

**实验报告**



**题目： 缓冲区溢出攻击实验**

**班 级： 2020211302**

**学 号： 2020211262**

**姓 名： 鄭毓恒**

**学 院： 计算机学院**

**2021年 12 月 4 日**

一、实验目的  
1. C语言程序的机器级表示。  
2. 掌握GDB调试器的用法。  
3. C编译器生成的x86-64机器代码，理解不同控制结构生成的基本指令模式，过程的实现。

4. 掌握两种缓冲区攻击方法，进一步理解软件漏洞的危害。

1. 实验环境
2. SecureCRT（10.120.11.12）
3. Linux
4. Objdump命令反汇编
5. GDB调试工具
6. 积分榜（http://10.120.11.13:19310/scoreboard）

**报告邮寄（最迟时间：2020年11月29日晚23：59）：**

**大一班（1-4班）：**clavicle@bupt.edu.cn

三、实验内容

登录bupt1服务器，在home目录下可以找到一个targetn.tar文件，解压后得到如下文件：

README.txt；

ctarget；

rtarget；

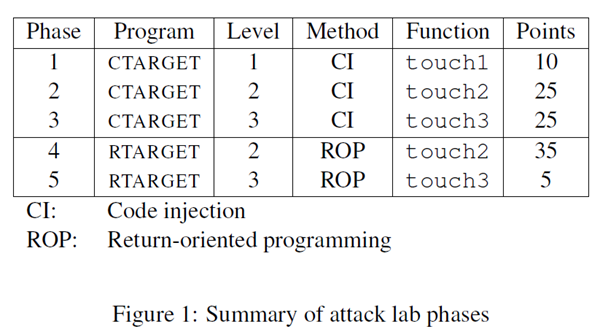
cookie.txt；

farm.c；

hex2raw。

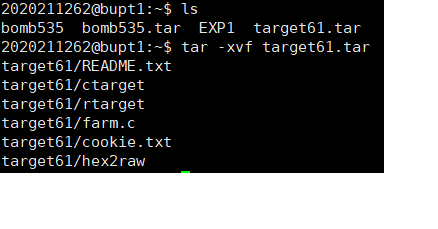
ctarget和rtarget运行时从标准输入读入字符串，这两个程序都存在缓冲区溢出漏洞。通过代码注入的方法实现对ctarget程序的攻击，共有3关，输入一个特定字符串，可成功调用touch1，或touch2，或touch3就通关，并向计分服务器提交得分信息；通过ROP方法实现对rtarget程序的攻击，共有2关，在指定区域找到所需要的小工具，进行拼接完成指定功能，再输入一个特定字符串，实现成功调用touch2或touch3就通关，并向计分服务器提交得分信息；否则失败，但不扣分。因此，本实验需要通过反汇编和逆向工程对ctraget和rtarget执行文件进行分析，找到保存返回地址在堆栈中的位置以及所需要的小工具机器码。实验的具体内容见实验说明，尤其需要认真阅读各阶段的Some Advice提示。

本实验包含了5个阶段（或关卡），难度逐级递增。各阶段分数如下所示：

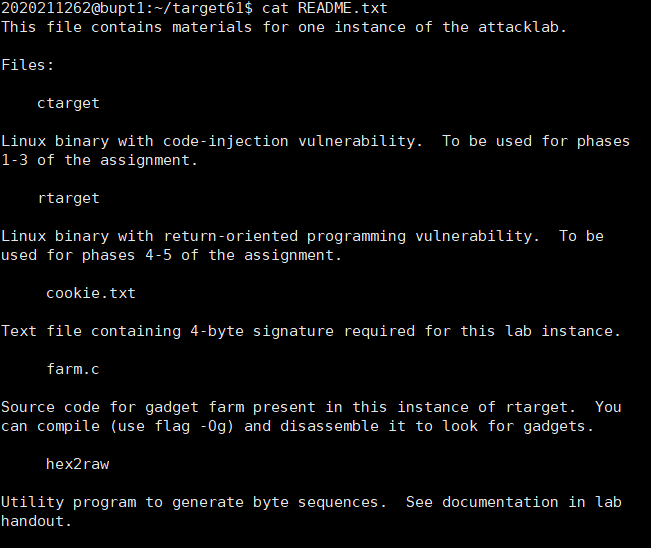


四、实验步骤及实验分析

准备工作：

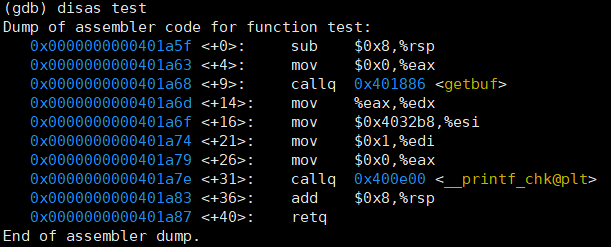


首先，将主目录中的target51.tar进行解压，得到一个新文件夹和六个文件。

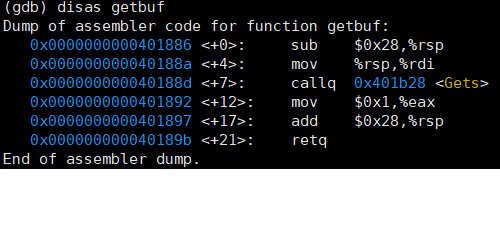


阅读README.txt文件，里面写了其他五个文件各自的作用。阶段1到3的攻击目标是ctarget可执行文件，用植入代码的形式攻击。阶段4和5的攻击目标是rtarget可执行文件，用ROP方式攻击。文件cookie.txt存储了一段实验需要的数据。文件farm.c存储了rtarget程序的一部分源文件。程序hex2raw用来将输入的十六进制转换为字符串。

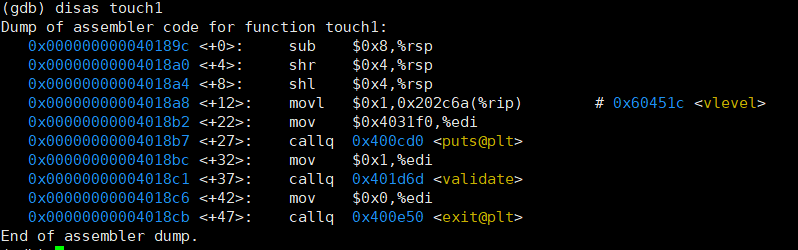
阶段1：



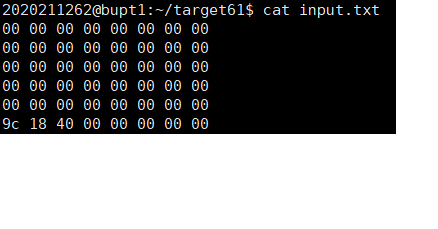
根据实验说明，程序将会调用test函数，在gdb中反汇编test函数。该函数又调用了getbuf函数。



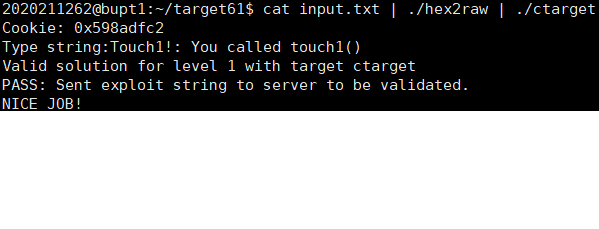
反汇编函数getbuf，该函数调用了Gets函数，也就是利用gets函数进行输入。而Gets函数读入输入的一行字符串且不会管该字符串的长度是否超出上限。并且，根据getbuf函数的汇编代码，可以看到分配给该函数的栈的大小为0x28（40）个字节。



阶段1需要让程序执行touch1函数，该函数的地址是0x40189c。透过输入过长的字符串，改写getbuf函数的返回地址，使其返回至0x40189c，返回并执行函数touch1.



函数getbuf在栈空间的大小是40个字节，也就是返回地址存储在rsp+40处。输入一串数值覆盖掉前40个字节，并从第40字节开始以小端方式输入touch1的地址，也就是将返回地址修改为touch1的地址，使程序返回touch1。



利用hex2raw程序将输入转换并立即执行ctarget程序，程序提示touch1函数被调用，阶段1完成。

阶段2：



1 void touch2(unsigned val)

2 {

3 vlevel = 2; /\* Part of validation protocol \*/

4 if (val == cookie) {

5 printf("Touch2!: You called touch2(0x%.8x)\n", val);

6 validate(2);

7 } else {

8 printf("Misfire: You called touch2(0x%.8x)\n", val);

9 fail(2);

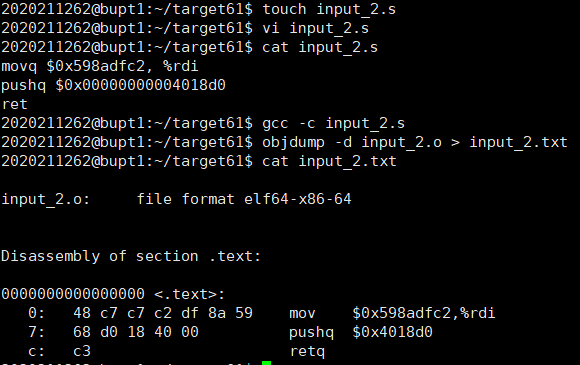
10 }

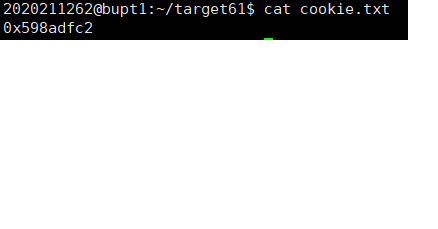
11 exit(0);

12 }

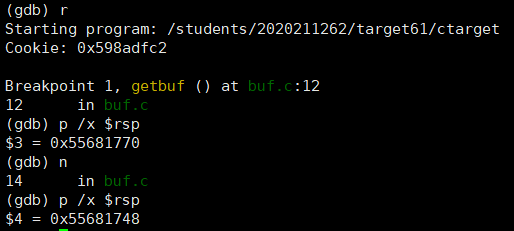
根据实验说明，touch2函数有一个参数，如果该参数等于cookie则通过阶段2。反汇编touch2函数，该函数的地址是0x4018d0。

由于需要传递参数，因此不能像阶段1简单修改返回地址。函数的第一个参数存储在rdi寄存器，因此需要将参数存在该寄存器然后再调用touch2函数。实验说明要求不能使用jmp或call指令，因此用ret指令返回至touch2函数。

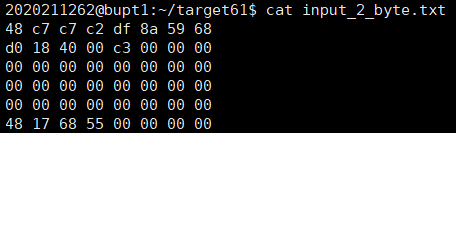




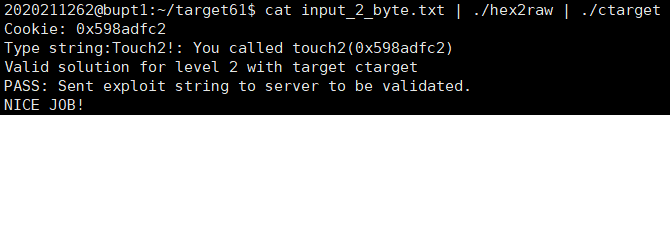
需要植入的代码如上，将cookie传递给寄存器rdi，将touch2函数的地址推入栈中再返回，即返回至touch2函数。编译该代码并反汇编，得到它们的字节表示序列。



植入的代码将会透过getbuf输入，就是存在getbuf的栈顶。因此，需要将getbuf函数的返回地址修改至它的栈顶。通过gdb，找到栈顶地址为0x55681748。



要输入的十六进制如上，代码不需要以小端输入。Getbuf函数将返回至栈顶执行植入的代码。



利用hex2raw程序将输入转换并立即执行ctarget程序，程序提示touch2函数被正确调用，阶段2完成。

阶段3：

1 /\* Compare string to hex represention of unsigned value \*/

2 int hexmatch(unsigned val, char \*sval)

3 {

4 char cbuf[110];

5 /\* Make position of check string unpredictable \*/

6 char \*s = cbuf + random() % 100;

7 sprintf(s, "%.8x", val);

8 return strncmp(sval, s, 9) == 0;

9 }

10

11 void touch3(char \*sval)

12 {

13 vlevel = 3; /\* Part of validation protocol \*/

14 if (hexmatch(cookie, sval)) {

15 printf("Touch3!: You called touch3(\"%s\")\n", sval);

16 validate(3);

17 } else {

18 printf("Misfire: You called touch3(\"%s\")\n", sval);

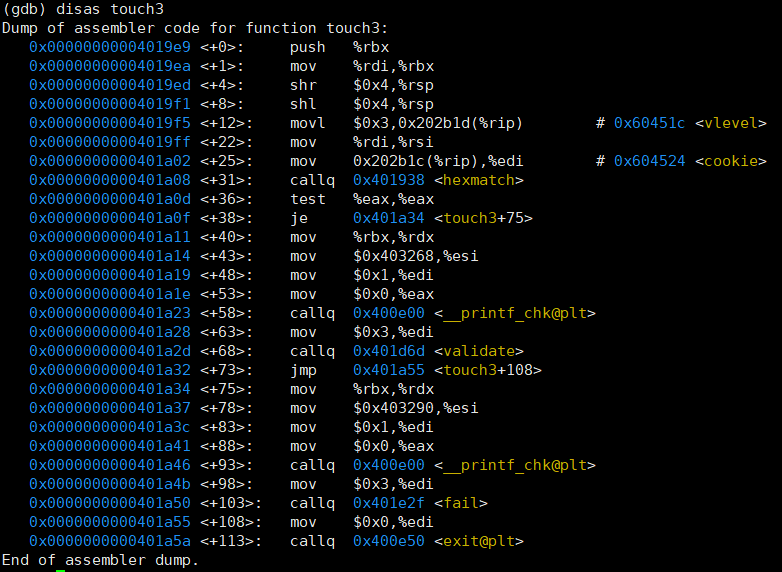
19 fail(3);

20 }

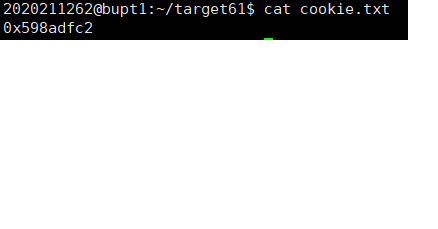
21 exit(0);

22 }

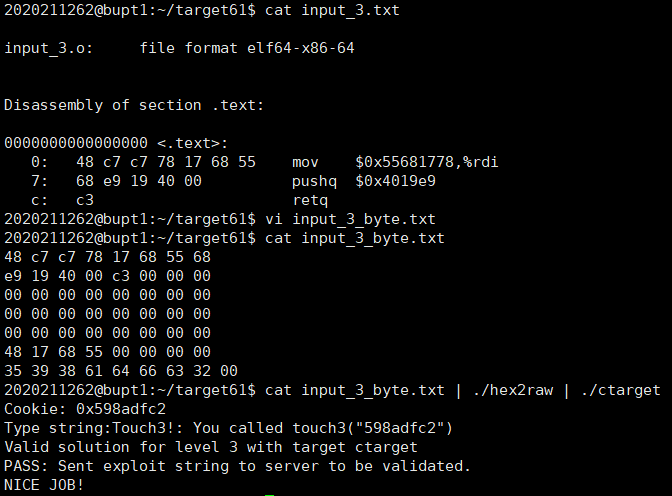
根据实验说明，touch3函数需要传递一个字符串作为参数，通过hexmatch函数将其与cookie进行比较，如果相等则成功调用，完成阶段3.



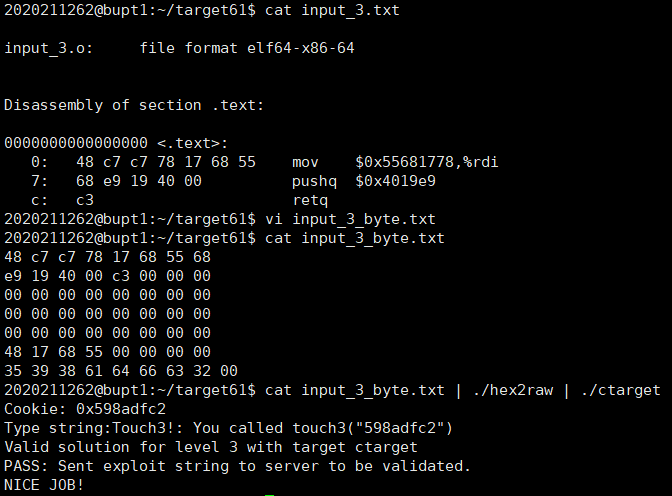
反汇编touch3函数，地址为0x4019e9.



数据cookie是0x598adfc2，转换为字符串的ASCII码为35 39 38 61 64 66 63 32 00（十六进制，00表示字符串结尾），所以需要输入的字符串为35 39 38 61 64 66 63 32 00.



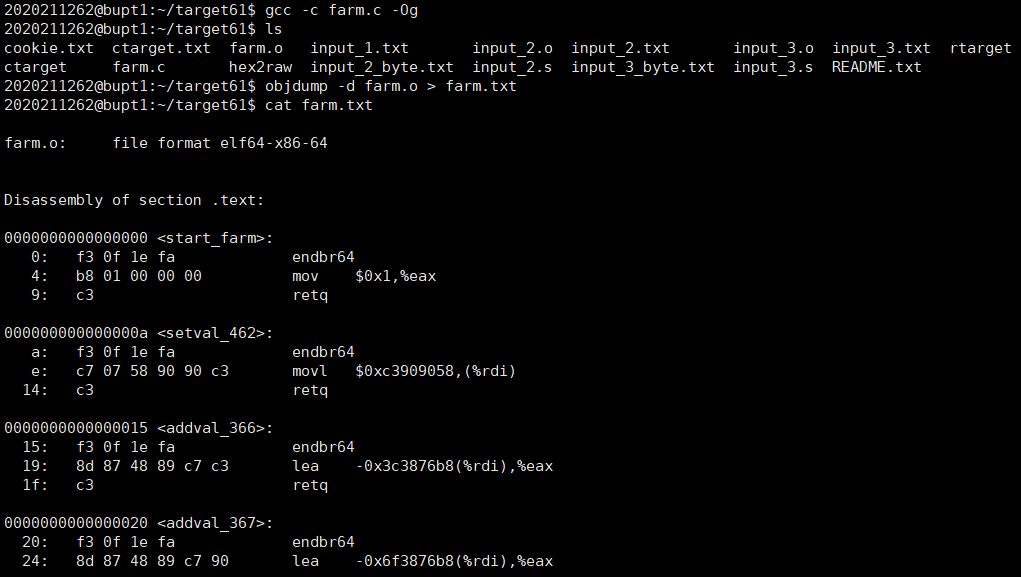
与阶段2类似，需要将getbuf的返回地址修改至栈顶，执行植入的代码。植入的代码将输入字符串的地址，也就是栈顶+0x30=0x55601778处，传递给rdi寄存器，将touch3函数的地址推入栈中，返回至touch3.



利用hex2raw程序将输入转换并立即执行ctarget程序，程序提示touch3函数被正确调用，阶段3完成。

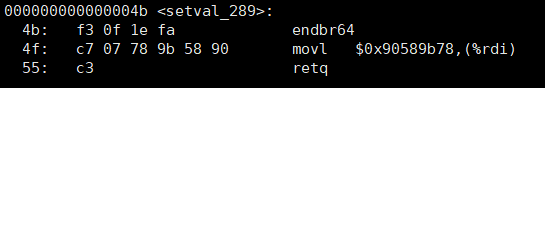
阶段4：

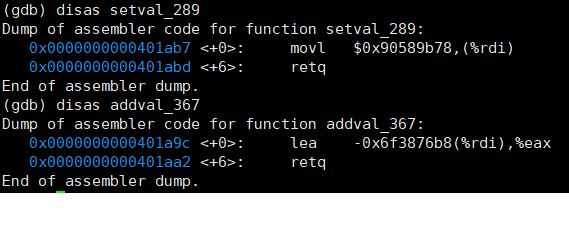
阶段4开始，栈的地址变为随机并且栈中的代码不能被执行，因此不能利用植入代码的方式进行攻击，利用ROP方式攻击，用原有的代码进行攻击。



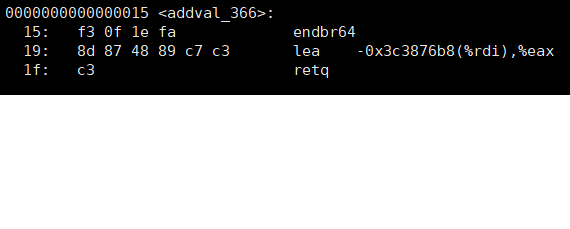
可在farm.c中寻找合适的代码。将其编译后再反汇编，寻找是否有合适的机器指令。

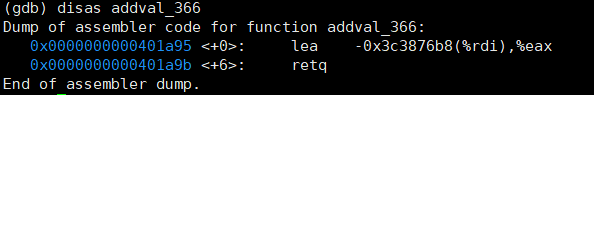
阶段4的任务是用ROP方式重做阶段2。阶段2需要将getbuf的返回地址修改至栈顶，再将cookie传递给寄存器rdi，调用touch2函数。但rtarget中，无法直接找到栈顶的地址。因此，可将getbuf的返回地址修改为一段程序原有的代码，执行相应的工作。



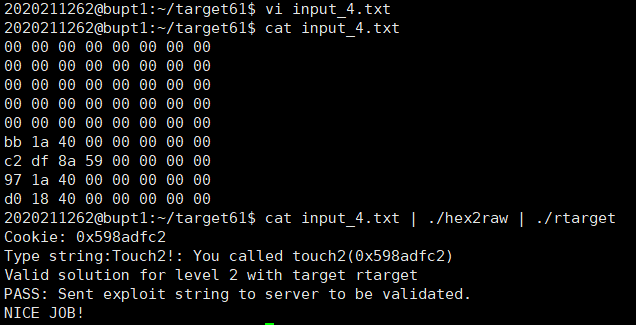


指令popq %rax的机器代码表示是58，可以用函数setval\_289从地址0x401abb处开始的58 90 c3代替。

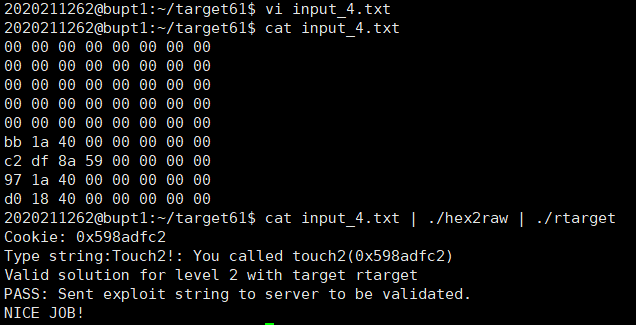




指令movq %rax, %rdi的序列是48 89 c7, 可用函数addval\_366的0x401a97处代替。



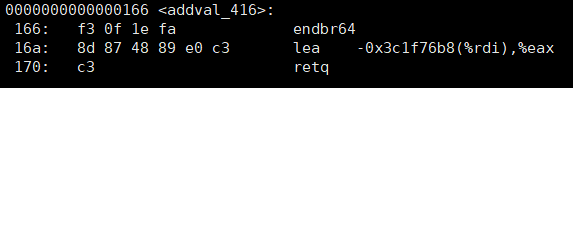
指令popq %rax后面，写上cookie的值，就能将其传递给rax寄存器。在输入的最后输入touch2函数的地址，也就是执行完前面的代码后将会执行touch2函数。

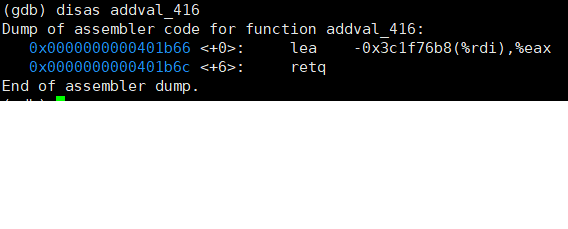


利用hex2raw程序将输入转换并立即执行rtarget程序，程序提示touch2函数被正确调用，阶段4完成。

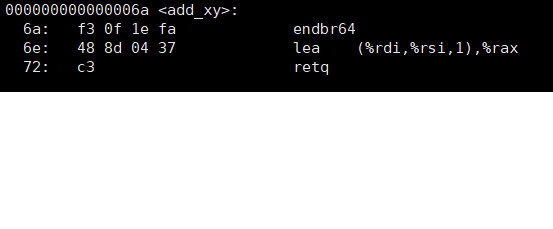
阶段5：

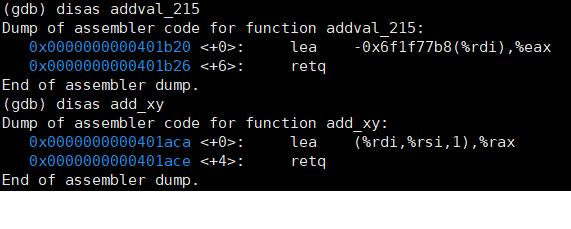
阶段5利用ROP重做阶段3。阶段3的任务是输入字符串，将字符串地址传递给寄存器rdi，调用touch3函数。在程序rtarget中，不清楚栈的地址，所以只能使用偏移量来确定字符串在栈的存储位置。



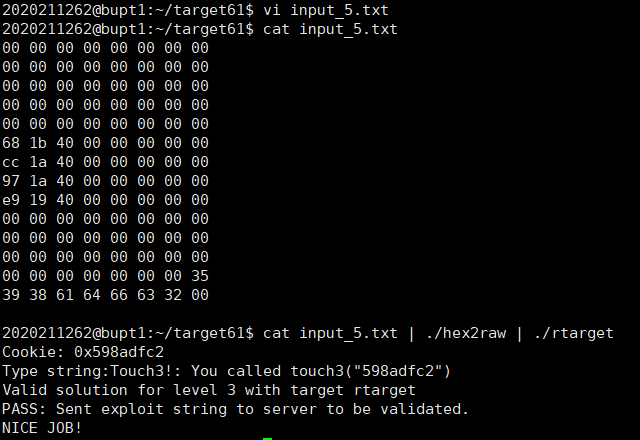


要用偏移量，首先要将寄存器rsp存储的栈顶位置传递给寄存器rax，该指令序列是48 89 e0.可以用函数addval\_416中0x401b68处的指令。

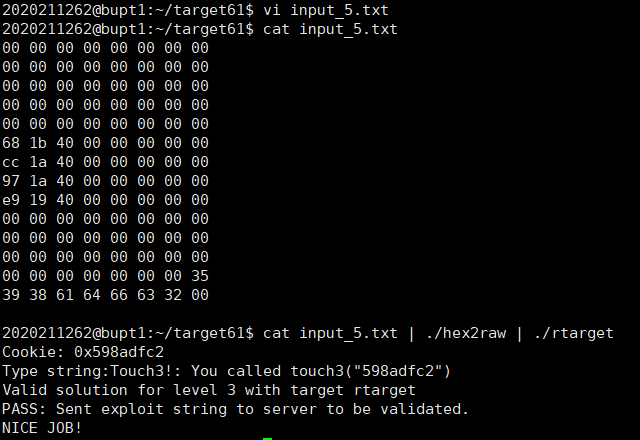




在add\_xy函数0x401acc中找到一段代码，序列04 37将寄存器rax的值加37。因此可以得到rsp+0x37处的地址，将字符串输入至此。



之后可以再次使用阶段4中找到的movq %rax, %rdi指令的地址0x401a97，然后再加上touch3函数的地址，最后在rsp+0x37处输入字符串。需要输入的整体如上。



利用hex2raw程序将输入转换并立即执行rtarget程序，程序提示touch3函数被正确调用，阶段5完成。

五、总结体会

通过本次实验，学到了通过植入代码和ROP两种方法攻击程序，对栈和缓冲区的结构和工作原理有了更深刻的了解，并明白了开发程序时防范攻击的大致方法和重要性。

实验阶段4和5开始，不清楚如何利用原有代码将数据传入寄存器，花费了不少时间。之后得到了同学和实验说明的提示，可以利用popq指令和数据将数据传入寄存器。

实验整体上手有困难，但只要完成一两个阶段，就能找到大致的通关思路，并且有详细的实验说明，整体完成的难度并不大。

六、诚信声明（不签扣10分）

需要填写如下声明，并在底部给出手写签名的电子版。

在完成本次实验过程中，我曾分别与以下各位同学就以下方面做过交流：

1、简单描述交流内容，例如：来自苏承烨的建议，缓冲区的大小可用getbuf函数开头对寄存器rsp的操作进行判断，并且阶段4和5中，如果无法通关，可以尝试寻找并使用其他功能相似的程序原有代码。

此外，我还参考了以下资料：

1. 实验3说明

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照0分计。

（签名）