实验五：内存分配、缓冲区溢出实验

实验 5.1：

一、实验目的

1. 掌握基本的函数调用规则。
2. 掌握局部变量在函数栈上的存放原则。
3. 理解数组越界的危害。

二、实验内容

1、 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;
}
```

三、实验要求

1. 运行程序 overflow1，观察出现的实验现象。

2. 反汇编源程序，根据汇编代码回答下面的问题：
   （1） 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。

   （2） 根据（1）的结果，推断出变量 buf[2]在内存中的开始地址。

3. 根据 2 的分析结果，解释出现实验现象的原因。

实验 5.2：

一、 实验目的

1. 掌握函数调用规则。
2. 理解标准输入函数 gets 的缺陷。
3. 掌握缓冲区溢出的基本原理。
4. 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。

4. 解释出现的实验现象。

5. 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）。