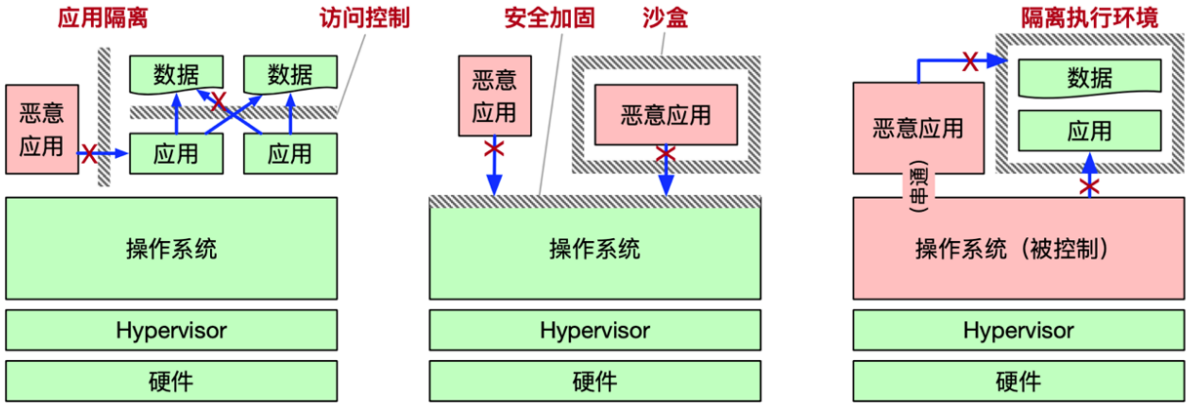
Lecture25 OS Security

1. 操作系统的安全服务

安全是操作系统的重要功能和服务

- **系统中有许多需要保护的数据**
  - 如账号密码、信用卡号、地理位置、照片视频等
  - 操作系统需要允许这些数据被合法访问，但不允许被非法访问
- **系统中可能存在许多恶意应用**
  - 操作系统需要与这些恶意应用作斗争，保护自己，限制对方
- **操作系统不可避免的存在漏洞**
  - 操作系统需要考虑自己被完全攻破的情况下依然提供一定的保护

操作系统安全的三个层次



(a) 基于操作系统的应用隔离与访问控制

(b) 操作系统对恶意应用的隔离与防御

(c) 操作系统不可信时对应用的保护

> (b)：像是防火墙
>
> (c)：像是黑匣子

层次一：基于OS的应用隔离与访问控制

- 威胁模型
  - OS是可信的，能够正常执行且不受攻击
  - 应用程序可能时恶意的，会窃取其他应用数据
  - 应用程序可能存在bug，导致访问其他应用数据

- 应用隔离
  - 内存数据隔离：依赖进程间不同虚拟地址空间的隔离
  - 文件系统隔离：文件系统是全局的，需要限制哪些应用不能访问哪些文件
    - 操作系统提供对文件系统的访问控制机制

层次二：OS对恶意应用的隔离与防御

- 威胁模型
  - OS存在bug和安全漏洞
  - OS的运行过程依然可信
  - 恶意应用利用OS漏洞攻击，获取更高权限或直接窃取其他应用的数据
- 操作系统防御
  - 防御常见的OS bug
  - 沙盒机制限制应用的运行

层次三：OS不可信时对应用的保护

- 威胁模型
  - OS不可信，有可能被攻击者完全控制
  - 恶意应用可能与操作系统传统发起攻击
- 基于更底层的应用保护
  - 基于Hypervisor的保护：可信基更小
  - 基于硬件Enclave的保护：硬件通常更可信
    - 可以不信任Hypervisor

操作系统安全的三个概念

- 可信计算基(Trusted Computing Base)
  - 为实现计算基系统安全保护的所有安全保护机制的集合
  - 包括软件、硬件和固件（硬件上的软件）
- 攻击面(Attacking Surface)
  - 一个组件被其他组件攻击的所有方法的集合
  - 可能来自上层、同层和底层
- 防御纵深(Defense in-depth)
  - 为系统设置多道防线，为防御增加荣誉，以进一步提高攻击难度

操作系统安全很难指标化

- 指标-1：千行代码的缺陷密度

  > 常用指标：每1000行代码的平均缺陷数量

- 指标-2：已发现的缺陷数量

安全目标（从数据角度）：CIA

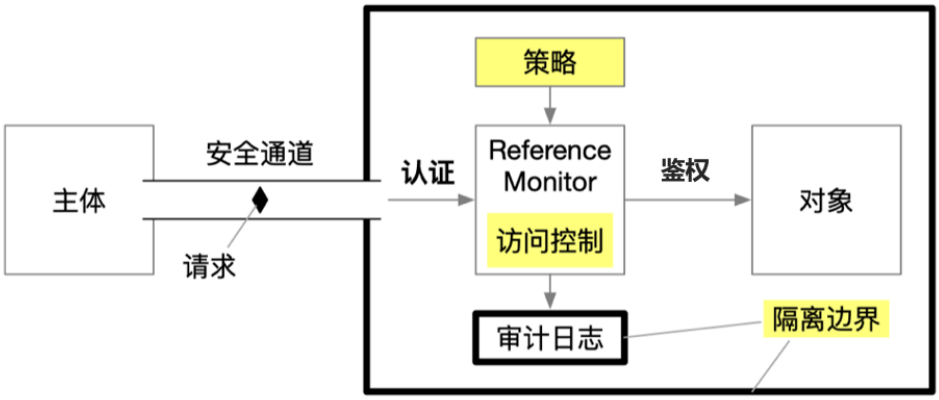
- 机密性(Confidentiality)
  - 常又称隐私性(Privacy)
  - 数据不能被未授权的主体窃取(即恶意读操作)
- 完整性(Integrity)
  - 数据不能被未授权的主体篡改(即恶意写操作)
- 可用性(Availability)
  - 数据能够被授权主体正常访问

2. 访问控制

访问控制与引用监视器

- **访问控制（Access Control）**
  - 按照访问实体的身份来限制其访问对象的一种机制
  - 为了实现对不同应用访问不同数据的权限控制
  - 包含"认证"和"授权"两个重要步骤
- **引用监视器（Reference Monitor）**
  - 是实现访问控制的一种方式
  - 主体必须通过引用（reference）的方式间接访问对象
  - Reference monitor 位于主体和对象之间，进行检查

引用监视器(Reference Monitor)机制



Reference Monitor 负责两件事：

1. **Authentication**：确定发起请求实体的**身份**，即**认证**
2. **Authorization**：确定实体确实拥有访问资源的**权限**，包含**授权**和**鉴权**

认证机制：你是谁

- 知道什么(Something you know)
- 有什么(Something you have)
- 是什么(Something you are)

认证：从用户到进程

- **进程与用户之间如何绑定?**
  - 每个进程的PCB中均包含了user字段
  - 每个进程都来自于父进程，继承了父进程的user
  - 用户在登录后运行的第一个进程（shell），初始化user字段
  - 在Windows下，窗口管理器会扮演类似shell的角色

访问控制矩阵

> 实体：类似于Linux中的用户
>
> 对象：类似于Linux中的文件

访问控制矩阵

	对象-1	对象-1	对象-3
实体-1	读/写	读/执行	读
实体-2		读/执行	读/写
实体-3	读		读/写

- **权限矩阵**
  - 对象与实体的关系
- **矩阵有多大?**
  - 假如系统中有 100 个用户，每种权限用 1 个 bit 来表示，那么每个文件都至少需要 300 个 bit 来表示 100 个用户 的 3 种权限
  - 假设这些 bit 都保存在 inode 中，通常 inode 的大小为 128-Byte 或 256-Byte，300 个 bit 相当于一个 inode 的 15% 至 30%
  - 每当新建一个用户的时候，都必须要更新所有 inode 中的权限 bit，不现实

> 我们需要对访问控制矩阵进行拆分
>
> - 横着的是：能力列表(Capability List)
> - 竖着的是：访问控制列表(Access Control List)

授权机制：POSIX的文件权限

- **将用户分为三类**
  - 文件拥有者、文件拥有组、其他用户组
  - 每个文件只需要用9个bit即可：3种权限（读-写-执行） x 3 类用户
- **何时检查用户权限?**
  - 每次打开文件时，进行鉴权和授权
    - open()包含可读/可写的参数，OS根据用户组进行检查（鉴权）
    - 引入fd，记录本次打开权限（授权），作为后续操作的参数
  - 每次操作文件时，根据fd信息进行检查（鉴权）

> - Windows使用不同于POSIX的ACL机制
>   - 以文件和目录为粒度
>   - 为多个用户和用户组设置不同的权限
> - 对比POSIX
>   - 只有3类用户/组

基于角色的访问控制（RBAC）

- **RBAC：将用户（人）与角色解耦的访问控制方法**
  - Role-Based Access Control
  - 提出了角色的概念，与权限直接相关
  - 用户通过拥有一个或多个角色，间接地拥有权限
  - "用户-角色"，以及"角色-权限"，一般都是多对多的关系
- **RBAC的优势**
  - 设定角色与权限之间的关系比设定用户与权限之间的关系更直观
  - 可一次性地更新所有拥有该角色用户的权限，提高了权限更新的效率
  - 角色与权限之间的关系比较稳定，而用户和角色之间的关系变化相对频繁
    - 设计者负责设定权限与角色的关系（机制）
    - 管理者只需要配置用户属于哪些角色（策略）

最小特权级原则：setuid机制

- **问题：passwd 命令如何工作?**
  - 用户有权限使用 passwd 命令修改自己的密码
  - 用户的密码保存在 /etc/shadow 中，用户无权访问
  - 本质上是以文件为单位的权限管理粒度过粗——怎么解决?
- **解决方法：运行 passwd 时使用 root 身份（RBAC的思想）**
  - 如何保证用户提权为root后只能运行passwd?
    - 在passwd的inode中增加一个SUID位，使得用户仅在执行该程序时才会被提权，执行完后恢复，从而将进程提权的时间降至最小
  - passwd程序本身的逻辑会保证某一个用户只能修改其自身的密码

```
-rwsr-xr-x 1 root 63736 Jul 27  2018 /usr/bin/passwd
```

> 但是setuid存在安全隐患，一旦passwd程序存在漏洞，如buffer-overflow导致的返回地址修改，则攻击者很容易以root身份通过ROP运行execv("/bin/sh")

权限控制的另一种思路：Capability

- **Capability表示一种能力**
  - 例如：读取/foo文件，写入/foo文件，等等
  - 有点像钥匙，能打开某一把锁的话就能进行某个操作
  - 每个进程可以拥有一组能力
- **Capability怎么实现？很多种方式**
  - 仅仅是一串bit，但必须保存在内核中，否则进程就可以任意伪造
    - 通常保存在进程的PCB中，在进程进行某个操作的时候内核检查
  - 可以把不同Capability的组合对应为ACL中的不同组
  - 因此使用Capability的控制粒度可以很细，而且不需要建立大量的组

fd与Capability的类似之处

- **文件描述符 fd 可以看做是 Capability 的一种实现**
  - 用户不能伪造 fd，必须通过内核打开文件（回顾 file_table/fd_table）
  - fd 只是一个指向保存在内核中数据结构的"指针"
  - 拥有 fd 就拥有了访问对应文件的权限
  - 一个文件可以对应不同 fd，相应的权限可以不同
- **fd 也可以在不同进程之间传递**
  - 父进程可以传递给子进程（回顾pipe）
  - 非父子进程之间可以通过 sendmsg 传递 fd

Linux的Capability

- **提供细粒度控制进程的权限**
  - 初衷：解决root用户权限过高的问题
- **需要注意，与前面说的Capability的不同**
  - 语义都是预先由内核定义，而不允许用户进程自定义
  - 不允许传递，而是在创建进程的时候，与该进程相绑定
  - 没有提供 Capability ID，无法通过 ID 索引内核资源进行操作

Capability 名称	具体描述
CAP_AUDIT_CONTROL	允许控制内核审计（启用和禁用审计，设置审计过滤规则，获取审计状态和过滤规则）
CAP_AUDIT_READ	允许读取审计日志（通过 multicast netlink socket）

- DAC与MAC
  - 自主访问控制(DAC: Discretionary Access Control)
    - 指一个对象的拥有者有权限决定该对象是否可以被其他人访问
    - 但是对部分场景（军队）来说，DAC过于灵活
  - 强制访问控制(MAC: Mandatory Access Control)
    - 由“系统”增加一些强制的、不可改变的规则
    - MAC与DAC可以结合，此时MAC的优先级更高