端口复用 :隐藏 嗅探与攻击

2012-08-23 08:50:03 By admin

前言

在WINDOWS的SOCKET服务器应用的编程中，如下的语句或许比比都是：

```
s=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = htonl(INADDR_ANY);
bind(s,(SOCKADDR *)&saddr,sizeof(saddr));
```

其实这当中存在在非常大的安全隐患，因为在winsock的实现中，对于服务器的绑定是可以多重绑定的，在确定多重绑定使用谁的时候，根据一条原则是
谁的指定最明确则将包递交给谁，而且没有权限之分
，也就是说低级权限的用户是可以重绑定在高级权限如服务启动的端口上的,这是非常重大的一个安全隐患。
在系统已开放的端口上进行通讯，只对输入的信息进行字符匹配，不对网络数据进行任何拦截、复制类操作，所以对网络数据的传输性能丝毫不受影响。
建立连接后服务端程序占用极少系统资源，被控端不会在系统性能上有任何察觉，通常被后门木马所利用。

攻击

这意味着什么？意味着可以进行如下的攻击：

1. 一个木马绑定到一个已经合法存在的端口上进行端口的隐藏，他通过自己特定的包格式判断是不是自己的包，如果是自己处理，如果不是通过127.0.0.1的地址交给真正的服务器应用进行处理。
2. 一个木马可以在低权限用户上绑定高权限的服务应用的端口，进行该处理信息的嗅探，本来在一个主机上监听一个SOCKET的通讯需要具备非常高的权限要求，但其实利用SOCKET重绑定，你可以轻易的监听具备这种SOCKET编程漏洞的通讯，而无须采用什么挂接，钩子或低层的驱动技术（这些都需要具备管理员权限才能达到）
3. 针对一些的特殊应用，可以发起中间人攻击，从低权限用户上获得信息或事实欺骗，如在guest权限下拦截telnet服务器的 23端口，如果是采用NTLM加密认证，虽然你无法通过嗅探直接获取密码，但一旦有admin用户通过你登陆以后，你的应用就完全可以发起中间人攻击，扮演这个登陆的用户通过SOCKET发送高权限的命令，到达入侵的目的。

其实，MS自己的很多服务的SOCKET编程都存在这样的问题， telnet,ftp,http的服务实现全部都可以利用这种方法进行攻击，在低权限用户上实现对SYSTEM应用的截听。包括W2K+SP3的IIS也都一样，那么如果你已经可以以低权限用户入侵或木马植入的话，而且对方又开启了这些服务的话，那就不妨一试。并且我估计还有很多第三方的服务也大多存在这个漏洞。

例子

解决的方法很简单，在编写如上应用的时候，绑定前需要使用setsockopt指定SO_EXCLUSIVEADDRUSE要求独占所有的端口地址，而不允许复用。这样其他人就无法复用这个端口了。
下面就是一个简单的截听ms telnet服务器的例子，在GUEST用户下都能成功进行截听，剩余的就是大家根据自己的需要，进行一些特殊剪裁的问题了 :如是隐藏，嗅探数据，高权限用户欺骗等。

C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <strings.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
//远程服务器端口和IP
#define PORT 1987
#define HOST "192.168.197.118"
//本机服务器端口和IP
#define LOCALPORT 2012
#define LOCALHOST "10.0.2.15"

//线程提供TCP服务器服务，绑定2012端口
void serve_proc(int port)
{
	printf("server start listening on port %d\n",port);
	int sock_fd;

	if ((sock_fd=socket(AF_INET, SOCK_STREAM, 0))==-1){
		printf("socket() error\n");
		exit(1);
	}

	struct sockaddr_in server, client;
	bzero(&server,sizeof(server));
	server.sin_family = AF_INET;
	server.sin_port = htons(port);
	server.sin_addr.s_addr = inet_addr(LOCALHOST);
	int reuseport = 1;

	//端口设置重用，不然会bind出错
	setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR, (const char *)&reuseport, sizeof(int));
	if (bind(sock_fd, (struct sockaddr *)&server, sizeof(struct sockaddr_in)) < 0){
		perror("bind error");
		exit(-1);
	}
	if (listen(sock_fd,5) < 0) {
```

```
        perror("listen error");
        exit(-1);
    }

    char recv_buf[100],send_buf[100];
    int conn_fd;
    int client_size = sizeof(struct sockaddr_in);
    while(1){
        conn_fd = accept(sock_fd, (struct sockaddr *)&client, &client_size);
        if (conn_fd < 0){
            perror("accept error");
            exit(-1);
        }
        while(1){
            int recv_num = recv(conn_fd, recv_buf, sizeof(recv_buf), 0);
            if(recv_num < 0){
                close(conn_fd);exit(-1);
            }
            recv_buf[recv_num] = '\0';
            if(strcmp(recv_buf,"quit")==0){
                break;
            }
            printf("server get %s\n", recv_buf);
            sprintf(send_buf, "server got %d bytes\n", recv_num);
            int send_num = send(conn_fd, send_buf, strlen(send_buf), 0);
            if(send_num < 0){
                close(conn_fd);exit(-1);
            }
        }
        close(conn_fd);
    }
}

int main(int argc, char *argv[])
{
    int sock_fd;
    struct sockaddr_in server,client;
    int flag = 0;
    int reuseport = 1;
    int local_port = LOCALPORT;

    if ((sock_fd=socket(AF_INET, SOCK_STREAM, 0))==-1){
        printf("socket() error\n");
        exit(1);
    }
    setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR, (const char *)&reuseport, sizeof(int));
    //连接其他服务器并发出数据
    bzero(&server,sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);
```

```
        server.sin_addr.s_addr = inet_addr(HOST);

        //使用本地指定端口建立TCP连接
        struct hostent *he;
        he = gethostbyname(LOCALHOST);
        bzero(&client,sizeof(client));
        client.sin_family = AF_INET;
        client.sin_port = htons(local_port);
        client.sin_addr = *((struct in_addr *)he->  h_addr);;
        setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR, (const char *)&reuseport, sizeof(int));
        if (bind(sock_fd, (struct sockaddr *)&client, sizeof(struct sockaddr_in)) < 0){
                perror("bind error");
                exit(-1);
        }
        if (connect(sock_fd,(struct sockaddr *)&server, sizeof(struct sockaddr)) < 0){
                perror("connect error");
                exit(1);
        }

        struct sockaddr_in local;
        socklen_t local_len = sizeof(local);
        //获取TCP连接的本地IP端口信息
        getsockname(sock_fd, (struct sockaddr *)&local, &local_len);
        char local_ip[100];
        inet_ntop(AF_INET, &local.sin_addr, local_ip, sizeof(local_ip));
        printf("local host %s and port %d\n", local_ip, ntohs(local.sin_port));
        local_port = ntohs(local.sin_port);

        //功能一 :绑定2012端口提供TCP服务
        pthread_t id;
        int ret = pthread_create(&id, NULL, (void *)serve_proc, (void *)local_port);
        if (ret!=0) {
                perror("create thread error");
                exit(-1);
        }

        int send_num, recv_num;
        char send_buf[100],recv_buf[100];

LBL:
        //功能二 :使用2012端口充当客户端访问远程TCP服务器
        printf("Input MSG: ");
        scanf("%s",send_buf);
        send_num = send(sock_fd, send_buf, strlen(send_buf), 0);
        if (send_num < 0) {
                perror("send error");
                exit(1);
        }
        goto LBL;
}
```

假如端口被socket使用过，并且利用socket.close()来关闭连接，但此时端口还没有释放，要经过一个TIME_WAIT的过程之后才能使用。为了实现端口的马上复用，可以选择setsockopt()函数来达到目的。

Python

```
import socket

tcp1=socket.socket(socket.AF_INET,socket.SOCK_STREAM)

tcp1.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)

tcp1.bind('1.1.1.1',12345)
```

此为tcp的例子，udp一样

关于错误

写Socket程序的时候经常会遇到这个问题 :如果自己的程序不小心崩溃了，重新启动程序的时候往往会在bind调用上失败，错误原因为Address Already In Use，往往要等待两分钟才能再次绑定。但是在很多的程序（比如nginx）中好像并不存在这个问题，就算被KILL了也能立刻重启。这个区别还是比较头痛的。

其实我猜Unix Socket编程这样的书上有讨论过这方面的问题，不过我竟然没有这方面的书籍（完全靠man看来也是行不通啊）。我曾经天真的以为，在收到SIGTERM这样的信号的时候把所有套接字全部关闭可以解决问题。后来才发现无济于事。Google了这方面的文章才知道，解决这个问题理论上有三种办法。

    - 1.SOCK_REUSEADDR:曾经在研究内网穿透的时候跟这个东西打过交道。如果一个监听的套接字被设置为允许地址复用，那么套接字绑定到的地址不会被独占，所以必然不会存在Address already in use的问题。而且如果有两个套接字绑定到同一个端口，也只允许一个套接字进行监听，后一个套接字在调用listen的时候会报错。因此这个方案显然是最佳方案。对于完全不知道我在说什么的童鞋，你们只要这么做

```
s = socket(AF_INET, SOCK_STREAM, 0);

/* What you need to do is add the following two line to your code */

unsigned value = 1;

setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &value, sizeof(value));

/* Then do other things */listen(s, SOMAXCONN);/* ... */
```

- 2.不过据说这样的做法容易产生安全性问题，某些操作系统（没有指明Linux或是BSD或是Windows）允许分别属于两个进程的两个套接字绑定到两个地址的同一个端口。大多数程序把套接字绑定到0.0.0.0这个地址上进行监听（也即监听所有网络设备），这种情形下另外一个进程可以绑定到127.0.0.1或者是其他网络设备的IP地址的同一个端口，并进行监听，就可能把外部的连接给拦截下来（因为127.0.0.1这样的IP是属于特定设备的，比0.0.0.0这虚拟设备更specific)。

而解决这个安全性的问题的方法其实也不难（其实换个没问题的操作系统就可以了不是？），只要把套接字绑定绑定到具体的网络设备的IP地址（比如绑定到127.0.0.1，或者a.b.c.d)即可，大不了为每个网络设备建一个套接字。如果实施起来有难度，只能考虑后面的两种方法。

- 3.让客户端先关闭连接。如果所有的连接关闭事件都在客户端首先发生，那么也不会存在这个问题。不过这种做法可能需要修改协议，而且貌似很容易恶意连接攻击。修改系统TCP超时时间，这种方法很不推荐。

参考

http://rootkit.blogbus.com/logs/32867319.html
http://blog.csdn.net/vhghhd/article/details/7531172
http://www.xfocus.net/projects/Xcon/2002/Xcon2002_noble_shi.pdf
http://fuzzexp.org/php-port-complex-with-the-use-of.html
http://fuzzexp.org/python-port-multiplexing-and-thread-operations.html
http://www.pudn.com/downloads159/sourcecode/others/detail709718.html
http://fuzzexp.org/php-port-complex-with-the-use-of.html
https://www.xfocus.net/bbs/index.php?act=SE&f=3&t=45480&p=181475
http://www.horseb.org/yajing2010.html
https://www.corelan.be/index.php/2011/07/27/metasploit-bounty-the-good-the-bad-and-the-ugly/