

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



DFRC Korea University
Digital Forensic Research Center

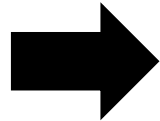
- ☐ **Introduction to Assignment**
- ☐ **Analysis Environment**
- ☐ **Problem Analysis and Solution**
- ☐ **Conclusion**

Introduction to Assignment

Initial Screen



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 : |
```

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

■ Static Analysis Tool

- Ghidra 11.4.2 Public

■ Dynamic Analysis Tool

- x64dbg

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...



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...:(
```

Basic Problems

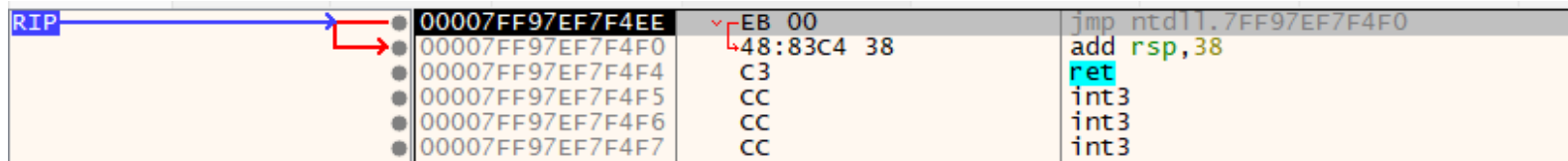
- [문제 1] 동적 분석과 정적 분석에 대해 간단히 설명하고, 리버싱에서 두 분석이 모두 필요한 이유에 대해 간단히 서술하시오. (1점)
 - **Dynamic Analysis?**
 - ➔ A method of analysis that observes a program's behavior while it is actually running. It involves using a debugger to analyze runtime behavior by examining **registers, memory changes, API calls, 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

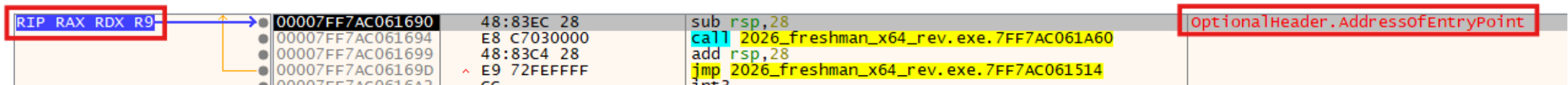
- [문제 1] 동적 분석과 정적 분석에 대해 간단히 설명하고, 리버싱에서 두 분석이 모두 필요한 이유에 대해 간단히 서술하시오. (1점)
 - Why are both analyses necessary in reverse engineering?
 - ➔ Static analysis alone makes it difficult to identify values determined at **runtime, conditional behavior, obfuscated code, or packed code**.
 - ➔ Dynamic analysis 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.

Basic Problems

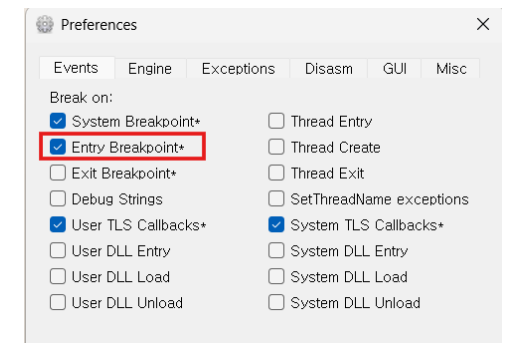
- [문제 2] 동적 분석 도구를 통해 문제 파일을 실행한 후, 해당 파일의 EP(Entry Point)를 식별하시오. (2점)



F9 Click



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



Core Problems

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

RIP RAX RDX R9					
00007FF7AC061690	48:83EC 28	sub rsp,28		OptionalHeader.AddressOfEntryPoint	
00007FF7AC061694	E8 C7030000	call 2026_freshman_x64_rev.exe.7FF7AC061A60			
00007FF7AC061699	48:83C4 28	add rsp,28			
00007FF7AC06169D	E9 72FEFFFF	jmp 2026_freshman_x64_rev.exe.7FF7AC061514			
00007FF7AC0616A2	CC	int3			
00007FF7AC0616A3	CC	int3			
00007FF7AC0616A4	40:53	push rbx			
00007FF7AC0616A6	48:83EC 20	sub rsp,20			
00007FF7AC0616AA	48:8BD9	mov rbx,rcx			
00007FF7AC0616AD	33C9	xor ecx,ecx			
00007FF7AC0616AF	FF15 6B190000	call qword ptr ds:[<SetUnhandledExceptionFilter>]			
00007FF7AC0616B5	48:8BCB	mov rcx,rbx			
00007FF7AC0616B8	FF15 5A190000	call qword ptr ds:[<unhandledExceptionFilter>]			
00007FF7AC0616BE	FF15 64190000	call qword ptr ds:[<GetCurrentProcess>]			
00007FF7AC0616C4	48:8BC8	mov rcx,rax			rax:EntryPoint
00007FF7AC0616C7	BA 090400C0	mov edx,C0000409			
00007FF7AC0616CC	48:83C4 20	add rsp,20			
00007FF7AC0616D0	5B	pop rbx			
00007FF7AC0616D1	- 48:FF25 98190000	jmp qword ptr ds:[<TerminateProcess>]			JMP.&TerminateProcess
00007FF7AC0616D8	48:894C24 08	mov qword ptr ss:[rsp+8],rcx			
00007FF7AC0616DD	48:83EC 38	sub rsp,38			
00007FF7AC0616E1	B9 17000000	mov ecx,17			
00007FF7AC0616E6	FF15 7C190000	call qword ptr ds:[<IsProcessorFeaturePresent>]			
00007FF7AC0616EC	85C0	test eax,eax			
00007FF7AC0616EE	74 07	je 2026_freshman_x64_rev.exe.7FF7AC0616F7			
00007FF7AC0616F0	B9 02000000	mov ecx,2			
00007FF7AC0616F5	CD 29	int 29			
00007FF7AC0616F7	48:8D0D 223A0000	lea rcx,qword ptr ds:[7FF7AC065120]			
00007FF7AC0616FE	E8 A9000000	call 2026_freshman_x64_rev.exe.7FF7AC0617AC			
00007FF7AC061703	48:8B4424 38	mov rax,qword ptr ss:[rsp+38]			rax:EntryPoint
00007FF7AC061708	48:8905 093B0000	mov qword ptr ds:[7FF7AC065218],rax			rax:EntryPoint
00007FF7AC06170F	48:8D4424 38	lea rax,qword ptr ss:[rsp+38]			rax:EntryPoint
00007FF7AC061714	48:83C0 08	add rax,8			rax:EntryPoint
00007FF7AC061718	48:8905 993A0000	mov qword ptr ds:[7FF7AC0651B8],rax			rax:EntryPoint
00007FF7AC06171F	48:8B05 F23A0000	mov rax,qword ptr ds:[7FF7AC065218]			rax:EntryPoint
00007FF7AC061726	48:8905 63390000	mov qword ptr ds:[7FF7AC065090],rax			rax:EntryPoint
00007FF7AC06172D	48:8B4424 40	mov rax,qword ptr ss:[rsp+40]			rax:EntryPoint

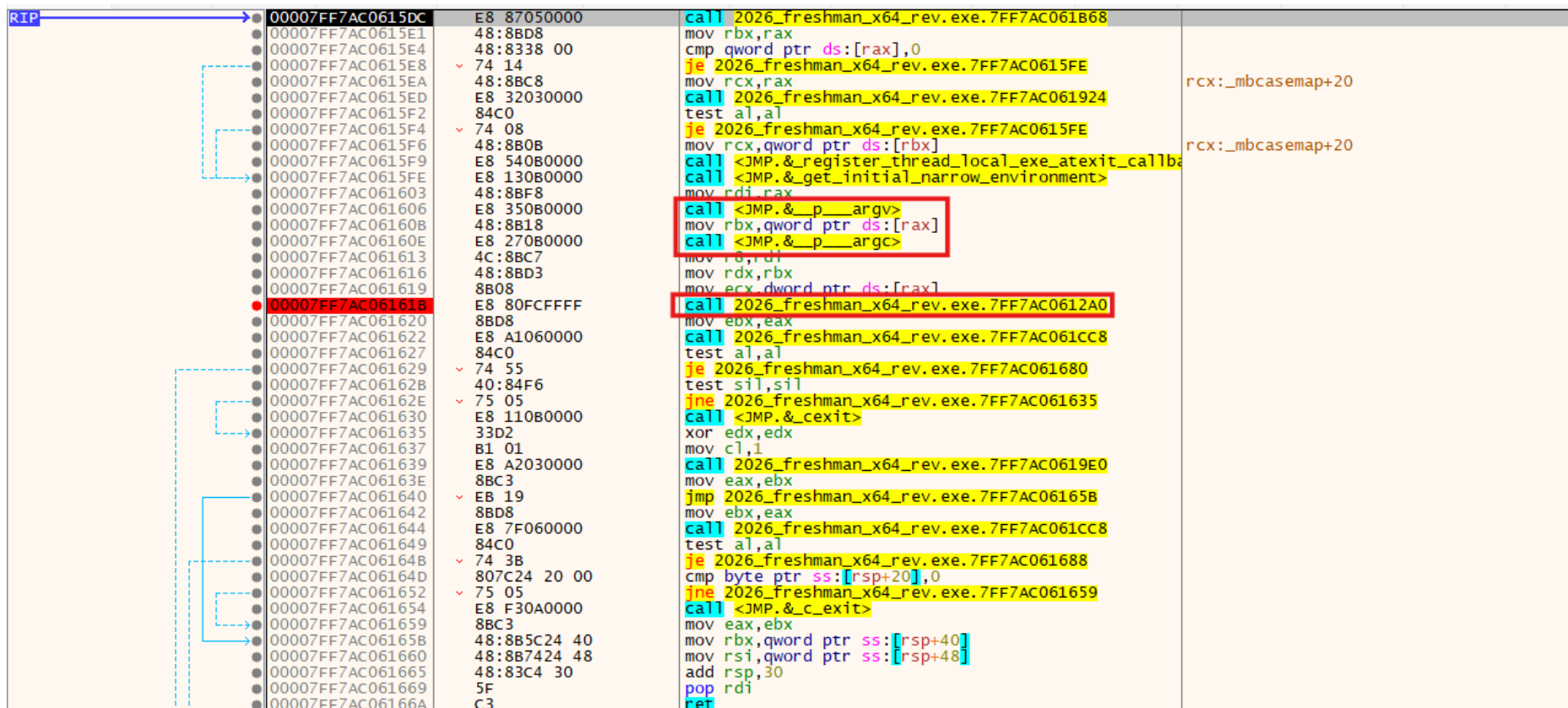
Core Problems

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

RIP	Address	Disassembly	Comment
→	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 bl,al	
	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	rdx:EntryPoint
	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	✓ E9 D9000000 jmp 2026_freshman_x64_rev.exe.7FF7AC06165B	
	00007FF7AC061582	48:8D15 7F1C0000 lea rdx,qword ptr ds:[7FF7AC063208]	rdx:EntryPoint
	00007FF7AC061589	48:8D0D 681C0000 lea rcx,qword ptr ds:[7FF7AC0631F8]	
	00007FF7AC061590	E8 870B0000 call <JMP.&_initterm_e>	
	00007FF7AC061595	C705 51400000 02000000 mov dword ptr ds:[7FF7AC0655F0],2	
	00007FF7AC06159F	✓ EB 08 jmp 2026_freshman_x64_rev.exe.7FF7AC0615A9	
	00007FF7AC0615A1	40:B6 01 mov sil,1	
	00007FF7AC0615A4	40:887424 20 mov byte ptr ss:[rsp+20],sil	
	00007FF7AC0615A9	8ACB mov cl,bl	
	00007FF7AC0615AB	E8 0C040000 call 2026_freshman_x64_rev.exe.7FF7AC0619BC	
	00007FF7AC0615B0	E8 AB050000 call 2026_freshman_x64_rev.exe.7FF7AC061B60	
	00007FF7AC0615B5	48:8BD8 mov rbx,rax	
	00007FF7AC0615B8	48:8338 00 cmp qword ptr ds:[rax],0	
	00007FF7AC0615BC	✓ 74 1E je 2026_freshman_x64_rev.exe.7FF7AC0615DC	
	00007FF7AC0615BE	48:8BC8 mov rcx,rax	
	00007FF7AC0615C1	E8 5E030000 call 2026_freshman_x64_rev.exe.7FF7AC061924	
	00007FF7AC0615C6	84C0 test al,al	
	00007FF7AC0615C8	✓ 74 12 je 2026_freshman_x64_rev.exe.7FF7AC0615DC	

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.



Address	Disassembly	Comment
00007FF7AC0615DC	call 2026_freshman_x64_rev.exe.7FF7AC061B68	
00007FF7AC0615E1	mov rbx, rax	
00007FF7AC0615E4	cmp qword ptr ds:[rax], 0	
00007FF7AC0615E8	je 2026_freshman_x64_rev.exe.7FF7AC0615FE	
00007FF7AC0615EA	mov rcx, rax	rcx:_mbcasemap+20
00007FF7AC0615ED	call 2026_freshman_x64_rev.exe.7FF7AC061924	
00007FF7AC0615F2	test al, al	
00007FF7AC0615F4	je 2026_freshman_x64_rev.exe.7FF7AC0615FE	
00007FF7AC0615F6	mov rcx, qword ptr ds:[rbx]	rcx:_mbcasemap+20
00007FF7AC0615F9	call <JMP.&_register_thread_local_exe_atexit_callback>	
00007FF7AC0615FE	call <JMP.&_get_initial_narrow_environment>	
00007FF7AC061603	mov rdi, rax	
00007FF7AC061606	call <JMP.&_p__argv>	
00007FF7AC06160B	mov rbx, qword ptr ds:[rax]	
00007FF7AC06160E	call <JMP.&_p__argc>	
00007FF7AC061613	mov r8, rdi	
00007FF7AC061616	mov rdx, rbx	
00007FF7AC061619	mov ecx, dword ptr ds:[rax]	
00007FF7AC06161B	call 2026_freshman_x64_rev.exe.7FF7AC0612A0	
00007FF7AC061620	mov ebx, eax	
00007FF7AC061622	call 2026_freshman_x64_rev.exe.7FF7AC061CC8	
00007FF7AC061627	test al, al	
00007FF7AC061629	je 2026_freshman_x64_rev.exe.7FF7AC061680	
00007FF7AC06162B	test sil, sil	
00007FF7AC06162E	jne 2026_freshman_x64_rev.exe.7FF7AC061635	
00007FF7AC061630	call <JMP.&_cexit>	
00007FF7AC061635	xor edx, edx	
00007FF7AC061637	mov cl, 1	
00007FF7AC061639	call 2026_freshman_x64_rev.exe.7FF7AC0619E0	
00007FF7AC06163E	mov eax, ebx	
00007FF7AC061640	jmp 2026_freshman_x64_rev.exe.7FF7AC06165B	
00007FF7AC061642	mov ebx, eax	
00007FF7AC061644	call 2026_freshman_x64_rev.exe.7FF7AC061CC8	
00007FF7AC061649	test al, al	
00007FF7AC06164B	je 2026_freshman_x64_rev.exe.7FF7AC061688	
00007FF7AC06164D	cmp byte ptr ss:[rsp+20], 0	
00007FF7AC061652	jne 2026_freshman_x64_rev.exe.7FF7AC061659	
00007FF7AC061654	call <JMP.&_c_exit>	
00007FF7AC061659	mov eax, ebx	
00007FF7AC06165B	mov rbx, qword ptr ss:[rsp+40]	
00007FF7AC061660	mov rsi, qword ptr ss:[rsp+48]	
00007FF7AC061665	add rsp, 30	
00007FF7AC061669	pop rdi	
00007FF7AC06166A	ret	

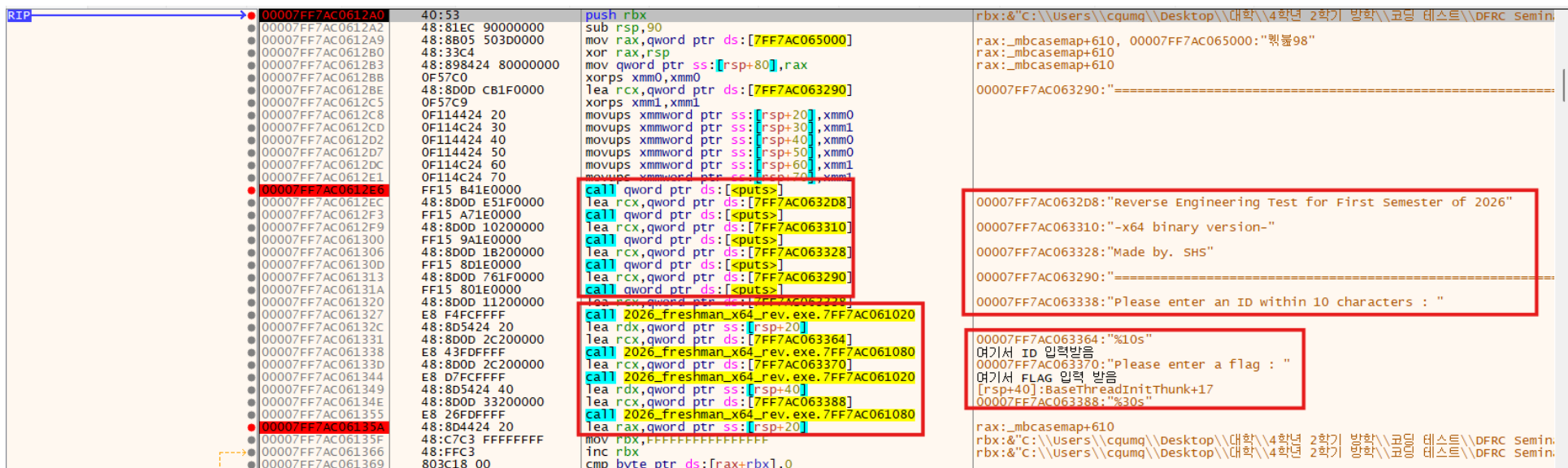
Core Problems

- [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	Address	Disassembly	Comment
	00007FF7AC0612A0	40:53 push rbx	rbx:&"C:\\Users\\cqumq\\Desktop\\대학\\4학년 2학기 방학\\코딩 테스트\\DFRC Semin
	00007FF7AC0612A2	48:81EC 90000000 sub rsp,90	
	00007FF7AC0612A9	48:8805 503D0000 mov rax,qword ptr ds:[7FF7AC065000]	rax:_mbcasemap+610, 00007FF7AC065000:"퀵98"
	00007FF7AC0612B0	48:33C4 xor rax,rsp	rax:_mbcasemap+610
	00007FF7AC0612B3	48:898424 80000000 mov qword ptr ss:[rsp+80],rax	rax:_mbcasemap+610
	00007FF7AC0612BB	0F57C0 xorps xmm0,xmm0	
	00007FF7AC0612BE	48:8D0D CB1F0000 lea rcx,qword ptr ds:[7FF7AC063290]	00007FF7AC063290:"=====
	00007FF7AC0612C5	0F57C9 xorps xmm1,xmm1	
	00007FF7AC0612C8	0F114424 20 movups xmmword ptr ss:[rsp+20],xmm0	
	00007FF7AC0612CD	0F114C24 30 movups xmmword ptr ss:[rsp+30],xmm1	
	00007FF7AC0612D2	0F114424 40 movups xmmword ptr ss:[rsp+40],xmm0	
	00007FF7AC0612D7	0F114424 50 movups xmmword ptr ss:[rsp+50],xmm0	
	00007FF7AC0612DC	0F114C24 60 movups xmmword ptr ss:[rsp+60],xmm1	
	00007FF7AC0612E1	0F114C24 70 movups xmmword ptr ss:[rsp+70],xmm1	
	00007FF7AC0612E6	FF15 B41E0000 call qword ptr ds:[<puts>]	
	00007FF7AC0612EC	48:8D0D E51F0000 lea rcx,qword ptr ds:[7FF7AC0632D8]	00007FF7AC0632D8:"Reverse Engineering Test for First Semester of 2026"
	00007FF7AC0612F3	FF15 A71E0000 call qword ptr ds:[<puts>]	00007FF7AC063310:"-x64 binary version-"
	00007FF7AC0612F9	48:8D0D 10200000 lea rcx,qword ptr ds:[7FF7AC063310]	00007FF7AC063328:"Made by. SHS"
	00007FF7AC061300	FF15 9A1E0000 call qword ptr ds:[<puts>]	00007FF7AC063290:"=====
	00007FF7AC061306	48:8D0D 1B200000 lea rcx,qword ptr ds:[7FF7AC063328]	00007FF7AC063338:"Please enter an ID within 10 characters : "
	00007FF7AC06130D	FF15 8D1E0000 call qword ptr ds:[<puts>]	
	00007FF7AC061313	48:8D0D 761F0000 lea rcx,qword ptr ds:[7FF7AC063290]	
	00007FF7AC06131A	FF15 801E0000 call qword ptr ds:[<puts>]	
	00007FF7AC061320	48:8D0D 11200000 lea rcx,qword ptr ds:[7FF7AC063338]	
	00007FF7AC061327	E8 F4FCFFFF call 2026_freshman_x64_rev.exe.7FF7AC061020	
	00007FF7AC06132C	48:8D5424 20 lea rdx,qword ptr ss:[rsp+20]	
	00007FF7AC061331	48:8D0D 2C200000 lea rcx,qword ptr ds:[7FF7AC063364]	00007FF7AC063364:"%10s"
	00007FF7AC061338	E8 43FDFFFF call 2026_freshman_x64_rev.exe.7FF7AC061080	여기서 ID 입력받음
	00007FF7AC06133D	48:8D0D 2C200000 lea rcx,qword ptr ds:[7FF7AC063370]	00007FF7AC063370:"Please enter a flag : "
	00007FF7AC061344	E8 D7FCFFFF call 2026_freshman_x64_rev.exe.7FF7AC061020	여기서 FLAG 입력 받음
	00007FF7AC061349	48:8D5424 40 lea rdx,qword ptr ss:[rsp+40]	[rsp+40]:BaseThreadInitThunk+17
	00007FF7AC06134E	48:8D0D 33200000 lea rcx,qword ptr ds:[7FF7AC063388]	00007FF7AC063388:"%30s"
	00007FF7AC061355	E8 26FDFFFF call 2026_freshman_x64_rev.exe.7FF7AC061080	
	00007FF7AC06135A	48:8D4424 20 lea rax,qword ptr ss:[rsp+20]	
	00007FF7AC06135F	48:C7C3 FFFFFFFF mov rbx,FFFFFFFFFFFFFFFF	rax:_mbcasemap+610
	00007FF7AC061366	48:FFC3 inc rbx	rbx:&"C:\\Users\\cqumq\\Desktop\\대학\\4학년 2학기 방학\\코딩 테스트\\DFRC Semin
	00007FF7AC061369	803C18 00 cmp byte ptr ds:[rax+rbx],0	rbx:&"C:\\Users\\cqumq\\Desktop\\대학\\4학년 2학기 방학\\코딩 테스트\\DFRC Semin

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.



```
40:53      push rbx
48:81E0 90000000  sub rsp,90
48:8B05 503D0000  mov rax,qword ptr ds:[7FF7AC065000]
48:33C4      xor rax,rsp
48:898424 80000000  mov qword ptr ss:[rsp+80],rax
0F57C0      xorps xmm0,xmm0
48:8D0D CB1F0000  lea rcx,qword ptr ds:[7FF7AC063290]
0F57C9      xorps xmm1,xmm1
0F114424 20      movups xmmword ptr ss:[rsp+20],xmm0
0F114424 30      movups xmmword ptr ss:[rsp+30],xmm1
0F114424 40      movups xmmword ptr ss:[rsp+40],xmm0
0F114424 50      movups xmmword ptr ss:[rsp+50],xmm0
0F114424 60      movups xmmword ptr ss:[rsp+60],xmm1
0F114424 70      movups xmmword ptr ss:[rsp+70],xmm1
FF15 B41E0000  call qword ptr ds:[puts]
48:8D0D E51F0000  lea rcx,qword ptr ds:[7FF7AC0632D8]
FF15 A71E0000  call qword ptr ds:[puts]
48:8D0D 10200000  lea rcx,qword ptr ds:[7FF7AC063310]
FF15 9A1E0000  call qword ptr ds:[puts]
48:8D0D 1B200000  lea rcx,qword ptr ds:[7FF7AC063328]
FF15 8D1E0000  call qword ptr ds:[puts]
48:8D0D 761F0000  lea rcx,qword ptr ds:[7FF7AC063290]
FF15 801E0000  call qword ptr ds:[puts]
48:8D0D 11200000  lea rcx,qword ptr ds:[7FF7AC0632D8]
E8 F4FCFFFF  call 2026_freshman_x64_rev.exe.7FF7AC061020
48:8D5424 20      lea rdx,qword ptr ss:[rsp+20]
48:8D0D 2C200000  lea rcx,qword ptr ds:[7FF7AC063364]
E8 43FDFFFF  call 2026_freshman_x64_rev.exe.7FF7AC061080
48:8D0D 2C200000  lea rcx,qword ptr ds:[7FF7AC063370]
E8 D7FCFFFF  call 2026_freshman_x64_rev.exe.7FF7AC061020
48:8D5424 40      lea rdx,qword ptr ss:[rsp+40]
48:8D0D 33200000  lea rcx,qword ptr ds:[7FF7AC063388]
E8 26FDFFFF  call 2026_freshman_x64_rev.exe.7FF7AC061080
48:8D4424 20      lea rax,qword ptr ss:[rsp+20]
48:C7C3 FFFFFFFF  mov rbx,FFFFFFFFFFFFFFFF
48:FFC3      inc rbx
803C18 00      cmp byte ptr ds:[rax+rbx],0
```

Registers:

- rax: 00007FF7AC065000: "Reverse Engineering Test for First Semester of 2026"
- rax: 00007FF7AC065000: "x64 binary version-"
- rax: 00007FF7AC065000: "Made by. SHS"
- rax: 00007FF7AC063290: "Please enter an ID within 10 characters : "
- rax: 00007FF7AC063364: "%10s"
- rax: 00007FF7AC063370: "Please enter a flag : "
- rax: 00007FF7AC063388: "%30s"

Core Problems

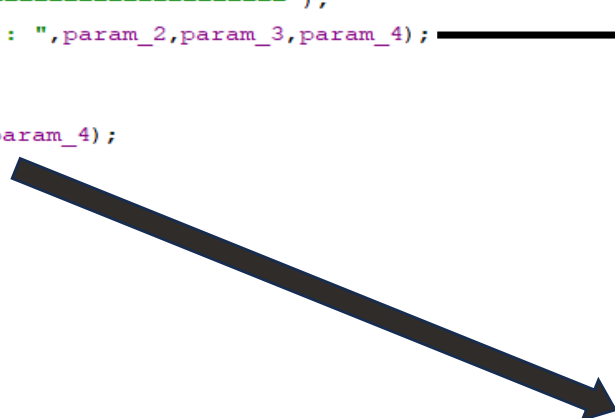
- [문제 4] main함수의 전체 코드 흐름에 대해 본인이 파악한 대로 **추론**하여 서술하시오.
(동적 및 정적 분석 중 본인이 편한 방법을 사용하되, 두 방법 모두 사용해도 무관함) (3점)
 - STEP 2) Through static analysis, it can be identified that in **FUN_1400012a0**, messages are **printed via puts**, and **user input is received** through **FUN_140001020** and **FUN_140001080**.

```
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;
```



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 = 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);
```



```
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;
}
```

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;
}
```

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;
}
```

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;
```

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

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;
}
```


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 = 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);
```

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;
}
```

Core Problems

- [문제 6] 프로그램의 **Key 생성 방식**에 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;
}
```

Core Problems

- [문제 6] 프로그램의 **Key 생성 방식**에 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;
}
```

Core Problems

- [문제 6] 프로그램의 **Key 생성 방식**에 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;
}
```

Core Problems

- [문제 6] 프로그램의 **Key 생성 방식**에 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;
}
```

Advanced Problems

- [문제 7] 현재까지 해결했던 문제들을 바탕으로, 제공된 프로그램의 정답을 출력하시오.
(프로그램의 ID, Flag, 성공 메시지 모두 나오도록 **캡처**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;
}
```

Advanced Problems

- [문제 7] 현재까지 해결했던 문제들을 바탕으로, 제공된 프로그램의 정답을 출력하시오.
(프로그램의 ID, Flag, 성공 메시지 모두 나오도록 **캡처**하여 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);
```


Advanced Problems

- [문제 7] 현재까지 해결했던 문제들을 바탕으로, 제공된 프로그램의 정답을 출력하시오.
(프로그램의 ID, Flag, 성공 메시지 모두 나오도록 **캡처**하여 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;
}
```

Advanced Problems

- [문제 7] 현재까지 해결했던 문제들을 바탕으로, 제공된 프로그램의 정답을 출력하시오.
(프로그램의 ID, Flag, 성공 메시지 모두 나오도록 **캡처**하여 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:** $(\text{FLAG}[\text{iVar2} + \text{param}_3] + 2) \wedge 0xF9 \wedge \text{param}_3$
 - **Right-hand side:** $0xF9 \wedge \text{param}_3 \wedge \text{fixed bytes}$
- Thus, $\text{FLAG}[\text{iVar2} + \text{param}_3] = \text{fixed_byte} - 2$.
- $(0x74, 0x35, 0x78, 0x67, 0x54, 0x75, 0x4B, 0x70, 0x69) - 2$
= `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 = 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;
}
```

Advanced Problems

- [문제 7] 현재까지 해결했던 문제들을 바탕으로, 제공된 프로그램의 정답을 출력하시오.
(프로그램의 ID, Flag, 성공 메시지 모두 나오도록 **캡처**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!
```

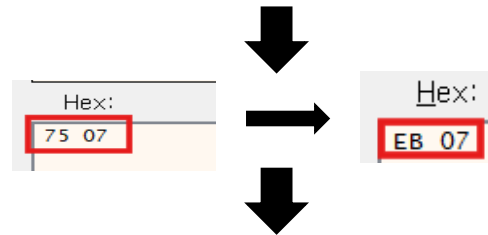
Advanced Problems

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

00007FF7AC0613A7	48:8D4C24 40	lea rcx,qword ptr ss:[rsp+40]	
00007FF7AC0613AC	E8 6FFFFFFFFF	call 2026_freshman_x64_rev.exe.7FF7AC061220	
00007FF7AC0613B1	48:8D0D D81F0000	lea rcx,qword ptr ds:[7FF7AC063390]	00007FF7AC063390:"[System] Congratulation! You have answered correctly!"
00007FF7AC0613B8	85C0	test rcx,rcx	
00007FF7AC0613BA	75 07	jne 2026_freshman_x64_rev.exe.7FF7AC0613C3	
00007FF7AC0613BC	48:8D0D 05200000	lea rcx,qword ptr ds:[7FF7AC0633C8]	00007FF7AC0633C8:"[System] Incorrect! Please try again...:("
00007FF7AC0613C3	FF15 D71D0000	call qword ptr ds:[<puts>]	
00007FF7AC0613C9	48:8D5424 40	lea rdx,qword ptr ss:[rsp+40]	
00007FF7AC0613CE	48:8D0D 1F200000	lea rcx,qword ptr ds:[7FF7AC0633F4]	00007FF7AC0633F4:"%s"
00007FF7AC0613D5	E8 A6FCFFFF	call 2026_freshman_x64_rev.exe.7FF7AC061080	



00007FF6DDA413BA	EB 07	jmp 2026_freshman_x64_rev.exe.7FF6DDA413C3	
00007FF6DDA413BC	48:8D0D 05200000	lea rcx,qword ptr ds:[7FF6DDA433C8]	00007FF6DDA433C8:"[System] Incorrect! Please try again...:("
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]	00007FF6DDA433F4:"%s"
00007FF6DDA413D5	E8 A6FCFFFF	call 2026_freshman_x64_rev.exe.7FF6DDA41080	
00007FF6DDA413DA	33C0	xor eax,eax	
00007FF6DDA413DC	48:8B8C24 80000000	mov rcx,qword ptr ss:[rsp+80]	

Advanced Problems

- [문제 8] 인라인 패치를 통해 입력 값과 관계 없이 항상 정답 메시지가 출력되도록 프로그램을 수정하십시오. (패치한 위치, 성공 메시지 모두 나오도록 **캡처**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 : jwqfwejnfojkn
[System] Welcome to DFRC!
[System] Congratulation! You have answered correctly!
|
```

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](#)

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