

课程目录

- ① [攻击面概述](#)
- ② [内核攻击面分类与枚举](#)
- ③ [内核漏洞原理](#)
- ④ [Linux 内核漏洞点深度解析](#)





- ① [攻击面概述](#)
- ② [内核攻击面分类与枚举](#)
- ③ [内核漏洞原理](#)
- ④ [Linux 内核漏洞点深度解析](#)

什么是攻击面

- 攻击面：攻击者可以与系统产生交互，并影响系统状态的位置
- 包含所有输入路径、解析逻辑、状态变化点
- 覆盖数据输入、控制流输入、元数据输入（如权限/结构字段）
- 不同攻击面影响程度不同：可控性 + 可达性 + 持久性

核心定义

攻击面 = 可控输入 + 可达代码 + 可改变状态



攻击面三要素

- **入口 (Entry)**: 攻击者能触发的接口, 如 `syscall`、`ioctl`、`net packet`
- **解析 (Parsing)**: 根据输入进行处理、校验、路径选择
- **状态改变 (State Change)**: 内核资源发生变化, 如内存、对象、权限

系统调用示例

- 用户传入指针、长度、结构体 (入口)
- `copy_from_user` 读取用户数据 (解析)
- 修改内核对象 `refcount / list / slab` (状态改变)

攻击面三要素

- **入口 (Entry)**: 攻击者能触发的接口, 如 `syscall`、`ioctl`、`net packet`
- **解析 (Parsing)**: 根据输入进行处理、校验、路径选择
- **状态改变 (State Change)**: 内核资源发生变化, 如内存、对象、权限

系统调用示例

- 用户传入指针、长度、结构体 (入口)
- `copy_from_user` 读取用户数据 (解析)
- 修改内核对象 `refcount / list / slab` (状态改变)

关键点

只有“能改变状态”的入口才构成真正攻击面

为什么要研究攻击面

- 漏洞利用链条从攻击面开始：没有入口 -> 无法触发漏洞
- 现代内核复杂度高 -> 总存在弱点（代码量、驱动质量）
- 安全分析第一步：确定入口、可控范围、影响范围
- 内核攻击面决定漏洞能否从用户态触发

安全目标

减少攻击面 & 降低攻击面可利用性

真实案例提示

CVE-2016-5195 Dirty COW:

攻击面 = 入口：write() + 解析：copy_on_write + 状态：文件内容

Linux 内核攻击面分类（宏观）

- 系统调用接口（`syscall`）
- 文件系统接口（`VFS / xattr / parser`）
- 网络协议栈（`socket / netfilter`）
- IO 设备驱动（`USB / GPU / WiFi`）
- 虚拟化与容器（`KVM / namespaces`）
- `eBPF / perf / tracing`
- 内核模块机制（`LKM`）
- 内存管理接口（`mmap / page fault / COW`）

特点

每一项都是外界影响内核逻辑的入口

观察

哪些攻击面最常出现 Oday?

攻击面示例：系统调用

- 用户可直接传入指针、长度、结构体
- 内核访问用户内存（`copy_from_user`）
- 内核依据用户输入操作资源
- 调度与抢占导致竞态

典型风险

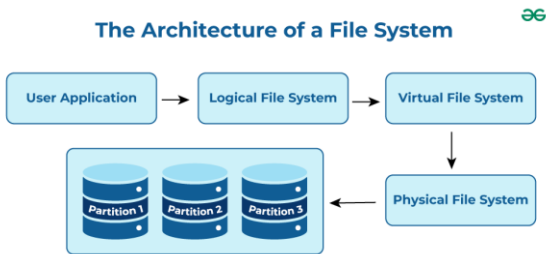
- 参数校验不足 -> OOB/UAF
- 竞态窗口 -> Dirty COW 类漏洞
- 引用计数错误 -> refcount overflow
- 类型混淆 -> struct layout

攻击面示例：文件系统

- 路径解析（.. / 符号链接）
- 权限检查（cred / capability）
- xattr 解析
- 文件格式解析（ext4 / btrfs metadata）
- 用户控 metadata -> 内核解析

经典问题

文件格式解析 = 最危险的解析逻辑之一（parser complexity）



攻击面示例：驱动程序

- ioctl 接口（结构体由用户构造）
- 内存映射（mmap）
- DMA / 物理内存直接访问
- 专有协议解析（厂商实现）

特点

驱动数量多、质量差 -> 内核漏洞主要来源

真实情况

70% 以上 Linux 内核 CVE 来自驱动

攻击面示例：eBPF

- 用户可上传字节码
- JIT 编译为内核指令
- 直接访问内核数据
- verifier 负责安全检查

风险

验证器缺陷 = 任意内核读写

案例提示

2022 年 eBPF JIT 漏洞链：LPE without files or syscalls

攻击面扩大的原因

- 内核不断新增功能
- 性能优先导致安全检查减少
- 可编程接口开放（eBPF、ioctl）
- 驱动生态庞大且缺乏审计
- 云环境需求增加内核可扩展性

趋势

攻击面持续增长

思考

能否减少攻击面，而不牺牲功能？

总结

- 攻击面是漏洞利用的起点
- 理解攻击面 = 理解漏洞产生机制
- 攻击面分析是漏洞挖掘核心能力
- 后续内容将围绕典型攻击面展开

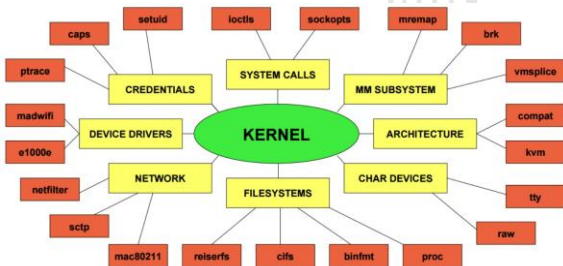




- ① [攻击面概述](#)
- ② [内核攻击面分类与枚举](#)
- ③ [内核漏洞原理](#)
- ④ [Linux 内核漏洞点深度解析](#)

内核攻击面概览

- 内核攻击面：用户态可控输入 -> 内核解析/执行
- 攻击者目标：
 - 进入内核态
 - 提权 / 任意读写 / 持久化
- 枚举角度：
 - 数据流入口
 - 代码执行入口
 - 信任边界



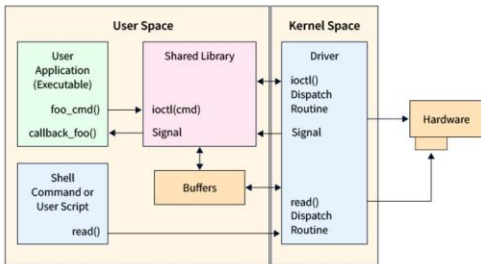
系统调用 (Syscall)

- 用户态 -> 内核态的主要入口
- 攻击向量：
 - 参数验证不足
 - 指针处理错误
 - 内存漏洞
- 可利用接口示例：
 - read/write/ioctl
 - mmap
 - clone/fork



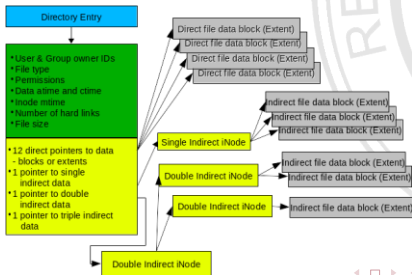
ioctl 攻击面

- 常见漏洞来源：
 - 未检查用户指针
 - 未验证结构体大小
 - `copy_from_user / copy_to_user` 错误
 - 类型混淆
- 驱动普遍暴露 `ioctl`
 - GPU / WiFi / Camera / File system
 - OEM 定制模块

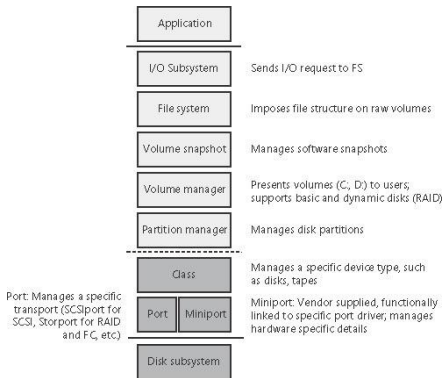


文件系统攻击面

- 文件解析器常见漏洞：
 - 路径遍历
 - 符号链接绕过
 - 元数据解析错误
 - Journal 重放
- 攻击来源：
 - FUSE 用户态文件系统
 - Mount 外部介质（U 盘）



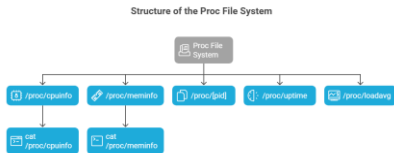
驱动程序 (Device Drivers)



- Linux 内核最大攻击面
- 特点:
 - 厂商自带
 - 质量参差
 - 安全审计薄弱
- 常见漏洞:
 - 任意内存读写
 - 不安全 `ioctl`
 - 锁同步缺失 (竞态)

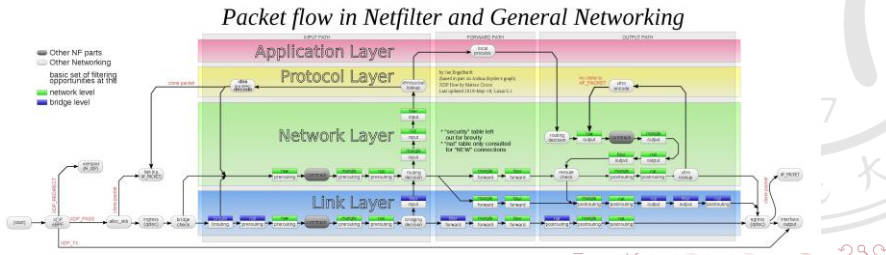
/proc 与 /sys 攻击面

- 特性：基于虚拟文件的“结构化 API”，动态生成内容（非持久化）
- 信息泄露：/proc/<pid>/maps、/proc/kallsyms 泄露地址/KASLR；调试跟踪接口泄露指针
- 权限与越界：可写 **sysfs attribute** 触发参数覆盖；未校验长度/格式导致整数溢出、越界写
- 竞态 & TOCTOU：用户快速修改 **sysfs** 与模块生命周期竞态（热插拔、CPU online/offline）
- 典型危险节点：/sys/kernel/debug/tracing、设备驱动自定义 **attributes**、/proc/driver/*



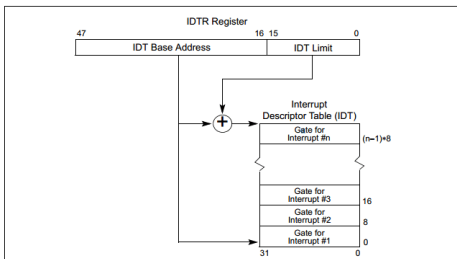
网络协议栈攻击面

- 数据可远程进入内核：数据包/帧经 NIC、socket、隧道直接驱动解析
- 漏洞类型：解析覆盖/ 状态机偏差/ 长度与聚合处理错误
 - 协议解析错误（边界检查不足、扩展/ 选项字段处理缺陷）
 - offload 处理缺陷（eBPF / XDP）JIT 类型混淆 / verifier 漏洞）
 - Netfilter / iptables（匹配扩展溢出、模块引用计数失衡）
 - WiFi 驱动（管理帧伪造、固件接口越界）



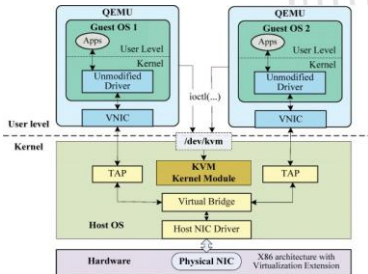
中断与调度攻击面

- 中断处理程序：硬中断入口，顶半部/底半部分离
- 软中断 / tasklet / workqueue：延迟执行与优先级队列，易被滥用制造负载
- 攻击点：并发可见性差、引用计数/生命周期复杂
 - 异步回调竞态：共享状态未同步引发数据竞争
 - use-after-free：回调触发于对象释放后
 - timer 回调错误：超时处理重入/越界



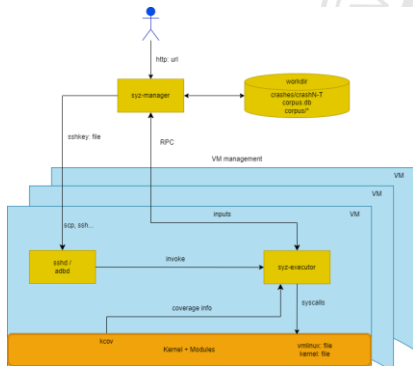
虚拟化攻击面

- hypervisor -> guest -> host: 多层隔离链条; 任一层逃逸可提升特权
- QEMU/KVM 攻击面: 设备模拟/IO 协议解析/共享内存 (vhost) 与迁移通道
- 常见漏洞:
 - virtio 解析错误: 描述符环边界/长度校验缺失
 - PCI passthrough: IOMMU 配置不当导致 DMA 越界
 - VM escape: 设备模型溢出/类型混淆实现跨边界执行



攻击面枚举方法

- 从用户输入路径分析
- 列举内核可调用接口
- 自动化工具：
 - syzkaller
 - kAFL
 - CodeQL



- ① [攻击面概述](#)
- ② [内核攻击面分类与枚举](#)
- ③ [内核漏洞原理](#)
- ④ [Linux 内核漏洞点深度解析](#)



本章内容结构

- **内存安全漏洞**：介绍 C 语言开发内核时常见的越界访问、UAF、未初始化内存等问题，解释它们在真实内核中的成因与破坏性。
- **同步与竞态漏洞**：多核环境下不同执行流对共享资源的无序访问导致状态错乱，是现代内核漏洞的重要来源。
- **权限与访问控制漏洞**：内核能力（**Capability**）与权限检查错误会导致任意操作接口暴露。
- **解析器 / 协议漏洞**：文件系统、网络协议、BPF 等复杂解析流程引入的边界问题与状态机缺陷。
- **接口滥用漏洞**：内核 API 被错误使用：`copy_from_user`、`refcount`、`ioctl`、`mmap` 等。
- **防御绕过与利用技巧**：从漏洞到提权的路径，介绍核心利用思路。

内存安全漏洞概述

- Linux 内核由 C 编写，缺乏自动越界检测、引用检查、生命周期管理。
- 内存安全漏洞规模庞大，是历史上最常见的内核漏洞来源。
- 典型漏洞类型包括：
 - 越界读写 (**Out-of-Bounds**)
 - **Use-After-Free** (**UAF**)
 - 未初始化内存泄露
 - **Double Free**
 - 空指针解引用
- 特点：一旦触发，通常可导致任意地址写、任意内存读、提权或内核崩溃。

越界访问 (Out-of-Bounds)

- 常由用户输入未校验长度导致，尤其出现在：
 - `copy_from_user / copy_to_user` 长度由用户控制
 - 协议/文件格式解析中访问数组
 - 对网络数据包 (`skb`) 的处理
 - `ring buffer / IO queue` 处理
- 效果：
 - 覆盖邻接对象的函数指针、`ops` 表、引用计数
 - 污染 `task_struct / cred` 结构
 - 内核直接崩溃（通常表现为 `BUG` 或 `Panic`）
- 为什么危险？越界“写”通常比越界“读”更致命，因为可以直接改变控制流。

Use-After-Free (UAF)

- UAF 本质：对象生命周期管理出错，释放后引用仍然可达。
- 内核中常见触发方式：
 - 文件描述符 race（多线程关闭同一 fd）
 - socket bind/connect 错误路径导致双重释放
 - RCU + 非 RCU 数据混用
 - timer/workqueue 在对象销毁后仍执行回调
- 危害：
 - 被攻击者重新分配（spray）并伪造结构体 -> 改写 vtable、ops、回调函数 -> 直接执行攻击代码
 - 内核提权的常见路径：伪造 cred 结构体

未初始化内存 (Uninitialized Memory)

- 未初始化内存暴露的是之前的内核内容，本质属于“信息泄露漏洞”。
- 常见触发点：
 - `kmalloc` 分配后未 `memset`
 - 栈变量未初始化直接 `copy_to_user`
 - 内核对用户空间结构体进行部分填充
- 危害：
 - 泄露内核指针 -> KASLR 失效
 - 泄露敏感数据（内存中的凭证、对象布局）
 - 有助于构造进一步攻击，如 `UAF` 或堆喷的精确利用

竞态条件（Race Condition）概述

- 多 CPU/多线程环境下对共享资源访问顺序不可预测。
- 如果开发者认为操作是“原子的”，但实际上分成多步，则可能被打断并被插入恶意逻辑。
- 特别严重的漏洞：TOCTOU（检查—使用之间的时间窗）
- 典型触发点：
 - 引用计数的竞争
 - 多线程同时执行 `ioctl`
 - 调用顺序依赖但缺乏锁保护
 - 异步回调（`timer/workqueue`）与同步流程交错
- 特点：同步漏洞隐蔽性极强，往往在高并发压力下才出现。

典型示例：检查与使用分离（TOCTOU）

- 示例逻辑：
 - ① 线程 A：检查资源可用
 - ② 线程 B：在 A 使用资源前销毁/修改
 - ③ 线程 A：继续使用已经不可用的资源 -> UAF / 崩溃
- 常见于：
 - 文件权限检查
 - 引用计数判断
 - 地址边界检查
 - 内核对象状态检查
- 为什么难发现？标准测试几乎无法触发，需要高频并发，安全研究者常通过 **fork** 炸机方式找到。

锁使用错误

- 开发者常见误区：
 - 忘记加锁（看似“应该是安全的”）
 - 使用了错误类型的锁（如需要 `spinlock` 却使用 `mutex`）
 - RCU 区域外访问 RCU 保护的對象
 - 锁粒度过小导致未保护到整个临界区
- 后果：
 - UAF、越界写、数据损坏
 - 死锁、live lock（系统 hang）
 - 驱动行为不一致（间歇性 BUG）
- 在漏洞利用中：一些错误锁实现甚至可以被攻击者主动规避，形成可靠的利用链。

权限漏洞 (Permission Issues)

- Linux 的 Capability 模型很复杂，许多驱动开发者不熟悉：
 - 普通用户不该拥有的功能
 - 允许任意读写设备寄存器
 - 提供调试接口却无权限检查
- 权限漏洞经常表现为“接口暴露”：
 - 一个 `ioctl` 可以直接读写内核地址
 - 一个 `debugfs` 文件可以执行敏感操作
 - `mmap` 返回物理地址导致 `SMAP/SMEP` 失效
- 这些漏洞往往无需复杂 `exploitation`，只需简单调用即可提权。

信息泄露 (Info Leak)

- 信息泄露是现代攻击链的关键一步：
 - 泄露内核指针 -> KASLR 退化
 - 泄露内存布局 -> 堆喷更精确
- 常见原因：
 - `copy_to_user` 未清理结构体填充区域
 - `/proc/kallsyms` 未隐藏符号
 - 字符设备/ 驱动返回的缓冲区未初始化
- 特点：信息泄露本身不致命，但一旦配合 UAF / OOB，可极大提升利用可靠性。

解析器漏洞 (Parser Bugs)

- 协议/文件系统解析器往往是复杂状态机：
 - 多层嵌套结构 (**ext4**、**xfs**)
 - 用户可构造格式 (网络报文)
 - 深度依赖上下文 (**BPF verifier**)
- 常见问题：
 - 长度字段不可信，导致越界访问
 - 状态机没有处理“异常跳转”
 - 解析深度过深导致栈溢出
- 硬盘镜像/数据包可以精心构造，绕开内核逻辑 -> 非常危险。



网络协议栈漏洞

- **Linux 网络协议栈是攻击面最暴露的部分之一：攻击者无需本地权限即可发送数据包触发漏洞。**
- 常见错误：
 - 报文长度检查不足
 - **offload (GSO/GRO)** 处理中的偏移错误
 - 解包过程未验证指针边界
- 攻击影响：
 - 远程崩溃
 - 远程数据泄露
 - 在某些情况下，远程 **RCE** (历史上已有多次)

接口滥用 (API Misuse)

- 最容易出现漏洞的类别，因为驱动开发者常不熟悉所有 API 的约束。
- 常见 API 使用错误：
 - `copy_from_user / copy_to_user` 未检查返回值
 - `get/put_user` 访问未对齐或越界
 - `ioctl` 直接信任用户传入指针
 - `mmap` 将物理地址映射给用户导致内核完全暴露
 - `refcount` 错误使用 `atomic_t` 导致 `underflow`
- 本质都是：** 把用户输入当成可信内容 **。

常见利用技巧 (Exploitation Techniques)

- 在现代防御存在的情况下，利用流程通常包含：
 - ① 泄露信息 -> 绕过 **KASLR**
 - ② 获得任意读写能力 (通过 UAF/OOB)
 - ③ 修改关键内核结构
- 常见攻击目标：
 - task_struct -> 改 cred 指针
 - cred 结构体 -> uid/gid 设置为 0
 - ops 表 / vtable -> 劫持函数指针
- 关键：内核中大量结构体可达、可喷、可伪造，使 exploitation 成为可工程化的过程。

内核防御与绕过策略

- 内核现代防御：
 - KASLR
 - SMEP / SMAP
 - 栈保护 (stack protector)
 - 内核堆隔离 (SLAB 分离)
 - Supervisor Mode Access Prevention
- 绕过方式：
 - 利用信息泄露恢复内核基址
 - ROP chain 在内核态执行
 - 利用 NULL 函数指针绕过 SMEP
 - 使用 userfaultfd 构造精确竞态窗口



小结

- 内核漏洞类型丰富，但本质都源于：**边界检查、生命周期管理、并发控制、权限控制不当**。
- 内核复杂度极高 -> 漏洞不可避免
- 理解漏洞成因有助于后续学习 **exploit** 和分析真实案例

- ① [攻击面概述](#)
- ② [内核攻击面分类与枚举](#)
- ③ [内核漏洞原理](#)
- ④ [Linux 内核漏洞点深度解析](#)



为什么要研究“漏洞点”？

- 即使攻击面很大，也不是所有代码都容易出漏洞
- 大量历史 CVE 集中在固定模式、固定机制
- 理解这些“Hotspots”能指导：
 - 漏洞挖掘
 - 内核审计
 - 安全设计
 - 代码改进
- 本节聚焦四类高危机制：
 - ① 引用计数（refcount）
 - ② 内存管理（slab、页框、mapping）
 - ③ 对象生命周期（lifetime）
 - ④ 并发与同步

关键视角

不是“哪里有接口”，而是“哪里最容易写错”。

漏洞点 1: 引用计数 (refcount)

- 内核大量对象依赖引用计数管理生命周期:
 - 文件对象 (file)
 - inode / dentry
 - socket
 - 内存页 (page)
 - 设备对象 (kobject/kref)
- 常见错误模式:
 - 多次 **put -> use-after-free**
 - 漏 **put -> 永久引用** · 造成资源泄漏
 - 非原子操作 -> 竞态下 **ref** 不一致
 - 路径过于复杂 -> 异常路径容易漏计数
- **refcount** 错误往往是链式漏洞的开端 (UAF、double-free)

特性

引用计数是“逻辑错误”与“并发错误”的交叉点。

引用计数为何如此危险？（内核案例总结）

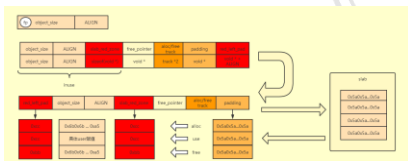
- **kref** 的语义非常严格：
 - **get/put** 必须 1:1
 - **put** 之后对象可能立即销毁
 - 锁保护不足会造成不同 CPU 间 **ref** 竞争
- 常见真实漏洞案例模式：
 - 错误使用 “**if (-refcount == 0)**” 而非原子 API
 - 在错误路径忘记 **kref_put**（尤其是 **goto** 清理分支）
 - 对象复用但忘记重新初始化 **kref**
 - **kref_put** 和 **free** 分离导致双重释放
- 教学意义：展示逻辑漏洞如何演变为可利用 **UAF**

漏洞点 2: 内存管理 (Memory Management)

- 内核 MM 子系统非常复杂, 包含:
 - slab 分配器 (kmalloc/kfree)
 - 页框分配
 - vma 管理 (maple tree/红黑树)
 - 用户内存访问 (copy_from_user)
 - 页面回收与 COW
- 常见漏洞模式:
 - 类型混淆 (**type confusion**)
 - **overlap**: 结构体重叠导致字段污染
 - 未初始化内存泄露 (**infoleak**)
 - 长度字段未检查 -> **slab OOB**
- 教学重点: MM 错误很少是简单越界, 多是“结构体语义被破坏”

SLAB/SLUB: 为什么漏洞特别多?

- 内核对象在 **slab** 中**高密度排列** (紧挨着)
- 写越界极易破坏邻近对象-> 从越界变成“能力升级”
- 典型漏洞点:
 - `copy_from_user` 未验证长度
 - 用户可控结构体包含大型数组
 - 数据与指针混排导致越界覆盖函数指针
- 重要案例模式:
 - OOB -> 污染 `next` 指针 -> 任意地址写
 - OOB -> 覆盖 `refcount` -> UAF/overflow
 - OOB -> 覆盖 `ops` 虚函数表 -> ROP 起点



漏洞点 3: 对象生命周期 (Lifetime Bugs)

- 内核对象 **often have**:
 - 创建路径复杂 (多分支)
 - 销毁路径更复杂 (异步回调、RCU、工作队列)
- 常见生命周期漏洞:
 - 提前释放 (**early free**) -> **UAF**
 - 延迟释放 (**late free**) -> **double free**
 - 异步释放 **race**: **RCU grace period** 未正确处理
 - 对象共享但未拷贝 -> **COW** 逻辑被破坏
- 漏洞本质: “对象什么时候不再被使用? 谁负责释放?”

典型生命周期错误模式（从真实漏洞抽象）

- 路径 **A**: get -> workqueue -> put（异步）
- 路径 **B**: error path -> put（同步）
- 若两者 race，可能：
 - 被 put 两次 -> 双重释放
 - workqueue 中引用 stale pointer -> UAF
- 常见于：
 - 网络协议栈（TCP 定时器）
 - 文件系统（inode 回收）
 - 驱动（buffer 提交后异步完成）

漏洞点 4: 并发与锁 (Concurrency Issues)

- Linux 内核是一个高度并发的系统:
 - 多核同时执行
 - 中断可在任意点打断
 - RCU/atomic/seqlock 多种同步机制混杂
- 常见并发错误:
 - 缺锁 -> **data race**
 - 锁顺序错误 -> 死锁 / **ABBA**
 - 部分保护 -> 半同步半异步访问
 - **RCU** 内存未隔离 -> 过早释放
- 并发漏洞常表现为:
 - 偶现性崩溃 (难复现)
 - race -> UAF
 - race -> 越界访问



本节小结

- 内核安全不仅是“入口”问题，更是“内部机制复杂度”问题
- 本节四大漏洞点最值得关注：
 - ① 引用计数：逻辑 + 并发错误的融合点
 - ② 内存管理：结构体密集排列 -> 调用链长
 - ③ 生命周期：异步事件和清理路径最危险
 - ④ 并发与锁：错误使用导致高危 race
- 这些漏洞点构成多数内核漏洞的根源
- 从挖掘到利用，都围绕这些机制展开