北京大学
PEKING UNIVERSITY

DirtyCred: Escalating Privilege in Linux Kernel

谢沅甫 2024. 12. 19

相关资料

北京大学 PEKING UNIVERSITY

- 论文 Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security
  - https://dl.acm.org/doi/10.1145/3548606.3560585
- github 仓库
  - https://github.com/markakd/DirtyCred
- blackhat presentation
  - https://i.blackhat.com/USA-22/Thursday/US-22-Lin-Cautious-A-New-Exploitation-Method.pdf

回顾 Dirty Pipe

北京大学 PEKING UNIVERSITY

- CVE-2022-0847
- 是一个 pipe 功能中未初始化的漏洞
- 影响 v5.8 及更高版本
- Data-only攻击，缓解机制无效
- 可以覆写任何拥有“读”权限的文件
- 可以在 Android 上提权

回顾 Dirty Pipe

北京大学 PEKING UNIVERSITY

- **Data-only 的能力很强**
  - 通用的利用方法
  - 能绕过 CFI 检查
  - 过去的防护无效

- **Dirty Pipe 的不足点**
  - 不能完成容器逃逸
  - 不是一个常规的攻击手段

DirtyCred 介绍

北京大学 PEKING UNIVERSITY

- 攻击思路
  - 交换 Linux kernel 中的 Credentials

- 好处
  - 是常规的攻击手段，攻击简单且影响范围大
  - 和 Dirty Pipe 类似通用的 Data-only 利用方法
  - 能进行容器逃逸攻击

和 Dirty Pipe 的比较

北京大学 PEKING UNIVERSITY

	Dirty Pipe	DirtyCred
• 常规攻击手段	✗	✓
• Data-only，通用的利用方法	✓	✓
• 能绕过 CFI	✓	✓
• 能容器逃逸	✗	✓
• 攻击仍然存在	✗	✓

Kernel Credential

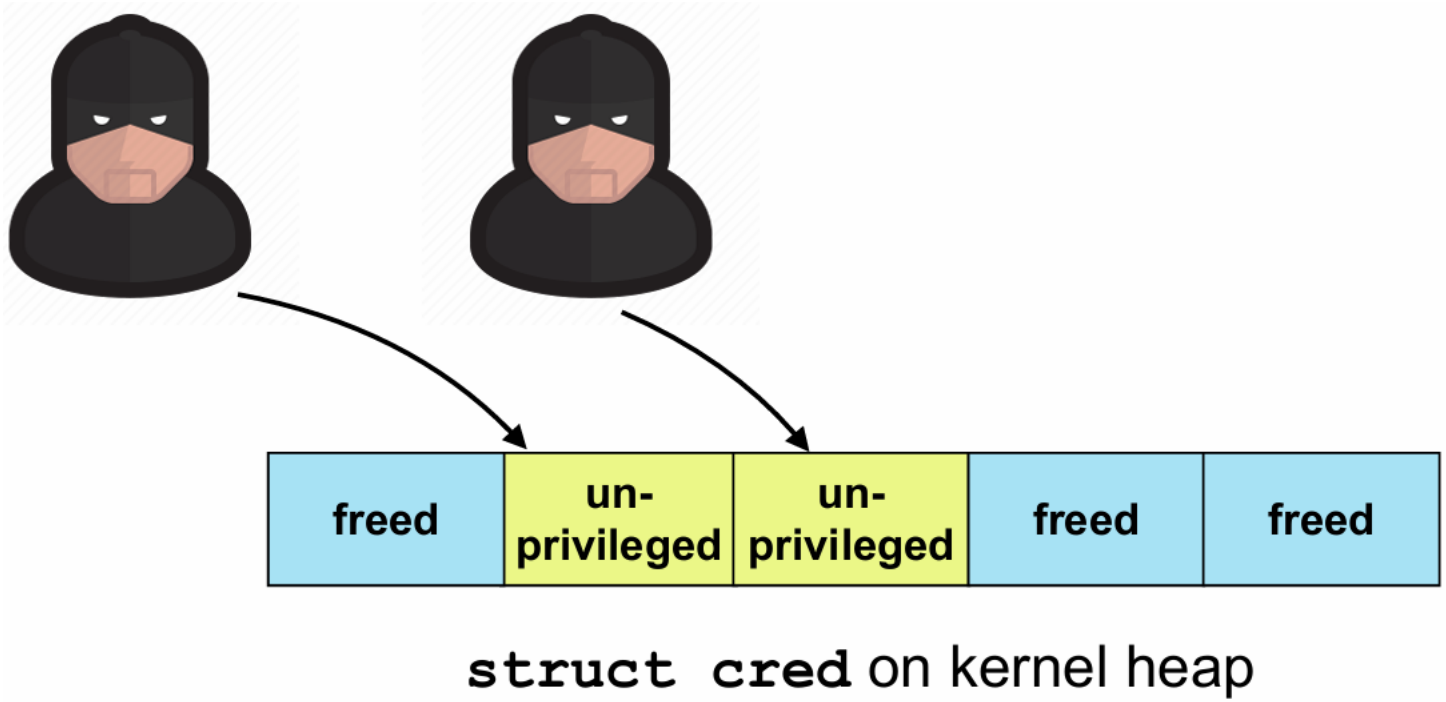
北京大学 PEKING UNIVERSITY

- **Credential 是一种在内核中携带 privilege 信息的结构体**
  - 在内核文档中定义
  - 体现了 **privilege** 和 **capability**
  - 两种主要类型：**task credentials** 和 **open file credentials**
  - credentials 存在安全检查

Source: https://www.kernel.org/doc/Documentation/security/credentials.txt

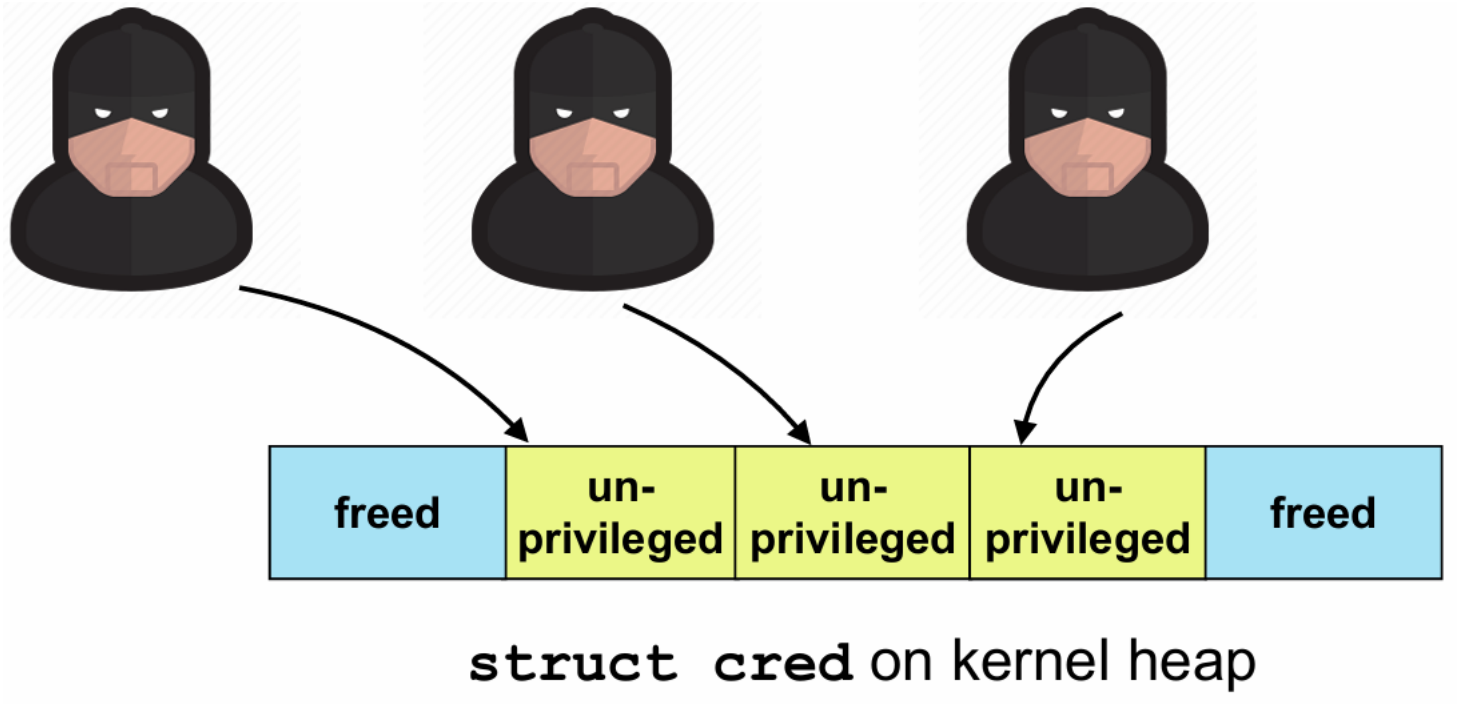
Task Credential

- **Struct cred** 在 kernel 中的实现



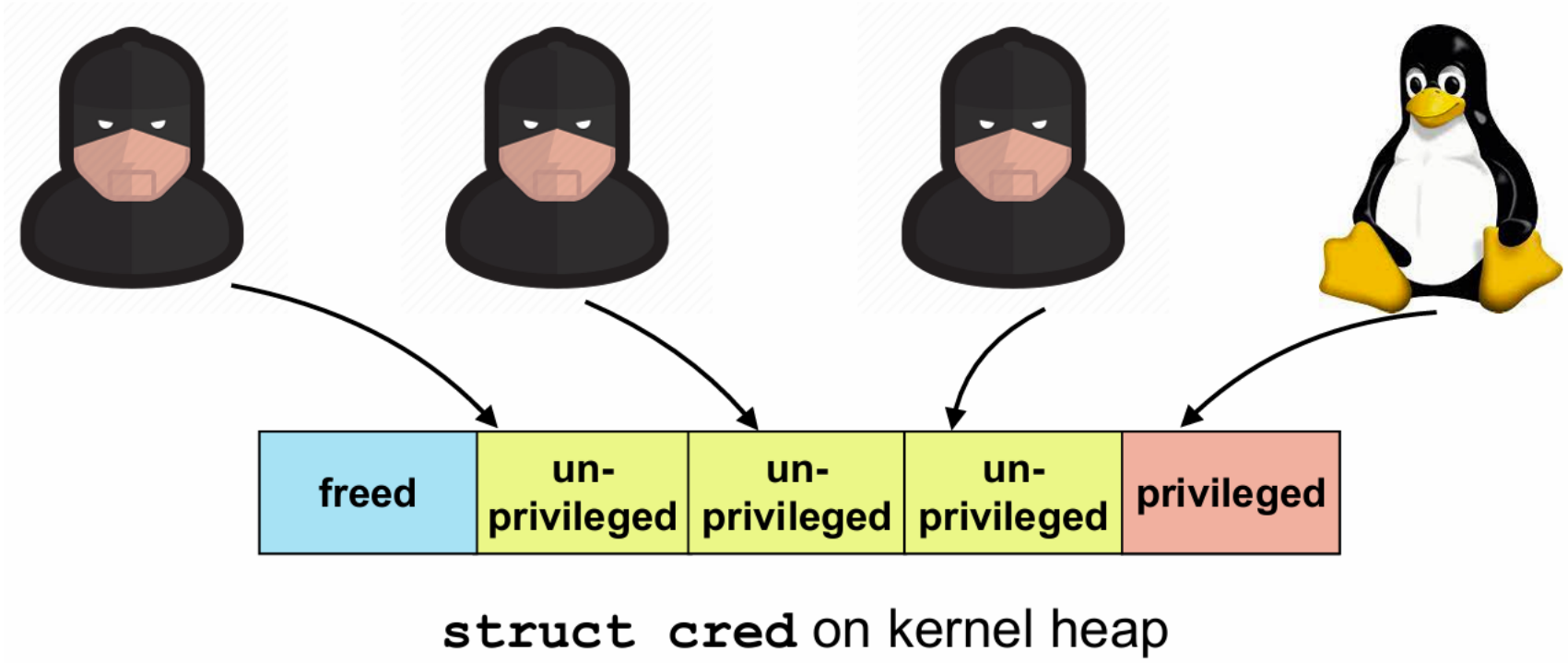
Task Credential

- **Struct cred** 在 kernel 中的实现



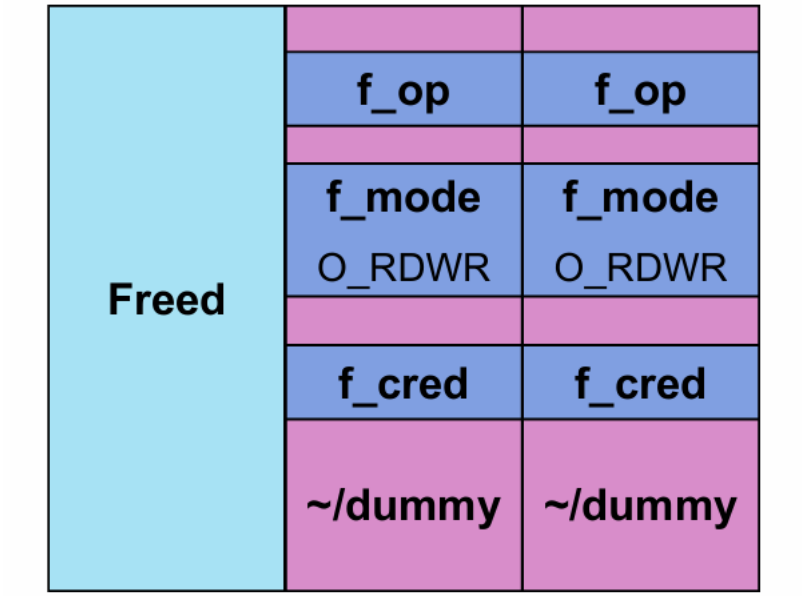
Task Credential

- **Struct cred** 在 kernel 中的实现



Open File Credentials

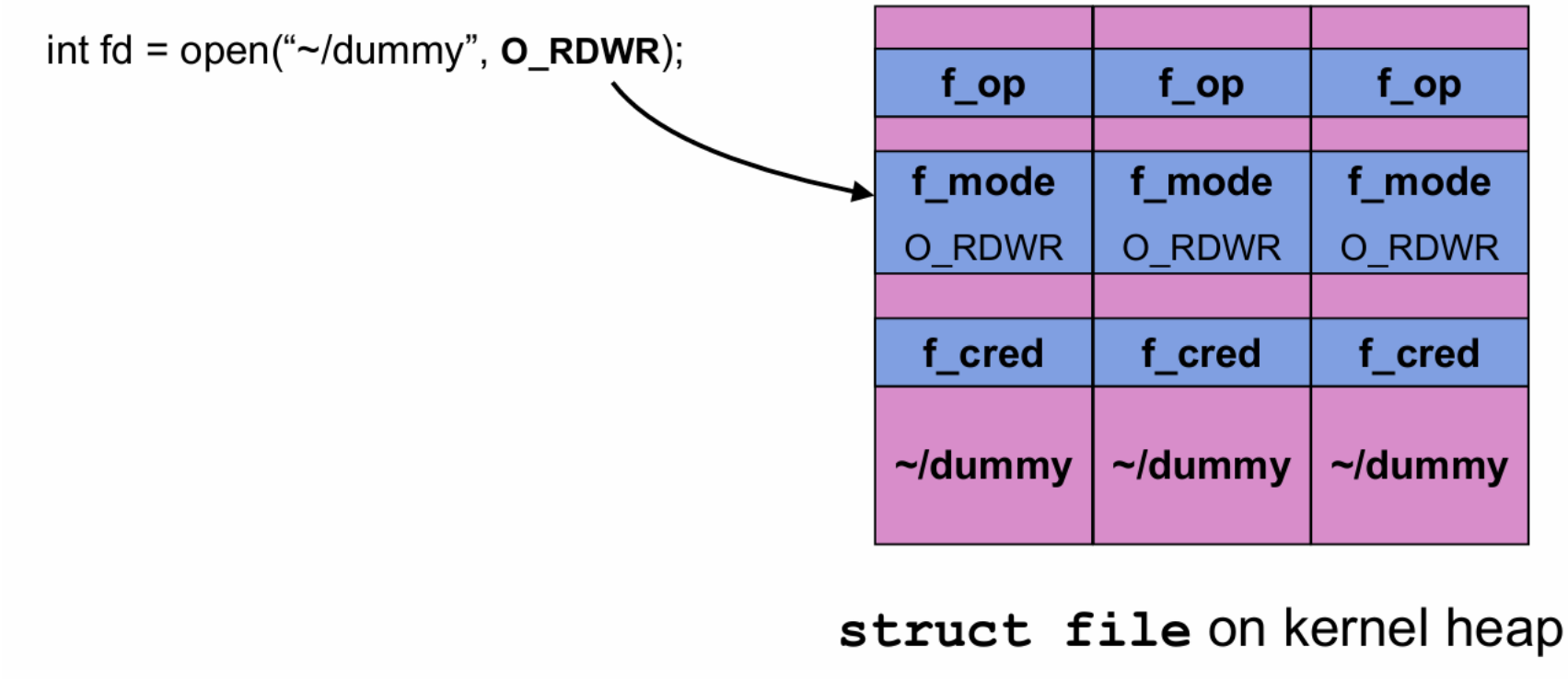
- **Struct file** 在 kernel 中的实现



`struct file` on kernel heap

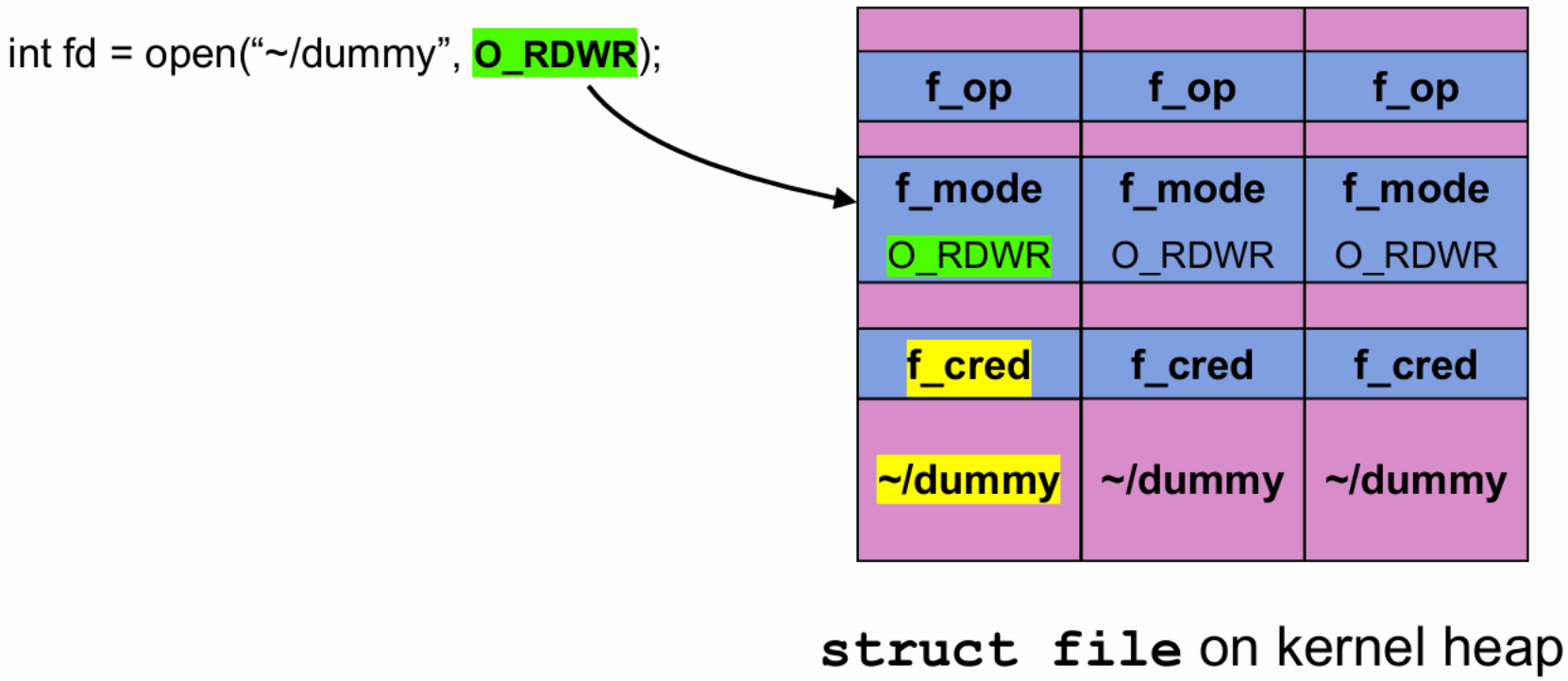
Open File Credentials

- **Struct file** 在 kernel 中的实现



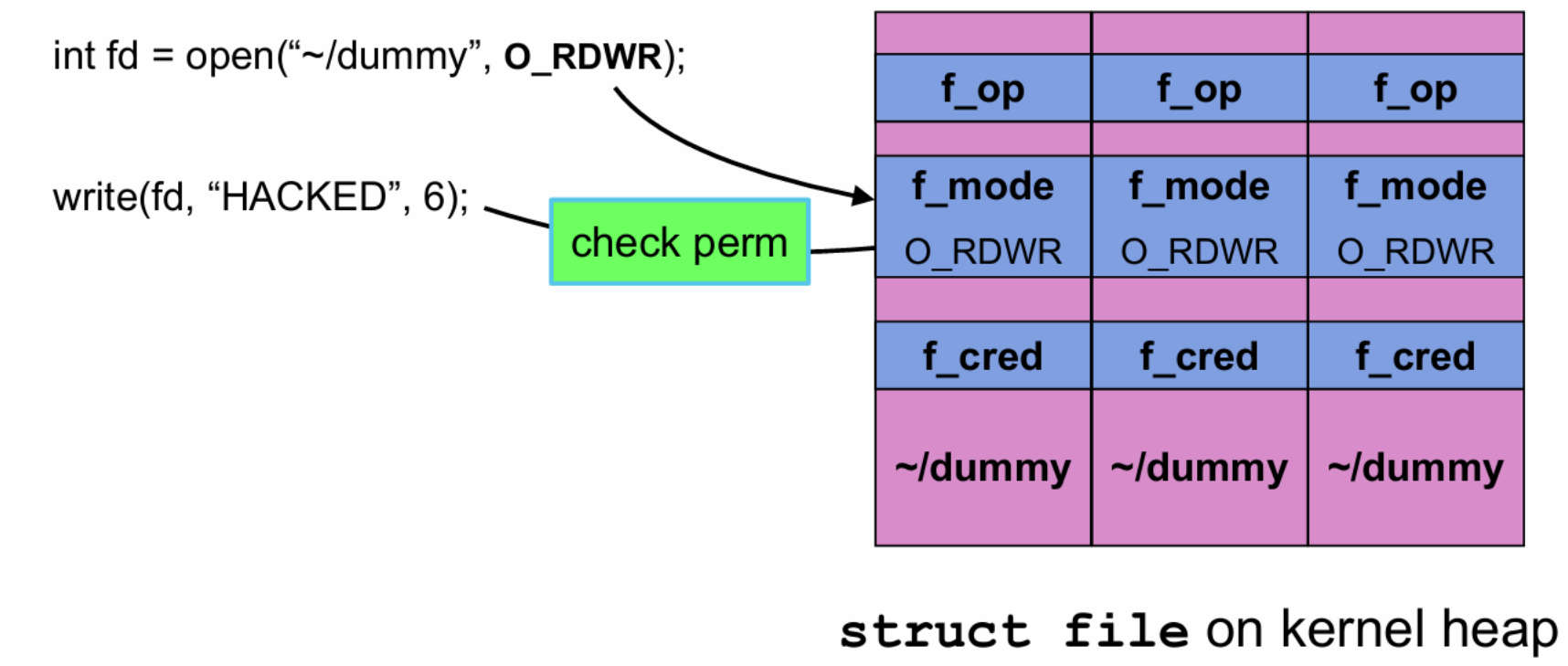
Open File Credentials

- **Struct file** 在 kernel 中的实现



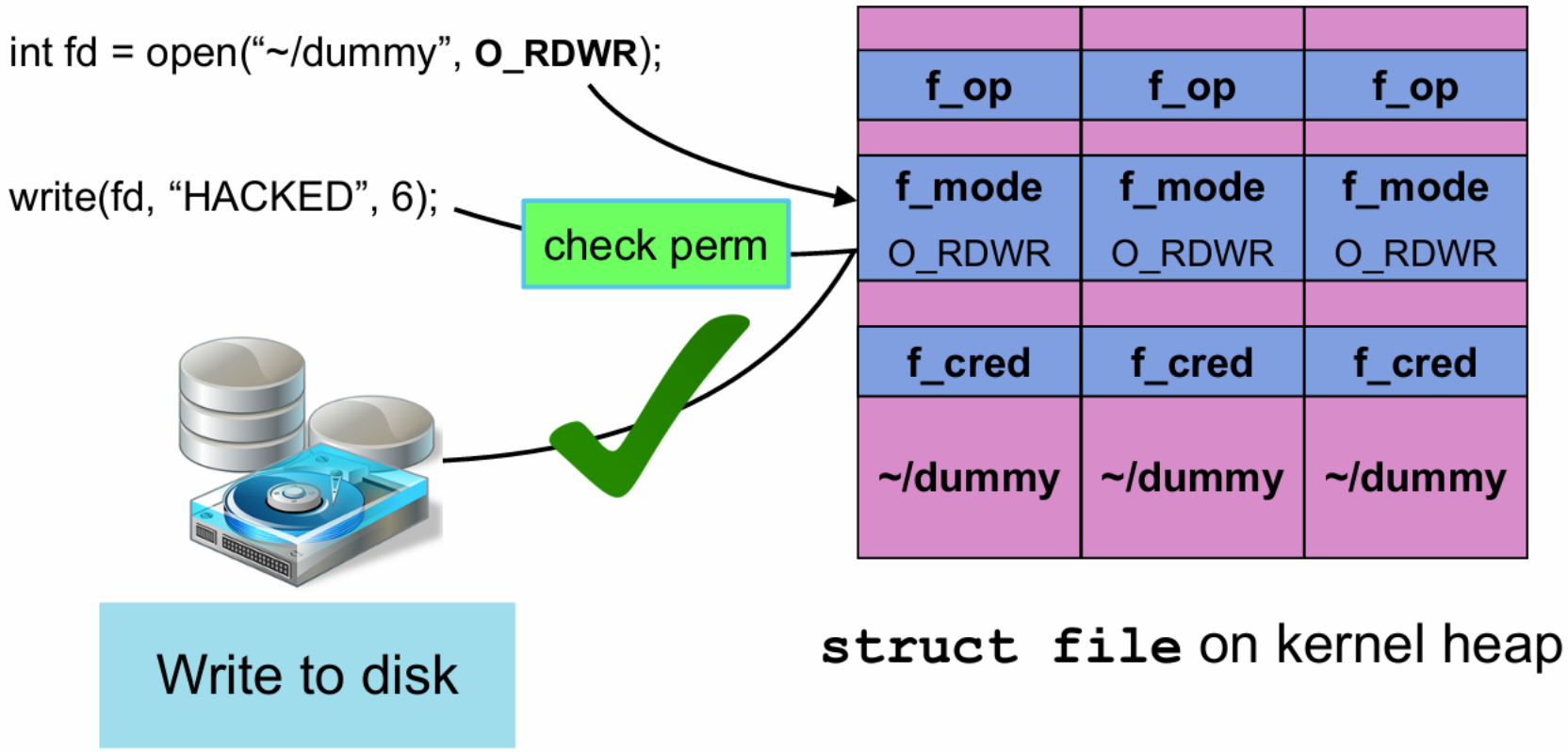
Open File Credentials

- **Struct file** 在 kernel 中的实现



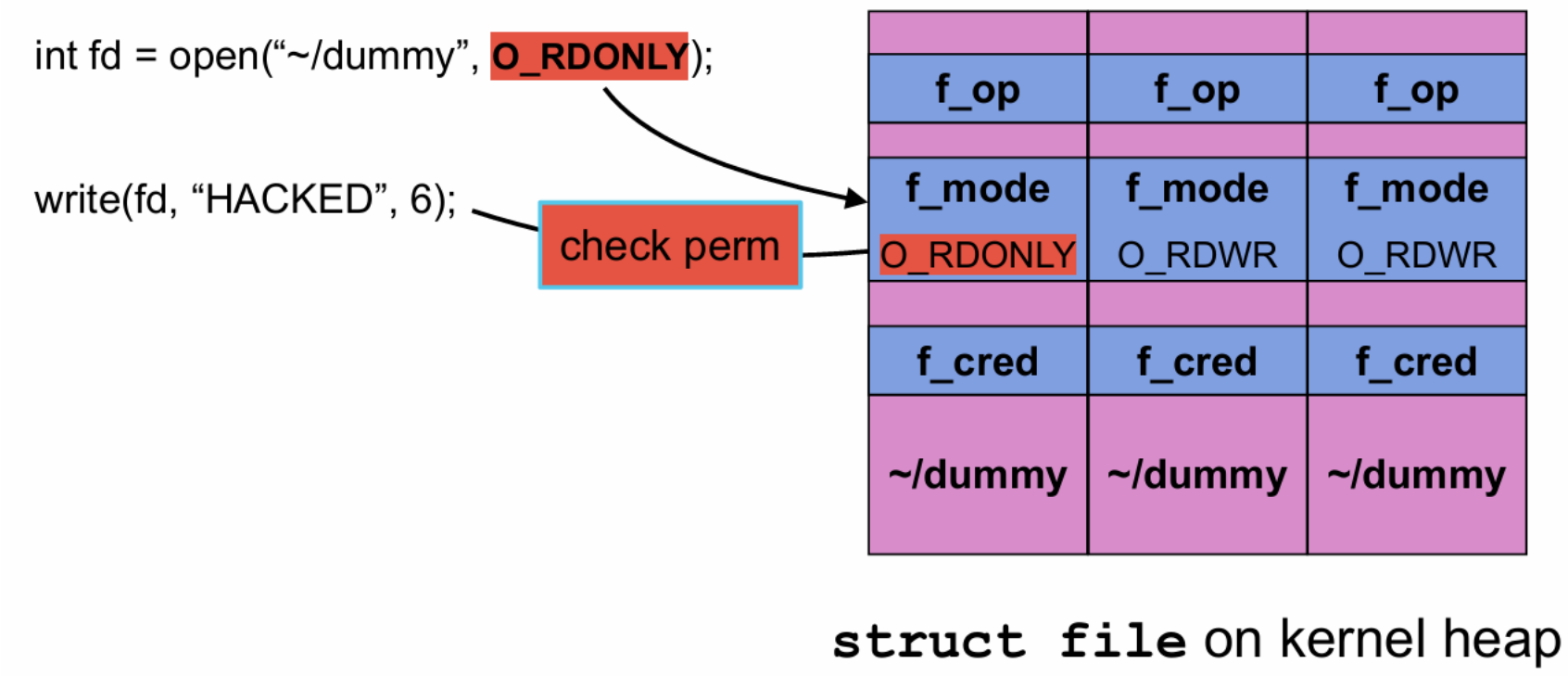
Open File Credentials

- **Struct file** 在 kernel 中的实现



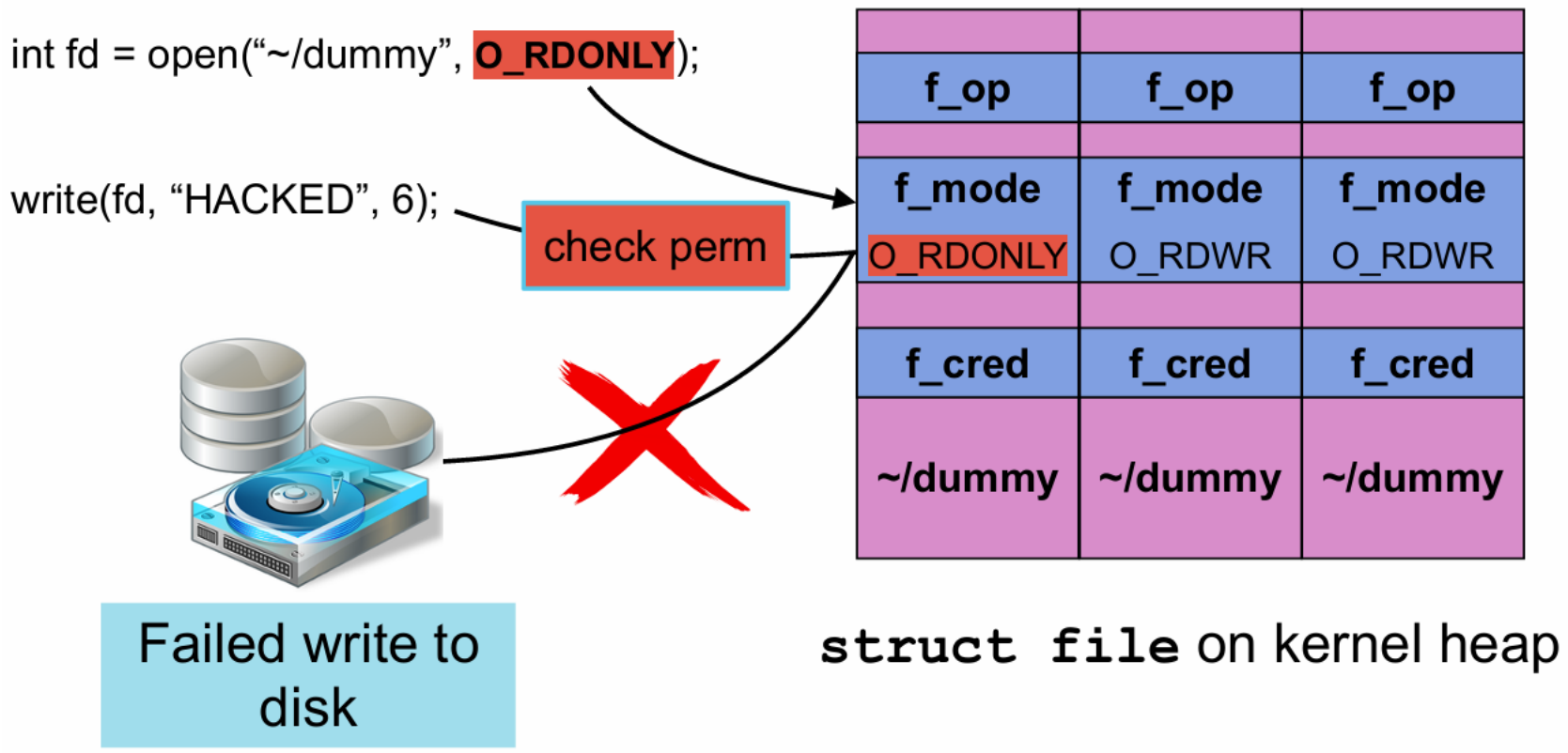
Open File Credentials

- **Struct file** 在 kernel 中的实现



Open File Credentials

- **Struct file** 在 kernel 中的实现



DirtyCred：Swapping Linux Kernel Credentials

北京大学 PEKING UNIVERSITY

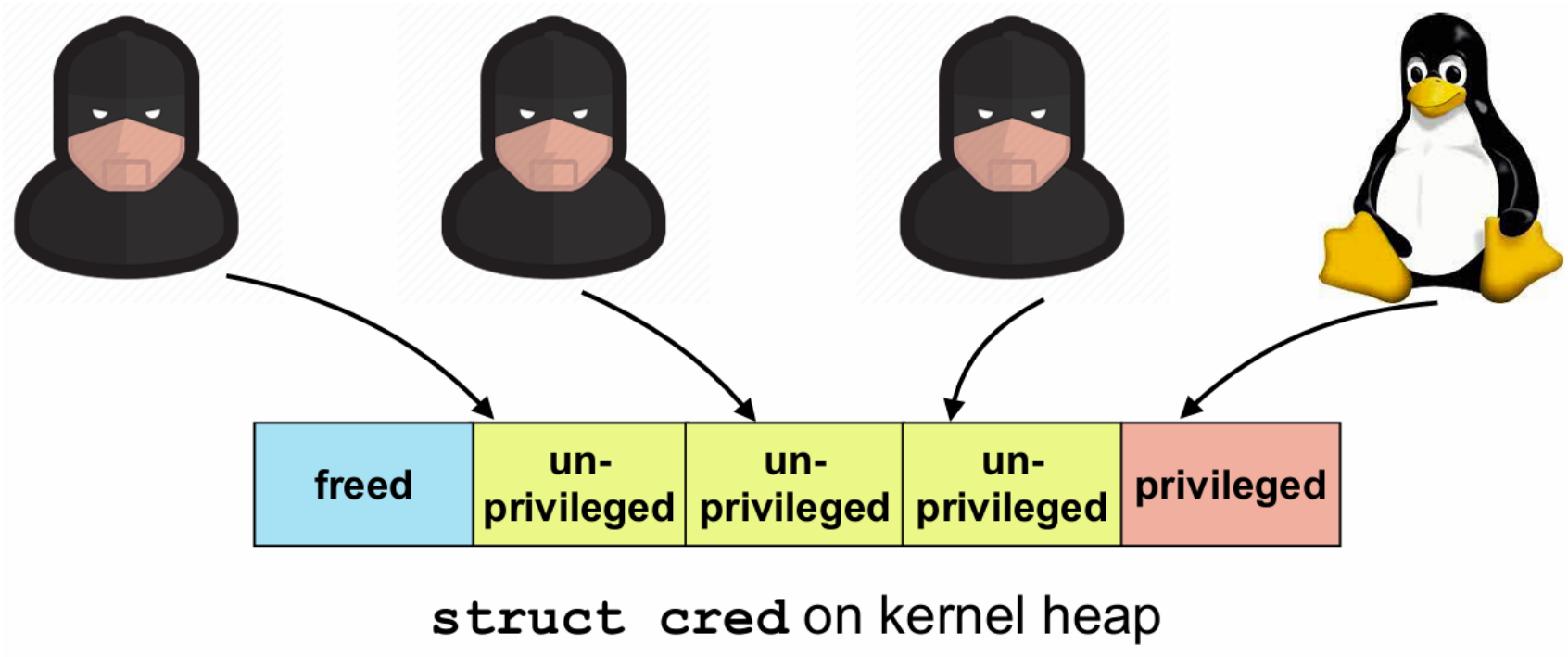
攻击思路

- 把 **unprivileged** credential 替换成 **privileged** 的

攻击路径

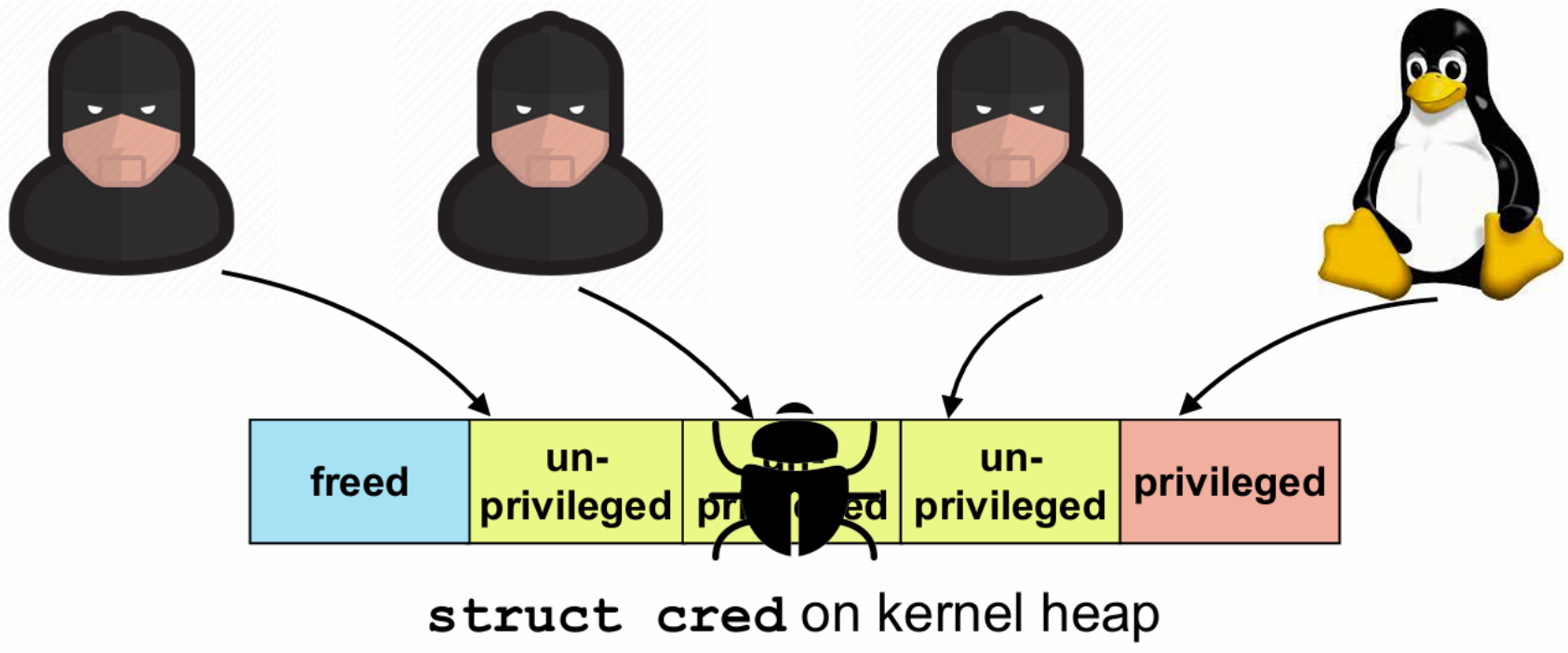
1. 攻击 task credentials (**struct cred**)
2. 攻击 open file credentials (**struct file**)

攻击 Task Credentials



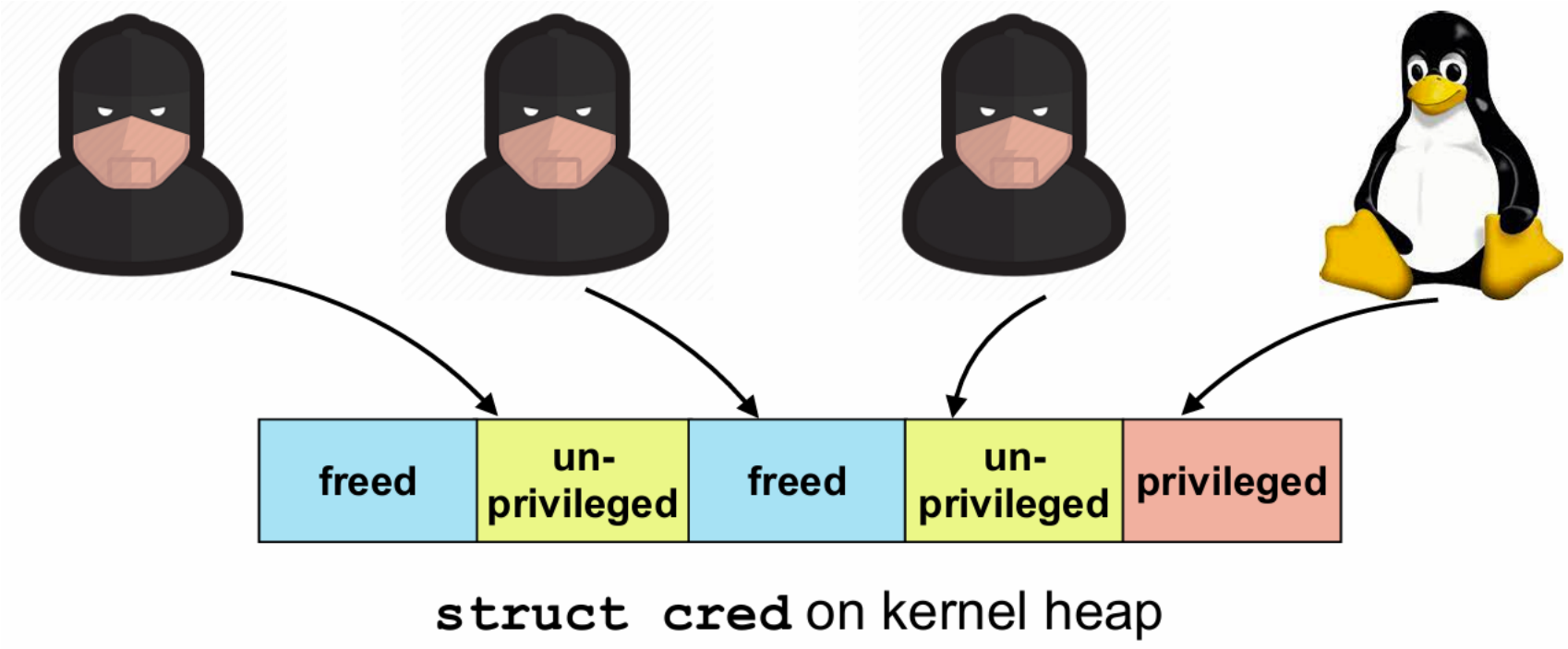
攻击 Task Credentials

步骤一：使用漏洞把 unprivileged credential 释放掉



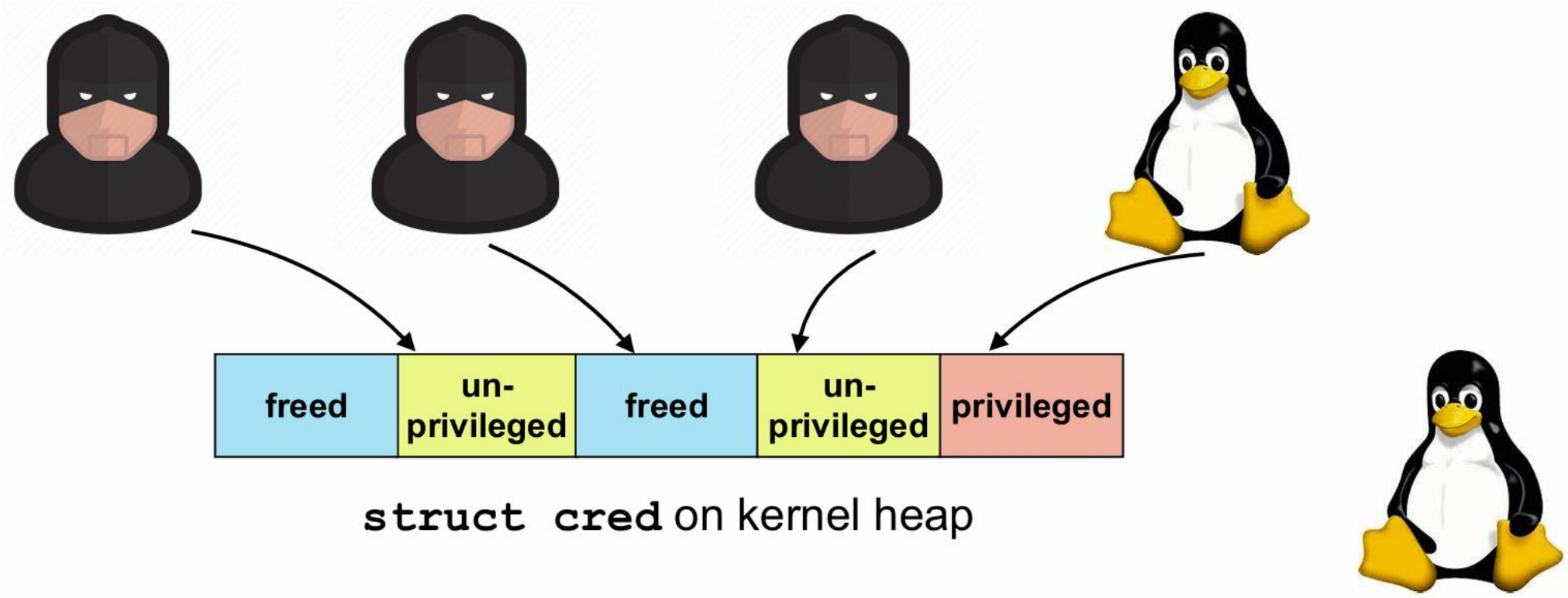
攻击 Task Credentials

步骤一：使用漏洞把 unprivileged credential 释放掉



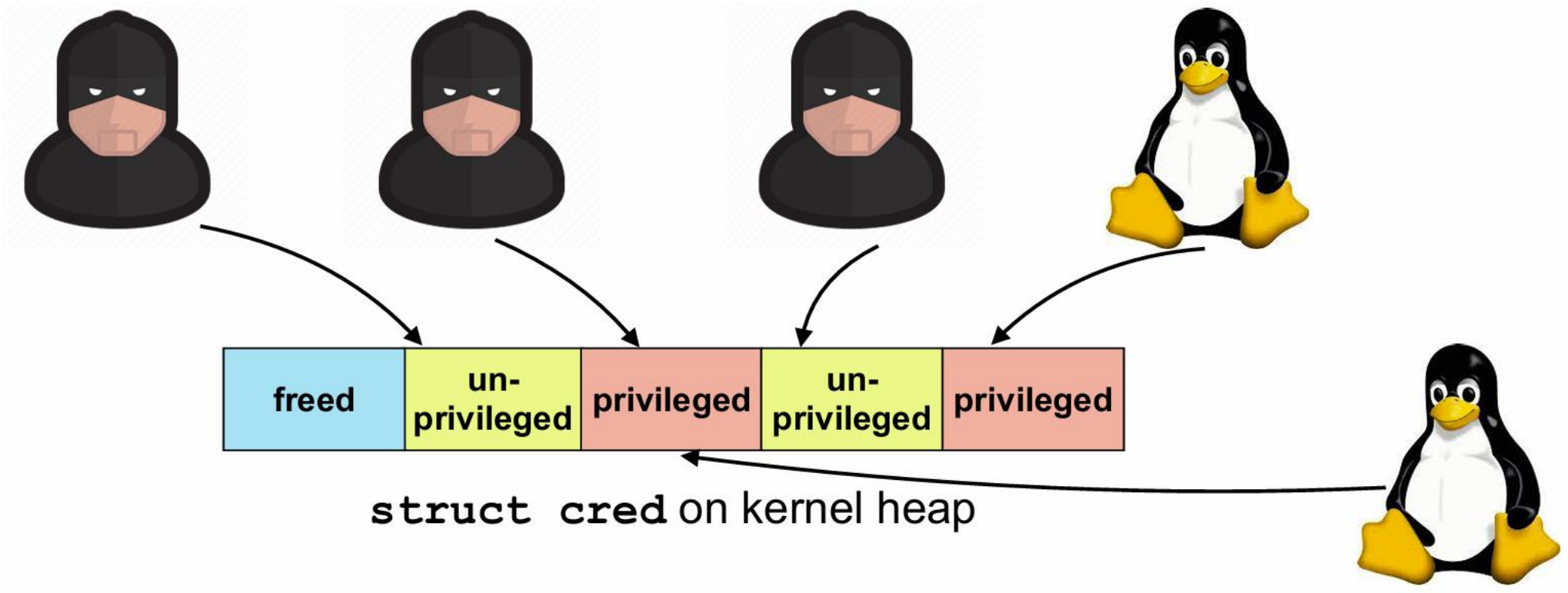
攻击 Task Credentials

步骤二：申请 privileged credential 放置在释放的内存中



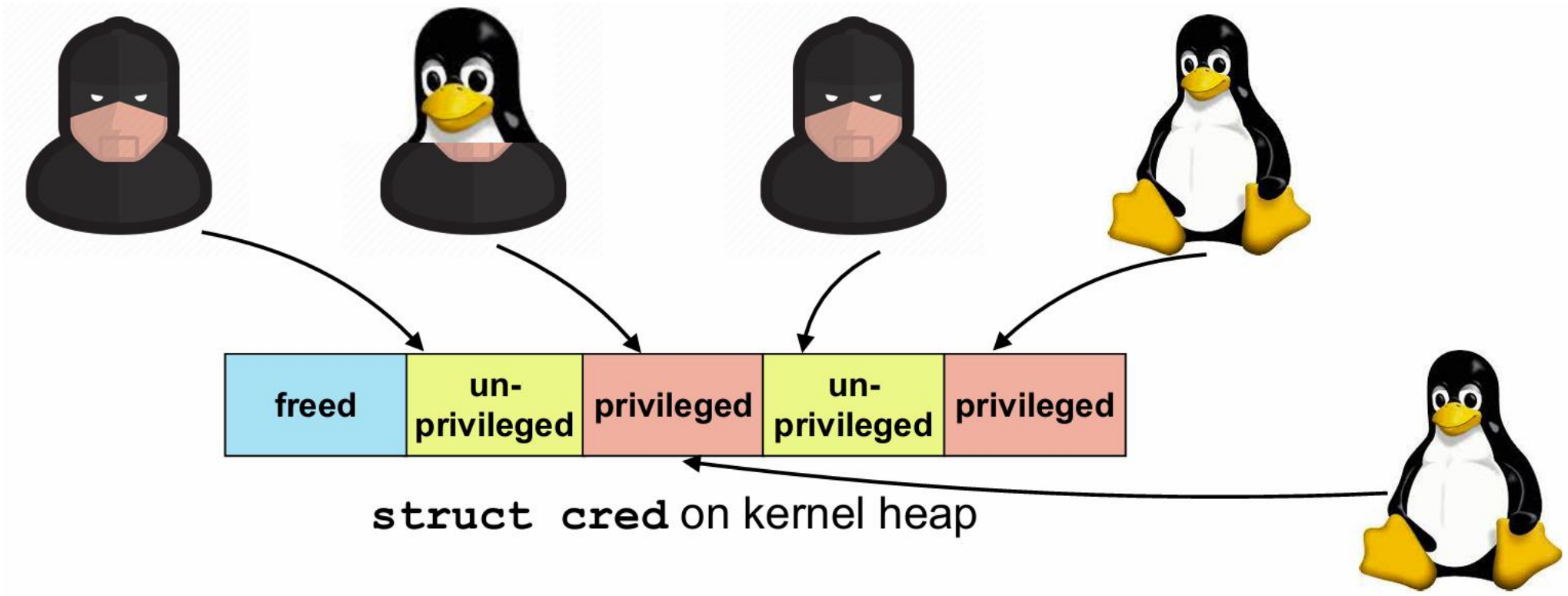
攻击 Task Credentials

步骤二：申请 privileged credential 放置在释放的内存中



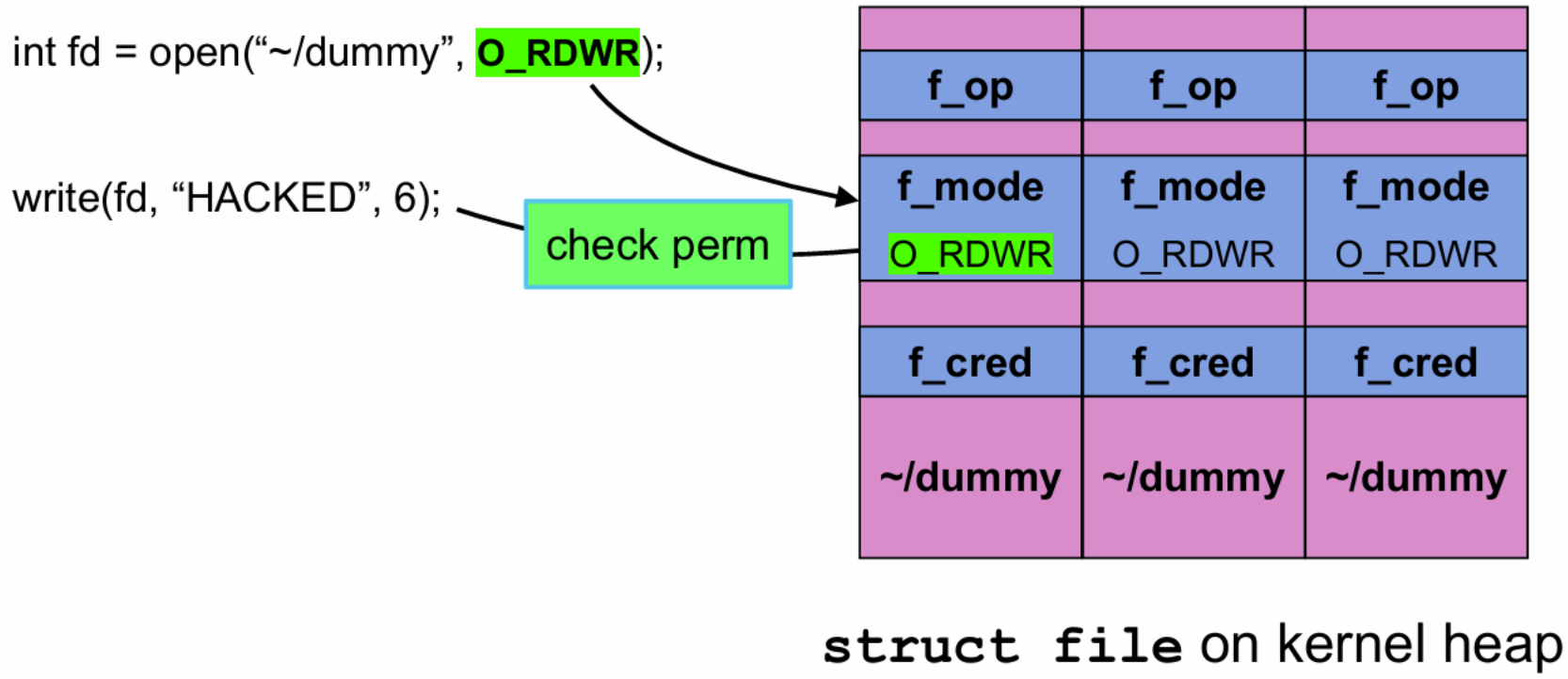
攻击 Task Credentials

步骤三：使用 privileged 特权进行操作



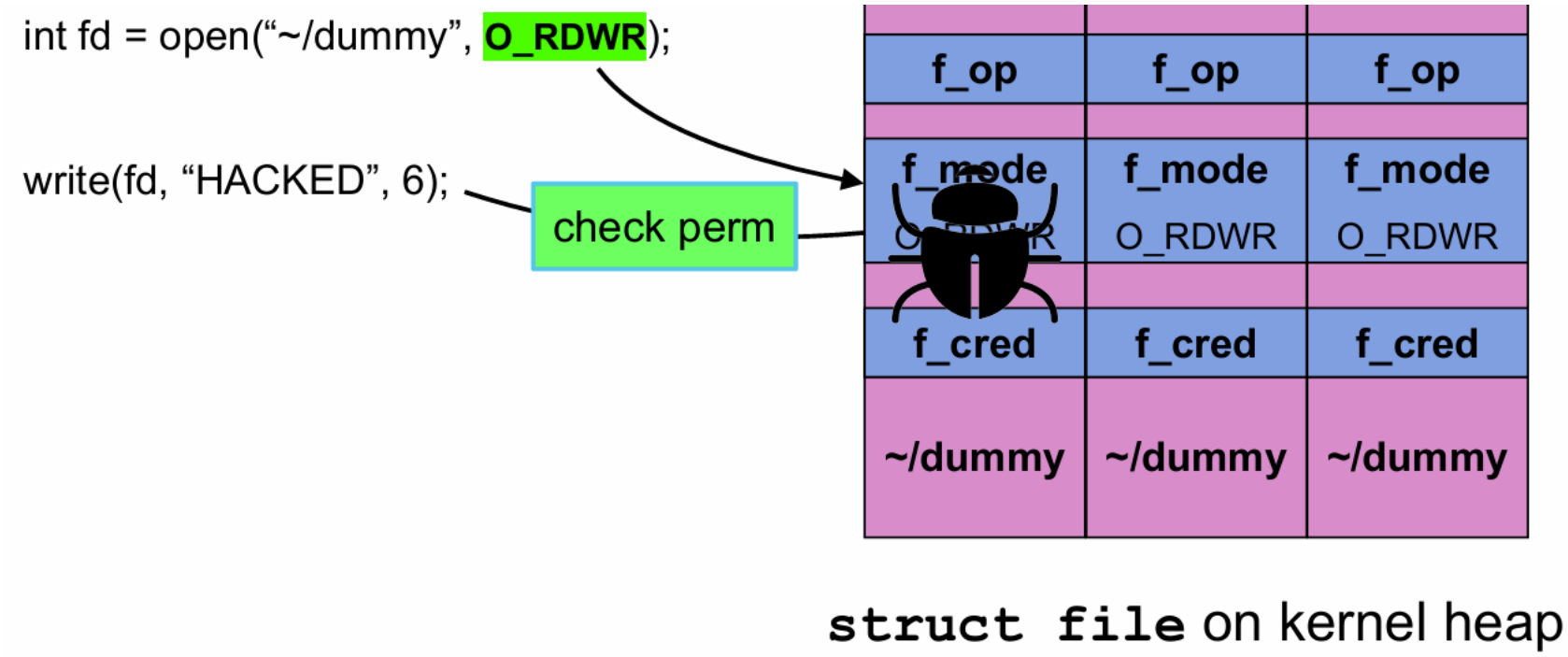
攻击 Open File Credentials

写入一个有写权限的文件



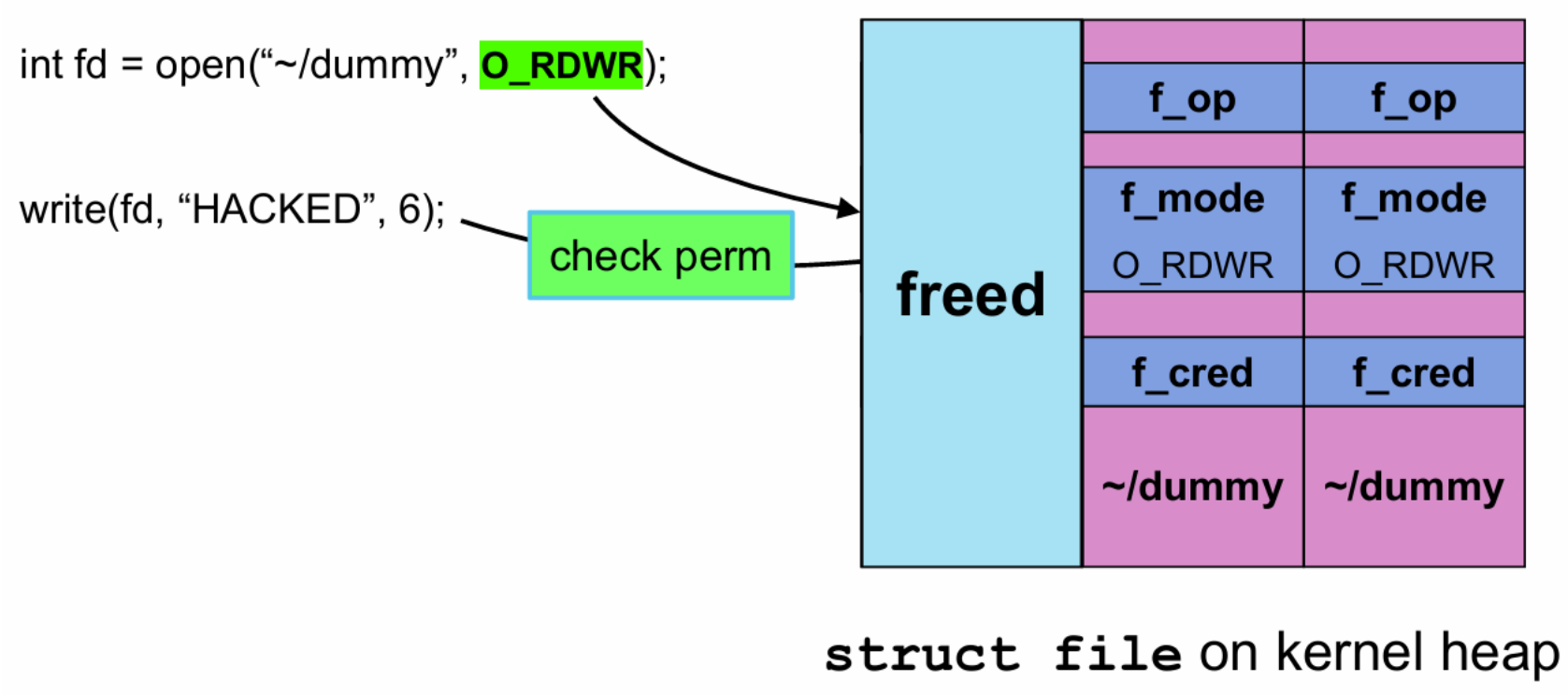
攻击 Open File Credentials

步骤一：在权限检查后，写操作前，用漏洞释放 **struct file**



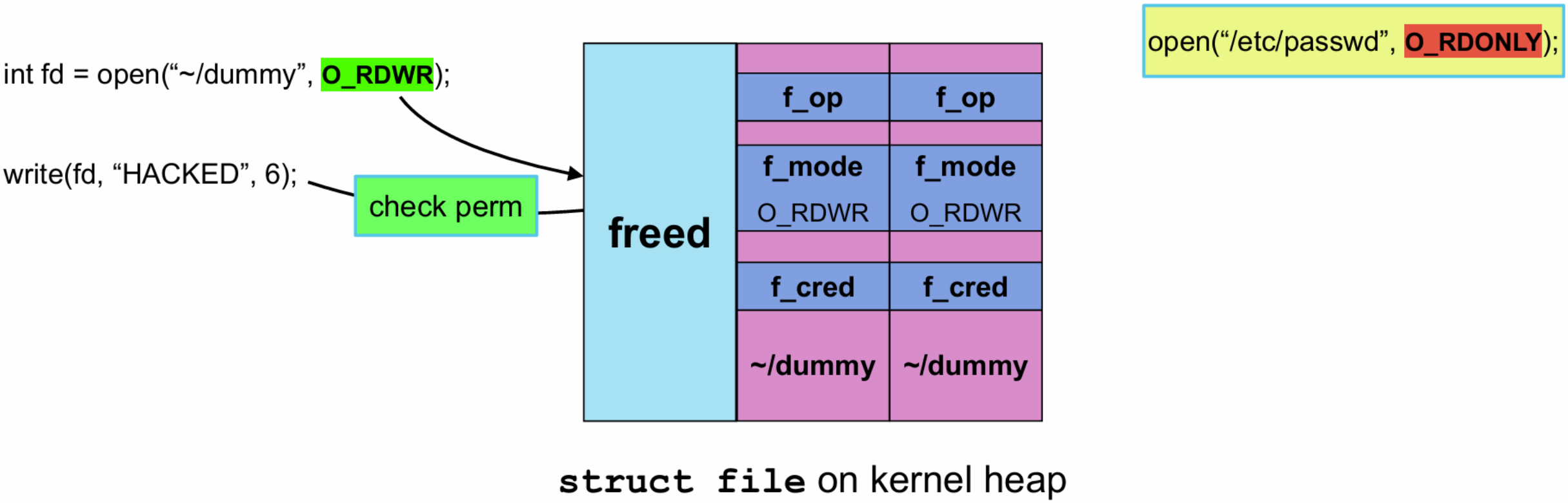
攻击 Open File Credentials

步骤一：在权限检查后，写操作前，用漏洞释放 **struct file**



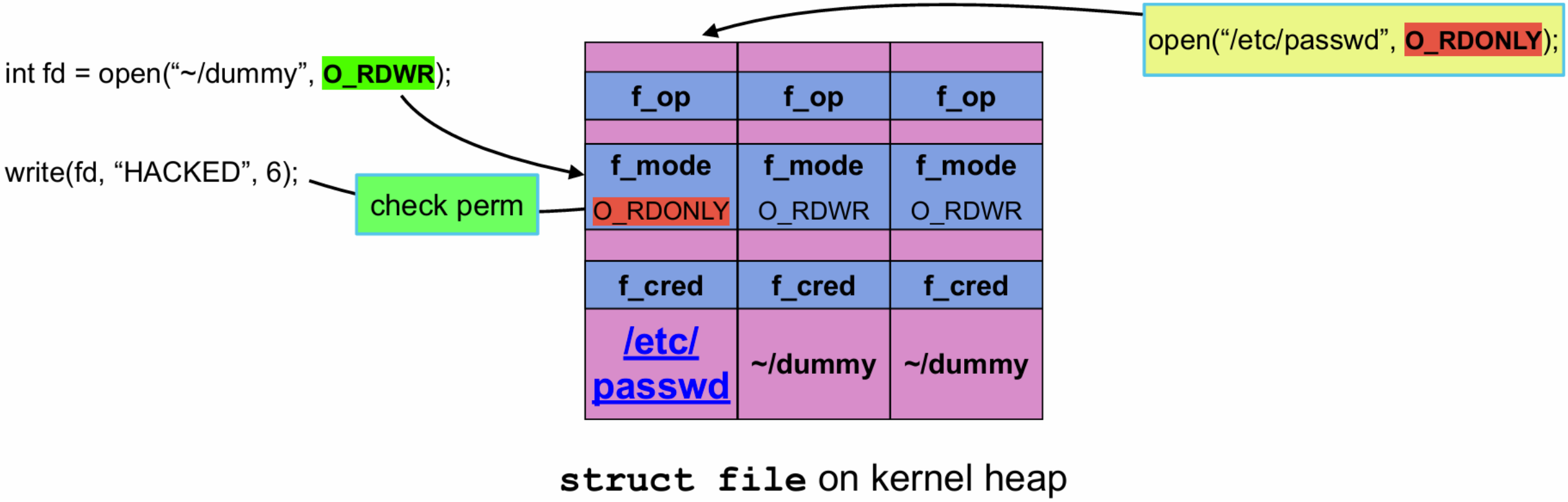
攻击 Open File Credentials

步骤二：申请一个只读文件的 struct file 到释放的内存中



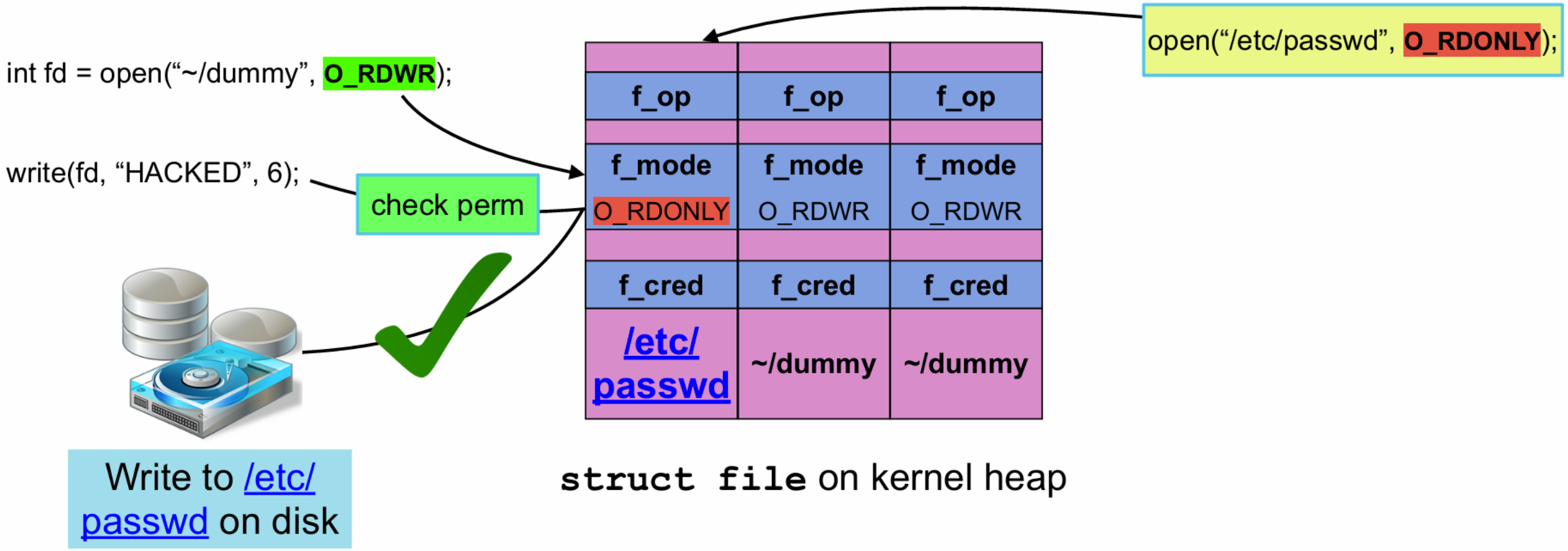
攻击 Open File Credentials

步骤二：申请一个只读文件的 struct file 到释放的内存中



攻击 Open File Credentials

步骤三：使用特权进行操作



DirtyCred：Swapping Linux Kernel Credentials

北京大学 PEKING UNIVERSITY

攻击步骤

1. 使用漏洞释放一个 unprivileged credential
2. 申请 privileged credential
3. 使用特权进行操作

Challenges

1. 如何释放 credentials
2. 如何在 unprivileged 用户下申请 privilege credentials (攻击 task credentials)
3. 如何稳固文件利用方法 (攻击 open file credentials)

Challenges

1. 如何释放 credentials
2. 如何在 unprivileged 用户下申请 privilege credentials (攻击 task credentials)
3. 如何稳固文件利用方法 (攻击 open file credentials)

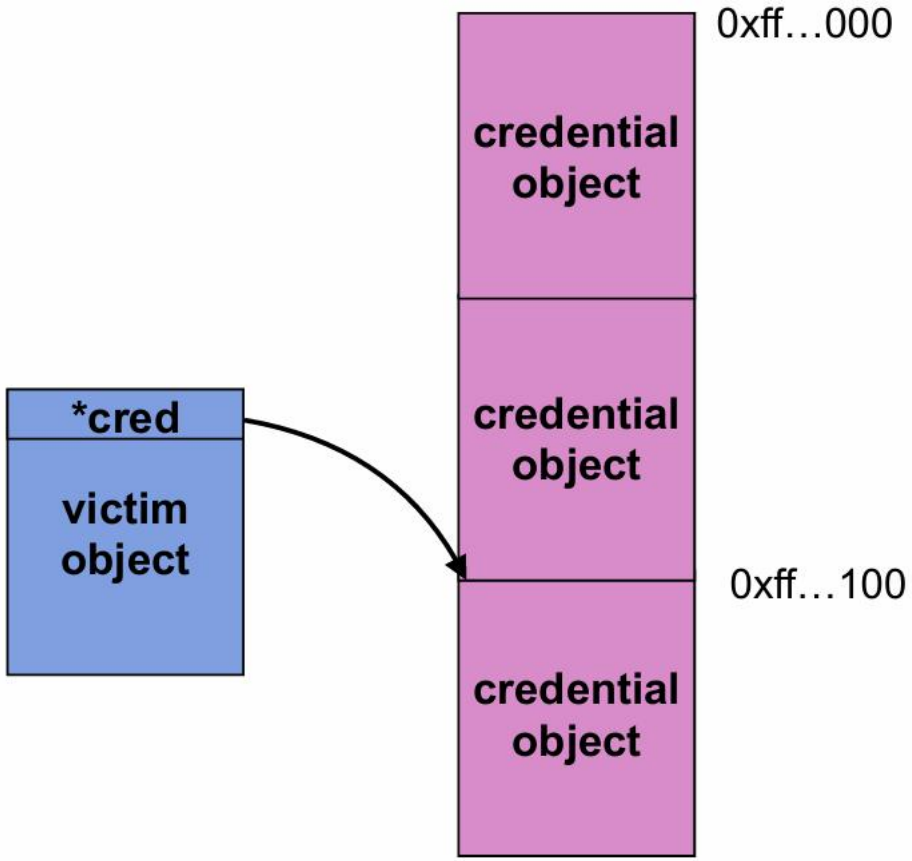
Challenge 1：如何释放 Credentials

- **struct cred** 和 **struct file** 都在 **dedicated** caches 中
- 大部分漏洞都发生在 **generic** caches
- 大部分漏洞没有 free 的能力

- 解决方法：迁移其他漏洞的能力
  - 迁移 Invalid-Write (例如 OOB & UAF write)
  - 迁移 Invalid-Free (例如 Double-Free)

Pivoting Invalid-Write

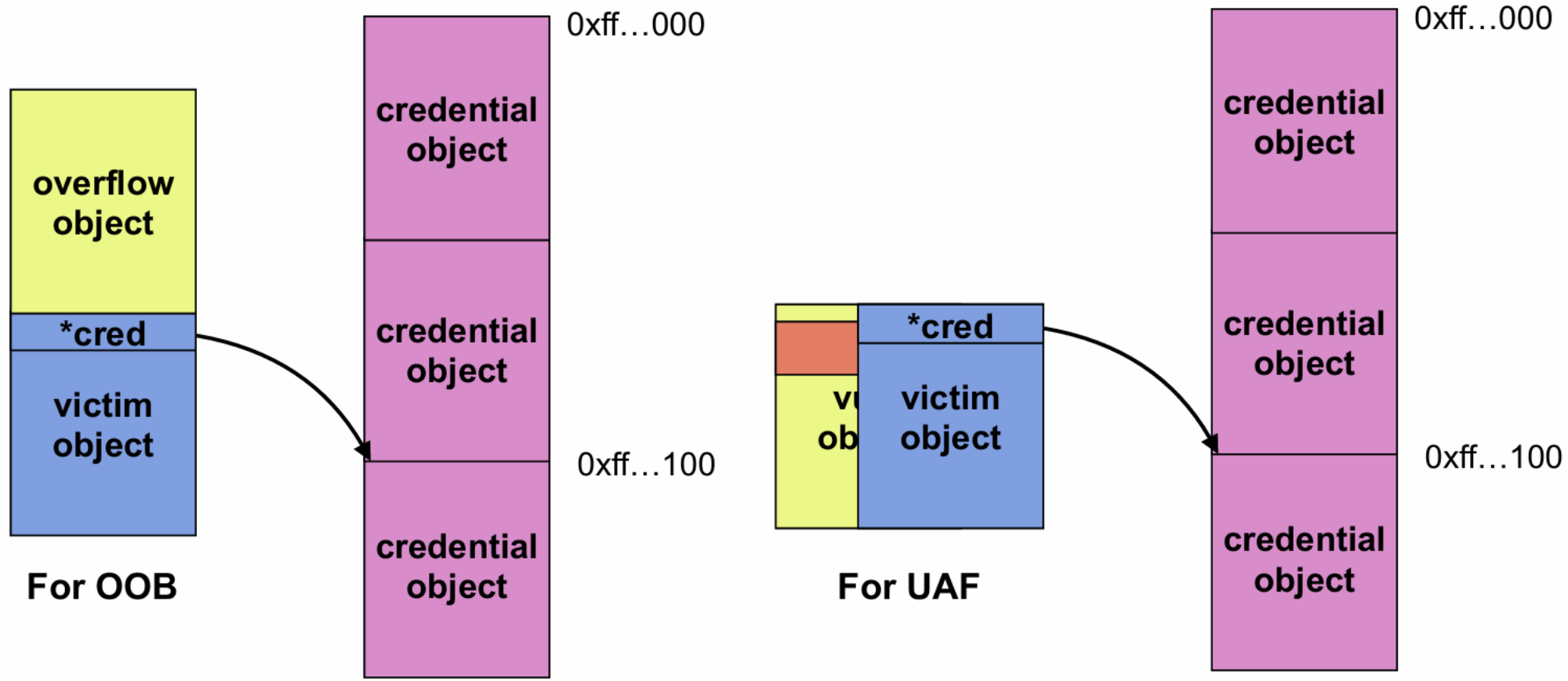
使用一个有 credentials 字段的结构体作为 victim




```
struct request_key_auth {
        struct rcu_head         rcu;
        struct key              *target_key;
        struct key              *dest_keyring;
        const struct cred       *cred;
        void                    *callout_info;
        size_t                  callout_len;
        pid_t                   pid;
        char                    op[8];
} __randomize_layout;
```

Pivoting Invalid-Write

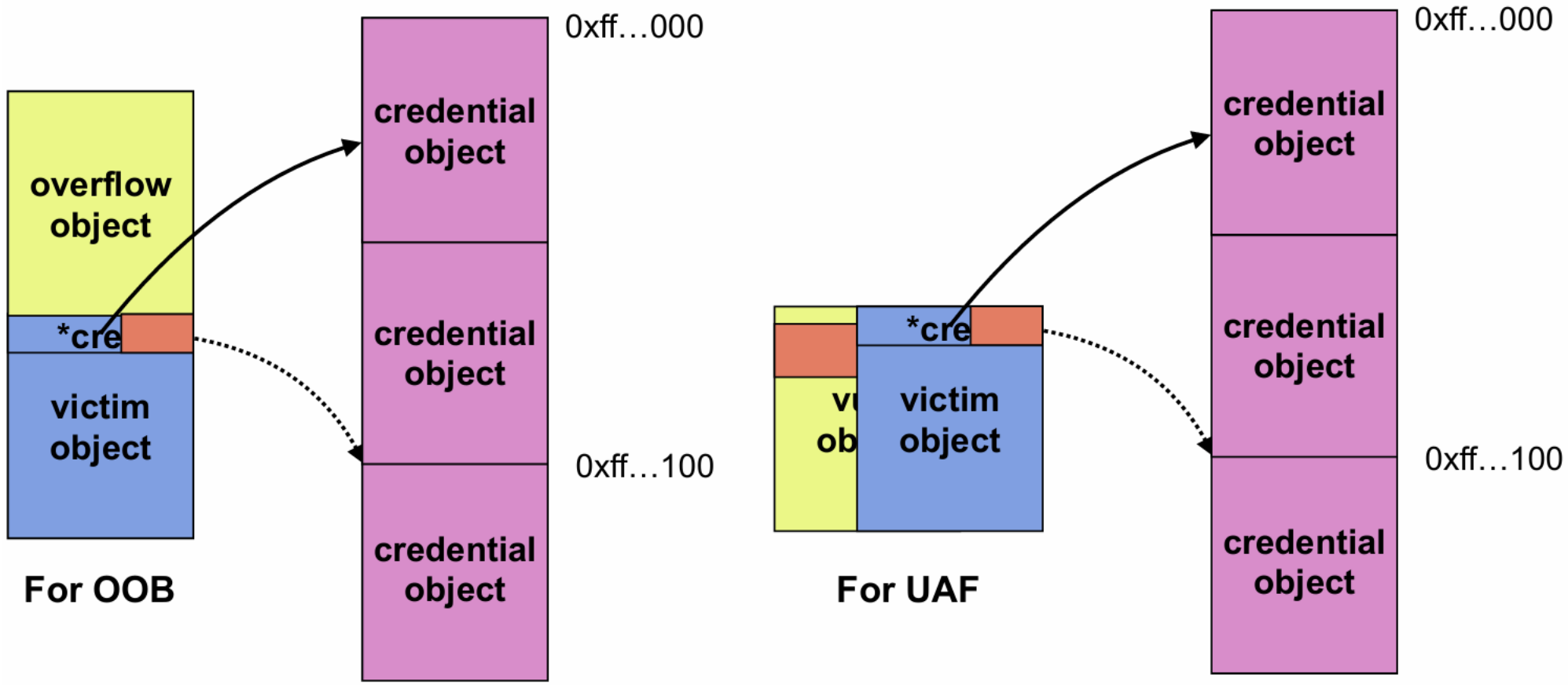
操作内存把 **struct cred*** 放置在受 overwrite 影响的区域



Pivoting Invalid-Write

北京大学 PEKING UNIVERSITY

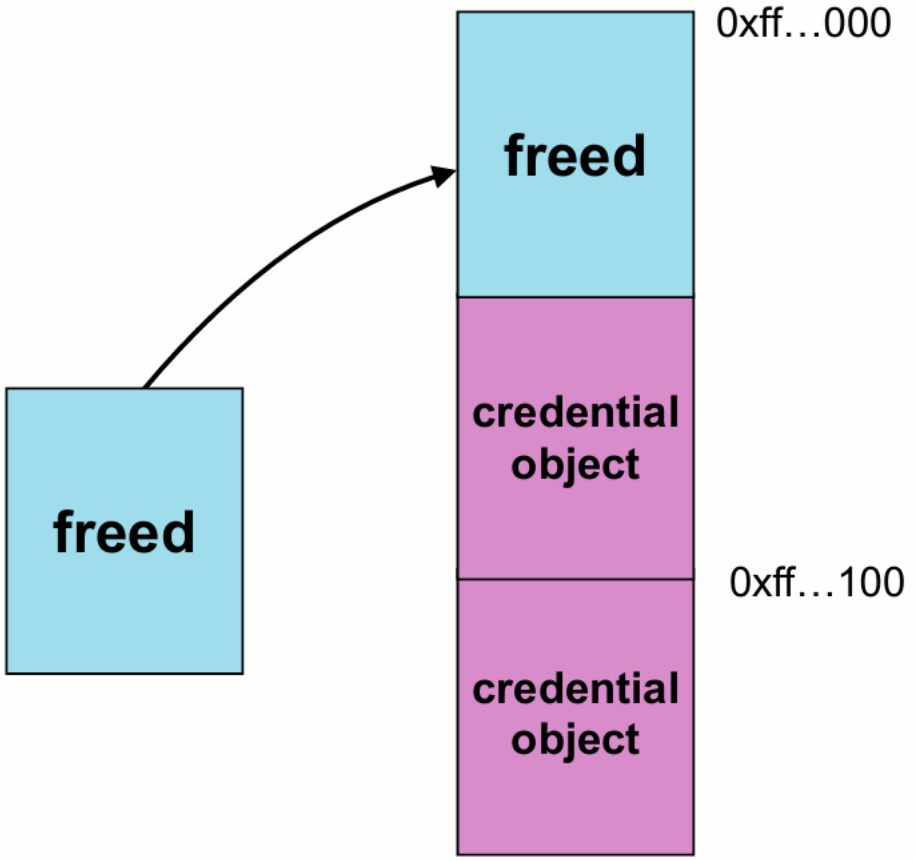
部分写覆盖 **struct cred*** 的地址值，让其指向其他 credential



Pivoting Invalid-Write

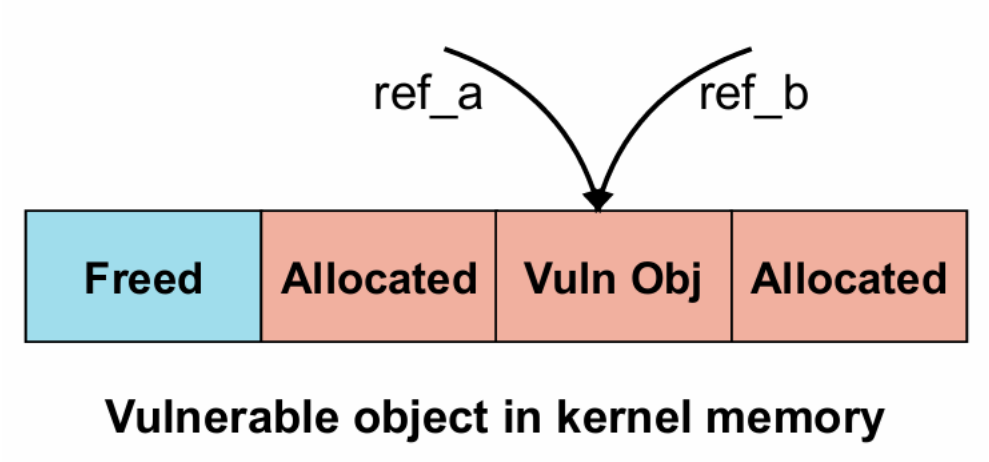
北京大学 PEKING UNIVERSITY

释放该结构体，连带 credential 也一并被释放



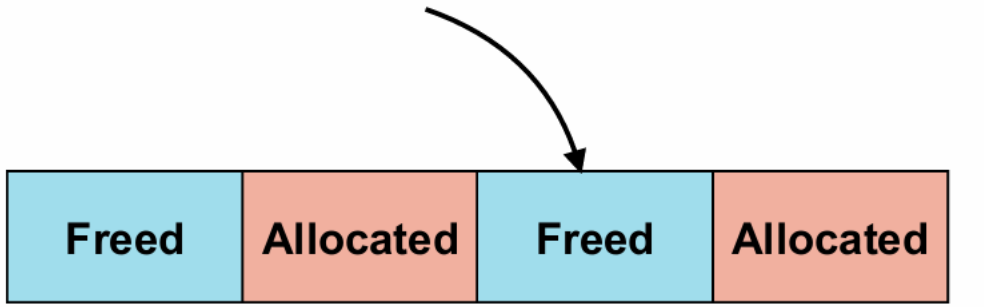
Pivoting Invalid-Free

同一个块被两个不同的指针引用



Pivoting Invalid-Free

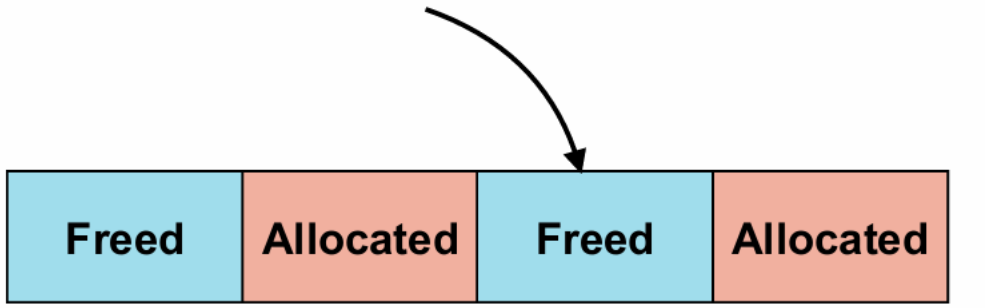
步骤一：触发漏洞，释放obj



Step 1. Trigger the vuln, free the vuln object with one reference

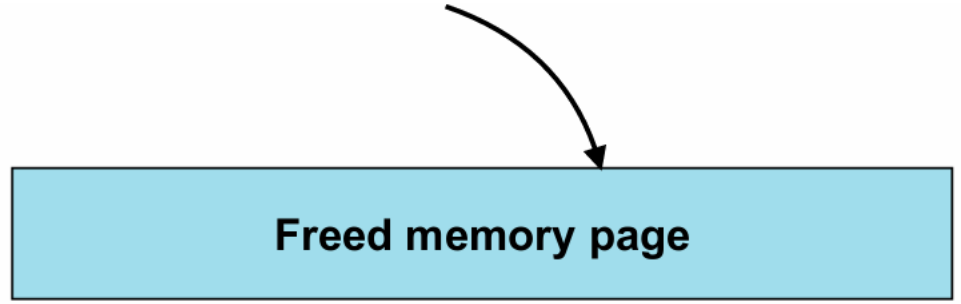
Pivoting Invalid-Free

步骤一：触发漏洞，释放obj



Step 1. Trigger the vuln, free the vuln object with one reference

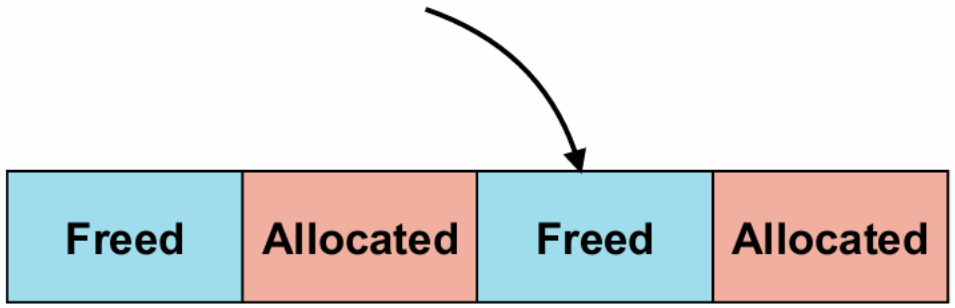
步骤二：释放整页内存



Step 2. Free the object in the memory cache to free the memory page

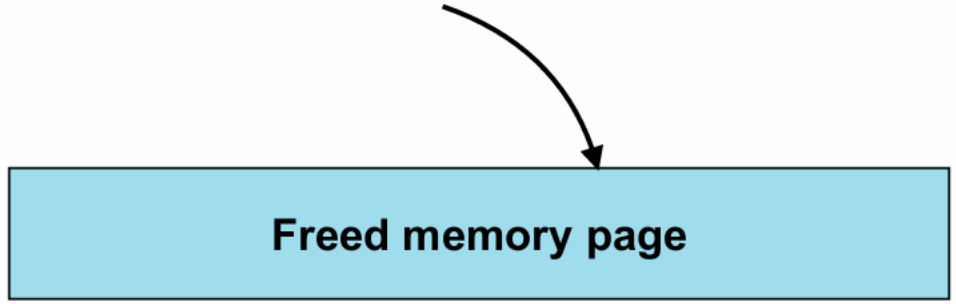
Pivoting Invalid-Free

步骤一：触发漏洞，释放obj



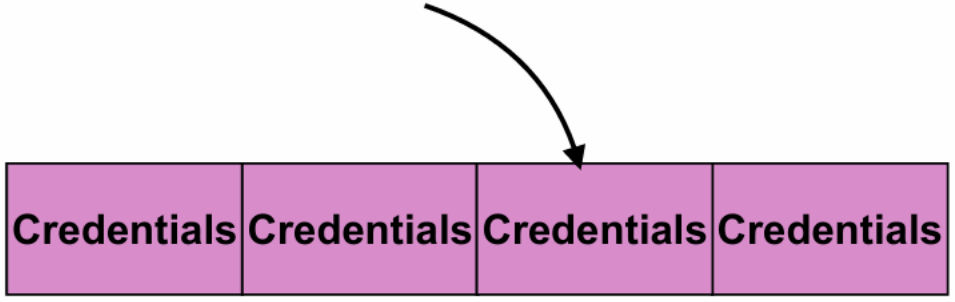
Step 1. Trigger the vuln, free the vuln object with one reference

步骤二：释放整页内存



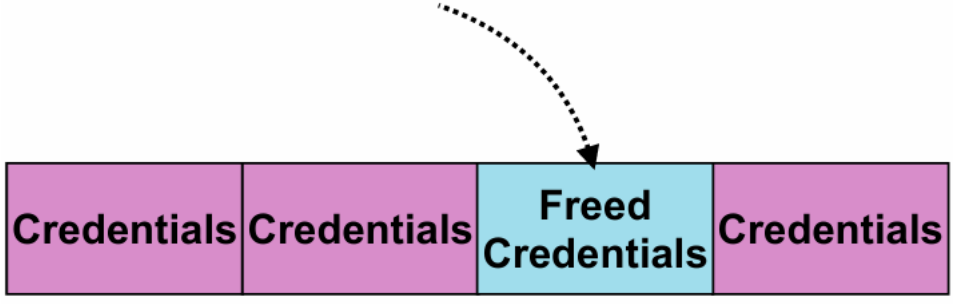
Step 2. Free the object in the memory cache to free the memory page

步骤三：申请 cred 取回释放的块



Step 3. *Allocate* credentials to reclaim the *freed* memory page (*Cross Cache Attack*)

步骤四：用另一个指针释放 cred



Step 4. Free the credentials with the left dangling reference

Challenges

1. 如何释放 credentials
2. 如何在 unprivileged 用户下申请 privilege credentials (攻击 task credentials)
3. 如何稳固文件利用方法 (攻击 open file credentials)

Challenge 2：申请 Privileged Task Credentials

北京大学 PEKING UNIVERSITY

- **Unprivileged** 用户只能申请 **Unprivileged** Task Credentials
- 等待 Privileged 用户申请 Task Credentials 会影响成功率

Challenge 2：申请 Privileged Task Credentials

北京大学 PEKING UNIVERSITY

解决办法一：触发 Privileged 用户进程

- 有 root SUID 权限的程序 (例如 su & mount)
- 在运行的守护进程 (例如 sshd)

解决办法二：触发 Privileged 内核线程

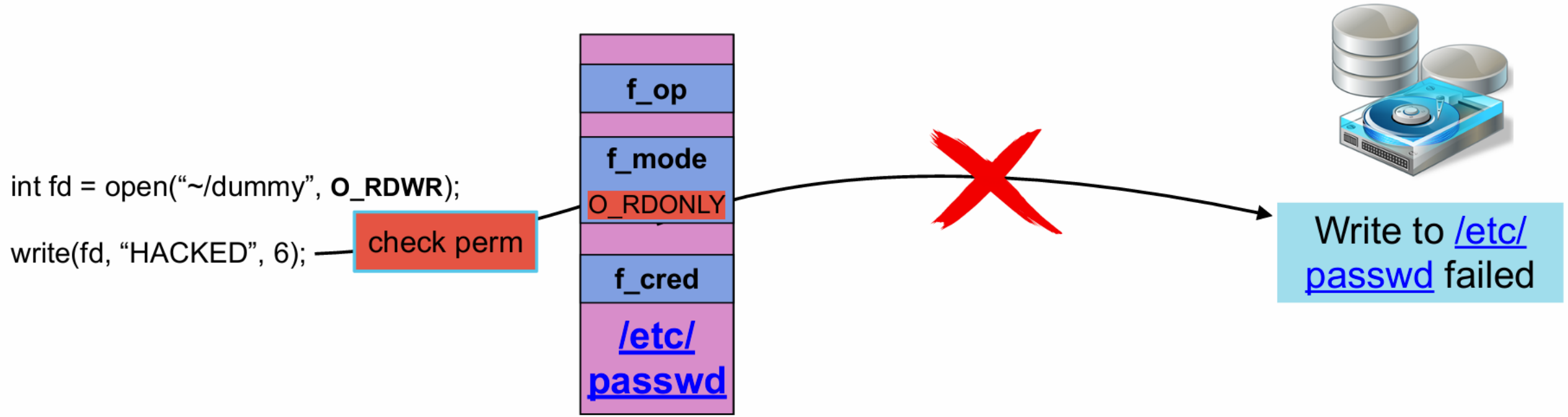
- 内核工作队列，spawn new workers
- Usermode helper，在用户态加载一些内核模块

Challenges

1. 如何释放 credentials
2. 如何在 unprivileged 用户下申请 privilege credentials (攻击 task credentials)
3. 如何稳固文件利用方法 (攻击 open file credentials)

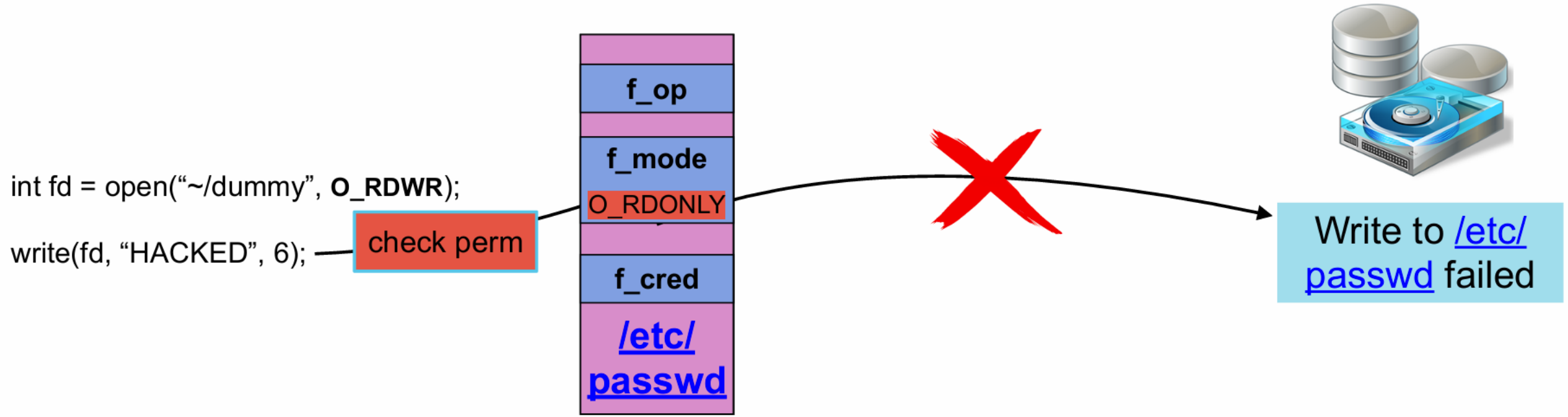
Challenge 3：Stabilizing File Exploitation

交换 file object 的操作发生在 **权限检查** 之前



Challenge 3：Stabilizing File Exploitation

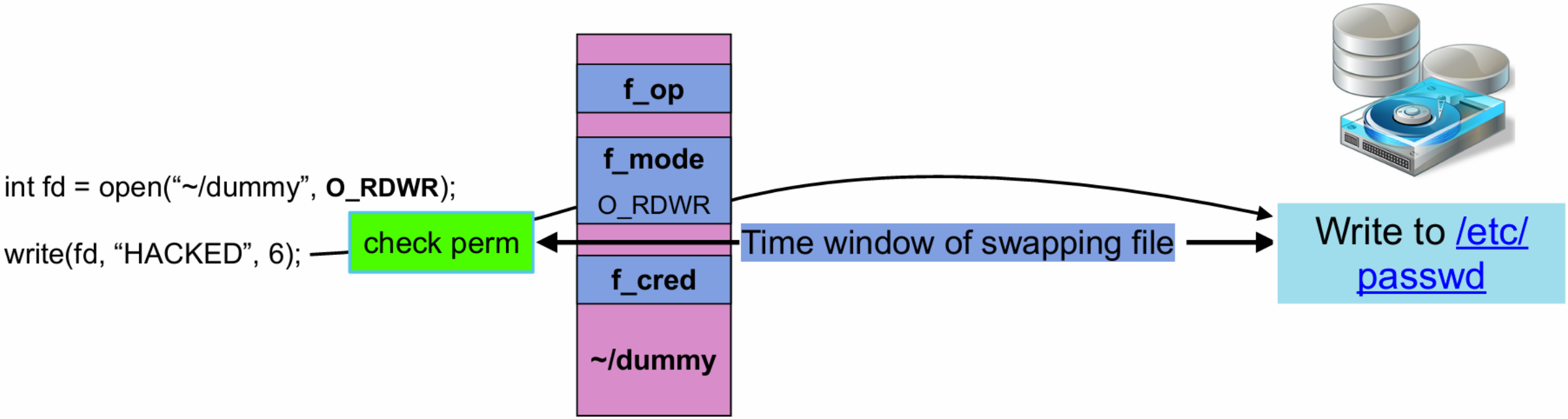
交换 file object 的操作发生在 **文件写入** 之后



Challenge 3：Stabilizing File Exploitation

交换 file object 的操作发生在 **权限检查** 之后，**文件写入** 之前

问题：操作的时间窗口很小



Challenge 3：Stabilizing File Exploitation

北京大学 PEKING UNIVERSITY

解决方法一：使用 Userfaultfd 或 FUSE

- 在访问用户空间的时候让内核暂停

解决方法一：Userfaultfd & FUSE

北京大学 PEKING UNIVERSITY

- 在 v4.13 之前，import_iovec 从用户空间拷贝数据时内核会暂停

```
ssize_t vfs_writev(...)
{
    // permission checks
    if (!(file->f_mode & FMODE_WRITE))
        return -EBADF;
    if (!(file->f_mode & FMODE_CAN_WRITE))
        return -EINVAL;

    ...
    // import iovec to kernel, where kernel would be paused
    // using userfaultfd & FUSE
    res = import_iovec(type, uvector, nr_segs,
                       ARRAY_SIZE(iovstack), &iov, &iter);
    ...
    // do file writev
}
```

解决方法一：Userfaultfd & FUSE

北京大学 PEKING UNIVERSITY

- 在 v4.13 之前，import_iovec 从用户空间拷贝数据时内核会暂停
- 在 CVE-2016-4557 中 Jann Horn 使用了这个利用方法
- 利用方法终止于 v4.13

解决方法一：Userfaultfd & FUSE

- 在 v4.13 之后使用 vfs_writev

```
ssize_t vfs_writev(...)
{
    ...
    // import iovec to kernel, where kernel would be paused
    // using userfaultfd
    res = import_iovec(type, uvector, nr_segs,
                       ARRAY_SIZE(iovstack), &iov, &iter);
    ...
    // permission checks
    if (!(file->f_mode & FMODE_WRITE))
        return -EBADF;
    if (!(file->f_mode & FMODE_CAN_WRITE))
        return -EINVAL;
    ...
    // do file writev
}
```

解决方法一：Userfaultfd & FUSE

北京大学 PEKING UNIVERSITY

- 在 generic_perform_write 处暂停
- 需要 prefaults 用户页
- 在 page fault 时暂停

```
ssize_t generic_perform_write(struct file *file,
                struct iov_iter *i, loff_t pos)
{
    /*
     * Bring in the user page that we will copy from _first_.
     * Otherwise there's a nasty deadlock on copying from the
     * same page as we're writing to, without it being marked
     * up-to-date.
     */
    if (unlikely(iov_iter_fault_in_readable(i, bytes))) {
        status = -EFAULT;
        break;
    }
    ...
    // call the write operation of the file system
    status = a_ops->write_begin(file, mapping, pos, bytes, flags,
                        &page, &fsdata);
    ...
}
```

Challenge 3：Stabilizing File Exploitation

北京大学 PEKING UNIVERSITY

解决方法一：使用 Userfaultfd 或 FUSE

- 在访问用户空间的时候让内核暂停

解决方法二：使用文件锁

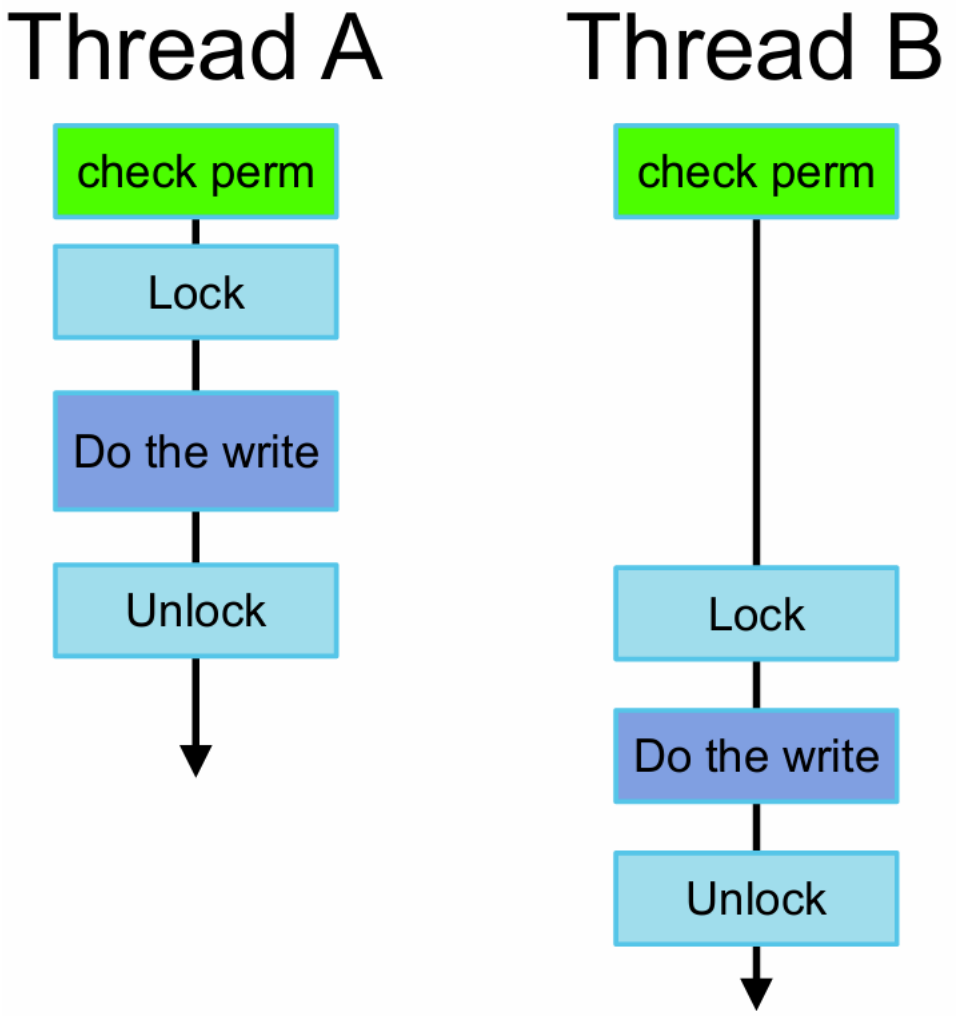
- 用文件锁暂停内核

解决方法二：File Lock

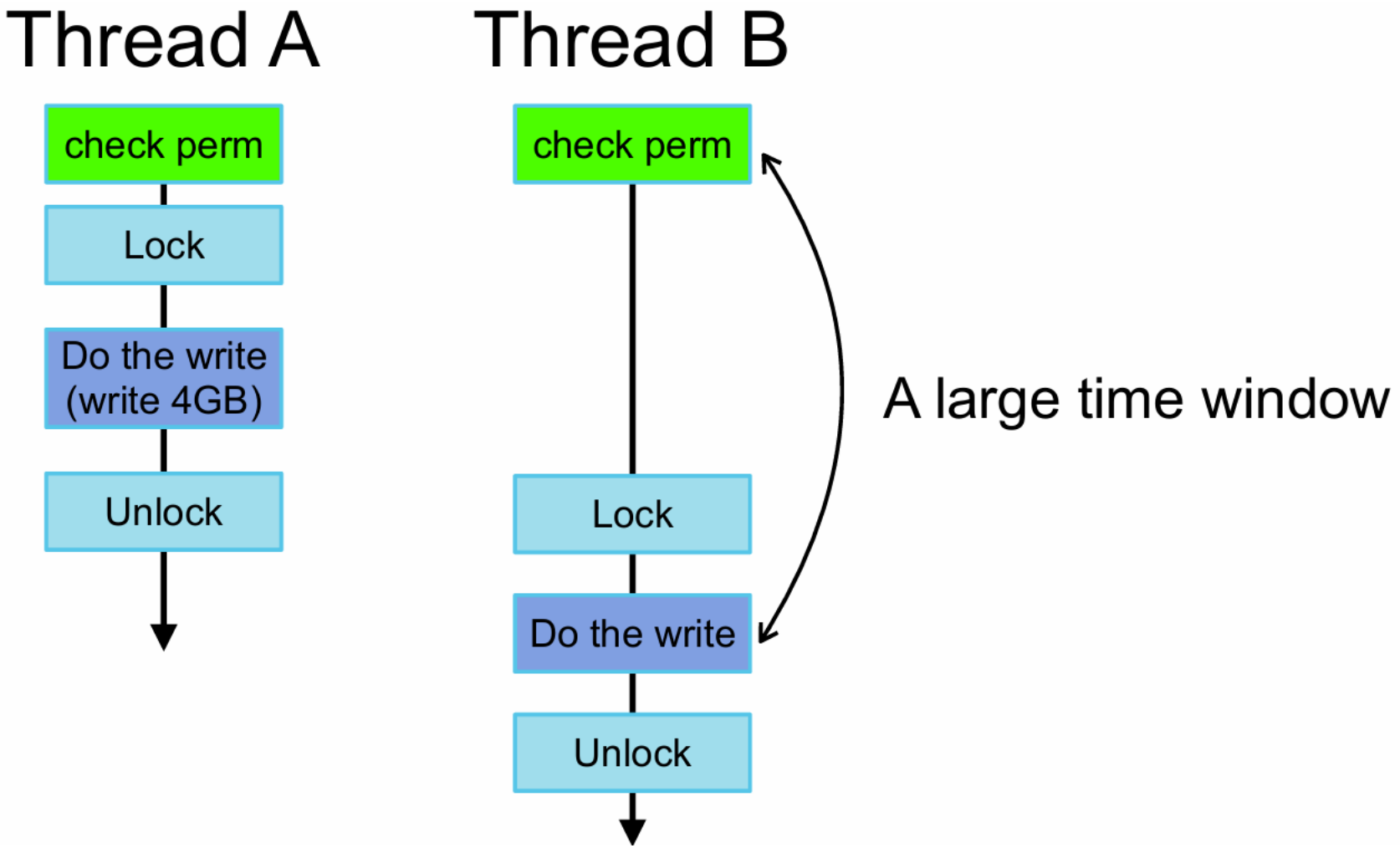
- 在文件写入时给文件上锁

```
static ssize_t ext4_buffered_write_iter(struct kiocb *iocb,
                    struct iov_iter *from)
{
    ssize_t ret;
    struct inode *inode = file_inode(iocb->ki_filp);
    inode_lock(inode);
    ...
    ret = generic_perform_write(iocb->ki_filp, from,
    ↪  iocb->ki_pos);
    ...
    inode_unlock(inode);
    return ret;
}
```

解决方法二：File Lock



解决方法二：File Lock



总结：DirtyCred 的优势

北京大学 PEKING UNIVERSITY

- 常规的攻击方法
  - 能攻击 container 和 Android
- 简单有效的
  - 不需要对抗 KASLR, CFI 等
  - Data-only method
- 攻击友好的
  - 让攻击变得更加通用
  - 可以让更多的漏洞变成类似 Dirty Pipe 的样式

如何防御 DirtyCred

北京大学 PEKING UNIVERSITY

- **核心问题**
  - 如何基于类型而不是特权做防御
- **解决思路**
  - 把 privileged credentials 做隔离
- **在哪里做隔离？**
  - 例如虚拟内存（vmalloc），没有 Cross-cache 攻击的可能

THANKS

Author：谢沅甫　　2024.12.19