

0. 과제

- 목표 : PE파일(x64)을 생성하는 프로그램 만들기
- 평가요소 :
 - ① Entrypoint, ImageBase, Section Alignment, File Alignment값을 받아 PE파일 생성.
(import table등 없어도 됨. section table은 text, data, rdata, rsrc 4개로 고정)
 - ② GUI로 제작할 것
 - ③ 사용자가 지정한 개수의 PE 파일을 임의 이름으로 생성할 것.

1. PE File

1-1) PE File

- PE-COFF(Portable Executable Common Object File Format) 의 약자이다.
- 32비트와 64비트에서의 구조가 다르며, Windows에서 독립적으로 실행 가능하다.

1-2) PE File(x64) 구조

Header (실행파일의 주요 정보)	DoS header (바이너리 정보)	64 Bytes
	DoS Stub Code (필수 구성요소 아님)	가변적
	PE Header (프로그램 실행 정보)	
	Optional Header (구조체 정보)	
	Data Directories (추가 구조체 정보)	
	Sections Table (각 섹션에 대한 정보)	섹션당 40 Bytes
Section 1 (실행파일의 콘텐츠)	Code (실행 내용)	가변적
	Imports (Windows 라이브러리 정보)	
	Data (code에서 사용되는 정보)	
...		
Section N (실행파일의 콘텐츠)	Code (실행 내용)	가변적
	Imports (Windows 라이브러리 정보)	
	Data (code에서 사용되는 정보)	

[표 1-1] PE File(x64) 구조

① DoS Header

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....yy..
PE	파일	시그니처	00	00	00	00	40	00	00	00	00	00	00	00	00@.....
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00PE Header 오프셋
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

[그림 1-1] DoS Header

- PE 파일의 첫 부분으로, 64 Bytes 크기가 고정되어있다.
- MZ(4D 5A) 시그니처를 가진다.
- 마지막 4바이트는 PE Header의 시작부분을 가리키고 있다.

② DoS Stub Code

```

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..*...!..Li!Th
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 mode....$.

```

[그림 1-2] DoS Stub

- DoS Header 바로 뒤에 위치한다.
- 보통 VisualStudio 등을 사용하여 컴파일 시, Linker가 알아서 삽입하지만, 없어도 된다.
- Windows가 아닌 DoS 환경에서 실행되었을 경우, 실행되는 코드로, 보통 "This Program must be run under Microsoft Windows" 라는 메시지를 출력하고 종료되는 코드가 삽입된다.
- 컴파일 시 Stub 옵션을 사용해서 원하는 Stub Code를 삽입할 수 있다!

③ PE Header (PE Signature(4 Bytes) + IMAGE FILE HEADER (20 Bytes))

```

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
0000 시그니처 50 45 00 00 4C 01 0E 00 4D CC 2C 5F 00 7E 00 00 PE..L...Mi,~...
00000090 E9 01 00 00 E0 00 07 01 é...ä...
Optional Header의 크기 PE파일 종류

```

[그림 1-3] PE Header

위치	정보
0 ~ 3	PE Signature (50 45 00 00)
4 ~ 5	프로세서 정보
6 ~ 7	총 섹션 개수
8 ~ 11	빌드 시각
12 ~ 15	Symbol Table Pointer
16 ~ 19	Symbol 개수
20 ~ 21	Optional Header의 크기 ¹⁾
22 ~ 23	PE파일 종류

[표 1-2] PE Header 구조

④ Optional Header (IMAGE OPTIONAL HEADER)

```

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
50 45 00 00 4C 01 0E 00 0B 01 02 20 00 30 00 00 PE..L.....
00 00 00 00 E0 00 07 01 0B 01 02 20 00 30 00 00 .P...ä...0..
00 00 00 00 Entry Point 위치 00 12 00 00 File Alignment .P...ä...
04 00 00 00 Image Base 00 00 40 00 00 10 00 00 00 02 00 00 .....0.....
00 20 01 00 00 04 00 00 00 04 00 00 00 10 00 00 Section Alignment 0 00 00
메모리에 올릴 PE file의 크기 00 00 10 00 00 10 00 00 00 02 00 00 .....*/.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 Data Directory 개수 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 Data Directories .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

[그림 1-4] Optional Header

1) 보통 224 Bytes이지만 Optional Hedaer의 일부인 Data Directory가 필요 없을 경우, 96 Bytes가 된다.

- 보통 224 Bytes이지만 Data Directory가 필요 없을 경우, 96 Bytes가 된다.
- Object 파일일 경우 없어도 되지만, EXE 파일일 경우 반드시 존재해야한다.

위치	정보	위치	정보
0 ~ 1	Magic Number (0B 01)	46 ~ 47	Image 마이너 버전
2	Linker 메이저 버전	48 ~ 49	SubSystem 메이저 버전
3	Linker 마이너 버전	50 ~ 51	SubSystem 마이너 버전
4 ~ 7	Code 크기	52 ~ 55	Win32 버전
8 ~ 11	초기화된 데이터 크기	56 ~ 59	Image 크기 ²⁾
12 ~ 15	초기화되지 않은 데이터 크기	60 ~ 63	Header 크기 ³⁾
16 ~ 19	Address of Entry Point(RVA)	64 ~ 67	Checksum
20 ~ 23	Base of Code	68 ~ 69	SubSystem
24 ~ 27	Base of Data	70 ~ 71	DLL Characteristics
28 ~ 31	Image Base ⁴⁾	72 ~ 75	Stack Reserve 크기
32 ~ 35	Section Alignment ⁵⁾	76 ~ 79	Stack Commit 크기
36 ~ 39	File Alignment ⁶⁾	80 ~ 83	Heap Reserve 크기
40 ~ 41	OS 메이저 버전	84 ~ 87	Heap Commit 크기
42 ~ 43	OS 마이너 버전	88 ~ 91	Loader Flags
44 ~ 45	Image 메이저 버전	92 ~ 95	RVA개수 및 크기

[표 1-3] Optional Header 구조

- 64 ~ 67번의 CheckSum을 작성하는 것이 마음에 걸려 CheckSum에 대하여 찾아보았으나, 다 행히 CheckSum 값은 커널 모드 드라이버나 어떤 시스템 DLL이 아닌 이상 모두 0으로 채워 넣어도 무방하다는 사실을 알게 되었다.
- 이에 어떤 필드가 없으면 안 되는지 하나하나 지워 나간 결과, [표1-3]의 노란 색 칠한 필드 이외에는 0으로 채워져 있어도 무방함을 알게 되었다.

⑤ Data Directory

- 일반적으로 128 Bytes 배열로 구성되어있다. (보통 16개의 Entry)
- 마지막 8 Bytes는 Padding값으로, 00으로 채워져 있다.
- 각 Entry는 Size(4 Bytes) + VirtualAddress(4 Bytes)로 이루어져 있다.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
Optional Header								00	00	00	00	00	00	00	00
00	80	00	00	E8	05	00	00	Entry Export								€..è..
Entry Import								00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
04	A0	00	00	18	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	30	81	00	00	E0	00	00	000...à...
Padding								00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	2E	74	65	78	Section Table			text...

[그림 1-5] Data Directory

2) Section Alignment × (섹션 수 + 1)

3) File Alignment의 배수

4) 가상메모리에서의 PE파일이 로딩 되는 주소. PE 로더는 Address of EntryPoint와 Image Base주소를 더해서 Entry Point를 만든다.

5) 각 섹션이 메모리에서 차지해야하는 최소 단위. 일반적으로 페이지 사이즈와 동일한 4,096값을 사용한다.

6) 디스크 상에서 각 섹션이 차지해야하는 최소 단위. 512부터 65,535 사이의 2의 n승 값만 적용 가능하다.

위치	정보	위치	정보
0 ~ 7	Entry Export	64 ~ 71	Entry GlobalPTR
8 ~ 15	Entry Import	72 ~ 79	Entry TLS
16 ~ 23	Entry Resource	80 ~ 87	Entry Load Config
24 ~ 31	Entry Exception	88 ~ 95	Entry Bound Import
32 ~ 39	Entry Security	96 ~ 103	Entry IAT
40 ~ 47	Entry BaseReLoc	104 ~ 111	Entry Delay Import
48 ~ 55	Entry Debug	112 ~ 119	Entry Com Descriptor
56 ~ 63	Entry Copyright	120 ~ 128	NULL

[표 1-4] Data Directory 구조

- 각 Entry는 크기(4 Bytes)와 VirtualAddress(4 Bytes)로 구성되어있다.
- EXE대상 주요 엔트리 정보를 담고 있는 대상을 노란 색으로 표시 해 보면, [표 1-4]와 같다.
- ① Entry Export : DLL에 존재하는 Export Table의 메모리상에서의 시작점과 크기에 대한 정보.
- ② Entry Import : DLL에 존재하는 Import Table의 메모리상에서의 시작점과 크기에 대한 정보.

⑥ Section Table

- 각 섹션에 대한 정보가 저장되는 영역.
- 섹션 당 40 Bytes로 구성되며, 시작위치, 크기, 메모리 번지, 권한 등을 저장한다.

[그림 1-6] Section Table

위치	정보	위치	정보
0 ~ 7	Section 이름	20 ~ 23	Pointer to Raw Data
8 ~ 11	Virtual Size	24 ~ 35	Padding
12 ~ 15	Virtual Address (RVA)	36 ~ 39	Characteristics
16 ~ 19	Size of Raw Data		

[표 1-5] Section Table 구조

이름	기능
.text	프로그램 실행 코드를 담고 있으며, IAT(Import Address Table)정보를 포함하고 있다.
.data	전역변수와 static 변수 할당을 위해 존재하는 영역으로, 읽기/쓰기가 가능하다.
.rdata	문자열과 상수 등을 담고 있는 영역으로, 읽기 전용이다.
.bss	초기화 되지 않은 전역변수를 담고 있는 영역으로, 보통 .data와 병합되기 때문에 메모리상에는 따로 존재하지 않는다.
.rsrc	아이콘, 커서 등의 리소스 관련 데이터들을 담고 있는 영역이다.
.idata	Import할 DLL과 관련 API들에 대한 정보를 담고 있는 영역이다.
.edata	Export할 API에 대한 정보를 담고있는 영역이다.

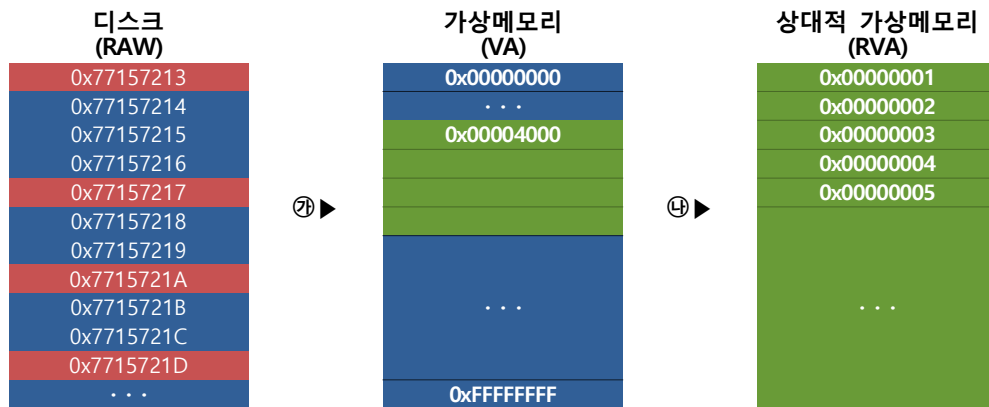
[표 1-6] Section 종류

⑦ Section

- 동일한 성질의 데이터가 모여 있는 연속적인 영역이다.

1-3) 용어

- ① RAW : PE파일 내에서 특정 데이터가 쓰인 주소
- ② VA(Virtual Address) : PE 파일이 메모리에 로드될 때 특정 데이터가 쓰이는 절대 주소
(VA = RVA + Imagebase)
- ③ RVA(Relative Virtual Address) : PE 파일이 메모리에 로드된 뒤 특정 데이터가 쓰이는 상대주소
(Imagebase로부터의 상대주소 - (offset))



㉞ : 디스크에 띄엄띄엄 저장된 데이터를 연속적인 주소를 가지도록 메모리에 로드.

㉟ : 가상 메모리(VA)의 한 영역(Sector)을 다시 0x00000000 부터의 상대적 주소로 나타냄.

- ④ IAT (Import Address Table) : 사용되는 라이브러리(DLL), 함수에 대한 정보가 담긴 테이블.
(함수명, 함수시작 주소 등 포함)
(IMAGE_IMPORT_DESCRIPTOR의 배열로 이루어져 있다)
(IMAGE_IMPORT_DESCRIPTOR로 로드 할 DLL개수 +1(마지막 NULL))

2. 프로그램 제작

2-1) 문자를 띄우는 PE파일 제작

- PE파일을 구성하는 요소 중, 값이 고정되는 헤더 영역을 먼저 만들기로 하였다.
- 먼저 각 헤더를 구성하는 바이너리 파일을 작성 한 다음, 아래 구문을 작성하여 코드로 옮겼다.

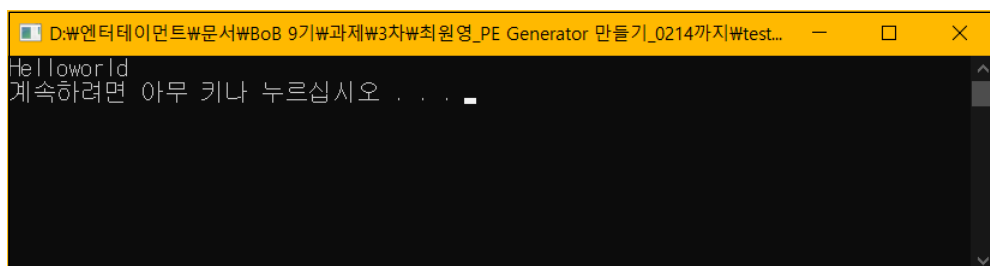
파일 바이너리를 읽고 10진수로 변환하여 화면에 출력하는 함수.py

```
def cropBinaryArea(filename):
    buffer = ''
    counter = 0
    with open(filename, 'rb') as f:
        byte = f.read(1)
        while byte != b'':
            counter += 1
            buffer += str(int(byte.hex(), 16)) + ' '
            if counter > 15:
                buffer += '\n'
                counter = 0
            byte = f.read(1)
    print(buffer.replace(' 00', ' 0').replace('\n00', '\n0'))
```

```
DOS_HEADER = [77, 90, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 0, 0, 0]
DOS_STUB = [14, 31, 186, 14, 0, 180, 9, 205, 33, 184, 1, 76, 205, 33, 84, 104,
105, 115, 32, 112, 114, 111, 103, 114, 97, 109, 32, 99, 97, 110, 110,
116, 32, 98, 101, 32, 114, 117, 110, 32, 105, 110, 32, 68, 79, 83, 32,
109, 111, 100, 101, 46, 13, 13, 10, 36, 0, 0, 0, 0, 0, 0, 0]
PE_HEADER = [80, 69, 0, 0, 76, 1, 4, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 224, 0, 3, 1]
```

[그림 2-1] 각 헤더 영역이 완성 된 모습

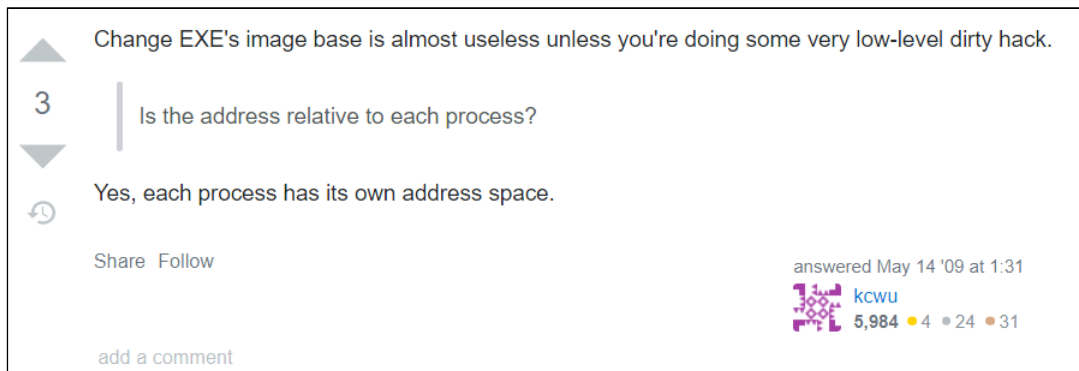
- Optional Header 까지는 한 땀 한 땀 제작이 가능하였으나, Data Directory, Section 부분은 내용도 길지만 어셈블리 언어를 바이너리로 작성해야 하여 복잡했다. 때문에 메시지를 팝업 시키는 PE파일을 공수 해 온 다음, Data Directory와 각 Section영역만 잘라오기로 하였다.
- 다행히 잘라온 내용과 앞서 살펴본 지식을 토대로 Helloworld를 출력하는 작은 PE파일 생성에 성공하였다.



[그림 2-2] 64비트 환경에서 정상 작동하는 PE 파일

2-2) Entrypoint, ImageBase, Section Alignment, File Alignment 값 수정 기능 부여

- 값을 수정하기 전의 Entrypoint는 text 영역 중에서 특정 위치를 가리키고 있었기 때문에 값을 변경할 경우, 추가 연산이 필요 해 보였다.
- ImageBase는 PE파일 실행 시 로딩 될 가상메모리 주소였기에 아무 값이나 넣으면 알아서 그 곳에 들어갈 줄 알았는데 아니었다. 다른 프로세스와 충돌을 피하기 위해 EXE 파일은 0x00400000 주소로 고정 되어 있었다.
- 그래도 과제를 하는 김에 이것도 바꿀 수 없을까 싶어 수소문 하였지만, 변태 같은 해킹을 하려는 것이 아니라면 하지 말라는 답변을 볼 수 있었다.



[그림 2-3] 각 프로세스별 주소가 할당되어있다는 답변

- 때문에 ImageBase의 경우, 이상한 값을 넣었을 땐, 실행 불가 경고 메시지와 함께 해당 값을 작성한 프로그램을 추출하도록 작성하기로 하였다.
- Section Alignment는 각 섹션이 메모리에서 차지해야하는 최소 단위였으며, 페이지(4,096 Bytes) 단위의 크기를 가졌다.
- 한번 4,096의 배수 값을 마구 넣어 보았으나, 4,096 이외의 값에서는 PE파일이 작동하지 않았다.
- File Alignment는 각 섹션이 디스크 상에서 차지해야하는 최소 단위였으며, 512 ~ 65535 사이의 값이 들어갈 수 있었다.
- 할 수 없다는 부정적인 결과만 계속 나오고 있었지만, 분명 값을 변경할 수 있으니 멘토님께서 과제로 내 주셨을 것이라 생각하며, 직접 상관관계를 분석 해 보기로 하였다.

대상	값	대상	값
AddressOfEntryPoint	SectionAlignment + 691	ImageBase	SectionAlignment x 1024
SectionAlignment	4096	FileAlignment	512
ImageSize	SectionAlignment x 5	HeaderSize	FileAlignment x 2

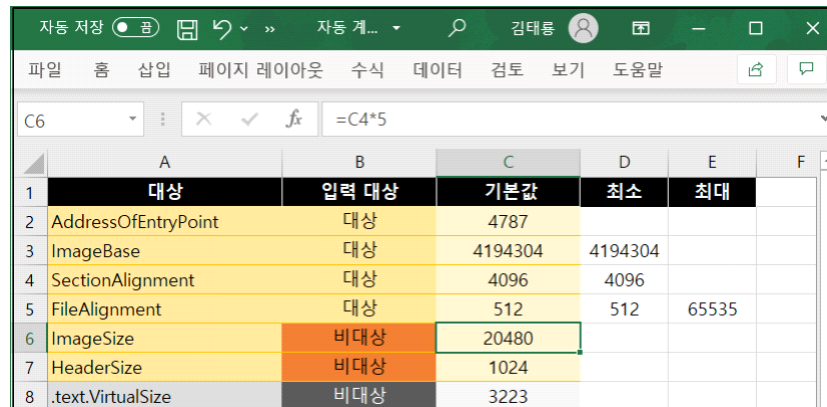
[표 2-1] 생성된 PE의 헤더 영역 관계 표

대상	.text	.rdata	.data	.rsrc
VirtualSize	3223	2720	904	480
VirtualAddress	SectionAlignment x 1	SectionAlignment x 2	SectionAlignment x 3	SectionAlignment x 4
SizeOfRawData	3584	3072	512	512
PointerToRawData	HeaderSize	HeaderSize + .text.SizeOfRawData	HeaderSize + .text.SizeOfRawData + .rdata.SizeOfRawData	HeaderSize + .text.SizeOfRawData + .rdata.SizeOfRawData + .data.SizeOfRawData

[표 2-2] 생성된 PE의 헤더 영역 관계 표

- 직접 표를 제작 해 보니 어느 정도 윤곽이 보이기 시작했다.
- AddressOfEntryPoint는 첫 Section의 시작 지점에서 조금 떨어진 부분을 가리키고 있었다.
- ImageBase는 SectionAlignment의 배수로 설정되어 있었다.
- SectionAlignment는 페이지 단위(4096)의 배수로 설정되어 있었다.
- FileAlignment는 512의 배수로 설정되어있었다.
- ImageSize는 생성한 Section 수 + 헤더만큼의 공간을 확보해야 했다.
- HeaderSize는 FileAlignment의 배수로, Section이 시작되기 전 까지 크기였다.
- Section들은 VirtualSize가 크든 작든 SectionAlignment의 배수로 VirtualAddress가 결정되었다.

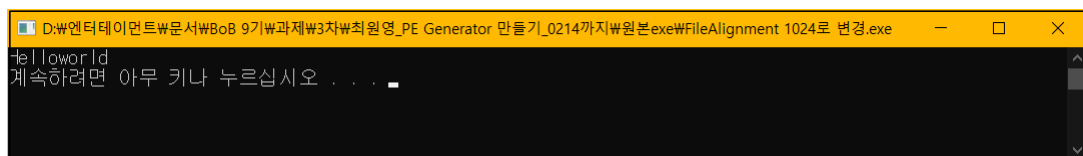
- Section들의 SizeOfRawData는 FileAlignment의 배수로 이루어졌으며, 빈공간은 0으로 채워졌다.
- 각 Section의 PointerToRawData는 이전 섹션들의 SizeOfRawData와 HeaderSize의 합이었다.



	A	B	C	D	E	F
1	대상	입력 대상	기본값	최소	최대	
2	AddressOfEntryPoint	대상	4787			
3	ImageBase	대상	4194304	4194304		
4	SectionAlignment	대상	4096	4096		
5	FileAlignment	대상	512	512	65535	
6	ImageSize	비대상	20480			
7	HeaderSize	비대상	1024			
8	.text.VirtualSize	비대상	3223			

[그림 2-4] 수식을 활용한 주소 값 자동 계산용 시트

- 자동 계산 시트를 이용하여 각 주요 대상의 기본 값을 변경 해 보고, 시트에 나온 값을 토대로 PE파일을 생성 해 보았다.



[그림 2-5] FileAlignment값을 1024로 변경한 PE 파일이 잘 동작하는 모습

- FileAlignment를 1024로 변경한 계산 결과대로 PE파일을 구성시켰을 땐 잘 작동하였으나, Section Alignment, ImageBase, AddressOfEntryPoint 값을 변경했을 때는 제대로 동작하지 않았다.

대상		값	대상		값
AddressOfEntryPoint		SectionAlignment + 691	ImageBase		SectionAlignment x 1024
SectionAlignment		4096	FileAlignment		512의 배수
ImageSize		SectionAlignment x 5	HeaderSize		FileAlignment x 2
대상	.text	.rdata	.data	.rsrc	
VirtualSize	3223	2720	904	480	
VirtualAddress	SectionAlignment * 1	SectionAlignment * 2	SectionAlignment * 3	SectionAlignment * 4	
SizeOfRawData	3584	3072	512	512	
PointerToRawData	HeaderSize	HeaderSize + .text.SizeOfRawData	HeaderSize + .text.SizeOfRawData + .rdata.SizeOfRawData	HeaderSize + .text.SizeOfRawData + .rdata.SizeOfRawData+ data SizeOfRawData	

[표 2-3] 알맞은 알고리즘

- 그래도 FileAlignment 계산식을 알아냈기에 해당 식대로 다시 코드를 고쳐 냄으로 써 FileAlignment 값은 사용자 지정이 가능 해 졌다.
- 하지만 FileAlignment 값이 SectionAlignment 값 보다 커질 수 없기 때문에, 설정되지 않은 EntryPoint, SectionAlignment, ImageSize에 대하여 적용 방법을 찾아보기로 하였다.

- 물리적 위치가 아닌 가상 주소, 상대적 가상 주소 값 수정을 위해 문서를 읽던 도중, EntryPoint 값은 RVA라서 ImageBase와 VA에서의 위치 값을 더한 것임을 알게 되었다.

대상	VirtualAddress	R-VirtualAddress	RawAddress
EntryPoint	4199091	4787 (SectionAlignment + 691)	1715
.text	4198400	4096 (SectionAlignment × 1)	1024
.rdata	4202496	8192 (SectionAlignment × 2)	4608
.data	4206592	12288 (SectionAlignment × 3)	7680
.rsrc	4210688	16384 (SectionAlignment × 4)	8192
EntryImport	4203844	9540 (.rdata + 1348)	5956
EntryResource	4210688	16384 (.rsrc)	8192
EntryDebug	4202768	8464 (.rdata + 272)	4880
EntryLoadConfig	4202880	8576 (.rdata + 384)	4992
EntryIAT	4202496	8192 (.rdata)	4608
SectionAlignment = 4096, ImageBase = 4194304			

[표 2-4] RAW, VA, RVA 계산 값

- [표 2-4]와 같은 결과가 나왔기에, 이에 맞게 DataDirectory와 헤더 정보를 수정하는 알고리즘으로 변경하였으나, 제대로 실행되지 않았다.
- 하나의 PE 파일만 분석했기 때문이라 생각되어, 인터넷을 돌며 Section Alignment가 8192인 다른 PE파일을 찾아보았다.

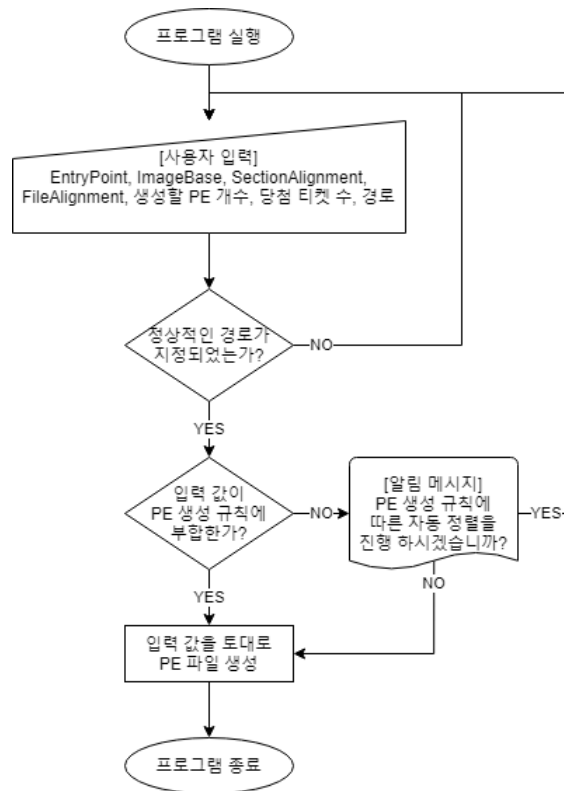
대상	VirtualAddress	R-VirtualAddress	RawAddress
EntryPoint	4821314	627010 (SectionAlignment × 76 + 4418)	619330
.text	4202496	8192 (SectionAlignment × 1)	512
.rsrc	4825088	630784 (SectionAlignment × 4)	619520
.reloc	5439488	1245184 (SectionAlignment × 2)	1229824
EntryImport	4821232	626928 (0)	619248
EntryResource	4825088	630784 (.rsrc)	619520
EntryDebug	4820920	626616 (0)	618936
EntryIAT	4202496	8192 (.text)	512
SectionAlignment = 8192, ImageBase = 4194304			

[표 2-5] Section Alignment가 8192인 PE파일

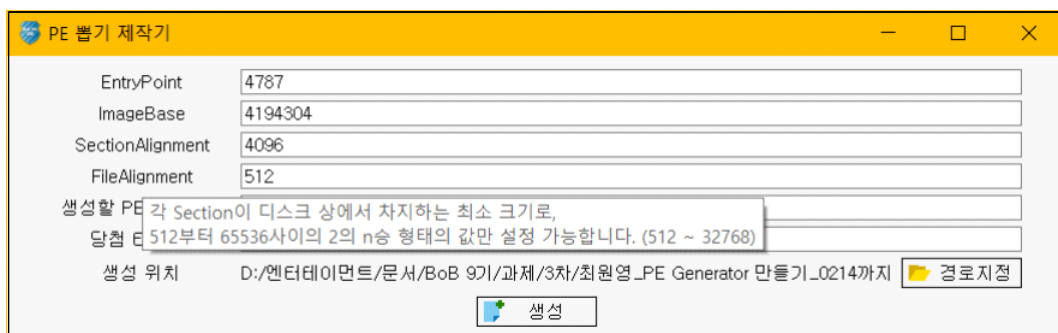
- 확실한건 EntryResource의 RVA값은 .rsrc와 동일하다는 것뿐이었다.

2-3) GUI 제작

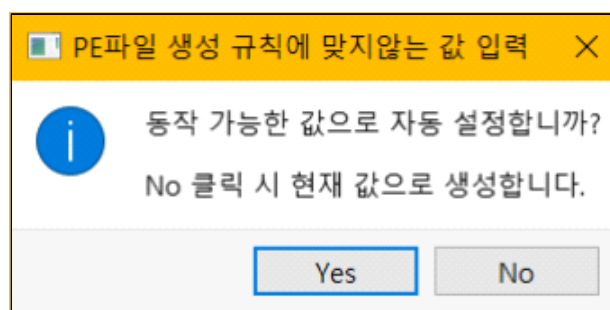
- 시간이 많지 않았기에, 일단 메모리 관련 변수 수정 기능은 후일로 미뤄 두고, 먼저 GUI를 제작해 보기로 하였다.



[그림 2-6] 프로그램 동작 알고리즘



[그림 2-7] 완성된 GUI

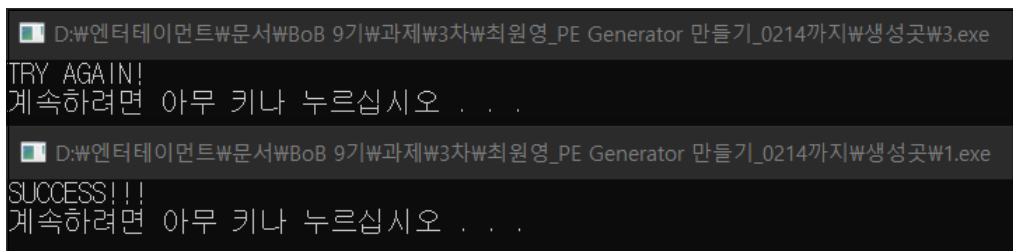


[그림 2-8] 자동 정렬 기능

« 최원영_PE Generator 만들기_02... > 생성곳

이름	수정한 날짜	유형
1.exe	2021-01-19 오전 2:28	응용 프로
2.exe	2021-01-19 오전 2:28	응용 프로
3.exe	2021-01-19 오전 2:28	응용 프로

[그림 2-9] 옵션에 맞게 자동 생성된 PE 파일



[그림 2-10] 정상적으로 출력된 당첨 메시지와 꽁 메시지

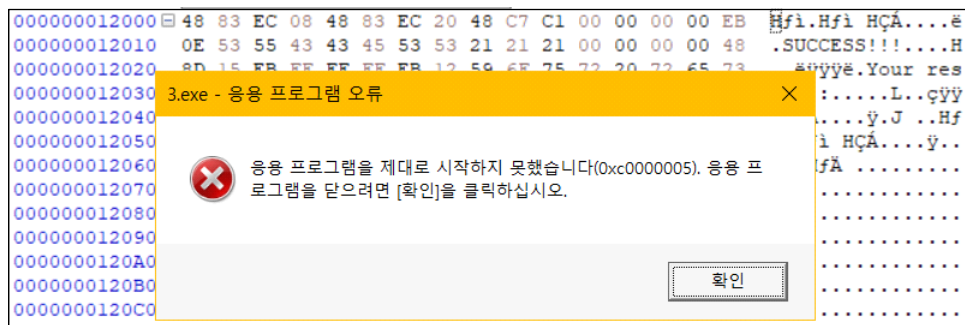
2-4) Entrypoint, ImageBase, Section Alignment 재도전

- Make Break Make... 안되면 다 부수고 처음부터 만들라는 개발자들 사이 우스갯소리가 있다.
- 그리하여 샘플 EXE 파일을 다른 것으로 교체하고, 코드를 다시 제작하였다.
- 코드를 교체하면서 지금까지 해 왔던 32비트 PE파일 구조 대신, 64비트 PE+ 파일 구조를 기반으로 제작하였으며, 대부분의 RVA값을 개발하기 쉬운 주소로 변경하였다.

50 45 00 00 64 86 04 00 0F A4 80 90 00 00 00 00	PE..dt...x€....
00 00 00 00 F0 00 2F 00 0B 02 00 00 00 00 008./.....
00 00 00 00 00 00 00 00 00 10 00 00 00 00 00[].....
00 00 87 00 00 00 00 00 00 10 00 00 00 02 00	..#.....
00 00 00 00 00 00 00 00 04 00 00 00 00 00 00

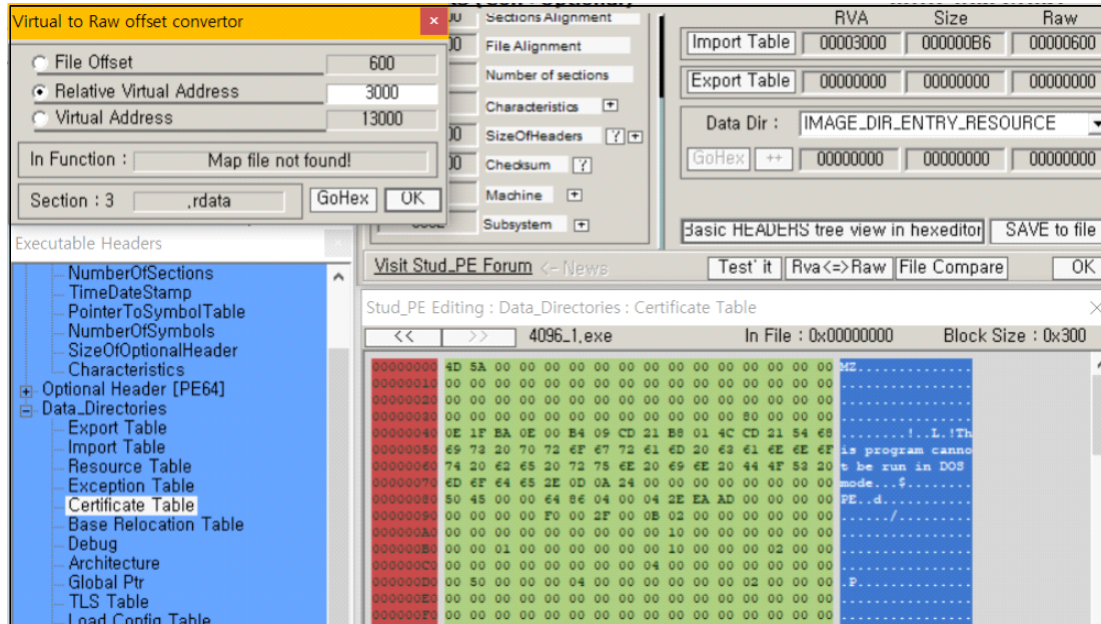
[그림 2-11] EntryPoint값이 SectionAlignment와 동일한 PE파일을 생성한 모습

- 다시 제작하는 과정에서 ImageBase는 반드시 65536의 배수여야함을 알게 되었으며, 이를 코드에 적용시켰더니, ImageBase 변경 기능도 정상 동작하게 되었다.
- 남은 것은 EntryPoint와 SectionAlignment 수정이었는데, EntryPoint를 SectionAlignment와 동일하게 잡아 주었음에도 계속 에러가 났다.
- 이에 직접 메모리도 열어 계산을 해 보았지만, EntryPoint, SectionAlignment 어느 하나 잘못 된 부분이 없었다.



[그림 2-12] 메모리상 정확한 EntryPoint에 도달하였으나, 에러가 난 모습

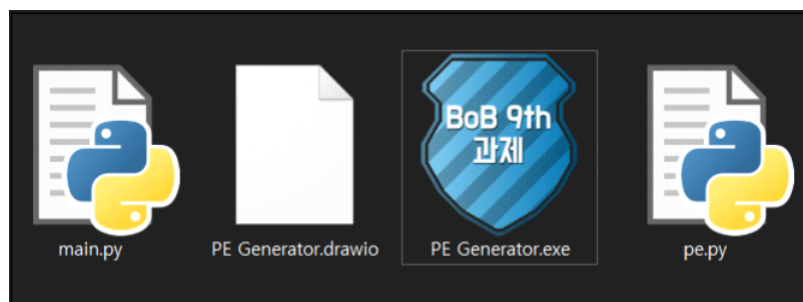
- 0xC0000005 에러는 메모리 관련 오류가 발생했을 때 나타나지만, SectionAlignment와 EntryPoint에는 문제가 없었으므로 이외의 요소를 살펴보기로 하였다.
- Stud PE를 이용하여 SectionAlignment가 8192인 경우와, 4096인 경우를 비교 해 보았으나, 모두 정확한 주소를 지목하고 있었기에, 이유를 알 수 없었다.



[그림 2-13] Stud_PE를 이용하여 분석하는 모습

2-5) Pyinstaller로 Python스크립트를 프로그램으로 추출하기

- Pyinstaller를 설치 한 다음 pyinstaller --onefile --noconsole --icon=icon.ico main.py 명령을 입력하여 아이콘을 부착한 하나의 실행 파일로 만들었다.

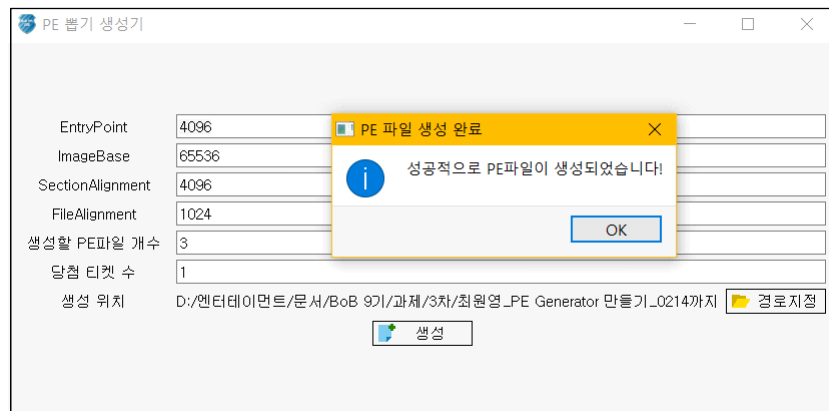


[그림 2-14] 실행파일 생성에 성공한 모습

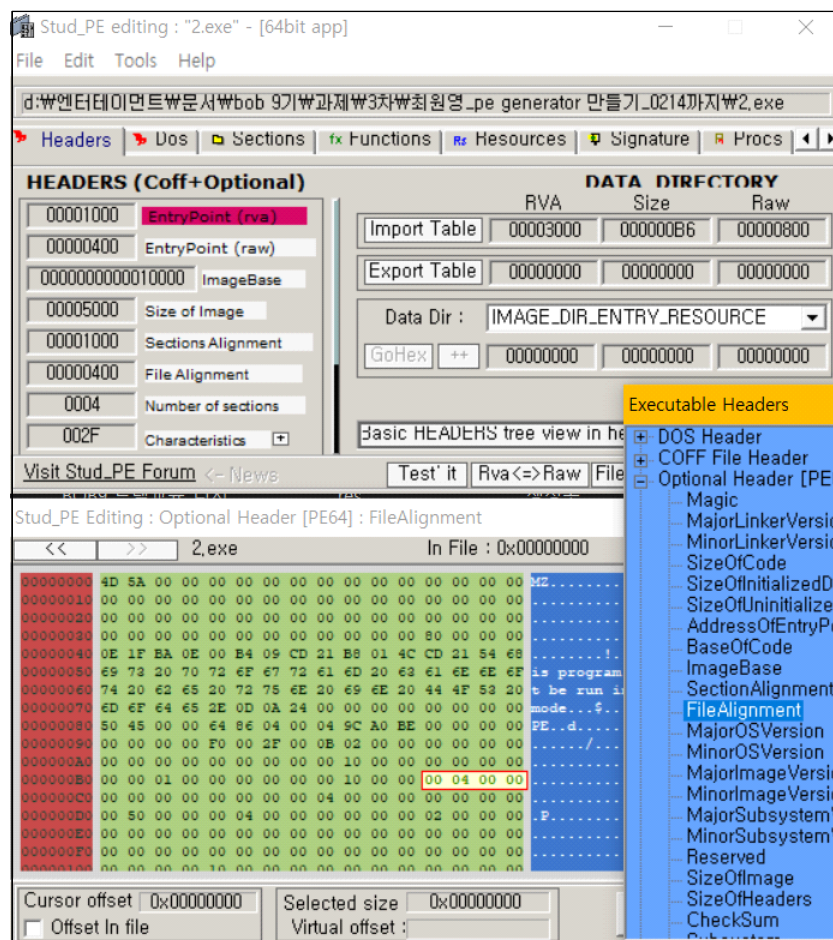
3. 결과

- Entrypoint, ImageBase, Section Alignment, File Alignment값을 입력 받고, 입력 받은 대로 PE+(64비트) 파일 생성에 성공하였다.
- text, data, rdata, rsrc 4개의 Section Table을 추가하는데 성공하였다.
- GUI 화면 입히기에 성공하였다.

- 사용자가 지정한 개수의 PE 파일을 임의 이름으로 생성하는데 성공하였다.



[그림 3-1] 잘 작동하는 모습



[그림 3-2] 입력 받은 대로 PE 파일이 잘 만들어 진 모습