SECURITY FACTORY

리버싱 이 정도는 알아야지

Chapter 02. PE File Format

http://securityfactory.tistory.com

목 차

Chapter 02

PE File Format

SECURITY FACTORY

1. PE File Format 이해하기

PE File Format 은 “Windows 실행 파일의 구성 방식”입니다. Windows 에서 동작하는 실행 파일은 어떻게 구성되어야 하는지, 그 원칙을 정의해 놓은 것이죠. 우리가 흔히 얘기하는 응용 프로그램과 실행 파일을 포함해서, Windows 에서 실행 가능한 모든 파일이 PE 파일 구성에 맞게 작성되어 있습니다.

아직 잘 이해가 안되나요? 조금 더 쉽게 얘기해 봅시다. 인간은 크게 머리와 몸, 팔, 다리로 구성되어 있습니다. 세부적으로 살펴보면 더 많은 특징들이 있고, 이런 모든 조건에 충족하는 동물을 인간이라고 부릅니다. 그런데 여기서 “인간은 왜 이렇게 만들어졌을까?”를 따지기 시작하면, 정확한 이유는 누구도 모릅니다. 진화의 관점에서 논하기도 하고, 환경적인 요인을 들기도 하지만 하나의 가능성에 불과하죠. 그래서 이러한 논쟁이 가끔은 무의미하게 느껴지기도 합니다. 그냥 조물주가 세상을 만드는 과정에서 나름의 합리적인 기준으로 인간의 몸을 구성했고, 우리는 그에 따르는 것일 뿐입니다.

PE 파일도 같습니다. PE File Format 은 Microsoft 가 운영체제를 설계하는 과정에서, 실행 파일의 합리적이고 효율적인 구성을 고민하고, 선택한 방식입니다. 그리고 우리는 Windows 운영체제를 사용하는 소비자의 입장에서 그 규칙에 따라야 합니다. 그렇다면 PE 파일은 언제, 어떻게 만들어 질까요?

보통 PE 파일을 만들 때, VisualStudio 와 같은 개발 도구를 사용합니다. 개발자가 코드를 작성하면, 개발도구의 컴파일러가 PE File Format 에 맞게 실행파일로 만들어 주죠.

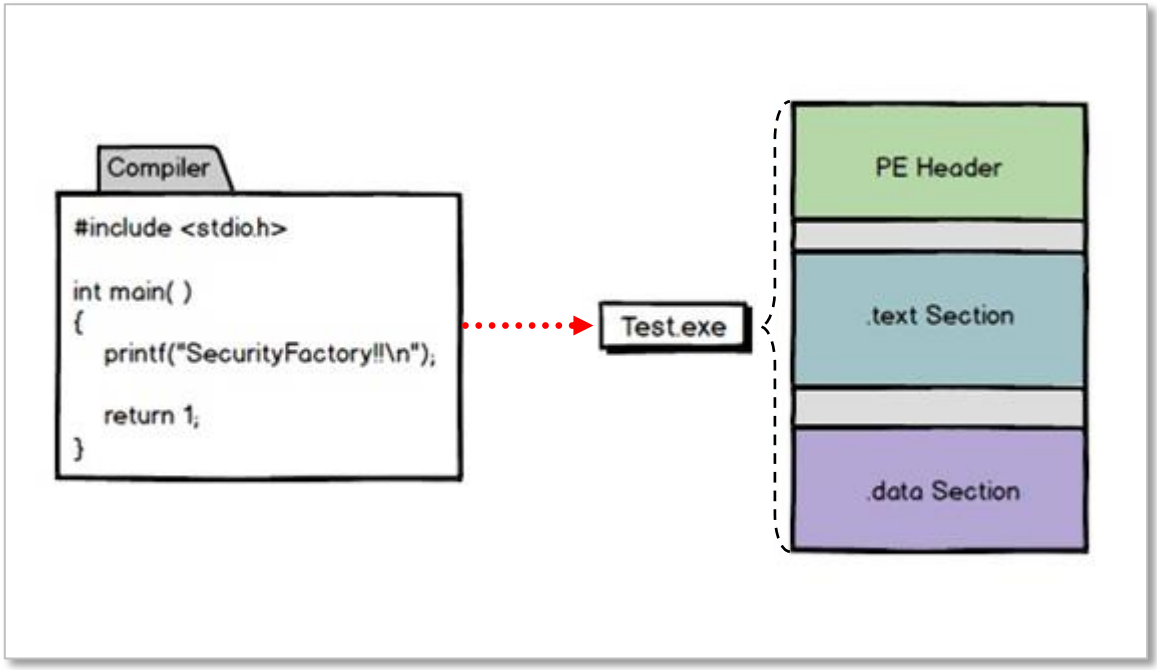


그림 1-1 Test.exe 컴파일

그러면 직접 Test.exe 를 컴파일 해봅시다. 간단하게 “printf(“SecurityFactory!!₩n”);” 코드로 진행하겠습니다.

※ Note

블로그: http://bitly.kr/udWs
페이스북: http://bitly.kr/OrHQ

블로그: http://bitly.kr/udWs
페이스북: http://bitly.kr/OrHQ

※ 참고
컴파일 과정과 관련된 이론을 살펴보면, 대략 “전처리기 → 컴파일러 → 어셈블러 → 링커”의 과정을 거칩니다. 본 문서는 컴파일 과정을 공부하기 위한 것이 아니라, PE 파일을 이해하고 어떻게 공부해야 하는지 알기 위한 것 입니다. 그리고 그에 가장 적합하게 쉽게 풀어서 설명한 것이니 참고하시기 바랍니다.

2. PE 헤더 훑어보기

리버싱 공부를 시작하는 분들이 처음부터 좌절하는 이유는 PE File Format이 버티고 있기 때문입니다. 의욕은 앞서는데 그 의욕을 잠재우기 충분할 만큼 지루하죠. 알 수 없는 값들을 보고 있으면, 내가 지금 뭘 하고 있는 건지, 이게 리버싱인지 의문이 들기도 합니다. 실제로 PE 파일이 무엇이고, 왜 존재하는지 모르고 접근하면 그냥 영어 단어 외우기 밖에 안됩니다. 나무를 보기 전에 숲을 보라고 했습니다. ‘1. PE File Format 이해하기’를 충분히 숙지하기 바랍니다.

PE 헤더는 작은 구조체들이 모여서 만들어진 데이터 덩어리입니다. 구조체는 또 뭐냐고요? 구조체는 값을 담을 수 있는 주머니입니다. 단, 변수와 다르게 크기가 정해져 있고, 그 안에 약속된 값이 들어갈 수 있도록 구획이 나뉘어있습니다.

왜 나눠 놨을까요? 효율성을 위해서입니다. 방을 어지른 상태로 두는 것보다 정해진 자리에 정리해 놓으면, 물건을 찾을 때 더 편하고 쉽겠죠? 같은 이치입니다.

PE 헤더는 크게 DOS Header와 DOS Stub, NT Header, Section Headers로 구성되어 있습니다.

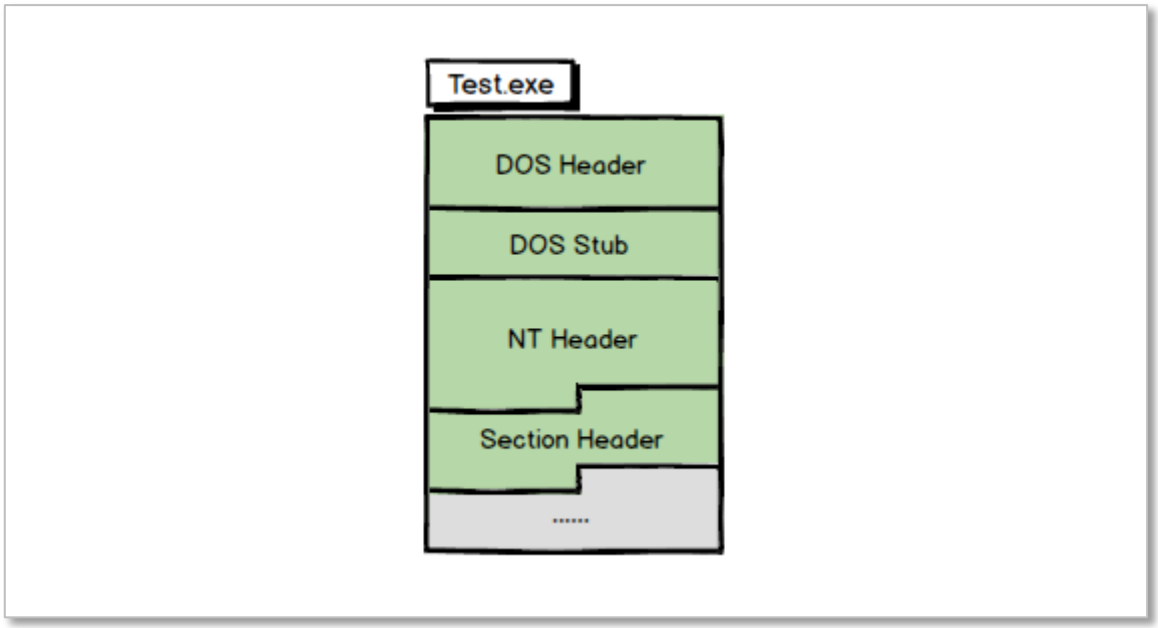


그림 2-1 PE 헤더

각 항목의 의미와 역할입니다. 파일 실행과 관련된 중요 정보들은 대부분 NT Header에 기록되어 있습니다.

DOS Header	DOS Header는 DOS와 호환성을 위해서 만들었습니다. 파일의 처음에 위치하고, 0x40 크기를 가집니다.
DOS Stub	PE 파일이 MS-DOS에서 실행 될 경우, 화면에 출력될 메시지와 코드가 기록되어 있습니다. DOS Stub은 옵션이기 때문에 파일 실행에 영향이 없습니다. 크기가 일정하지 않고, 없어도 되는 영역입니다.
NT Header	파일 실행에 필요한 전반적인 정보를 가지고 있습니다. 0xF8 크기를 가집니다.
Section Header	각 섹션의 속성 정보를 가지고 있습니다.

표 2-1 PE 헤더 주요 항목 정보

조금 더 세부적으로 들어가 봅시다. 실행에 필요한 최소한의 값들만 살펴보겠습니다. 가볍게 읽고, 이해하면 됩니다. (각 구조체의 원형을 살펴보고 싶다면, “보충 자료”를 참고하기 바랍니다.)

2.1. DOS Header

PE File Format은 당시 널리 사용되던 DOS 파일에 대한 하위 호환성을 고려해서 만들었습니다. 그 결과,

PE 헤더의 제일 앞부분에는 DOS Header가 존재합니다.

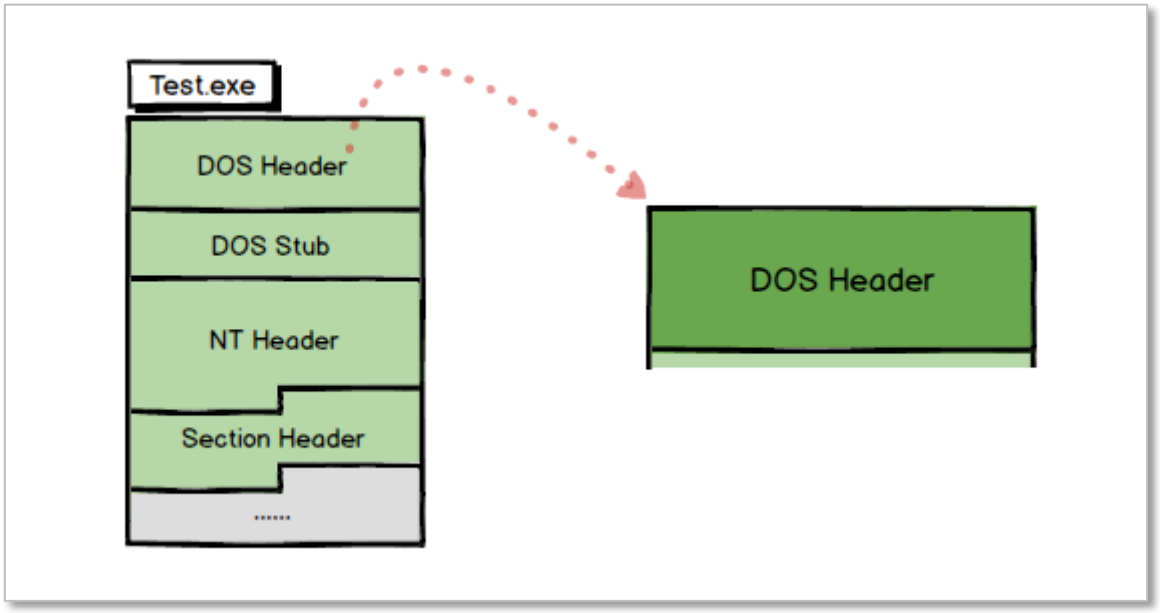


그림 2-2 DOS 헤더

DOS Header는 IMAGE_DOS_HEADER 구조체에 값을 채워서 만든 것입니다. 크기는 0x40입니다.

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	**DOS Sign**															
00000010																
00000020																
00000030													**NT 헤더 오프셋**			

DOS Header에서 우리가 알아야 할 값들은 다음과 같습니다.

항목	크기	설명
DOS Signature	2byte	DOS 헤더는 PE 파일의 가장 처음에 위치하고, “MZ(0x4D5A)”라는 시작 값을 가진다.
NT Header Offset	4byte	NT 헤더의 시작 지점에 대한 정보를 가지고 있다.

표 2-2 DOS 헤더 주요 항목 정보

2.2. DOS Stub

DOS Stub은 16비트 명령코드와 데이터의 혼합으로 이루어져 있습니다. DOS Stub의 존재여부는 옵션이고, 크기도 일정하지 않습니다. (DOS Stub이 없어도 파일 실행에는 문제가 없습니다.)

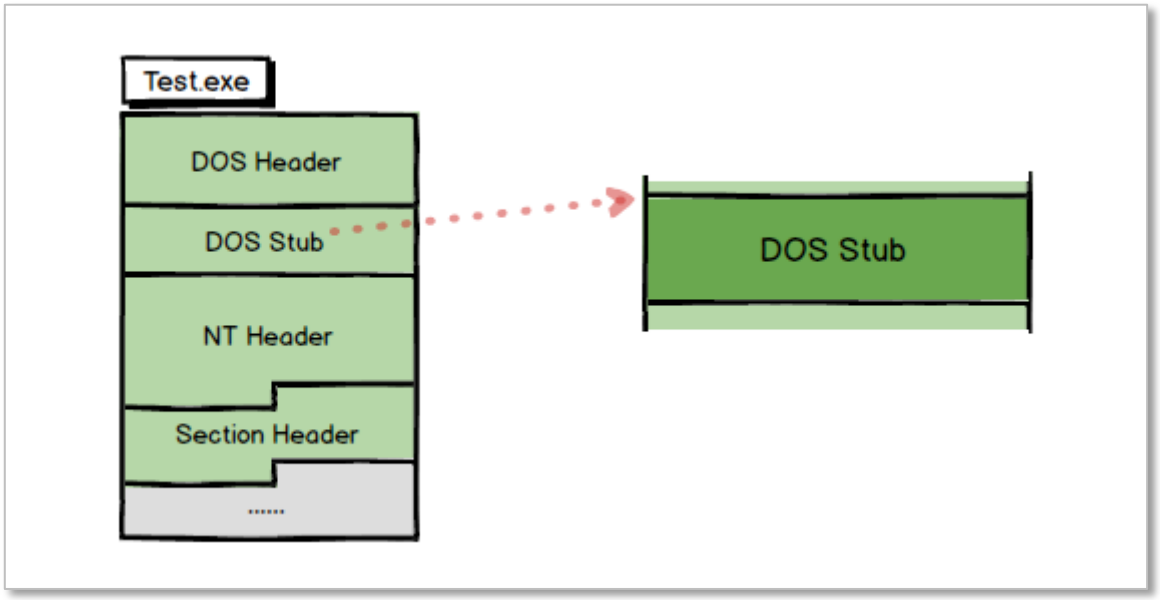


그림 2-3 DOS Stub

2.3. NT Header

NT Header는 파일 실행에 필요한 전반적인 정보를 가지고 있습니다.

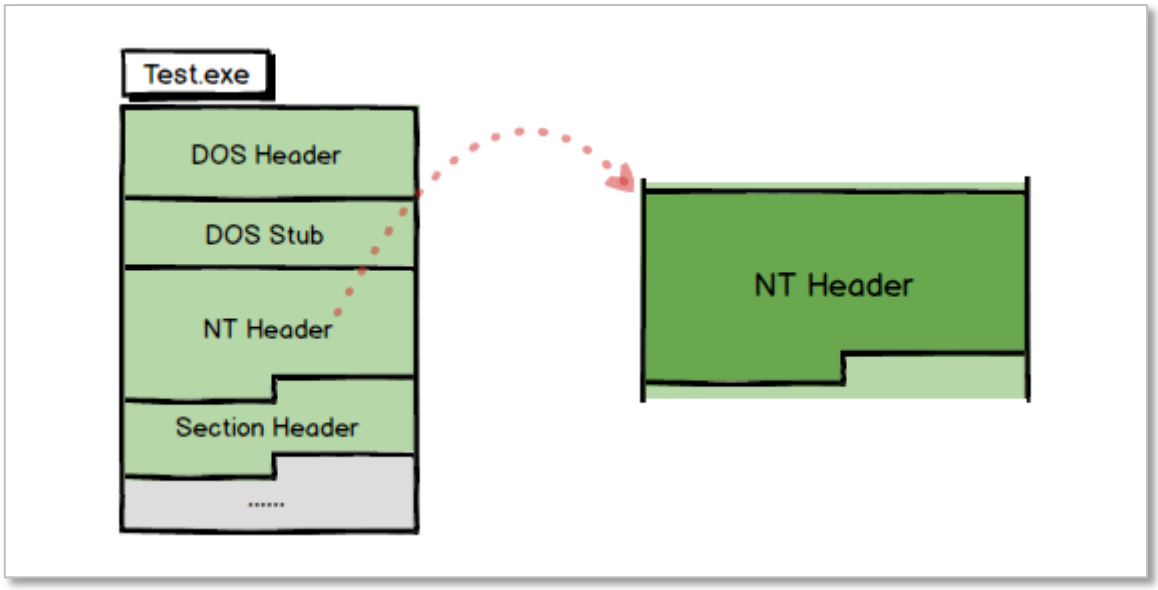


그림 2-4 NT 헤더

총 3개의 멤버로 구성되어 있는데, 특이하게도 2, 3번 멤버가 각각이 하나의 작은 구조체입니다. (PE 헤더는 작은 구조체들이 모여서 만들어진 데이터 덩어리이라고 했습니다.)

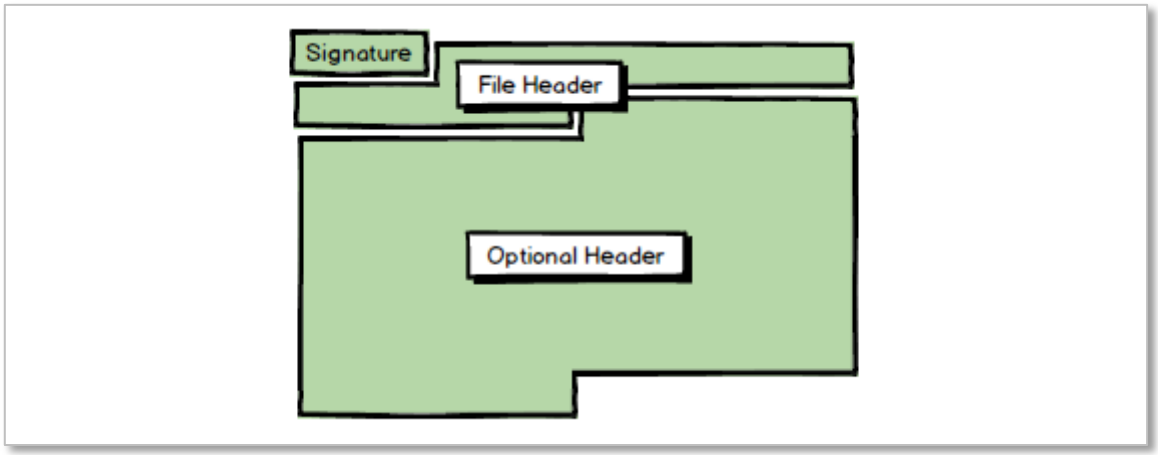
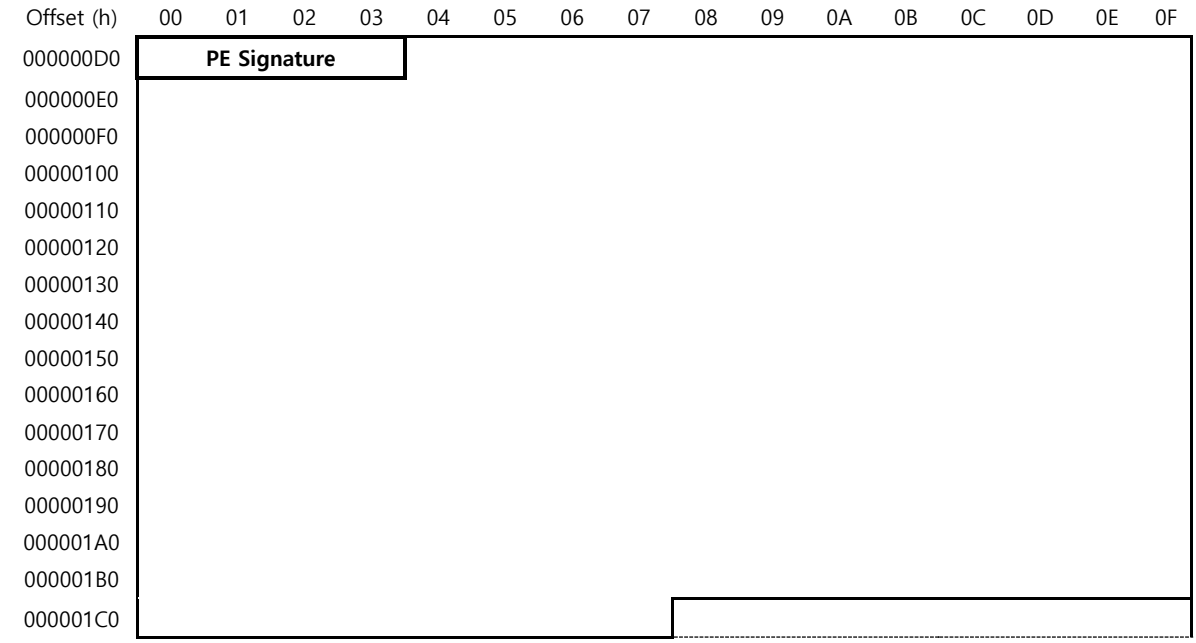


그림 2-5 NT 헤더를 구성하는 멤버

2.3.1 PE Signature

NT 헤더의 첫 번째 값은 PE 시그니처입니다. “이 파일은 PE 파일이야.”라고 명시해놓은 거죠. PE 로더는 이 값을 보고, 올바른 PE 파일인지 확인합니다.

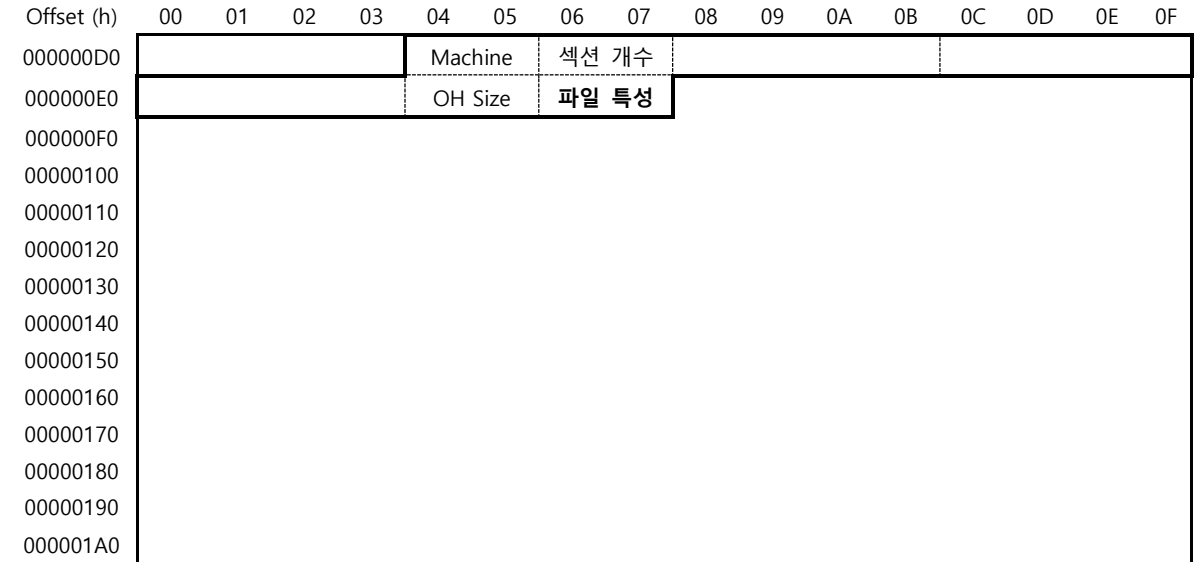


항목	크기	설명
PE Signature	4byte	“0x50450000” 값을 가진다.

표 2-3 PE 시그니처 정보

2.3.2 File Header

File Header는 파일의 물리 정보를 가지고 있습니다. IMAGE_FILE_HEADER 구조체에 값을 채워서 만들어 집니다.



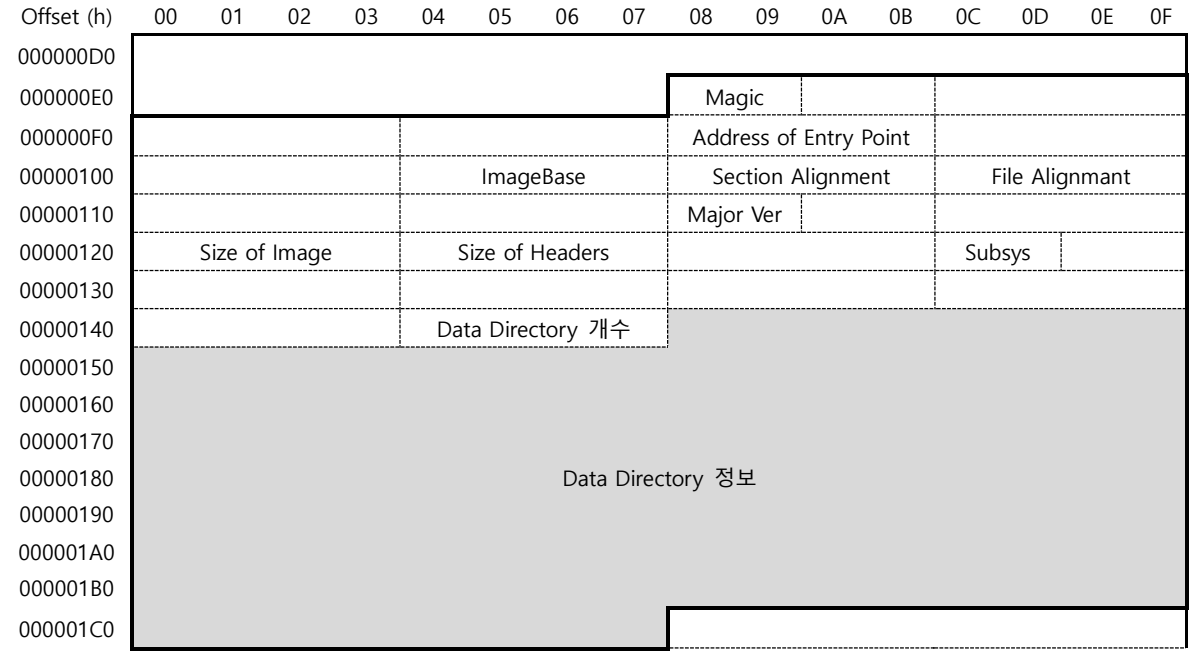
File Header에서 우리가 알아야 할 값들은 다음과 같습니다.

항목	크기	설명
Machine	2byte	파일이 실행될 UPU 타입 정보를 가지고 있다. - Intel 386 → 0x014C - Intel 64 → 0x0200
Number of Sections	2byte	섹션 개수 정보를 가지고 있다.
Size of Optional Header	2byte	Optional Header의 크기 값을 가지고 있다. 일반적으로 0xE0 값을 가진다.
Characteristics	2byte	파일의 속성 정보를 가지고 있다. 이 값으로 EXE와 DLL을 구분한다. - 실행 가능한 파일 → 0x0002 - 32bit 실행 파일 → 0x0100 - DLL 파일 → 0x2000

표 2-4 File 헤더 주요 항목 정보

2.3.3 Optional Header

Optional Header는 파일의 논리 정보를 가지고 있습니다. IMAGE_OPTIONAL_HEADER32 구조체에 값을 채워서 만들어 집니다.



다음은 Optional Header에서 우리가 알아야 할 값 입니다.

항목	크기	설명
Magic	2byte	Optional Header가 IMAGE_OPTIONAL_HEADER32 구조체인 경우 0x010B, IMAGE_OPTIONAL_HEADER64 구조체인 경우 0x020B 값을 가진다.
Address of EntryPoint	4byte	파일의 코드 시작 지점에 대한 RVA 값을 가지고 있다.
Image Base	4byte	파일이 메모리에 올라갈 때, 기준이 되는 주소 값을 가지고 있다. 기본적으로 EXE는 0x00400000, DLL은 0x10000000 값을 가진다.
Section Alignment & File Alignment	4byte	PE 파일의 Body 부분은 섹션으로 나뉘어져 있다. FileAlignment는 파일에서 섹션의 최소 단위를 나타내고, SectionAligment는 메모리에서 섹션의 최소 단위를 나타낸다. 파일과 메모리에서 섹션 크기는 반드시 FileAlignment와 SectionAligment의 배수가 되어야 한다.
Major O/S Version	2byte	-
Size of Image	4byte	메모리에 올라간 파일 데이터의 크기 값을 나타낸다.
Size of Headers	4byte	DOS Header, NT Header, Section Header를 합친 총 크기 값을 나타낸다.
Subsystem	2byte	-
Number of Data Directories	4byte	Data Directory의 개수를 나타낸다.
Data Directories	-	Data Directories는 Number of Data Directories 필드의 개수만큼 활성화된다. 일반적으로 0x10개의 정보를 가진다.

표 2-5 Optional 헤더 주요 항목 정보

[그림 2-6]은 Sample 01.exe의 Data Directories입니다. 일반적으로 PE 파일은 이와 같은 Data Directories 구성을 가집니다. 우리는 여기서 EXPORT Table과 IMPORT Table의 크기와 위치 정보가 Data Directories에 기록되어 있다는 정도만 알면 됩니다.

00000148	00000000	RVA	EXPORT Table
0000014C	00000000	Size	
00000150	0000543C	RVA	IMPORT Table
00000154	0000003C	Size	
00000158	00000000	RVA	RESOURCE Table
0000015C	00000000	Size	
00000160	00000000	RVA	EXCEPTION Table
00000164	00000000	Size	
00000168	00000000	Offset	CERTIFICATE Table
0000016C	00000000	Size	
00000170	00000000	RVA	BASE RELOCATION Table
00000174	00000000	Size	
00000178	00000000	RVA	DEBUG Directory
0000017C	00000000	Size	
00000180	00000000	RVA	Architecture Specific Data
00000184	00000000	Size	
00000188	00000000	RVA	GLOBAL POINTER Register
0000018C	00000000	Size	
00000190	00000000	RVA	TLS Table
00000194	00000000	Size	
00000198	00000000	RVA	LOAD CONFIGURATION Table
0000019C	00000000	Size	
000001A0	00000000	RVA	BOUND IMPORT Table
000001A4	00000000	Size	
000001A8	00005000	RVA	IMPORT Address Table
000001AC	000000A8	Size	
000001B0	00000000	RVA	DELAY IMPORT Descriptors
000001B4	00000000	Size	
000001B8	00000000	RVA	CLI Header
000001BC	00000000	Size	
000001C0	00000000	RVA	
000001C4	00000000	Size	

그림 2-6 Sample 01.exe의 Data Directories

2.4. Section Header

Section Header는 PE 파일이 가지는 섹션 수만큼 존재합니다.

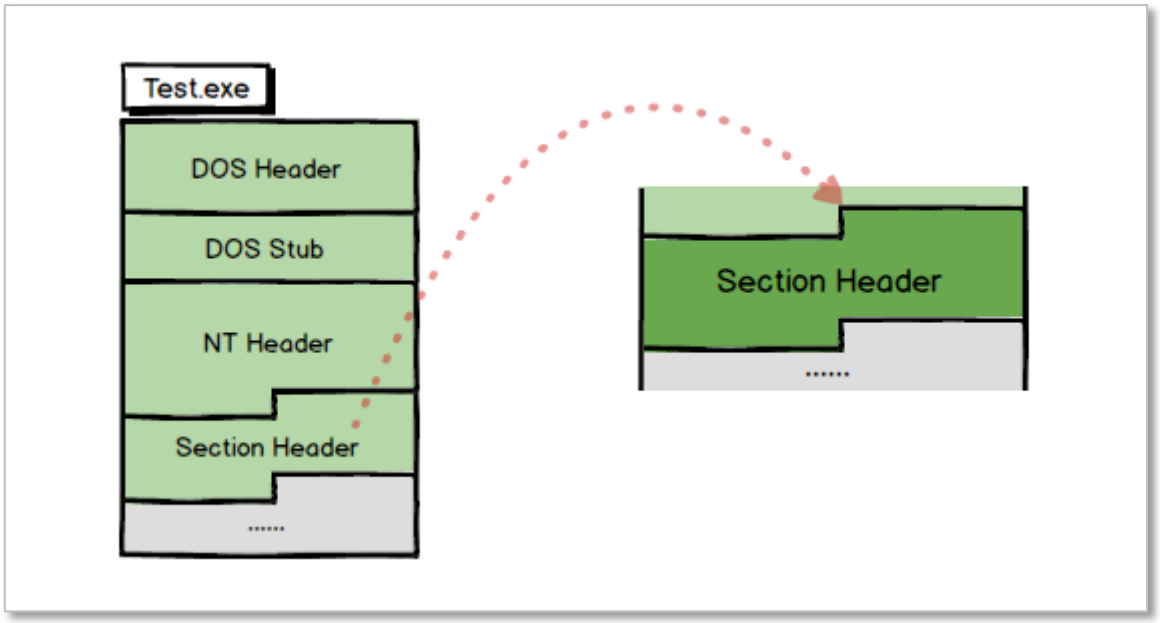
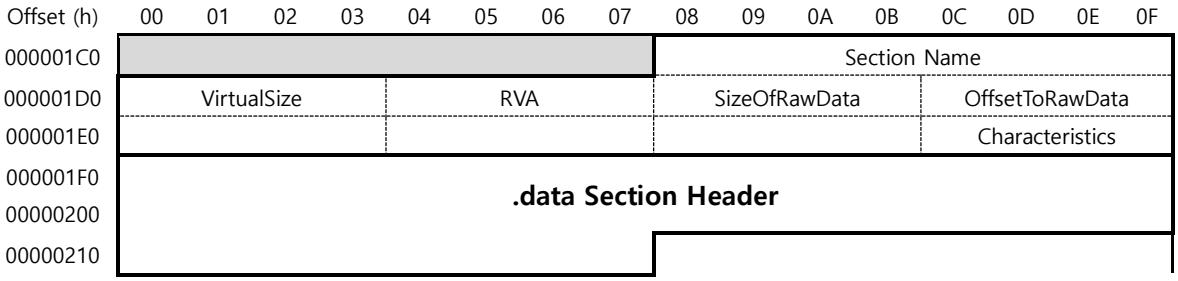


그림 2-7 Section 헤더



다음은 Section Header에서 우리가 알아야 할 값 입니다.

항목	크기	설명
Section Name	8byte	섹션의 이름 정보를 가지고 있다.
VirtualSize	4byte	메모리에서 섹션이 차지하는 크기이다.
RVA(VirtualAddress)	4byte	메모리에서 섹션의 시작 주소이다.
SizeOfRawData	4byte	파일에서 섹션이 차지하는 크기이다.
OffsetToRawData	4byte	파일에서 섹션의 시작 주소이다.
Characteristics	4byte	섹션의 속성 정보를 가지고 있다.

표 2-6 Section 헤더 주요 항목 정보

※ VA(Virtual Address) & RVA(Relative Virtual Address)
특정 값의 위치 정보를 표현할 때, OffSet(파일에서의 위치)과 VA(메모리에서의 위치), RVA(메모리에서의 상대 위치)를 사용합니다. 여기서 RVA는 파일이 메모리에 올라간 시점에서, 파일의 시작 주소를 기준으로 떨어져 있는 상대 위치입니다. 그래서 VA는 Image Base 값과 RVA 값을 더하면 됩니다.

2.5. IAT와 EAT

IAT(Import Address Table)와 EAT(Export Address Table)에 대한 설명이 필요해 보입니다. Windows 운영체제는 EXE가 DLL이라는 비서를 두고 활용할 수 있게 만들어 놓았다고 했습니다. 이를 가능하게 하는 것이 IAT와 EAT 메커니즘입니다.

항목	설명
IAT (Import Address Table)	DLL이 제공하는 함수들 중에서 사용하는 것들에 대한 정보를 기술해 놓은 테이블이다.
EAT (Export Address Table)	DLL 자신이 서비스하는 함수에 대한 정보를 기술해 놓은 테이블이다.

표 2-7 IAT / EAT 개념

무슨 말인지 알 듯하면서도 잘 모르겠네요. 예를 들어 볼까요? 덕구네 집에는 10명의 비서가 있습니다. 비서들은 각자의 역할이 있고, 그에 맞는 업무들도 따로 가지고 있습니다. 그리고 이를 명시해 놓았습니다. 명시된 업무 내용을 보고 필요한 일을 시킬 수 있도록 만들어 놓은 겁니다. 이게 EAT(Export Address Table)입니다.

비서명	업무 번호	업무 내용
비서 A	work 1	집 안 청소를 한다.
	work 2	집 안을 환기 시킨다.
	work 3	집 안을 보수한다.

표 2-8 비서 A의 Export Address Table

비서명	업무 번호	업무 내용
비서 B	work 1	음식 주문을 한다.
	work 2	정원을 관리한다.
	work 3	개를 산책시킨다.
	work 4	날씨를 체크한다.

표 2-9 비서 B의 Export Address Table

그런데 덕구는 10명의 비서가 어떤 업무를 담당하는지 다 외우질 못합니다. 그래서 아침에 일어나면 자신

이 할 일이 뭔지 먼저 확인하고, 그 일에 필요한 비서와 세부적인 업무를 수첩에 기록해 놓습니다. 그래야 일을 하면서 필요할 때 불러다가 요청할 수 있거든요. 이게 IAT(Import Address Table)입니다. EXE가 동작하면서 필요한 DLL과 그 DLL이 제공하는 API 중에서, 필요한 API의 호출 정보를 기록해 놓은 테이블이죠. (사실 EXE 뿐만 아니라 DLL도 IAT를 가집니다만 크게 개의치 마세요.)

수 첩		
[비서 A] work 1	[비서 A] work 3	
[비서 B] work 2	[비서 B] work 3	[비서 B] work 4

표 2-10 덕구의 Import Address Table

다음은 Sample 01.exe의 Beep( ) API 호출 코드입니다. 필자가 말 한 대로라면 Sample 01.exe는 Beep( ) 호출 주소를 자신의 IAT에 기록해 놓았을 겁니다. 그리고 실제 호출은 IAT에 기록되어 있는 Beep( ) 호출 주소를 참고해서 이루어져야 합니다.

```
00401000 ┌$ 68 00030000  PUSH    300              ┌Duration = 768. ms
00401005 │. 68 00020000  PUSH    200              │Frequency = 200 (512.)
0040100A │. FF15 0050400 CALL    DWORD PTR DS:    └Beep
DS:[00405000]=76CF4A71 (kernel32.Beep)
```

그림 2-8 Beep( ) API 호출 코드

확인해볼까요? “CALL DWORD PTR DS:[0x00405000]” 코드는 “0x00405000 주소에 기록되어 있는 4byte 주소 값을 호출한다.”는 의미입니다. 여기서 0x00405000은 IAT 영역이고, 여기에 Beep( ) 호출 주소가 기록되어 있을 것으로 보입니다. 0x00405000 주소로 이동해보니, 역시나 Beep( ) API 호출 주소가 있습니다.

Address	Value	Comment
00405000	76CF4A71	kernel32.Beep
00405004	76CA6B3D	kernel32.GetStringTypeExA
00405008	76CC528C	kernel32.LCMapStringW
0040500C	76CC8EAF	kernel32.GetCommandLineA
00405010	76CB28F1	kernel32.GetVersion
00405014	76CCBBE2	kernel32.ExitProcess
00405018	76CB2BBD	kernel32.TerminateProcess
0040501C	76CBD7A0	kernel32.GetCurrentProcess
00405020	76CD0651	kernel32.UnhandledExceptionFilter
00405024	76CBD75A	kernel32.GetModuleFileNameA

그림 2-9 Sample 01.exe의 Import Address Table

그러면 API 호출 주소를 IAT에 기록하는 건 누구일까요? 이 역시 PE 로더에서 담당합니다. PE 로더는 파일을 메모리에 올리는 과정에서 필요한 DLL을 같이 로드하고, 해당 DLL이 Export 하는 API들 중에서 필요한 API의 호출 주소를 알아낸 다음 IAT에 기록합니다.

※ 참고_DLL의 EAT에서 API 호출 주소를 알아내는 방법
실행 파일이 API를 호출할 때 IAT를 참조하기도 하지만, API의 호출 주소를 직접 알아내는 경우도 있습니다. (이를 직접 호출이라고 합니다.) 그래서 그 과정이 어떻게 되는지 알 필요가 있습니다만, 아직은 이르다고 판단됩니다. 'Step 03'에서 상세히 다루겠습니다.

3. PE 파일 만들기

PE File Format을 효과적으로 공부하고 싶다면, 다음 세가지를 추천합니다. 이론적인 공부보다 직접 만들거나 분석하면서 익혀야 온전히 내 것이 됩니다.

① 바이너리를 채워가면서, 직접 PE 파일을 만든다.
② Virus 파일을 분석하고, 전용백신을 만든다.
③ PEview.exe와 같은 PE Parser를 만든다.

이 중에서 PE File Format을 정리하는 의미로 직접 실행파일을 만들어보겠습니다.

- 준비물: HxD.exe, OllyDBG.exe
- 문서: PE 파일 만들기.pdf

우리가 만들고자 하는 Challenge 01.exe는 비프음과 함께 메시지 박스를 출력합니다. 많이 익숙한 기능이죠? 네, Sample 01.exe와 동일한 기능을 개발 도구를 사용하지 않고 구현해보는 겁니다.

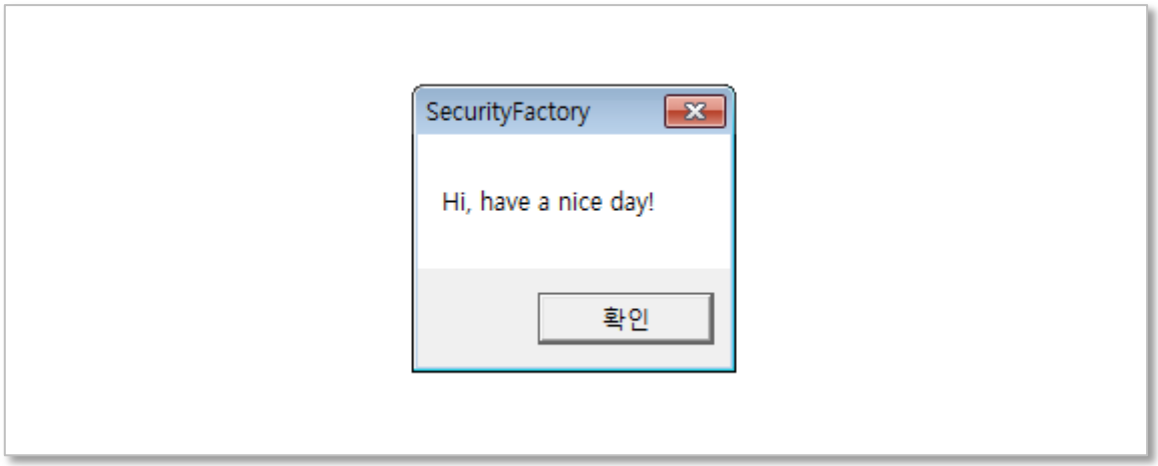


그림 3-1 Sample 01.exe 실행 결과

'PE 파일 만들기.pdf'를 열어보세요. 여기에 우리가 만들 파일이 있습니다. (값을 하나씩 채우다 보면 PE 파일이 만들어질 겁니다.)

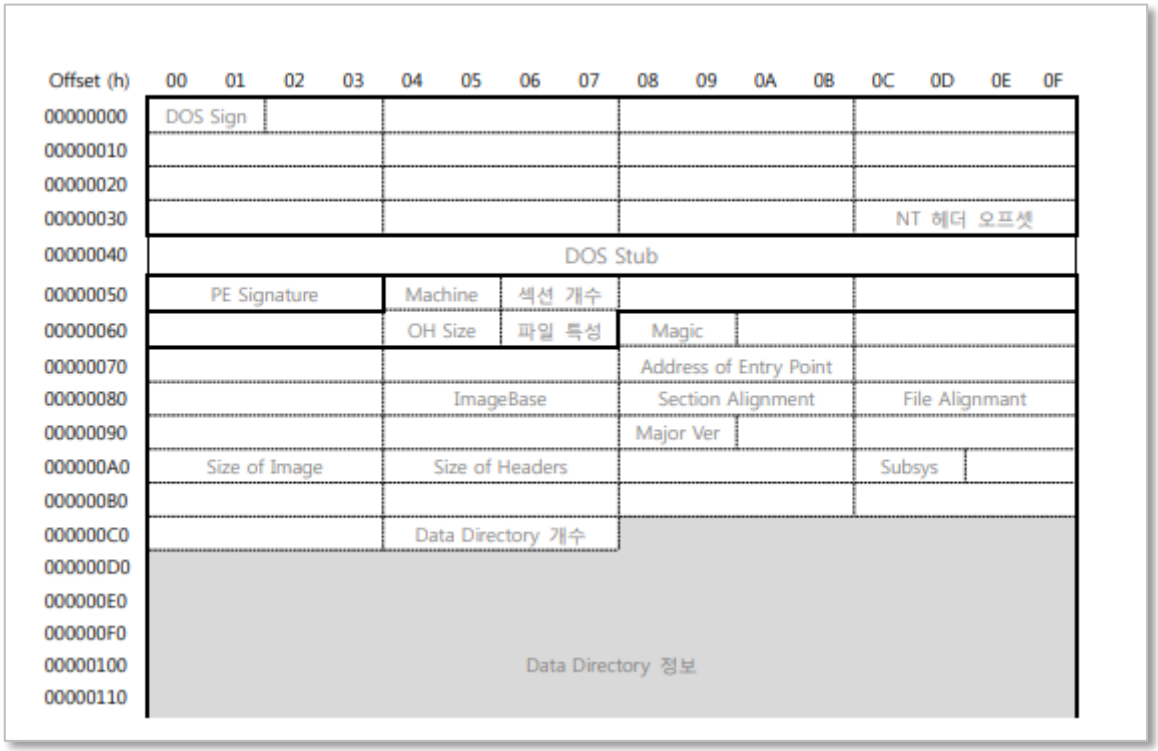


그림 3-2 Challenge 01 만들기.docx

표를 같이 보면서 값을 채워나가겠습니다.

※ Note

블로그: http://bitly.kr/udWs
페이스북: http://bitly.kr/OrHQ

리버싱 이 정도는 알아야지

발행일 | 2018 년 11 월
발행자 | SecurityFactory
페북 주소 | http://bitly.kr/OrHQ
이메일 | itseeyou@naver.com


SECURITY FACTORY
http://securityfactory.tistory.com