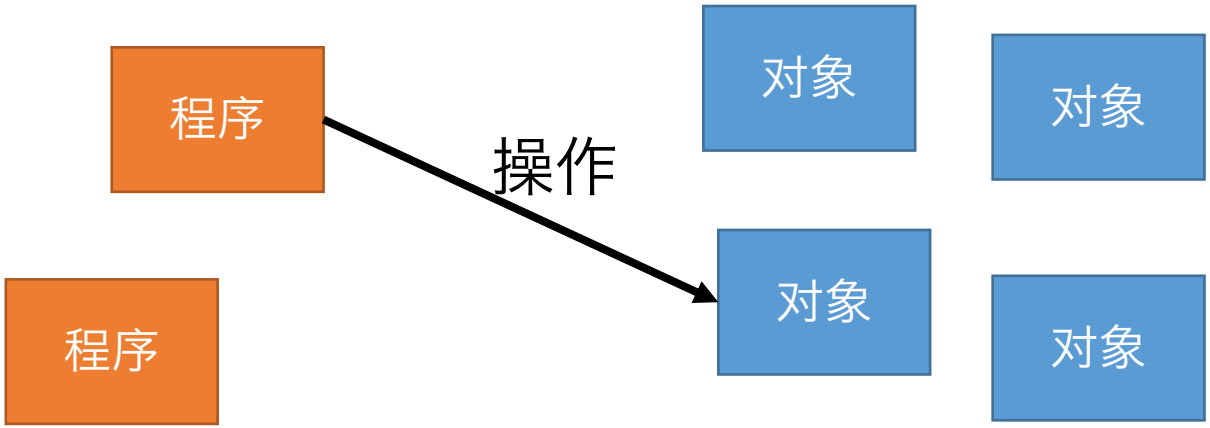
操作系统安全

蒋炎岩

南京大学 | 计算机软件研究所 | 系统与软件分析研究组

安全

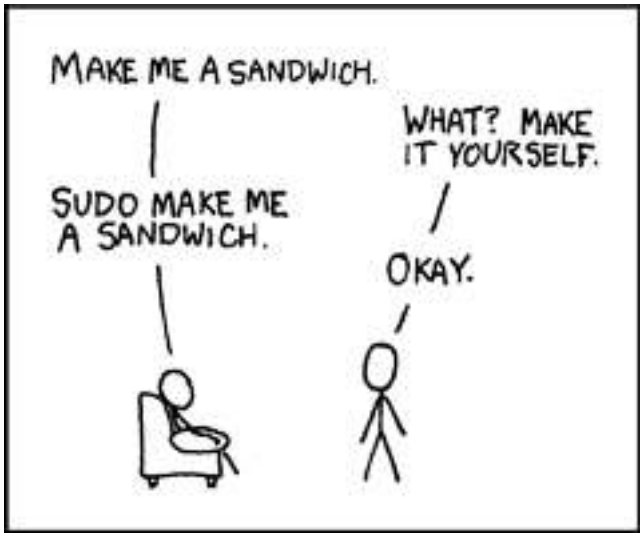
- Confidentiality, Availability, Integrity
  - 别人无法看到我不想让他看到的东西
  - 别人无法阻止我做我能做得事
  - 别人无法恶意破坏我拥有的东西



访问控制

访问控制

- 限制对操作系统对象的访问

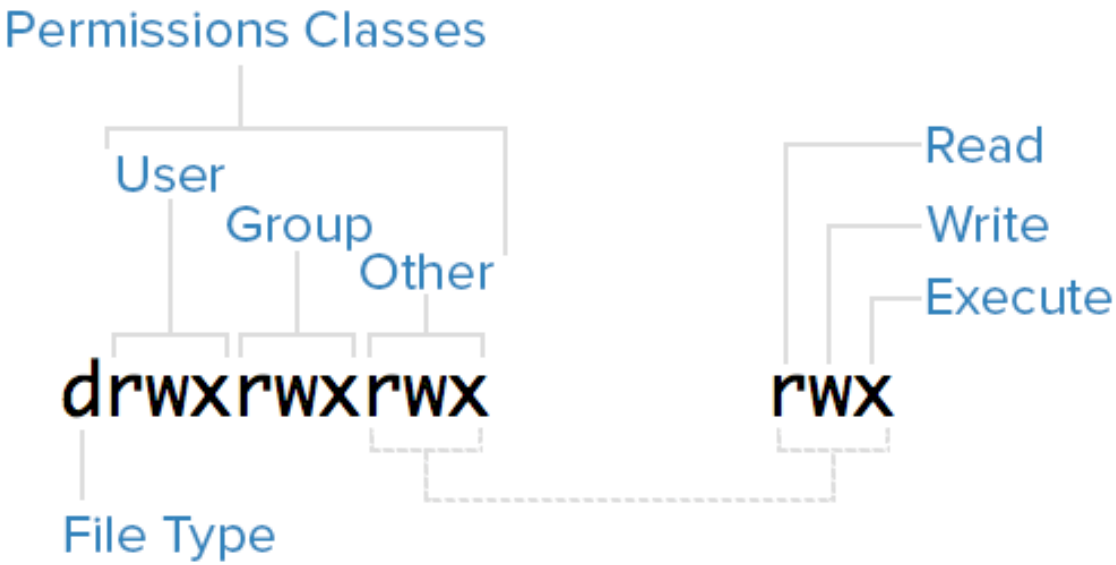


- 大家经常遇到permission denied
  - 不能执行一个普通文件
  - 不能修改一些文件(/etc/...)

- 原则：只给运行的程序最少的够用的权限
  - 万一应用程序出了问题/有漏洞，也不会危害系统
  - 在root下工作是个极坏的习惯

The UNIX Solution

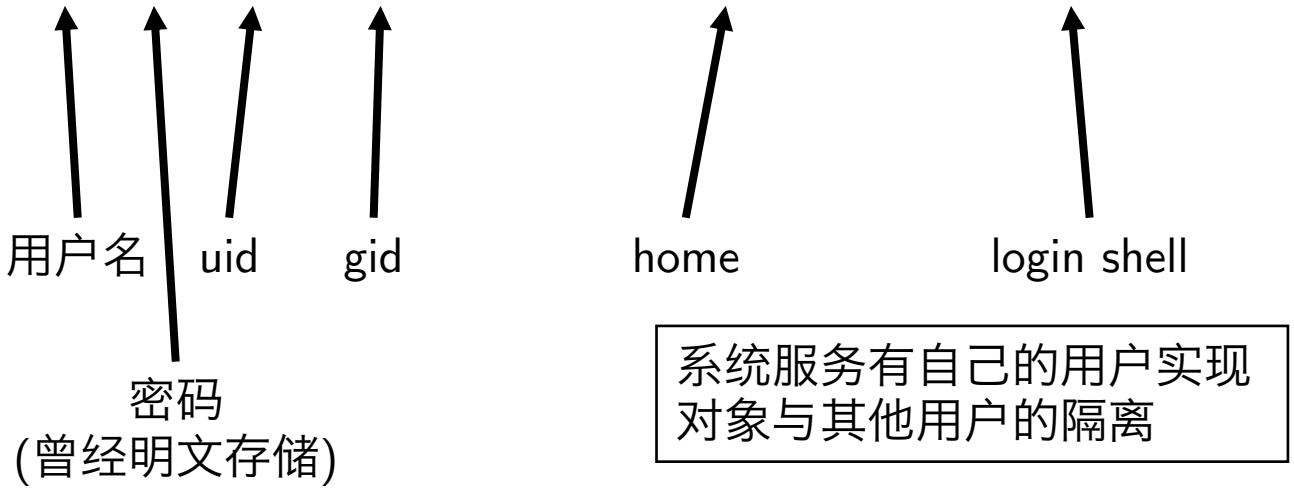
- Everything is a file
  - 操作系统中的对象都用文件来表示
  - 所以也以文件为对象进行访问控制



用户和组

- Everything is a file – 存储用户信息的数据库也是个文件
  - /etc/passwd每行存储一个用户信息

- 创建用户代价很低
  - `lxd:x:105:65534::/var/lib/lxd/:/bin/false`



UNIX访问控制：非常显著的问题

- 如何在多个用户/组之间共享文件(其他人无权访问)?
  - 再创建一个新的用户组

- 打补丁(POSIX ACL: Access Control List, man 5 acl)
  - 但这让操作系统的行为变得复杂
  - ACL是一种metadata，但长度不确定
    - 所有文件系统实现都要改了
    - 如果我忽然让你在OSLab2里实现……
    - 杀死一个程序员只要改三次需求就行了

实现访问控制

- 思考题：怎样在OSLab中实现访问控制？
  - (工程问题)在对象访问之前检查是否有权限即可

- 困难的问题：访问控制能保证系统安全吗？
  - 访问控制策略需要被正确设置
    - 服务器上的home被设置成rwxr-xr-x，然后被各种偷窥……
  - 除此之外
    - 操作系统实现正确
    - 进程执行不向其他进程泄露任何信息
    - 用户进程不能访问内核数据
    - ……

实现su

- su – switch user
  - 系统允许在运行时切换用户
  - 在适当地时候进行身份验证
    - 输入密码、扫描指纹……
    - int su(int uid, auth_t *auth);不是一个好的解决办法：为什么?
- Linux内核使用一个专门的模块管理身份认证
  - PAM (Pluggable Authentication Modules)
  - /etc/pam.d查看PAM的设置

不完美的世界 (1)

不完美的世界(1)：规约 ≠ 实现

- 访问控制只是一个“期望”
  - 或者说像是软件的需求
  - 在软件里，希望用户/请求不能访问无权访问的对象

- 提出需求 ≠ 实现正确
  - CPU、操作系统、库函数、应用……实现的都正确才行
  - 一个bug (比如su命令)就可能导致致命的后果
    - sudo真的有bug啊 (CVE-2017-1000367)

httpd引发的血案

- MiniLab 7：HTTP发送目录中的文件
  - 原则上，只能访问目录中的对象
  - 但如果你的实现直接把GET的路径拼接……

```
$ ./httpd -p 8000 /var/www
> Listening 0.0.0.0:8000
> GET /
> 200 OK -> /var/www/index.html
> GET ../../etc/passwd
> 200 OK -> /etc/passwd
```

各种各样针对服务的注入

Sanity Check的重要性

- 每个同学有缺课次数
  - 绕过了校验改成了-99999999
  - 所有同学就从选课列表里消失了
    - 后来联系了教务系统的内部人士才解决问题

课程列表

课程号	课程名	班级名	授课对象	上课时间地点	学生列表	打印预览	导出名单	教学周历	网络课程	考试时间	考试地点	临时停调课
22020230	操作系统	操作系统(20172-22020230-1)	2016 匡亚明-计算机班	周一 第3-4节 仙I-103 1-18周 周四 第5-6节 仙I-103 1-18周	查看	预览	导出(学号占双行) 导出(学号占单行)	编辑		请咨询院系或上课老师		申请

学生列表

学号	姓名	所属专业	缺课次数

提交

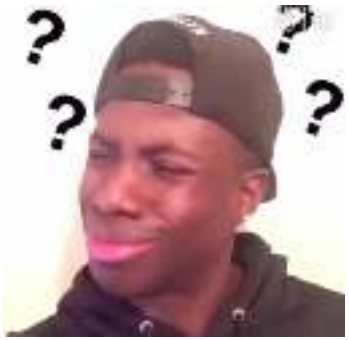
另一个例子

- (细节我记不清了)

- 宿舍楼下好像有个能自助打印的机器
  - 不知道做了什么，能够绕过前端的检查
  - 于是可以打印-1页
  - 然后校园卡里就多出钱了！
    - 猜测最后结算的时候校园卡中心会把总和转账给公司

库函数：更危险

- libpng：显示png图片的开源库
  - 比如在浏览器里显示图片就用libpng
  - 坏人
    - 架设恶意网站，诱导你点击
    - 你：我反正就看看，不输密码(信任浏览器是安全的)
    - 浏览器里存储的密码被偷走了

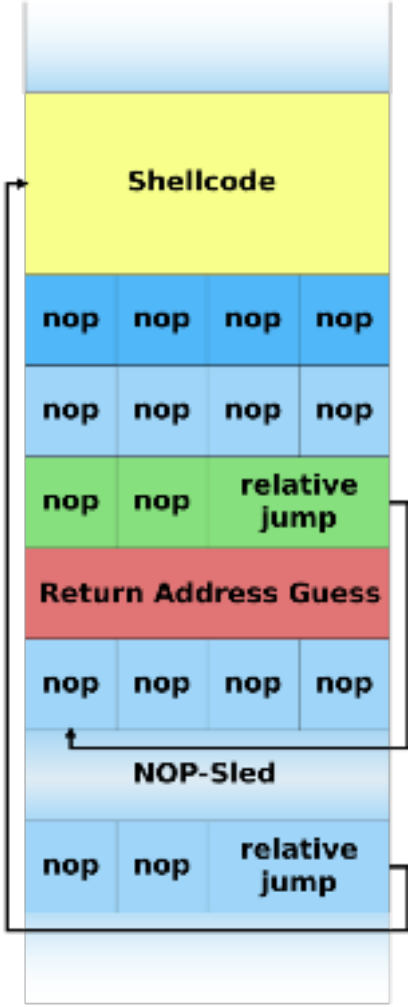


Heap-based buffer overflow in the `png_combine_row` function in libpng before 1.5.21 and 1.6.x before 1.6.16, when running on 64-bit systems, might allow context-dependent attackers to execute arbitrary code via a "very wide interlaced" PNG image.

(CVE-2014-9495)
(CVE = Common Vulnerabilities & Exposures)

人畜无害的strcpy

- 在栈上定义char buf[]然后strcpy非常危险
  - 返回地址是保存在堆栈上的!
  - 如果被覆盖，就可能造成安全漏洞
    - 只要返回地址被nop覆盖
    - 就能保证执行shellcode
- 这实际上触发了undefined behavior (UB)
  - 允许任何事发生(崩溃、机器爆炸、……)
  - UB可以说是C语言的一个巨坑
    - 导致安全漏洞、编译器难测试……



Linux Kernel也是软件

- 你们会犯的错误，其他开发者也会犯
  - 整数溢出 (CVE-2016-3135)
  - 数据结构忘记初始化 (CVE-2016-4470)
  - use-after-free (CVE-2016-4794)
  - ……
- 于是经常会打补丁……

小结：码农的辛苦

- 有一天也许你们会进入互联网公司，维护真正的代码，你的boss希望你的项目
  - 功能要做对
    - 代码总得对吧，要经得起测试(OSLab谁做谁知道)
  - 非功能要做好
    - 性能要好、交互要友好、……
  - 安全要保障
    - 永远不知道你背后的人怎么拿你没想过的输入捅你刀子
- 为什么说码农是个青春饭?
  - 做错了一些微小的地方就会损失分数、金钱、声誉、饭碗、……

不完美的世界 (2)

不完美的世界(2)：你的假设未必成立

- 操作系统安全的两个假设
  - 应用程序无法观测操作系统内核执行、代码/数据
  - 应用程序无法观测其他程序的执行、代码/数据
  - (其实我自己也不太知道到底做了多少假设)

- 看起来很合理
  - ring0-ring3保护不就是做这个事的么？
  - 进程本来就访问不了另一个进程的地址空间啊？
  - 且慢……

一个著名的案例

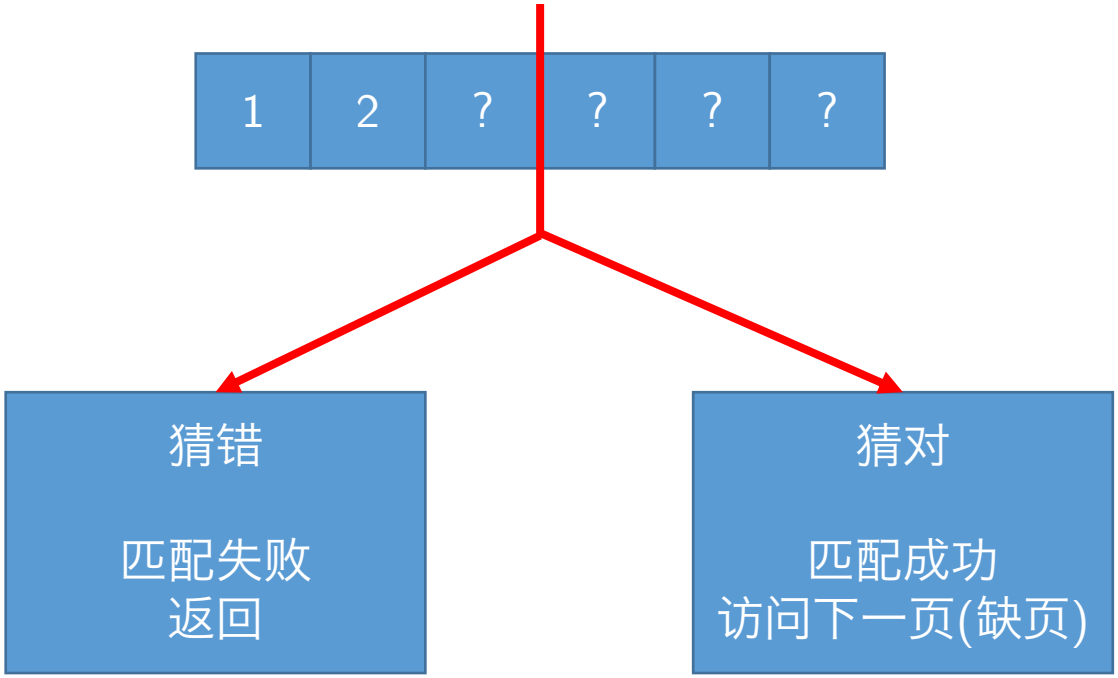
- 看起来很对的密码校验代码

```
int kernel_auth(const char *user_buf) {
  const char *passwd = get_password();
  int len = strlen(passwd);
  for (int i = 0; i < len; i++) {
    if (user_buf[i] != passwd[i]) {
      return -1;
    }
  }
  return 0;
}
```

- 如果能观测内核代码的执行，那就坏事了
  - 用performance counter来读取分支计数器……然后……

Side Channel：时间

- 假设已经知道密码以“12”开头
  - 把密码的下一位(user_buf)卡在页边界上，并让下一页缺页
  - 猜测密码的下一位是什么，根据返回的时间就能知道是否猜对
    - $O(L \cdot |\Sigma|)$次就能猜出密码；思考题：如何修复漏洞？



漏洞：分析

- 我们的一些基本假设遭到了动摇
  - 用户进程无法读/写操作系统内存
    - 看起来是存储保护的规约实现的(读/写导致#PF)
    - 但CPU可能有bug啊 ← Intel最近就这么火了
  - 用户进程不能profile操作系统内核执行
    - 只要能偷出1bit信息(有/无缺页)，就非常危险
- 构造安全(side-channel-free)的系统还是个open problem

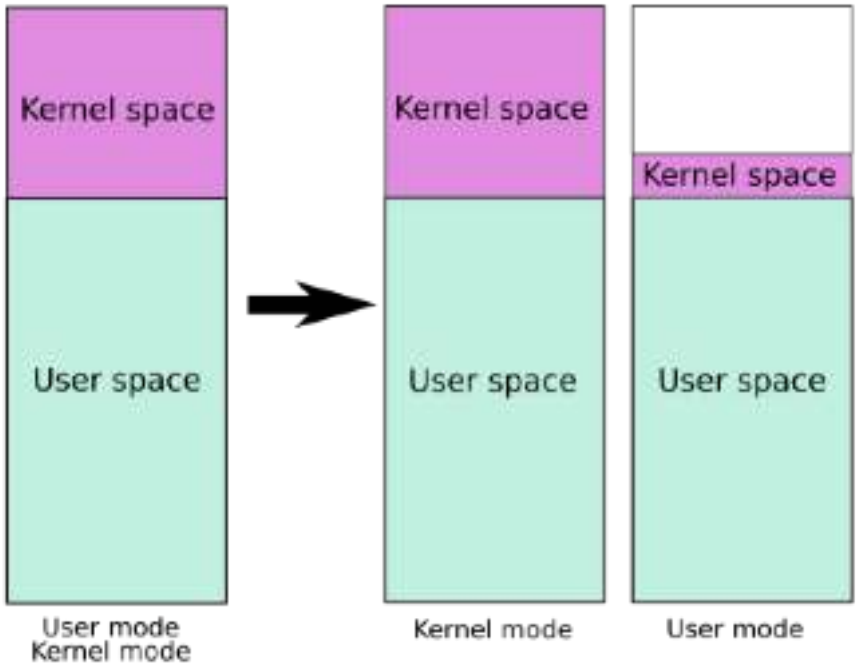
Spectre

- 假设char *ptr指向内核地址
  - *ptr是非法的(导致#PF)
- 但是如果我们弄一个数组arr，一开始全都不在缓存里

- 然后执行arr[L * (*ptr)] = 0;
  - *ptr先执行，照道理应该会#PF
  - 但因为CPU是流水线&乱序执行的，会投机执行之后的赋值
    - arr[L * (*ptr)]的缓存给fetch之后才#PF
  - 扫描一遍arr，看访问的时间就偷出了*ptr的值(读取内核数据)

操作系统：修复Spectre

- KPTI (Kernel Page Table Isolation)
  - 绕了一大圈，做了一堆优化，最后又回到原点了

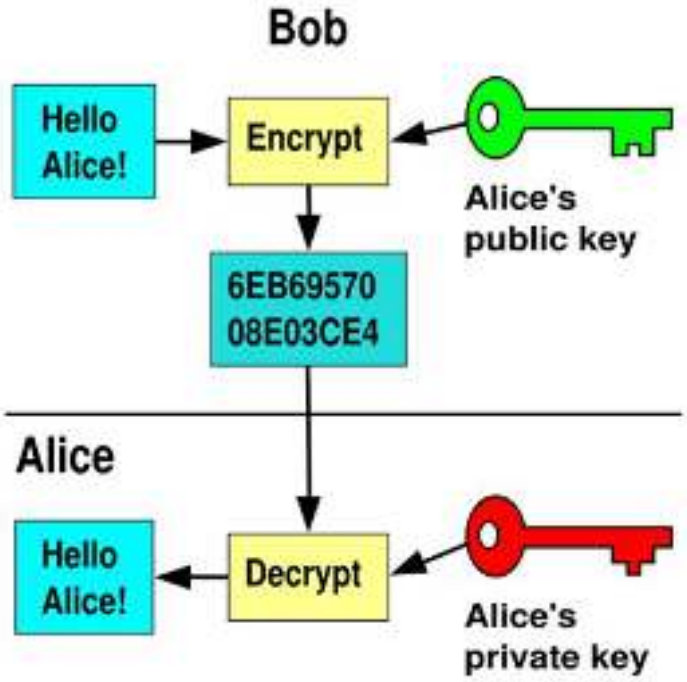


  - 虚拟化环境下可以做得更好[1]

[1] Z. Hua, D. Du, Y. Xia, H. Chen, and B. Zang. EPTI: Efficient Defense Against Meltdown Attack for Unpatched VMs. In *Proc. of USENIX ATC*, 2018.

那些脑洞大开的Side Channels

- 一台计算机反复运行解密算法
  - 精心构造密文
  - 读取CPU的负载
  - 逐个bit偷出private key



安全：永恒的话题

- 没有人能保证自己的软件没有bug
  - 苹果、微软、Google每天都在被发现漏洞
  - 开源世界已经是dependency hell了
    - 中等规模的程序就会依赖上亿行代码