# Task 1

## DLL Injection Assignment

**Team Members: Sayed Md Abu, Siyu Deng, Ige Tosin O**

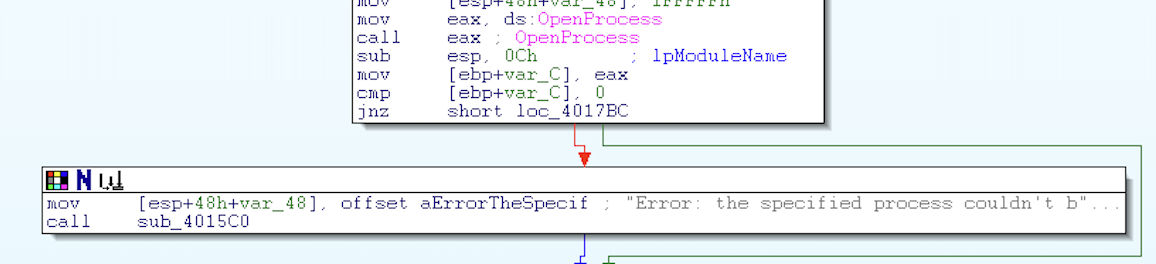
Finish the partially implemented Injector template Injector.cpp by reverse engineering the Injector\_answer.exe.

By reviewing Injector.cpp file, we found that forceProcessToLoadDLL function needs to be complete. Since we are required to make the code has the same functionality as the Injector\_answer.exe. Now, let’s use IDA Pro to inspect Injector\_answer.exe.

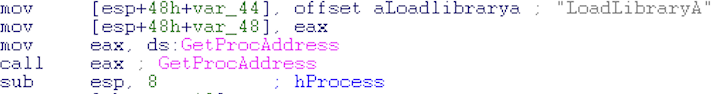
Since we know dll injection requires memory space allocation, we can search where VirtualAllocEx has been called to find out which function handles dll injection job.

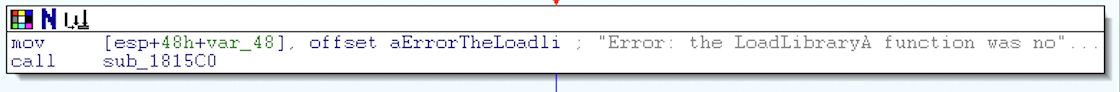
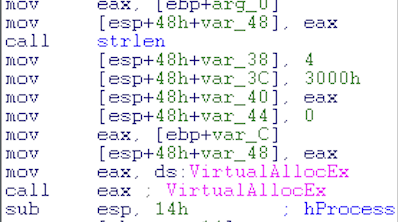
Luckily, there is only one function called VirtualAllocEx, that function is called sub\_401781. Note, function address will change after each execution, probably the instructor enabled ASLR (Address Space Layout Randomization) functionality when compiled the code. We renamed it as “forceProcessToLoadDLL”.

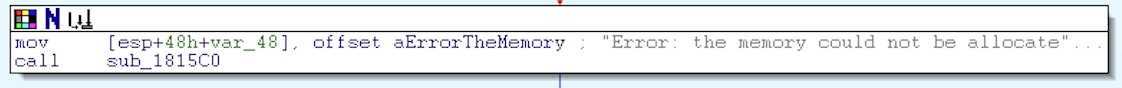
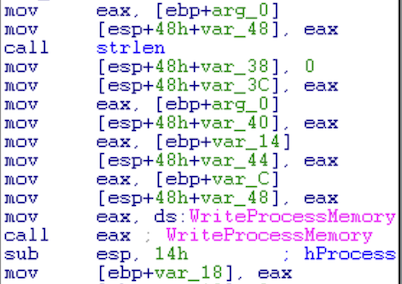
Now, let’s exam how does sub\_401781 work. sub\_401781 implements the injection by the following steps:

Get handle of victim process, print error message if failed

Get handle of kernel32.dll (where LoadLibraryA function locates in)Image

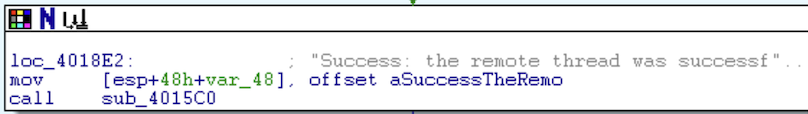
Retrieve the address of “LoadLibraryA” function

print Error message if failed.Allocate memory space for the malicious dll’s pathname

print error message if failedWrite the malicious dll to the process’s memory

print error message if failedCreates a thread that runs in the virtual address space of the process

print error message if failedotherwise print success message



Therefore, we get the idea how to finish up the TODO section. The completed source code is saved in file **Injector.cpp**. For convenience, we renamed variable names and function name in IDA pro. The altered database is saved in **deliverable1.idb**.