Linux内核通用堆喷射技术详解

mss**** / 2018-09-28 09:46:14 / 浏览数 3255

原文：https://cyseclabs.com/blog/linux-kernel-heap-spray

简介

这个内核堆喷射技术曾经在beVX研讨会期间进行过相应的演示，之后，还曾被用于内核IrDA子系统（Ubuntu 16.04）的0day攻击。与已知的堆喷射不同，该技术适用于非常小的对象（大小不超过8或16字节）或需要控制前N个字节的对象（即目标对象中没有不受控制的头部）。这

话虽这么说，堆喷射（分配多个对象来填充内存块）绝对适用于其他与堆相关的漏洞，如内核堆溢出。

目前，对于已经公开的各种堆喷射，例如add_key()，send[m]msg()和msgsnd()等，还存在许多错误的理解。所有这些喷射技术都存在对象大小限制或无法用用户空间数据

以下用户空间代码将触发如下所示的do_msgsnd执行路径。

```
#define BUFF_SIZE 96-48

struct {
   long mtype;
   char mtext[BUFF_SIZE];
} msg;

memset(msg.mtext, 0x42, BUFF_SIZE-1);
msg.mtext[BUFF_SIZE] = 0;
msg.mtype = 1;

int msqid = msgget(IPC_PRIVATE, 0644 | IPC_CREAT);

msgsnd(msqid, &msg, sizeof(msg.mtext), 0);
```

首先，在[1]处的load_msg()中将分配96字节的对象（48字节是struct msg_msg + 48字节用于消息正文）：

```
long do_msgsnd(int msqid, long mtype, void __user *mtext,
               size_t msgsz, int msgflg)
{
       struct msg_queue *msq;
       struct msg_msg *msg;
       int err;
       struct ipc_namespace *ns;

       ns = current->nsproxy->ipc_ns;

       if (msgsz > ns->msg_ctlmax || (long) msgsz < 0 || msqid < 0)
               return -EINVAL;
       if (mtype < 1)
               return -EINVAL;

[1]     msg = load_msg(mtext, msgsz);
...
```

然后，将用户空间缓冲区内容复制到新分配的内核缓冲区中：

```
struct msg_msg *load_msg(const void __user *src, size_t len)
{
       struct msg_msg *msg;
       struct msg_msgseg *seg;
       int err = -EFAULT;
       size_t alen;

       msg = alloc_msg(len);
       if (msg == NULL)
               return ERR_PTR(-ENOMEM);

       alen = min(len, DATALEN_MSG);
[2]     if (copy_from_user(msg + 1, src, alen))
               goto out_err;
```

```
        for (seg = msg->next; seg != NULL; seg = seg->next) {
                len -= alen;
                src = (char __user *)src + alen;
                alen = min(len, DATALEN_SEG);
                if (copy_from_user(seg + 1, src, alen))
                        goto out_err;
        }
...
```

alloc_msg()函数的实现代码如下所示。其中，消息的最大长度为DATALEN_MSG（4048字节）。如果消息长度大于DATALEN_MSG，则该消息被认为是由多段组成的，并

```
static struct msg_msg *alloc_msg(size_t len)
{
        struct msg_msg *msg;
        struct msg_msgseg **pseg;
        size_t alen;

[3]      alen = min(len, DATALEN_MSG);
        msg = kmalloc(sizeof(*msg) + alen, GFP_KERNEL);
        if (msg == NULL)
                return NULL;

        msg->next = NULL;
        msg->security = NULL;

        len -= alen;
        pseg = &msg->next;
        while (len > 0) {
                struct msg_msgseg *seg;
                alen = min(len, DATALEN_SEG);
[4]              seg = kmalloc(sizeof(*seg) + alen, GFP_KERNEL);
                if (seg == NULL)
                        goto out_err;
                *pseg = seg;
                seg->next = NULL;
                pseg = &seg->next;
                len -= alen;
        }
...
```

考虑到总是有一个长度为48字节的不受控制的头部，因此，对于kmalloc-8、16、32高速缓存中的目标对象，或者需要控制前48个字节的任何其他目标对象（例如，函数指

理想情况下，通用堆喷射应满足以下条件：

1. 对象大小由用户控制。即使对于非常小的对象（例如，kmalloc-8）也没有限制。
2. 对象内容由用户控制。在对象的开头部分没有不受控制的头部。
3. 在漏洞利用阶段，目标对象应该"呆"在内核中。这一点对复杂的UAF和竞争条件漏洞利用来说特别有用。

上面展示的msgsnd喷射技术仅满足上表中的第3个条件。这个代码路径的好处是，分配的对象会"呆"在内核中，直到进程终止或调用msgrcv函数从队列中弹出其中一条消息

在内核中，还有几条满足前两个条件的路径，但是却不满足第三个条件。根据定义，这些通常都被认为是喷射技术，因为它们用受控的用户数据来喷射一段内存空间。唯一的

userfaultfd+setxattr通用堆喷射技术

为了实现通用的堆喷射技术，一般方法采用kmalloc->kfree执行路径之一(满足条件1和2)，并将其与userfaultfd结合使用，以满足第3个条件。

userfaultfd有一个详细的手册页，不仅介绍了其用法，同时还提供了关于如何设置页面错误处理程序线程的示例。其基本思想是，处理用户空间中的页面错误。例如，在用户
= mmap(0, 0x1000, ...,
MAP_ANON|...)，如果立即对testptr页面进行读或写操作的话，就会触发内核空间中的页面错误处理程序。使用userfaultfd，可以通过单个线程在用户空间中处理/解决这些

接下来，我们的任务是寻找满足上述条件1和2的内核执行路径。例如，setxattr()系统调用具有以下kmalloc->kfree路径：

```
static long
setxattr(struct dentry *d, const char __user *name, const void __user *value,
         size_t size, int flags)
{
        int error;
        void *kvalue = NULL;
        void *vvalue = NULL;    /* If non-NULL, we used vmalloc() */
```

```
        char kname[XATTR_NAME_MAX + 1];

        if (flags & ~(XATTR_CREATE|XATTR_REPLACE))
                return -EINVAL;

        error = strncpy_from_user(kname, name, sizeof(kname));
        if (error == 0 || error == sizeof(kname))
                error = -ERANGE;
        if (error < 0)
                return error;

        if (size) {
                if (size > XATTR_SIZE_MAX)
                        return -E2BIG;
[5]              kvalue = kmalloc(size, GFP_KERNEL | __GFP_NOWARN);
                if (!kvalue) {
                        vvalue = vmalloc(size);
                        if (!vvalue)
                                return -ENOMEM;
                        kvalue = vvalue;
                }
[6]              if (copy_from_user(kvalue, value, size)) {
                        error = -EFAULT;
                        goto out;
                }
                if ((strcmp(kname, XATTR_NAME_POSIX_ACL_ACCESS) == 0) ||
                    (strcmp(kname, XATTR_NAME_POSIX_ACL_DEFAULT) == 0))
                        posix_acl_fix_xattr_from_user(kvalue, size);
        }

        error = vfs_setxattr(d, kname, kvalue, size, flags);
out:
        if (vvalue)
                vfree(vvalue);
        else
[7]              kfree(kvalue);
        return error;
}
```

首先，为一个对象分配内存空间（大小由用户控制，具体参加第[5]行代码），然后在第[6]行代码中，将用户空间数据复制到分配的对象内存空间中。该路径满足我们的堆喷

我们的思路是，将该执行路径与userfaultfd用户空间缓冲区/页面结合起来，以便在执行流程达到第[5]行时，在用户空间线程中处理相应的页面错误。然后，可以让该线程无

```
struct target {
   void (*fn)();
   unsigned long something;
};
```

首先，在用户空间中分配两个相邻内存页面，并将目标对象放在这两个页面的边界上；其中，前8个字节（即函数指针）放在第一页中，其余字节放在第二页中。然后，使用

堆喷射

当setxattr()中的内核执行路径变为第[6]行时，函数指针（8字节）将被复制到kvalue中，但是任何复制剩余字节的尝试，都会将执行权限转移到我们的用户空间线程，进而

小结

在研讨会期间，我们通过这种技术成功攻击了最近的DCCP
UAF漏洞（目标对象大小为16字节，其中前8个字节为函数指针）和IrDA子系统中的另一个0day漏洞（对于该易受攻击的对象，长度为56个字节，其中前16个字节为目标指

上一篇：VPNFilter更新，加入7个新... 下一篇：通用Cereal总线上的OATme...

1. 0 条回复

- 动动手指，沙发就是你的了！

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板