

Pwndbg와 함께하는

# Reversing\_Wargame

✓  
**목차**

01

PwnDbg?

02

PwnDbg :  
Why?

03

PwnDbg :  
Contents

04

Vs :  
x64Dbg

05

Training :  
Small Counter

06

Training :  
Check Function  
Argument

07

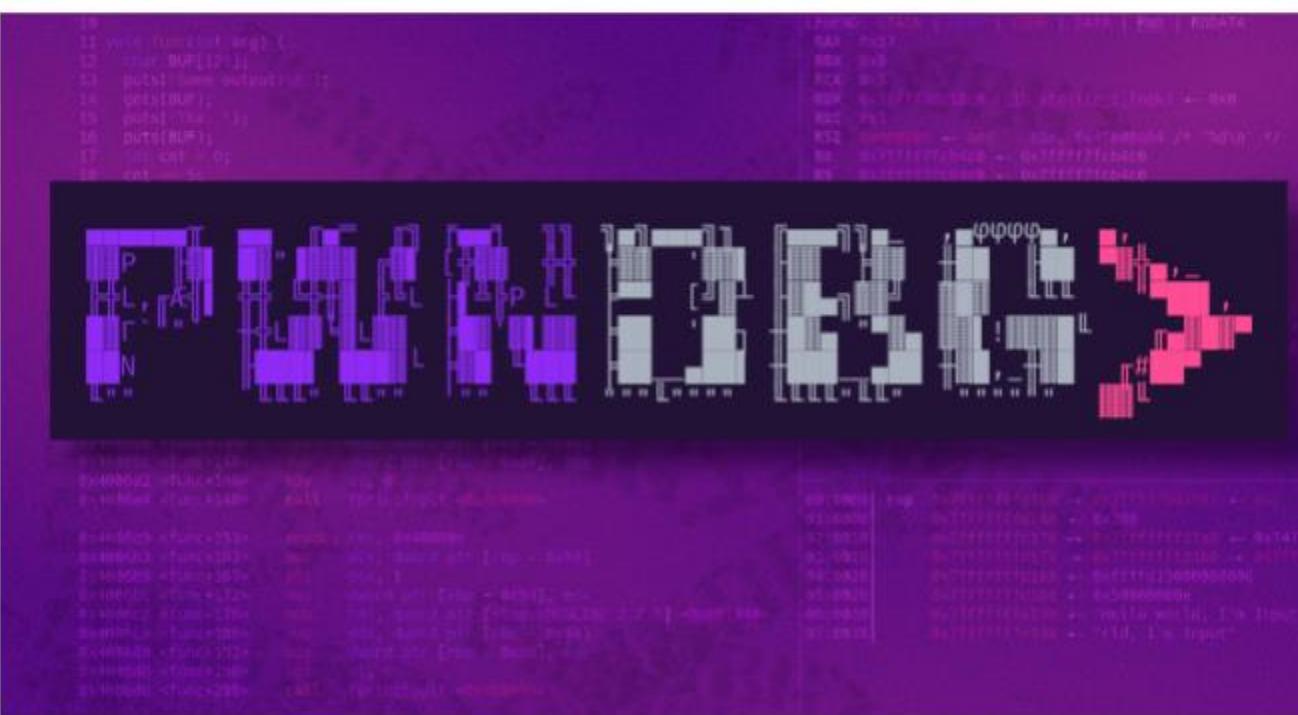
Training :  
Check Return  
Value

08

Training :  
ez\_rev

Reversing\_with

# 01 | PwnDbg?



**GDB에서 실행된 프로그램의 주요 메모리들의 상태를  
가독성 있게 표현해주는 인터페이스 플러그인**

- GDB (GNU's DeBugger) : 리눅스의 대표 Debugger 중 하나
- 디버거를 이용하여 프로그램의 실행 과정을 자세히 관찰하려면 컴퓨터의 각종 메모리를 한눈에 파악할 수 있는 것이 좋다.

## 02 | PwnDbg : Why?

```
7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00 ELF.....  
02 00 03 00 01 00 00 00 00 90 04 08 34 00 00 00 .....4.  
10 23 00 00 00 00 00 00 34 00 20 00 03 00 28 00 .#....4. ....(.  
07 00 06 00 01 00 00 00 00 00 00 00 80 04 08 .....  
00 80 04 08 94 00 00 00 94 00 00 00 04 00 00 00 .....  
00 10 00 00 01 00 00 00 00 10 00 00 00 90 04 08 .....  
00 90 04 08 05 02 00 00 05 02 00 00 05 00 00 00 .....  
00 10 00 00 01 00 00 00 00 20 00 00 00 A0 04 08 .....  
00 A0 04 08 0A 01 00 00 10 01 00 00 06 00 00 00 .....  
00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Commands are comma separated.

잘못 된 PE 파일!

**PE 파일 ( .exe, .dll ... ) 은 윈도우에서,  
ELF 파일은 리눅스/유닉스에서!**

- 윈도우 실행 파일인 PE 파일의 경우 ollyDbg, x64Dbg에서 디버깅 가능!
- 그렇기에 당연히 리눅스/유닉스 실행파일인 ELF 파일의 경우 리눅스/유닉스 환경에서 디버깅 하는 것!

# 03 | PwnDbg : Contents

```

Breakpoint 14, 0x0000000000401d3d in xor_with_key ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]

*RAX 0x72
RBX 0x7fffffffdfc0 ← 0x7752734e40635e25 ('%^c@NsRw')
*RCX 0x18
*RDX 0x3
RDI 0x18
RSI 0x7
R8 0x0
R9 0x4eeba0 ← 'E*aATiRtrArEhURZx[r=TURZg\n'
R10 0xfffffffffffffb8
R11 0x0
R12 0x19
R13 0x7fffffffef8 → 0x7fffffff43a ← '/mnt/c/Jihoon/2024/My Own/Reversing/Dreamhack/ez_rev/prob'
R14 0x4e17c8 (__preinit_array_start) → 0x401b80 (frame_dummy) ← endbr64
R15 0x2
RBP 0x4b6042 ← 0x73716b72736b71 /* 'qksrkqs' */
RSP 0x7fffffffdfc0 → 0x7fffffff208 → 0x7fffffff474 ← 'SHELL=/bin/bash'
RIP 0x401d3d (xor_with_key+61) ← xor byte ptr [rbx + rcx], al
[ DISASM / x86-64 / set emulate on ]

0x401d30 <xor_with_key+48>    mov    eax, ecx
0x401d32 <xor_with_key+50>    cdq
0x401d33 <xor_with_key+51>    idiv   esi
0x401d35 <xor_with_key+53>    movsd  rdx, edx
0x401d38 <xor_with_key+56>    movzx  eax, byte ptr [rbp + rdx]
0x401d3d <xor_with_key+61>    xor    byte ptr [rbx + rcx], al
0x401d40 <xor_with_key+64>    mov    rax, rcx
0x401d43 <xor_with_key+67>    add    rcx, 1
0x401d47 <xor_with_key+71>    cmp    rax, rdi
0x401d4a <xor_with_key+74>    jne    xor_with_key+48
0x401d4c <xor_with_key+76>    pop    rbx
[ STACK ]

00:0000| rsp 0x7fffffffdfc0 → 0x7fffffff208 → 0x7fffffff474 ← 'SHELL=/bin/bash'
01:0008| 0x7fffffffdfc0 → 0x7fffffffdfc0 ← 0x7752734e40635e25 ('%^c@NsRw')
02:0010| 0x7fffffffdfc0 ← 0x1
03:0018| 0x7fffffffdfc0 → 0x4019f6 (main+118) ← mov esi, 3
04:0020| rbx 0x7fffffffdfc0 ← 0x7752734e40635e25 ('%^c@NsRw')
05:0028| 0x7fffffffdfc0 ← 0x5a5155725f734372 ('rCs_rUQZ')
06:0030| 0x7fffffffdfc0 ← 0x4248565427685a7a ("zZh'TVHB")
07:0038| 0x7fffffffdfc0 → 0x450014 (getcwd+916) ← 0xfffffb8c3c748ffff

[ BACKTRACE ]

* 0     0x401d3d xor_with_key+61
1     0x4019f6 main+118
2     0x40218a __libc_start_call_main+106
3     0x403a27 __libc_start_main_impl+2599
4     0x401ab5 _start+37

pwndbg>
Continuing.
KKKKKKKKKKKKKKKKKK[Inferior 1 (process 119) exited normally]

```

레지스터, 코드, 스택 등의 정보를  
한눈에

- 레지스터 : CPU가 바로 사용할 수 있는 데이터
- Disassemble Code : 프로그램의 코드를 인  
간이 이해할 수 있는 언어로 역해석한 것
- 스택 : 함수의 호출과 관계되는 지역 변수와 매  
개변수가 저장되는 영역

# 04 | PwnDbg Vs : x64Dbg

The screenshot shows the x64Dbg debugger interface. The assembly pane displays a series of assembly instructions, many of which are identical or very similar, likely due to a loop or a repetitive pattern. The registers pane shows standard x64 registers (RAX, RBX, RCX, RDX, RBP, RSP, RSI, RDI) with their current values. The stack pane shows the current stack state, including the Return Address (RIP) at 00007FFEB804D4E4, which points to the nt!NtReadFileScatter function. The memory dump panes show the raw binary data of the memory dump, including ASCII and hex representations.

GUI의 편의성을 이길 순 없지만..

- CLI에서 실행되는 pwndbg  
→ 많은 정보를 한번에 보여주기 어려움
- 가독성에서도 크게 떨어지지 않고 CLI 이기에 속도도 더 빠르다!
- 그리고.. 리눅스 실행 파일을 디버깅 하려면 어쩔 수 없다.

# 05

---

## Training : Small Counter

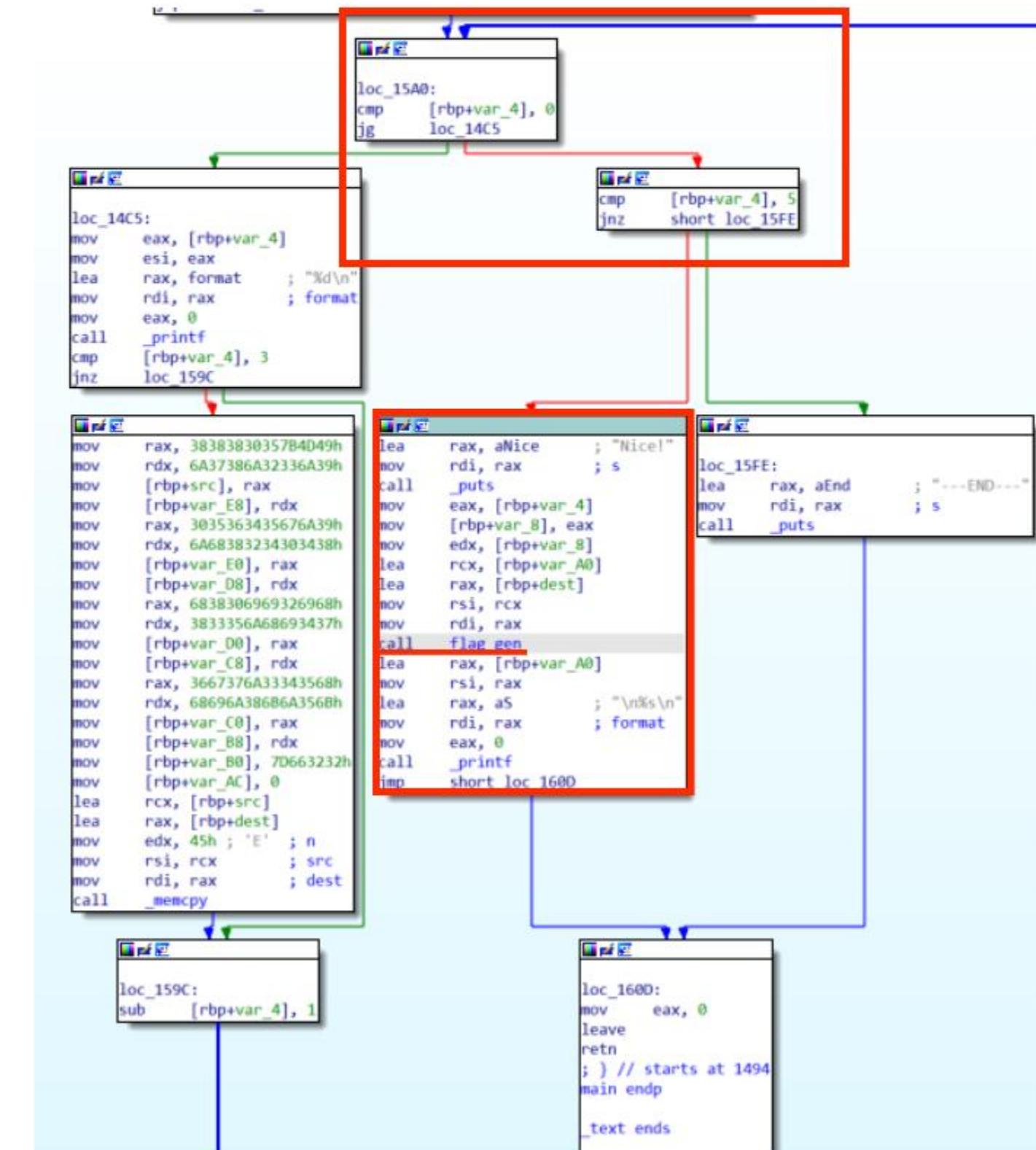
실행 : 10부터1까지 숫자를 세고 끝난다.

```
hoodscp@LAPTOP-0F5PGRC9:~/Dreamhack/small counter$ ./chall
----Counter----
10
9
8
7
6
5
4
3
2
1
----END----
hoodscp@LAPTOP-0F5PGRC9:~/Dreamhack/small counter$ █
```

05

# Training : Small Counter

정적분석(IDA) : 실행에서 확인했던 값이 0과 같거나 작으면서, 5여야 flag\_gen 함수에 도달?



## 05

# Training :

## Small Counter

동적분석(PwnDbg) : 분기점에 BP(명령어 b)

```

0x5555555555a0 <main+268>    cmp    dword ptr [rbp - 4], 0
► 0x5555555555a4 <main+272>    ✓jg    main+49                                <main+49>
    ↓
0x5555555555c5 <main+49>      mov    eax, dword ptr [rbp - 4]
0x5555555555c8 <main+52>      mov    esi, eax
0x5555555555ca <main+54>      lea    rax, [rip + 0xb41]
0x5555555555d1 <main+61>      mov    rdi, rax
0x5555555555d4 <main+64>      mov    eax, 0

00:0000| rsp 0x7fffffffdfc0 ← 0x0
... ↓    7 skipped

► 0 0x5555555555a4 main+272
1 0x7ffff7db6d90 __libc_start_call_main+128
2 0x7ffff7db6e40 __libc_start_main+128
3 0x555555555105 _start+37

pwndbg> b
Breakpoint 3 at 0x5555555555a4
pwndbg> █

```

05

# Training :

## Small Counter

첫 분기점을 지난 후 원하는 스택의 값을 변경  
(명령어 set)

스택의 값이 바뀌었는지 확인 (명령어 x)

continue (명령어 c) : 분기점 통과, flag 획득.

```

0x5555555555a0 <main+268>    cmp    dword ptr [rbp - 4], 0
0x5555555555a4 <main+272>    jg     main+49          <main+49>
                                cmp    dword ptr [rbp - 4], 5
▶ 0x5555555555aa <main+278>    jne   main+362        <main+362>
0x5555555555ae <main+282>    ↓
                                lea    rax, [rip + 0xalc]
0x5555555555fe <main+362>    mov    rdi, rax
0x555555555605 <main+369>    call   puts@plt      <puts@plt>
0x555555555608 <main+372>
                                mov    eax, 0

0x55555555560d <main+377>
```

00:0000	rsp 0x7fffffffdf0	← 'IM{508889j32j87j9jg54650840428hjhi2ii08h74i
01:0008	-0e8 0x7fffffffdf0	← '9j32j87j9jg54650840428hjhi2ii08h74ihj538h54
02:0010	-0e0 0x7fffffffdf0	← '9jg54650840428hjhi2ii08h74ihj538h543j7g6k5j
03:0018	-0d8 0x7fffffffdf0	← '840428hjhi2ii08h74ihj538h543j7g6k5jk8jih22f
04:0020	-0d0 0x7fffffff000	← 'hi2ii08h74ihj538h543j7g6k5jk8jih22f}'
05:0028	-0c8 0x7fffffff008	← '74ihj538h543j7g6k5jk8jih22f}'
06:0030	-0c0 0x7fffffff010	← 'h543j7g6k5jk8jih22f}'
07:0038	-0b8 0x7fffffff018	← 'k5jk8jih22f}'

```

▶ 0 0x5555555555aa main+278
1 0x7ffff7db6d90 __libc_start_call_main+128
2 0x7ffff7db6e40 __libc_start_main+128
3 0x55555555105 _start+37
```

```

pwndbg> set *(int *)($rbp-4)=5
pwndbg> x $rbp-4
0x7fffffff0cc: 0x00000005
```

```

pwndbg> c
Continuing.
Nice!
```

```

1782837817480325047325629763469728648923648926398432
[Inferior 1 (process 142) exited normally]
pwndbg>
```

06

# Training : Check Function Argument

실행 : 아래의 함수가 인자값으로 flag를 갖고 있다고 한다. 출력은 해주지 않는다.

```
hoodscp@LAPTOP-0F5PGRC9:~/Dreamhack/Check_Funtion_Argument$ ./check_function_argument
Below function takes the flag as an argument :)
Can you see that?
hoodscp@LAPTOP-0F5PGRC9:~/Dreamhack/Check_Funtion_Argument$
```

## 06

# Training : Check Function Argument

정적분석(IDA)

```

main proc near
; __ unwind {
endbr64
push rbp
mov rbp, rsp
lea rax, s ; "Below function to
mov rdi, rax ; s
call __puts
mov rax, cs:qword_4040D0
mov rdi, rax
call sub_4015E2 ; "Can you see that?"
lea rax, aCanYouSeeThat ; "Can you see that?"
mov rdi, rax ; s
call __puts
mov eax, 0
pop rbp
retn
; } // starts at 4015F1
main endp

```

```

__int64 __fastcall sub_401535(__int64 a1)
{
    __int64 result; // rax
    void *ptr; // [rsp+10h] [rbp-10h]
    __int64 v3; // [rsp+18h] [rbp-8h]

    sub_4012B7(a1, 11LL);
    sub_4011F6(a1, &unk_40200F);
    sub_40126A(a1, 99LL);
    ptr = (void *)sub_401301(a1, 70LL);
    sub_40138D(ptr);
    v3 = sub_401449(ptr);
    free(ptr);
    result = v3;
    qword_4040D0 = v3;
    return result;
}

```

```

void sub_4015E2()
{
    ;
}

```

## 06

# Training :

## Check

## Function

## Argument

동적분석(PwnDbg) : rdi = destination index

```
[ REGISTERS / show-flags
RAX 0x405340 ← 'ooh you figured out me :)' Flag is DH{1782837817480325047325629
RBX 0x0
RCX 0xfffff7ea1887 (write+23) ← cmp rax, -0x1000 /* 'H=' */
RDX 0x1
*RDI 0x405340 ← 'ooh you figured out me :)' Flag is DH{1782837817480325047325629
RSI 0x1
R8 0x0
R9 0x405390 ← 'Below function takes the flag as an argument :)\n'
R10 0x77
R11 0x246
R12 0xfffffffffe1a8 → 0xfffffffffe3e4 ← '/mnt/c/Jihoon/2024/My Own/Reversing/Dreamhack/Ch
R13 0x4015f1 ← endbr64
R14 0x403e18 → 0x4011c0 ← endbr64
R15 0xfffff7ffd040 (_rtld_global) → 0xfffff7ffe2e0 ← 0x0
RBP 0xfffffffffe090 ← 0x1
RSP 0xfffffffffe090 ← 0x1
*RIP 0x401612 ← call 0x4015e2
[ DISASM / x86-
0x4015f9    lea    rax, [rip + 0xa20]
0x401600    mov    rdi, rax
0x401603    call   puts@plt                                <puts@plt>
0x401608    mov    rax, qword ptr [rip + 0x2ac1]
0x40160f    mov    rdi, rax
► 0x401612    call   0x4015e2                                <0x4015e2>
0x401617    lea    rax, [rip + 0xa32]
0x40161e    mov    rdi, rax
0x401621    call   puts@plt                                <puts@plt>
0x401626    mov    eax, 0
0x40162b    pop    rbp
```

07

---

# Training : Check Return Value

실행 : 이전 문제와 비슷하지만 인자값이 아닌 리턴값에 flag가 있는듯 하다.

```
hoodscp@LAPTOP-0F5PGRC9:~/Dreamhack/Check_Return_Value$ ./check_return_value
OK. I will return flag.
I have returned the flag :)
hoodscp@LAPTOP-0F5PGRC9:~/Dreamhack/Check_Return_Value$ █
```

## 07

# Training :

## Check

# Return Value

정적분석(IDA)

```

main proc near
; __ unwind {
endbr64
push    rbp
mov     rbp, rsp
lea      rax, s           ; "OK."
mov     rdi, rax          ; s
call    __puts
lea      rax, unk_404080
mov     rdi, rax
call    sub_40152B
lea      rax, aIHaveReturnedT ; "I have returned the flag :)"
mov     rdi, rax          ; s
call    __puts
mov     eax, 0
pop     rbp
retn
; } // starts at 4015B6
main endp

```

```

int64 __fastcall sub_40152B(__int64 a1)
{
void *ptr; // [rsp+10h] [rbp-10h]
__int64 v3; // [rsp+18h] [rbp-8h]

sub_4012B7(a1, 44LL);
sub_4011F6(a1, &unk_40200B);
sub_40126A(a1, 77LL);
ptr = (void *)sub_401301(a1, 70LL);
sub_401388(ptr);
v3 = sub_401444(ptr);
free(ptr);
return v3;
}

```

## 07

# Training :

## Check

# Return Value

동적분석(PwnDbg) : rax : 함수의 반환값을 저장

```
[ REGISTERS / show-flags
*RAX 0x405750 ← 'ooh you figured out me :) Flag is DH1782837817480325047325629763
RBX 0x0
*RCX 0x8
*RDX 0x405
*RDI 0x7
*RSI 0x405010 ← 0x0
*R8 0x4056b0 ← 0x405
*R9 0x0
*R10 0x7fffff7d9a4d0 ← 0xf001200004dde
*R11 0x4e33747c449eb663
R12 0xfffffffffe1b8 → 0xfffffffffe3fa ← '/mnt/c/Jihoon/2024/My Own/Reversing/Dreamhack/Ch
R13 0x4015b6 ← endbr64
R14 0x403e18 → 0x4011c0 ← endbr64
R15 0x7ffff7ffd040 (_rtld_global) → 0x7ffff7ffe2e0 ← 0x0
RBP 0x7fffffff0a0 ← 0x1
RSP 0x7fffffff0a0 ← 0x1
*RIP 0x4015dc ← lea rax, [rip + 0xa4a]
[ DISASM / x86-
0x4015c5    mov    rdi, rax
0x4015c8    call   puts@plt                                <puts@plt>
0x4015cd    lea    rax, [rip + 0x2aac]
0x4015d4    mov    rdi, rax
0x4015d7    call   0x40152b                                <0x40152b>
▶ 0x4015dc    lea    rax, [rip + 0xa4a]
0x4015e3    mov    rdi, rax
0x4015e6    call   puts@plt                                <puts@plt>
0x4015eb    mov    eax, 0
0x4015f0    pop    rbp
0x4015f1    ret
```

# 08

---

## Training : ez\_rev

실행 : 입력값을 넣으면.. 비웃음 당한다.

```
hoodscp@LAPTOP-0F5PGRC9:~/Dreamhack/ez_rev$ ./prob  
Input: 1234  
KKKKKKKKKKKKKhoodscp@LAPTOP-0F5PGRC9:~/Dreamhack/ez_rev$ █
```

## 08

# Training : ez\_rev

정적분석(IDA) : 수많은 함수들 이후에 strcmp

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    int v3; // edx
    int v4; // ecx
    int v5; // r8d
    int v6; // r9d
    __int64 v7; // rdx
    char _0[40]; // [rsp+0h] [rbp+0h] BYREF
    unsigned __int64 vars28; // [rsp+28h] [rbp+28h]

    vars28 = __readfsqword(0x28u);
    __printf_chk(1LL, "Input: ", envp);
    __isoc99_scanf((unsigned int)"%26s", (unsigned int)_0, v3, v4, v5, v6, _0[0]);
    shift_right(_0, 3LL);
    xor_with_key(_0, off_4E50F0);
    shift_left(_0, 3LL);
    xor_with_key(_0, off_4E50F0);
    shift_right(_0, 3LL);
    if ( !strcmp("|1|GHyRrsfwxmsIrietznhIhj", _0) )
        __printf_chk(1LL, "Correct!", v7);
    else
        __printf_chk(1LL, "KKKKKKKKKKKKKK", v7);
    return 0;
}
```

# 08

## Training : ez\_rev

정적분석(IDA) : 여러 함수들.. 일일이 분석할 수도 있겠지만?

EZ\_REV\_19

```
int64 __fastcall shift_right(_BYTE *a1, int a2)
{
    __int64 result; // rax
    int v3; // r12d
    __int64 v4; // r14
    char v5; // bl
    int v6; // [rsp+Ch] [rbp-4Ch]
    __int64 v7; // [rsp+10h] [rbp-48h]
    __int64 v8; // [rsp+18h] [rbp-40h]

    result = j_strlen_ifunc();
    v6 = result;
    if ( a2 > 0 )
    {
        result = (int)result;
        v3 = 0;
        v4 = (unsigned int)(result - 2);
        v7 = (int)result - 1LL;
        v8 = (int)result - 2LL - v4;
        do
        {
            while ( 1 )
            {
                v5 = a1[v7];
                if ( v6 <= 1 )
                    break;
                ++v3;
                result = j_memmove(&a1[v7 - v4], &a1[v8], v4 + 1);
                *a1 = v5;
                if ( v3 == a2 )
                    return result;
            }
            ++v3;
            *a1 = v5;
        } __fastcall xor_with_key(__int64 a1, __int64 a2)
        {
            int v3; // r12d
            __int64 result; // rax
            int v5; // esi
            __int64 v6; // rcx

            v3 = ((__int64 (*)(void))j_strlen_ifunc)();
            result = j_strlen_ifunc(a2);
            if ( v3 > 0 )
            {
                v5 = result;
                v6 = 0LL;
                do
                {
                    *(_BYTE *)(a1 + v6) ^= *(_BYTE *)(a2 + (int)v6 % v5);
                    result = v6++;
                } while ( result != v3 - 1 );
            }
            return result;
        }
    }
}
```

## 08

Training :  
ez\_rev

동적분석(PwnDbg) : 입력

```

▶ 0x4019b9 <main+57>    call   __isoc99_scanf           <__isoc99_scanf>
    format: 0x4b600c ← 0x7c6c7c0073363225 /* '%26s' */
    vararg: 0x7fffffffdf0 → 0x4e17a8 → 0x4e69a0 (_nl_global_locale) → 0x4e18a

    0x4019be <main+62>    mov    esi, 3
    0x4019c3 <main+67>    mov    rdi, rbp
    0x4019c6 <main+70>    call   shift_right          <shift_right>

    0x4019cb <main+75>    mov    rsi, qword ptr [rip + 0xe371e] <k2>
    0x4019d2 <main+82>    mov    rdi, rbp

00:0000  rsi rbp rsp 0x7fffffffdf0 → 0x4e17a8 → 0x4e69a0 (_nl_global_locale) → 0x
01:0008  +008          0x7fffffffdf8 ← 0x0
02:0010  +010          0x7fffffffdf0 → 0x7fffffff018 → 0x40218a (__libc_start_call_
03:0018  +018          0x7fffffffdf8 → 0x455238 (_dl_non_dynamic_init+2456) ← cmp qw
04:0020  +020          0x7fffffff000 ← 0x600000000
05:0028  +028          0x7fffffff008 ← 0x56583180316cc100
06:0030  +030          0x7fffffff010 ← 0x2
07:0038  +038          0x7fffffff018 → 0x40218a (__libc_start_call_main+106) ← mov e

▶ 0      0x4019b9 main+57
1      0x40218a __libc_start_call_main+106
2      0x403a27 __libc_start_main_impl+2599
3      0x401ab5 _start+37

pwndbg>
Input: 01234567890123456789012345

```

## 08

# Training :

## ez\_rev

동적분석(PwnDbg) : shift\_right 함수 기능

```

*RAX 0xfffffffffdfe1 ← '4501234567890123456789012'
RBX 0xfffffffffe208 → 0xfffffffffe474 ← 'SHELL=/bin/bash'
*RCX 0x1a
*RDX 0x19
*RDI 0xfffffffffdfe1 ← '4501234567890123456789012'
*RSI 0xfffffffffdfe0 ← '34501234567890123456789012'
R8 0x0
R9 0x4eeba0 ← '01234567890123456789012345\n'
R10 0xfffffffffffffb8
R11 0x0
R12 0x1
R13 0xfffffffffe1f8 → 0xfffffffffe43a ← '/mnt/c/Jihoon/2024/My Own/Reversing/Dreamha
R14 0x4e17c8 (__preinit_array_start) → 0x401b80 (frame_dummy) ← endbr64
R15 0x2
RBP 0xfffffffffdfe0 ← '34501234567890123456789012'
RSP 0xfffffffffdfe0 ← '34501234567890123456789012'
*RIP 0x4019cb (main+75) ← mov rsi, qword ptr [rip + 0xe371e]
[ DISASM ]
0x4019b7 <main+55> xor    eax, eax
0x4019b9 <main+57>  call   __isoc99_scanf
<__isoc99_scanf>

0x4019be <main+62>  mov    esi, 3
0x4019c3 <main+67>  mov    rdi, rbp
0x4019c6 <main+70>  call   shift_right
<shift_right>
▶ 0x4019cb <main+75>  mov    rsi, qword ptr [rip + 0xe371e] <k2>
0x4019d2 <main+82>  mov    rdi, rbp
0x4019d5 <main+85>  call   xor_with_key
<xor_with_key>

0x4019da <main+90>  mov    esi, 3
0x4019df <main+95>  mov    rdi, rbp
0x4019e2 <main+98>  call   shift_left
<shift_left>

```

# 08

## Training : ez\_rev

동적분석(PwnDbg) : xor\_key 함수 기능

xor 연산 부분에 BP 후 분석 :

71, 6b, 73, 72, 6b, 71, 73 순으로 xor 연산

```
*RAX 0x71
RBX 0x7fffffffdfc0 ← '34501234567890123456789012'
RCX 0x0
RDX 0x0
RDI 0x19
RSI 0x7
R8 0x0
R9 0x4eeba0 ← '01234567890123456789012345\n'
R10 0xfffffffffffffb8
R11 0x0
R12 0x1a
R13 0x7fffffffelf8 → 0x7fffffff43a ← '/mnt/c/Jihoon/2024/My Own/Reversing/Dreamhack/ez_rev'
R14 0x4e17c8 (__preinit_array_start) → 0x401b80 (frame_dummy) ← endbr64
R15 0x2
RBP 0x4b6042 ← 0x73716b72736b71 /* 'qksrkqs' */
RSP 0x7fffffffdfc0 → 0x7fffffff208 → 0x7fffffff474 ← 'SHELL=/bin/bash'
*RIP 0x401d3d (xor_with_key+61) ← xor byte ptr [rbx + rcx], al [ DISASM / x86-64 / ]
```

```
0x401d30 <xor_with_key+48>    mov    eax, ecx
0x401d32 <xor_with_key+50>    cdq
0x401d33 <xor_with_key+51>    idiv   esi
0x401d35 <xor_with_key+53>    movsxd rdx, edx
0x401d38 <xor_with_key+56>    movzx  eax, byte ptr [rbp + rdx]
▶ 0x401d3d <xor_with_key+61>    xor    byte ptr [rbx + rcx], al
0x401d40 <xor_with_key+64>    mov    rax, rcx
0x401d43 <xor_with_key+67>    add    rcx, 1
0x401d47 <xor_with_key+71>    cmp    rax, rdi
0x401d4a <xor_with_key+74>    jne    xor_with_key+48
↓
0x401d30 <xor_with_key+48>    mov    eax, ecx [ xor_with_key+48 ]
```

```
[ STACK ]
```

00:0000	rsp 0x7fffffffdfc0 → 0x7fffffff208 → 0x7fffffff474 ← 'SHELL=/bin/bash'
01:0008	0x7fffffffdfc8 → 0x7fffffffdfc0 ← '34501234567890123456789012'
02:0010	0x7fffffffdfd0 ← 0x1
03:0018	0x7fffffffdfd8 → 0x4019da (main+90) ← mov esi, 3
04:0020	rbx 0x7fffffffdfc0 ← '34501234567890123456789012'
05:0028	0x7fffffffdfc8 ← '567890123456789012'
06:0030	0x7fffffffdfff0 ← '3456789012'
07:0038	0x7fffffffdf8 ← 0x3231 /* '12' */

```
[ BACKTRACE ]
```

- ▶ 0 0x401d3d xor\_with\_key+61
- 1 0x4019da main+90
- 2 0x40218a \_\_libc\_start\_call\_main+106
- 3 0x403a27 \_\_libc\_start\_main\_impl+2599
- 4 0x401ab5 \_start+37

```
pwndbg> b
Breakpoint 3 at 0x401d3d
pwndbg> 
```

# 08

## Training : ez\_rev

동적분석(PwnDbg) : shift\_left 함수 기능

```
0x401d4d <xor_with_key+77>    pop   rbp
0x401d4e <xor_with_key+78>    pop   r12
0x401d50 <xor_with_key+80>    ret
                                ↓
0x4019da <main+90>           mov    esi, 3
0x4019df <main+95>           mov    rdi, rbp
► 0x4019e2 <main+98>           call   shift_left          <shift_left>
                                rdi: 0xfffffffffffffe0 ← 'B_FBFZC@E^EESHC@Y@F^GDIRCCY'
                                rsi: 0x3
                                rdx: 0x4
                                rcx: 0x1a

0x4019e7 <main+103>          mov    rsi, qword ptr [rip + 0xe3702] <k2>
0x4019ee <main+110>          mov    rdi, rbp
0x4019f1 <main+113>          call   xor_with_key         <xor_with_key>

0x4019f6 <main+118>          mov    esi, 3
0x4019fb <main+123>          mov    rdi, rbp
```

```
0x4019da <main+90>          mov    esi, 3
0x4019df <main+95>          mov    rdi, rbp
0x4019e2 <main+98>          call   shift_left          <shift_left>

0x4019e7 <main+103>          mov    rsi, qword ptr [rip + 0xe3702] <k2>
0x4019ee <main+110>          mov    rdi, rbp
► 0x4019f1 <main+113>          call   xor_with_key         <xor_with_key>
                                rdi: 0xfffffffffffffe0 ← 'BZC@E^EESHC@Y@F^GDIRCCYB_F'
                                rsi: 0x4b6042 ← 0x73716b72736b71 /* 'qksrkqs' */
                                rdx: 0x19
                                rcx: 0x1a

0x4019f6 <main+118>          mov    esi, 3
0x4019fb <main+123>          mov    rdi, rbp
0x4019fe <main+126>          call   shift_right         <shift_right>

0x401a03 <main+131>          mov    ecx, 0x1a
0x401a08 <main+136>          lea    rsi, [rip + 0xb4602]
```

## 08

Training :  
ez\_rev

역연산 코드 작성

```

shift_right(_0, 3LL);
xor_with_key(_0, off_4E50F0);
shift_left(_0, 3LL);
xor_with_key(_0, off_4E50F0);
shift_right(_0, 3LL);
if ( !strcmp("1|GHyRrsfwxmsIrietznhIhj", _0) )
    _printf_chk(1LL, "Correct!", v7);

```

```

char arr[] = {"GHyRrsfwxmsIrietznhIhj|1|"}; // shift 후 첫번째 xor 를 위한 값

int xor_key[] = {0x71,0x6b,0x73,0x72,0x6b,0x71,0x73};
// xor 에 사용되는 키

for (int i=0; i < 25; i++)
{
    printf("0x%02x, ", arr[i] ^ xor_key[i%7]);
}
0x36, 0x23, 0xa, 0x20, 0x19, 0x2, 0x15, 0x6, 0x13, 0x1e, 0x1, 0x22, 0x3, 0x1a, 0x14, 0x1f, 0x9, 0x1c, 0x3, 0x38, 0x
1b, 0x1b, 0x17, 0x1f, 0xe,

```

## 08

---

# Training : ez\_rev

역연산 코드 작성

```
shift_right(_0, 3LL);
xor_with_key(_0, off_4E50F0);
shift_left(_0, 3LL);
xor_with_key(_0, off_4E50F0);
shift_right(_0, 3LL);
if ( !strcmp("|1|GHyRrsfwxmsIrietznhIhj", _0) )
    _printf_chk(1LL, "Correct!", v7);
```

```
int arrr[] = { 0x17, 0x1f, 0xe, 0x36, 0x23, 0xa, 0x20, 0x19, 0x2, 0x15, 0x6, 0x13, 0x1e, 0x1,
               0x22, 0x3, 0x1a, 0x14, 0x1f, 0x9, 0x1c, 0x3, 0x38, 0x1b, 0x1b };
// shift, xor, shift 후 xor 를 위한 값

for (int i=0; i < 25; i++)
{
    printf("%c", arrr[i] ^ xor_key[i%7]);
}
```

ft}DH{shift,shift,shift  
17828378174803250473256297

Thank you!

# Q & A

PwnDbg 는 거들뿐.. IDA가 최고다..