课程设计

班级：191211

姓名：孙鹤轩

学号：20211000156

设计内容：

1 二进制程序逆向工程

2 缓冲区溢出攻击

3 程序的链接

设计目标：

1 加深对计算机系统的理解和掌握：程序的机器级表示、汇编与反汇编、二进制程序分析与

调试、逆向工程；函数调用规则、栈结构、缓冲区溢出攻击原理、方法与防范；程序链接中

符号解析、重定位等基本概念、位置无关代码和 ELF 文件的基本组成；

2 从程序员角度认识计算机系统，分析高级语言对应的机器行为及其对程序执行结果和性能

的影响，解决计算机系统设计、程序开发过程中的关键问题；

3 掌握计算机系统思维，理解高级语言中数据、运算、过程调用和 I/O 操作等在计算机系统

中的实现方法，将程序设计、汇编语言、系统结构、操作系统、编译链接中的重要概念贯穿

起来；

4 掌握各种开源的编译调试工具，能够对分析优化程序设计，提高在代码调试、性能提升、

软件移植和鲁棒性等方面的能力。

设计任务：

1 学习 MOOC 内容

https://www.icourse163.org/learn/NJU-1449521162

第五周 二进制程序逆向工程

第 1 讲 二进制炸弹实验：概述

第 2 讲 二进制炸弹实验：字符串比较

第 3 讲 二进制炸弹实验：浮点数表示

第 4 讲 二进制炸弹实验：课后实验

https://www.icourse163.org/learn/NJU-1449521162

第六周 缓冲区溢出攻击

第 1 讲 缓冲区溢出攻击实验：概述

第 2 讲 缓冲区溢出攻击实验：目标程序与辅助工具

第 3 讲 缓冲区溢出攻击实验：Level 0

第 4 讲 缓冲区溢出攻击实验：Level 1 及课后实验

https://www.icourse163.org/learn/NJU-1449521162

第七周 程序的链接

第 1 讲 链接与 ELF 实验：概述

第 2 讲 链接与 ELF 实验：静态数据与 ELF 数据节

第 3 讲 链接与 ELF 实验：指令与 ELF 代码节及课后实验

2.1二进制逆向工程

2.1.1循环（phase2）

实验结果及分析如下：

Phase2的汇编代码如下：

080494ee <phase\_2>:

80494ee: 55 push %ebp

80494ef: 89 e5 mov %esp,%ebp

80494f1: 53 push %ebx

80494f2: 83 ec 34 sub $0x34,%esp

80494f5: 83 ec 04 sub $0x4,%esp

80494f8: 6a 09 push $0x9

80494fa: 8d 45 d0 lea -0x30(%ebp),%eax

80494fd: 50 push %eax

80494fe: ff 75 08 pushl 0x8(%ebp)

8049501: e8 a1 06 00 00 call 8049ba7 <read\_n\_numbers>

8049506: 83 c4 10 add $0x10,%esp

8049509: 85 c0 test %eax,%eax

804950b: 75 07 jne 8049514 <phase\_2+0x26>

804950d: b8 00 00 00 00 mov $0x0,%eax

8049512: eb 60 jmp 8049574 <phase\_2+0x86>

8049514: 8b 45 d0 mov -0x30(%ebp),%eax

8049517: 3d 86 00 00 00 cmp $0x86,%eax

804951c: 74 0c je 804952a <phase\_2+0x3c>

804951e: e8 a6 09 00 00 call 8049ec9 <explode\_bomb>

8049523: b8 00 00 00 00 mov $0x0,%eax

8049528: eb 4a jmp 8049574 <phase\_2+0x86>

804952a: c7 45 f4 01 00 00 00 movl $0x1,-0xc(%ebp)

8049531: eb 36 jmp 8049569 <phase\_2+0x7b>

8049533: 8b 45 f4 mov -0xc(%ebp),%eax

8049536: 8b 54 85 d0 mov -0x30(%ebp,%eax,4),%edx

804953a: 8b 45 f4 mov -0xc(%ebp),%eax

804953d: 83 e8 01 sub $0x1,%eax

8049540: 8b 4c 85 d0 mov -0x30(%ebp,%eax,4),%ecx

8049544: 8b 5d f4 mov -0xc(%ebp),%ebx

8049547: b8 00 00 00 00 mov $0x0,%eax

804954c: 29 d8 sub %ebx,%eax

804954e: 01 c0 add %eax,%eax

8049550: 01 c8 add %ecx,%eax

8049552: 83 c0 01 add $0x1,%eax

8049555: 39 c2 cmp %eax,%edx

8049557: 74 0c je 8049565 <phase\_2+0x77>

8049559: e8 6b 09 00 00 call 8049ec9 <explode\_bomb>

804955e: b8 00 00 00 00 mov $0x0,%eax

8049563: eb 0f jmp 8049574 <phase\_2+0x86>

8049565: 83 45 f4 01 addl $0x1,-0xc(%ebp)

8049569: 83 7d f4 08 cmpl $0x8,-0xc(%ebp)

804956d: 7e c4 jle 8049533 <phase\_2+0x45>

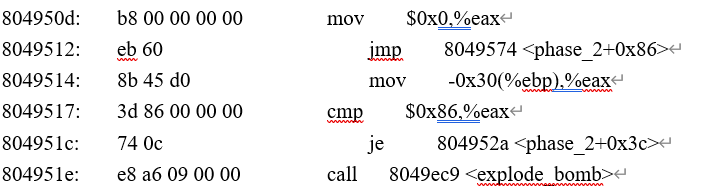
804956f: b8 01 00 00 00 mov $0x1,%eax

8049574: 8b 5d fc mov -0x4(%ebp),%ebx

8049577: c9 leave

8049578: c3 ret

首先我们找到调用bomb炸弹的地方，第一处为：



前面的指令中我们发现是将常熟0x86与eax的值进行了比较，如果不相等就继续往下执行调用bomb炸弹；推测其第一个输入的数应为0x86=134；接着开始了循环代码：

804952a: c7 45 f4 01 00 00 00 movl $0x1,-0xc(%ebp)

8049531: eb 36 jmp 8049569 <phase\_2+0x7b>

8049533: 8b 45 f4 mov -0xc(%ebp),%eax

8049536: 8b 54 85 d0 mov -0x30(%ebp,%eax,4),%edx

804953a: 8b 45 f4 mov -0xc(%ebp),%eax

804953d: 83 e8 01 sub $0x1,%eax

8049540: 8b 4c 85 d0 mov -0x30(%ebp,%eax,4),%ecx

8049544: 8b 5d f4 mov -0xc(%ebp),%ebx

8049547: b8 00 00 00 00 mov $0x0,%eax

804954c: 29 d8 sub %ebx,%eax

804954e: 01 c0 add %eax,%eax

8049550: 01 c8 add %ecx,%eax

8049552: 83 c0 01 add $0x1,%eax

8049555: 39 c2 cmp %eax,%edx

8049557: 74 0c je 8049565 <phase\_2+0x77>

8049559: e8 6b 09 00 00 call 8049ec9 <explode\_bomb>

804955e: b8 00 00 00 00 mov $0x0,%eax

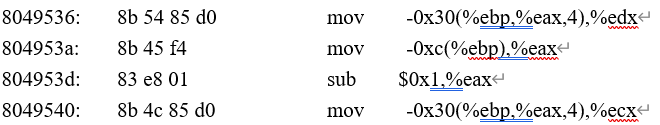
8049563: eb 0f jmp 8049574 <phase\_2+0x86>

8049565: 83 45 f4 01 addl $0x1,-0xc(%ebp)

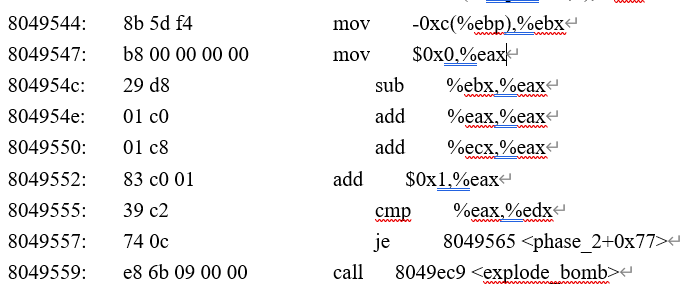
8049569: 83 7d f4 08 cmpl $0x8,-0xc(%ebp)

804956d: 7e c4 jle 8049533 <phase\_2+0x45>

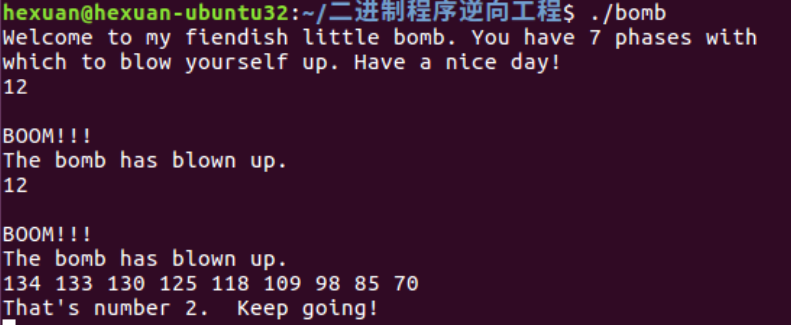
首先在8049569处比较了常数0x8与-0xc(%ebp)处的值，并且在上一步执行了addl $0x1,-0xc(%ebp)加一操作，所以-0xc(%ebp)处的值，该循环执行9次，那么一共应读取了9个数字



从这几句和前面的入栈情况代码可以看出edx存放的是a[i]，ecx处存放的是a[i-1]

从这里推测出循环内的逻辑应为a[i]=a[i-1]-2\*i+1，其中a[0]即是我们输入的第一个数134，所以我们可以递推出后面的数，整个数组内的数应为[134,133,130,125,118,109,98,85,70]

实验结果如下：



2.1.2条件/分支（phase3）实验结果及分析如下：

Phase3的汇编代码如下：

08049579 <phase\_3>:

8049579: 55 push %ebp

804957a: 89 e5 mov %esp,%ebp

804957c: 83 ec 18 sub $0x18,%esp

804957f: c7 45 f4 00 00 00 00 movl $0x0,-0xc(%ebp)

8049586: c7 45 f0 00 00 00 00 movl $0x0,-0x10(%ebp)

804958d: 8d 45 e8 lea -0x18(%ebp),%eax

8049590: 50 push %eax

8049591: 8d 45 ec lea -0x14(%ebp),%eax

8049594: 50 push %eax

8049595: 68 0f a2 04 08 push $0x804a20f

804959a: ff 75 08 pushl 0x8(%ebp)

804959d: e8 2e fb ff ff call 80490d0 <\_\_isoc99\_sscanf@plt>

80495a2: 83 c4 10 add $0x10,%esp

80495a5: 89 45 f0 mov %eax,-0x10(%ebp)

80495a8: 83 7d f0 01 cmpl $0x1,-0x10(%ebp)

80495ac: 7f 0f jg 80495bd <phase\_3+0x44>

80495ae: e8 16 09 00 00 call 8049ec9 <explode\_bomb>

80495b3: b8 00 00 00 00 mov $0x0,%eax

80495b8: e9 95 00 00 00 jmp 8049652 <phase\_3+0xd9>

80495bd: 8b 45 ec mov -0x14(%ebp),%eax

80495c0: 2d fe 00 00 00 sub $0xfe,%eax

80495c5: 83 f8 09 cmp $0x9,%eax

80495c8: 77 63 ja 804962d <phase\_3+0xb4>

80495ca: 8b 04 85 18 a2 04 08 mov 0x804a218(,%eax,4),%eax

80495d1: ff e0 jmp \*%eax

80495d3: c7 45 f4 bd 00 00 00 movl $0xbd,-0xc(%ebp)

80495da: eb 5d jmp 8049639 <phase\_3+0xc0>

80495dc: c7 45 f4 bd 00 00 00 movl $0xbd,-0xc(%ebp)

80495e3: eb 54 jmp 8049639 <phase\_3+0xc0>

80495e5: c7 45 f4 e5 03 00 00 movl $0x3e5,-0xc(%ebp)

80495ec: eb 4b jmp 8049639 <phase\_3+0xc0>

80495ee: c7 45 f4 bd 00 00 00 movl $0xbd,-0xc(%ebp)

80495f5: eb 42 jmp 8049639 <phase\_3+0xc0>

80495f7: c7 45 f4 e5 03 00 00 movl $0x3e5,-0xc(%ebp)

80495fe: eb 39 jmp 8049639 <phase\_3+0xc0>

8049600: c7 45 f4 bd 00 00 00 movl $0xbd,-0xc(%ebp)

8049607: eb 30 jmp 8049639 <phase\_3+0xc0>

8049609: c7 45 f4 e5 03 00 00 movl $0x3e5,-0xc(%ebp)

8049610: eb 27 jmp 8049639 <phase\_3+0xc0>

8049612: c7 45 f4 e5 03 00 00 movl $0x3e5,-0xc(%ebp)

8049619: eb 1e jmp 8049639 <phase\_3+0xc0>

804961b: c7 45 f4 bd 00 00 00 movl $0xbd,-0xc(%ebp)

8049622: eb 15 jmp 8049639 <phase\_3+0xc0>

8049624: c7 45 f4 e5 03 00 00 movl $0x3e5,-0xc(%ebp)

804962b: eb 0c jmp 8049639 <phase\_3+0xc0>

804962d: e8 97 08 00 00 call 8049ec9 <explode\_bomb>

8049632: b8 00 00 00 00 mov $0x0,%eax

8049637: eb 19 jmp 8049652 <phase\_3+0xd9>

8049639: 8b 45 e8 mov -0x18(%ebp),%eax

804963c: 39 45 f4 cmp %eax,-0xc(%ebp)

804963f: 74 0c je 804964d <phase\_3+0xd4>

8049641: e8 83 08 00 00 call 8049ec9 <explode\_bomb>

8049646: b8 00 00 00 00 mov $0x0,%eax

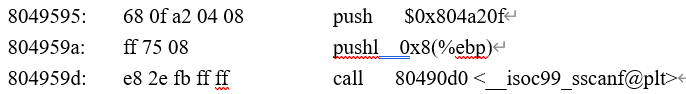
804964b: eb 05 jmp 8049652 <phase\_3+0xd9>

804964d: b8 01 00 00 00 mov $0x1,%eax

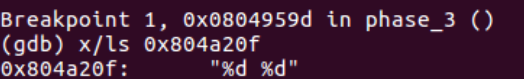
8049652: c9 leave

8049653: c3 ret

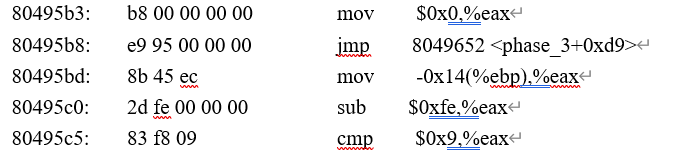
从汇编代码中看到了很多jmp指令推测其应该是switch语句



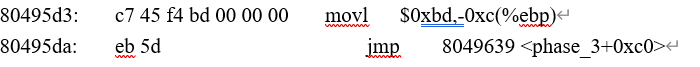
这里调用了sscanf函数，该函数第二个参数为格式化字符串，使用gdb查看他的值，如下：



所以推测应该是输入了两个整数，我们来看switch语句体中的case条件



从这里看到将eax-0xfe的值与0x9比较，0xfe= 254，推测case的范围为254-254+9，即254-263

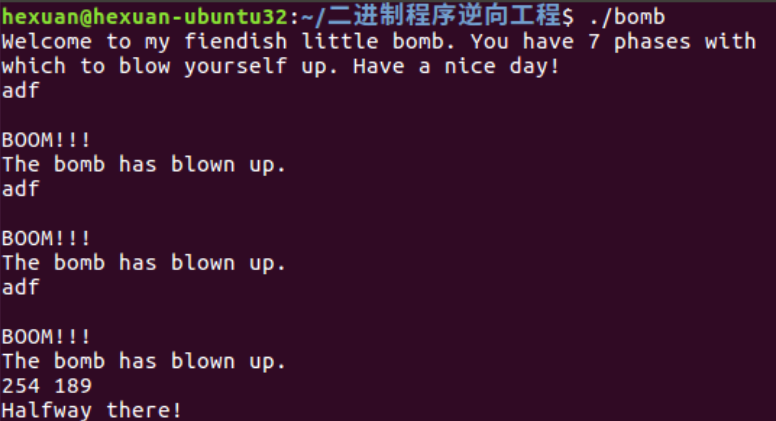


这里可以看出第一个比较的值为0xbd= 189，所以第一个成立条件可以是输入254,189

经过同样的分析得到所有的答案可以为：

(254-255) 189 || (256-258) 997 || 259 189 || (260-261) 997 || 262 189 || 263 997

实验结果如下：



2.2 缓冲区溢出攻击

2.1 bang（level2）

实验结果及分析：

任务目标为嗲用bang函数，bang函数的汇编代码如下：

08049453 <bang>:

8049453: 55 push %ebp

8049454: 89 e5 mov %esp,%ebp

8049456: 83 ec 08 sub $0x8,%esp

8049459: a1 a8 d1 04 08 mov 0x804d1a8,%eax

804945e: 89 c2 mov %eax,%edx

8049460: a1 a0 d1 04 08 mov 0x804d1a0,%eax

8049465: 39 c2 cmp %eax,%edx

8049467: 75 25 jne 804948e <bang+0x3b>

8049469: a1 a8 d1 04 08 mov 0x804d1a8,%eax

804946e: 83 ec 08 sub $0x8,%esp

8049471: 50 push %eax

8049472: 68 a0 b0 04 08 push $0x804b0a0

8049477: e8 e4 fb ff ff call 8049060 <printf@plt>

804947c: 83 c4 10 add $0x10,%esp

804947f: 83 ec 0c sub $0xc,%esp

8049482: 6a 02 push $0x2

8049484: e8 32 09 00 00 call 8049dbb <validate>

8049489: 83 c4 10 add $0x10,%esp

804948c: eb 16 jmp 80494a4 <bang+0x51>

804948e: a1 a8 d1 04 08 mov 0x804d1a8,%eax

8049493: 83 ec 08 sub $0x8,%esp

8049496: 50 push %eax

8049497: 68 c5 b0 04 08 push $0x804b0c5

804949c: e8 bf fb ff ff call 8049060 <printf@plt>

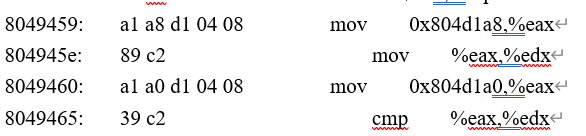
80494a1: 83 c4 10 add $0x10,%esp

80494a4: 83 ec 0c sub $0xc,%esp

80494a7: 6a 00 push $0x0

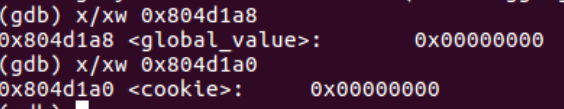
80494a9: e8 82 fc ff ff call 8049130 <exit@plt>

不同于前两关，我们并不能修改全局变量的值，所以我们应该在buf数组中写入一段汇编代码来进行更改值操作。首先我们确定全局变量的存储地址



这段代码将两个地址处的值进行了比较，所以应该一个为全局变量一个为我们的cookie值

通过gdb调试查看两个地址处的内容，结果如下：



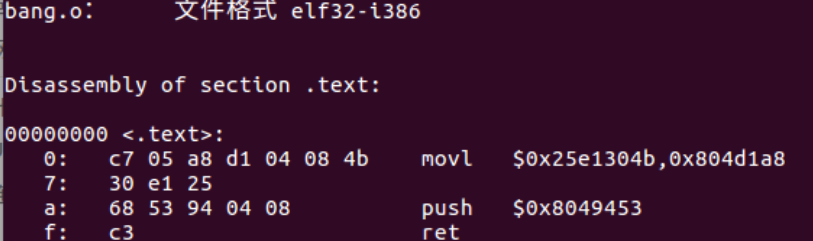
所以0x804d1a8为全局变量的存储地址，另一个为cookie的存储地址，所以我们编写如下汇编代码：

movl $0x25e1304b,0x804d1a8//将cookie值存储到全局变量的地址

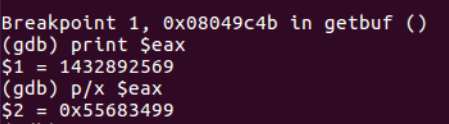
push $0x08049453//将bang函数首地址入栈

ret

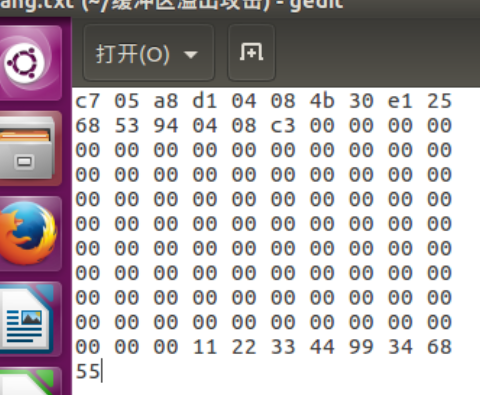
将改代码汇编再反汇编后结果如下：



在Gets函数前设置断点，通过gdb找出buf的首地址：



根据机器代码和buf首地址构造如下字符串：



实验成功！