Arun Kulkarni
December 10, 2020
COSC310 Final Project

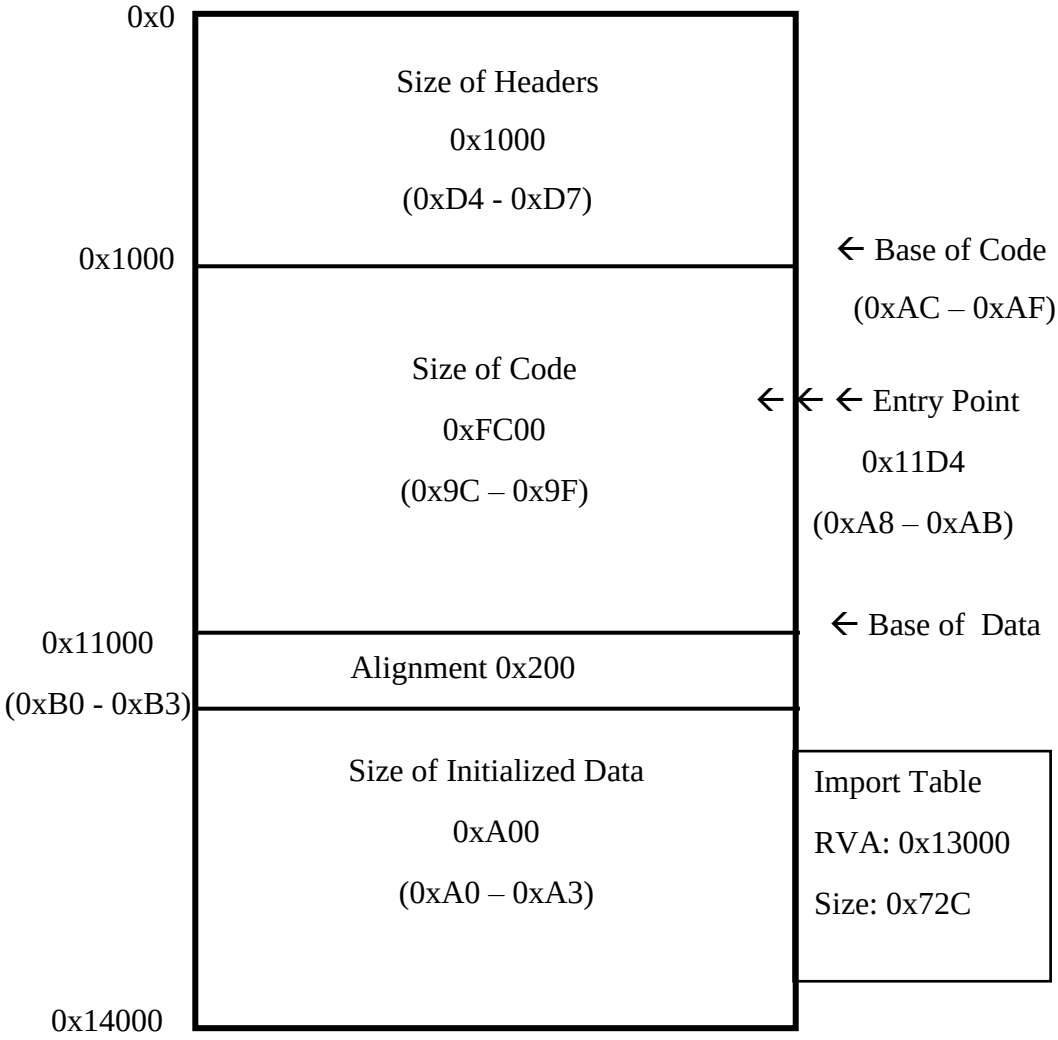
Introduction

For this project, I chose the binary file from Blackboard called **diff.exe**. This file is an MZ DOS 32-bit PE file, as identified by the ASCII string "MZ" at the beginning of the file.

Using PeView, I found that this file contains multiple sections to the header: IMAGE_DOS_HEADER, MS_DOS Stub Program, IMAGE_NT_HEADERS (Signature, IMAGE_FILE_HEADER, IMAGE_OPTIONAL_HEADER), and IMAGE_SECTION_HEADERs for .text, .data, .bss, and .idata.

Rebuilt Memory Map

diff.exe Memory Map – MZ DOS PE32



Additional Information

Size: 68,096 bytes = 0x10A00 (diff.exe)

Image Base: 0x400000

Size of Image: 0x14000

Virtual Addresses: 0x1000, 0x11000, 0x12000, 0x13000

Header Files

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	MZ	DOS														
0	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00
	signature		bytes on last page of file		pages in file		relocations		size of hdr in pgraphs		min extra pgraphs		max extra pgraphs		initial rel. SS	
1	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00
	initial SP		checksum		initial IP		initial rel CS		offset to reloc. table		overlay number		reserved		reserved	
2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	reserved		reserved		OEM ID		OEM Info.		reserved		reserved		reserved		reserved	
3	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00
	reserved		reserved		reserved		reserved		reserved		reserved		offset to new exe header			
4	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54 **T**	68 **H**
5	69 **I**	73 **S**	20	70 **P**	72 **R**	6F **O**	67 **G**	72 **R**	61 **A**	6D **M**	20	63 **C**	61 **A**	6E **N**	6E **N**	6F **O**
6	74 **T**	20	62 **B**	65 **E**	20	72 **R**	75 **U**	6E **N**	20	69 **I**	6E **N**	20	44 **D**	4F **O**	53 **S**	20
7	6D **M**	6F **O**	64 **D**	65 **E**	2E	0D	0D	0A	24	00	00	00	00	00	00	00
8	50 **P**	45 **E**	00	00	4C	01	04	00	FC	7E	7E	3A	00	00	00	00
					machine i386		# of sections		Time Date Stamp				ptr to symbol table			
9	00	00	00	00	E0	00	0F	01	0B	01	02	37	00	FC	00	00
	number of symbols				opt. header size		characteristics		magic		major/minor link. version		size of code			
A	00	0A	00	00	00	08	00	00	D4	11	00	00	00	10	00	00
	size of init. data				size of uninit. data				address of entry point				base of code			
B	00	10	01	00	00	00	40	00	00	10	00	00	00	02	00	00
	base of data				image base				section alignment				file alignment			
C	04	00	00	00	01	00	00	00	04	00	00	00	00	00	00	00
	major/minor OS version				major/minor image version				maj/min subsys. vers.				WIN32 version value			

NT Headers MS-DOS Stub Program DOS HEADER

NT Headers

.text

.data

.bss

D	00	40	01	00	00	04	00	00	00	00	00	00	03	00	00	00
	size of image				**size of headers**					**checksum**			**subsystem**		**DLL chars.**	
E	00	00	00	02	00	10	00	00	00	00	10	00	00	10	00	00
	size of stack reserve				**size of stack commit**				**size of heap reserve**				**size of heap commit**			
F	00	00	00	00	10	00	00	00	00	00	10	00	00	10	00	00
	loader flags				**number of data dirs.**				**RVA/Size Export Table**							
10	00	30	01	00	2C	07	00	00	00	00	00	00	00	00	00	00
	RVA/Size Import Table								**RVA/Size Resource Table**							
11	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	RVA/Size Exception Table								**RVA/Size Certificate Table**							
12	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	RVA/Size Base Relocation Table								**RVA/Size DEBUG directory**							
13	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	RVA/Size Architecture Specific Data								**RVA/Size Global Pointer Register**							
14	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	RVA/Size TLS Table								**RVA/Size LOAD CONFIGURATION Table**							
15	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	RVA/Size BOUND IMPORT Table								**RVA/Size IMPORT Address Table**							
16	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	RVA/Size DELAY IMPORT Descriptors								**RVA/Size CLI Header**							
17	00	00	00	00	00	00	00	00	2E **.**	74 **t**	65 **e**	78 **x**	74 **t**	00	00	00
18	B8	FA	00	00	00	10	00	00	00	FC	00	00	00	04	00	00
	Virtual Size				**RVA**				**Size of Raw Data**				**Pointer to Raw Data**			
19	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	60
	Pointer to Relocations				**Pointer to line numbers**				**# Relocations/Line Nos.**				**Characteristics**			
1A	2E **.**	64 **d**	61 **a**	74 **t**	61 **a**	00	00	00	8C	00	00	00	00	10	01	00
									Virtual Size				**RVA**			
1B	00	02	00	00	00	00	01	00	00	00	00	00	00	00	00	00
	Size of Raw Data				**Pointer to Raw Data**				**Pointer to Relocations**				**Pointer to line numbers**			
1C	00	00	00	00	40	00	00	C0	2E **.**	62 **b**	73 **s**	73 **s**	00	00	00	00
	# Relocations/Line Nos.				**Characteristics**											
1D	60	06	00	00	00	20	01	00	00	00	00	00	00	00	00	00
	Virtual Size				**RVA**				**Size of Raw Data**				**Pointer to Raw Data**			

.idata

1E	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	C0
	Pointer to Relocations				**Pointer to line numbers**				**# Relocations/Line Nos.**				**Characteristics**			
1F	2E	69	64	61	74	61	00	00	2C	07	00	00	00	30	01	00
	.	**i**	**d**	**a**	**t**	**a**			**Virtual Size**				**RVA**			
20	00	08	00	00	00	02	01	00	00	00	00	00	00	00	00	00
	Size of Raw Data				**Pointer to Raw Data**				**Pointer to Relocations**				**Pointer to line numbers**			
21	00	00	00	00	40	00	00	C0	00	00	00	00	00	00	00	00
	# Relocations/Line Nos.				**Characteristics**											

Analysis

To find all of this information about the headers, as mentioned, I used PeView. This next section outlines the C analysis I did of the executable

I first loaded the file into the C program, opening it as read-only. I then used the built-in C functions fseek(), ftell(), and rewind() to determine the size of the file. I allocated a buffer equivalent to the size of the file, casting the buffer as type unsigned char. This allowed me to parse through the data byte-by-byte.

```c
//declare vars
FILE *pFile, *output;
long lSize;
unsigned char *buffer;
size_t result;

//open read file diff.exe
pFile = fopen("diff.exe","r");
if (pFile == NULL){
    fputs("Could not open diff.exe\n",stderr);
    exit(1);
}

//open write file, syscalls.txt
output = fopen("syscalls.txt","w");

//obtain file size
fseek(pFile, 0, SEEK_END); //seek to end of file
lSize = ftell(pFile); //get file pointer
rewind(pFile); //rewind to beginning of file

//allocate memory for entire file
//add one byte for NULL character to terminate
//memory string
buffer = (unsigned char *)malloc(sizeof(unsigned char)*lSize + 1);
if (buffer == NULL) {
    fputs("Memory error", stderr);
    exit(2);
}
```

I then copied the file into the buffer, and looped through all the data looking for the bytes **0xFF25**, as there **are no FF15 system calls in this executable.** When these bytes were found, I printed the next 4 bytes (for the 32-bit address) to a text file called syscalls.txt, along with the addresses of the calls in 32-bit hex format. I had to reformat from Little Endian to Big Endian for the addresses to be readable.

```
//set memory to zero before copying in
memset(buffer, 0, sizeof(unsigned char)*lSize + 1);

//copy file into buffer
result = fread(buffer, sizeof(unsigned char), lSize, pFile);
if (result != lSize){
    fputs("Reading error", stderr);
    exit(3);
}

//whole file now loaded into memory buffer
//crawl through file looking for FF 25 opcodes
//print calls and addresses to syscalls.txt
printf("Parsing diff.exe for FF 25 System calls...\n");
printf("Calls and location addresses in syscalls.txt\n");
fprintf(output,"FF 25 System Call:\t\tLocation Address:\n\n");
for (int i = 0; i < lSize; i += 8){
    if (buffer[i] == 0xFF){
        if (buffer[i + 1] == 0x25 || buffer[i + 1] == 0x15){
            fprintf(output,"%02x %02x %02x %02x %02x %02x",buffer[i],
                buffer[i+1],buffer[i+2],buffer[i+3],buffer[i+4],buffer[i+5]);

            fprintf(output,"\t\t0x%02x%02x%02x%02x\n",buffer[i+4],buffer[i+3],buffer[i+2]);
        }
    }
}

//terminate
fclose(pFile);
fclose(output);
free(buffer);
return 0;
```

When the program (ArunProject) is compiled and run, we are given a message that the system calls are being output to a file.

```
arun@arun-VirtualBox:~/Documents/COSC310/Project$ ./ArunProject.sh
Script File ArunProject.sh
Done
arun@arun-VirtualBox:~/Documents/COSC310/Project$ ./ArunProject
Parsing diff.exe for FF 25 System calls...
Calls and location addresses in syscalls.txt
```

Running the bash command “cat syscalls.txt,” we see that the FF 25 calls have been successfully parsed, along with the location addresses of the calls in readable Big-Endian form. A total of 72 FF 25 calls were parsed, along with the associated addresses.

```
arun@arun-VirtualBox:~/Documents/COSC310/Project$ cat syscalls.txt
FF 25 System Call:              Location Address:

ff 25 b4 31 41 00               0x4131b4
ff 25 9c 31 41 00               0x41319c
ff 25 ac 31 41 00               0x4131ac
ff 25 a8 31 41 00               0x4131a8
ff 25 98 31 41 00               0x413198
ff 25 a0 31 41 00               0x4131a0
ff 25 a4 31 41 00               0x4131a4
ff 25 b0 31 41 00               0x4131b0
ff 25 94 31 41 00               0x413194
ff 25 dc 31 41 00               0x4131dc
ff 25 e4 31 41 00               0x4131e4
ff 25 a8 32 41 00               0x4132a8
ff 25 c0 31 41 00               0x4131c0
ff 25 50 32 41 00               0x413250
ff 25 e0 31 41 00               0x4131e0
ff 25 80 32 41 00               0x413280
ff 25 cc 31 41 00               0x4131cc
ff 25 ec 31 41 00               0x4131ec
ff 25 f4 31 41 00               0x4131f4
ff 25 20 32 41 00               0x413220
ff 25 18 32 41 00               0x413218
ff 25 9c 32 41 00               0x41329c
ff 25 a0 32 41 00               0x4132a0
ff 25 f8 31 41 00               0x4131f8
ff 25 10 32 41 00               0x413210
ff 25 14 32 41 00               0x413214
ff 25 0c 32 41 00               0x41320c
ff 25 6c 32 41 00               0x41326c
ff 25 84 32 41 00               0x413284
ff 25 5c 32 41 00               0x41325c
ff 25 8c 32 41 00               0x41328c
ff 25 04 32 41 00               0x413204
ff 25 88 32 41 00               0x413288
ff 25 64 32 41 00               0x413264
ff 25 fc 31 41 00               0x4131fc
ff 25 78 32 41 00               0x413278
ff 25 60 32 41 00               0x413260
ff 25 c8 31 41 00               0x4131c8
ff 25 a4 32 41 00               0x4132a4
ff 25 4c 32 41 00               0x41324c
ff 25 48 32 41 00               0x413248
ff 25 7c 32 41 00               0x41327c
ff 25 58 32 41 00               0x413258
ff 25 40 32 41 00               0x413240
ff 25 70 32 41 00               0x413270
```

As seen in the table above, there is no “symbol table” located in this executable. However, there is an “IMPORT Address Table.” Upon looking at this table, I found that the addresses given by PeView do not match the addresses found when doing the parsing in C. I believe that this is because the address values are changed when they are resolved by the operating system in which they are being currently run. Below I have an image of the system call names located in the IMPORT address table, not the symbol table.

pFile	Data	Description	Value
00010384	000132B0	Hint/Name RVA	007B ExitProcess
00010388	000132C0	Hint/Name RVA	0279 SetUnhandledExceptionFilter
0001038C	00000000	End of Imports	KERNEL32.dll
00010394	000132E0	Hint/Name RVA	000B _dup2
00010398	000132E8	Hint/Name RVA	001E _fstat
0001039C	000132F4	Hint/Name RVA	002E _lseek
000103A0	00013300	Hint/Name RVA	0034 _open
000103A4	00013308	Hint/Name RVA	003A _read
000103A8	00013310	Hint/Name RVA	0006 _close
000103AC	0001331C	Hint/Name RVA	003E _setmode
000103B0	00013328	Hint/Name RVA	0041 _spawnl
000103B4	00013334	Hint/Name RVA	0049 _stat
000103B8	00000000	End of Imports	msvcrt.dll
000103C0	0001333C	Hint/Name RVA	0076 _cexit
000103C4	00013348	Hint/Name RVA	0089 _cwait
000103C8	00013354	Hint/Name RVA	0093 _errno
000103CC	00013360	Hint/Name RVA	00A9 _fileno
000103D0	0001336C	Hint/Name RVA	00AA _findclose
000103D4	0001337C	Hint/Name RVA	00AB _findfirst
000103D8	0001338C	Hint/Name RVA	00AD _findnext
000103DC	00013398	Hint/Name RVA	00B2 _fmode
000103E0	000133A4	Hint/Name RVA	00B5 _fpreset
000103E4	000133B0	Hint/Name RVA	00DE _iob
000103E8	000133B8	Hint/Name RVA	015A _pipe
000103EC	000133C0	Hint/Name RVA	0175 _setmode
000103F0	000133CC	Hint/Name RVA	0184 _stat
000103F4	000133D4	Hint/Name RVA	0027 __getmainargs
000103F8	000133E4	Hint/Name RVA	018B _stricmp
000103FC	000133F0	Hint/Name RVA	01FE abort
00010400	000133F8	Hint/Name RVA	0205 atexit
00010404	00013404	Hint/Name RVA	0207 atoi
00010408	0001340C	Hint/Name RVA	0210 ctime
0001040C	00013414	Hint/Name RVA	0213 exit
00010410	0001341C	Hint/Name RVA	0216 fclose
00010414	00013428	Hint/Name RVA	0218 ferror
00010418	00013434	Hint/Name RVA	0219 fflush
0001041C	00013440	Hint/Name RVA	0222 fprintf
00010420	0001344C	Hint/Name RVA	0228 free
00010424	00013454	Hint/Name RVA	0230 fwrite
00010428	00013460	Hint/Name RVA	0234 getenv
0001042C	0001346C	Hint/Name RVA	023A isalnum
00010430	00013478	Hint/Name RVA	023B isalpha
00010434	00013484	Hint/Name RVA	023C iscntrl
00010438	00013490	Hint/Name RVA	023D isdigit
0001043C	0001349C	Hint/Name RVA	0240 islower
00010440	000134A8	Hint/Name RVA	0241 isprint
00010444	000134B4	Hint/Name RVA	0242 ispunct
00010448	000134C0	Hint/Name RVA	0243 isspace

0001044C	000134CC	Hint/Name RVA	0244 isupper
00010450	000134D8	Hint/Name RVA	003B __p__environ
00010454	000134E8	Hint/Name RVA	0252 isxdigit
00010458	000134F4	Hint/Name RVA	025B malloc
0001045C	00013500	Hint/Name RVA	025F memchr
00010460	0001350C	Hint/Name RVA	0261 memcpy
00010464	00013518	Hint/Name RVA	0263 memset
00010468	00013524	Hint/Name RVA	0266 perror
0001046C	00013530	Hint/Name RVA	0268 printf
00010470	0001353C	Hint/Name RVA	0269 putc
00010474	00013544	Hint/Name RVA	026A putchar
00010478	00013550	Hint/Name RVA	026E qsort
0001047C	00013558	Hint/Name RVA	0271 realloc
00010480	00013564	Hint/Name RVA	0279 signal
00010484	00013570	Hint/Name RVA	027C sprintf
00010488	0001357C	Hint/Name RVA	0280 strcat
0001048C	00013588	Hint/Name RVA	0282 strcmp
00010490	00013594	Hint/Name RVA	0284 strcpy
00010494	000135A0	Hint/Name RVA	028A strncmp
00010498	000135AC	Hint/Name RVA	028B strncpy
0001049C	000135B8	Hint/Name RVA	028D strrchr
000104A0	000135C4	Hint/Name RVA	029A time
000104A4	000135CC	Hint/Name RVA	029D tolower
000104A8	000135D8	Hint/Name RVA	004E __set_app_type
000104AC	00000000	End of Imports	msvcrt.dll

Conclusion

For this project, I chose the binary executable named **diff.exe** from the provided files in BlackBoard. I identified the type of binary file, and rebuilt the memory stack from the data provided using PeView. Additionally, I constructed a table with all of the hexadecimal data from all of the header files, and labeled all sections appropriately.

After this, I wrote a C program to parse FF 25 system calls and their location addresses, and printed them all to an output file. I found the IMPORT Address Table, and all of the system call names in the table, but the addresses were different so my tracing was unable to go any further.

From this project I learned some basic fundamentals of binary executable reverse-engineering, including how to use disassemblers, how to crawl through raw data looking for information, and how to re-build a memory stack given header files. In the future I hope to explore this type of work further, as it has many “real-world” applications in the realm of computer science and systems programming.