Name: Mukul Rai

Student ID: 700748568

GitHub Link for Project: https://github.com/raimukul/Malware_Project

Project 2

You should parse two given applications in project1 for their all IMAGE_DIRECTORY_ENTRY_IMPORT (including INT table, Name/Hint Table), all IMAGE_DIRECTORY_ENTRY_BASERELOC, and all Section headers. The codes should be written in C or C++. Each of the structure could have more than one and you should parse all of them.
For example, =you should print their values similar as below.
e_magic: 5A4D
e_cblp: 90
e_cp: 3
e_crlc : 0
e_cparhdr: 4
…
All values should be in hexadecimal.

Code (Using C Programming Language)

```c
#include <stdio.h>
#include <windows.h>

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }
```

```
    HANDLE fileHandle = CreateFile(argv[1], GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
NULL);
    if (fileHandle == INVALID_HANDLE_VALUE)
    {
        printf("Error opening file %s\n", argv[1]);
        return 1;
    }

    HANDLE mappingHandle = CreateFileMapping(fileHandle, NULL, PAGE_READONLY, 0, 0, NULL);
    if (mappingHandle == NULL)
    {
        printf("Error creating file mapping\n");
        CloseHandle(fileHandle);
        return 1;
    }

    LPVOID mapView = MapViewOfFile(mappingHandle, FILE_MAP_READ, 0, 0, 0);
    if (mapView == NULL)
    {
        printf("Error creating file mapping view\n");
        CloseHandle(mappingHandle);
        CloseHandle(fileHandle);
        return 1;
    }

    PIMAGE_DOS_HEADER dosHeader = (PIMAGE_DOS_HEADER)mapView;
```

```
if (dosHeader->e_magic != IMAGE_DOS_SIGNATURE)
{
    printf("Invalid DOS signature\n");
    UnmapViewOfFile(mapView);
    CloseHandle(mappingHandle);
    CloseHandle(fileHandle);
    return 1;
}

PIMAGE_NT_HEADERS ntHeaders = (PIMAGE_NT_HEADERS)((LPBYTE)dosHeader + dosHeader->e_lfanew);
if (ntHeaders->Signature != IMAGE_NT_SIGNATURE)
{
    printf("Invalid NT signature\n");
    UnmapViewOfFile(mapView);
    CloseHandle(mappingHandle);
    CloseHandle(fileHandle);
    return 1;
}

PIMAGE_DATA_DIRECTORY importDirectory = &ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT];
printf("Import Directory (virtual address): 0x%08X, size: % 4X\n", importDirectory->VirtualAddress, importDirectory->Size);

PIMAGE_DATA_DIRECTORY exportDirectory = &ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT];
printf("Export Directory (virtual address): 0x%08X, size: % 4X\n", exportDirectory->VirtualAddress, exportDirectory->Size);

PIMAGE_DATA_DIRECTORY resourceDirectory = &ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_RESOURCE];
```

```
    printf("Resource Directory (virtual address): 0x%08X, size: % 4X\n", resourceDirectory->VirtualAddress, resourceDirectory->Size);

    PIMAGE_DATA_DIRECTORY exceptionDirectory = &ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXCEPTION];
    printf("Exception Directory (virtual address): 0x%08X, size: % 4X\n", exceptionDirectory->VirtualAddress, exceptionDirectory->Size);

    PIMAGE_DATA_DIRECTORY securityDirectory = &ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_SECURITY];
    printf("Security Directory (virtual address): 0x%08X, size: % 4X\n", securityDirectory->VirtualAddress, securityDirectory->Size);

    PIMAGE_DATA_DIRECTORY baserelocDirectory = &ntHeaders-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC];
    printf("Base Relocation Table (virtual address): 0x%08X, size: % 4X\n", baserelocDirectory->VirtualAddress, baserelocDirectory->Size);

    PIMAGE_DATA_DIRECTORY debugDirectory = &ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_DEBUG];
    printf("Debug Directory (virtual address): 0x%08X, size: % 4X\n", debugDirectory->VirtualAddress, debugDirectory->Size);

    PIMAGE_DATA_DIRECTORY architectureDirectory = &ntHeaders-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_ARCHITECTURE];
    printf("Architecture Specific Data (virtual address): 0x%08X, size: % 4X\n", architectureDirectory->VirtualAddress, architectureDirectory->Size);

    PIMAGE_DATA_DIRECTORY globalptrDirectory = &ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_GLOBALPTR];
    printf("RVA of GP (virtual address): 0x%08X, size: % 4X\n", globalptrDirectory->VirtualAddress, globalptrDirectory->Size);

    PIMAGE_DATA_DIRECTORY tlsDirectory = &ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_TLS];
    printf("TLS Directory (virtual address): 0x%08X, size: % 4X\n", tlsDirectory->VirtualAddress, tlsDirectory->Size);
```

```
    PIMAGE_DATA_DIRECTORY loadconFigDirectory = &ntHeaders-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG];
    printf("Load Configuration Directory (virtual address): 0x%08X, size: % 4X\n", loadconFigDirectory->VirtualAddress, loadconFigDirectory->Size);

    PIMAGE_DATA_DIRECTORY boundImportDirectory = &ntHeaders-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT];
    printf("Bound Import Directory in headers (virtual address): 0x%08X, size: % 4X\n", boundImportDirectory->VirtualAddress,
boundImportDirectory->Size);

    PIMAGE_DATA_DIRECTORY iatDirectory = &ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IAT];
    printf("Import Address Table (virtual address): 0x%08X, size: % 4X\n", iatDirectory->VirtualAddress, iatDirectory->Size);

    PIMAGE_DATA_DIRECTORY delayImportDirectory = &ntHeaders-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT];
    printf("Delay Load Import Descriptors (virtual address): 0x%08X, size: % 4X\n", delayImportDirectory->VirtualAddress, delayImportDirectory-
>Size);

    PIMAGE_DATA_DIRECTORY descriptorDirectory = &ntHeaders-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR];
    printf("COM Runtime descriptor (virtual address): 0x%08X, size: % 4X\n", descriptorDirectory->VirtualAddress, descriptorDirectory->Size);

     PIMAGE_DATA_DIRECTORY relocDirectory = &ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC];
    printf("IMAGE_DIRECTORY_ENTRY_BASERELOC (virtual address): 0x%08X, size: % 4X\n", relocDirectory->VirtualAddress, relocDirectory-
>Size);

    UnmapViewOfFile(mapView);
```

```
    CloseHandle(mappingHandle);
    CloseHandle(fileHandle);

    return 0;
}
```

Output

1. *Output for AntivirusPlatinum.exe*

```
D:\Projects\Malware_Project\Project 02>project2.exe AntivirusPlatinum.exe
Import Directory (virtual address): 0x00000000, size:    0
Export Directory (virtual address): 0x00021000, size: 3E60
Resource Directory (virtual address): 0x00000000, size:    0
Exception Directory (virtual address): 0x00000000, size:    0
Security Directory (virtual address): 0x000122A0, size:   1C
Base Relocation Table (virtual address): 0x00000000, size:    0
Debug Directory (virtual address): 0x00000000, size:    0
Architecture Specific Data (virtual address): 0x00000000, size:    0
RVA of GP (virtual address): 0x00000000, size:    0
TLS Directory (virtual address): 0x00000000, size:    0
Load Configuration Directory (virtual address): 0x00012000, size:  2A0
Bound Import Directory in headers (virtual address): 0x00000000, size:    0
Import Address Table (virtual address): 0x00000000, size:    0
Delay Load Import Descriptors (virtual address): 0x00000000, size:    0
COM Runtime descriptor (virtual address): 0x7865742E, size:   74
IMAGE_DIRECTORY_ENTRY_BASERELOC (virtual address): 0x00000000, size:    0

D:\Projects\Malware_Project\Project 02>
```

2. *Output For Stardust.exe*