

2026 상반기 DFRC 신입생 세미나

Reversing Assignment

2026-01-20 (Tue)

조민혁 (Jo Min Hyuk)

cgumgek8@gmail.com



고려대학교 정보보호대학원
Korea University
School of Cybersecurity

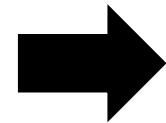


- **Introduction to Assignment**
- **Analysis Environment**
- **Problem Analysis and Solution**
- **Conclusion**

Introduction to Assignment

Introduction to Assignment

Initial Screen



```
=====
Reverse Engineering Test for First Semester of 2026
-x64 binary version-
Made by. SHS
=====
Please enter an ID within 10 characters : |
```

Introduction to Assignment

Notice

- **No security vulnerabilities involved**
 - 1. Solve the problem using only **static and dynamic analysis**.
 - 2. Solving the problem using methods other than reverse engineering that exploit vulnerabilities will **not receive any additional credit**.
- **Submit answers including both the solving process and the final answer**
 - 1. Be sure to include the **core parts** in your answer.
 - 2. For problems where screen capture is possible, **include screen captures** in your answer.

Analysis Environment

Environment and Tools

■ Analysis Environment

- Windows 11, x64 Arch

OS 이름
버전
기타 OS 설명
OS 제조업체
시스템 이름
시스템 제조업체
시스템 모델
시스템 종류
시스템 SKU
프로세서

Microsoft Windows 11 Home
10.0.26100 빌드 26100
사용할 수 없음
Microsoft Corporation
MINHYUK
SAMSUNG ELECTRONICS CO., LTD.
950QCG
x64 기반 PC
SCAI-A5A5-A5A5-A5A5-PAJC
Intel(R) Core(TM) i5-1035G4 CPU @ 1.10GHz, 1498Mhz, 4 코어, 8 논리 프...

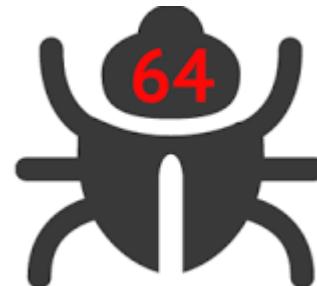
■ Static Analysis Tool

- Ghidra 11.4.2 Public



■ Dynamic Analysis Tool

- x64dbg



Problem Analysis and Solution

Problem Analysis and Solution

Launch Program

2026_freshman_x64_rev.exe.exe

```
=====
Reverse Engineering Test for First Semester of 2026
-x64 binary version-
Made by. SHS
=====
```

Please enter an ID within 10 characters : |

Input ID

```
=====
Reverse Engineering Test for First Semester of 2026
-x64 binary version-
Made by. SHS
=====
```

Please enter an ID within 10 characters : minhyuk

Please enter a flag : |

Input flag

```
=====
Reverse Engineering Test for First Semester of 2026
-x64 binary version-
Made by. SHS
=====
```

Please enter an ID within 10 characters : minhyuk

Please enter a flag : min

[System] Welcome to DFRC!

[System] Incorrect! Please try again...:(

Problem Analysis and Solution

Basic Problems

- [문제 1] 동적 분석과 정적 분석에 대해 간단히 설명하고, 리버싱에서 두 분석이 모두 필요한 이유에 대해 간단히 서술하시오. (1점)

- **Dynamic Analysis?**

→ A method of analysis that observes a program's behavior while it is actually running. It involves using a debugger to analyze runtime behavior by examining **registers, memory changes, API calls, branches, and executed code**.

- **Static Analysis?**

→ A method of analysis performed without executing the program. It analyzes the overall structure of the program by examining binaries or source code, **including assembly code, control flow, strings, and function structures**.

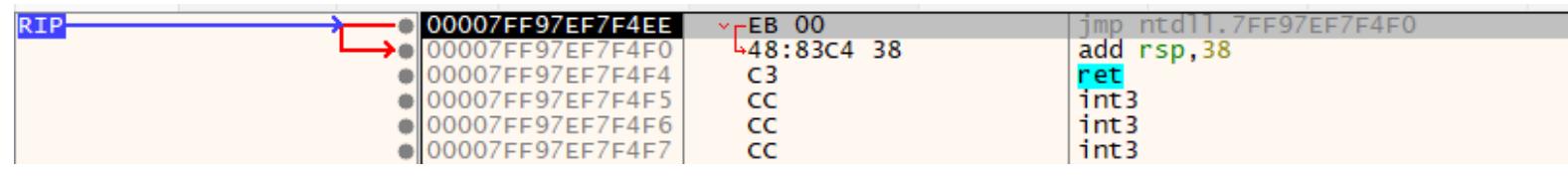
Basic Problems

- [문제 1] 동적 분석과 정적 분석에 대해 간단히 설명하고, 리버싱에서 두 분석이 모두 필요한 이유에 대해 간단히 서술하시오. (1점)
 - Why are both analyses necessary in reverse engineering?
 - Static analysis alone makes it difficult to identify values determined at **runtime, conditional behavior, obfuscated code, or packed code.**
 - Dynamic analysis alone makes it hard to systematically understand **the entire structure of the binary or all possible execution paths.**
 - Therefore, **static analysis** is used to understand **the overall structure and determine an analysis strategy**, while **dynamic analysis** is used to analyze **actual behavior and hidden logic**, making reverse engineering more effective.

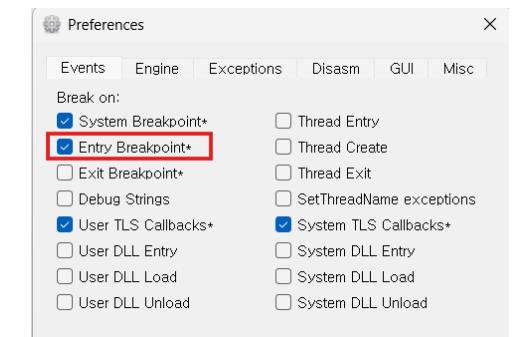
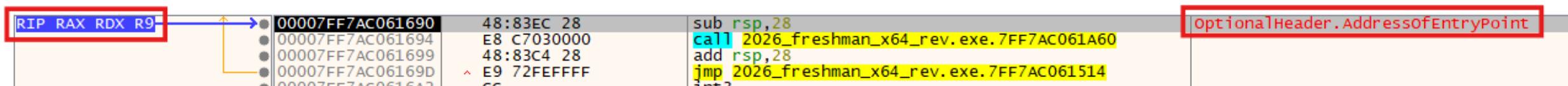
Problem Analysis and Solution

Basic Problems

- [문제 2] 동적 분석 도구를 통해 문제 파일을 실행한 후, 해당 파일의 EP(Entry Point)를 식별하시오. (2점)



↓ F9 Click

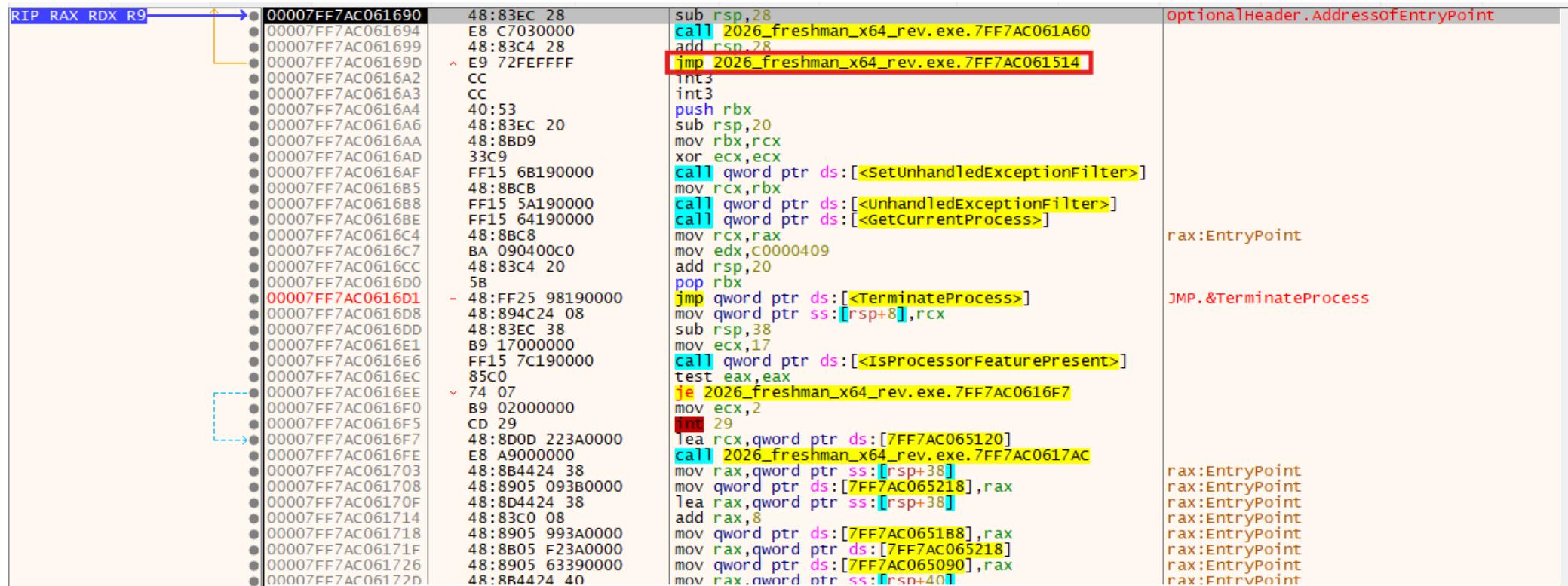


- Answer: Address of Entry Point (EP) is **0x7FF7AC061690**

Problem Analysis and Solution

Core Problems

- [문제 3] 동적 분석을 통해 EP로부터 main함수를 찾아가는 과정을 서술하시오. (2점)
 - STEP 1-1) 'F8' Click, then **jmp** occurs.



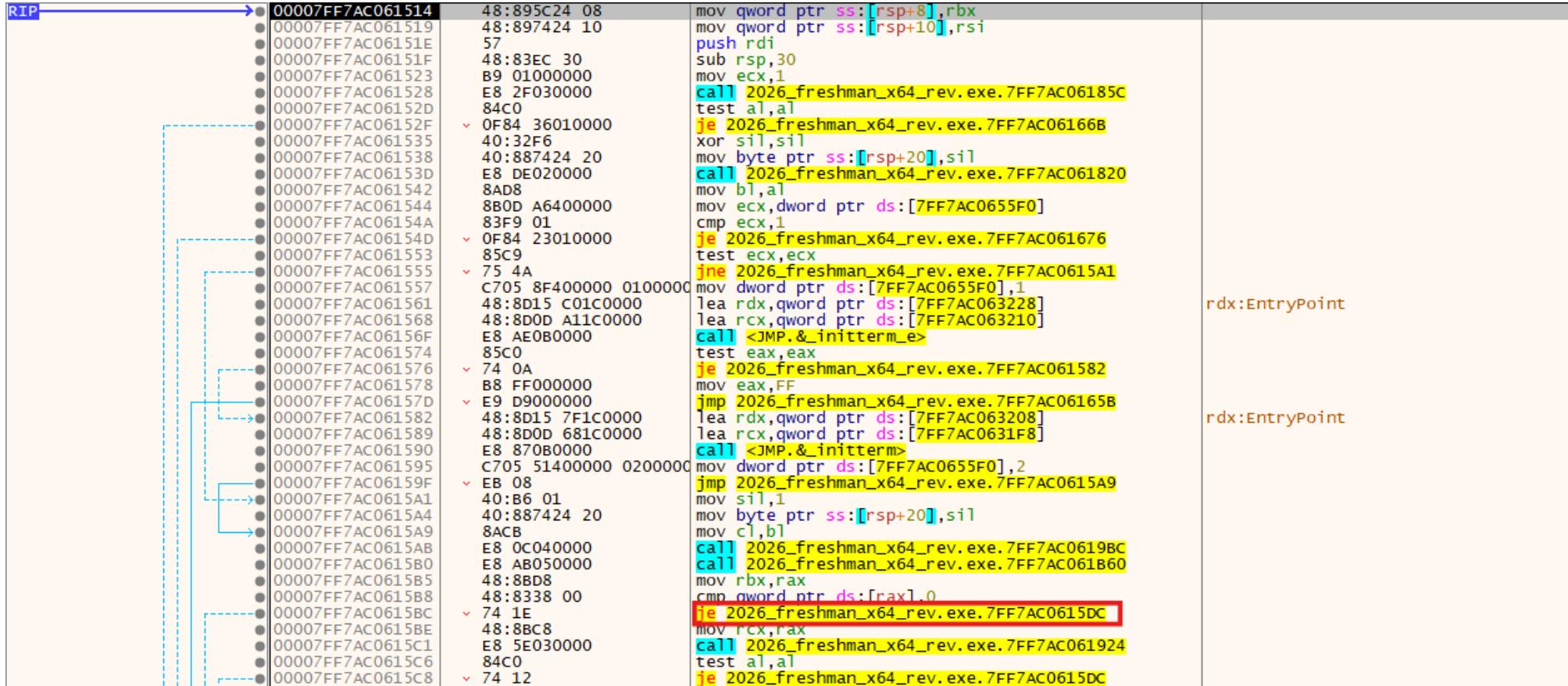
The screenshot shows a debugger interface with assembly code and memory dump panes. A yellow box highlights a **jmp** instruction at address 00007FF7AC061514. A blue dashed box highlights the assembly code for the main function, which starts with **call 2026_freshman_x64_rev.exe.7FF7AC061A60**. The assembly code includes various instructions like **sub rsp, 28**, **call qword ptr ds:[<SetUnhandledExceptionFilter>]**, and **jmp qword ptr ds:[<TerminateProcess>]**. The memory dump pane shows the corresponding memory addresses and values.

RIP	RAX	RDX	R9	Address	Value	Disassembly	Comments
00007FF7AC061690	48:83EC 28	E8 C7030000		sub rsp, 28		call 2026_freshman_x64_rev.exe.7FF7AC061A60	OptionalHeader.AddressOfEntryPoint
00007FF7AC061694	48:83C4 28			add rsp, 28			
00007FF7AC061699				jmp 2026_freshman_x64_rev.exe.7FF7AC061514			
00007FF7AC06169D	CC			int3			
00007FF7AC0616A2	CC			int3			
00007FF7AC0616A3	40:53			push rbx			
00007FF7AC0616A4	48:83EC 20			sub rsp, 20			
00007FF7AC0616A6	48:8BD9			mov rbx, rcx			
00007FF7AC0616AA	33C9			xor ecx, ecx			
00007FF7AC0616AD	FF15 6B190000			call qword ptr ds:[<SetUnhandledExceptionFilter>]			
00007FF7AC0616AF	48:8BCB			mov rcx, rbx			
00007FF7AC0616B5	FF15 5A190000			call qword ptr ds:[<UnhandledExceptionFilter>]			
00007FF7AC0616B8	FF15 64190000			call qword ptr ds:[<GetCurrentProcess>]			
00007FF7AC0616BE	48:8BC8			mov rcx, rax			
00007FF7AC0616C4	BA 090400C0			mov edx, C0000409			
00007FF7AC0616C7	48:83C4 20			add rsp, 20			
00007FF7AC0616CC	5B			pop rbx			
00007FF7AC0616D0				jmp qword ptr ds:[<TerminateProcess>]			JMP.&TerminateProcess
00007FF7AC0616D1	- 48:FF25 98190000			mov qword ptr ss:[rsp+8], rcx			
00007FF7AC0616D8	48:894C24 08			sub rsp, 38			
00007FF7AC0616DD	48:83EC 38			mov ecx, 17			
00007FF7AC0616E1	B9 17000000			call qword ptr ds:[<IsProcessorFeaturePresent>]			
00007FF7AC0616E6	FF15 7C190000			test eax, eax			
00007FF7AC0616EC	85C0			je 2026_freshman_x64_rev.exe.7FF7AC0616F7			
00007FF7AC0616EE	74 07			mov ecx, 2			
00007FF7AC0616F0	B9 02000000			int 29			
00007FF7AC0616F5	CD 29			lea rcx, qword ptr ds:[7FF7AC065120]			
00007FF7AC0616F7	48:8D0D 223A0000			call 2026_freshman_x64_rev.exe.7FF7AC0617AC			
00007FF7AC0616FE	E8 A9000000			mov rax, qword ptr ss:[rsp+38]			
00007FF7AC061703	48:8B4424 38			mov qword ptr ds:[7FF7AC065218], rax			
00007FF7AC061708	48:8905 093B0000			lea rax, qword ptr ss:[rsp+38]			
00007FF7AC06170F	48:8D4424 38			add rax, 8			
00007FF7AC061714	48:83C0 08			mov qword ptr ds:[7FF7AC0651B8], rax			
00007FF7AC061718	48:8905 993A0000			mov rax, qword ptr ds:[7FF7AC065218]			
00007FF7AC06171F	48:8B05 F23A0000			mov qword ptr ds:[7FF7AC065090], rax			
00007FF7AC061726	48:8905 63390000			mov rax, qword ptr ss:[rsn+40]			
00007FF7AC06172D	48:8B4424 40						

Problem Analysis and Solution

Core Problems

- [문제 3] 동적 분석을 통해 EP로부터 main함수를 찾아가는 과정을 서술하시오. (2점)
 - STEP 1-2) 'F8' Click, then **je** occurs.

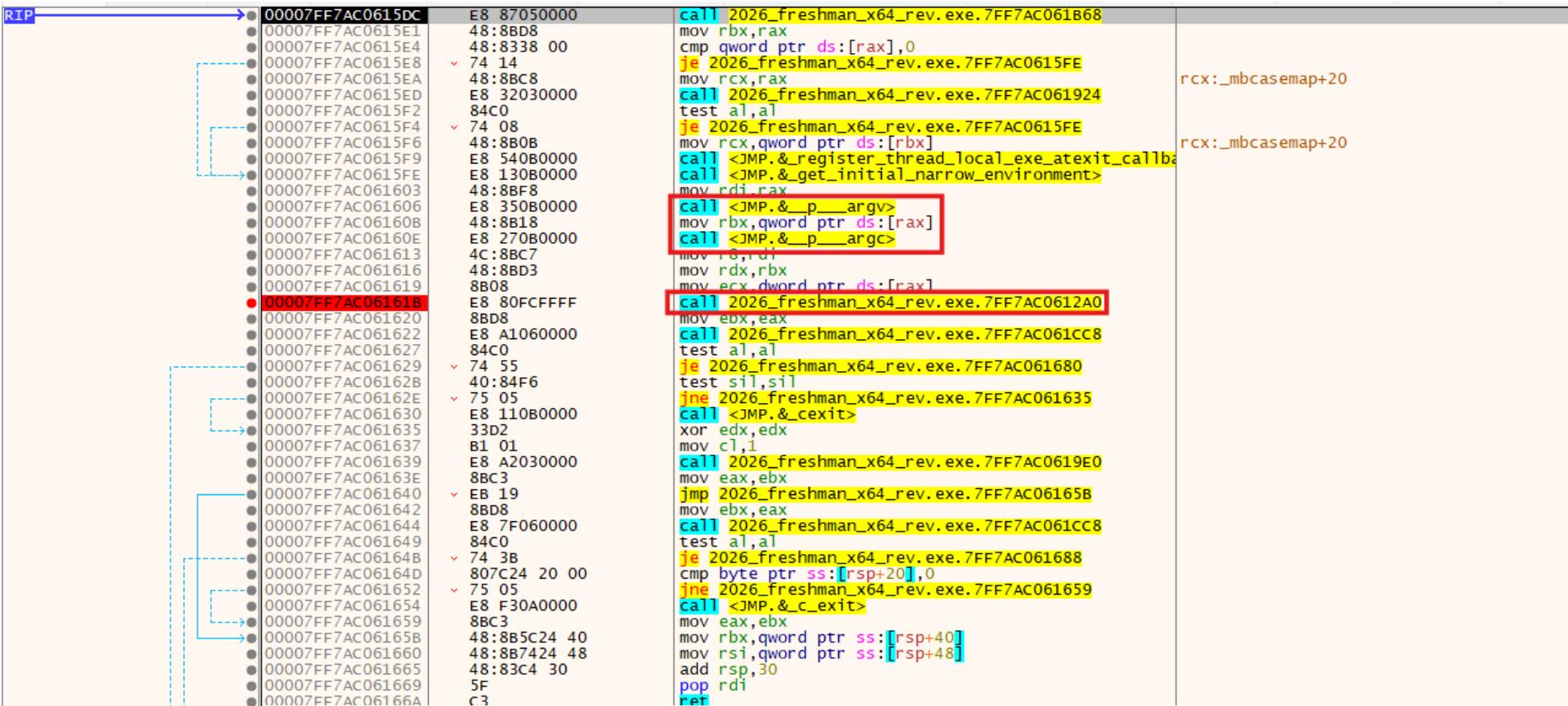


RIP	Address	Instruction
00007FF7AC061514	48:895C24 08	mov qword ptr ss:[rsp+8],rbx
00007FF7AC061519	48:897424 10	mov qword ptr ss:[rsp+10],rsi
00007FF7AC06151E	57	push rdi
00007FF7AC06151F	48:83EC 30	sub rsp,30
00007FF7AC061523	B9 01000000	mov ecx,1
00007FF7AC061528	E8 2F030000	call 2026_freshman_x64_rev.exe.7FF7AC06185C
00007FF7AC06152D	84C0	test al,al
00007FF7AC06152F	0F84 36010000	je 2026_freshman_x64_rev.exe.7FF7AC06166B
00007FF7AC061535	40:32F6	xor sil,sil
00007FF7AC061538	40:887424 20	mov byte ptr ss:[rsp+20],sil
00007FF7AC06153D	E8 DE020000	call 2026_freshman_x64_rev.exe.7FF7AC061820
00007FF7AC061542	8AD8	mov b1,a1
00007FF7AC061544	8B0D A6400000	mov ecx,dword ptr ds:[7FF7AC0655F0]
00007FF7AC06154A	83F9 01	cmp ecx,1
00007FF7AC06154D	0F84 23010000	je 2026_freshman_x64_rev.exe.7FF7AC061676
00007FF7AC061553	85C9	test ecx,ecx
00007FF7AC061555	75 4A	jne 2026_freshman_x64_rev.exe.7FF7AC0615A1
00007FF7AC061557	C705 8F400000 01000000	mov dword ptr ds:[7FF7AC0655F0],1
00007FF7AC061561	48:8D15 C01C0000	lea rdx,qword ptr ds:[7FF7AC063228]
00007FF7AC061568	48:8D0D A11C0000	lea rcx,qword ptr ds:[7FF7AC063210]
00007FF7AC06156F	E8 AE0B0000	call <JMP.&_initterm_e>
00007FF7AC061574	85C0	test eax,eax
00007FF7AC061576	74 0A	je 2026_freshman_x64_rev.exe.7FF7AC061582
00007FF7AC061578	B8 FF000000	mov eax,FF
00007FF7AC06157D	00007FF7AC061582	jmp 2026_freshman_x64_rev.exe.7FF7AC061658
00007FF7AC061589	48:8D15 7F1C0000	lea rdx,qword ptr ds:[7FF7AC063208]
00007FF7AC061590	48:8D0D 681C0000	lea rcx,qword ptr ds:[7FF7AC0631F8]
00007FF7AC061595	E8 870B0000	call <JMP.&_initterm>
00007FF7AC06159F	C705 51400000 02000000	mov dword ptr ds:[7FF7AC0655F0],2
00007FF7AC0615A1	00007FF7AC0615A4	jmp 2026_freshman_x64_rev.exe.7FF7AC0615A9
00007FF7AC0615A9	40:B6 01	mov sil,1
00007FF7AC0615AB	40:887424 20	mov byte ptr ss:[rsp+20],sil
00007FF7AC0615B0	8ACB	mov cl,b1
00007FF7AC0615B5	E8 0C040000	call 2026_freshman_x64_rev.exe.7FF7AC0619BC
00007FF7AC0615B8	E8 AB050000	call 2026_freshman_x64_rev.exe.7FF7AC061B60
00007FF7AC0615BC	48:8BD8	mov rbx,rax
00007FF7AC0615BE	48:8338 00	cmp qword ptr ds:[rax].0
00007FF7AC0615C1	74 1E	je 2026_freshman_x64_rev.exe.7FF7AC0615DC
00007FF7AC0615C6	48:8BC8	mov rcx,rax
00007FF7AC0615C8	E8 5E030000	call 2026_freshman_x64_rev.exe.7FF7AC061924
	84C0	test al,al
	74 12	je 2026_freshman_x64_rev.exe.7FF7AC0615DC

Problem Analysis and Solution

Core Problems

- [문제 3] 동적 분석을 통해 EP로부터 main함수를 찾아가는 과정을 서술하시오. (2점)
 - STEP 2) Then, we can find 'argv', 'argc', and call occurs. Let's keep clicking F8.



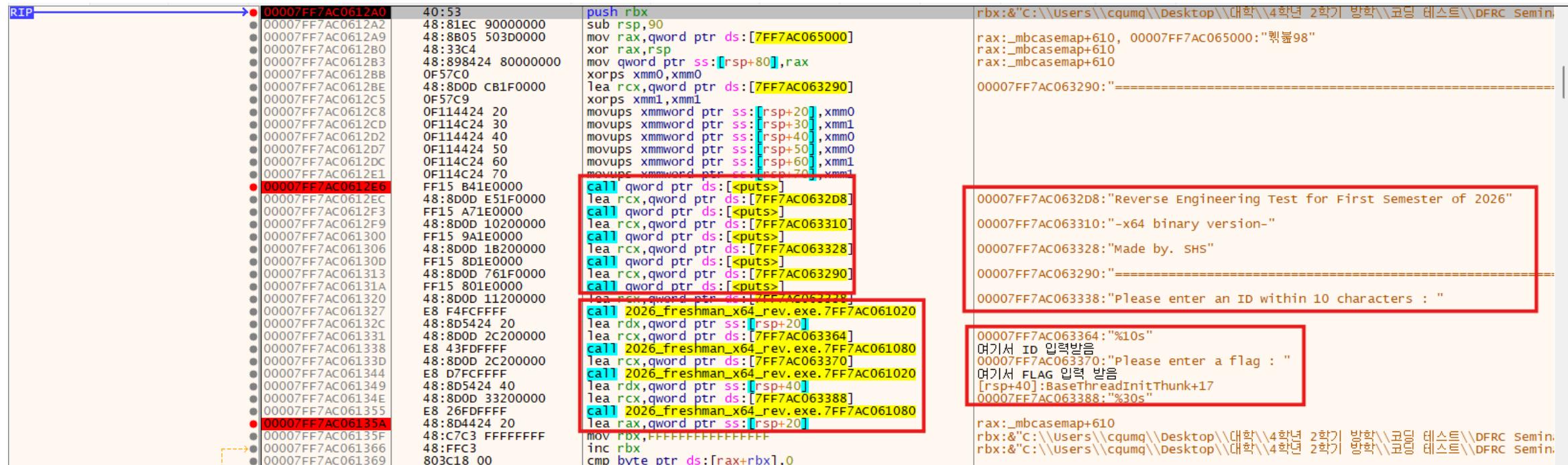
The screenshot shows a debugger interface with two panes. The left pane displays a memory dump with addresses from 00007FF7AC0615DC to 00007FF7AC06166A. The right pane shows the corresponding assembly code. A red box highlights the instruction `call 2026_freshman_x64_rev.exe.7FF7AC0612A0`, which is the entry point of the main function. Several blue dashed boxes highlight specific memory locations, likely argv and argc. The assembly code includes various system calls and local variable declarations.

Address	Value	Assembly
00007FF7AC06161B	E8 80FCFFF	call 2026_freshman_x64_rev.exe.7FF7AC0612A0
00007FF7AC061620	8BD8	mov rbx,rax
00007FF7AC061622	E8 A1060000	cmp qword ptr ds:[rax],0
00007FF7AC061627	84C0	je 2026_freshman_x64_rev.exe.7FF7AC0615FE
00007FF7AC061629	v 74 55	mov rcx,rax
00007FF7AC06162B	40:84F6	call 2026_freshman_x64_rev.exe.7FF7AC061924
00007FF7AC06162E	v 75 05	test al,al
00007FF7AC061630	E8 110B0000	je 2026_freshman_x64_rev.exe.7FF7AC0615FE
00007FF7AC061635	33D2	mov rdx,rdi
00007FF7AC061637	B1 01	mov rdx,rbx
00007FF7AC061639	E8 A2030000	mov ecx,dword ptr ds:[rax]
00007FF7AC06163E	88C3	call <JMP.&_p_argv>
00007FF7AC061640	v 74 19	mov rbx,qword ptr ds:[rax]
00007FF7AC061642	8BD8	call <JMP.&_p(argc>
00007FF7AC061644	E8 7F060000	mov rax,rdx
00007FF7AC061649	84C0	mov edx,rdx
00007FF7AC06164B	v 74 3B	mov eax,rdx
00007FF7AC06164D	807C24 20 00	call 2026_freshman_x64_rev.exe.7FF7AC061CC8
00007FF7AC061652	v 75 05	test al,al
00007FF7AC061654	E8 F30A0000	je 2026_freshman_x64_rev.exe.7FF7AC061688
00007FF7AC061659	88C3	cmp byte ptr ss:[rsp+20],0
00007FF7AC06165B	48:8B5C24 40	ine 2026_freshman_x64_rev.exe.7FF7AC061635
00007FF7AC061660	48:8B7424 48	call <JMP.&_cexit>
00007FF7AC061665	48:83C4 30	xor edx,edx
00007FF7AC061669	5F	mov cl,1
00007FF7AC06166A	C3	call 2026_freshman_x64_rev.exe.7FF7AC0619E0

Problem Analysis and Solution

Core Problems

- [문제 3] 동적 분석을 통해 EP로부터 main함수를 찾아가는 과정을 서술하시오. (2점)
 - STEP 3) Then we can find **console message**. Therefore, this part is **main function**.

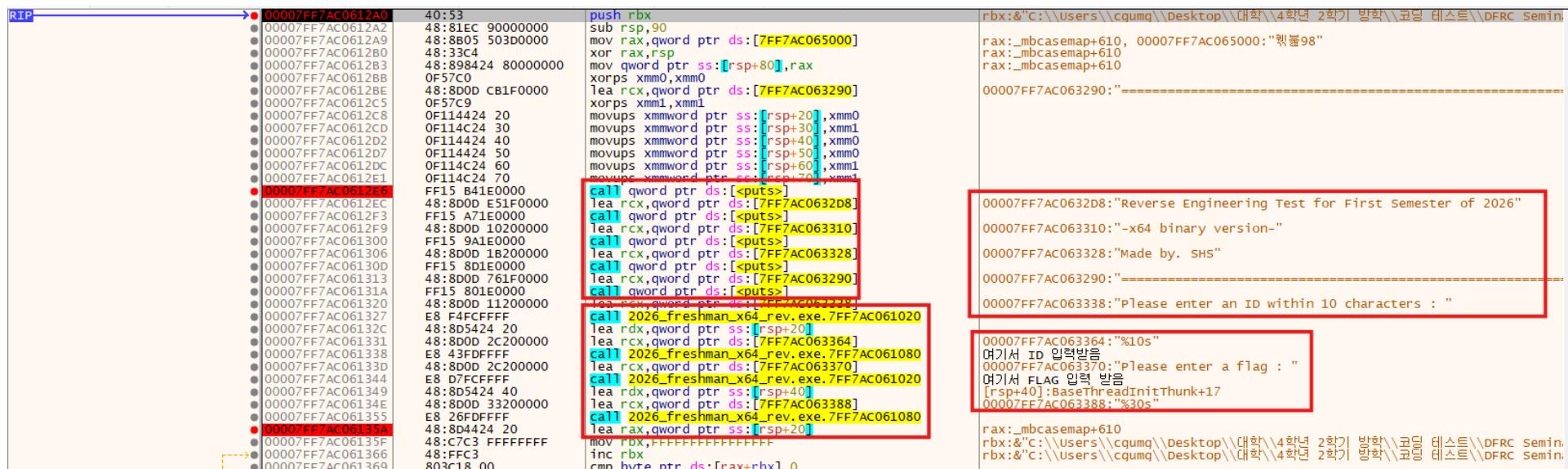


RIP → 00007FF7AC0612A0 40:53 push rbx
sub rsp,90
mov rax,qword ptr ds:[7FF7AC065000]
xor rax,rspxor
mov qword ptr ss:[rsp+80],rax
xors xmm0,xmm0
lea rcx,qword ptr ds:[7FF7AC063290]
xors xmm1,xmm1
movups xmmword ptr ss:[rsp+20],xmm0
movups xmmword ptr ss:[rsp+30],xmm1
movups xmmword ptr ss:[rsp+40],xmm0
movups xmmword ptr ss:[rsp+50],xmm0
movups xmmword ptr ss:[rsp+60],xmm1
movups xmmword ptr ss:[rsp+70],xmm1
call qword ptr ds:[<puts>]
lea rcx,qword ptr ds:[7FF7AC0632D8]
call qword ptr ds:[<puts>]
lea rcx,qword ptr ds:[7FF7AC063310]
call qword ptr ds:[<puts>]
lea rcx,qword ptr ds:[7FF7AC063328]
call qword ptr ds:[<puts>]
lea rcx,qword ptr ds:[7FF7AC063290]
call qword ptr ds:[<puts>]
lea rcx,qword ptr ds:[7FF7AC063338]
call 2026_freshman_x64_rev.exe.7FF7AC061020
lea rdx,qword ptr ss:[rsp+20]
lea rcx,qword ptr ds:[7FF7AC063364]
call 2026_freshman_x64_rev.exe.7FF7AC061080
lea rcx,qword ptr ds:[7FF7AC063370]
call 2026_freshman_x64_rev.exe.7FF7AC061020
lea rdx,qword ptr ss:[rsp+40]
lea rcx,qword ptr ds:[7FF7AC063388]
call 2026_freshman_x64_rev.exe.7FF7AC061080
lea rax,qword ptr ss:[rsp+20]
mov rbx,FFFFFFFFFFFFFFF
inc rbx
cmp byte ptr ds:[rax+rbx],0
rbx:&"C:\\\\Users\\\\cqumg\\\\Desktop\\\\대학\\\\4학년 2학기 방학\\\\코딩 테스트\\\\DFRC Seminar\\\\reverse98"
rax:_mbcasemap+610, 00007FF7AC065000:"_reverse98"
rax:_mbcasemap+610
rax:_mbcasemap+610
00007FF7AC063290:"=====
00007FF7AC0632D8:"Reverse Engineering Test for First Semester of 2026"
00007FF7AC063310:"-x64 binary version-"
00007FF7AC063328:"Made by. SHS"
00007FF7AC063290:"=====
00007FF7AC063338:"Please enter an ID within 10 characters : "
00007FF7AC063364:"%10s"
여기서 ID 입력받음
00007FF7AC063370:"Please enter a flag : "
여기서 FLAG 입력 받음
[rsp+40]:BaseThreadInitThunk+17
00007FF7AC063388:"%30s"
rax:_mbcasemap+610
rbx:&"C:\\\\Users\\\\cqumg\\\\Desktop\\\\대학\\\\4학년 2학기 방학\\\\코딩 테스트\\\\DFRC Seminar\\\\reverse98"
rbx:&"C:\\\\Users\\\\cqumg\\\\Desktop\\\\대학\\\\4학년 2학기 방학\\\\코딩 테스트\\\\DFRC Seminar\\\\reverse98"

Problem Analysis and Solution

Core Problems

- [문제 4] main함수의 전체 코드 흐름에 대해 본인이 파악한 대로 **추론**하여 서술하시오.
(동적 및 정적 분석 중 본인이 편한 방법을 사용하되, 두 방법 모두 사용해도 무관함) (3점)
 - STEP 1) Through **dynamic analysis**, it can be observed that the program **prints a message** using the puts function and **receives input** from the user. However, the subsequent execution flow was **too complex** to fully trace using dynamic analysis alone.



The screenshot shows a debugger interface with assembly code on the left and memory dump on the right. The assembly code is annotated with several red boxes highlighting specific sections of the code, likely corresponding to the points of interest mentioned in the problem statement. The memory dump shows the actual strings being printed and the input being read.

Key annotations in the assembly code:

- A red box highlights the first few lines of the main function, including the initial stack setup and the first puts call.
- A red box highlights a sequence of puts calls that print the program's name and version information.
- A red box highlights another sequence of puts calls that ask the user to enter an ID.
- A red box highlights the final puts call that prints the placeholder for the user input.

Key annotations in the memory dump:

- A red box highlights the printed strings: "_reverse engineering test for first semester of 2026", "-x64 binary version-", "Made by. SHS", and "Please enter an ID within 10 characters :".
- A red box highlights the user input area where the string "%10s" is shown.
- A red box highlights the final printed string: "Please enter a flag :".

Problem Analysis and Solution

Core Problems

- [문제 4] main함수의 전체 코드 흐름에 대해 본인이 파악한 대로 **추론**하여 서술하시오.
(동적 및 정적 분석 중 본인이 편한 방법을 사용하되, 두 방법 모두 사용해도 무관함) (3점)
 - STEP 2) Through static analysis, it can be identified that in **FUN_1400012a0**, messages are **printed via puts**, and **user input is received** through **FUN_140001020** and **FUN_140001080**.

```
void FUN_1400012a0(undefined8 param_1,undefined8 param_2,undefined8 param_3,undefined8 param_4)

puts("=====");
puts("Reverse Engineering Test for First Semester of 2026");
puts("-x64 binary version-");
puts("Made by. SHS");
puts("=====");
FUN_140001020("Please enter an ID within 10 characters : ",param_2,param_3,param_4); →
pcVar3 = local_78;
FUN_140001080(&DAT_140003364,pcVar3,param_3,param_4);
FUN_140001020("Please enter a flag : ",pcVar3,param_3,param_4);
FUN_140001080(&DAT_140003388,local_58,param_3,param_4);

local_res10 = param_2;
local_res18 = param_3;
local_res20 = param_4;
uVar1 = _acrt_iob_func(1);
puVar2 = (undefined8 *)FUN_140001000();
__stdio_common_vfprintf(*puVar2,uVar1,param_1,0,&local_res10);
return;
```



```
local_res10 = param_2;
local_res18 = param_3;
local_res20 = param_4;
uVar1 = _acrt_iob_func(0);
puVar2 = (undefined8 *)FUN_140001010();
__stdio_common_vfscanf(*puVar2,uVar1,param_1,0,&local_res10);
return;
```

Problem Analysis and Solution

Core Problems

- [문제 4] main함수의 전체 코드 흐름에 대해 본인이 파악한 대로 **추론**하여 서술하시오.
(동적 및 정적 분석 중 본인이 편한 방법을 사용하되, 두 방법 모두 사용해도 무관함) (3점)
 - STEP 3) After that, **FUN_140001110** is called, indicating that it performs a certain function.
 - Since it appears to **modify param_1**, it is presumed to **manipulate the input value**.

```
uVar6 = 0xffffffffffffffffff;
do {
    uVar5 = uVar6 + 1;
    lVar1 = uVar6 + 1;
    uVar6 = uVar5;
} while (local_78[lVar1] != '\0');
lVar1 = 0;
do {
    pcVar3 = local_78 + lVar1;
    *(char *)((longlong)&local_68 + lVar1) = *pcVar3;
    lVar1 = lVar1 + 1;
} while (*pcVar3 != '\0');
iVar4 = (int)uVar5;
FUN_140001110((undefined *)&local_68,iVar4);
```



```
void FUN_140001110(undefined1 *param_1,int param_2)

{
    undefined1 uVar1;
    undefined1 *puVar2;
    longlong lVar3;

    puts("[System] Welcome to DFRC!");
    lVar3 = (longlong)(param_2 / 2);
    if (0 < param_2 / 2) {
        puVar2 = param_1 + (longlong)param_2 + -1;
        do {
            uVar1 = *param_1;
            *param_1 = *puVar2;
            param_1 = param_1 + 1;
            *puVar2 = uVar1;
            lVar3 = lVar3 + -1;
            puVar2 = puVar2 + -1;
        } while (lVar3 != 0);
    }
    return;
}
```

Problem Analysis and Solution

Core Problems

- [문제 4] main함수의 전체 코드 흐름에 대해 본인이 파악한 대로 **추론**하여 서술하시오.
(동적 및 정적 분석 중 본인이 편한 방법을 사용하되, 두 방법 모두 사용해도 무관함) (3점)
 - STEP 4) After that, **FUN_140001180** is called, indicating that it performs a certain function.
 - Since it appears to **XOR arbitrary bytes**, it is inferred that this function is related to **key generation**.

FUN_140001180 (&local_68, iVar4, &local_38);



```
void FUN_140001180(void *param_1,int param_2,void *param_3)

{
    byte bVar1;
    size_t _Size;

    bVar1 = FUN_1400010e0();
    _Size = (_size_t)param_2;
    bVar1 = bVar1 ^ (byte)param_2;
    memcpy(param_3,param_1,_Size);
    *(byte *)(_Size + 2 + (longlong)param_3) = bVar1 ^ 0x78;
    *(byte *)(_Size + (longlong)param_3) = bVar1 ^ 0x74;
    *(byte *)(_Size + 3 + (longlong)param_3) = bVar1 ^ 0x67;
    *(byte *)(_Size + 1 + (longlong)param_3) = bVar1 ^ 0x35;
    *(byte *)(_Size + 4 + (longlong)param_3) = bVar1 ^ 0x54;
    *(byte *)(_Size + 5 + (longlong)param_3) = bVar1 ^ 0x75;
    *(byte *)(_Size + 6 + (longlong)param_3) = bVar1 ^ 0x4b;
    *(byte *)(_Size + 7 + (longlong)param_3) = bVar1 ^ 0x70;
    *(byte *)(_Size + 8 + (longlong)param_3) = bVar1 ^ 0x69;
    return;
}
```

Problem Analysis and Solution

Core Problems

- [문제 4] main함수의 전체 코드 흐름에 대해 본인이 파악한 대로 **추론**하여 서술하시오.
(동적 및 정적 분석 중 본인이 편한 방법을 사용하되, 두 방법 모두 사용해도 무관함) (3점)
 - STEP 5) After that, **FUN_140001220** is called, indicating that it performs a certain function.
 - Since **strcmp** is used, the input value is compared with the correct **answer**.
 - Afterward, the function returns whether the input is correct through **additional validation**.

uVar2 = **FUN_140001220 (local_58, (char *) &local_38, iVar4);**



```
undefined8 FUN_140001220(char *param_1,char *param_2,int param_3)

{
    byte bVar1;
    int iVar2;
    undefined8 uVar3;
    char *_Str1;
    char *_Str2;

    _Str1 = param_1;
    _Str2 = param_2;
    bVar1 = FUN_1400010e0();
    iVar2 = strcmp(_Str1,_Str2,(longlong)param_3);
    if (iVar2 == 0) {
        iVar2 = 0;
        do {
            if ((byte)(param_1[iVar2 + param_3] + 2U ^ bVar1 ^ (byte)param_3) != param_2[iVar2 + param_3])
                goto LAB_140001282;
            iVar2 = iVar2 + 1;
        } while (iVar2 < 9);
        uVar3 = 1;
    }
    else {
LAB_140001282:
        uVar3 = 0;
    }
    return uVar3;
}
```

Problem Analysis and Solution

Core Problems

- [문제 4] main함수의 전체 코드 흐름에 대해 본인이 파악한 대로 **추론**하여 서술하시오.
(동적 및 정적 분석 중 본인이 편한 방법을 사용하되, 두 방법 모두 사용해도 무관함) (3점)
 - STEP 6) After that, depending on whether the input is correct, a success or failure message is displayed, and the program then terminates

```
pcVar3 = "[System] Congratulation! You have answered correctly!";
if ((int)uVar2 == 0) {
    pcVar3 = "[System] Incorrect! Please try again....:";
}
puts(pcVar3);
FUN_140001080(&DAT_1400033f4,local_58,uVar5,param_4);
FUN_140001410(local_18 ^ (ulonglong)auStack_98);
return;
```

Problem Analysis and Solution

Core Problems

- [문제 4] main함수의 전체 코드 흐름에 대해 본인이 파악한 대로 **추론**하여 서술하시오.
(동적 및 정적 분석 중 본인이 편한 방법을 사용하되, 두 방법 모두 사용해도 무관함) (3점)

- Summary of the **overall code flow** of the **main function**

STEP 2) Calls **FUN_140001020** and **FUN_140001080** to execute **puts**, **printf**, and **scanf**

STEP 3) Calls **FUN_140001110** to **manipulate the input value**

STEP 4) Calls **FUN_140001180** to perform **key generation**

STEP 5) Calls **FUN_140001220** to **compare** the input value with the correct answer

STEP 6) **Outputs a success or failure message** depending on whether the answer is correct and **terminates** the program

Problem Analysis and Solution

Core Problems

- [문제 5] 아래의 문자열을 찾고, 해당 문자열이 위치한 **함수**의 동작 원리 및 역할에 대해 자세히 서술하시오. (4점)

- Welcome to DFRC!
- Based on Ghidra, it can be found in **FUN_140001110**

```
void FUN_140001110(undefined1 *param_1,int param_2)

{
    undefined1 uVar1;
    undefined1 *puVar2;
    longlong lVar3;

    puts "[System] Welcome to DFRC!";
    lVar3 = (longlong)(param_2 / 2);
    if (0 < param_2 / 2) {
        puVar2 = param_1 + (longlong)param_2 + -1;
        do {
            uVar1 = *param_1;
            *param_1 = *puVar2;
            param_1 = param_1 + 1;
            *puVar2 = uVar1;
            lVar3 = lVar3 + -1;
            puVar2 = puVar2 + -1;
        } while (lVar3 != 0);
    }
    return;
}
```

Problem Analysis and Solution

Core Problems

- Operation mechanism and role

- **local_68** receives the value after the ID is stored in **local_78** and passed through **pcVar3**.

That is, **local_68** represents the **ID** and corresponds to **param_1**.

- **iVar4** receives the value after **uVar5** stores the length of the ID.

That is, **iVar4** represents the ID size and corresponds to **param_2**.

```
FUN_140001020("Please enter an ID within 10 characters : ",param_2,param_3,param_4);
pcVar3 = local_78;
FUN_140001080(&DAT_140003364,pcVar3,param_3,param_4);
FUN_140001020("Please enter a flag : ",pcVar3,param_3,param_4);
FUN_140001080(&DAT_140003388,local_58,param_3,param_4);
uVar6 = 0xffffffffffffffffffff;
do {
    uVar5 = uVar6 + 1;
    lVar1 = uVar6 + 1;
    uVar6 = uVar5;
} while (local_78[lVar1] != '\0');
lVar1 = 0;
do {
    pcVar3 = local_78 + lVar1;
    *(char *)((longlong)&local_68 + lVar1) = *pcVar3;
    lVar1 = lVar1 + 1;
} while (*pcVar3 != '\0');
iVar4 = (int)uVar5;
FUN_140001110(undefined1 *)&local_68[iVar4];
```

Problem Analysis and Solution

Core Problems

■ Operation mechanism and role

- **puVar2** points to the end of the ID array.
- **uVar1** stores a backup of **param_1**.
- **param_1** is assigned the value of **puVar2**.
 - That is, by assigning **param_1** to the end of the ID array, the value of **param_1** is being reversed.
 - Therefore, this function can be identified as an **id_reverse function**.

```
void FUN_140001110(undefined1 *param_1,int param_2)
{
    undefined1 uVar1;
    undefined1 *puVar2;
    longlong lVar3;

    puts("[System] Welcome to DFRC!");
    lVar3 = (longlong)(param_2 / 2);
    if (0 < param_2 / 2) {
        puVar2 = param_1 + (longlong)param_2 + -1;
        do {
            uVar1 = *param_1;
            *param_1 = *puVar2;
            param_1 = param_1 + 1;
            *puVar2 = uVar1;
            lVar3 = lVar3 + -1;
            puVar2 = puVar2 + -1;
        } while (lVar3 != 0);
    }
    return;
}
```

Problem Analysis and Solution

Core Problems

- [문제 6] 프로그램의 Key 생성 방식에 대해 서술하시오. (4점)

```
void FUN_140001180(void *param_1,int param_2,void *param_3)

{
    byte bVar1;
    size_t _Size;

    bVar1 = FUN_1400010e0();
    _Size = (size_t)param_2;
    bVar1 = bVar1 ^ (byte)param_2;
    memcpy(param_3,param_1,_Size);
    *(byte *)(_Size + 2 + (longlong)param_3) = bVar1 ^ 0x78;
    *(byte *)(_Size + (longlong)param_3) = bVar1 ^ 0x74;
    *(byte *)(_Size + 3 + (longlong)param_3) = bVar1 ^ 0x67;
    *(byte *)(_Size + 1 + (longlong)param_3) = bVar1 ^ 0x35;
    *(byte *)(_Size + 4 + (longlong)param_3) = bVar1 ^ 0x54;
    *(byte *)(_Size + 5 + (longlong)param_3) = bVar1 ^ 0x75;
    *(byte *)(_Size + 6 + (longlong)param_3) = bVar1 ^ 0x4b;
    *(byte *)(_Size + 7 + (longlong)param_3) = bVar1 ^ 0x70;
    *(byte *)(_Size + 8 + (longlong)param_3) = bVar1 ^ 0x69;
    return;
}
```

Problem Analysis and Solution

Core Problems

■ [문제 6] 프로그램의 Key 생성 방식에 대해 서술하시오. (4점)

- local_68 holds the ID in its **reversed state**. (**param_1**)
- iVar4 represents the **length of the ID**. (**param_2**)
- local_38 is a buffer used to store the **result of the function execution**. (**param_3**)

```
FUN_140001180(&local_68,iVar4,&local_38);
```



```
void FUN_140001180(void *param_1,int param_2,void *param_3)

{
    byte bVar1;
    size_t _Size;

    bVar1 = FUN_1400010e0();
    _Size = (size_t)param_2;
    bVar1 = bVar1 ^ (byte)param_2;
    memcpy(param_3,param_1,_Size);
    *(byte *)(_Size + 2 + (longlong)param_3) = bVar1 ^ 0x78;
    *(byte *)(_Size + (longlong)param_3) = bVar1 ^ 0x74;
    *(byte *)(_Size + 3 + (longlong)param_3) = bVar1 ^ 0x67;
    *(byte *)(_Size + 1 + (longlong)param_3) = bVar1 ^ 0x35;
    *(byte *)(_Size + 4 + (longlong)param_3) = bVar1 ^ 0x54;
    *(byte *)(_Size + 5 + (longlong)param_3) = bVar1 ^ 0x75;
    *(byte *)(_Size + 6 + (longlong)param_3) = bVar1 ^ 0x4b;
    *(byte *)(_Size + 7 + (longlong)param_3) = bVar1 ^ 0x70;
    *(byte *)(_Size + 8 + (longlong)param_3) = bVar1 ^ 0x69;
    return;
}
```

Problem Analysis and Solution

Core Problems

■ [문제 6] 프로그램의 Key 생성 방식에 대해 서술하시오. (4점)

- First, the function **FUN_1400010e0** is called and its return value is stored in **bVar1**.
- The function **FUN_1400010e0** always **returns 0xF9**.

```
void FUN_140001180(void *param_1,int param_2,void *param_3)

{
    byte bVar1;
    size_t _Size;

    bVar1 = FUN_1400010e0();
    _Size = (size_t)param_2;
    bVar1 = bVar1 ^ (byte)param_2;
    memcpy(param_3,param_1,_Size);
    *(byte *)(_Size + 2 + (longlong)param_3) = bVar1 ^ 0x78;
    *(byte *)(_Size + (longlong)param_3) = bVar1 ^ 0x74;
    *(byte *)(_Size + 3 + (longlong)param_3) = bVar1 ^ 0x67;
    *(byte *)(_Size + 1 + (longlong)param_3) = bVar1 ^ 0x35;
    *(byte *)(_Size + 4 + (longlong)param_3) = bVar1 ^ 0x54;
    *(byte *)(_Size + 5 + (longlong)param_3) = bVar1 ^ 0x75;
    *(byte *)(_Size + 6 + (longlong)param_3) = bVar1 ^ 0x4b;
    *(byte *)(_Size + 7 + (longlong)param_3) = bVar1 ^ 0x70;
    *(byte *)(_Size + 8 + (longlong)param_3) = bVar1 ^ 0x69;
    return;
}
```



```
undefined1 FUN_1400010e0(void)

{
    if (DAT_14000565c == 0) {
        DAT_140005668 = 0xf9;
        DAT_14000565c = 1;
        return 0xf9;
    }
    return DAT_140005668;
}
```

Problem Analysis and Solution

Core Problems

■ [문제 6] 프로그램의 Key 생성 방식에 대해 서술하시오. (4점)

- `_Size` represents the length of the ID, and `bVar1` is obtained by XOR-ing `0xF9` with the ID length.
- After that, the reversed ID is copied into `param_3`.
- Then, `bVar1` is XOR-ed with a specific fixed byte value and stored in `param_3`.
- That is, `param_3[0 ~ _Size-1]` contains the reversed ID value.
- `param_3[_Size ~ _Size+8]` contains the result of `0xF9 ^ ID length ^ {0x74, 0x35, 0x78, 0x67, 0x54, 0x75, 0x4B, 0x70, 0x69}`.
- Ultimately, `local_38 (param_3)` consists of `reversed_ID | (0xF9 ^ ID length ^ fixed bytes)`.

```
void FUN_140001180(void *param_1,int param_2,void *param_3){  
    byte bVar1;  
    size_t _Size;  
  
    bVar1 = FUN_1400010e0();  
    _Size = (size_t)param_2;  
    bVar1 = bVar1 ^ (byte)param_2;  
    memcpy(param_3,param_1,_Size);  
    *(byte *)(_Size + 2 + (longlong)param_3) = bVar1 ^ 0x78;  
    *(byte *)(_Size + (longlong)param_3) = bVar1 ^ 0x74;  
    *(byte *)(_Size + 3 + (longlong)param_3) = bVar1 ^ 0x67;  
    *(byte *)(_Size + 1 + (longlong)param_3) = bVar1 ^ 0x35;  
    *(byte *)(_Size + 4 + (longlong)param_3) = bVar1 ^ 0x54;  
    *(byte *)(_Size + 5 + (longlong)param_3) = bVar1 ^ 0x75;  
    *(byte *)(_Size + 6 + (longlong)param_3) = bVar1 ^ 0x4b;  
    *(byte *)(_Size + 7 + (longlong)param_3) = bVar1 ^ 0x70;  
    *(byte *)(_Size + 8 + (longlong)param_3) = bVar1 ^ 0x69;  
}  
return;
```

Problem Analysis and Solution

Advanced Problems

- [문제 7] 현재까지 해결했던 문제들을 바탕으로, 제공된 프로그램의 정답을 출력하시오.
(프로그램의 ID, Flag, 성공 메시지 모두 나오도록 **캡쳐**하여 답안을 작성) (4점)

```
uVar2 = FUN_140001220(local_58,(char *)&local_38,iVar4);
```



```
undefined8 FUN_140001220(char *param_1,char *param_2,int param_3)

{
    byte bVar1;
    int iVar2;
    undefined8 uVar3;
    char *_Str1;
    char *_Str2;

    _Str1 = param_1;
    _Str2 = param_2;
    bVar1 = FUN_1400010e0();
    iVar2 = strcmp(_Str1,_Str2,(longlong)param_3);
    if (iVar2 == 0) {
        iVar2 = 0;
        do {
            if ((byte)(param_1[iVar2 + param_3] + 2U ^ bVar1 ^ (byte)param_3) != param_2[iVar2 + param_3])
                goto LAB_140001282;
            iVar2 = iVar2 + 1;
        } while (iVar2 < 9);
        uVar3 = 1;
    }
    else {
LAB_140001282:
        uVar3 = 0;
    }
    return uVar3;
}
```

Problem Analysis and Solution

Advanced Problems

- [문제 7] 현재까지 해결했던 문제들을 바탕으로, 제공된 프로그램의 정답을 출력하시오.
(프로그램의 ID, Flag, 성공 메시지 모두 나오도록 캡쳐하여 답안을 작성) (4점)

- local_58 is input flag. (param_1)
- local_38은 (reverse_ID | 0xF9 ^ ID_length ^ fixed_bytes) (param_2)
- iVar4는 ID length (param_3)

```
FUN_140001080(&DAT_140003364,pcVar3,param_3,param_4);
FUN_140001020 "Please enter a flag : ".pcVar3,param_3,param_4;
FUN_140001080(&DAT_140003388,local_58,param_3,param_4);
uVar6 = 0xffffffffffffffff;
do {
    uVar5 = uVar6 + 1;
    lVar1 = uVar6 + 1;
    uVar6 = uVar5;
} while (local_78[lVar1] != '\0');
lVar1 = 0;
do {
    pcVar3 = local_78 + lVar1;
    *(char *)((longlong)&local_68 + lVar1) = *pcVar3;
    lVar1 = lVar1 + 1;
} while (*pcVar3 != '\0');
iVar4 = (int)uVar5;
FUN_140001110((undefined1 *)&local_68,iVar4);
FUN_140001180(&local_68,iVar4,&local_38);
uVar5 = uVar5 & 0xffffffff;
uVar2 = FUN_140001220 local_58,(char *)&local_38,iVar4);
```

Problem Analysis and Solution

Advanced Problems

- [문제 7] 현재까지 해결했던 문제들을 바탕으로, 제공된 프로그램의 정답을 출력하시오.
(프로그램의 ID, Flag, 성공 메시지 모두 나오도록 캡쳐하여 답안을 작성) (4점)

- `_Str1` contains **local_58** (input flag).
- `_Str2` contains **local_38** (`reversed_ID | 0xF9 ^ ID_length ^ fixed_bytes`).
- **bVar1** stores the value **0xF9**.
- Using **strncmp**, the flag is compared with `reversed_ID` for the length of the ID.
- Therefore, the **prefix of the flag must match the reversed ID**.

```
undefined8 FUN_140001220(char *param_1,char *param_2,int param_3)

{
    byte bVar1;
    int iVar2;
    undefined8 uVar3;
    char *_Str1;
    char *_Str2;

    _Str1 = param_1;
    _Str2 = param_2;
    bVar1 = FUN_1400010e0();
    iVar2 = strncmp(_Str1,_Str2,(longlong)param_3);
    if (iVar2 == 0) {
        iVar2 = 0;
        do {
            if ((byte)(param_1[iVar2 + param_3] + 2U ^ bVar1 ^ (byte)param_3) != param_2[iVar2 + param_3])
                goto LAB_140001282;
            iVar2 = iVar2 + 1;
        } while (iVar2 < 9);
        uVar3 = 1;
    }
    else {
LAB_140001282:
        uVar3 = 0;
    }
    return uVar3;
}
```

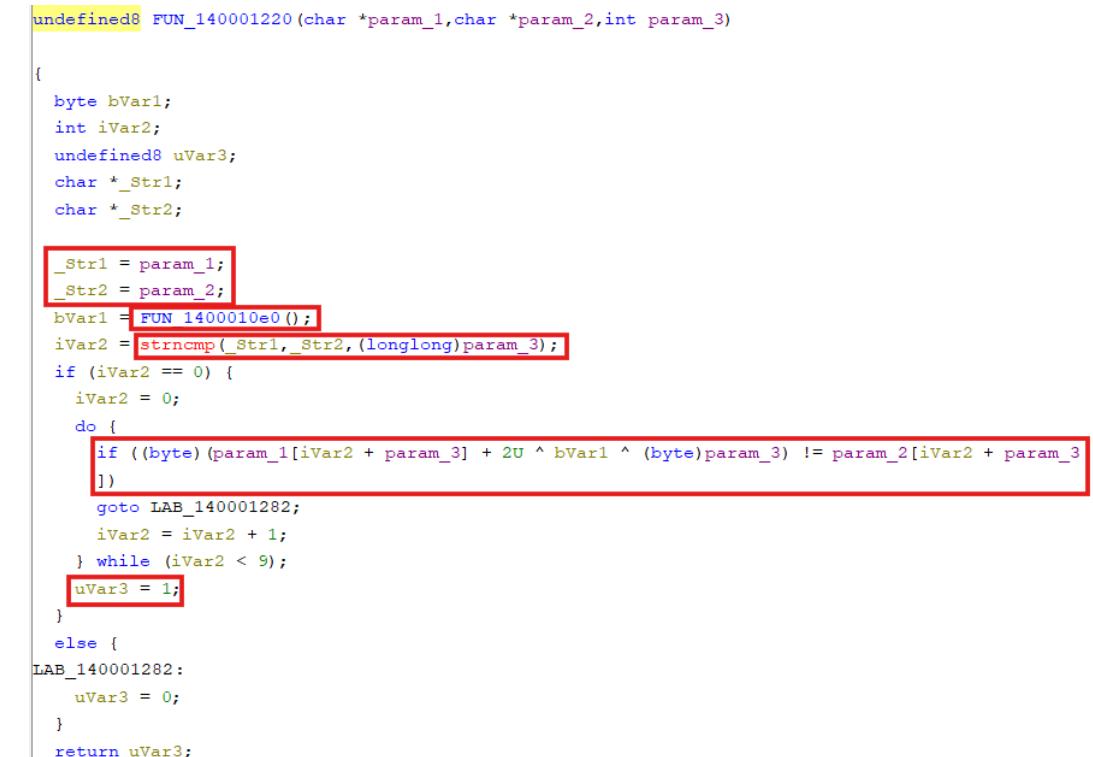
Problem Analysis and Solution

Advanced Problems

- [문제 7] 현재까지 해결했던 문제들을 바탕으로, 제공된 프로그램의 정답을 출력하시오.
(프로그램의 ID, Flag, 성공 메시지 모두 나오도록 캡쳐하여 답안을 작성) (4점)

- After that, **comparison of the remaining part** begins using `param_1[iVar2 + param_3]`.
- The condition can be interpreted as follows:
 - **Left-hand side:** $(\text{FLAG}[iVar2 + param_3] + 2) \wedge 0xF9 \wedge param_3$
 - **Right-hand side:** $0xF9 \wedge param_3 \wedge \text{fixed bytes}$
- Thus, **FLAG[iVar2 + param_3] = fixed_byte - 2**.
- $(0x74, 0x35, 0x78, 0x67, 0x54, 0x75, 0x4B, 0x70, 0x69) - 2$
 $= t5xgTuKpi - 2$
 $= r3veRsInG$

Therefore, the suffix of the flag is **r3veRsInG**.



```
undefined8 FUN_140001220(char *param_1,char *param_2,int param_3)
{
    byte bVar1;
    int iVar2;
    undefined8 uVar3;
    char *_Str1;
    char *_Str2;

    _Str1 = param_1;
    _Str2 = param_2;
    bVar1 = FUN_1400010e0();
    iVar2 = strncmp(_Str1, _Str2, (longlong)param_3);
    if (iVar2 == 0) {
        iVar2 = 0;
        do {
            if ((byte)(param_1[iVar2 + param_3] + 2U ^ bVar1 ^ (byte)param_3) != param_2[iVar2 + param_3])
                goto LAB_140001282;
            iVar2 = iVar2 + 1;
        } while (iVar2 < 9);
        uVar3 = 1;
    }
    else {
        LAB_140001282:
        uVar3 = 0;
    }
    return uVar3;
}
```

Problem Analysis and Solution

Advanced Problems

- [문제 7] 현재까지 해결했던 문제들을 바탕으로, 제공된 프로그램의 정답을 출력하시오.
(프로그램의 ID, Flag, 성공 메시지 모두 나오도록 **캡쳐**하여 답안을 작성) (4점)
 - Therefore, composition of the flag is 'reverse_ID | r3veRsInG'

```
=====
Reverse Engineering Test for First Semester of 2026
-x64 binary version-
Made by. SHS
=====
Please enter an ID within 10 characters : minhyuk
Please enter a flag : kuyhnimr3veRsIng
[System] Welcome to DFRC!
[System] Congratulation! You have answered correctly!
```

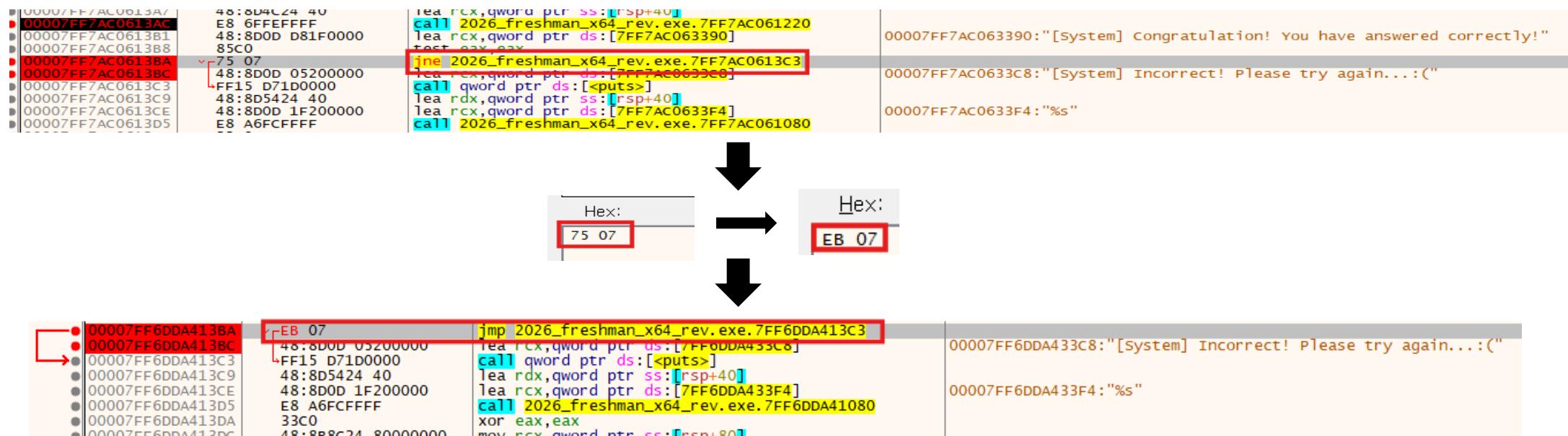
Problem Analysis and Solution

Advanced Problems

- [문제 8] 인라인 패치를 통해 입력 값과 관계 없이 항상 정답 메시지가 출력되도록 프로그램을 수정하시오. (패치한 위치, 성공 메시지 모두 나오도록 **캡쳐**하여 답안을 작성)

(5점)

- During dynamic analysis, the execution path changes depending on the **jne** instruction. Therefore, an inline patch was applied by changing **jne** to **jmp** (**Unconditional jump**).



The screenshot shows a debugger interface with assembly code and memory dump panes. In the assembly pane, a red box highlights the **jne** instruction at address 00007FF7AC0613C3. The memory dump pane shows the byte sequence 75 07 at that address. A flow diagram below illustrates the patch: 75 07 is converted to EB 07. The bottom assembly pane shows the modified code where the **jne** has been replaced by **jmp**.

```
00007FF7AC0613AC 48:8D4C24 40    lea rcx,qword ptr ss:[rsp+40]
00007FF7AC0613B1 E8 6FFFFFFF    call 2026_freshman_x64_rev.exe.7FF7AC061220
00007FF7AC0613B8 48:8D0D D81F0000  Test eax,eax
00007FF7AC0613BA 85C0
00007FF7AC0613BC 75 07    jne 2026_freshman_x64_rev.exe.7FF7AC0613C3
00007FF7AC0613C3 48:8D0D 05200000  lea rcx,qword ptr ds:[7FF7AC0633C8]
00007FF7AC0613C9 FF15 D71D0000  call qword ptr ds:[<puts>]
00007FF7AC0613CE 48:8D5424 40  lea rdx,qword ptr ss:[rsp+40]
00007FF7AC0613D5 48:8D0D 1F200000  lea rcx,qword ptr ds:[7FF7AC0633F4]
00007FF7AC0613D5 E8 A6FCFFFF  call 2026_freshman_x64_rev.exe.7FF7AC061080
00007FF7AC063390: "[System] Congratulation! You have answered correctly!"
00007FF7AC0633C8: "[System] Incorrect! Please try again...:("
00007FF7AC0633F4: "%s"

00007FF6DDA413BA 48:EB 07    jmp 2026_freshman_x64_rev.exe.7FF6DDA413C3
00007FF6DDA413BC 48:8D0D 05200000  lea rcx,qword ptr ds:[7FF6DDA433C8]
00007FF6DDA413C3 FF15 D71D0000  call qword ptr ds:[<puts>]
00007FF6DDA413C9 48:8D5424 40  lea rdx,qword ptr ss:[rsp+40]
00007FF6DDA413CE 48:8D0D 1F200000  lea rcx,qword ptr ds:[7FF6DDA433F4]
00007FF6DDA413D5 E8 A6FCFFFF  call 2026_freshman_x64_rev.exe.7FF6DDA41080
00007FF6DDA433C8: "[System] Incorrect! Please try again...:("
00007FF6DDA433F4: "%s"
```

Problem Analysis and Solution

Advanced Problems

- [문제 8] 인라인 패치를 통해 입력 값과 관계 없이 항상 정답 메시지가 출력되도록 프로그램을 수정하시오. (패치한 위치, 성공 메시지 모두 나오도록 **캡쳐**하여 답안을 작성)
(5점)

```
=====
Reverse Engineering Test for First Semester of 2026
-x64 binary version-
Made by. SHS
=====
Please enter an ID within 10 characters : minhyuk
Please enter a flag : jwqfwejinfojkn
[System] Welcome to DFRC!
[System] Congratulations! You have answered correctly!
|
```

Problem Analysis and Solution

Additional Problems

- [문제 9] 리버싱과 관련된 경험 (연구, 개인 공부, 대회, 워게임 등) 이 있다면 적어주세요. (1점)
 - [Ransomware Reversing]
 - I have experience analyzing the **Conti ransomware** as part of a previous undergraduate assignment.
 - At the time, I identified **symmetric and asymmetric cryptographic functions** and their **invocation points** through **static analysis** (using objdump) and **dynamic analysis** (using gdb).
 - I then neutralized the symmetric encryption via **ELF injection**-based binary injection, and blocked the asymmetric encryption by overriding dynamic libraries using **LD_PRELOAD**.
 - [Copy-and-Patch JIT Compiler Reversing]
 - Conducted reverse engineering on an open-source **Copy-and-Patch JIT Compiler for research purposes**.
 - Identified that **system calls are used to change memory permissions** to executable when frequently used code regions are copy-and-patched.
 - Exploited this behavior to **obtain a shell**, and the research findings were **published as a paper**.
 - **Reference-1:** [Conti-Ransomware Reversing](#)
 - **Reference-2:** [Copy-and-Patch JIT Compiler Reversing Paper](#)

Problem Analysis and Solution

Additional Problems

- [문제 10] 본인이 연구실에서 리버싱을 통해 진행하고 싶은 연구 주제를 자유롭게 적어주세요. (1점)
 - Research on Incident Response and Digital Forensics through Reverse Engineering Analysis of RaaS-based Ransomware (**Draft**)
 - **Research Motivation**
: The widespread adoption of Ransomware-as-a-Service (RaaS) has increased the complexity of analyzing ransomware-related security incidents.
 - **Research Objective**
: This study aims to analyze the internal behavior and configuration of ransomware through reverse engineering and to apply the findings to incident response and digital forensics.
 - **Expected Outcomes**
: The analysis is expected to support accurate damage assessment, reconstruction of attack timelines, and identification of attacks conducted by the same threat actor.

Conclusion

Conclusion

- This problem effectively demonstrated the efficiency of both dynamic and static analysis.
- First, the entry point (EP) was identified through dynamic analysis, and the overall behavior of the main function was inferred.
- Subsequently, the behavior and roles of each function called within main were analyzed.
- By understanding the behavior and roles of these functions, the structure of the FLAG could be derived.
- Additionally, a success message could be obtained simply by modifying the machine code through an inline patch.

- From a personal perspective, this assignment helped me regain interest in reverse engineering after having stepped away from it for some time, and it clearly illustrated why both static and dynamic analysis need to be used together.

Thank You

Q & A



고려대학교 정보보호대학원
Korea University
School of Cybersecurity



DFRC Korea University
Digital Forensic Research Center