CVE-2016-6187 Exploiting Linux kernel heap off-by-one 利用堆大小差一错误爆破 Linux 内核（上）

Background（背景）——off-by-one(大小差一)错误介绍

详细了解请移步这里 off-by-one（https://en.wikipedia.org/wiki/Off-by-one_error），大小差一错误是一类常见的程序设计错误。这方面有一个经典的例子 OpenSSH.去 Google 搜索关键词“OpenSSH off-by-one”可以了解相关状况。具体来说，

1. if(id < 0 || id > channels_alloc)...

2. if(id < 0 || id >= channels_alloc)...

第二句应该是正确的写法。举个更通俗的例子：

```
int a[5],i;
for(i = 1;i < = 5;i++)
    a[i]=0;
```

上述代码定义了长度为 5 的数组 a,循环的目的是给数组元素初始化，赋值为 0.但是,循环下标从 1 开始到 5,出现了 a[5]=0,这样的不存在的数组元素.这就是典型的“差一错误”(off-by-one).

Introduction（前言）

我认为我决定介绍该漏洞的理由是因为当我把它发在推特上时，我收到了一些私信说这个内核路径不存在漏洞（找不到漏洞在哪）或者该漏洞不能利用。另一个的理由就是我想在实际漏洞利用中尝试 `userfaultfd()` 系统调用，我需要一个真实的 UAF 漏洞来进行实验。

首先，我不知道这个漏洞影响到哪些 Linux 发行版本的内核。我检查了 ubuntu 的最新发行版本 Yakkety，发现其没有受该漏洞影响。漏洞在 https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=bb646cdb12e75d82258c2f2e7746d5952d3e321a 被引入，在 https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=30a46a4647fd1df9cf52e43bf467f0d9265096ca 中被修复。

因为我需要一个含有该漏洞的 ubuntu 内核，我在 ubuntu16.04（x86_64）上编译了 4.5.1 内核。另外该漏洞仅仅影响像 ubuntu 一样默认使用 AppArmor 作为 LSM（Linux 安全模块）的发行版本，像 centos 使用 SELinux 就不受该漏洞的影响。

Vulnerability（漏洞介绍）

当写入`/proc/self/attr/current` 会使用内核中的 `proc_pid_attr_write()` 函数。接下来介绍的代码是漏洞被引入之前的：

```
static ssize_t proc_pid_attr_write(struct file * file, const char __user * b
uf,
                                   size_t count, loff_t *ppos)
{
        struct inode * inode = file_inode(file);
        char *page;
        ssize_t length;
        struct task_struct *task = get_proc_task(inode);


        length = -ESRCH;
        if (!task)
                goto out_no_task;
        if (count > PAGE_SIZE)                          [1]
                count = PAGE_SIZE;


        /* No partial writes. */
        length = -EINVAL;
        if (*ppos != 0)
                goto out;


        length = -ENOMEM;
        page = (char*)__get_free_page(GFP_TEMPORARY);     [2]
        if (!page)
```

```
                goto out;


        length = -EFAULT;
        if (copy_from_user(page, buf, count))            [3]
                goto out_free;


        /* Guard against adverse ptrace interaction */
        length = mutex_lock_interruptible(&task-;>signal->cred_guard_mutex);
        if (length < 0)
                goto out_free;


        length = security_setprocattr(task,
                                      (char*)file->f_path.dentry->d_name.name,
                                      (void*)page, count);


...
```

buf 参数代表用户提供的缓冲区（和长度 count）被写入到`/proc/self/attr/current` 。在[1]处执行检查以确保用户缓冲区最多只会写入 one page（默认 4096 个字节）。[2]和[3]中先在内核空间分配 one page 然后将用户空间缓冲区复制到新分配的页。该页将被传递给内核安全模块（例如 AppArmour,SELinux,Smack）的`security_setprocattr 函数` 。如果是 ubuntu 将触发`apparmor_setprocattr()` 函数，代码显示如下：

```
static int apparmor_setprocattr(struct task_struct *task, char *name,
                                void *value, size_t size)
{
        struct common_audit_data sa;
        struct apparmor_audit_data aad = {0,};
        char *command, *args = value;
        size_t arg_size;
        int error;
```

```
        if (size == 0)
                return -EINVAL;
        /* args points to a PAGE_SIZE buffer, AppArmor requires that
         * the buffer must be null terminated or have size <= PAGE_SIZE -1
         * so that AppArmor can null terminate them
         */
        if (args[size - 1] != '\0') {                   [4]
                if (size == PAGE_SIZE)
                        return -EINVAL;
                args[size] = '\0';
        }
...
```

在[4]处，如果用户提供的缓冲区最后的字节不为空并且缓冲区大小不等于页的大小时，在缓冲区结尾添加一位字符串结束符；如果用户提供的缓冲区超过（或者等于）单页的大小（在[2]处分配的），直接返回错误。

接下来显示的代码为引入该漏洞的（因为更改了[3]处的代码）`proc_pid_attr_write()`：

```
static ssize_t proc_pid_attr_write(struct file * file, const char __user * b
uf,
                                   size_t count, loff_t *ppos)
{
        struct inode * inode = file_inode(file);
        void *page;
        ssize_t length;
        struct task_struct *task = get_proc_task(inode);

        length = -ESRCH;
        if (!task)
                goto out_no_task;
```

```
        if (count > PAGE_SIZE)
                count = PAGE_SIZE;

        /* No partial writes. */
        length = -EINVAL;
        if (*ppos != 0)
                goto out;

        page = memdup_user(buf, count);                 [5]
        if (IS_ERR(page)) {
                length = PTR_ERR(page);
                goto out;
        }

        /* Guard against adverse ptrace interaction */
        length = mutex_lock_interruptible(&task-;>signal->cred_guard_mutex);
        if (length < 0)
                goto out_free;

        length = security_setprocattr(task,
                                      (char*)file->f_path.dentry->d_name.name,
                                      page, count);
...
```

不像 `__get_free_page()`, `memdup_user()` 分配了一块被 count 参数指定大小的内存并且复制用户提供的数据到其中。因此，内核内存被分配的大小不再是严格的 4096 个字节（最大也可能是 4096 个字节）。现在假设用户提供的数据是 128 个字节大小并且缓冲区的最后字节不为空。当 `apparmor_setprocattr()` 被触发，按照下面的代码逻辑 `args[128]` 将被设置为 0：

```
   if (args[size - 1] != '\0') {
```

```
            if (size == PAGE_SIZE)
                    return -EINVAL;
            args[size] = '\0';
    }
```

因为 args 的内存在堆中被动态分配，args 的内存的下一个字节会被重写为 NULL。因为该漏洞只能重写一个字节大小的内核内存，传统的重写函数指针的漏洞利用方法在这里并不适用。一个可行的方案是在一些对象上（像 args 对象同样大小的）重写一个引用计数器并且然后触发一个 UAF（谢谢 Nicolas Tripar 的建议）。如果你下一周将会在 Ruxcon，尽管在计数器溢出的主题，在内核中利用计数器溢出检验我的演讲吧。对象引用计数器（`atomic_t` type = signed int 所代表）通常是结构的第一个成员。因为计数器的值通常对于大多数对象都是在 255 以下的，重写像一个对象最低有效的字节会清除计数器并且导致一个标准的 UAF。然而为了利用这个漏洞，我决定用一个不同的方法：重写 SLUB freelist 指针。

未完待续……

译者：rodster
校对：song
原文链接：https://cyseclabs.com/blog/cve-2016-6187-heap-off-by-one-exploit
原文作者：Vitaly Nikolenko