旁注　函数 getpeername 的安全漏洞

2002 年，从事 FreeBSD 开源操作系统项目的程序员意识到，他们对 getpeername 函数的实现存在安全漏洞。代码的简化版本如下：

```
1  /*
2   * Illustration of code vulnerability similar to that found in
3   * FreeBSD's implementation of getpeername()
4   */
5
6  /* Declaration of library function memcpy */
7  void *memcpy(void *dest, void *src, size_t n);
8
9  /* Kernel memory region holding user-accessible data */
10 #define KSIZE 1024
11 char kbuf[KSIZE];
12
13 /* Copy at most maxlen bytes from kernel region to user buffer */
14 int copy_from_kernel(void *user_dest, int maxlen) {
15     /* Byte count len is minimum of buffer size and maxlen */
16     int len = KSIZE < maxlen ? KSIZE : maxlen;
17     memcpy(user_dest, kbuf, len);
18     return len;
19 }
```

在这段代码里，第 7 行给出的是库函数 memcpy 的原型，这个函数是要将一段指定长度为 n 的字节从内存的一个区域复制到另一个区域。

从第 14 行开始的函数 copy_from_kernel 是要将一些操作系统内核维护的数据复制到指定的用户可以访问的内存区域。对用户来说，大多数内核维护的数据结构应该是不可读的，因为这些数据结构可能包含其他用户和系统上运行的其他作业的敏感信息，但是显示为 kbuf 的区域是用户可以读的。参数 maxlen 给出的是分配给用户的缓冲区的长度，这个缓冲区是用参数 user_dest 指示的。然后，第 16 行的计算确保复制的字节数据不会超出源或者目标缓冲区可用的范围。

不过，假设有些怀有恶意的程序员在调用 copy_from_kernel 的代码中对 maxlen 使用了负数值，那么，第 16 行的最小值计算会把这个值赋给 len，然后 len 会作为参数 n 被传递给 memcpy。不过，请注意参数 n 是被声明为数据类型 size_t 的。这个数据类型是在库文件 stdio.h 中(通过 typedef)被声明的。典型地，对 32 位程序它被定义为 unsigned int，对 64 位程序定义为 unsigned long。既然参数 n 是无符号的，那么 memcpy 会把它当作一个非常大的正整数，并且试图将这样多字节的数据从内核区域复制到用户的缓冲区。虽然复制这么多字节(至少 2^{31} 个)实际上不会完成，因为程序会遇到进程中非法地址的错误，但是程序还是能读到它没有被授权的内核内存区域。

我们可以看到，这个问题是由于数据类型的不匹配造成的：在一个地方，长度参数是有符号数；而另一个地方，它又是无符号数。正如这个例子表明的那样，这样的不匹配会成为缺陷的原因，甚至会导致安全漏洞。幸运的是，还没有案例报告有程序员在 FreeBSD 上利用了这个漏洞。他们发布了一个安全建议，“FreeBSD-SA-02:38.signed-error”，建议系统管理员如何应用补丁消除这个漏洞。要修正这个缺陷，只要将 copy_from_kernel 的参数 maxlen 声明为类型 size_t，也就是与 memcpy 的参数 n 一致。同时，我们也应该将本地变量 len 和返回值声明为 size_t。