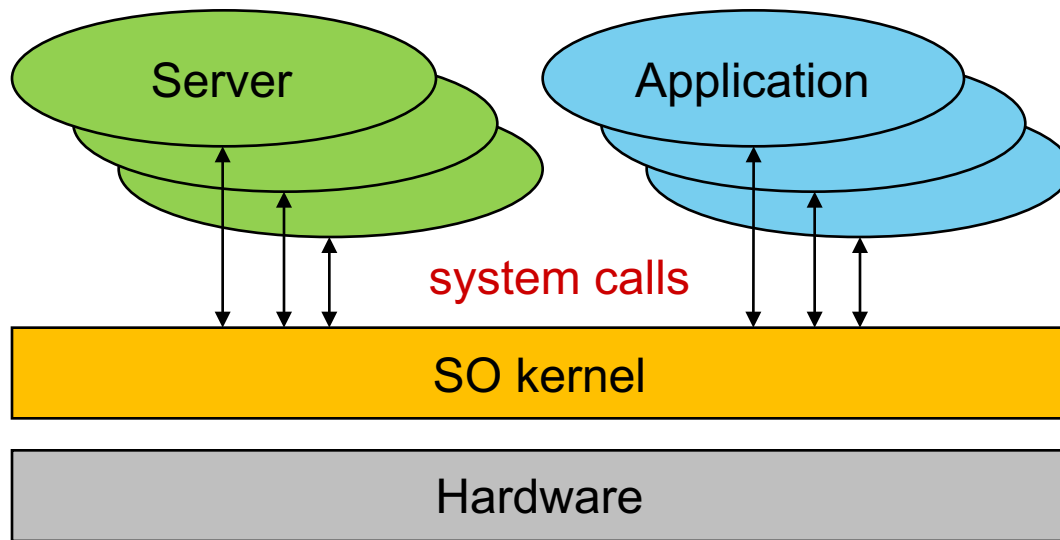


Security in Operating Systems

Operating system



user-mode:

Execute in normal CPU mode,
no access to privileged instructions

supervisor mode:

Execute in privileged CPU mode,
has access to privileged instructions

Operating system kernel mission

Initialize the hardware (booting)

Virtualize the hardware providing interface to applications

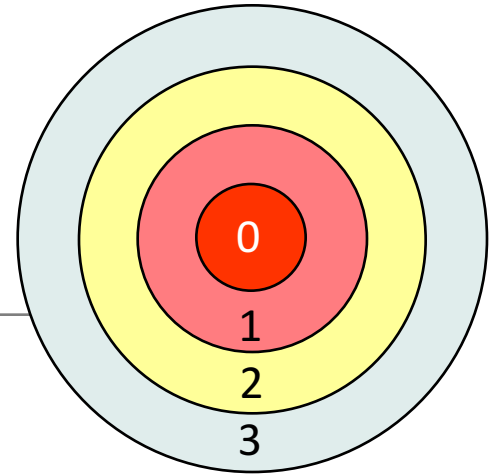
- Computational model

Enforce protection policies and provide protection mechanisms

- Against involuntary mistakes
- Against non-authorized activities

Provide a Virtual File System

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 mode) and 3 (user-mode)

Transfer of control between rings requires special gates

- The ones that are used by syscalls

Execution of virtual machines

Common approach

- Software-based virtualization
- Direct execution of guest user-mode code (ring 3)
- Binary translation of privileged code (ring 0)
 - Guest OS kernels remain unchanged, but do not run directly on the host machine
 - Drivers at ring 0 provide access to reserved opcodes

Hardware-assisted virtualization

- Full virtualization
- There is a ring -1 below ring 0
 - Intel VT-x and AMD-V. Reserved for the Hypervisor
- 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
 - Keyboard, graphical screen, text console, mouse
 - 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

- Establish during a login operation
- User ID (UID)

All activities are executed on behalf of a UID

- The UID allows it to establish what is allowed/denied to activities
- 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 an 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

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 it entities of the computational model

Subjects

- Usually local processes
 - Through the syscall API
 - A syscall is not an ordinary function call
- But also messages from other hosts

Mandatory access controls

An OS kernel is full of 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
- Resource reservation limits

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 overruled
- 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 file descriptors through the OS kernel
- The OS kernel has full control over file descriptors
- File descriptors can only be granted to other processes through the OS kernel
- Changes in the protection of files does not impact existing file descriptors

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

Entities:

- A 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
- The owner can be a UID or a GID
- The owner has no special rights over the ACL

Subjects:

- Users (UIDs)
- Groups (GIDs)
 - The group “Everyone” stands for anybody

Windows NTFS file protection: Variable-size, discretionary ACLs

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 counts for defining the rights of the process

UID change

- Ordinary application
 - eUID = rUID = UID of process the executed exec
 - eUID cannot be changed (unless = 0)
- Set-UID application
 - eUID = UID of application, rUID = initial process UID
 - eUID can revert to rUID
- rUID cannot change

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)

Is 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

Is used to protect the file system from potentially problematic applications

- e.g. public servers, downloaded applications
- But it is not bullet proof!

Login:

Not an OS kernel operation

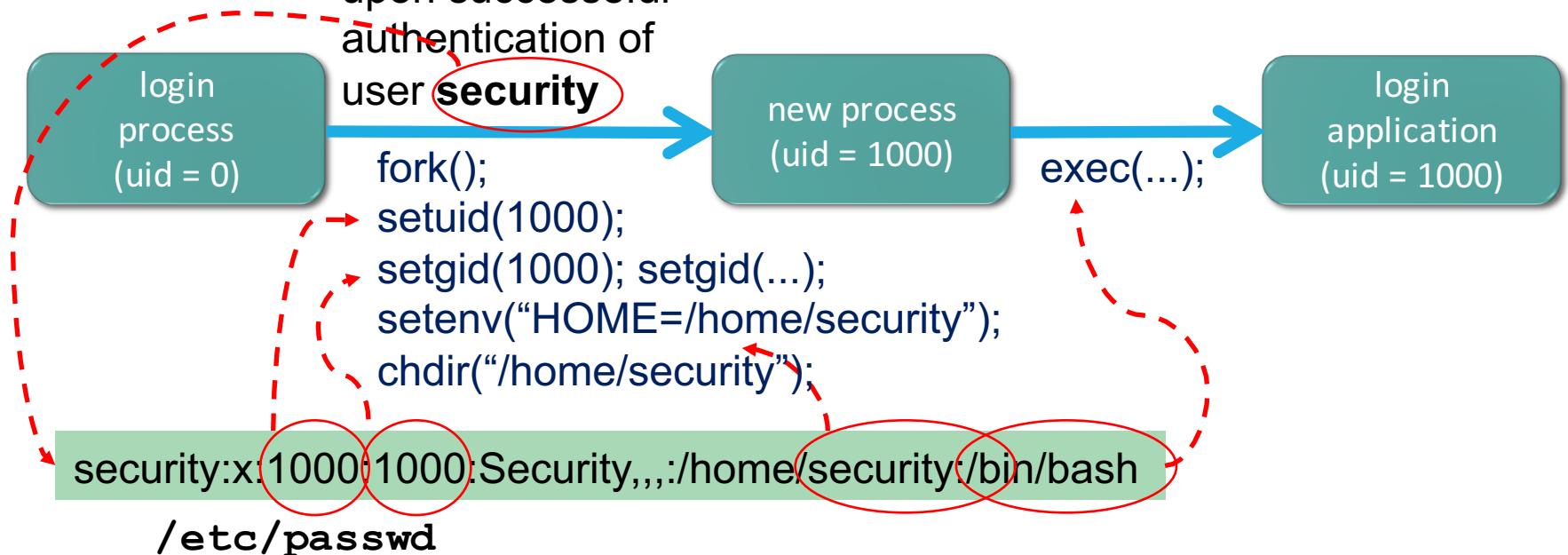
How it works

1. A privileged login application presents a login interface for getting users' credentials
 - A username/password pair
 - Biometric data
 - Smartcard and activation PIN
2. 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
3. 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 upon successful authentication of user **security**



Login in Linux: Password validation process

The username is used to fetch a UID and a GID from the `/etc/passwd` file

- And a set of additional GIDs in the `/etc/group` file

The supplied password is transformed using a unidirectional hash function

- Currently SHA512

The result is checked against a value stored in the file `/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
- `/etc/shadow` can only be read by root
 - Protection against dictionary attacks

Login in Linux: Password validation process

```
security:$6$4ijrA0yQ$qA.qP1MSOEGPJ0QZ276ITs0vuiJfNLPXg  
3m1uZmPZSIhiy74QJ5Hdsff0UBVSkRBqVM9q1s4kgCd5G71uqsON/  
16645:0:99999:7:::
```

- **Username:** security
- **Digest Mode:** 6 = SHA512
- **Salt:** 4ijrA0yQ
- **Hash:** 6\$4ijrA0yQ\$qA.qP1MSOEGPJ0QZ276ITs0vuiJfNLPXg3m1uZmPZSIhiy74QJ5Hdsff0UBVSkRBqVM9q1s4kgCd5G71uqsON/
- **Last password change:** 16645
- **Minimum days between password change:** 0
- **Maximum days between password change:** 99999
- **Warn days before maximum:** 7
- **Inactive days after expire:** undefined
- **Expire date:** undefined