Protection against dictionary attacks