1. 本实验中，`Linux`内核进行提权的两个函数分别是？为什么这两个函数可以组合起来进行提权操作?

- `Linux`内核进行提权的两个函数分别是`commit_creds()`和`prepare_kernel_cred()`，使用`commit_creds(prepare_kernel_cred(0))`即可提权。
- 函数`prepare_kernel_cred()`：
  - 原型：`struct cred *prepare_kernel_cred(struct task_struct *daemon);`。
  - 功能：`Prepare a set of credentials for a kernel service`，即为内核服务准备一个安全凭证`cred`。
  - 参数：`@daemon is used to provide a base for the security record, but can be NULL. If @daemon is supplied, then the security data will be derived from that; otherwise they'll be set to 0 and no groups, full capabilities and no keys`。即`@daemon`参数提供一个安全记录基础，也可以为空：若`@daemon`不为空，则以此为基础生成安全数据`security data`；若`@daemon`为空，则以`init_cred`为基础（`init_cred`的`uid`、`gid`等均为`root`，权限为`CAP_FULL_SET`），安全数据`security data`将被置为`0`、无组别、具有所有权限且没有`keys`，即为`root`权限。
- 函数`commit_creds()`：
  - 原型：`int commit_creds(struct cred *new);`。
  - 功能：`Install new credentials upon the current task`，即为当前任务安装新的安全凭证。
  - 参数：`@new: The credentials to be assigned`，即要赋给当前任务的安全凭证。
- 因此调用函数`commit_creds(prepare_kernel_cred(0))`，即把`prepare_kernel_cred(0)`获得的具有`root`权限的安全凭证赋给当前任务，从而完成了提权。

2. 简单介绍空指针引用漏洞的形成原因以及利用方法。

- 形成原因：在指针变量使用之前未判断该指针是否为空，导致即使指针为空时也可以进行调用。若此时调用的函数未被攻击，则大概率会因为尝试访问非法地址导致系统崩溃；若此时调用的函数被攻击者人为设置成提权函数，则此次调用会被攻击者利用，形成空指针引用漏洞。
- 利用方法：与用户空间零虚拟地址映射漏洞相结合，首先利用用户空间零虚拟地址映射漏洞，使得攻击程序可以映射到零虚拟地址（是使用具有标志位`MAP_GROWSDOWN`的`mmap`、向内存中写入数据`LD_DEBUG=help su 1>&{memfd}`和`lseek`组合实现的），然后攻击程序将`payload`从零地址开始写入，其中重要的`callback`部分置为提权函数的入口地址，然后与内存模块交互，使其指针变量被置为空，则此时正好指向攻击程序写入的`payload`，再使内存模块调用`callback`，即可时内存模块执行提权函数，使攻击程序获得`root`权限，完成漏洞利用。

3. 实验`1`中为什么要使用`alloc`、`callback`、`free`、`setxattr`、`callback`这种顺序与内核模块交互?

- 首先`alloc`，使得内核模块的`drill.item`指向新分配的一段内存，并且`drill.item->callback`被置为正常的函数`drill_callback`；
- 然后`callback`，此时执行正常的函数（这一步与攻击无关）；
- 然后`free`，此时`alloc`申请的内存被释放，使得攻击程序可以分配到这一块释放的内存，且没有对内核模块的`drill.item`进行置空，导致留有释放后使用的隐患；

- 攻击程序调用`setxattr`，其内部调用了`kvmalloc()`函数，尝试分配与之前释放掉的内存大小相等的一块内存，由于之前的`free`和内存管理的缓存机制，此时分配到的内存很大概率为刚才释放的内存，之后在分配的内存中写入`payload`，即使得函数指针指向提权函数；
- 第二次`callback`即为进行攻击尝试提权，若攻击成功，则内核模块将执行攻击方指定的提权函数`commit_creds(prepare_kernel_cred(0))`，攻击程序获得`root`权限。

4. 简单介绍释放后使用漏洞的形成原因以及利用方法。

- 形成原因：在`Linux`的内存管理中，当操作的内存满足一定条件时，`free()`操作的内存不会立即被释放，而是会进入缓存，留给后续程序继续分配和使用，因此存在分配到之前释放的内存的可能；同时，在`free()`操作后，未将指向这块内存的指针变量置为`NULL`，且能继续使用该指针变量，进而存在重新分配到该内存并对其数据进行修改再在原程序中使用未置空的指针变量进行攻击的可能。
- 利用方法：若内核模块中指向分配的内存的指针变量在`free()`后未置为`NULL`，则可以通过申请与模块中释放的内存的大小相同的内存，尝试恰好分配到之前分配给内核模块的内存（在该实验中使用的是`setxattr()`函数，其内部调用了`kvmalloc()`函数且由参数指定申请的内存大小，因此适用于尝试分配与之前释放的内存大小相同的内存，从而获得内核模块内部指针变量指向的内存地址），并向该内存写入攻击`payload`（在该实验中，`payload`的重要部分即为`drill_item_t`结构体中`callback`函数指针对应的位置，将其置为提权函数`commit_creds(prepare_kernel_cred(0))`即`root_it`的入口地址）。由于内核模块中的指针变量未置为`NULL`，此时可以再次使内核模块执行这段被两次分配的内存区域上相应位置的函数地址，而`payload`中的对应位置被改动为提权函数的入口地址，因此在内核模块调用了提权函数，从而得到了`root`权限，完成漏洞利用。

5. 实验`2`使用哪两个漏洞进行组合？为什么它们组合后才可以利用空指针引用漏洞?

- 使用了用户空间零虚拟地址映射漏洞和内核模块空指针解引用漏洞进行组合。
- 用户空间零虚拟地址映射漏洞使得`mmap`能够映射到虚拟零地址，但无法进行提权操作；而单纯依靠内核模块空指针解引用漏洞只会大概率造成系统崩溃，也无法使攻击者得到`root`权限；但二者相组合，就相当于能够在指定的零地址写入`payload`，并能够使得内存模块调用从空地址开始的一段内存，二者结合即可让内存模块执行攻击者使用`payload`中相应字节指向的提权函数，完成提权，从而完成漏洞利用。