Reversing

사이버국방학과 김희연

INDEX

1 리버싱이란 무엇인가

2 기초 지식 (레지스터, 메모리, 리틀엔디언, 어셈블리어, 함수호출규약)

3 실습

What is Reversing?

What is Reversing?

소프트웨어 관점에서의 리버스 엔지니어링: ***'프로그램 분석'***

정적 분석:

파일을 실행하지 않고 겉모습을 관찰하여 분석

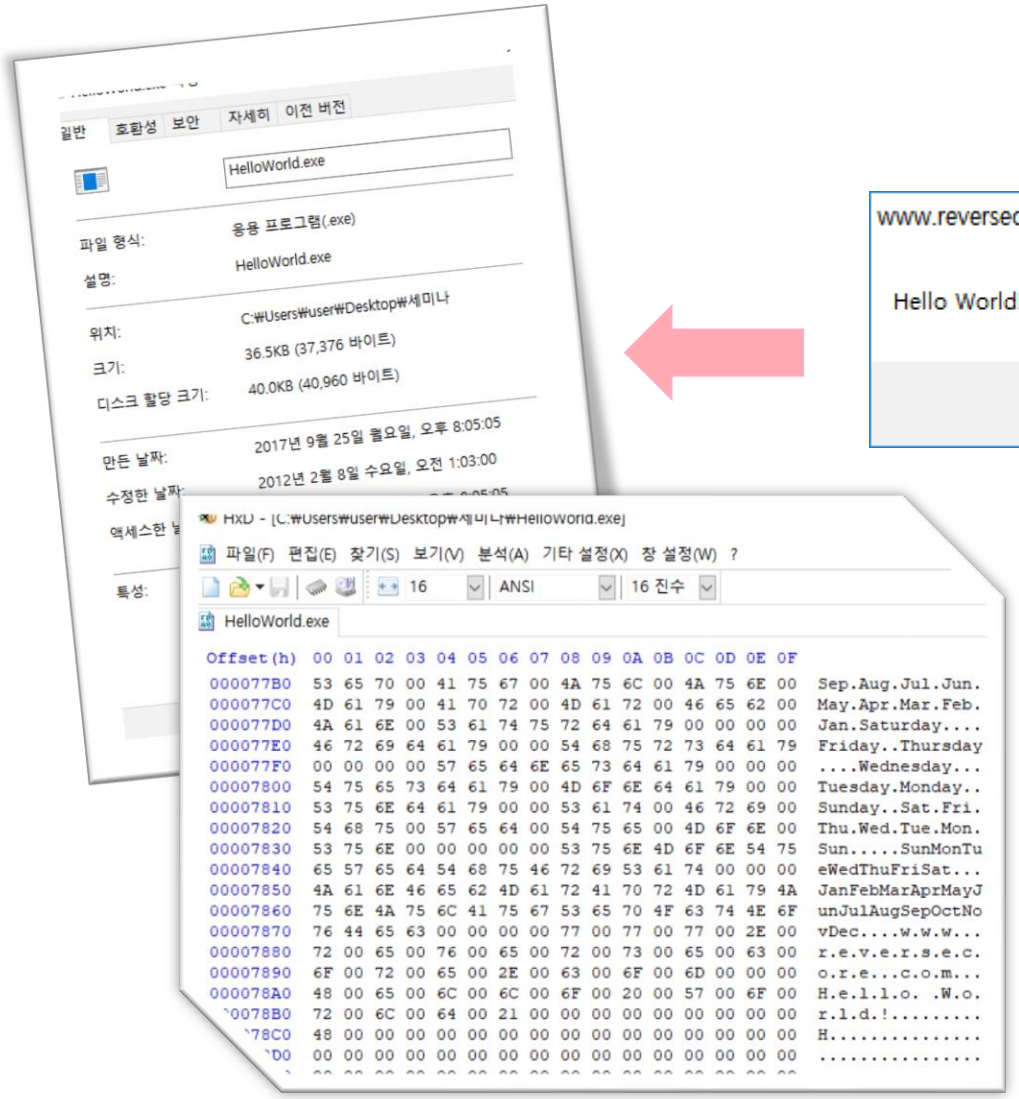
ex) 파일 종류(EXE, DLL, DOC 등), 파일 크기, 파일 헤더 정보, 실행 압축 여부, 내부 문자열....

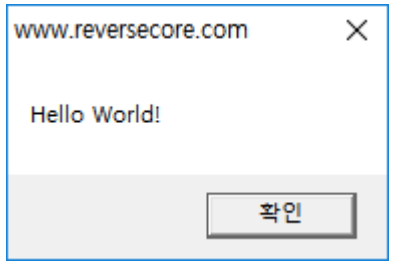
동적 분석:

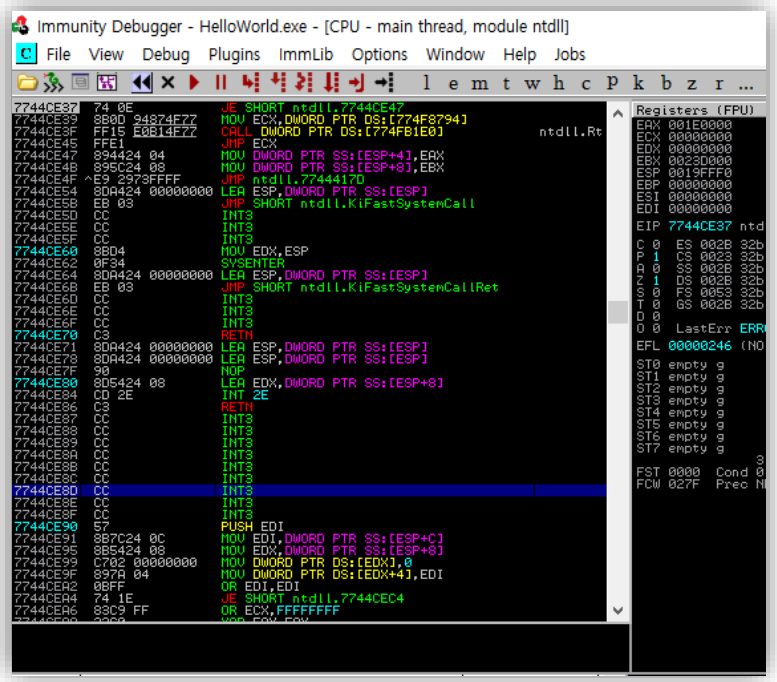
파일을 직접 실행시켜서 그 행위 분석, 코드 흐름과 메모리 상태 분석

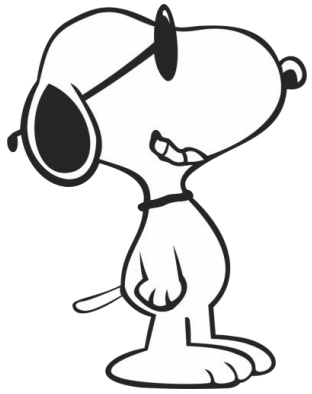
주로 디버거를 이용하여 프로그램 내부 구조와 동작 원리 분석

What is Reversing?









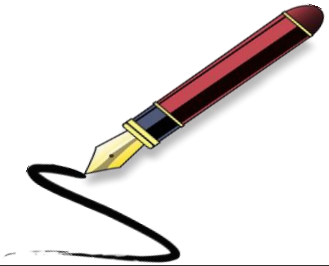
Immunity Debugger

Background

Source Code -> Assembly Code -> Binary Code

```
#include <stdio.h>

int main(void){
   printf("Hello World!");
}
```



```
push        ebp
mov         ebp, esp
movzx       ecx, [ebp+12]
pop         ebp
lea         eax, [edx]
add         eax, edx
shr         eax, 8
```

```
10110111011010101111
00011010101111000102
01000111100110101013
11001101110110111014
00010001110101010005
11000101010100010016
```

Background : register

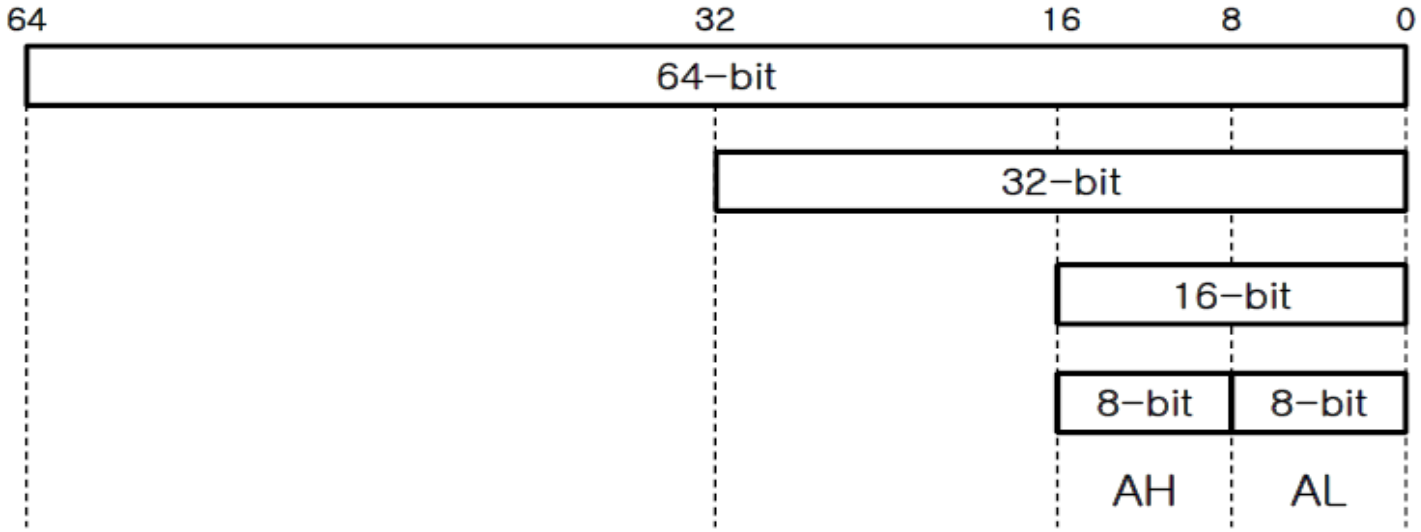
레지스터: CPU 내부에 존재하는 다목적 저장 공간 // 메모리는 RAM!

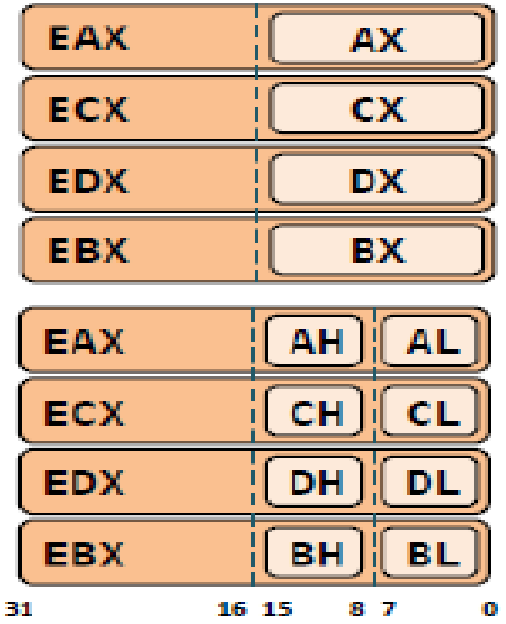
범용 레지스터(General Purpose Register)

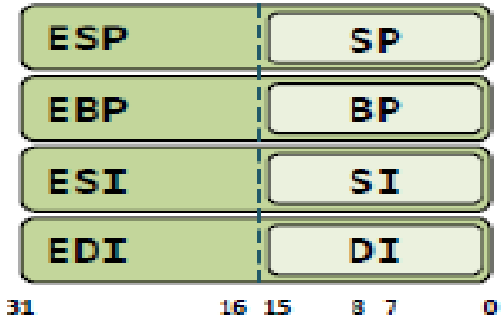
분류	이름	역할	분류	이름	역할
기본연산	EAX	산술 논리 연산 함수 반환 값	메모리 이동	ESI	데이터 복사/비교 시 Source Data의 주소
	EBX	배열의 인덱스 값		EDI	데이터 복사/비교 시 Destination Data의 주소
	ECX	반복문 카운터	스택 관리	ESP	스택 프레임 최상단 주소(가장 낮은 주소)
	EDX	부호 확장 명령		EBP	스택 프레임 최하단 주소 (가장 높은 주소)
실행	EIP	다음 실행할 명령어 주소 저장			

Background : register

- ✓ 8-bit: AH, AL
- ✓ 16-bit: AX
- ✓ 32-bit: EAX
- ✓ 64-bit: RAX







Background : memory

코드 영역

: 실행되는 프로그램의 코드가 저장되는 메모리 공간

데이터 영역

: 전역변수와 정적변수의 값이 저장되는 메모리 공간

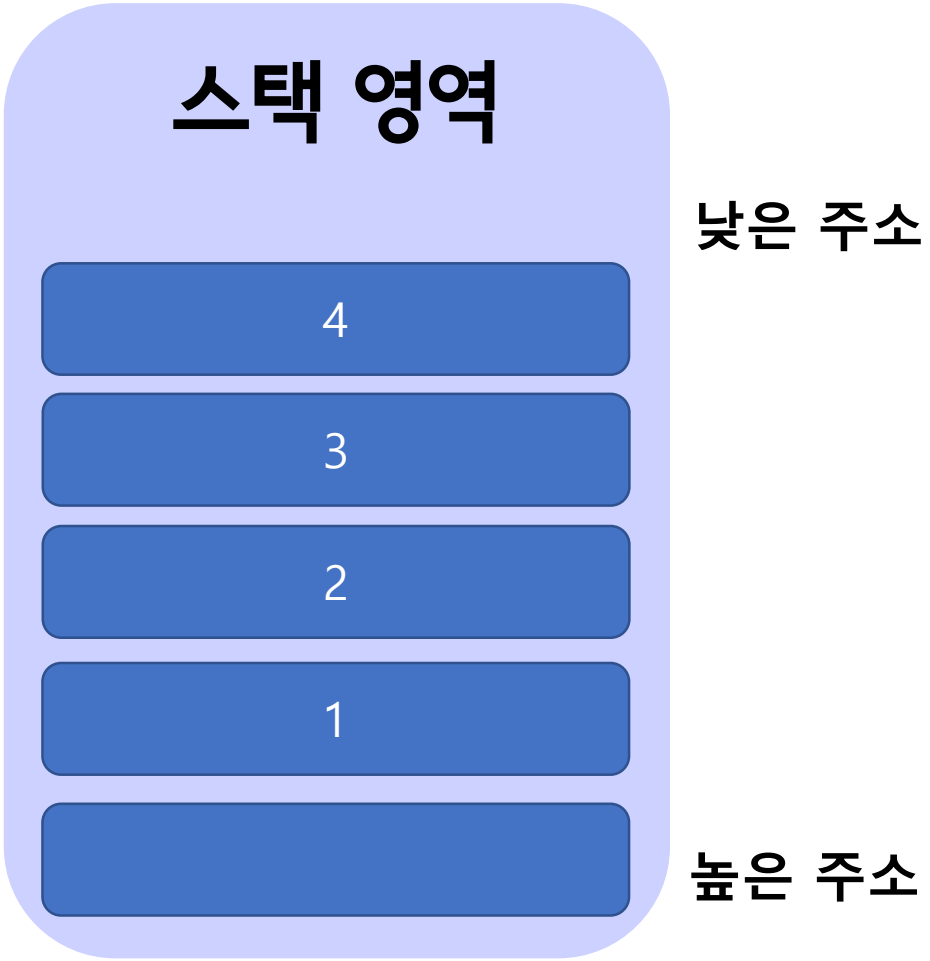
힙 영역

: 사용자가 원하는 시점에 메모리 할당(malloc), 소멸(free)
하도록 할 수 있는 변수들이 할당되는 메모리 공간

스택 영역

: 지역변수와 매개변수 값이 저장되는 메모리 공간

Background : stack



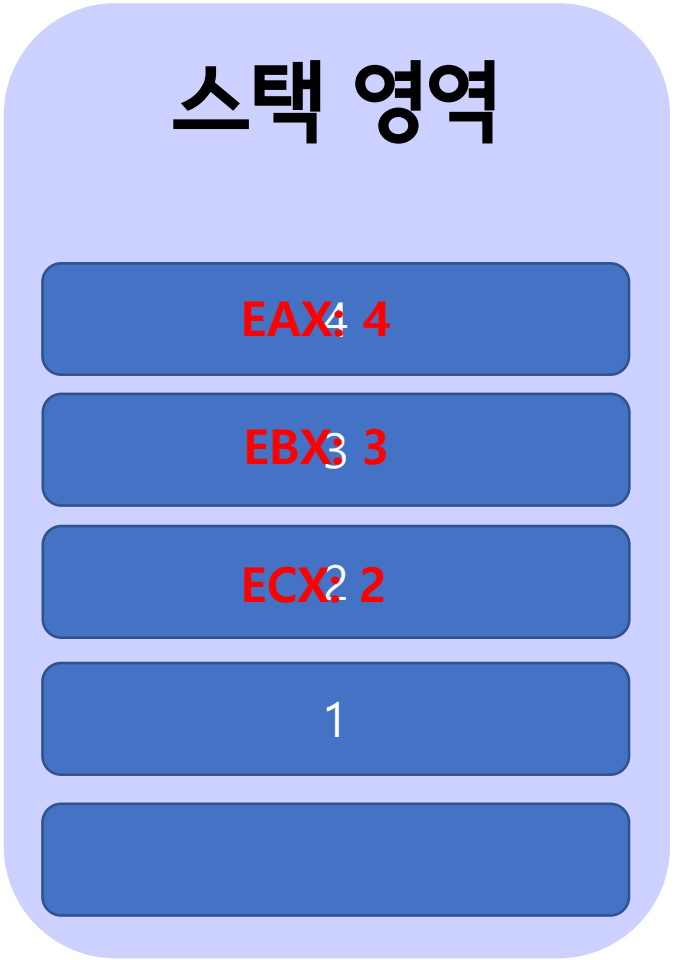
PUSH 1

PUSH 2

PUSH 3

PUSH 4

Background : stack



낮은 주소

높은 주소

```
POP EAX
POP EBX
POP ECX
```

Background : Little Endian

바이트 오더링: 데이터 저장하는 방식

Little Endian

a=0x12345678

a의 주소가 0x100일 경우, 메모리 상태

0x100	*0x101*	*0x102*	*0x103*
0x78	0x56	0x34	0x12

ex) Intel x86 CPU
산술 연산과 데이터의 타입 확장/축소 시 효율

Big Endian

a=0x12345678

a의 주소가 0x100일 경우, 메모리 상태

0x100	*0x101*	*0x102*	*0x103*
0x12	0x34	0x56	0x78

ex) 네트워크 프로토콜, RISC 계열의 CPU

Background : assembly

데이터 전달	MOV, LEA, PUSH, POP
산술 연산	INC, DEX, ADD, SUB, MUL, DIV
논리 연산	AND, OR, XOR, NOT, SHL, SHR
비교 연산	CMP, TEST
흐름 제어	JMP, JZ, JNZ, JE, JNE, JG, JGE, JL, JLE
함수 생성 / 종료	CALL, RET

Background : assembly

```
MOV EAX, ECX

MOV EAX, [ECX]

MOV [0x0100BEE8], 0x12345678
```

EAX	0000000
ECX	0100BEE0
EDX	0000000
EBX	0000000

0x100BEE0	01 00 00 E0
0x100BEE4	02 00 00 E4
0x100BEE8	03 00 00 E8
0x100BEEC	04 00 00 EC
0x100BEF0	05 00 00 F0

Background : assembly

POP EBP

PUSH [ECX]

EBP 0100BEE0
ECX 0100BEEC
EBP 00000001

	78 56 34 21
	04 00 00 EC
	01 00 00 00
0x100BEE0	01 00 00 E0
0x100BEE4	02 00 00 E4
0x100BEE8	03 00 00 E8
0x100BEEC	04 00 00 EC
0x100BEF0	05 00 00 F0

✓ PUSH 될 때마다
ESP는 4씩 증가

✓ 스택은 높은 곳에서
낮은 곳으로 쌓인다

Background : assembly

```
POP EBP

POP [ECX]
```

ESP	0100BEE0
ECX	0100BEEC
EBP	00000001

0x100BEE0	01 00 00 E0
0x100BEE4	02 00 00 E4
0x100BEE8	03 00 00 E8
0x100BEEC	04 00 00 EC
0x100BEF0	05 00 00 F0

✓ POP 될 때마다 ESP 는 4씩 감소

Background : assembly

```
ADD EAX, [ECX]

ADD EAX, 8

SUB [0x0100BEE8], 0x12345678
```

EAX	0000003
ECX	0100BEE0
EDX	0000000
EBX	0000000

0x100BEE0	04 00 00 00
0x100BEE4	02 00 00 E4
0x100BEE8	79 56 34 12
0x100BEEC	04 00 00 EC
0x100BEF0	05 00 00 F0

Background : assembly

MOV EAX, 0x60AB10

분기문: JMP EAX

레지스터 EIP는 Program Counter를 의미

실행할 Instruction을 가리키는 Pointer!

EIP: 0x60AB10

MOV EAX, 0x60AB10

MOV BX, 0x12

CMP BX, 0x11

조건분기문: JE EAX

MOV EAX, 0x60AB10

함수 불러오기: CALL EAX

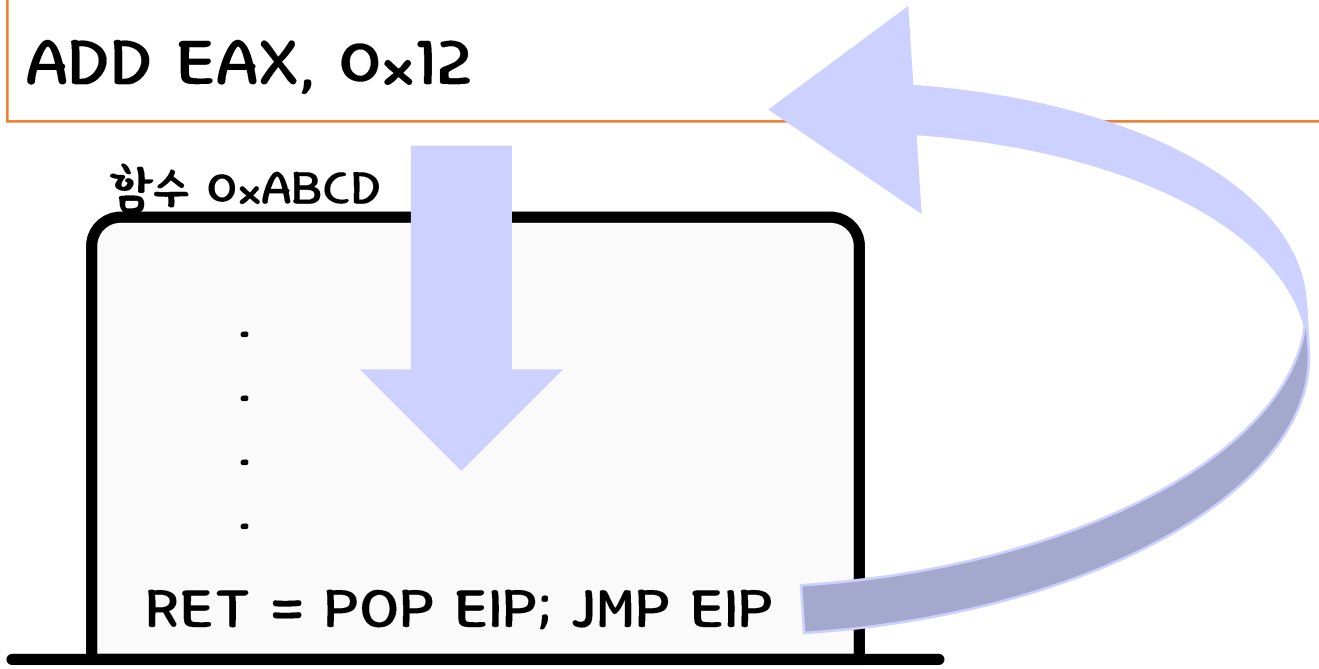
CALL 0x60AB10도 가능!

Background : assembly

JMP와 CALL의 차이 : CALL은 다시 돌아온다

CALL 0xABCD -> PUSH next EIP; JMP 0xABCD;

ADD EAX, 0x12



Background : calling convention

cdecl, stdcall, fastcall 등 다양한 방식이 있으나,

함수 파라미터를 스택을 통해 전달한다는 공통점이 있다!

cdecl: 주로 C언어에서 사용되는 방식

ADD (1, 2)

ADD 함수의 주소: 0x401000

PUSH 2

PUSH 1

CALL 0x401000

*** stdcall에서도 동일한 방식 사용, fastcall에서는 함수 파라미터 전달에 레지스터 사용**

실습: CrackMe!

Immunity Debugger 다운로드 링크:

https://www.immunityinc.com/products/debugger/

HxD(헥스 에디터) 다운로드 링크:

https://mh-nexus.de/en/downloads.php?product=HxD20

실습파일 다운로드 링크:

https://github.com/sonysame/seminar

실습: Immunity Debugger 사용법

<단축키>

F2 : Breakpoint

Ctrl+F2 : Restart

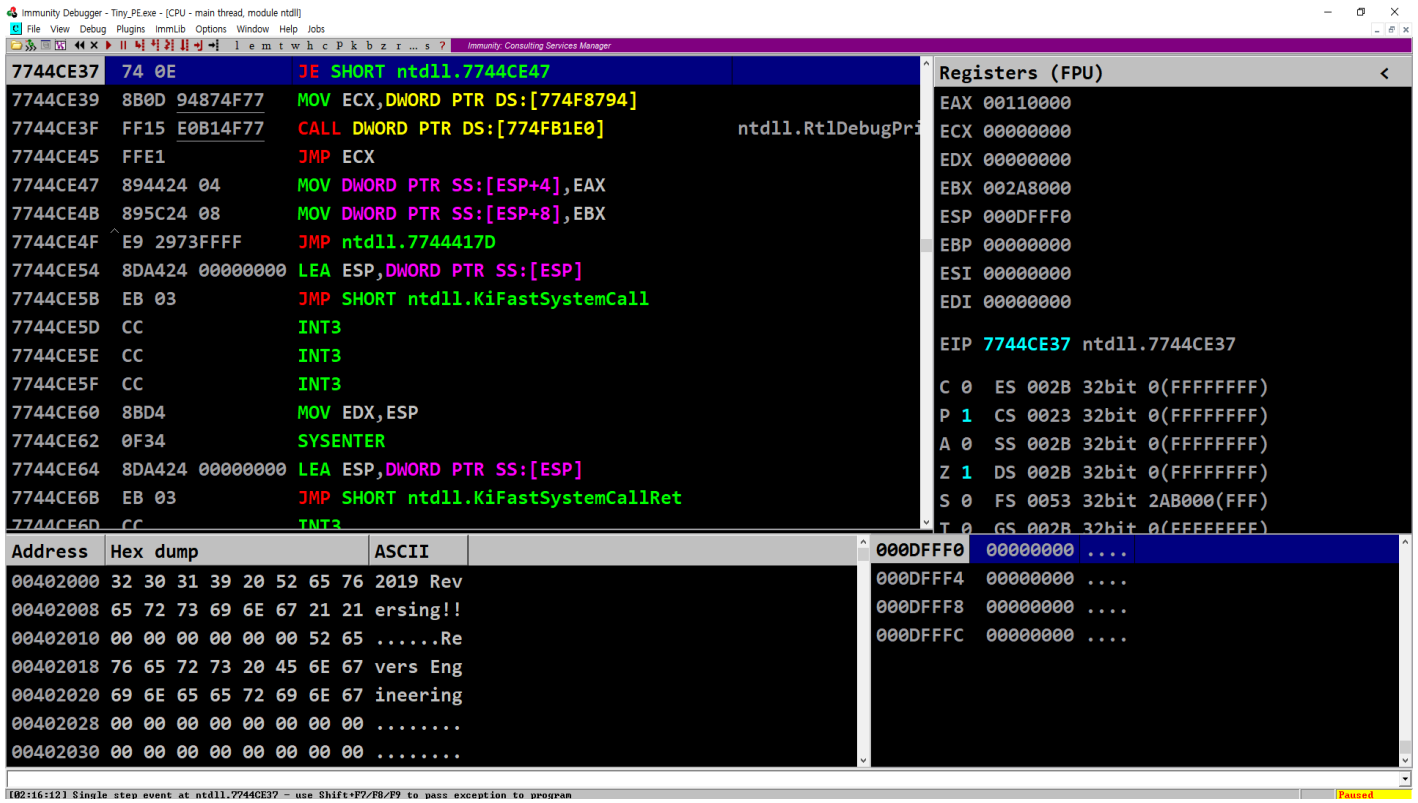
F7 : Step In

F8 : Step Over

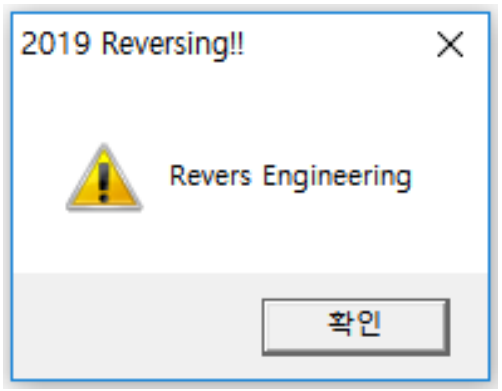
F9 : Run

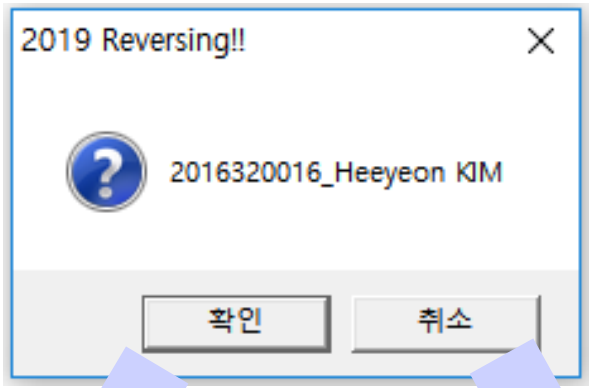
SpaceBar : Edit

Ctrl+G : Find Address

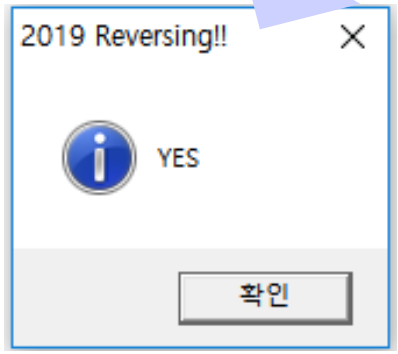


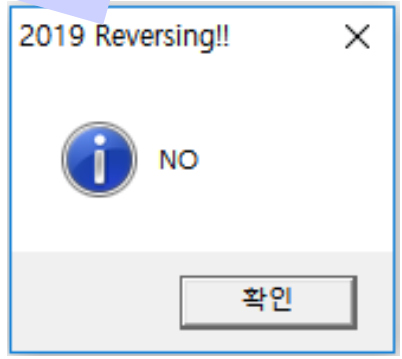
실습: Tiny_PE





1. 파란 물음표 문양, 학번_이름+공백+성, 확인/취소
2. 확인->파란 느낌표 문양, YES
3. 취소->파란 느낌표 문양, NO





Thank You