Myths About External C2

xret2pwn.github.io/Myths-About-External-C2

September 14, 2022

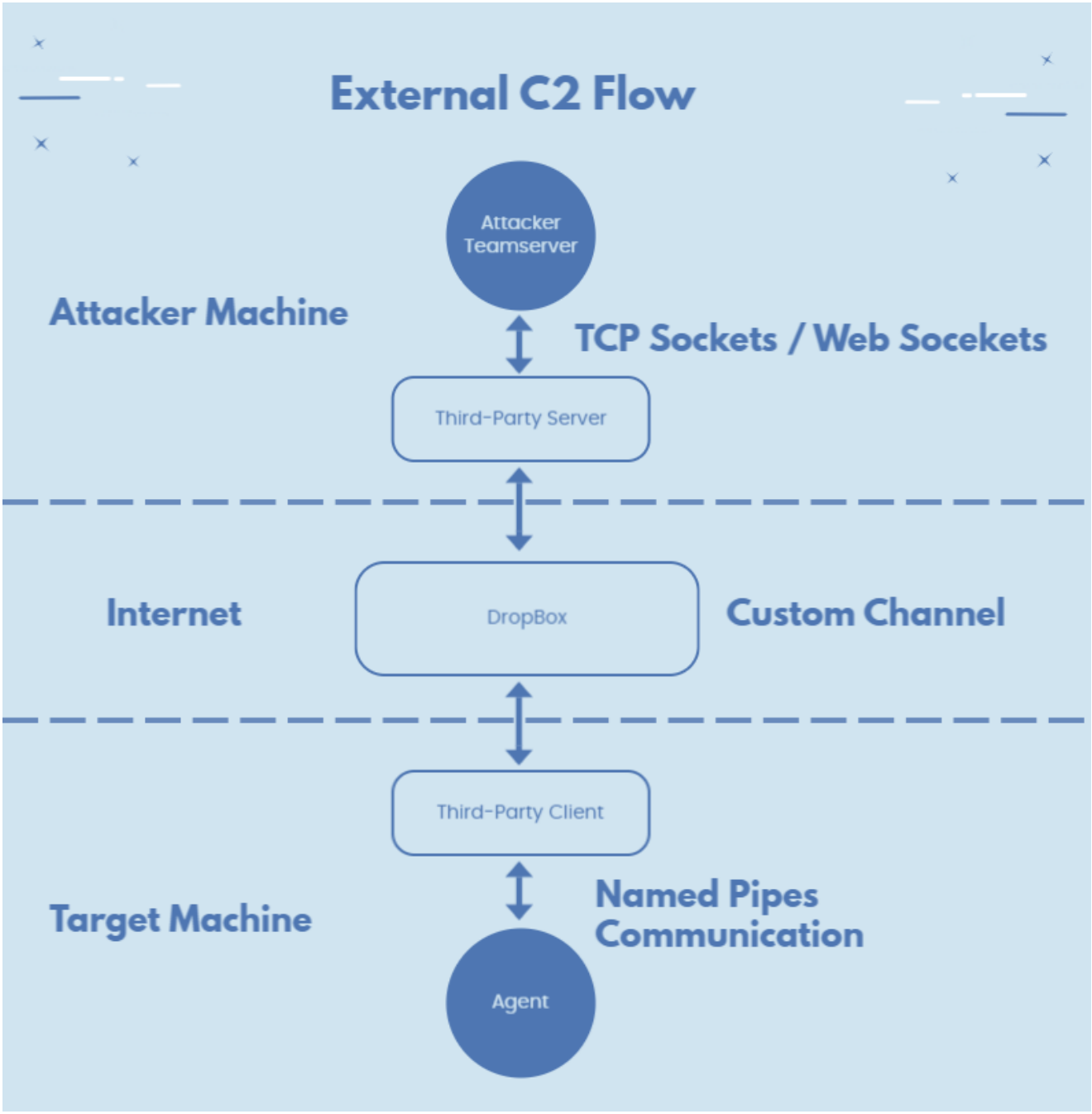
In this blog post I will show you how to build a External C2 in your C2. Excuse me I can not show any actual code in my C2 Framework (Falcon One).

Table of Content

What is the External C2?

Cobalt Strike 3.6 introduced a new feature that's called External C2, to provide the operator a power to build his own communication channel.

I will go through why it's powerful feature, but before that I would let you imagen how is the communication should be.

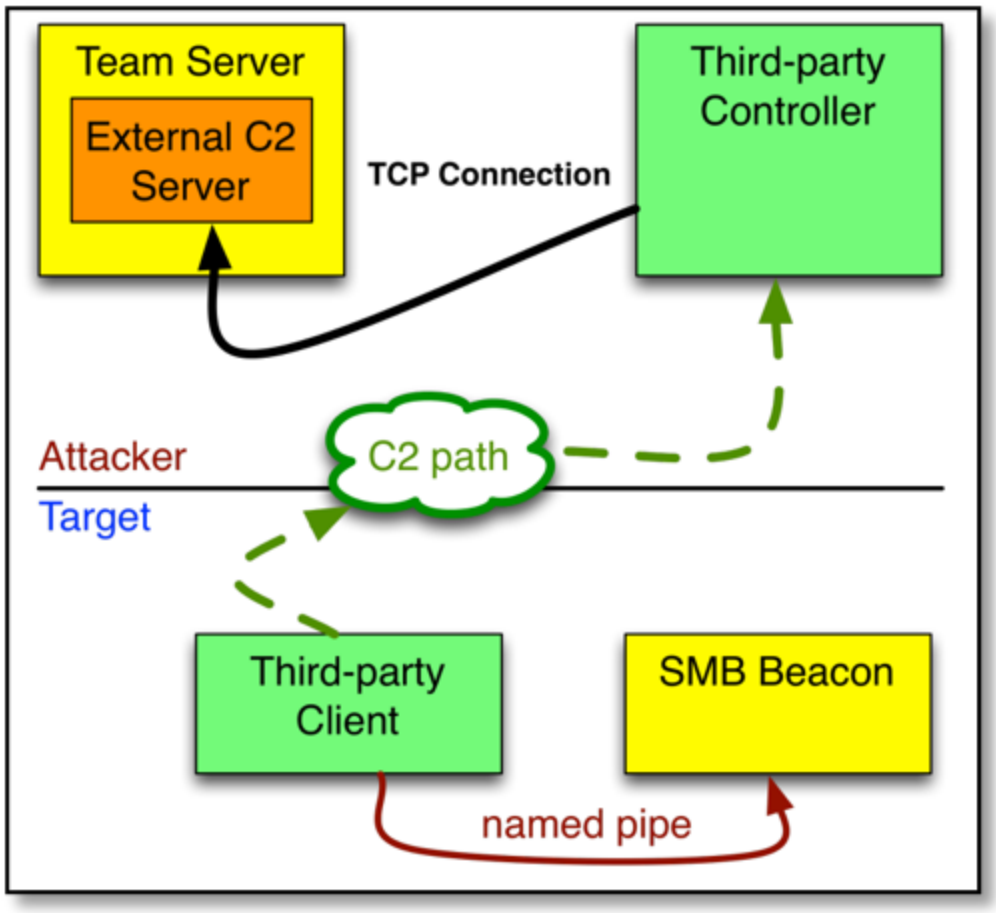


As you see in the above flow, in the Attacker Machine we found the teamserver, and Third-Party Server we called it “Connector”, The teamserver should provide a local port connection (In our case its TCP sockets), to let the third-party server connects with it to take the commands from the teamserver and send it to the custom channel which is Dropbox in our case, then when the commands get executed and the results on the Dropbox then the third-party server will take these results and send it back to the teamserver.

And in the Target Machine we can found Third-Party client and the Main Agent. In the normal case the thrid-party should locate a memory space to inject the shellcode of the agent into it and execute the agent, but in our case I will run the agent manully. when the agent start running it’ll create a named pipe to send and receive the data with the third-party client.

and the main goal of the third-party is take the new tasks (commands) from the Dropbox and send it to the agent process through named pipe and receive the task results and send it back to the dropbox.

Anthor graph could be more clearn made by MDSec



Is it useful?

I will let you answer this question by yourself.
Lets assume there is company uses/trust dropbox for example. and allows the in and out connection.
Looking for some words of wisdom? it is unrealistic to use untrusted domain to get your shell connection. So, in this case you can use dropbox to initial your connection and start communicating via trusted domain. So, is it useful or not?

Understand Windows pipes

To be aware there is alot of abuses you can do in named pipe I may go through them in the up coming posts. but now lets talk about the mechanism of named pipes.
Named pipe is documented by Microsoft, Full documentation of Named Pipe

What is the Windwos Pipe?

Windows Pipe helps when you need to communicate with two applications / processes using shared memory.
You can interact with this shared memory like file object, like ReadFile() and WriteFile().
Named Pipe is implemented on **First in First out (FiFo)** which means when you write data to a pipe and needs to read it, once you read it the data will not be available anymore, the data will be poped out.
But what if we have three application communicates togther and need to keep the data available. In this case you can use WinAPI function that's called `PeekNamedPipe` in windows. I will go through it soon, but shortly this API can be used to read the data without removing.

Types of Windows Pipes

There is two types of windows pipes.

1. Named Pipes.
2. Anonymous pipes.

Named Pipes have a name, but anonymous pipes doesn't have name.

For Example, Named piped like `\\.\Pipe\JustExample`

How it works?

Pipes is implemented based on a server and client, which the server create a named pipe then the client communicates through it.
Named pipes has two methods of communication. `half-duplex` and `duplex`.
Half-duplex will let the client side write data to the server with no repsond.
In Duplex the client side can write data to the server and the server write to the client.

Build a Basic Demo

So, now you have a good base knowledge to start building the demo.

Demo Structure

I'm going to build a python script to emulate the teamserver that python script will open a localhost TCP connection. Then we will build a third-party server that takes Tasks (Commands) and send it to the third-party client which should be in the target machine and send back the task results back to the third-party server.
So, the final flow will be like following:

1. Teamserver opens a localhost TCP connection.
2. Third-party server will take the tasks from the teamserver and send to the client, and receive the task results from to thrid-party client to send it back to the teamserver.
3. Third-party client will take the tasks from the third-party and send it to the agent process via named pipe and receive the task results via named pipe and send it back to the third-party server.

> In this structure we didn't use any custom channel like dropbox, Because this is just for a demonstration but you can build that between the Third-party connectors (server and client). And figure these connectors to send and receive the tasks and task results via that custom channel (Dropbox, Slack, etc.)

Teamserver

I have built a quick python script to enumlat the teamserver, because I don't want to share my C2 source code at this moment I just need to re-write some parts and clean most of it and I'm too lazy to do that. So I decided to build a quick python script that sends tasks to the third-party server through TCP connection.

```
import socket, struct
from time import sleep

print("""How to Implement External C2 in your C2 like Cobalt Strike..
Author: @RET2_pwn
>>                      [[[Teamserver]]]
>> This python script is not my real teamserver of my C2 framwork.
""")
def send_msg(sock, msg):

    msglength = struct.pack('<I', len(msg))
    sock.sendall(msglength)
    sock.sendall(msg.encode())


def recv_msg(sock):

    raw_msglen = recvall(sock, 4)
    if not raw_msglen:
        return None
    msglen = struct.unpack('<I', raw_msglen)[0]

    return recvall(sock, msglen)


def recvall(sock, n):

    data = bytearray()
    while len(data) < n:
        packet = sock.recv(n - len(data))
        if not packet:
            return None
        data.extend(packet)
    return data


with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind(("127.0.0.1", 1337))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        while True:
            command = input("> ")
            send_msg(conn, command)
            data = recv_msg(conn)
            print("[*] Received Data: %s" % data.decode())
```

Third-party Server

As we agreed the third-party server just takes the data form the teamserver to deliver it to whatever channel you pick. in this case I'll do a basic setup by passing the third-part server tasks to the another third-party client which located in the target machine.

```
import socket, struct
from time import sleep


print("""How to Implement External C2 in your C2 like Cobalt Strike..
Author: @RET2_pwn
>>                    [[[Third-Party Server]]]
>> This python script is not my real Third-party server of my C2 framwork.
""")

def send_msg(sock, msg):


    msglength = struct.pack('<I', len(msg))
    print("[+] Sending Data length: ", msglength)
    sock.sendall(msglength+msg)


def recv_msg(sock):

    raw_msglen = recvall(sock, 4)

    print("[+] Receiving Data: %s" % raw_msglen)

    if not raw_msglen:
        return None


    msglen = struct.unpack('<I', raw_msglen)[0]

    return recvall(sock, msglen)


def recvall(sock, n):

    data = bytearray()
    while len(data) < n:
        packet = sock.recv(n - len(data))
        if not packet:
            return None
        data.extend(packet)
    return data


with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
    server.bind(("127.0.0.1", 31337)) # Start TCP Server to send and receive data
between third-party server and client.
    server.listen()
    conn, addr = server.accept()
    with conn:
        print(f"Connected by {addr}")
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
```

```
teamserver_connector:
            teamserver_connector.connect(("127.0.0.1", 1337)) # Intial connection
between the teamserver when then third-party client connect with the third-party
server.
            while True:
                print("="*40)

                data = recv_msg(teamserver_connector)

                print("[+] Receiving Data: %s" % data)

                send_msg(conn,data)
                data = recv_msg(conn)

                print("[+] Receiving Data: %s" % data)

                send_msg(teamserver_connector,data)
```

Named Pipe Communication

So, Now we have both teamserver and third-party server. So, what else we need to do? We need to implement the named pipe creation, write ,and read functions in the agent(Payload/Implant/Beacon), then write our third-party client functions.

> Just For Clarify: Cobalt Strike or any other C2s has a Shellcode generator feature, to generate a shellcode for your agent. What that means? That means they can just allocate a virtual memory space to inject the shellcode when the third-party client begin starting then continue the third-party normal actions like listening to the named pipe to send and receive the data. But in our case I will not generate a shellcode for the agent I will run separately.

Named Pipe Server

If you remember when we spoke about Windows Pipes. You can interact with it like a file object. So all we need to do is:

1. Create our named pipe.
2. wait for a client process to connect.
3. Read and Write into through the Named pipe handle.

Named Pipe Client

Its so simple all we need to do is:

1. Open handle for our Named Pipe like any file by using CreateFileA.
2. Read and Write into our Named pipe handle.

How is the Agent looks like?

For sure when our agent start execution will create our named pipe. Then will wait until someone connect to it to start reading and writing data into the pipe.

Create Named pipe

Here is Example on main function.

```
int main()
{

    HANDLE hPipe;
    char *buf = ( char* )malloc( BUFFER_MAX_SIZE );
    DWORD dwRead;


    hPipe = CreateNamedPipe( TEXT( "\\\\.\\pipe\\ExternalC2Myths" ),
        PIPE_ACCESS_DUPLEX,
        PIPE_TYPE_BYTE | PIPE_READMODE_BYTE | PIPE_WAIT,
        1,
        1024 * 16,
        1024 * 16,
        NMPWAIT_USE_DEFAULT_WAIT,
        NULL );


    while ( hPipe != INVALID_HANDLE_VALUE )
    {
        printf( "[+] Named pipe successfully created.\n" );
        if ( ConnectNamedPipe(hPipe, NULL) != FALSE )
        {
            printf( "[+] Third-Party client successfully connected.\n" );

            while ( TRUE )
            {
                printf( "====================================\n" );

                dwRead = ReadData( hPipe, buf );
                printf( "[*] Reading Data: %s\n", buf );


                printf( "[*] Writing Data.\n" );
                WriteData( hPipe, buf, dwRead + 1 );
            }
        }

    }
    DisconnectNamedPipe( hPipe );
    return 0;
}
```

> NOTE: Buffer_Max_Size you will need to define it by adding the following: `#define BUFFER_MAX_SIZE 1024 * 1024`

If you noticed we have opened the named pipe through `CreateNamedPipe` WinAPI function. this function is well documented by Microsoft, but for a quick view we just made our pipe `PIPE_ACCESS_DUPLEX` that means the third-party client can read and write into it. and made just accept one connection you can notice that in the arguement number 4.

Write Function

The write function will take 3 arguments:

1. The handle of the named pipe.
2. Data to write.
3. Data length.

And its like TCP sockets we will need to add our data size before the data to makesure the client and sever side will read all the data.

So our function should be like that

```
void WriteData( HANDLE hNamedPipe, char* DATA, DWORD DataLength ) {

    DWORD wrote = 0;

    WriteFile(hNamedPipe, (void*)&DataLength, 4, &wrote, 0);
    FlushFileBuffers(hNamedPipe);

    WriteFile(hNamedPipe, DATA, DataLength, &wrote, 0);
    FlushFileBuffers(hNamedPipe);

    printf("[*] Wrote Data: %s ,Data Size: %i\n", DATA, wrote);

}
```

> NOTE: Makesure that you do FlushFileBuffers after writing the data to makesure that you wrote all you need to write.

Read Function

The read function will take 2 arguments:

1. The handle of the named pipe.
2. pointer to data variable

And as we did in write function we just wrote the first 4 bytes with the size of the next data. we will need to get the 4 for bytes to know the exact data we need to read. and that could be done through the following code sample:

```
DWORD ReadData( HANDLE hNamedPipe, char* DATA ) {
    DWORD  temp = 0;

    int size = 0, total = 0;

    ReadFile( hNamedPipe, (char*)&size, 4, &temp, 0 );

    printf( "[*] Reading Data Size: %i\n", size );


    while ( size > total ) {

        ReadFile( hNamedPipe, DATA + total , size - total, &temp, 0 );
        total += temp;
        printf( "[*] Data: %s\n", DATA );
        printf( "[*] Total : %i\n", total );
        printf( "[*] Size : %i\n", size );

    }

    return size;
}
```

That's how our agent looks like.

Build our third-party client

This client should be mix of the agent and the third-party server, because we will need to connect to the TCP server to receive the command and write it into the named pipe then wait until the agent execute the command and write again the data into the pipe then our client will read the data and send it back to the TCP server, then the TCP server will send the data to the teamserver.

Sockets

So, I have created function called connect to connect to the TCP server.

```
SOCKET Connect(string ipAddress, int port) {

    WSAData data;
    WORD    ver = MAKEWORD(2, 2);
    int     wsResult = WSAStartup(ver, &data);

    if (wsResult != 0)
    {
        printf("Can't start Winsock, Err # %i\n", wsResult);
        return 0 ;
    }

    SOCKET  sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == INVALID_SOCKET)
    {
        printf("Can't create socket, Err # %i\n", WSAGetLastError());
        WSACleanup();
        return 0 ;
    }

    sockaddr_in hint;
    hint.sin_family = AF_INET;
    hint.sin_port = htons(port);
    inet_pton(AF_INET, ipAddress.c_str(), &hint.sin_addr);


    int connResult = connect(sock, (sockaddr*)&hint, sizeof(hint));

    if (connResult == SOCKET_ERROR)
    {
        printf("Can't connect to server, Err #%i\n", WSAGetLastError());
        closesocket(sock);
        WSACleanup();
        return 0;
    }
    return sock;

}
```

This function will return the opened socket handle to be able to send and receive message later.
and here is our send and receive functions.

Send TCP Function

```
int SendData( SOCKET sock, char* DATA, DWORD DataLength ) {
    send (sock, (char*)&DataLength , 4, 0 );
    printf( "[*] Sending Data Size: %i, \t Data: %s\n", sizeof DATA, DATA);

    return send( sock, DATA, DataLength , 0 );
}
```

Receive TCP Function

```
int RecData(SOCKET sock, char *DATA) {

    DWORD  total = 0, temp = 0;

    int size = 0;

    recv(sock, (char*)&size, 4, 0);
    printf("[*] Data Size: %i\n", size);

    while (total < size) {
        temp = recv(sock, DATA + total, size - total, 0);
        total += temp;
    }

    return size;
}
```

Open Named pipe handle

Opening Named pipe handle is like opening a file could be done through `CreateFileA` WinAPI.

```
HANDLE hNamedPipe =  INVALID_HANDLE_VALUE;
while ( hNamedPipe == INVALID_HANDLE_VALUE )
{
    hNamedPipe = CreateFileA( "\\\\.\\pipe\\ExternalC2Myths", GENERIC_READ |
GENERIC_WRITE, 0, NULL, OPEN_EXISTING, SECURITY_SQOS_PRESENT | SECURITY_ANONYMOUS,
NULL );
}
```

You can notice the while loop, because the third-party may try to open handle for the named pipe before the agent create the pipe.

Pipe Write/Read Function

Write Function

```
void WriteData( HANDLE hNamedPipe, char* DATA, DWORD DataLength ) {

    DWORD wrote = 0;

    WriteFile( hNamedPipe, ( void* )&DataLength, 4, &wrote, 0 );
    FlushFileBuffers(hNamedPipe);

    WriteFile( hNamedPipe, DATA, DataLength, &wrote, 0 );
    FlushFileBuffers(hNamedPipe);

    printf( "[*] Wrote Data: %s ,Data Size: %i\n", DATA, wrote);

}
```

Read Function

```
DWORD ReadData( HANDLE hNamedPipe, char* DATA ) {

    DWORD  temp = 0;

    int size = 0, total = 0;

    ReadFile(hNamedPipe, (char*)&size, 4, &temp, 0);

    printf("[*] Reading Data Size: %i\n", size);


    while (size > total) {

        ReadFile(hNamedPipe, DATA + total, size - total, &temp, 0);
        total += temp;
        printf("[*] Data: %s\n", DATA);
        printf("[*] Total : %i\n", total);
        printf("[*] Size : %i\n", size);

    }

    return size;
}
```

Execution

The Execution flow should be like that.

1. Teamserver (To open the localhost TCP Connection).
2. Third-party Server (To open a TCP Connection between the third-party client and connect to the teamserver).
3. Agent (To create the named pipe).
4. Third-party Client (To connect to the third-party TCP connection and open handle to the named pipe).

Teamserver

```
┌──(root㉿PwnBox)-[/mnt/c/ExternalC2Demo]
└─# python3 teamserver.py
How to Implement External C2 in your C2 like Cobalt Strike..
Author: @RET2_pwn
>>                        [[[Teamserver]]]
>> This python script is not my real teamserver of my C2 framwork.

Connected by ('127.0.0.1', 9985)
> External C2 Demo
[*] Received Data: External C2 Demo
>
```

Third-party Server

```
┌──(root㉿PwnBox)-[/mnt/c/ExternalC2Demo]
└─# python3 ThirdpartyServer.py
How to Implement External C2 in your C2 like Cobalt Strike..
Author: @RET2_pwn
>>                    [[[Third-Party Server]]]
>> This python script is not my real Third-party server of my C2 framwork.

Connected by ('127.0.0.1', 9984)
========================================
[+] Receiving Data: bytearray(b'\x10\x00\x00\x00')
[+] Receiving Data: bytearray(b'External C2 Demo')
[+] Sending Data length:  b'\x10\x00\x00\x00'
[+] Receiving Data: bytearray(b'\x11\x00\x00\x00')
[+] Receiving Data: bytearray(b'External C2 Demo\x00')
[+] Sending Data length:  b'\x11\x00\x00\x00'
========================================
```

Agent

```
PS C:\ExternalC2Demo\Agent\x64\Release> .\Agent.exe
[+] Named pipe successfully created.
[+] Third-Party client successfully connected.
==================================
[*] Reading Data Size: 16
[*] Data: External C2 Demo
[*] Total : 16
[*] Size : 16
[*] Reading Data: External C2 Demo
[*] Writing Data.
[*] Wrote Data: External C2 Demo ,Data Size: 17
==================================
```

Third-party Client

```
PS C:\ExternalC2Demo\Third-party-Client\x64\Release> .\Third-party-Client.exe
====================================
[*] Data Size: 16
[*] Receiving Data: External C2 Demo
[*] Wrote Data: External C2 Demo ,Data Size: 16
[*] Reading Data Size: 17
[*] Data: External C2 Demo
[*] Total : 17
[*] Size : 17
[*] Reading Size: 17
[*] Sending Data Size: 8,        Data: External C2 Demo
[*] Data Sent
====================================
```

If you have feedback please go ahead and DM me on Twitter, See you in the next blogpost.