## 实验五：内存分配、缓冲区溢出实验

**实验5.1：**

1. 实验目的
2. 掌握基本的函数调用规则。
3. 掌握局部变量在函数栈上的存放原则。
4. 理解数组越界的危害。
5. 实验内容
6. 使用命令gcc overflow1.c -fno-stack-protector –o overflow1编译overflow1.c，overflow1.c代码如下：

#include <stdio.h>

#include <stdlib.h>

void why\_here(void)

{

printf("why u r here?!\n");

exit(0);

}

int foo(){

int buff[1];

buff[2] = (int)why\_here;

return 0;

}

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

{

foo();

return 0;

}

1. 实验要求
2. 运行程序overflow1，观察出现的实验现象。
3. 反汇编源程序，根据汇编代码回答下面的问题：
4. 假设内存地址0xbfffef30-0xbfffef33保存foo函数的返回地址，请推断出数组buf的开始地址。

1. 根据（1）的结果，推断出变量buf[2]在内存中的开始地址。

1. 根据2的分析结果，解释出现实验现象的原因。

**实验5.2：**

1. 实验目的
2. 掌握函数调用规则。
3. 理解标准输入函数gets的缺陷。
4. 掌握缓冲区溢出的基本原理。
5. 了解防止缓冲区溢出的防御方法。

二、实验内容

1、使用命令gcc overflow2.c -fno-stack-protector –o overflow2编译overflow2.c，overflow2.c代码如下：

#include <stdio.h>

void doit(void)

{

char buf[8];

gets(buf);

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

}

int main(void)

{

printf("So... The End...\n");

doit();

printf("or... maybe not?\n");

return 0;

}

三、实验要求

1. 运行overflow2，尝试输入任意个数的字符，如字符’a’，猜测需要连续输入几个字符才能产生segmentation fault？

2. 反汇编源程序，验证1中的猜测。假设数组buf的开始地址为0xbfffef30，输入N个字符’a’后出现segmentation fault。查阅gets函数的资料，如man。请写出输入N+6个字符后，内存地址范围[0xbfffef30, 0xbfffef30+N+5]的存放情况。

3.根据2的假设，请写出未被污染前，地址范围0xbfffef30+N到0xbfffef30+N+7原始存放的数据。

4.将源程序中的gets函数替换为fgets函数，然后重新实验。比较fgets函数和gets函数的异同，简要说明采用fgets函数为何无法完成上述攻击。

5. 查阅相关资料，简要阐述抵御缓冲区溢出的两种防御方法（栈随机化和栈破坏检测）的主要思想。

Hints：使用gdb查看栈的存储情况，跟踪程序执行过程中ebp寄存器的变化，以及观察程序printf的输出等，使用gdb TUI模式的layout regs命令可以实时监测各个寄存器的变化，在使用gdb前先把程序分析清楚，gdb只是最后的救命稻草！

**实验5.3：**

一、实验目的

1. 掌握不同机器平台不同类型整数的大小限制。
2. 理解整数溢出的危害。
3. 了解常用的抵御整数溢出的防御方法。

二、实验内容

1. 使用命令gcc –o overflow3 overflow3.c 编译overflow3.c文件。

#include <stdio.h>

#include <stdlib.h>

typedef unsigned int uint;

int\* foo(uint len,int \*array)

{

int \*buf, i;

buf = malloc(len\*sizeof(int));

printf("malloc %lu bytes\n",len\*sizeof(int));

if(buf == NULL) {

return NULL;

}

printf("loop time: %d[0x%x]\n",len,len);

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

{

buf[i] = array[i];

}

return buf;

}

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

int array[5] = {1, 2, 3, 4, 5};

foo(atoi(argv[1]), array);

return 0;

}

三、实验要求

1. 分别以参数./main 20和./main 1073741824运行程序，观察出现的现象。

2. 回答32位平台size\_t类型的范围，计算出两次分配内存大小的不同。

3. 解释出现的实验现象。

4. 尝试malloc 0个字节，并分析现有Linux允许malloc 0个字节的合理性（开放题）。