

Whols 리버싱 스터디

Mungsul

Review

- Intel x86 어셈블리
- 스택
- 콜링 컨벤션

오늘 할 것

- 디버거
- IDA 7.0 Freeware
- 디버거 이론
- EXE 동적분석 (ollydbg)
- ELF
- gdb

디버거

- 프로그램을 디버그 하는데 쓰이는 프로그램
- intel x86에는 디버깅을 위한 Interrupt들이 CPU에 디자인 되어 있음.
- 디버거는 프로그램을 검사할 수 있는 여러 수단을 갖고 있음.
 - break point
 - single stepping
 - run, continue

디버거

- 종류

Ollydbg, immunity dbg, IDA, x96 debugger 등 많다.

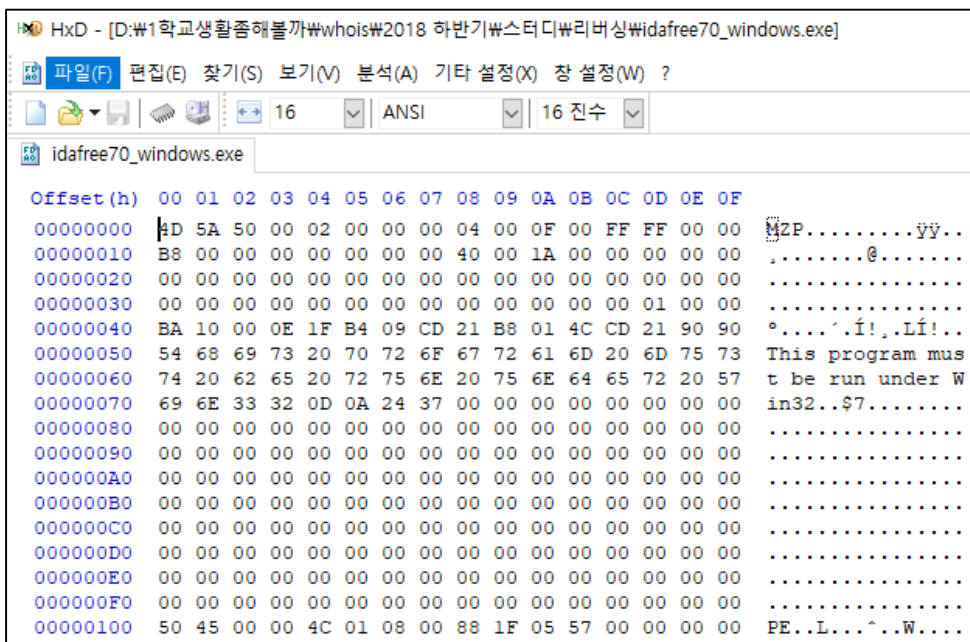
- 대부분 code, stack, register 등을 보여주는 UI를 갖고 있음

IDA (Interactive Disassembler)

- x86, x64, arm, mips 등 여러 아키텍처들에 대한 분석이 가능한 Debugger
- PE 뿐만 아니라 리눅스 실행파일인 ELF도 분석 가능.

PE

- Windows에서 사용하는 실행 파일 구조.



The screenshot shows a hex editor window titled 'HxD - [D:\₩1학교생활준해볼까₩whois₩2018 하반기₩스터디₩리버싱₩idafree70_windows.exe]'. The menu bar includes '파일(F)', '편집(E)', '찾기(S)', '보기(V)', '분석(A)', '기타 설정(X)', '창 설정(W)', and '?'. The toolbar shows icons for file operations and a status bar indicating '16' columns, 'ANSI' encoding, and '16' rows. The main window displays the hex data for 'idafree70_windows.exe'. The first few bytes of the file are shown in a table with columns for Offset (h) and hex values. The first byte is 0x5A, which is the 'MZ' signature. The rest of the header is shown in hex and ASCII. The ASCII column shows the text 'MZP.....ÿÿ..' followed by a null terminator, and then the text 'This program must be run under Windows'.

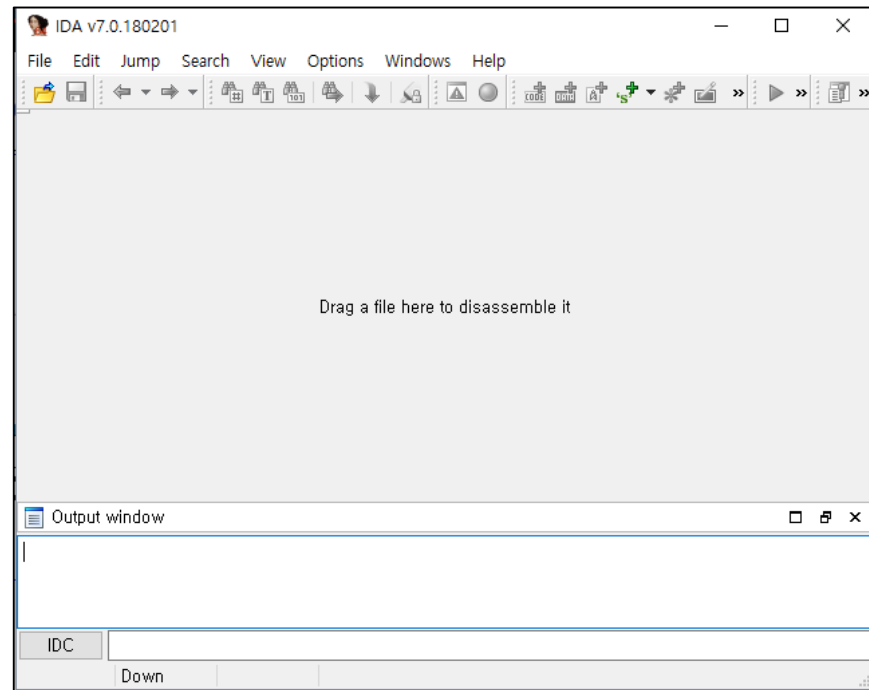
Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	5A	50	00	02	00	00	00	04	00	0F	00	FF	FF	00	00	MZP.....ÿÿ..
00000010	B8	00	00	00	00	00	00	40	00	1A	00	00	00	00	00@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00
00000040	BA	10	00	0E	1F	B4	09	CD	21	B8	01	4C	CD	21	90	°.....í!..Lí!..
00000050	54	68	69	73	20	70	72	6F	67	72	61	6D	20	6D	75	This program mus
00000060	74	20	62	65	20	72	75	6E	20	75	6E	64	65	72	20	t be run under W
00000070	69	6E	33	32	0D	0A	24	37	00	00	00	00	00	00	00	in32..\$7.....
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000100	50	45	00	00	4C	01	08	00	88	1F	05	57	00	00	00	PE...L...^..W....

당장 hex editor로 아무 EXE 파일이나 열어봐도 헤더인 MZ가 보인다.

IDA

- free version 다운로드 가능

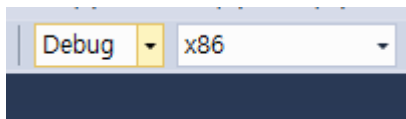
https://www.hex-rays.com/products/ida/support/download_freeware.shtml



Sample 파일 제작

```
ConsoleApplication2
1  #include <stdio.h>
2  int main( )
3  {
4      printf("Hell, World\n");
5  }
```

Visual Studio 2017 기준



Debug로 설정하면

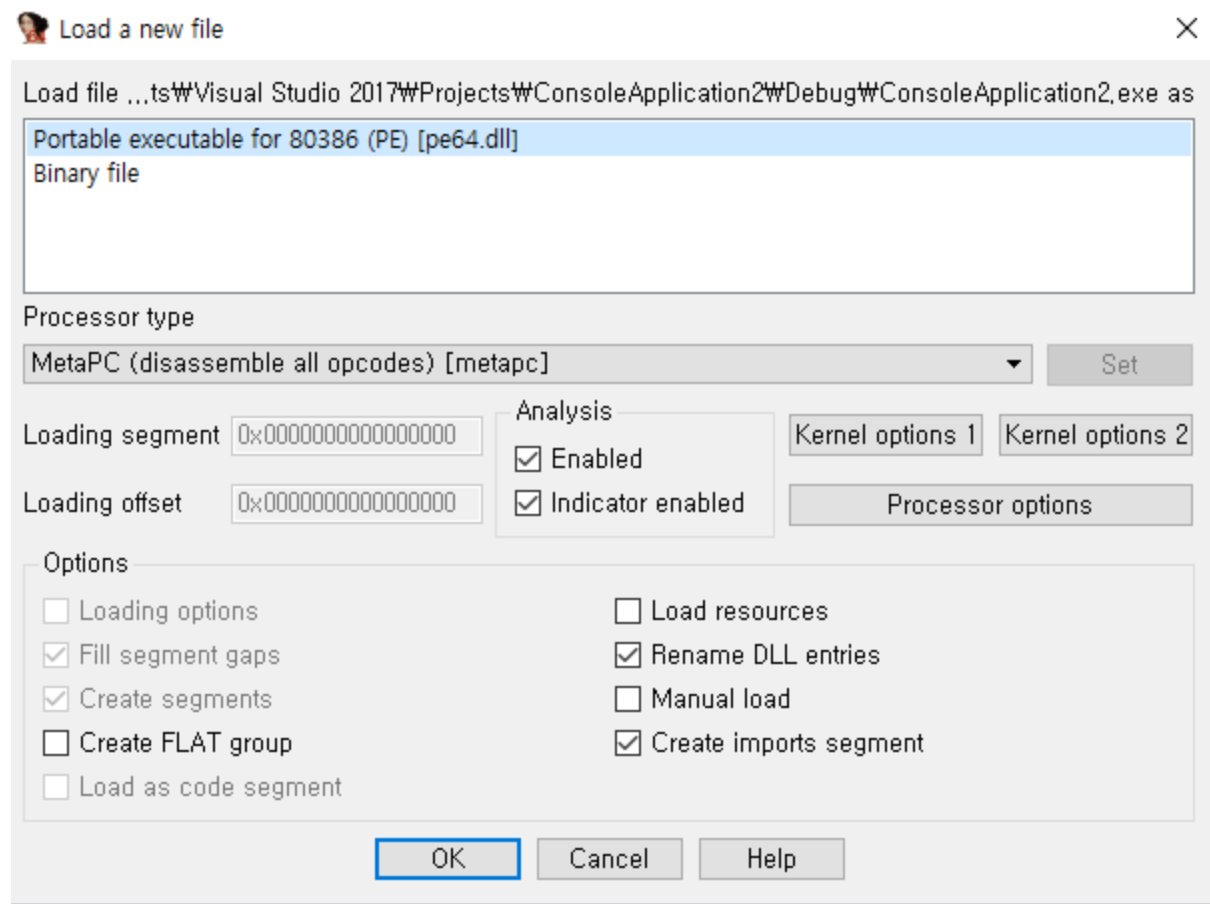
내 PC > OS (C:) > 사용자 > Mungsul > 문서 > Visual Studio 2017 > Projects > ConsoleApplication2			
이름	수정한 날짜	유형	크기
.vs	2018-09-19 오전...	파일 폴더	
ConsoleApplication2	2018-09-19 오전...	파일 폴더	
Debug	2018-09-19 오전...	파일 폴더	
ConsoleApplication2.sln	2018-09-19 오전...	Visual Studio Sol...	2KB

프로젝트 경로에 Debug라는 디렉토리가 생성된다.
Release일 경우 Release 디렉토리가 생성됨.

<< OS (C:) > 사용자 > Mungsul > 문서 > Visual Studio 2017 > Projects > ConsoleApplication2 > Debug			
이름	수정한 날짜	유형	크기
ConsoleApplication2.exe	2018-09-19 오전...	응용 프로그램	37KB
ConsoleApplication2.ilc	2018-09-19 오전...	Incremental Linke...	294KB
ConsoleApplication2.pdb	2018-09-19 오전...	Program Debug ...	388KB

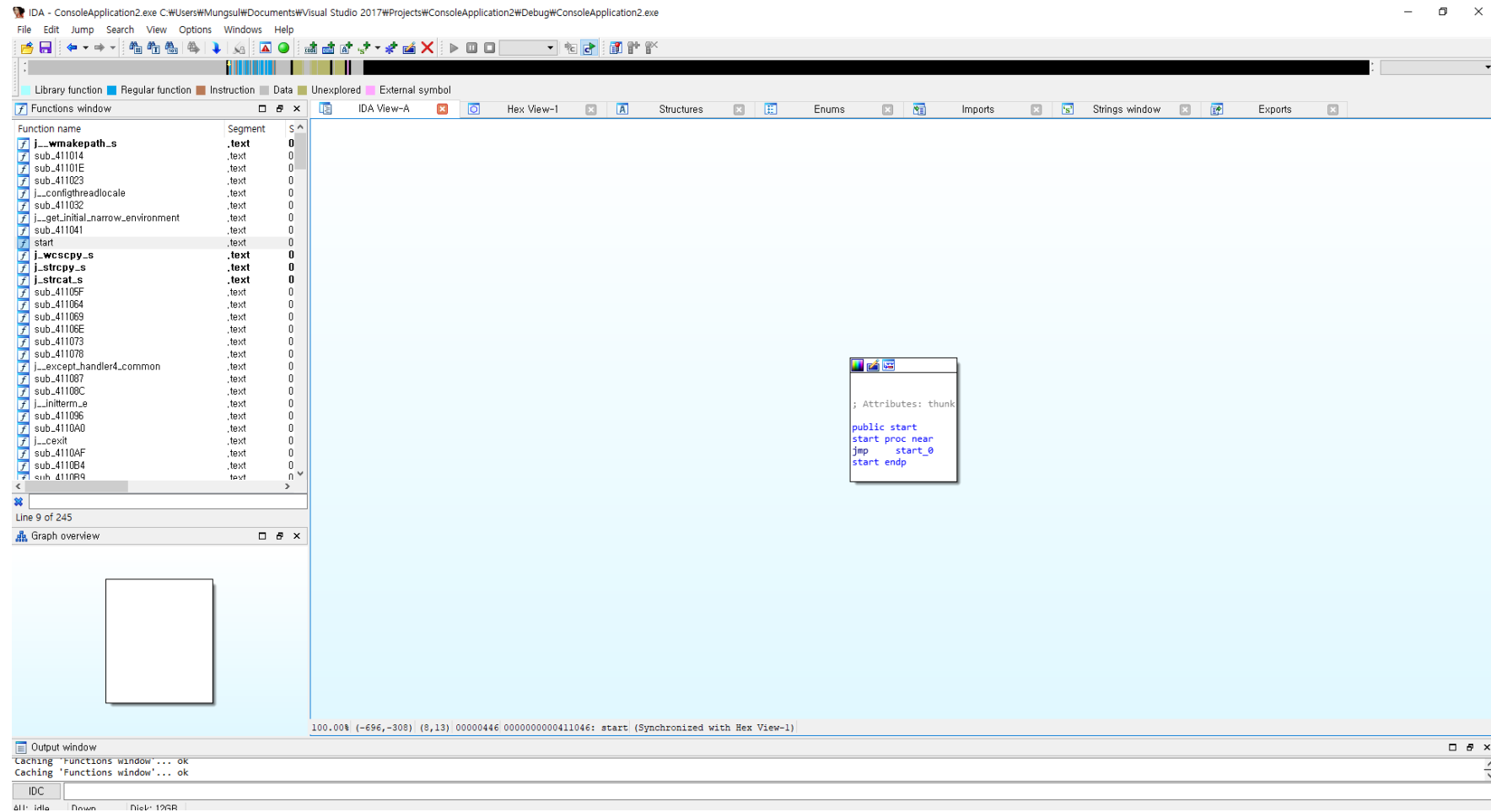
Debug 디렉토리에 빌드된 exe가 생성된다.

IDA

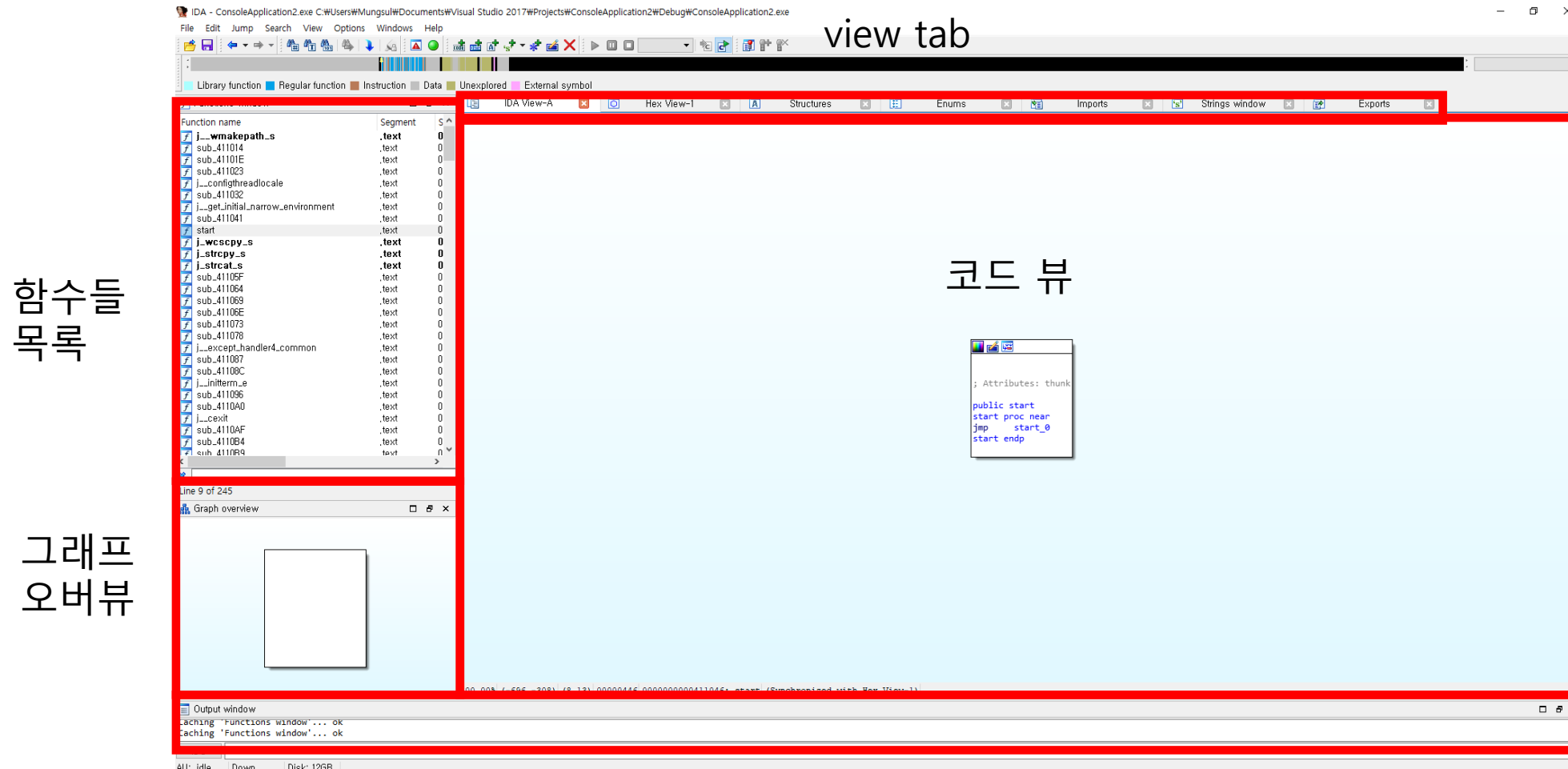


해당 파일을 Load

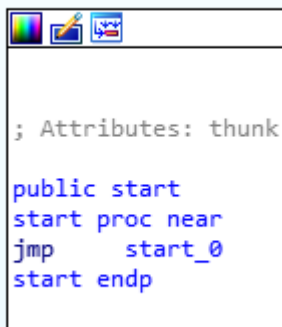
IDA



IDA

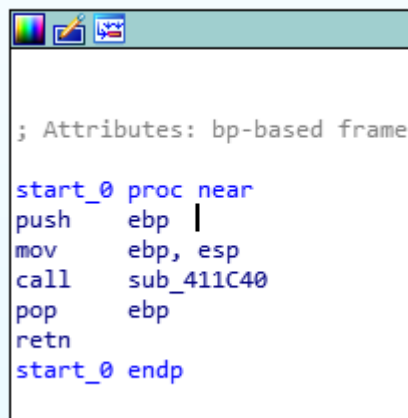


IDA



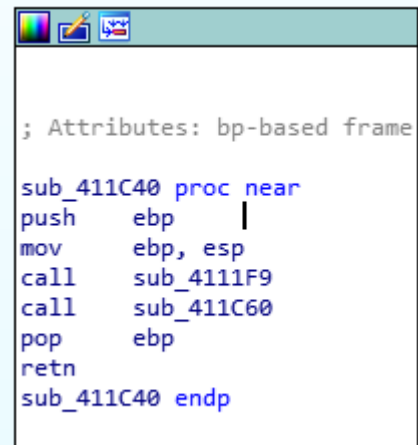
```
; Attributes: thunk  
  
public start  
start proc near  
jmp     start_0  
start endp
```

IDA에서는 알아서 entry point
를 찾아서 Focusing 해줌



```
; Attributes: bp-based frame  
  
start_0 proc near  
push    ebp |  
mov     ebp, esp  
call    sub_411C40  
pop     ebp  
retn  
start_0 endp
```

start_0 를 더블클릭하면
해당 함수를 타고 들어감.



```
; Attributes: bp-based frame  
  
sub_411C40 proc near  
push    ebp |  
mov     ebp, esp  
call    sub_4111F9  
call    sub_411C60  
pop     ebp  
retn  
sub_411C40 endp
```

임의로 이름지어진
함수 또한 마찬가지로

IDA

- 대부분의 디버거에서는 Symbol이 있는 함수들의 이름은 표시해줌.
- Symbol이 없는 경우 함수의 이름을 표시하지 않고 주소만 나타남.
- IDA의 경우 Symbol이 없는 함수는 앞에 sub_ 라는 prefix가 붙음.

main 함수 찾기

- 프로그램의 시작은 main 함수가 아님!
- 프로그램의 실제 시작 주소는 entry point(ep)로 ep에서 프로그램을 정상적으로 구동시키기 위한 작업들을 해줌
- 그러한 작업들을 진행하고 난 뒤에 main함수가 호출됨!

main 함수 찾기

```
; Attributes: bp-based frame
sub_411C60 proc near
var_40= dword ptr -40h
var_3C= dword ptr -3Ch
var_38= dword ptr -38h
var_34= dword ptr -34h
var_30= dword ptr -30h
Code= dword ptr -2Ch
var_28= dword ptr -28h
var_24= dword ptr -24h
var_20= dword ptr -20h
var_1A= byte ptr -1Ah
var_19= byte ptr -19h
ms_exc= CPPEH_RECORD ptr -18h

push    ebp
mov     ebp, esp
push    0FFFFFFEh
push    offset stru_417EA8
push    offset SEH_412030
mov     eax, large fs:0
push    eax
add     esp, 0FFFFFFD0h
push    ebx
push    esi
push    edi
mov     eax, __security_cookie
xor     [ebp+ms_exc.registration.ScopeTable], eax
xor     eax, ebp
push    eax
lea     eax, [ebp+ms_exc.registration]
mov     large fs:0, eax
mov     [ebp+ms_exc.old_esp], esp
push    1
call    sub_411280
add     esp, 4
movzx   eax, al
test    eax, eax
jnz     short loc_411CAB

loc_411DFC:
mov     ecx, [ebp+ms_exc.exc_ptr]
mov     edx, [ecx]
mov     eax, [edx]
mov     [ebp+var_30], eax
mov     ecx, [ebp+ms_exc.exc_ptr]
push    ecx
mov     edx, [ebp+var_30]
push    edx
call    j__seh_filter_exe
add     esp, 8
retn

loc_411E17:
mov     esp, [ebp+ms_exc.old_esp]
mov     eax, [ebp+var_30]
mov     [ebp+var_34], eax
call    sub_411122
movzx   ecx, al
test    ecx, ecx
jnz     short loc_411E35
```

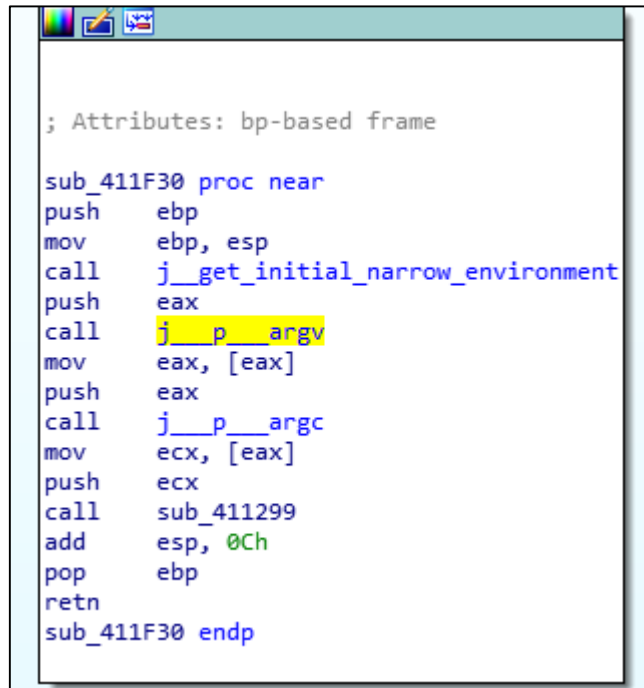
start를 따라간다고 main 함수가 무조건 나오는 것은 아님.

Visual Studio에서 Debug로 빌드된 프로그램은 Main함수 찾기가 까다로울 수 있음.

Release는 그나마 나음.

하지만 Visual Studio 버전마다도 main함수 호출 부분이 달라서 까다로움.

main 함수 찾기



```
; Attributes: bp-based frame

sub_411F30 proc near
push    ebp
mov     ebp, esp
call    j__get_initial_narrow_environment
push    eax
call    j__p__argv
mov     eax, [eax]
push    eax
call    j__p__argc
mov     ecx, [eax]
push    ecx
call    sub_411299
add     esp, 0Ch
pop     ebp
retn
sub_411F30 endp
```

바이너리 많이 뜯으면서 알게된 규칙

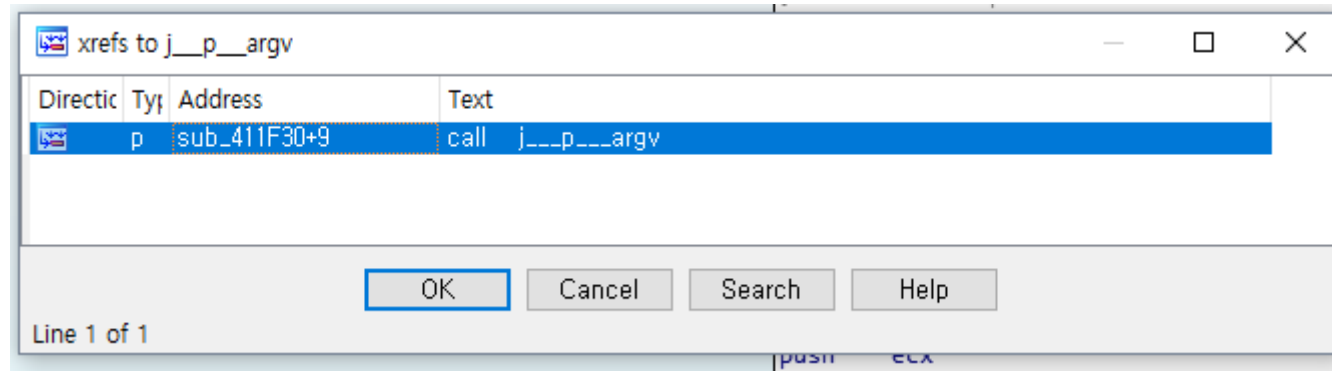
PE에서는 argv, argc 관련된 함수를 호출하고 main 함수를 부른다.

즉, 왼쪽 화면에서는 sub_411299가 main함수임.

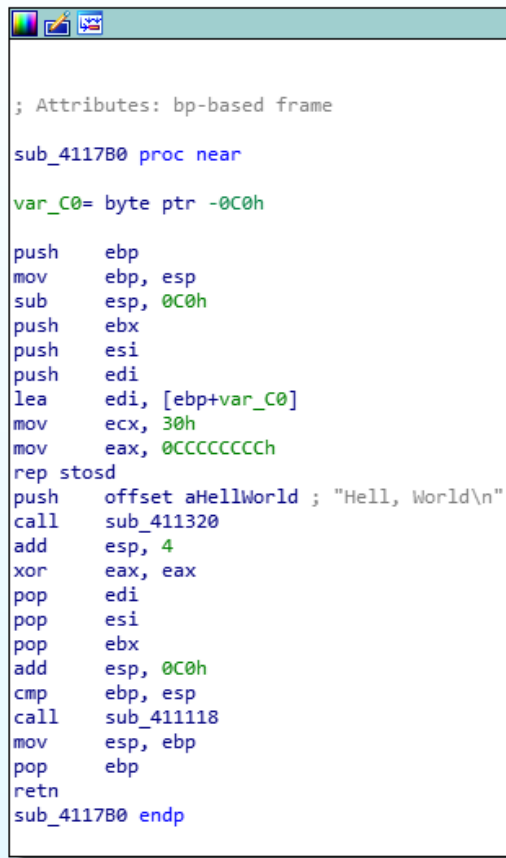
절대적인 기준은 아님.

Xref

- IDA의 강력한 기능
- 변수, 함수, 데이터가 어느 부분에서 참조되는지 알려줌.
- 함수, 변수 등을 선택하고 x key를 누르면 됨.



main 함수 찾기

A screenshot of a window displaying assembly code. The code is for a function named sub_4117B0, which is a near procedure with a bp-based frame. It starts by pushing ebp, moving esp to ebp, and subtracting 0C0h from esp. Then it pushes ebx, esi, edi, and lea edi to [ebp+var_C0]. It moves ecx to 30h and eax to 0CCCCCCCCh, then enters a loop with rep stosd. After the loop, it pushes the offset to aHellWorld, calls sub_411320, adds 4 to esp, xors eax with itself, pops edi, esi, ebx, and adds 0C0h to esp. It compares ebp and esp, calls sub_411118, moves esp to ebp, pops ebp, and returns.

```
; Attributes: bp-based frame

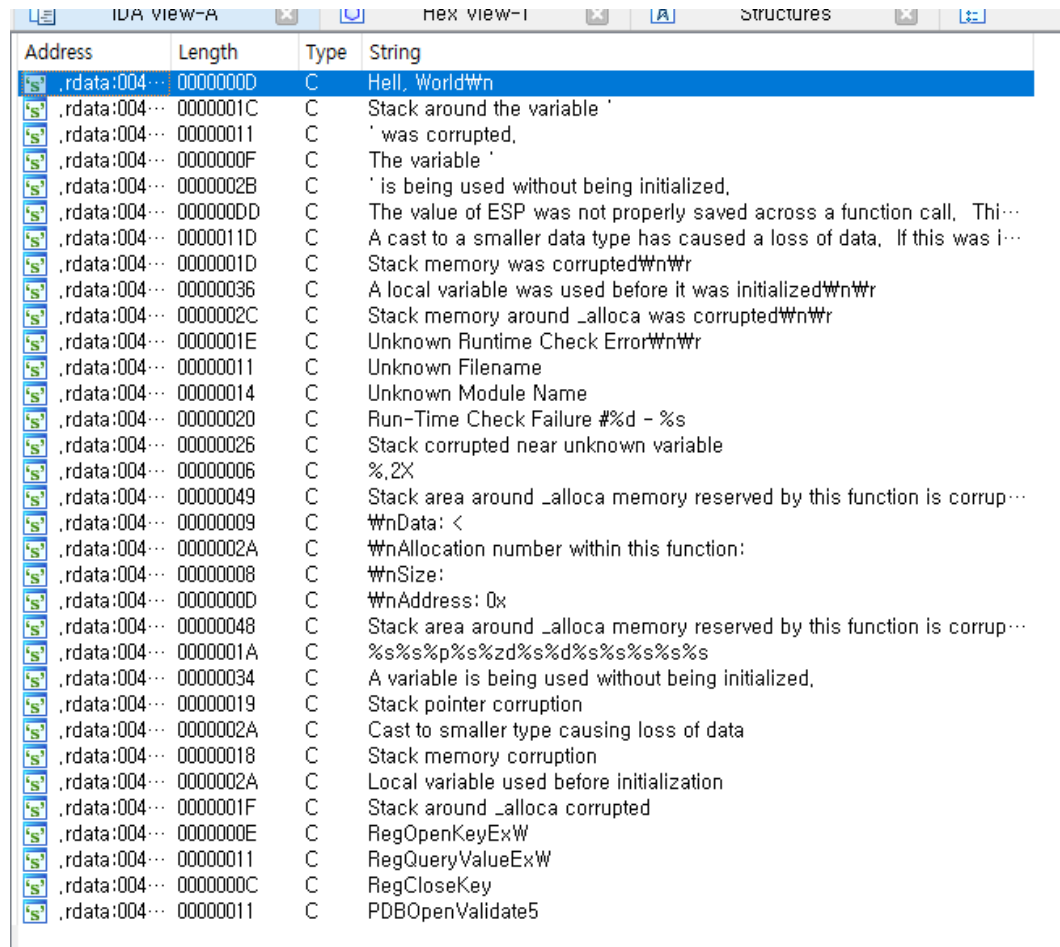
sub_4117B0 proc near

var_C0= byte ptr -0C0h

push    ebp
mov     ebp, esp
sub     esp, 0C0h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_C0]
mov     ecx, 30h
mov     eax, 0CCCCCCCCh
rep stosd
push    offset aHellWorld ; "Hell, World\n"
call    sub_411320
add     esp, 4
xor     eax, eax
pop     edi
pop     esi
pop     ebx
add     esp, 0C0h
cmp     ebp, esp
call    sub_411118
mov     esp, ebp
pop     ebp
retn
sub_4117B0 endp
```

찾았다. main 함수

문자열 확인



Address	Length	Type	String
.rdata:004...	0000000D	C	Hell, WorldWn
.rdata:004...	0000001C	C	Stack around the variable '
.rdata:004...	00000011	C	' was corrupted,
.rdata:004...	0000000F	C	The variable '
.rdata:004...	0000002B	C	' is being used without being initialized,
.rdata:004...	000000DD	C	The value of ESP was not properly saved across a function call, Thi...
.rdata:004...	0000011D	C	A cast to a smaller data type has caused a loss of data, If this was i...
.rdata:004...	0000001D	C	Stack memory was corruptedWnWr
.rdata:004...	00000036	C	A local variable was used before it was initializedWnWr
.rdata:004...	0000002C	C	Stack memory around _alloca was corruptedWnWr
.rdata:004...	0000001E	C	Unknown Runtime Check ErrorWnWr
.rdata:004...	00000011	C	Unknown Filename
.rdata:004...	00000014	C	Unknown Module Name
.rdata:004...	00000020	C	Run-Time Check Failure #%d - %s
.rdata:004...	00000026	C	Stack corrupted near unknown variable
.rdata:004...	00000006	C	%,2X
.rdata:004...	00000049	C	Stack area around _alloca memory reserved by this function is corrup...
.rdata:004...	00000009	C	WnData: <
.rdata:004...	0000002A	C	WnAllocation number within this function:
.rdata:004...	00000008	C	WnSize:
.rdata:004...	0000000D	C	WnAddress: 0x
.rdata:004...	00000048	C	Stack area around _alloca memory reserved by this function is corrup...
.rdata:004...	0000001A	C	%s%s%p%s%zd%s%d%s%s%s%s
.rdata:004...	00000034	C	A variable is being used without being initialized,
.rdata:004...	00000019	C	Stack pointer corruption
.rdata:004...	0000002A	C	Cast to smaller type causing loss of data
.rdata:004...	00000018	C	Stack memory corruption
.rdata:004...	0000002A	C	Local variable used before initialization
.rdata:004...	0000001F	C	Stack around _alloca corrupted
.rdata:004...	0000000E	C	RegOpenKeyExW
.rdata:004...	00000011	C	RegQueryValueExW
.rdata:004...	0000000C	C	RegCloseKey
.rdata:004...	00000011	C	PDBOpenValidate5

Shift-F12

리버싱 팁

- 프로그램에서 문자열은 쓸 수 밖에 없다.
- 사용 문자열을 파악해서 해당 문자열을 어디서 참조하는지 알면 그 부분이 관련 루틴일 확률이 높다.

리버싱 팁

- 예시 - 리버싱 문제를 푸는 상황

```
#include <stdio.h>
int main()
{
    int a = 0;
    scanf("%d", &a);
    if (a == 56535)
    {
        printf("Correct\n");
    }
    else
    {
        printf("Wrong\n");
    }
}
```

예시 프로그램.

Address	Length	Type	String
.rdata:00401009	00000009	C	Correct\n
.rdata:0040101C	0000001C	C	Stack around the variable 'a' was corrupted.
.rdata:00401031	00000011	C	'a' was corrupted.
.rdata:0040104F	0000000F	C	The variable 'a' is being used without being initialized.
.rdata:0040102B	0000002B	C	'a' is being used without being initialized.
.rdata:00401007	00000007	C	Wrong\n

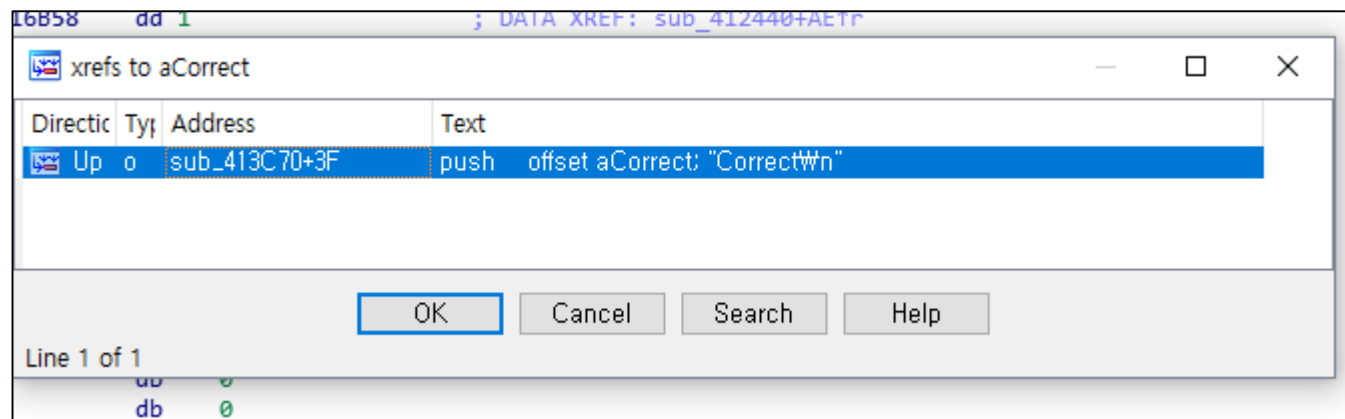
문자열을 먼저 확인

Correct랑 Wrong이 있는걸 보니 저 문자열을 사용하는 부분이 비교 루틴이겠군!

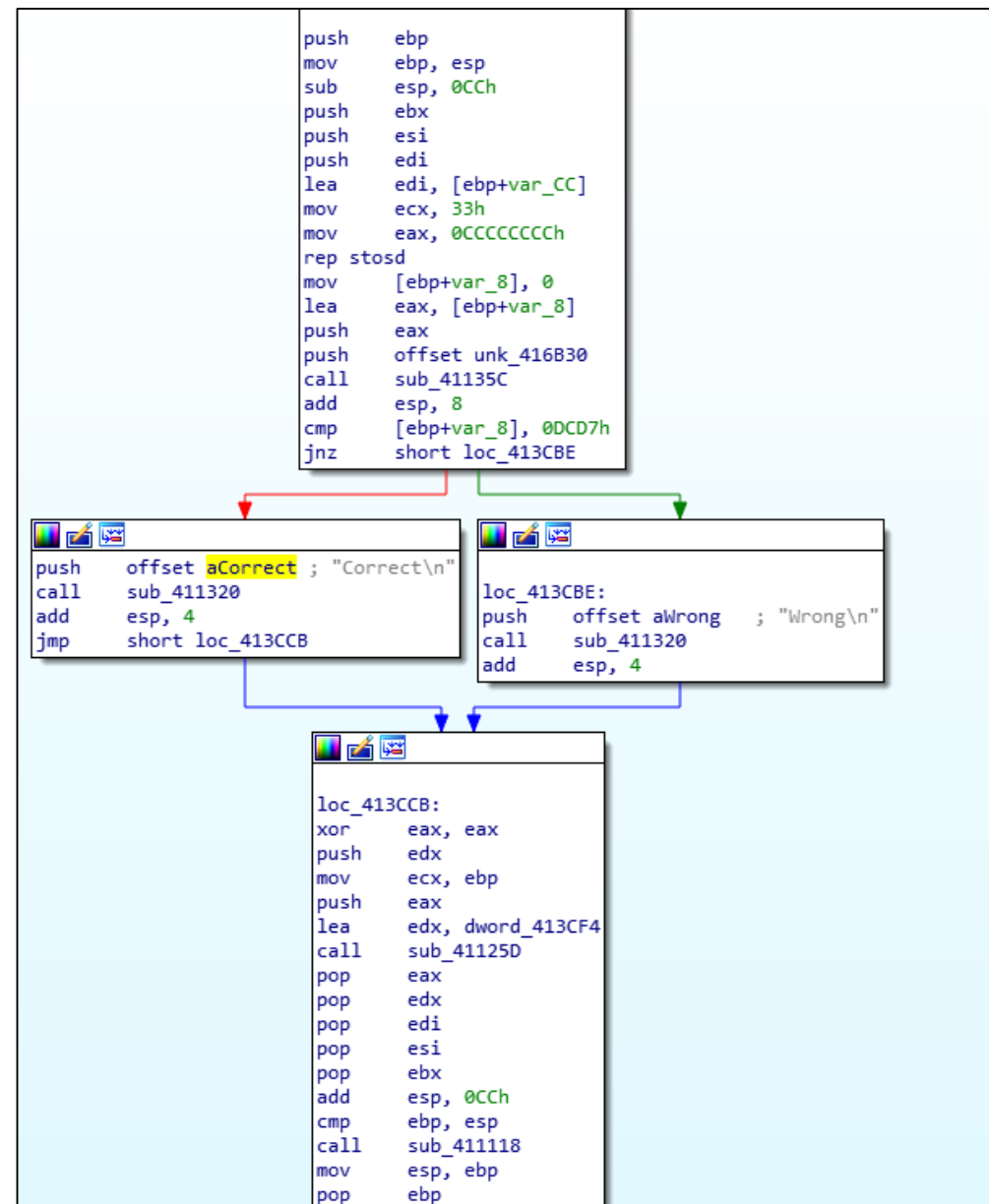
리버싱 팁

```
ata:00416B33 db 0
ata:00416B34 aCorrect db 'Correct',0Ah,0
ata:00416B3D align 10h
ata:00416B40 ; LPCSTR lpMultiByteStr
ata:00416B40 lpMultiByteStr dd offset aTheValueOf
```

문자열을 따라가서



xrefs를 이용 (x키)



루틴 파악

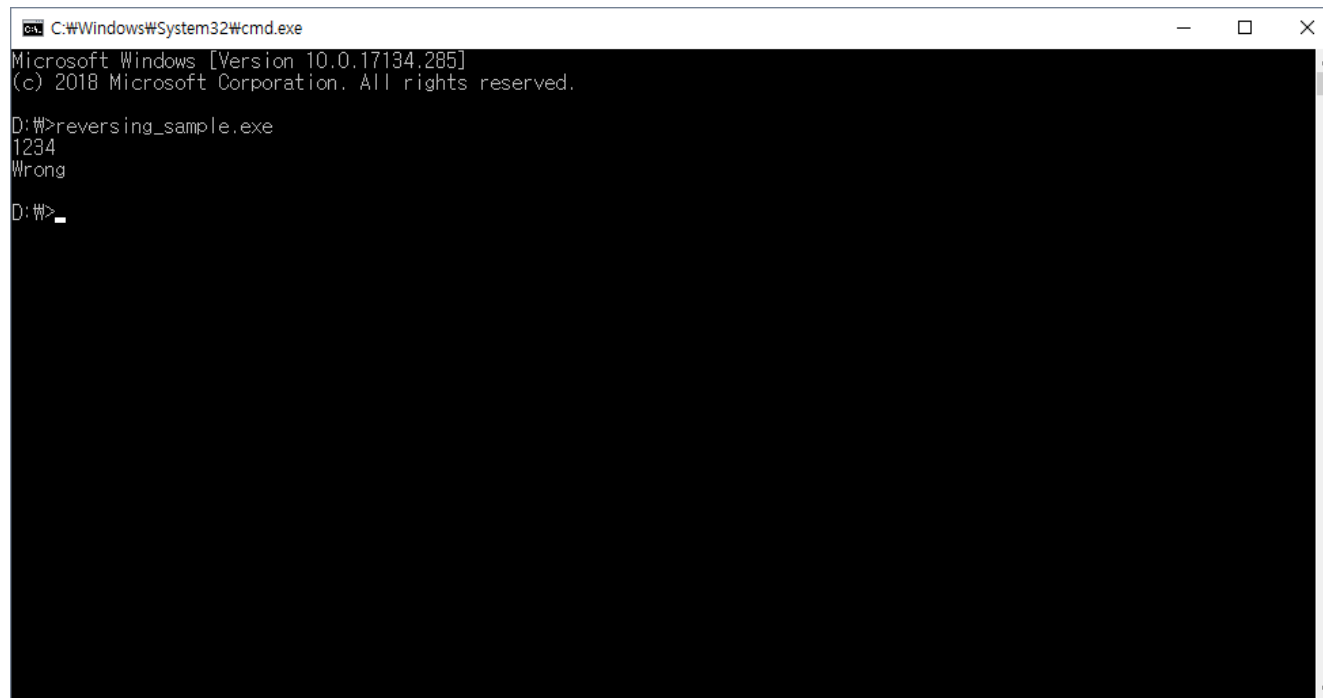
리버싱 팁

- Windows는 Console 뿐만 아니라 GUI로 구성되어 있는 프로그램이 많다.
- 그래서 main함수만 찾는다고 다 되지 않는다.
- 심지어 IDA에서 자동으로 main 함수를 찾아주지 않는 경우도 있다.
- 문자열 기반 검색을 많이 활용하는 것이 좋다.

IDA 써보기

- Sample 프로그램 (2)

<https://drive.google.com/open?id=1s7tq7tOmnsBRVeISCZWGMQxTi1sb1aS->

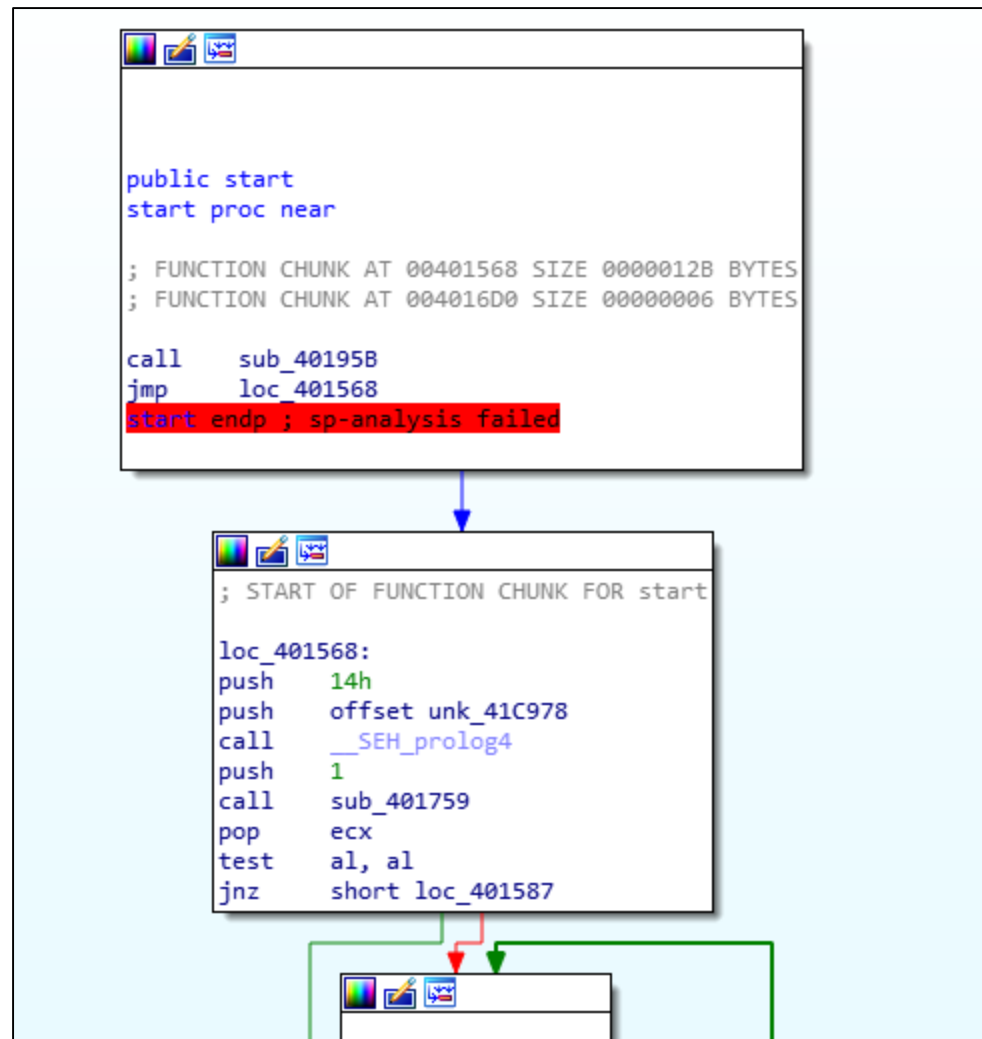


```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17134.285]
(c) 2018 Microsoft Corporation. All rights reserved.

D:\>reversing_sample.exe
1234
Wrong
D:\>
```

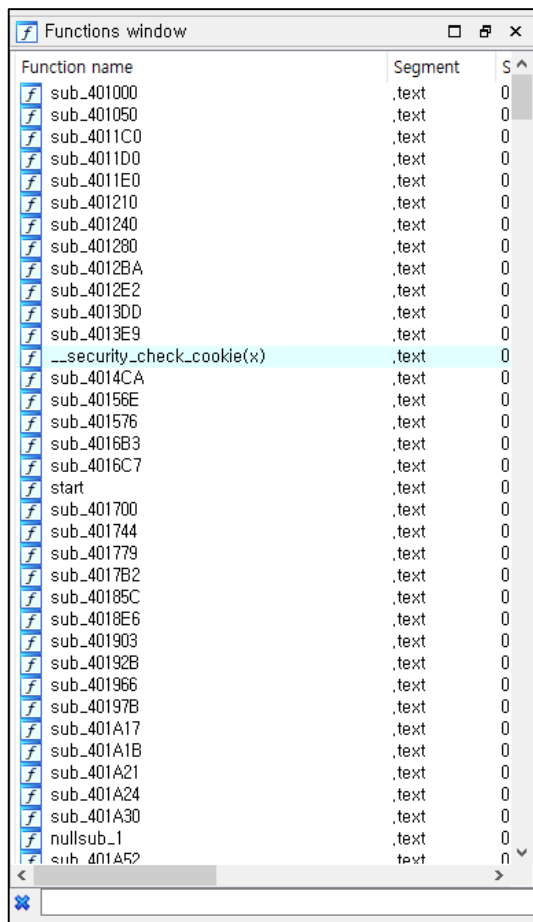
실행하면 입력받고 출력하는 간단한 프로그램.

IDA 써보기

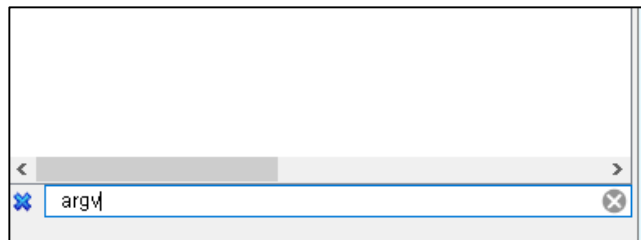


무작정 열어본다..

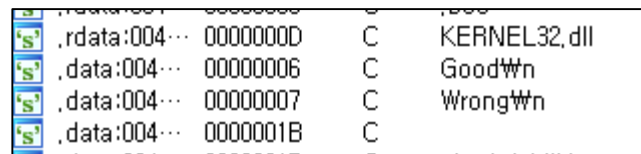
IDA 써보기



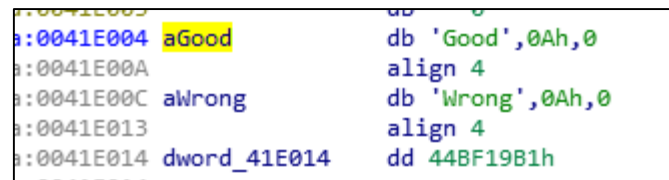
Function window 에서
ctrl + F를 누르면 함수
검색가능



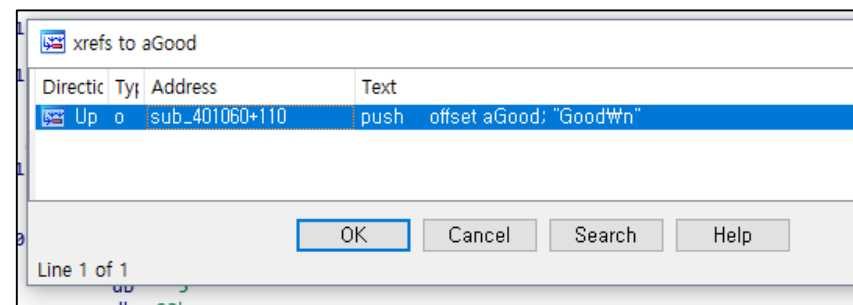
argv를 검색했지만
나오지 않는다...



문자열 검색을 이용.

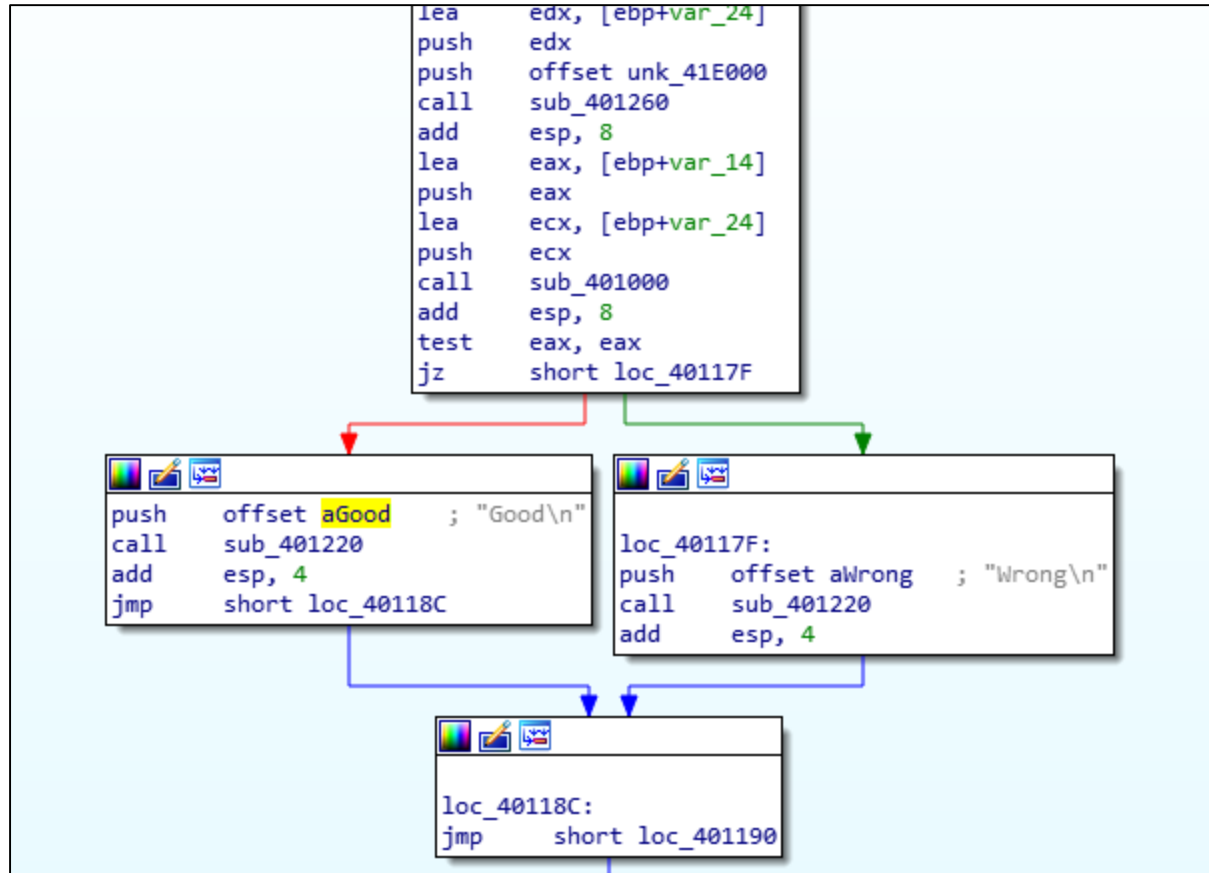


문자열을 따라가서..



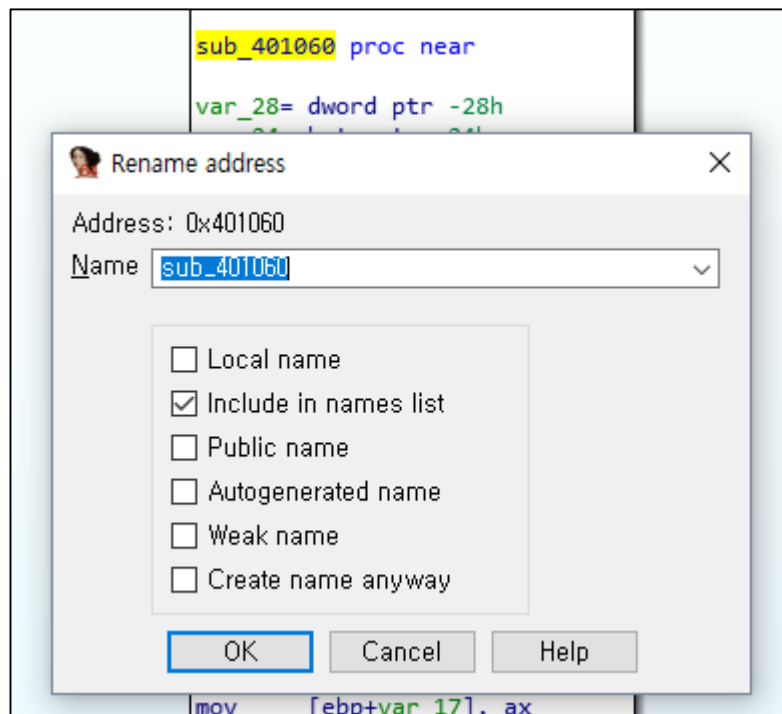
xref 이용

IDA 써보기

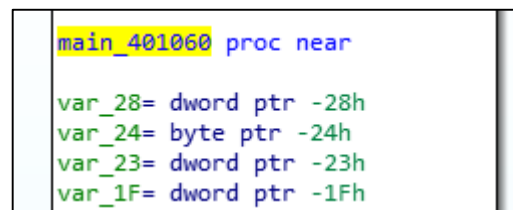
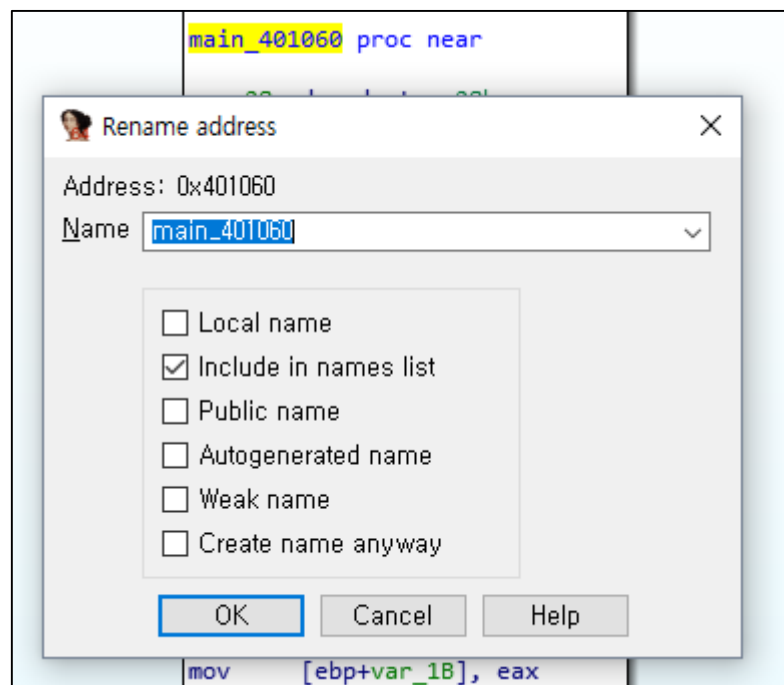


무언가 비교하는 듯한 루틴이 보임!

IDA 써보기



해당 함수에 네이밍을 할 수 있다. (n키)



IDA 써보기

- sub_401060이 main 함수라고 생각한 이유
 - start 부분에서 호출됨.
 - 사용자가 추가한듯한 문자열이 참조됨 (Good, Wrong)
 - 그냥 감으로..
 - 등등
- 사실, IDA pro 에선 main 함수를 잘 찾아줌.

IDA 써보기

- 아쉽게도 IDA Freeware에서는 동적 분석 기능을 제공하지 않는다, main 함수도 잘 안찾아준다.
- 동적 분석을 하려면 다른 디버거를 사용해야함 (ollydbg, immunity debugger 등)
- 오늘은 정적분석만. 머리로 따라가봅시다.

IDA 써보기

```
main_401060 proc near
var_28= dword ptr -28h
var_24= byte ptr -24h
var_23= dword ptr -23h
var_1F= dword ptr -1Fh
var_18= dword ptr -18h
var_17= word ptr -17h
var_14= byte ptr -14h
var_4= dword ptr -4
push    ebp
mov     ebp, esp
sub     esp, 28h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
mov     [ebp+var_24], 0
xor     eax, eax
mov     [ebp+var_23], eax
mov     [ebp+var_1F], eax
mov     [ebp+var_18], eax
mov     [ebp+var_17], ax
mov     ecx, 1
imul    edx, ecx, 0
```

스택 프레임
var_14 : -0x14

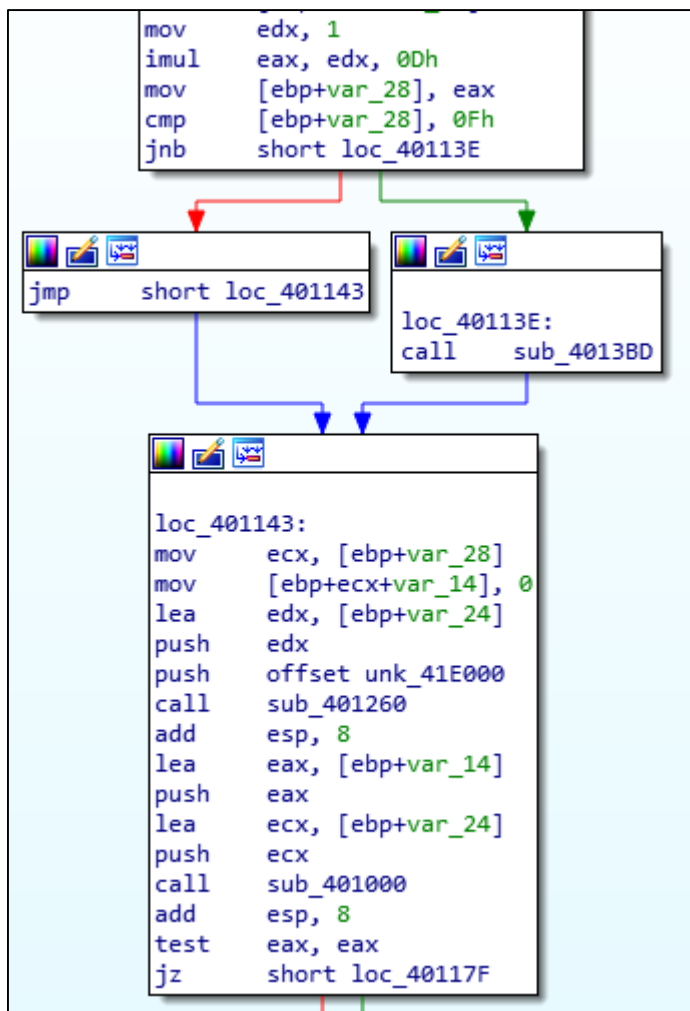
지역 변수를 0
으로 초기화

```
mov     ecx, 1
imul    edx, ecx, 0
mov     [ebp+edx+var_14], 74h
mov     eax, 1
shl     eax, 0
mov     [ebp+eax+var_14], 68h
mov     ecx, 1
shl     ecx, 1
mov     [ebp+ecx+var_14], 69h
mov     edx, 1
imul    eax, edx, 3
mov     [ebp+eax+var_14], 73h
mov     ecx, 1
shl     ecx, 2
mov     [ebp+ecx+var_14], 5Fh
mov     edx, 1
imul    eax, edx, 5
mov     [ebp+eax+var_14], 70h
mov     ecx, 1
imul    edx, ecx, 6
mov     [ebp+edx+var_14], 34h
mov     eax, 1
imul    ecx, eax, 7
mov     [ebp+ecx+var_14], 35h
mov     edx, 1
shl     edx, 3
mov     [ebp+edx+var_14], 35h
mov     eax, 1
imul    ecx, eax, 9
mov     [ebp+ecx+var_14], 77h
mov     edx, 1
imul    eax, edx, 0Ah
mov     [ebp+eax+var_14], 30h
mov     ecx, 1
imul    edx, ecx, 0Bh
mov     [ebp+edx+var_14], 72h
mov     eax, 1
imul    ecx, eax, 0Ch
mov     [ebp+ecx+var_14], 64h
mov     edx, 1
imul    eax, edx, 0Dh
mov     [ebp+var_28], eax
cmp     [ebp+var_28], 0Fh
jnb     short loc_40113E
```

지역 변수에 값을 1byte 씩 저장.
ebp-0x14 부터가 char형 배열
임을 추정할 수 있음.

mov [ebp + reg + var_14], x
reg가 1씩 증가하는 형태임.

IDA 써보기



edx : 1
eax = edx * 0xD

mov [ebp+var_28], eax

[ebp+var_28]이 0xF보다 작지
않으면 loc_40112E로 넘어감.

[ebp+var_28] 이 0xD기 때문에
가지 않음.

맨 끝에 NULL byte를 하나 넣음

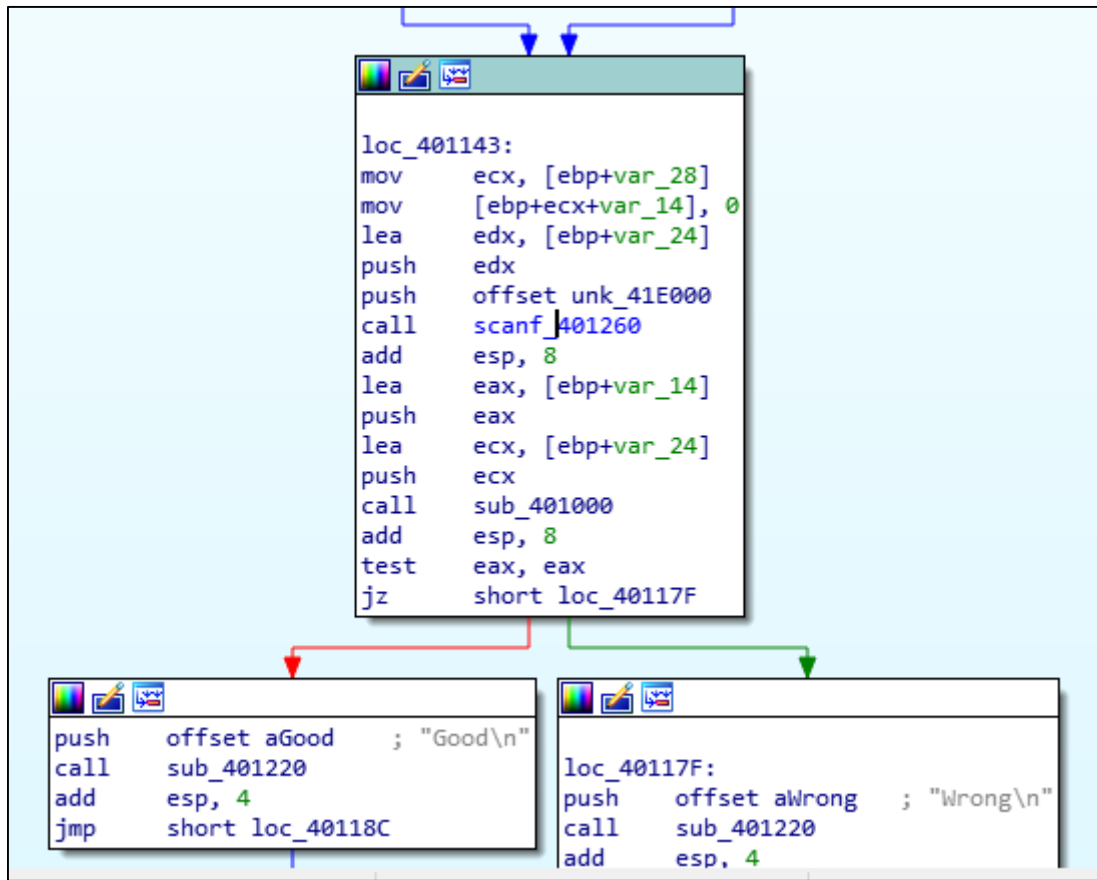
sub_401260(unk_41E000, &var_24);

```
.data:0041E000          ;org 41E000h
.data:0041E000 unk_41E000 db 25h ; %
.data:0041E001          db 73h ; s
.data:0041E002          db 0
.data:0041E003          db 0
```

"%s" 가 첫번째 인자에 들어가는
대표적인 함수는 scanf가 있음.

sub_401260 함수를 scanf로 rename 하자.

IDA 써보기



scanf("%s", &var24);

eax = sub_401000(&var_24, &var_14)

test eax, eax
jz branch

1. and 연산의 결과가 0이면
zf가 1로 설정됨.
and 연산 결과가 1이면
zf가 0으로 설정됨.

2. 조건부 jmp문은 (jz 포함)
zf가 1일 경우 jmp함

즉, eax가 0이면
loc_401196으로 jmp
하게 되는 것.

sub_401000 함수의
결과가 1이어야 Good
을 출력함.

IDA 써보기

```
sub_401000 proc near  
  
var_4= dword ptr -4  
arg_0= dword ptr 8  
arg_4= dword ptr 0Ch  
  
push    ebp  
mov     ebp, esp  
push    ecx  
mov     [ebp+var_4], 0  
mov     [ebp+var_4], 0  
jmp     short loc_40101D
```

sub_401000 함수 분석

```
validate_401000 proc near
```

함수 rename

```
push    ebp  
mov     ebp, esp  
push    ecx  
mov     [ebp+var_4], 0  
mov     [ebp+var_4], 0  
jmp     short loc_40101D
```

```
loc_40101D:  
mov     ecx, [ebp+arg_4]  
push    ecx  
call    sub_4087E0  
add     esp, 4  
cmp     [ebp+var_4], eax  
jge     short loc_40104A
```

```
mov     edx, [ebp+arg_4]  
add     edx, [ebp+var_4]  
movsx   eax, byte ptr [edx]  
mov     ecx, [ebp+arg_0]  
add     ecx, [ebp+var_4]  
movsx   edx, byte ptr [ecx]  
cmp     eax, edx  
jz      short loc_401048
```

```
loc_401048:  
jmp     short loc_401014
```

```
xor     eax, eax  
jmp     short loc_40104F
```

```
loc_40104A:  
mov     eax, 1
```

```
loc_401014:  
mov     eax, [ebp+var_4]  
add     eax, 1  
mov     [ebp+var_4], eax
```

```
loc_40104F:  
mov     esp, ebp  
pop     ebp  
retn  
sub_401000 endp
```

전체 흐름은 이렇게 생겼다.

IDA 써보기

```
mov edx, [ebp+arg_4]
add edx, [ebp+var_4]
movsx eax, byte ptr [edx]
```

⇒ `eax = arg_4[var_4]` 와 같은 구문

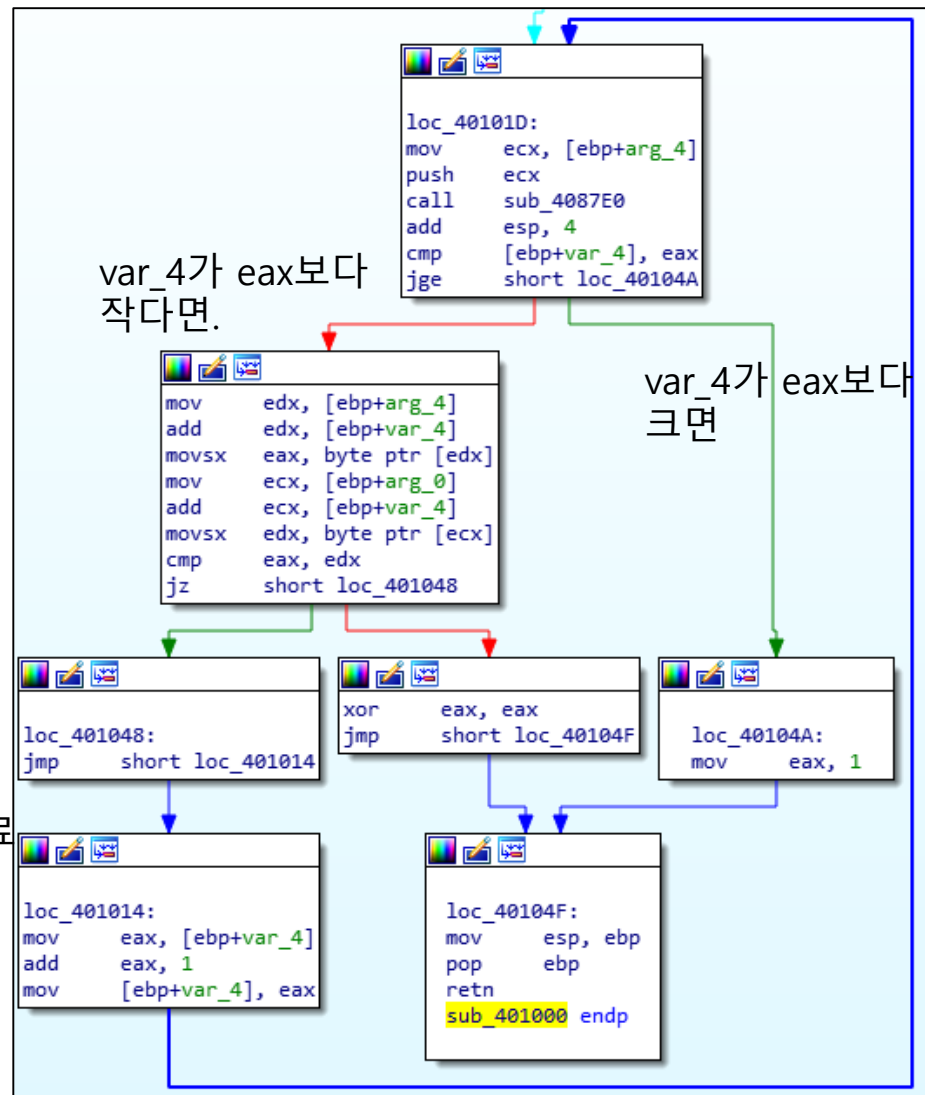
```
mov ecx, [ebp+arg_0]
add ecx, [ebp+var_4]
movsx edx, byte ptr [ecx]
```

⇒ ? 와 같은 구문

`cmp, jz` : `eax`와 `edx`가 같으면 `jmp`
같지 않으면 `eax`에 0을 넣고 함수 종료

`[ebp+var_4]` 를 1씩 증가시킴.
⇒ `var_4`는 index 변수

처음 `[ebp+var_4]` 에는 0이 들어가 있다.



`eax = sub_4087E0(2번째 인자);`
* `sub_4087E0`은 `strlen()` 함수

`var_4`는 index 변수이고 반복은 두 번째 인자의 길이만큼 함.
만약 모든 반복이 끝나면 `eax`에 1을 넣고 함수 종료

IDA 써보기

- 분석으로 유추한 validate 함수 의사코드

```
int validate(char *arg1, char*arg2)
{
    for(i=0;i<strlen(arg2);i++)
    {
        if(arg1[i] != arg2[i])
            return 0;
    }
    return 1;
}
```

IDA 썬보기

```
mov     ecx, 1
imul    edx, ecx, 0
mov     [ebp+edx+var_14], 74h
mov     eax, 1
shl     eax, 0
mov     [ebp+eax+var_14], 68h
mov     ecx, 1
shl     ecx, 1
mov     [ebp+ecx+var_14], 69h
mov     edx, 1
imul    eax, edx, 3
mov     [ebp+eax+var_14], 73h
mov     ecx, 1
shl     ecx, 2
mov     [ebp+ecx+var_14], 5Fh
mov     edx, 1
imul    eax, edx, 5
mov     [ebp+eax+var_14], 70h
mov     ecx, 1
imul    edx, ecx, 6
mov     [ebp+edx+var_14], 34h
mov     eax, 1
imul    ecx, eax, 7
mov     [ebp+ecx+var_14], 35h
mov     edx, 1
shl     edx, 3
mov     [ebp+edx+var_14], 35h
mov     eax, 1
imul    ecx, eax, 9
mov     [ebp+ecx+var_14], 77h
mov     edx, 1
imul    eax, edx, 0Ah
mov     [ebp+eax+var_14], 30h
mov     ecx, 1
imul    edx, ecx, 08h
mov     [ebp+edx+var_14], 72h
mov     eax, 1
imul    ecx, eax, 0Ch
mov     [ebp+ecx+var_14], 64h
mov     edx, 1
imul    eax, edx, 0Dh
mov     [ebp+var_28], eax
cmp     [ebp+var_28], 0Fh
jnb     short loc_40112E
```

main 함수의 &var_14가 validate 함수의 2번째 인자로 넘어감.

main 함수의 var_14는 왼쪽 부분과 같이 초기화 됨.

즉, password는

0x74 0x68 0x69 0x73 0x5f 0x70 0x34 0x35 0x 35 0x77 0x30 0x72 0x64

⇒ "this_p455w0rd"

디버거 이론

- 디버거에는 디버깅을 위한 수단들이 있다.
 - break point
 - single stepping
 - program run
 - memory edit
 - etc..

breakpoint

- 프로그램이 실행되다가 breakpoint를 만나면 멈춤!
- 이는 CPU에서 특정한 트리거가 발동되면 Interrupt를 발생시키도록 했기 때문.

breakpoint

- Software breakpoint

- 보통 대부분의 브레이크 포인트는 Software breakpoint를 뜻함
- breakpoint를 걸면 해당 opcode를 0xCC로 바꿈.
- 프로그램이 0xCC를 만나면 인터럽트를 호출하게 되고 그 인터럽트를 디버거가 처리하면서 원래 opcode로 돌려놓음

- Hardware breakpoint

- CPU에 있는 DR0~DR3 레지스터에 세팅하는 브레이크 포인트
- 단 4개만 사용 가능. Software breakpoint보다 더 low 함.

- Memory breakpoint

- 메모리 페이지를 Guard page 상태로 만들고 접근 시 예외를 발생시킴으로써 프로그램을 중지하게 함.

single stepping

- Step Into
 - call, rep 과 같은 명령이 있을 시 넘어가지 않고 실행
- Step Out
 - call, rep과 같은 명령을 넘어가고 실행
 - 실행을 안시키는 것이 아님
 - call 명령같은 경우 함수 호출이 완료된 다음 주소로 돌아옴.

ollydbg

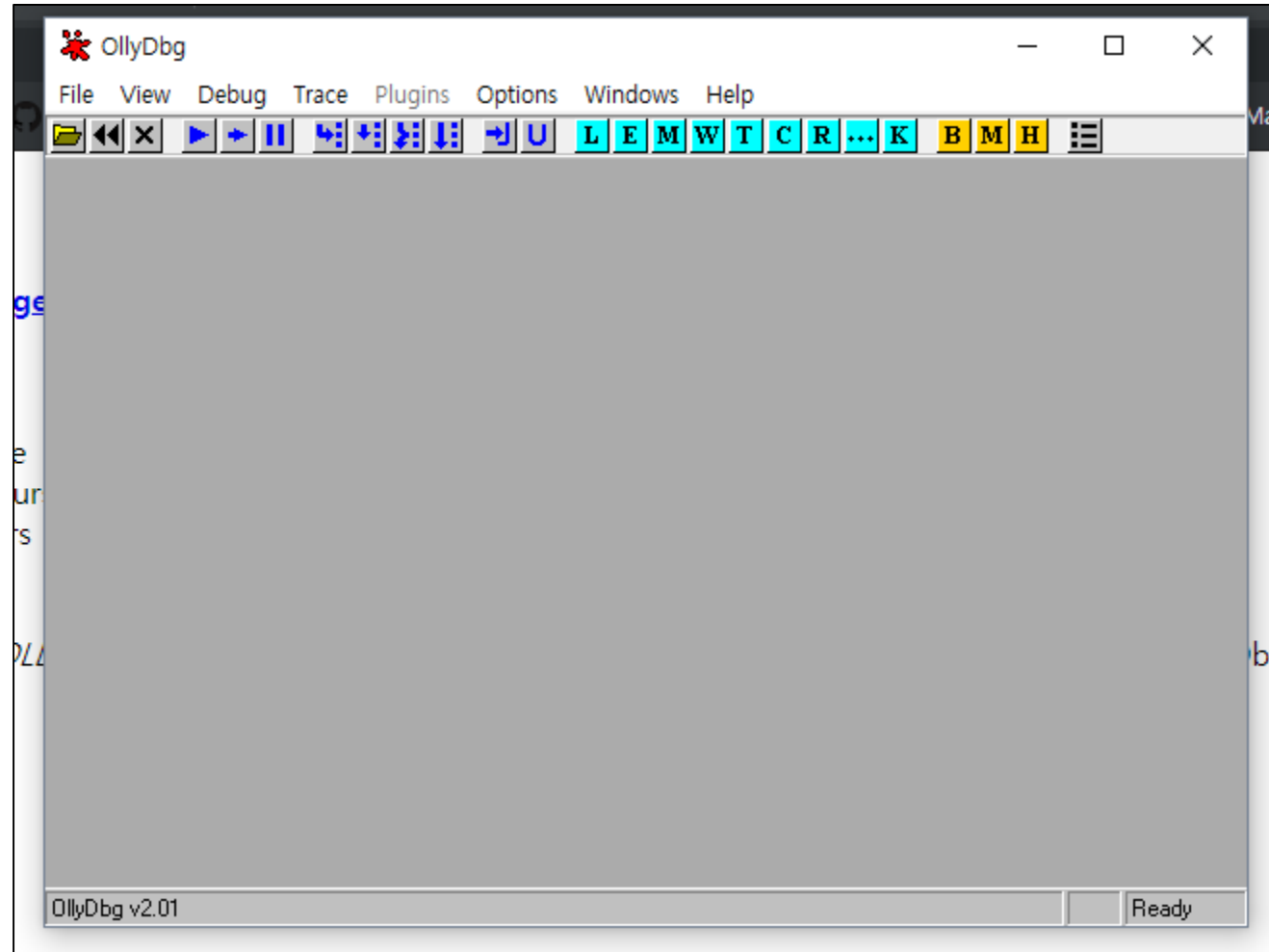
- 유명하고 널리 쓰이는 디버거. 동적분석 가능. 플러그인이 많음.
- <http://www.ollydbg.de/version2.html>

September 27, 2013 - version 2.01. [OllyDbg, empty language file](#)

New version with many new features, among them:

- Help on 77 pages. Please read it first - most of new features a

ollydbg



ollydbg

저번 시간에 사용했던 Sample 프로그램을 분석해보자.

OllyDbg - reversing_sample.exe - [CPU - main thread, module reversing_sample]

File View Debug Trace Plugins Options Windows Help

00B616BF JNE SHORT 00B616C6
00B616C1 CALL 00B6939A
00B616C6 MOV DWORD PTR SS:[EBP-4],-2
00B616CD MOV EAX,DWORD PTR SS:[EBP-20]
00B616D0 CALL 00B61C86
00B616D5 RETN
00B616D6 CALL 00B6195B
00B616DB JMP 00B61568
00B616E0 PUSH EBP
00B616E1 MOV EBP,ESP
00B616E3 MOV EAX,DWORD PTR SS:[ARG.1]
00B616E6 PUSH ESI
00B616E7 MOV ECX,DWORD PTR DS:[EAX+3C]
00B616EA ADD ECX,EAX
00B616EC MOVZX EAX,WORD PTR DS:[ECX+14]
00B616F0 LEA EDI,[ECX+18]
00B616F3 ADD EDI,EAX
00B616F5 MOVZX EAX,WORD PTR DS:[ECX+6]
00B616F9 IMUL ESI,EAX,28
00B616FC ADD ESI,EDI
00B616FE CMP EDI,ESI
00B61700 JE SHORT 00B61718
00B61702 MOV ECX,DWORD PTR SS:[ARG.2]
00B61705 CMP ECX,DWORD PTR DS:[EDX+0C]
00B61708 JB SHORT 00B61714
00B6170A MOV EAX,DWORD PTR DS:[EDX+8]
00B6170D ADD EAX,DWORD PTR DS:[EDX+0C]
00B61710 CMP ECX,EAX
00B61712 JS SHORT 00B61720
00B61714 ADD EDI,28
00B61717 CMP EDI,ESI
00B61719 JNE SHORT 00B61705
Dest=reversing_sample.00B6195B

Registers (FPU)
EAX: A9EEC8D5
ECX: 00B616D6 reversing_sample.<ModuleEntryPoint>
EDX: 00B616D6 reversing_sample.<ModuleEntryPoint>
EBX: 00E8A000
ESP: 00DCFD00
EBP: 00DCFD00
ESI: 00B616D6 reversing_sample.<ModuleEntryPoint>
EDI: 00B616D6 reversing_sample.<ModuleEntryPoint>
EIP: 00B616D6 reversing_sample.<ModuleEntryPoint>
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit E8D000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
D 0
O 0 LastErr 00000000 ERROR_SUCCESS
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
Last cwnd 0000:00000000
XMM0 00000000 00000000 00000000 00000000
XMM1 00000000 00000000 00000000 00000000
XMM2 00000000 00000000 00000000 00000000

reversing_sample.<ModuleEntryPoint>

Address	Hex	dump	ASCII		
00B7E000	73 00 00 47 6F 6F 64 0A 00 00 00 57 72 6F 6E	Good Wron		00DCFD00	75418484
00B7E010	67 0A 00 00 B1 19 BF 44 50 08 73 EC FF FF FF	Good Wron		00DCFD04	00E8A000
00B7E020	01 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD08	75418460
00B7E030	01 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD0C	A9EEC8D5
00B7E040	FF FF FF FF 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD0E	00DCFD00
00B7E050	20 05 93 19 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD10	7731305A
00B7E060	00 00 00 00 00 00 00 00 00 00 00 00 01 20	Good Wron		00DCFD14	00E8A000
00B7E070	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD18	16F23578
00B7E080	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD1C	00000000
00B7E090	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD20	00000000
00B7E0A0	00 00 00 00 02 20 00 01 00 00 00 00 00 00	Good Wron		00DCFD24	00000000
00B7E0B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD28	00000000
00B7E0C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD2C	00000000
00B7E0D0	00 00 00 00 00 00 00 00 00 00 00 00 02 20	Good Wron		00DCFD30	00000000
00B7E0E0	02 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD34	16F23578
00B7E0F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD38	00000000
00B7E100	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD3C	00000000
00B7E110	02 00 00 00 0C 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD40	00000000
00B7E120	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD44	00000000
00B7E130	00 00 00 00 00 00 00 00 FF FF FF 00 00 00	Good Wron		00DCFD48	00000000
00B7E140	00 00 00 00 00 00 00 00 80 00 0A 0A 0A 00	Good Wron		00DCFD4C	00000000
00B7E150	FF FF FF 00 00 00 00 00 00 00 00 01 00 00	Good Wron		00DCFD50	00000000
00B7E160	00 00 00 00 01 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD54	00000000
00B7E170	01 00 00 00 00 00 00 00 00 00 00 00 00 00	Good Wron		00DCFD58	00000000

Module <Mod_772A> (anonymous)

Paused

ollydbg

코드뷰

심볼 뷰

메모리뷰

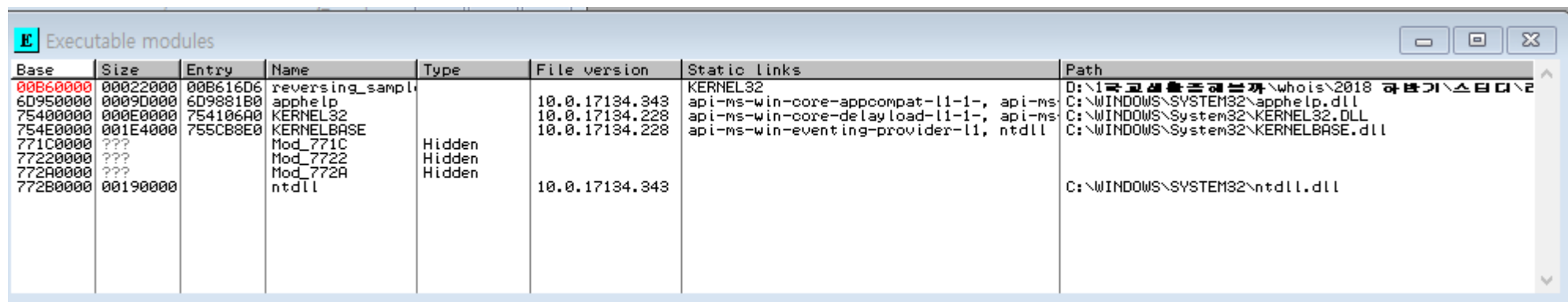


레지스터 뷰

스택 뷰

ollydbg

- 모듈 확인



Base	Size	Entry	Name	Type	File version	Static links	Path
00B60000	00022000	00B616D6	reversing_sample.exe			KERNEL32	D:\1국고생물종해분자\whois\2018 하반기\스튜디오2\...
6D950000	0009D000	6D9881B0	apphelp.dll		10.0.17134.343	api-ms-win-core-appcompat-l1-1-, api-ms-	C:\WINDOWS\SYSTEM32\apphelp.dll
75400000	000E0000	754106A0	KERNEL32		10.0.17134.228	api-ms-win-core-delayload-l1-1-, api-ms-	C:\WINDOWS\System32\KERNEL32.DLL
754E0000	001E4000	755CB8E0	KERNELBASE		10.0.17134.228	api-ms-win-eventing-provider-l1, ntdll	C:\WINDOWS\System32\KERNELBASE.dll
771C0000	???		Mod_771C	Hidden			
77220000	???		Mod_7722	Hidden			
772A0000	???		Mod_772A	Hidden			
772B0000	00190000		ntdll		10.0.17134.343		C:\WINDOWS\SYSTEM32\ntdll.dll

Alt + E 는 메모리에 불러와진 모듈의 목록을 보여준다.

EXE 프로그램이 실행될 때는 프로그램의 코드 뿐만 아니라 사용하는 라이브러리 함수의 코드가 들어있는 DLL 또한 로딩된다.

ollydbg

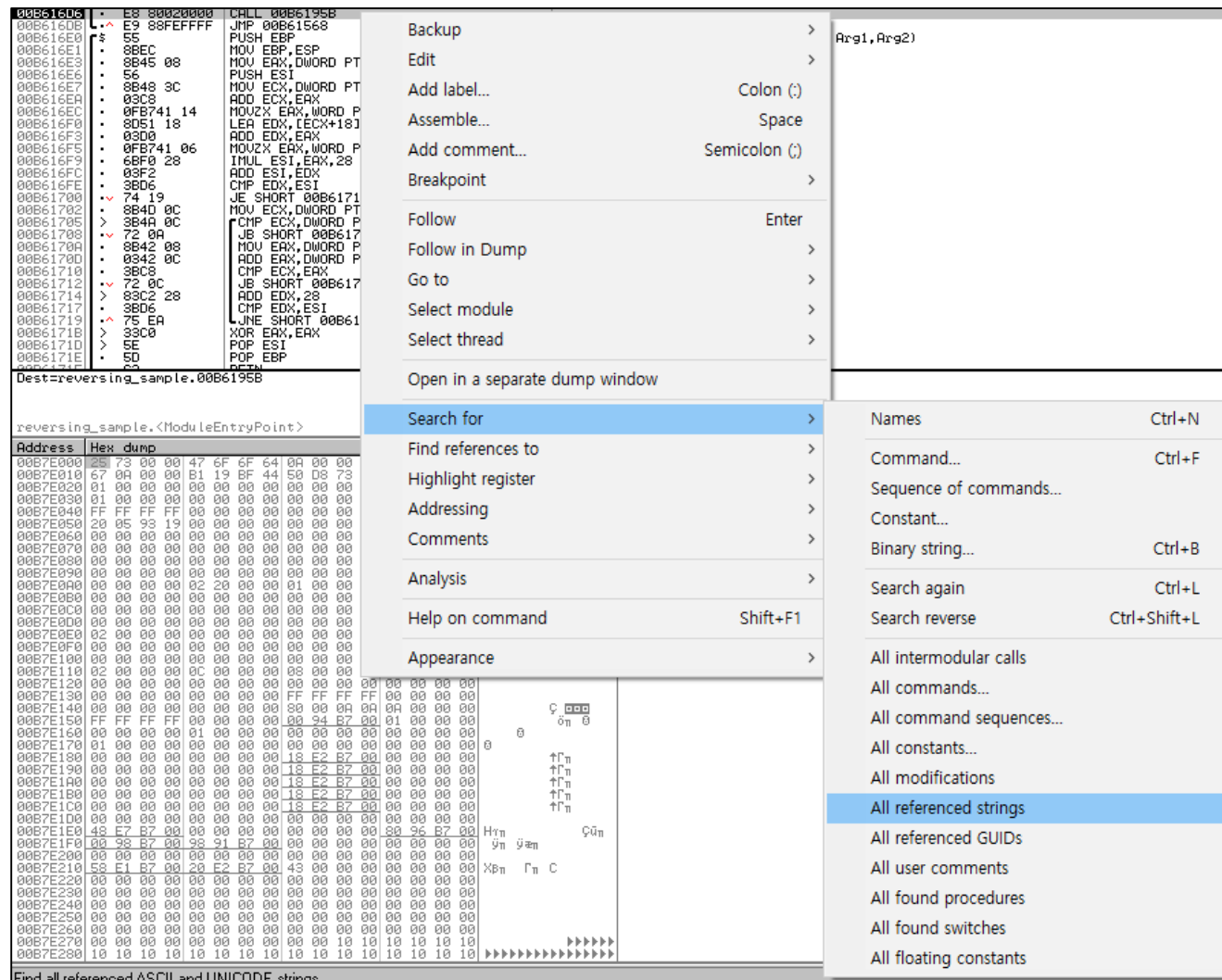
00B616C8	7 0F45 FC 7E FF	MOV DWORD PTR SS:[EBP-4],-2	
00B616CD	• 8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]	
00B616D0	> E8 B1050000	CALL 00B61C86	
00B616D5	• C3	RETN	
00B616D6	• E8 80020000	CALL 00B6195B	
00B616DB	• E9 80FEFFFF	JMP 00B61568	
00B616E0	\$ 55	PUSH EBP	reversing_sample.00B616E0(guessed Arg1,Arg2)
00B616E1	• 8BEC	MOV EBP,ESP	
00B616E3	• 8B45 08	MOV EAX,DWORD PTR SS:[ARG.1]	
00B616E6	• 56	PUSH ESI	
00B616E7	• 8B48 3C	MOV ECX,DWORD PTR DS:[EAX+3C]	
00B616EA	• 03C8	ADD ECX,EAX	
00B616EC	• 0FB741 14	MOVZX EAX,WORD PTR DS:[ECX+14]	
00B616F0	• 8D51 18	LEA EDX,[ECX+18]	
00B616F3	• 03D0	ADD EDX,EAX	
00B616F5	• 0FB741 06	MOVZX EAX,WORD PTR DS:[ECX+6]	
00B616F9	• 6BF0 28	IMUL ESI,EAX,28	
00B616FC	• 03F2	ADD ESI,EDX	
00B616FE	• 3BD6	CMP EDX,ESI	
00B61700	• 74 19	JE SHORT 00B6171B	
00B61702	• 8B4D 0C	MOV ECX,DWORD PTR SS:[ARG.2]	

IDA Freeware로 봤을 때와 주소가 좀 다른데, 이는 프로그램이 실행되면서 메모리에 직접 매핑되었기 때문임.

그래도 Offset은 동일하기 때문에 Base만 다름.

ollydbg

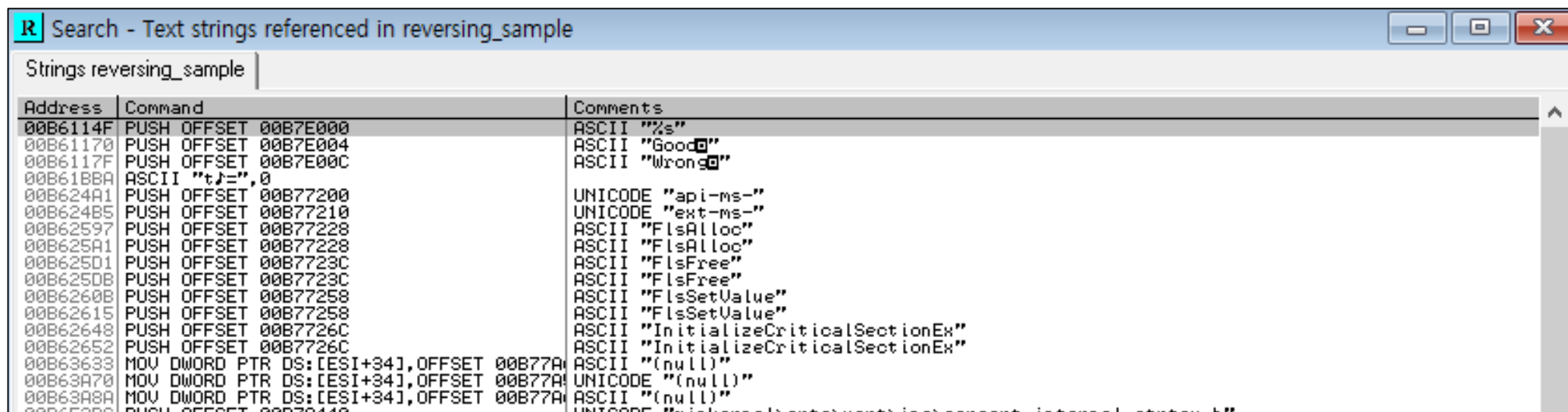
• 문자열 탐색



오른쪽 클릭 -> Search for -> All referenced strings

ollydbg

- 참조되는 문자열을 바로 볼 수 있다.



Address	Command	Comments
00B6114F	PUSH OFFSET 00B7E000	ASCII "%s"
00B61170	PUSH OFFSET 00B7E004	ASCII "Good"
00B6117F	PUSH OFFSET 00B7E00C	ASCII "Wrong"
00B61BBA	ASCII "tj=",0	
00B624A1	PUSH OFFSET 00B77200	UNICODE "api-ms-"
00B624B5	PUSH OFFSET 00B77210	UNICODE "ext-ms-"
00B62597	PUSH OFFSET 00B77228	ASCII "FlsAlloc"
00B625A1	PUSH OFFSET 00B77228	ASCII "FlsAlloc"
00B625D1	PUSH OFFSET 00B7723C	ASCII "FlsFree"
00B625DB	PUSH OFFSET 00B7723C	ASCII "FlsFree"
00B6260B	PUSH OFFSET 00B77258	ASCII "FlsSetValue"
00B62615	PUSH OFFSET 00B77258	ASCII "FlsSetValue"
00B62648	PUSH OFFSET 00B7726C	ASCII "InitializeCriticalSectionEx"
00B62652	PUSH OFFSET 00B7726C	ASCII "InitializeCriticalSectionEx"
00B63633	MOV DWORD PTR DS:[ESI+34],OFFSET 00B77A...	ASCII "(null)"
00B63A70	MOV DWORD PTR DS:[ESI+34],OFFSET 00B77A...	UNICODE "(null)"
00B63A8A	MOV DWORD PTR DS:[ESI+34],OFFSET 00B77A...	ASCII "(null)"
00B63B0C	PUSH OFFSET 00B77A40	UNICODE "c:\kernel\ext\user\api\kernel.internal.statu.b"

ollydbg

00B61060	55	PUSH EBP	
00B61061	8BEC	MOV EBP,ESP	
00B61063	83EC 28	SUB ESP,28	
00B61066	A1 18E0B700	MOV EAX,DWORD PTR DS:[0B7E018]	
00B61068	33C5	XOR EAX,EBP	
00B6106D	8945 FC	MOV DWORD PTR SS:[LOCAL.1],EAX	
00B61070	C645 DC 00	MOV BYTE PTR SS:[LOCAL.9],0	
00B61074	33C0	XOR EAX,EAX	
00B61076	8945 D0	MOV DWORD PTR SS:[EBP-23],EAX	
00B61079	8945 E1	MOV DWORD PTR SS:[EBP-1F],EAX	
00B6107C	8945 E5	MOV DWORD PTR SS:[EBP-1B],EAX	
00B6107F	66:8945 E9	MOV WORD PTR SS:[EBP-17],AX	
00B61083	B9 01000000	MOV ECX,1	
00B61088	6BD1 00	IMUL EDX,ECX,0	
00B6108B	C64415 EC 74	MOV BYTE PTR SS:[EDX+EBP-14],74	
00B61090	B8 01000000	MOV EAX,1	
00B61095	C1E0 00	SHL EAX,0	Shift out of range
00B61098	C64405 EC 68	MOV BYTE PTR SS:[EAX+EBP-14],68	
00B6109D	B9 01000000	MOV ECX,1	
00B610A2	D1E1	SHL ECX,1	
00B610A4	C6440D EC 69	MOV BYTE PTR SS:[ECX+EBP-14],69	
00B610A9	BA 01000000	MOV EDX,1	
00B610AE	6BC2 03	IMUL EAX,EDX,3	
00B610B1	C64405 EC 73	MOV BYTE PTR SS:[EAX+EBP-14],73	
00B610B6	B9 01000000	MOV ECX,1	
00B610BB	C1E1 02	SHL ECX,2	
00B610BE	C6440D EC 5F	MOV BYTE PTR SS:[ECX+EBP-14],5F	
00B610C3	BA 01000000	MOV EDX,1	
00B610C8	6BC2 05	IMUL EAX,EDX,5	
00B610CB	C64405 EC 70	MOV BYTE PTR SS:[EAX+EBP-14],70	
00B610D0	B9 01000000	MOV ECX,1	
00B610D5	6BD1 06	IMUL EDX,ECX,6	
00B610D8	C64415 EC 34	MOV BYTE PTR SS:[EDX+EBP-14],34	
00B610DD	B8 01000000	MOV EAX,1	
00B610E2	6BC8 07	IMUL ECX,EAX,7	
00B610E5	C6440D EC 35	MOV BYTE PTR SS:[ECX+EBP-14],35	
00B610EA	BA 01000000	MOV EDX,1	
00B610EF	C1E2 03	SHL EDX,3	
00B610F2	C64415 EC 35	MOV BYTE PTR SS:[EDX+EBP-14],35	
00B610F7	B8 01000000	MOV EAX,1	
00B610FC	6BC8 09	IMUL ECX,EAX,9	
00B610FF	C6440D EC 77	MOV BYTE PTR SS:[ECX+EBP-14],77	
00B61104	BA 01000000	MOV EDX,1	
00B61109	6BC2 0A	IMUL EAX,EDX,0A	
00B6110C	C64405 EC 30	MOV BYTE PTR SS:[EAX+EBP-14],30	
00B61111	B9 01000000	MOV ECX,1	
00B61116	6BD1 0B	IMUL EDX,ECX,0B	
00B61119	C64415 EC 72	MOV BYTE PTR SS:[EDX+EBP-14],72	
00B6111E	B8 01000000	MOV EAX,1	
00B61123	6BC8 0C	IMUL ECX,EAX,0C	
00B61126	C6440D EC 64	MOV BYTE PTR SS:[ECX+EBP-14],64	
00B6112B	BA 01000000	MOV EDX,1	
00B61130	6BC2 0D	IMUL EAX,EDX,0D	
00B61133	8945 D8	MOV DWORD PTR SS:[LOCAL.10],EAX	
00B61136	837D D8 0F	CMP DWORD PTR SS:[LOCAL.10],0F	
00B6113A	73 02	JAE SHORT 00B6113E	
00B6113C	EB 05	JMP SHORT 00B61143	
00B6113E	E8 7A020000	CALL 00B613BD	
00B61143	8B4D 00	MOV ECX,DWORD PTR SS:[LOCAL.10]	

문자열을 따라가다보면 main 함수를 찾을 수 있다.

ollydbg

00B6105F	CC	INT3
00B61060	55	PUSH EBP
00B61061	8BEC	MOV EBP,ESP
00B61063	83EC 28	SUB ESP,28
00B61066	A1 18E0B700	MOV EAX,DWORD PTR DS:[0B7E018]
00B61068	33C5	XOR EAX,EBP
00B6106D	8945 FC	MOV DWORD PTR SS:[LOCAL.1],EAX
00B61070	C645 DC 00	MOV BYTE PTR SS:[LOCAL.9],0
00B61074	33C0	XOR EAX,EAX
00B61076	8945 DD	MOV DWORD PTR SS:[EBP-23],EAX
00B61079	8945 E1	MOV DWORD PTR SS:[EBP-1F],EAX
00B6107C	8945 E5	MOV DWORD PTR SS:[EBP-1B],EAX
00B6107F	66:8945 E9	MOV WORD PTR SS:[EBP-17],AX
00B61083	B9 01000000	MOV ECX,1

F2 : breakpoint

00B6105F	CC	INT3
00B61060	55	PUSH EBP
00B61061	8BEC	MOV EBP,ESP
00B61063	83EC 28	SUB ESP,28
00B61066	A1 18E0B700	MOV EAX,DWORD PTR DS:[0B7E018]
00B61068	33C5	XOR EAX,EBP
00B6106D	8945 FC	MOV DWORD PTR SS:[LOCAL.1],EAX
00B61070	C645 DC 00	MOV BYTE PTR SS:[LOCAL.9],0
00B61074	33C0	XOR EAX,EAX
00B61076	8945 DD	MOV DWORD PTR SS:[EBP-23],EAX
00B61079	8945 E1	MOV DWORD PTR SS:[EBP-1F],EAX
00B6107C	8945 E5	MOV DWORD PTR SS:[EBP-1B],EAX
00B6107F	66:8945 E9	MOV WORD PTR SS:[EBP-17],AX
00B61083	B9 01000000	MOV ECX,1
00B61088	6BD1 00	IMUL EDX,ECX,0

F9 : Run

00B6105F	CC	INT3
00B61060	55	PUSH EBP
00B61061	8BEC	MOV EBP,ESP
00B61063	83EC 28	SUB ESP,28
00B61066	A1 18E0B700	MOV EAX,DWORD PTR DS:[0B7E018]
00B61068	33C5	XOR EAX,EBP
00B6106D	8945 FC	MOV DWORD PTR SS:[LOCAL.1],EAX
00B61070	C645 DC 00	MOV BYTE PTR SS:[LOCAL.9],0
00B61074	33C0	XOR EAX,EAX
00B61076	8945 DD	MOV DWORD PTR SS:[EBP-23],EAX
00B61079	8945 E1	MOV DWORD PTR SS:[EBP-1F],EAX
00B6107C	8945 E5	MOV DWORD PTR SS:[EBP-1B],EAX
00B6107F	66:8945 E9	MOV WORD PTR SS:[EBP-17],AX
00B61083	B9 01000000	MOV ECX,1
00B61088	6BD1 00	IMUL EDX,ECX,0
00B6108B	C64415 FC 74	MOV BYTE PTR SS:[FDX+FBP-14],74

Stepping

F7 : Step Into

F8 : Step Out

Registers (FPU)	
EAX	0123FED8
ECX	00000000
EDX	00000000
EBX	00E8A000
ESP	000CFD2C
EBP	000CFD54
ESI	00B7EE04 reversing_sample.00B7EE04
EDI	0123DB90
EIP	00B61066 reversing_sample.00B61066
C 0	ES 002B 32bit 0(FFFFFFFF)
P 0	CS 0023 32bit 0(FFFFFFFF)
A 1	SS 002B 32bit 0(FFFFFFFF)
Z 0	DS 002B 32bit 0(FFFFFFFF)
S 0	FS 0053 32bit E8000(FFF)
T 0	GS 002B 32bit 0(FFFFFFFF)
O 0	
0 0	LastErr 00000000 ERROR_SUCCESS
EFL	00000212 (NO,B,NE,A,NS,PO,GE,G)
ST0	empty 0.0
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0
ST4	empty 0.0
ST5	empty 0.0
ST6	empty 0.0
ST7	empty 0.0
FST	0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 (GT)
FCW	027F Prec NEAR,53 Mask 1 1 1 1 1 1
Last cmd	0000:00000000
MM0	00000000 00000000 00000000 00000000
MM1	00000000 00000000 00000000 00000000
MM2	00000000 00000000 00000000 00000000
MM3	00000000 00000000 00000000 00000000
MM4	00000000 00000000 00000000 00000000
MM5	00000000 00000000 00000000 00000000
MM6	00000000 00000000 00000000 00000000
MM7	00000000 00000000 00000000 00000000
MXCSR	00001F80 FZ 0 DZ 0 Err 0 0 0 0 0 0
	Rnd NEAR Mask 1 1 1 1 1 1

스텝이 진행될 때
변경된 레지스터는
빨간색으로 점멸됨.

ollydbg

이 함수는 scanf

이 함수는 validate

00B6114B	• 8D55 DC	LEA EDX,[LOCAL.9]	
00B6114E	• 52	PUSH EDX	
00B6114F	• 68 00E0B700	PUSH OFFSET 00B7E000	[Arg2 => OFFSET LOCAL.9 Arg1 = ASCII "%s"
00B61154	• E8 07010000	CALL 00B61260	reversing_sample.00B61260
00B61159	• 83C4 08	ADD ESP,8	
00B6115C	• 8D45 EC	LEA EAX,[LOCAL.5]	
00B6115F	• 50	PUSH EAX	[Arg2 => OFFSET LOCAL.5
00B61160	• 8D4D DC	LEA ECX,[LOCAL.9]	Arg1 => OFFSET LOCAL.9
00B61163	• 51	PUSH ECX	reversing_sample.00B61000
00B61164	• E8 97FEFFFF	CALL 00B61000	
00B61169	• 83C4 08	ADD ESP,8	
00B6116C	• 85C0	TEST EAX,EAX	
00B6116E	• 74 0F	JZ SHORT 00B6117F	
00B61170	• 68 04E0B700	PUSH OFFSET 00B7E004	ASCII "Good"
00B61175	• E8 A6000000	CALL 00B61220	
00B6117A	• 83C4 04	ADD ESP,4	
00B6117D	• EB 0D	JMP SHORT 00B6118C	
00B6117F	> 68 0CE0B700	PUSH OFFSET 00B7E00C	ASCII "Wrong"
00B61184	• E8 97000000	CALL 00B61220	

scanf에서 브레이크 포인트를 걸고 Run을 하자.

ollydbg

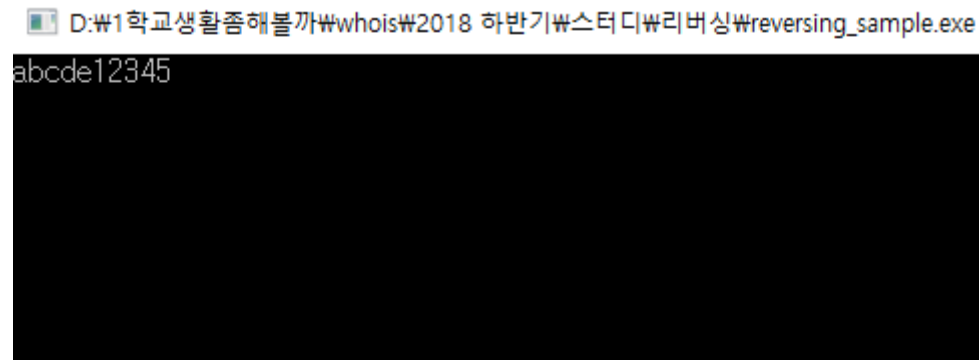
00B6114B	• 8D55 DC	LEA EDX,[LOCAL.9]	
00B6114E	• 52	PUSH EDX	[Arg2 => OFFSET LOCAL.9 Arg1 = ASCII "%s"
00B6114F	• 68 00E0B700	PUSH OFFSET 00B7E000	
00B61154	• E8 07010000	CALL 00B61260	reversing_sample.00B61260
00B61159	• 83C4 08	ADD ESP,8	
00B6115C	• 8D45 EC	LEA EAX,[LOCAL.5]	[Arg2 => OFFSET LOCAL.5
00B6115F	• 50	PUSH EAX	Arg1 => OFFSET LOCAL.9
00B61160	• 8D4D DC	LEA ECX,[LOCAL.9]	reversing_sample.00B61000
00B61163	• 51	PUSH ECX	
00B61164	• E8 97FEFFFF	CALL 00B61000	
00B61169	• 83C4 08	ADD ESP,8	
00B6116C	• 85C0	TEST EAX,EAX	

Step Into (F7)을 하면 scanf 함수 내부로 들어가게 됨.
그러므로 Step Out(F8)

ollydbg

00B6114D	52	PUSH EDI	
00B6114E	52	PUSH EDI	
00B6114F	68 00E0B700	PUSH OFFSET 00B7E000	[Arg2 => OFFSET LOCAL.9 Arg1 = ASCII "%s"
00B61154	E8 07010000	CALL 00B61260	reversing_sample.00B61260
00B61159	83C4 08	ADD ESP,8	
00B6115C	8D45 EC	LEA EAX,[LOCAL.5]	
00B6115F	50	PUSH EAX	[Arg2 => OFFSET LOCAL.5
00B61160	8D4D DC	LEA ECX,[LOCAL.9]	
00B61163	51	PUSH ECX	[Arg1 => OFFSET LOCAL.9
00B61164	E8 97FEFFFF	CALL 00B61000	reversing_sample.00B61000
00B61169	83C4 08	ADD ESP,8	
00B6116C	52	PUSH EDI	

디버거가 진행되지 않음.
왜? 입력을 기다리고 있기 때문.



입력을 해주자.

ollydbg

00B6114E	. 52	PUSH EDX	[Arg2 => OFFSET LOCAL.9 Arg1 = ASCII "%s"
00B6114F	. 68 00E0B700	PUSH OFFSET 00B7E000	
00B61154	. E8 07010000	CALL 00B61260	reversing_sample.00B61260
00B61159	. 83C4 08	ADD ESP,8	
00B6115C	. 8D45 EC	LEA EAX,[LOCAL.5]	[Arg2 => OFFSET LOCAL.5
00B6115F	. 50	PUSH EAX	
00B61160	. 8D4D DC	LEA ECX,[LOCAL.9]	Arg1 => OFFSET LOCAL.9
00B61163	. 51	PUSH ECX	reversing_sample.00B61000
00B61164	. E8 97FEFFFF	CALL 00B61000	
00B61169	. 83C4 08	ADD ESP,8	

이제서야 넘어옴.

ollydbg

00B6114B	8055 DC	LEA EDX,[LOCAL.9]	
00B6114E	52	PUSH EDX	[Arg2 => OFFSET LOCAL.9 Arg1 = ASCII "%s"
00B6114F	68 00E0B700	PUSH OFFSET 00B7E000	reversing_sample.00B61260
00B61154	E8 07010000	CALL 00B61260	
00B61159	83C4 08	ADD ESP,8	
00B6115C	8045 EC	LEA EAX,[LOCAL.5]	[Arg2 => OFFSET LOCAL.5
00B6115F	50	PUSH EAX	Arg1 => OFFSET LOCAL.9
00B61160	8040 DC	LEA ECX,[LOCAL.9]	reversing_sample.00B61000
00B61163	51	PUSH ECX	
00B61164	E8 97FEFFFF	CALL 00B61000	
00B61169	83C4 08	ADD ESP,8	
00B6116C	85C0	TEST EAX,EAX	
00B6116E	74 0F	JZ SHORT 00B6117F	
00B61170	68 04E0B700	PUSH OFFSET 00B7E004	ASCII "Good"
00B61175	E8 A6000000	CALL 00B61220	
00B6117A	83C4 04	ADD ESP,4	
00B6117D	EB 0D	JMP SHORT 00B6118C	
00B6117F	68 0CE0B700	PUSH OFFSET 00B7E00C	ASCII "Wrong"
00B61184	E8 97000000	CALL 00B61220	
00B61189	83C4 04	ADD ESP,4	
00B6118C	EB 02	JMP SHORT 00B61190	
00B6118E	EB 02	JMP SHORT 00B61192	
00B61190	33C0	XOR EAX,EAX	
00B61192	8B4D FC	MOV ECX,DWORD PTR SS:[EBP-4]	
00B61195	33CD	XOR ECX,EBP	
00B61197	E8 FD020000	CALL 00B61499	
00B6119C	00FF	CALL ESP	

Stack address=00DCFD30 (current registers)
EDX=0000000A (current registers)

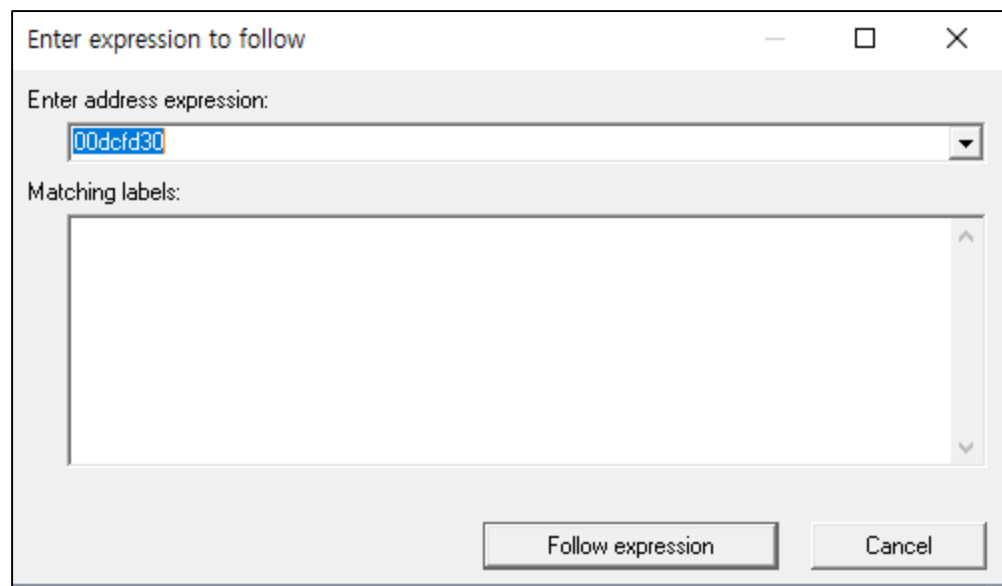
ollydbg에서는 지역변수를 [LOCAL.X] 와 같이 표현함.
이는 [EBP-X]와 같은 표현

[LOCAL.9] 가 어딘지 알려줌. 여기서는
Stack Address => 0x00DCFD30

ollydbg

Address	Hex dump	ASCII
00B7E000	25 73 00 00 47 6F 6F 64 0A 00 00 00 57 72 6F 6E	%s Good! Wron
00B7E010	67 0A 00 00 AF 27 8C 13 50 08 73 EC FF FF FF FF	g >> 'i!!F+s
00B7E020	01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0
00B7E030	2F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	/
00B7E040	01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0
00B7E050	20 05 93 19 00 00 00 00 00 00 00 00 00 00 00 00	0 0 0
00B7E060	E2 FF 23 01 08 FF 23 01 01 00 00 00 41 20 00 00	r #0 0 #00 A
00B7E070	00 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00	
00B7E080	FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00	
00B7E090	00 00 00 00 A0 0F 00 00 00 00 00 00 00 00 00 00	0*
00B7E0A0	00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00	0

메모리 뷰에서 ctrl+G



아까 찾은 스택 주소를 넣으면

ollydbg

Address	Hex dump																ASCII
00DCFD30	61	62	63	64	65	31	32	33	34	35	00	00	00	00	00	00	abode12345
00DCFD40	74	68	69	73	5F	70	34	35	35	77	30	72	64	00	B6	00	this_p455w0rd
00DCFD50	04	25	AF	EC	9C	FD	DC	00	60	16	B6	00	01	00	00	00	◆%>0002 ■ ' _ 0
00DCFD60	90	DB	23	01	D8	FE	23	01	CC	25	AF	EC	D6	16	B6	00	■ #01 ■ #01f%>00π-
00DCFD70	D6	16	B6	00	00	A0	E8	00	00	00	00	00	D6	16	B6	00	π- á\$ π-
00DCFD80	D6	16	B6	00	68	FD	DC	00	00	00	00	00	E8	FD	DC	00	π- h² ■ \$² ■
00DCFD90	90	1E	B6	00	28	11	C4	EC	00	00	00	00	B0	FD	DC	00	É▲ (←-∞ ■² ■
00DCFDA0	84	84	41	75	00	A0	E8	00	60	84	41	75	D5	C8	EE	A9	ääAu á\$ 'äAu fter
00DCFDB0	F8	FD	DC	00	5A	30	31	77	00	A0	E8	00	78	85	F2	16	°² ■ Z01w á\$ xâ¿-
00DCFDC0	00	00	00	00	00	00	00	00	00	A0	E8	00	00	00	00	00	á\$
00DCFDD0	00	00	00	00	00	00	00	00	00	00	00	00	78	85	F2	16	xâ¿-
00DCFDE0	BC	FD	DC	00	00	00	00	00	00	FE	DC	00	80	25	32	77	u² ■ ■ ■ Q%2w
00DCDF00	10	85	14	61	00	00	00	00	08	FE	DC	00	2A	30	31	77	▷â¶a ■ ■ *01w
00DCFE00	FF	FF	FF	FF	BD	EC	32	77	00	00	00	00	00	00	00	00	u²w
00DCFE10	D6	16	B6	00	00	A0	E8	00	00	00	00	00	00	00	00	00	π- á\$
00DCFE20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

우리가 넣은 값이 보임 !

ollydbg

다시 돌아와서.

00B6114B	• 8D55 DC	LEA EDX,[LOCAL.9]	
00B6114E	• 52	PUSH EDX	
00B6114F	• 68 00E0B700	PUSH OFFSET 00B7E000	[Arg2 => OFFSET LOCAL.9 Arg1 = ASCII "%s"
00B61154	• E8 07010000	CALL 00B61260	reversing_sample.00B61260
00B61159	• 83C4 08	ADD ESP,8	
00B6115C	• 8D45 EC	LEA EAX,[LOCAL.5]	
00B6115F	• 50	PUSH EAX	[Arg2 => OFFSET LOCAL.5
00B61160	• 8D4D DC	LEA ECX,[LOCAL.9]	
00B61163	• 51	PUSH ECX	Arg1 => OFFSET LOCAL.9
00B61164	• E8 97FEFFFF	CALL 00B61000	reversing_sample.00B61000
00B61169	• 83C4 08	ADD ESP,8	

validate 함수에 break point를 걸고 Run

ollydbg

00B6114E	• 52	PUSH EDX	[Arg2 => OFFSET LOCAL.9
00B6114F	• 68 00E0B700	PUSH OFFSET 00B7E000	Arg1 = ASCII "%s"
00B61154	• E8 07010000	CALL 00B61260	reversing_sample.00B61260
00B61159	• 83C4 08	ADD ESP,8	
00B6115C	• 8D45 EC	LEA EAX,[LOCAL.5]	[Arg2 => OFFSET LOCAL.5
00B6115F	• 50	PUSH EAX	Arg1 => OFFSET LOCAL.9
00B61160	• 8D4D DC	LEA ECX,[LOCAL.9]	
00B61163	• 51	PUSH ECX	
00B61164	• E8 97FEFFFF	CALL 00B61000	reversing_sample.00B61000
00B61169	• 83C4 08	ADD ESP,8	

validate 함수는 Step Into(F7)로 들어가서 분석하자.

ollydbg

00B61000	\$ 55	PUSH EBP	reversing_sample.00B61000(guessed Arg1,Arg2)
00B61001	8BEC	MOV EBP,ESP	
00B61003	51	PUSH ECX	
00B61004	C745 FC 0000	MOV DWORD PTR SS:[LOCAL.1],0	
00B6100B	C745 FC 0000	MOV DWORD PTR SS:[LOCAL.1],0	
00B61012	EB 09	JMP SHORT 00B6101D	
00B61014	8B45 FC	MOV EAX,DWORD PTR SS:[LOCAL.1]	
00B61017	83C0 01	ADD EAX,1	
00B6101A	8945 FC	MOV DWORD PTR SS:[LOCAL.1],EAX	
00B6101D	8B4D 0C	MOV ECX,DWORD PTR SS:[ARG.2]	
00B61020	51	PUSH ECX	
00B61021	E8 BA770000	CALL 00B687E0	
00B61026	83C4 04	ADD ESP,4	
00B61029	3945 FC	CMP DWORD PTR SS:[LOCAL.1],EAX	
00B6102C	7D 1C	JGE SHORT 00B6104A	
00B6102E	8B55 0C	MOV EDX,DWORD PTR SS:[ARG.2]	
00B61031	0355 FC	ADD EDX,DWORD PTR SS:[LOCAL.1]	
00B61034	0FBE02	MOVSX EAX,BYTE PTR DS:[EDX]	
00B61037	8B4D 08	MOV ECX,DWORD PTR SS:[ARG.1]	
00B6103A	034D FC	ADD ECX,DWORD PTR SS:[LOCAL.1]	
00B6103D	0FBF11	MOVSX EDI,BYTE PTR DS:[ECX]	
00B61040	3BC2	CMP EAX,EDX	
00B61042	74 04	JE SHORT 00B61048	
00B61044	33C0	XOR EAX,EAX	
00B61046	EB 07	JMP SHORT 00B6104F	
00B61048	EB CA	JMP SHORT 00B61014	
00B6104A	B8 01000000	MOV EAX,1	
00B6104F	8BE5	MOV ESP,EBP	
00B61051	5D	POP EBP	
00B61052	C3	RETN	
00B61053	CC	INT3	

저번시간에 정적 분석했듯이
인자로 넘어온 두 배열을 한
글자 씩 비교하는 부분임.

ollydbg

00B61000	55	PUSH EBP
00B61001	8BEC	MOV EBP,ESP
00B61003	51	PUSH ECX
00B61004	C745 FC 0000	MOV DWORD PTR SS:[LOCAL.1],0
00B6100B	C745 FC 0000	MOV DWORD PTR SS:[LOCAL.1],0
00B61012	EB 09	JMP SHORT 00B6101D
00B61014	8B45 FC	MOV EAX,DWORD PTR SS:[LOCAL.1]
00B61017	83C0 01	ADD EAX,1
00B6101A	8945 FC	MOV DWORD PTR SS:[LOCAL.1],EAX
00B6101D	8B4D 0C	MOV ECX,DWORD PTR SS:[ARG.2]
00B61020	51	PUSH ECX
00B61021	E8 BA770000	CALL 00B687E0
00B61026	83C4 04	ADD ESP,4
00B61029	3945 FC	CMP DWORD PTR SS:[LOCAL.1],EAX
00B6102C	7D 1C	JGE SHORT 00B6104A
00B6102E	8B55 0C	MOV EDX,DWORD PTR SS:[ARG.2]
00B61031	0355 FC	ADD EDX,DWORD PTR SS:[LOCAL.1]
00B61034	0FB0 02	MOVSX EAX,BYTE PTR DS:[EDX]
00B61037	8B4D 08	MOV ECX,DWORD PTR SS:[ARG.1]
00B6103A	034D FC	ADD ECX,DWORD PTR SS:[LOCAL.1]
00B6103D	0FB0 11	MOVSX EDX,BYTE PTR DS:[ECX]
00B61040	3BC2	CMP EAX,EDX
00B61042	74 04	JE SHORT 00B61048
00B61044	33C0	XOR EAX,EAX
00B61046	EB 07	JMP SHORT 00B6104F
00B61048	EB CA	JMP SHORT 00B61014
00B6104A	B8 01000000	MOV EAX,1
00B6104F	8BE5	MOV ESP,EBP
00B61051	5D	POP EBP
00B61052	C3	RETN

CMP EAX, EDX에 Breakpoint를 걸고 Run 한다.

Registers (FPU)	
EAX	00000074
ECX	00DCFD30 ASCII "abode12345"
EDX	00000061
EBX	00E8A000
ESP	00DCFD18
EBP	00DCFD1C
ESI	00B7EE04 reversing_sample.00B7EE04
EDI	0123DB90
EIP	00B61040 reversing_sample.00B61040
C 0	ES 002B 32bit 0(FFFFFFFF)
P 1	CS 0023 32bit 0(FFFFFFFF)
A 0	SS 002B 32bit 0(FFFFFFFF)
Z 0	DS 002B 32bit 0(FFFFFFFF)
S 0	FS 0053 32bit E8D000(FFF)
T 0	GS 002B 32bit 0(FFFFFFFF)
D 0	
O 0	LastErr 00000000 ERROR_SUCCESS
EFL	00000206 (NO,NB,NE,A,NS,PE,GE,G)

EAX : 0x74

EDX : 0x61

서로 다른 것을 볼 수 있음.

EDX는 우리가 입력한 값이고 [ARG.1]에서 옴
EAX는 대상 값이고 [ARG.2]에서 옴.
그러면 [ARG.2]를 확인하면 됨.

ollydbg

00B6102C	70 1C	JGE SHORT 00B61044
00B6102E	8B55 0C	MOV EDX,DWORD PTR SS:[ARG.2]
00B61031	0355 FC	ADD EDX,DWORD PTR SS:[LOCAL.1]
00B61034	0FB0 02	MOVSX EAX,BYTE PTR DS:[EDX]
00B61037	8B4D 08	MOV ECX,DWORD PTR SS:[ARG.1]
00B6103A	034D FC	ADD ECX,DWORD PTR SS:[LOCAL.1]
00B6103D	0FB0 11	MOVSX EDX,BYTE PTR DS:[ECX]
00B61040	3BC2	CMP EAX,EDX
00B61042	74 04	JE SHORT 00B61048
00B61044	33C0	XOR EAX,EAX
00B61046	EB 07	JMP SHORT 00B6104F
00B61048	EB 0A	JMP SHORT 00B61014
00B6104A	B8 01000000	MOV EAX,1
00B6104F	8BE5	MOV ESP,EBP
00B61051	5D	POP EBP
00B61053	CC	RET

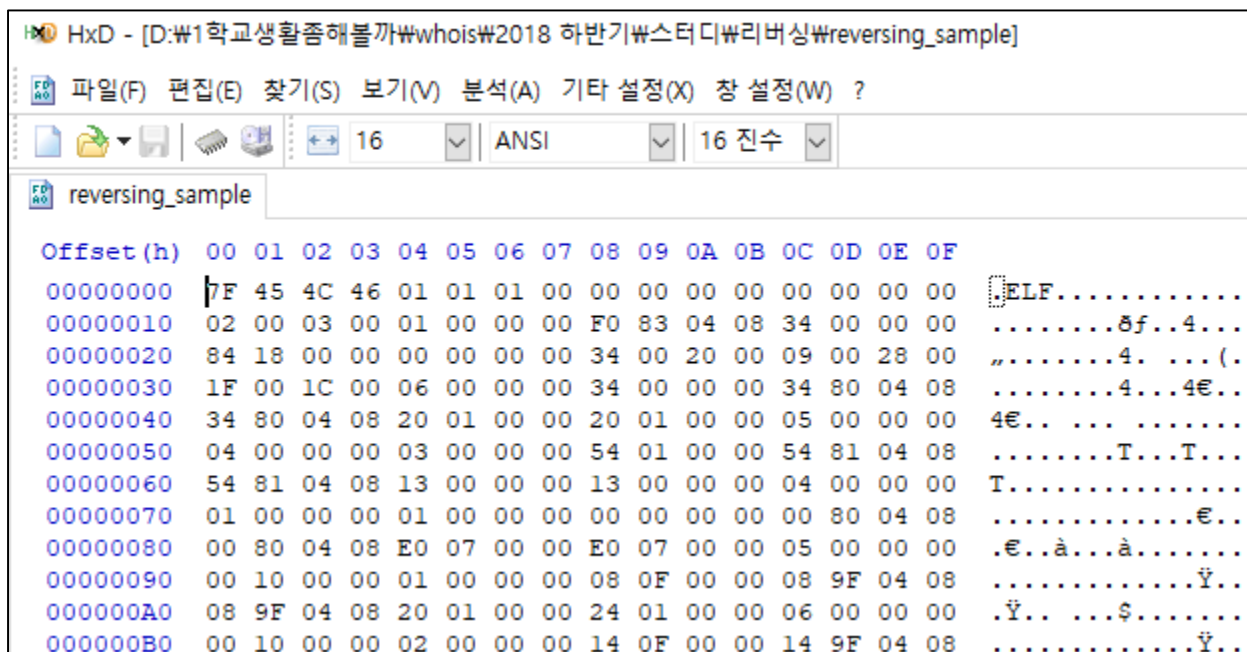
[ARG.2]가 참조되는 부분을 클릭하고.

```
Stack [00DCFD28]=00DCFD40, ASCII "this_p455w0rd" (current registers)
EDX=00000061 (decimal 97.) (current registers)
```

해당 값을 심볼 뷰에서 확인하면
대상 문자열을 찾아낼 수 있음.

ELF

- 리눅스에서 실행되는 바이너리.
- gcc -o a a.c 로 만들어진 a 파일 또한 ELF 구조임



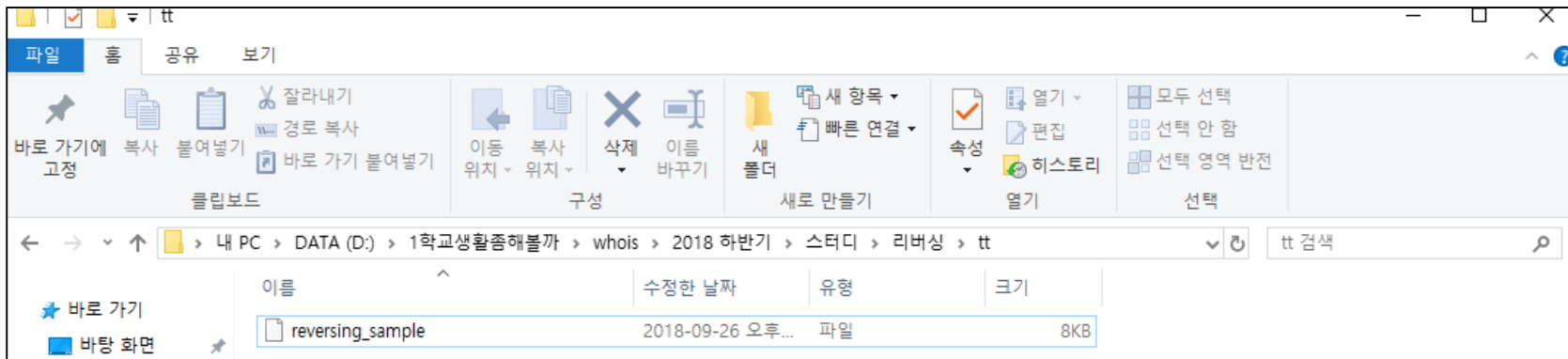
```
HxD - [D:\#1학교생활준해볼까\whois\2018 하반기\스터디\리버싱\reversing_sample]
파일(F) 편집(E) 찾기(S) 보기(V) 분석(A) 기타 설정(X) 창 설정(W) ?
16 ANSI 16 진수
reversing_sample

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00 ELF.....
00000010 02 00 03 00 01 00 00 00 F0 83 04 08 34 00 00 00 .....8f..4...
00000020 84 18 00 00 00 00 00 00 34 00 20 00 09 00 28 00 .....4. ...(.
00000030 1F 00 1C 00 06 00 00 00 34 00 00 00 34 80 04 08 .....4...4€..
00000040 34 80 04 08 20 01 00 00 20 01 00 00 05 00 00 00 4€.. ... ..
00000050 04 00 00 00 03 00 00 00 54 01 00 00 54 81 04 08 .....T...T...
00000060 54 81 04 08 13 00 00 00 13 00 00 00 04 00 00 00 T.....
00000070 01 00 00 00 01 00 00 00 00 00 00 00 00 80 04 08 .....€..
00000080 00 80 04 08 E0 07 00 00 E0 07 00 00 05 00 00 00 .€..à...à.....
00000090 00 10 00 00 01 00 00 00 08 0F 00 00 08 9F 04 08 .....ÿ..
000000A0 08 9F 04 08 20 01 00 00 24 01 00 00 06 00 00 00 .ÿ.. ...$.
000000B0 00 10 00 00 02 00 00 00 14 0F 00 00 14 9F 04 08 .....ÿ..
```

ELF

- IDA는 ELF또한 분석 가능합니다.

<https://drive.google.com/open?id=1tfy3qzYPUHfmCxKJFuLEss0Euh2qtsgg>

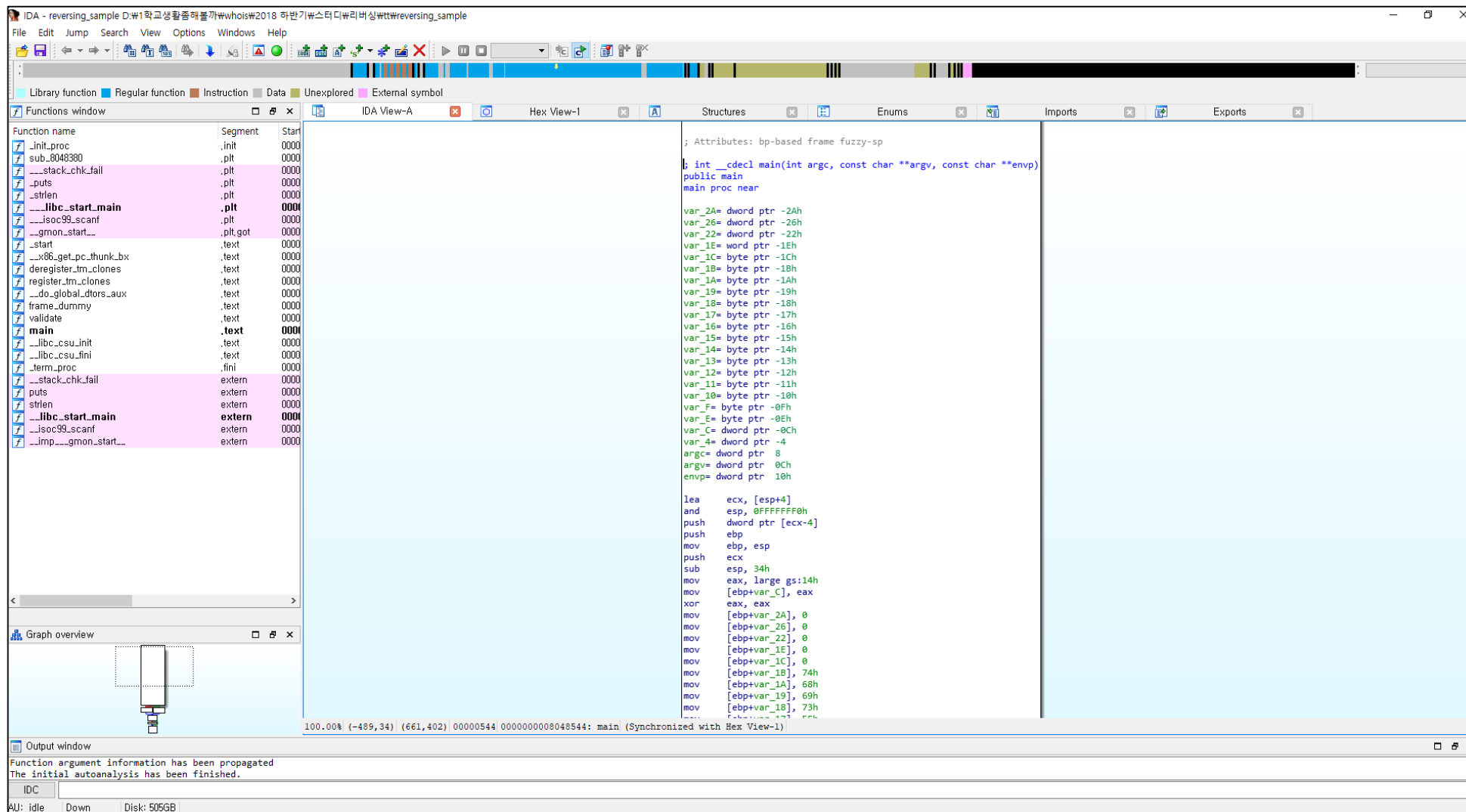


ELF

- 리눅스 환경이라면 실행 가능한 파일임!

```
win32virus@ubuntu:~/whois/reversing/week3$ ./reversing_sample  
123  
Wrong  
win32virus@ubuntu:~/whois/reversing/week3$
```

ELF



main 함수도 찾아줌..!

ELF

```
lea    ecx, [esp+4]
and     esp, 0FFFFFFF0h
push    dword ptr [ecx-4]
push    ebp
mov     ebp, esp
push    ecx
sub     esp, 34h
mov     eax, large gs:14h
mov     [ebp+var_C], eax
xor     eax, eax
mov     [ebp+var_2A], 0
mov     [ebp+var_26], 0
mov     [ebp+var_22], 0
mov     [ebp+var_1E], 0
mov     [ebp+var_1C], 0
mov     [ebp+var_1B], 74h
mov     [ebp+var_1A], 68h
mov     [ebp+var_19], 69h
mov     [ebp+var_18], 73h
mov     [ebp+var_17], 5Fh
mov     [ebp+var_16], 70h
mov     [ebp+var_15], 34h
mov     [ebp+var_14], 35h
mov     [ebp+var_13], 35h
mov     [ebp+var_12], 77h
mov     [ebp+var_11], 30h
mov     [ebp+var_10], 72h
mov     [ebp+var_F], 64h
mov     [ebp+var_E], 0
sub     esp, 8
lea     eax, [ebp+var_2A]
push    eax
push    offset unk_80486B0
call    ___isoc99_scanf
add     esp, 10h
sub     esp, 8
lea     eax, [ebp+var_1B]
push    eax
lea     eax, [ebp+var_2A]
push    eax
call    validate
add     esp, 10h
test    eax, eax
jz      short loc_80485F4
```

Visual Studio로 컴파일 한 것
에 비해 코드가 깔끔하다.

Symbol 또한 살아있어 원래
함수 이름을 볼 수 있다.

ELF

Function name	Segment	Start
f _init_proc	.init	0000
f sub_8048380	.plt	0000
f ___stack_chk_fail	.plt	0000
f _puts	.plt	0000
f _strlen	.plt	0000
f ___libc_start_main	.plt	0000
f ___isoc99_scanf	.plt	0000
f __gmon_start__	.plt.got	0000
f _start	.text	0000
f __x86_get_pc_thunk_bx	.text	0000
f deregister_tm_clones	.text	0000
f register_tm_clones	.text	0000
f __do_global_ctors_aux	.text	0000
f frame_dummy	.text	0000
f validate	.text	0000
f main	.text	0000
f __libc_csu_init	.text	0000
f __libc_csu_fini	.text	0000
f _term_proc	.fini	0000
f ___stack_chk_fail	extern	0000
f puts	extern	0000
f strlen	extern	0000
f ___libc_start_main	extern	0000
f ___isoc99_scanf	extern	0000
f __imp___gmon_start__	extern	0000

EXE보다 간결하고 Import 하는 함수가 적음.

```
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use32
assume cs:text
;org 80483F0h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing

; Attributes: noreturn fuzzy-sp

public _start
_start proc near
xor     ebp, ebp
pop     esi
mov     ecx, esp
and     esp, 0FFFFFF0h
push    eax
push    esp                ; stack_end
push    edx                ; rtld_fini
push    offset __libc_csu_fini ; fini
push    offset __libc_csu_init ; init
push    ecx                ; ubp_av
push    esi                ; argc
push    offset main        ; main
call    __libc_start_main
hlt
_start endp
```

ELF에서는 __libc_start_main 함수가 main함수를 호출함.

GDB

- GNU project debugger
- 만든사람 : 리차드 스톨만
- gdb 바이너리
- ELF 바이너리 전용

GDB

- gdb 프로그램 이름


```
ubuntu@ip-172-31-8-45:~/whois$ ls
a  a.c  bb  bb.s
ubuntu@ip-172-31-8-45:~/whois$ gdb a
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a...(no debugging symbols found)...done.
(gdb)
```


GDB 명령

- 코드 확인

disas [심볼 이름]

ex) disas main

```
(gdb) disas main
Dump of assembler code for function main:
   0x080483f7 <+0>:      push    %ebp
   0x080483f8 <+1>:      mov     %esp,%ebp
   0x080483fa <+3>:      call   0x80483ed <func1>
   0x080483ff <+8>:      pop     %ebp
   0x08048400 <+9>:      ret
End of assembler dump.
(gdb) 
```

GDB 명령

- 브레이크 포인트 걸기

b *[주소]

b *[심볼]

ex1) b *main+4

ex2) b *0x08048400

GDB 명령

- 프로그램 시작

(gdb) r

- 프로그램 재개, 브레이크 포인트 걸리고 난 후

(gdb) c

GDB 명령

- 데이터 조회할 때

기본 형태 : x/[number][type] [주소]

ex1) x/20wx \$esp

ex2) x/100wx 0x08048400

GDB 명령

- type 종류

b : 1바이트

w : 4바이트

g : 8바이트

s : 문자열

i : instruction

ex3) x/s 0x08048400

ex4) x/20i \$eip

GDB 명령

- 한 명령씩 실행

si : step instruction, 함수 내부로 들어감

ni : next instruction, 함수 내부로 들어가지 않음

```
(gdb) disas main
Dump of assembler code for function main:
   0x080483f7 <+0>:      push    %ebp
   0x080483f8 <+1>:      mov     %esp,%ebp
=> 0x080483fa <+3>:      call    0x80483ed <func1>
   0x080483ff <+8>:      pop     %ebp
   0x08048400 <+9>:      ret
End of assembler dump.
(gdb) x/i $eip
=> 0x80483fa <main+3>:  call    0x80483ed <func1>
(gdb) █
```

GDB 명령

- ni를 쳤을 때

```
(gdb) ni
0x080483ff in main ()
(gdb) x/i $eip
=> 0x080483ff <main+8>:  pop    %ebp
(gdb) █
```

호출되는 함수 건너 뛴!

- si를 쳤을 때


```
(gdb) si
0x080483ed in func1 ()
(gdb) x/i $eip
=> 0x080483ed <func1>:  push   %ebp
(gdb) █
```

호출되는 함수 안으로 들어감!

GDB 명령

- 레지스터 상태 보기

info reg

```
(gdb) info reg
eax                0x1          1
ecx                0x77e9ef82     2011819906
edx                0xffffd734     -10444
ebx                0xf7fcd000     -134426624
esp                0xffffd704     0xffffd704
ebp                0xffffd708     0xffffd708
esi                0x0           0
edi                0x0           0
eip                0x80483ed       0x80483ed <func1>
eflags             0x246         [ PF ZF IF ]
cs                 0x23          35
ss                 0x2b          43
ds                 0x2b          43
es                 0x2b          43
fs                 0x0           0
gs                 0x63          99
(gdb) 
```


GDB 명령

- set disassembly-flavor intel
intel 문법으로 어셈블리 출력

```
(gdb) set disassembly-flavor intel
(gdb) disas main
Dump of assembler code for function main:
0x08048544 <+0>:    lea     ecx,[esp+0x4]
0x08048548 <+4>:    and     esp,0xffffffff
0x0804854b <+7>:    push   DWORD PTR [ecx-0x4]
0x0804854e <+10>:   push   ebp
0x0804854f <+11>:   mov     ebp,esp
0x08048551 <+13>:   push   ecx
0x08048552 <+14>:   sub     esp,0x34
0x08048555 <+17>:   mov     eax,gs:0x14
0x0804855b <+23>:   mov     DWORD PTR [ebp-0xc],eax
0x0804855e <+26>:   xor     eax,eax
0x08048560 <+28>:   mov     DWORD PTR [ebp-0x2a],0x0
0x08048567 <+35>:   mov     DWORD PTR [ebp-0x26],0x0
0x0804856e <+42>:   mov     DWORD PTR [ebp-0x22],0x0
0x08048575 <+49>:   mov     WORD PTR [ebp-0x1e],0x0
```

동적분석 & 정적분석

- 정적분석과 동적분석을 적절히 혼합해서 잘 사용해야함
- 정적분석으로 큰 그림을 그리고 동적분석으로 스케치를 하는 식
- 정적분석에서 보이지 않았던 부분이 동적분석에서 보일 수 있음 (메모리 상태 등)

Summary

- IDA 7.0 freeware
- main 함수는 프로그램의 시작주소가 아니다.
- IDA를 통한 정적분석하기
- 분석을 통한 판단 및 추론

과제

- 샘플 프로그램을 Debug로 빌드했을 때와 Release로 빌드했을 때, IDA로 열면 무엇이 다른지 비교해보기.
- reversing.kr Easy Crack 풀기.
- openCTF simple reversing 풀기.

과제2

- openctf simple reversing 동적분석으로 풀어오기
- reversing.kr Easy Keygen 풀어오기
- reversing.kr Easy ELF 풀어오기