Lecture26 OS Vulnerability

1. 操作系统内核漏洞

漏洞分类的三个角度

- 漏洞类型（指攻击所利用的漏洞类型）

  > 栈/堆缓冲区溢出、整形溢出、空指针/指针计算错误、内存暴露、use-after-free、double-free、未初始化读取、格式化字符串错误、竞争条件错误、参数检查错误、认证检查错误，等等

- 攻击模块（指攻击所利用漏洞所在的模块）

  > 调度模块、内存管理模块、通信模块、文件系统、设备驱动等

- 攻击效果（指攻击的目的或攻击导致的结果）

  > 执行任意代码、内存篡改、窃取数据、拒绝服务、破坏硬件等

操作系统的漏洞

- **操作系统本身也是软件**
  - 同样存在各种漏洞，如缓冲区溢出、未初始化指针、竞争等
- **操作系统与一般应用软件不同**
  - 需要对硬件直接操作，与高级语言的抽象往往不匹配
    - 高级语言的内存安全等特性往往无法使用
  - 硬件语义的加入，为正确性判断带来了更多挑战
    - 例如对栈的操作，会使编译器失去上下文
  - 以数据表示权限等，使数据类攻击具有更强的能力
    - 例如页表的权限位，userid=0

思考问题

- 内核中哪些数据结构非常危险?

  > 所有记录权限、敏感数据的地方：页表、PCB、uid、gid、系统调用表、VFS函数表、IDT、键盘驱动、skb等等

2. 操作系统内核攻防

整型溢出漏洞

```
unsigned long count = /* from user space */;
if (count > 1<<30)
    return -EINVAL;
table = vmalloc(sizeof(struct
 ↪  rps_dev_flow_table) +
                  count * sizeof(struct
                   ↪  rps_dev_flow));
...
for (i = 0;i < count; i++)
    table->flow[i] = ...;
```

防御方法：增加对溢出的检查代码；利用自动化工具查找并修复

Return-to-user攻击(ret2usr)

- **内核错误地运行了用户态的代码**
  - 由于内核与应用程序共享同一个页表，内核运行时可以任意访问用户态的虚拟地址空间，内核可能执行位于用户态的代码
- **攻击者的常用方法**
  - 先在用户态中初始加载一段恶意代码，然后利用内核的某个漏洞，修改内核中的某个函数指针指向这段恶意代码的地址
  - 也可以利用内核的栈溢出漏洞，覆盖栈上的返回地址为恶意代码的地址，使内核在执行 ret 指令时跳转到位于用户态的代码

ret2usr攻击的防御方法

- **方法一：仔细检查内核中的每个函数指针**
  - 需对内核所有模块进行检查，很难做到 100% 的覆盖率
- **方法二：在陷入内核时修改页表，将用户态所有的内存都标记为不可执行**
  - 由于修改页表后必须要刷新 TLB 才能生效，因 此修改页表、刷新 TLB，以及后续运行触发 TLB miss 都会导致性能下降
  - 在返回用户态之前必须将页表恢复，并再次刷掉 TLB，这样又会导致用户态执行时出现 TLB miss，因此对性能的影响非常大
- **方法三：硬件保证CPU处于内核态时不得运行任何用户态的代码**
  - 如 Intel 的 SMEP（Supervisor Mode Execution Prevention）技术
  - ARM 同样有类似 SMEP 的技术，称为 PXN（Privileged eXecute-Never）

> 但是可以通过直接映射的方式绕过SMEP
>
> 在3.8.13以及之前的Linux版本，将直接映射区域的权限设置为了“可读-可写-可执行”
>
> 这种利用直接映射区域的ret2usr攻击被称为“ret2dir"攻击

Rootkit：获取内核权限的恶意代码

- **Rootkit 是指以得到 root 权限为目的的恶意软件**
  - Rootkit 可以运行在用户态，也可以运行在内核态
- **用户态的Rootkit**
  - 可以将自己注入到某个具有 root 权限的进程中，并接收攻击者的命令
- **内核态的Rootkit**
  - 可以 hook 某个内核中的关键函数，从而在该函数被调用时触发运行
  - 可以是以内核线程的方式运行
  - 可以是修改内核中的系统调用表，用恶意代码来替换掉正常的系统调用

KASLR：内核地址布局随机化

- ASLR和KASLR
  - ASLR通过随机化地址空间布局来提高系统攻击难度
  - KASLR是对内核启用地址随机化
- KASLR可缓解ret2dir攻击
  - 攻击者需要知道用户态恶意代码在内核中直接映射区域的地址
  - KASLR通过将内核的虚拟地址布局进行随机化，使攻击者准确定位内核地址的难度大大提升

内核漏洞防御机制（一部分）

- **运行时工具**
  - SFI（Software Fault Isolation）：对内存做访问控制
  - 代码完整性保护：阻止非法代码运行
  - 用户态驱动：降低内核攻击面
  - 未初始化内存跟踪：防止未初始化内存被使用或复制到用户态
- **编译时工具**
  - 代码静态分析工具，通常需要开发者添加annotation