BÁO CÁO THỰC HÀNH

**Môn học: Cơ chế hoạt động của mã độc**

**Lab 6: DLL Injection**

*GVHD: Nguyễn Hữu Quyền*

1. **THÔNG TIN CHUNG:**

Lớp: NT230.N21.ATCL

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** | **Email** |
| 1 | Hoàng Văn Anh Đức | 20520890 | 20520890@gm.uit.edu.vn |
| 2 | Nguyễn Mạnh Cường | 20520421 | 20520421@gm.uit.edu.vn |
| 3 | Lê Quang Minh | 20520245 | 20520245@gm.uit.edu.vn |

1. **NỘI DUNG THỰC HIỆN:[[1]](#footnote-1)**

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Kết quả tự đánh giá** |
| 1 | C.1 Cơ chế static pointers | 100% |
| 2 | Tạo DLL Injection | 100% |

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Thành viên thực hiện** |
| 1 | 1+2 | Đức + Cường |

BÁO CÁO CHI TIẾT

## C.1. Cơ chế static pointers

Vào game tạo 2 villagers để food còn lại 100 cho dễ tìmtrong cheat engine

A screenshot of a video game

Description automatically generated

Tìm thấy address của food

A screenshot of a computer

Description automatically generated

Chỉnh sửa value lưu trong address này và vào game test thử

A screenshot of a computer

Description automatically generated

Sử dụng food

A screenshot of a computer

Description automatically generated

Giá trị trong cheat engine thay đổi => Tìm được đúng địa chỉ của food

A screenshot of a computer

Description automatically generated

**POINTER MAP**

Tạo pointer map lưu record address đầu tiên

A screenshot of a computer

Description automatically generated

Lưu pointermap

A screenshot of a computer

Description automatically generated

Scan pointermap đã tạo

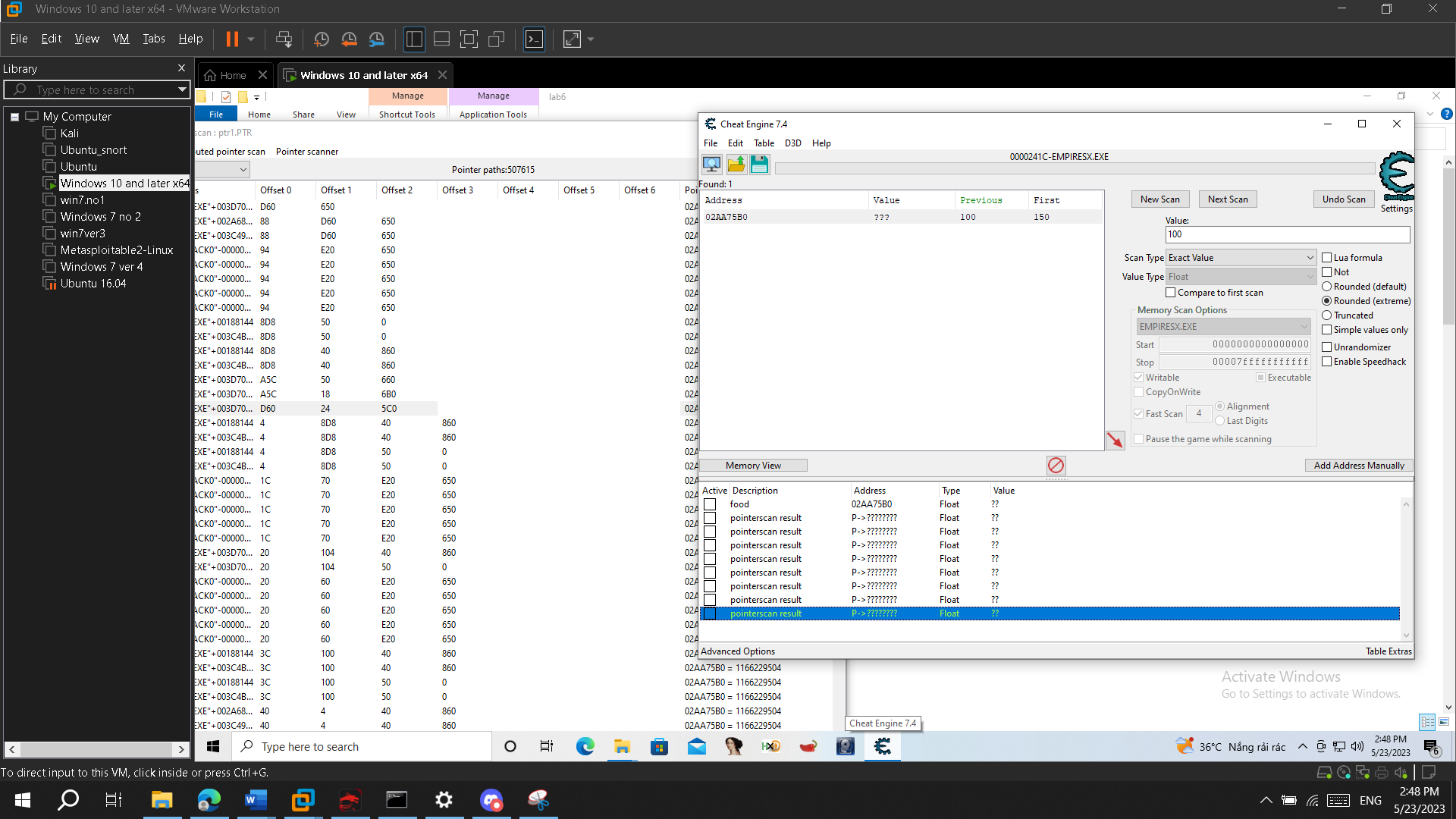
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Lưu offset qua cheat engine và mở lại game thì pointermap không tìm được dynamic address của biến food



Tiếp tục tìm address của biến food thông qua scan như lúc đầu , và tạo pointermap với nó

A screenshot of a computer

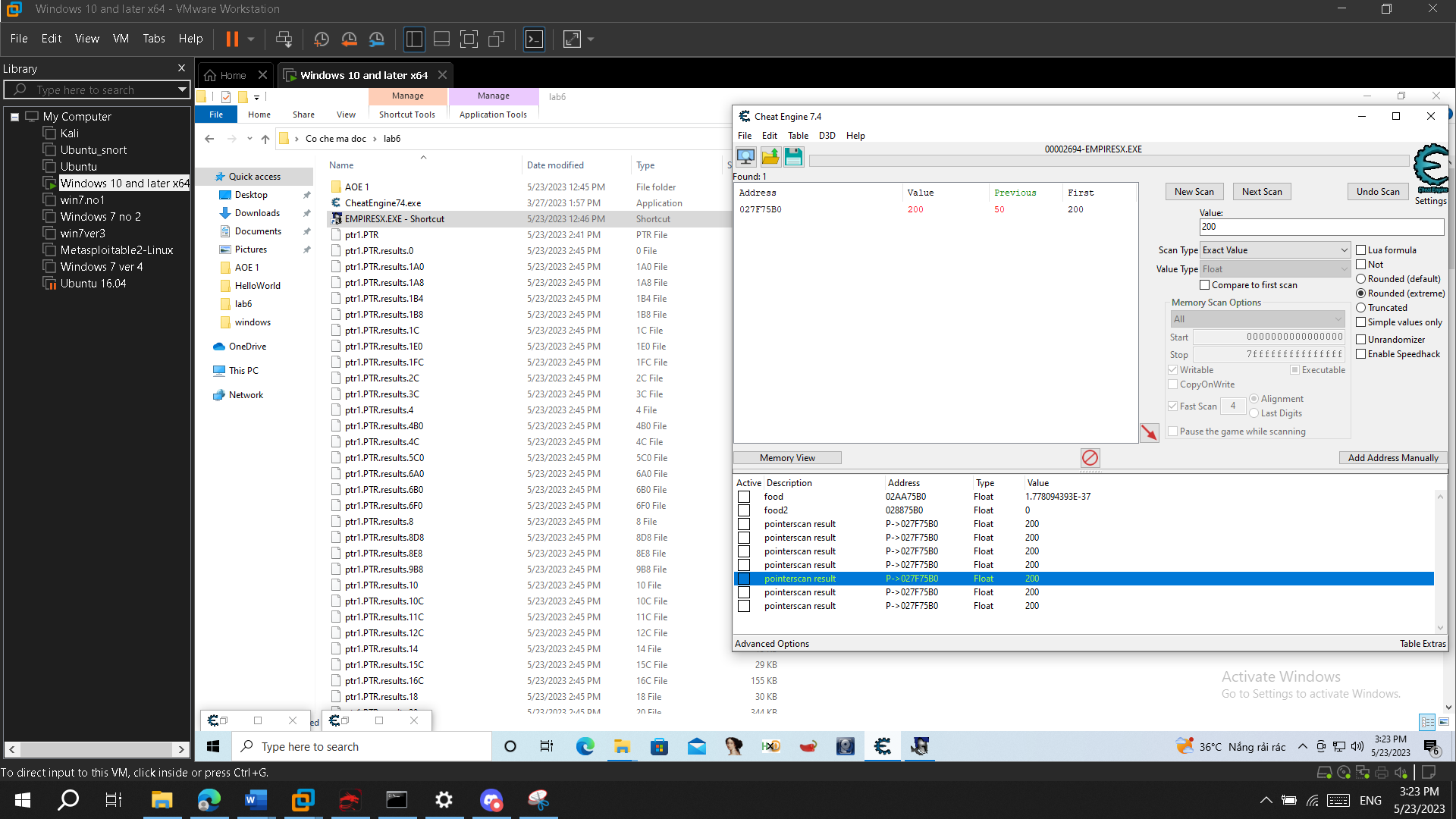
Description automatically generated

Scan pointermap , so sánh với pointermap ban đầu và lưu lại

A screenshot of a computer

Description automatically generated

Khởi động lại game thì đã tìm được biến và value của food



(Hình của pointermap đầu tiên khi khởi động lại game)

A screenshot of a computer

Description automatically generated

Chỉnh sửa biến

A screenshot of a computer

Description automatically generated

Vào game kiểm tra

A screenshot of a video game

Description automatically generated

Khởi động game lại và dùng cheat engine để mở thì vẫn tìm được dynamic address của biến food

A screenshot of a video game

Description automatically generated

A screenshot of a computer

Description automatically generated

* Pointermap trong Cheat Engine không lưu trực tiếp base address của một biến. Thay vào đó, pointermap được sử dụng để tạo ra một danh sách các địa chỉ và offsets dẫn đến một địa chỉ cụ thể trong bộ nhớ. Từ đó, base address của biến có thể được tính toán bằng cách cộng địa chỉ load của module và static base .
* Khi tạo một pointermap trên một địa chỉ cụ thể, nếu địa chỉ base và offsets không thay đổi sau khi khởi động lại chương trình, ta vẫn có thể có địa chỉ của biến đó sau khi khởi động lại.
  + Tuy nhiên, cần lưu ý rằng trong một số trường hợp, địa chỉ base và offsets có thể thay đổi khi chương trình khởi động lại. Điều này có thể xảy ra khi chương trình cấp phát lại vùng nhớ, thay đổi động các đối tượng trong bộ nhớ hoặc khi có sự can thiệp từ phía chương trình.
* Quá trình "scan pointermap" trong Cheat Engine được sử dụng để tìm và xác định các con trỏ (pointers) dẫn đến một địa chỉ cụ thể trong bộ nhớ.

A screenshot of a computer

Description automatically generated

**Khi chạy lại chương trình thì base address có thay đổi không ?**

* Địa chỉ gốc (base address) là địa chỉ bắt đầu của một khu vực bộ nhớ trong một chương trình , nó sẽ thay đổi khi chương trình được cấp phát lại vùng nhớ

**Offsets**

* Dù base address thay đổi, các offsets vẫn có thể truyền được vì chúng chỉ xác định sự dịch chuyển từ địa chỉ gốc đến biến.
* Các offsets này không phụ thuộc trực tiếp vào base address mà chỉ đóng vai trò như một lối đi để truy cập đến biến.
* Do đó, dù base address thay đổi, các offsets vẫn có thể sử dụng để tính toán địa chỉ chính xác của biến.

1. **C2 . Tạo DLL Injection và Demo phần điểm thức ăn được nâng lên sau Inject**.

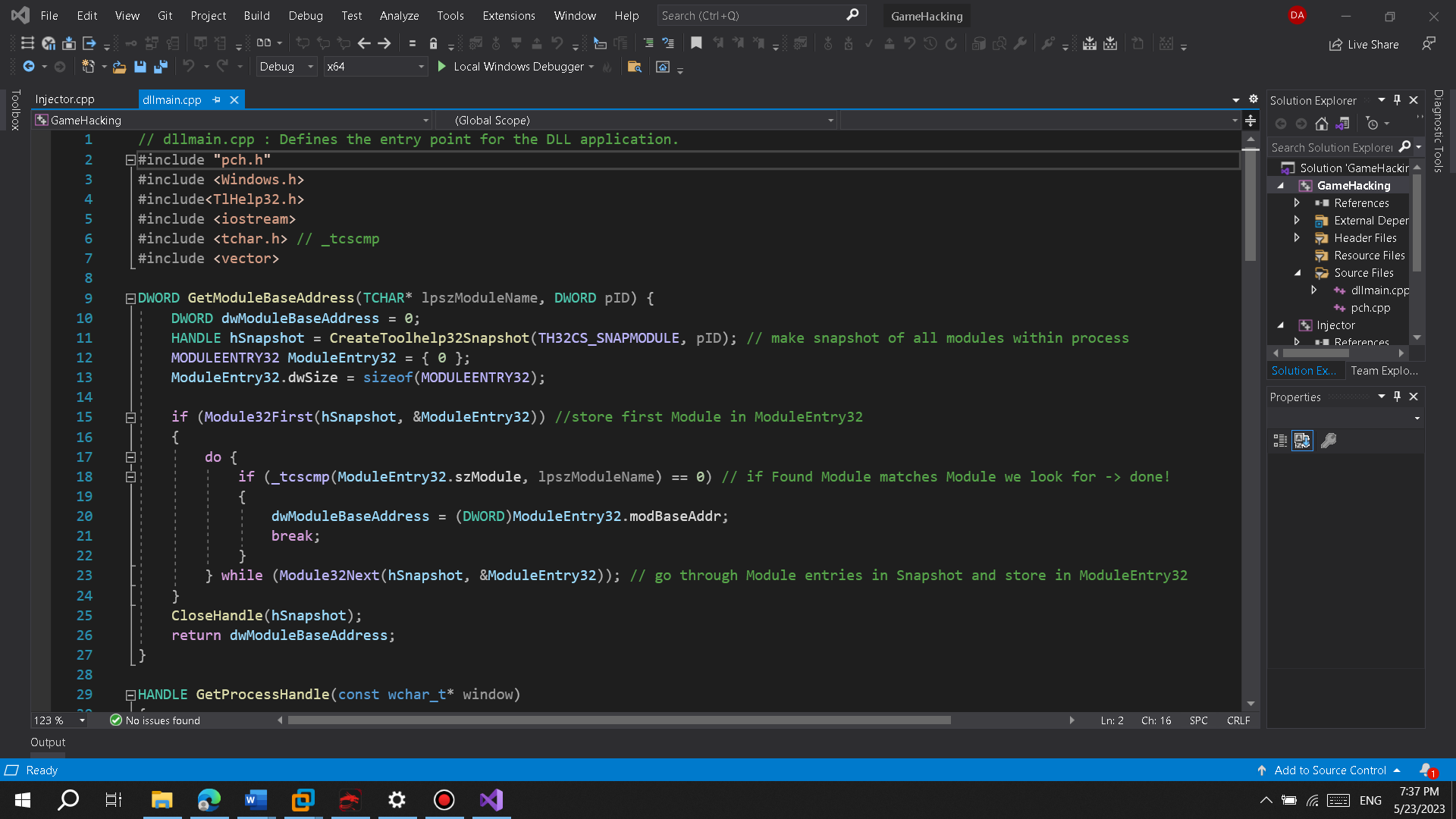
* Viết injector đơn giản để tiêm mã DLL vào quá trình trò chơi "Age of Empires Expansion".

A screenshot of a computer program

Description automatically generated with medium confidence

1. Sử dụng hàm FindWindow để tìm cửa sổ trò chơi dựa trên tên cửa sổ ("Age of Empires Expansion"). Nếu không tìm thấy cửa sổ, chương trình sẽ in ra thông báo "Start the game!" và kết thúc.
2. Sử dụng hàm GetWindowThreadProcessId để lấy Process ID (PID) của tiến trình trò chơi từ cửa sổ trò chơi.
3. Định nghĩa tên tệp tin DLL (GameHacking.dll) và lấy đường dẫn đầy đủ của tệp tin đó.
4. Mở tiến trình trò chơi bằng hàm OpenProcess với các quyền cần thiết để inject code (PROCESS\_CREATE\_THREAD, PROCESS\_QUERY\_INFORMATION, PROCESS\_VM\_OPERATION, PROCESS\_VM\_WRITE, PROCESS\_VM\_READ).
5. Cấp phát bộ nhớ trong tiến trình bằng hàm VirtualAllocEx.
6. Ghi dữ liệu của tệp tin DLL vào bộ nhớ bằng hàm WriteProcessMemory.
7. Tạo một luồng bằng hàm **CreateRemoteThread** và thực thi hàm LoadLibraryA trên bộ nhớ đã cấp phát.
8. Chờ đợi cho luồng thực thi xong bằng hàm **WaitForSingleObject**.
9. Đóng luồng và giải phóng bộ nhớ bằng các hàm tương ứng.
10. Đóng tiến trình trò chơi bằng hàm **CloseHandle**.

Sau khi đã có Injector, ta sẽ viết một DLL để tăng giá trị thức ăn lên 100 mỗi khi nhấn phím F6. Để biết được địa chỉ của vùng nhớ lưu trữ giá trị thức ăn, ta cần xác định địa chỉ cơ sở của tiến trình (vì cơ chế ASLR sẽ làm thay đổi không gian địa chỉ của tiến trình mỗi lần chạy). Dưới đây là định nghĩa của hàm để lấy địa chỉ cơ sở của tiến trình:



1. **Khai báo và khởi tạo biến**:
   1. **dwModuleBaseAddress**: Biến lưu trữ địa chỉ cơ sở của module (base address).
   2. **hSnapshot**: Biến lưu trữ handle của bản snapshot, sẽ được sử dụng để lấy thông tin về module trong tiến trình.
   3. **ModuleEntry32**: Biến kiểu MODULEENTRY32 dùng để lưu trữ thông tin về module.
2. **Tạo snapshot của tất cả các module trong tiến trình**:
   1. **CreateToolhelp32Snapshot(TH32CS\_SNAPMODULE, pID):** Hàm tạo một bản snapshot của tất cả các module trong tiến trình được xác định bởi pID.
3. **Lặp qua các module trong snapshot để tìm module cần tìm:** 
   1. **Module32First(hSnapshot, &ModuleEntry32):** Lấy thông tin của module đầu tiên và lưu trữ nó vào ModuleEntry32.
   2. **do { ... } while (Module32Next(hSnapshot, &ModuleEntry32)):** Tiếp tục lặp qua các module trong snapshot, lấy thông tin của từng module và lưu trữ nó vào ModuleEntry32.
4. **So sánh tên module và lấy địa chỉ cơ sở:** 
   1. **\_tcscmp(ModuleEntry32.szModule, lpszModuleName) == 0**: So sánh tên module trong ModuleEntry32 với lpszModuleName (tên module mà chúng ta muốn tìm).
   2. **(DWORD)ModuleEntry32.modBaseAddr**: Lấy địa chỉ cơ sở (base address) của module từ ModuleEntry32.
5. **Đóng handle của snapshot và trả về địa chỉ cơ sở:** 
   1. **CloseHandle(hSnapshot):** Đóng handle của bản snapshot sau khi đã sử dụng xong.
   2. **return dwModuleBaseAddress**: Trả về địa chỉ cơ sở của module.

Một khi đã có base address thì việc tiếp theo là thay đổi giá trị thức ăn

A screenshot of a computer program

Description automatically generated with medium confidence

Trong phần C.1, ta đã xác định offset từ base address đến con trỏ đầu tiên là 0x003C4B18. Chúng ta cũng đã biết các offset của các con trỏ tiếp theo, lần lượt là 0x3c, 0x100, 0x50, 0x0.

Bằng cách sử dụng hàm ReadProcessMemory, chúng ta có thể truy xuất và lấy giá trị từ các con trỏ này.

Sau khi tính toán được địa chỉ của vùng nhớ chứa giá trị thức ăn, chúng ta sử dụng hàm ReadProcessMemory để lấy giá trị thức ăn hiện tại. Sau đó, chúng ta tăng giá trị này lên 100 và sử dụng hàm WriteProcessMemory để ghi lại giá trị mới vào vùng nhớ tương ứng.Cụ thể :

1. Hàm ModifyFood nhận đối số là một handle của tiến trình (processHandle) để thực hiện các hoạt động đọc và ghi bộ nhớ của tiến trình đó.
2. Sử dụng hàm GetProcessId để lấy ID của tiến trình từ processHandle.
3. Khai báo và khởi tạo biến gameName để lưu tên của trò chơi ("EMPIRESX.EXE").
4. Gọi hàm GetModuleBaseAddress để lấy địa chỉ cơ sở (base address) của module trò chơi trong tiến trình. Địa chỉ cơ sở này được lưu vào biến gameBaseAddress.
5. Khai báo và khởi tạo biến offsetGameToBaseAddress để lưu trữ một offset cần thiết để tính toán địa chỉ điểm (points address).
6. Khai báo và khởi tạo vector pointsOffsets để lưu trữ các offset dùng để tính toán địa chỉ điểm.
7. Khai báo biến baseAddress để lưu giá trị tại địa chỉ gameBaseAddress + offsetGameToBaseAddress. Hàm ReadProcessMemory được sử dụng để đọc giá trị từ bộ nhớ của tiến trình và lưu vào baseAddress.
8. Thiết lập biến pointsAddress bằng baseAddress, đây là địa chỉ điểm ban đầu.
9. Sử dụng vòng lặp để duyệt qua các offset trong vector pointsOffsets. Trong mỗi vòng lặp, hàm ReadProcessMemory được sử dụng để đọc giá trị từ bộ nhớ của tiến trình và cập nhật pointsAddress cho vòng lặp tiếp theo. Cuối cùng, cộng thêm offset cuối cùng vào pointsAddress để có được địa chỉ điểm cuối cùng.
10. Khai báo và khởi tạo biến currentPoint để lưu giá trị hiện tại của điểm. Hàm ReadProcessMemory được sử dụng để đọc giá trị từ bộ nhớ của tiến trình và lưu vào currentPoint.
11. Tính toán giá trị mới của điểm (newPoints) bằng cách thêm 100 vào currentPoint.
12. Sử dụng hàm WriteProcessMemory để ghi giá trị mới (newPoints) vào địa chỉ điểm (pointsAddress) trong bộ nhớ của tiến trình. Kích thước được chỉ định là 4 byte.
13. **DEMO**

<https://youtu.be/DDvGTF_YGq4>

**Trong file .pdf hướng dẫn dùng 4 offsets vậy thì dùng 3,5,6,… offsets thì có hack được game không ? Vì sao ?**

* Đầu tiên biến food trong game là dynamic address nên khi tạo pointermap cho address và scan nó thì nó sẽ cho ra rất nhiều static address (static address + địa chỉ load module = base address ) và tùy vào từng static address thì sẽ có số lượng offsets khác nhau để tính toán để lấy được address của biến food
  + Kết quả scan trong file pdf là 4 offsets nên sẽ có công thức của 4 offsets
* Sẽ có những trường hợp scan được 6 offsets
* Dùng 3,5,6,… offsets tùy vào từng trường hợp khi scan pointermap thì vẫn hack được game như bình thường

1. Ghi nội dung công việc, các kịch bản trong bài Thực hành [↑](#footnote-ref-1)