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 1](#_Toc14288)

[2实验目的 1](#_Toc16974)

[3实验原理 1](#_Toc4054)

[3.1printf函数调用实现原理 1](#_Toc13989)

[3.2漏洞形成原因 1](#_Toc20657)

[4 实验步骤 2](#_Toc24850)

[4.1关闭内存地址随机化 2](#_Toc13975)

[4.2任意内存读操作 3](#_Toc31617)

[4.3写入地址 3](#_Toc4102)

[4.4修改a的值 5](#_Toc19867)

[4.5调整a的值为2023 5](#_Toc29859)

[5 实验心得体会 8](#_Toc11425)

# 1实验名称

格式化字符串漏洞

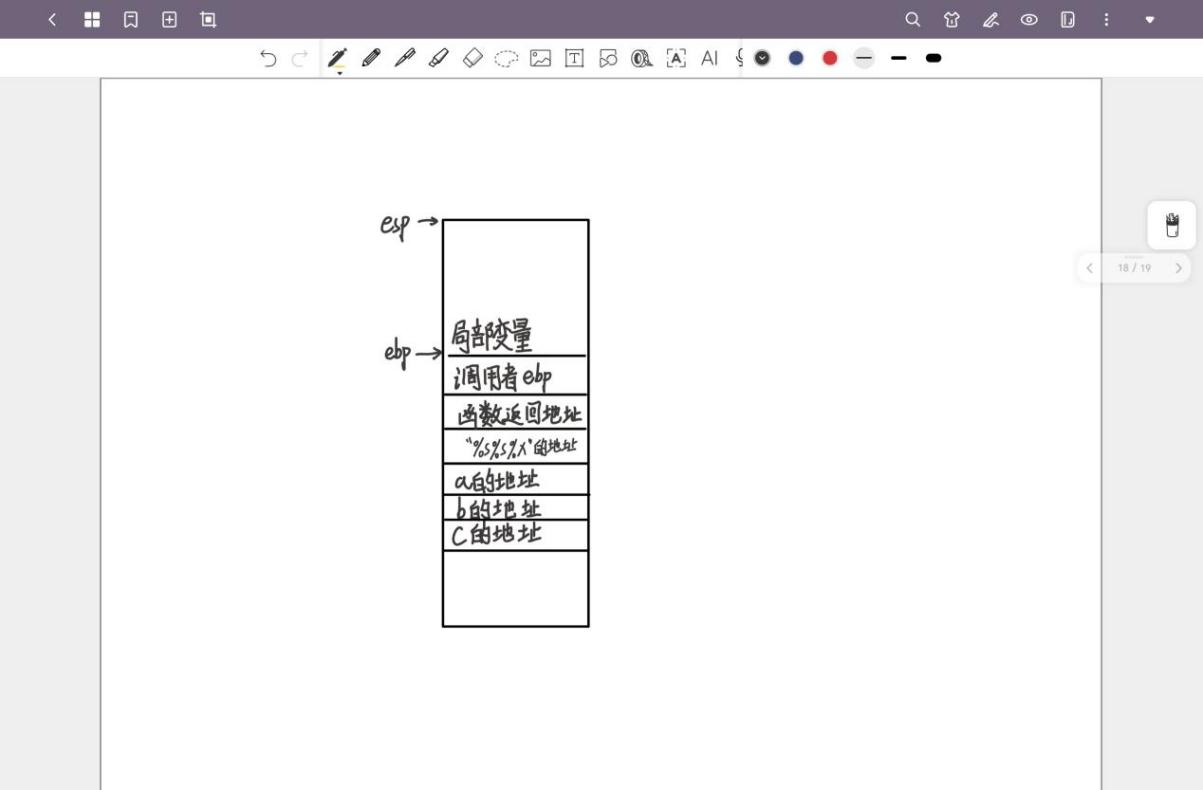
# 2实验目的

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

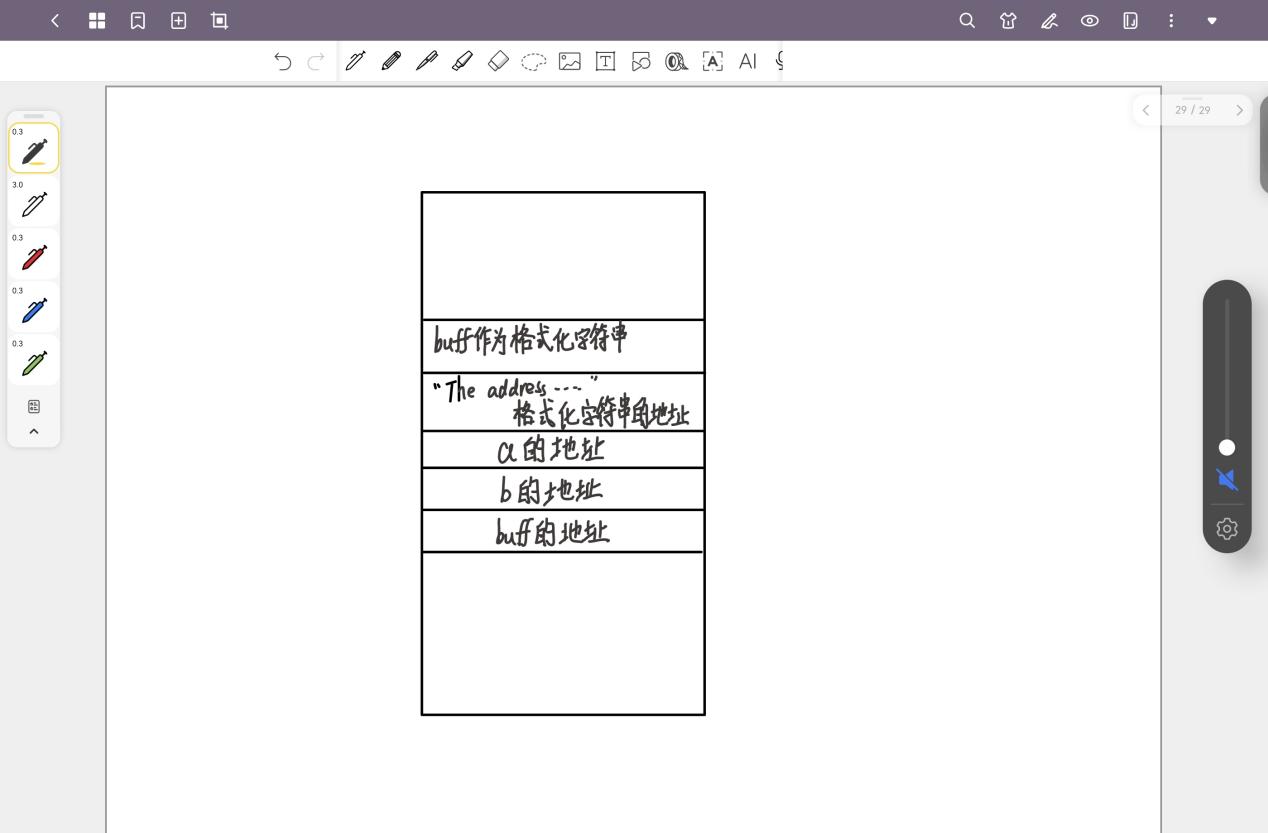
在规范的printf函数使用方法中，如果要输出某一个数组中的值需要写成printf(“%s”,a)的形式即若存在格式化字符串则应当存在等量的变量名，但是也存在有printf(a)的写法，该种写法在底层执行的操作是首先将数组的首地址入栈，然后调用printf函数。

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。

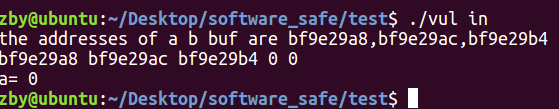
而且如果采用这种写法，在数组a中写入格式化字符串例如%x，则会输出当前格式化字符串地址之后的内存中的值，或者在a中写入%n会直接在对应的内存中进行写入操作，将函数输出的字符个数写入到指向的位置中，借此可以实现对内存中任意地址的读写操作。

# 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

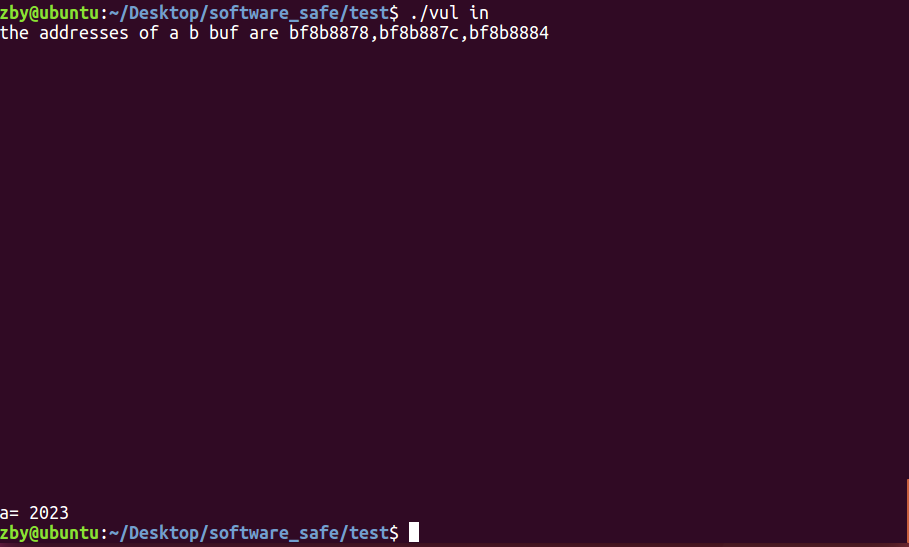


在输入中构造三个%x验证这一堆栈结构得到的结果如下图所示



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

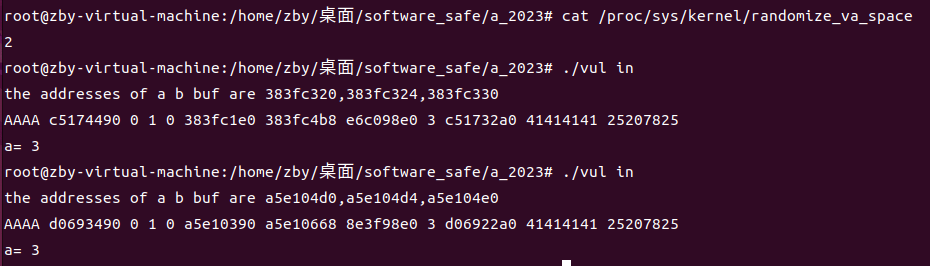
# 5 题目改进

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

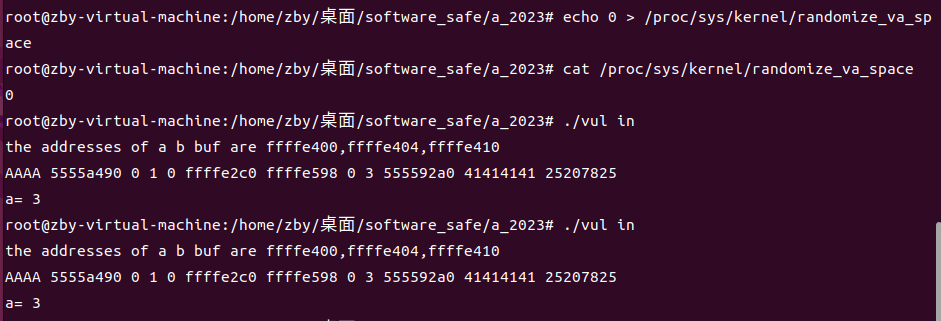
## 5.1关闭内存地址随机化

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。

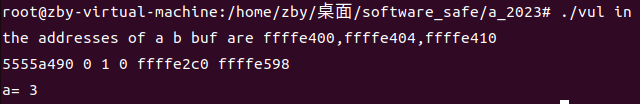


将系统的内存地址随机化关闭之后运行的结果如下图所示，可以发现每一次程序运行过后变量的地址都没有发生改变。

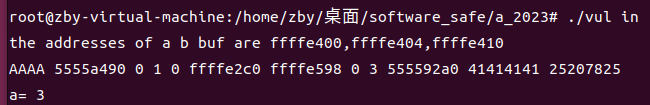


## 5.2任意内存读操作

在数组buff中放入数个%x运行的结果如下图所示。



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。



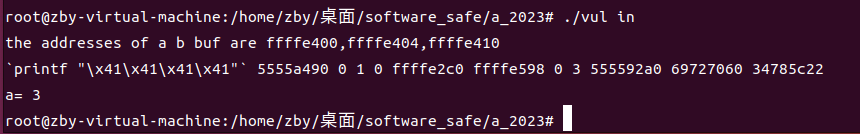
可以发现AAAA的ascii码写入了第10个偏移的位置。

## 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



得到运行结果如下图所示

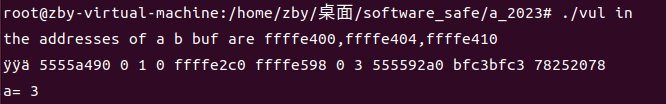


观察发现程序并未识别printf命令，不能完成对应内存地址的写操作。推测原因可能是printf命令只有在命令行中使用反引号转义符才能够生效。

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

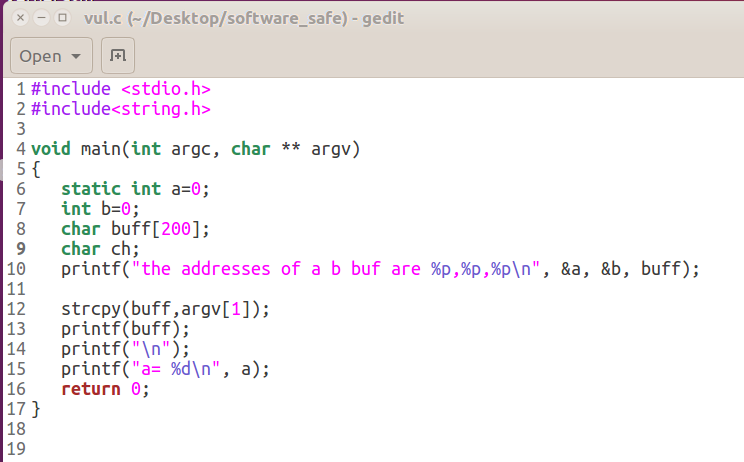


函数运行结果如下图所示，并不能实现将数组的开始元素写为a变量的地址。

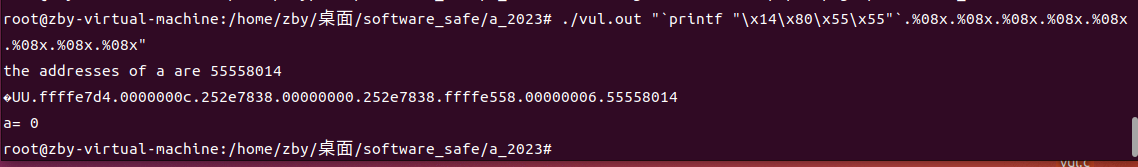


推测可能的原因是Linux系统中不能正常识别拓展ascii码。

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



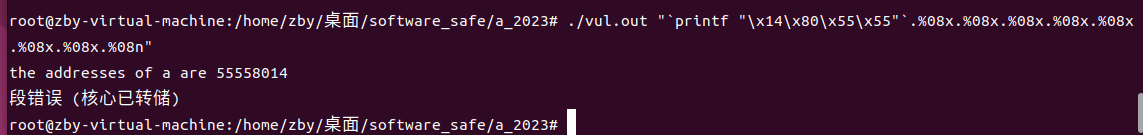
将输入构造成"`printf "\x2c\xa0\x04\x08"`.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x"的形式，得到的结果如下图所示



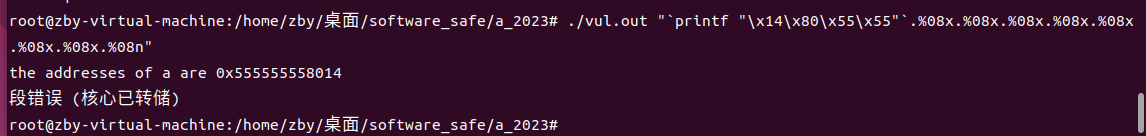
能够将数组第0项的位置修改为a所对应的地址。

## 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

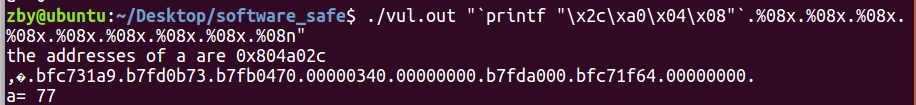


出现段错误报错，将打印a的地址的指令修改为%p观察a的地址得到的结果如下图所示



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

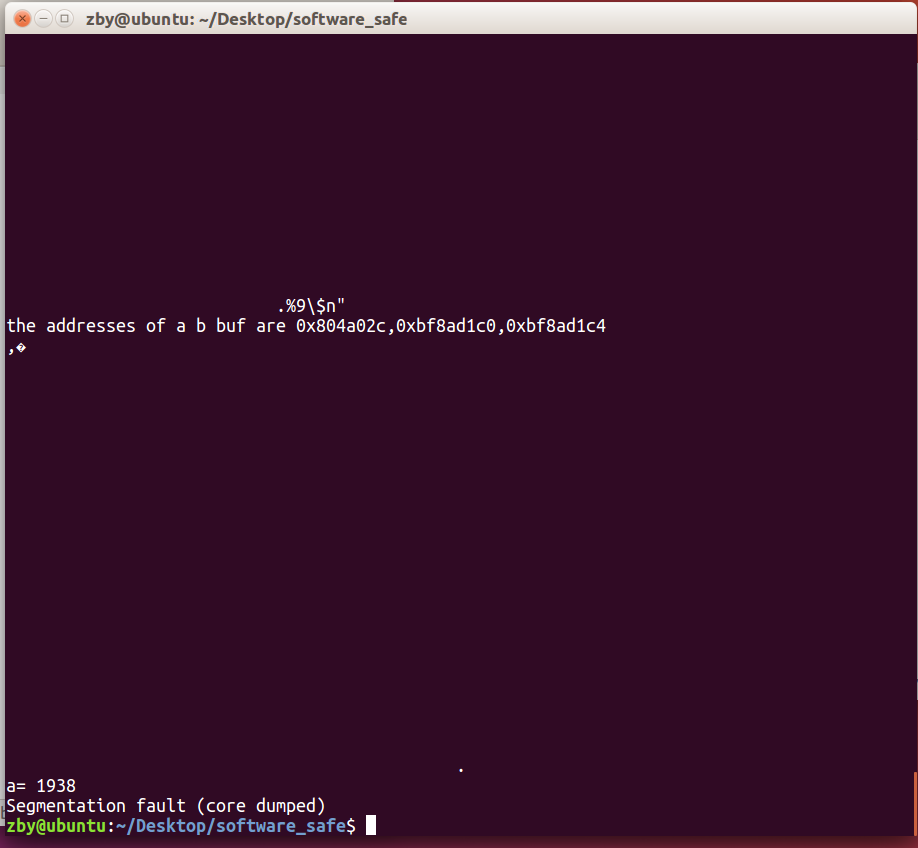


## 5.5调整a的值为2023

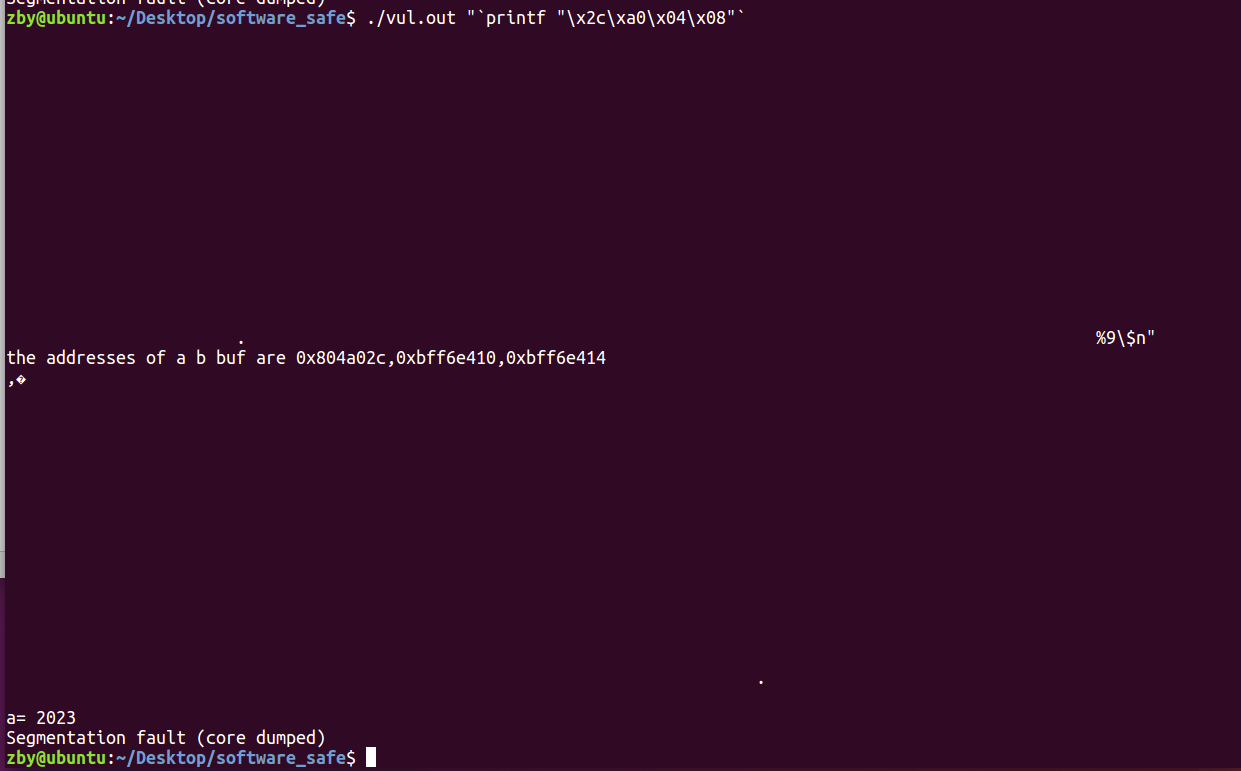
调整a的值，在读取命令%x前补入空格后得到的结果如下图所示，



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。