

**实验报告**



**题目： 缓冲区溢出攻击**

**班 级： 2022211313**

**学 号： 2022211388**

**姓 名： 陈韵涵**

**学 院： 计算机学院**

**2023年 11月26日**

一、实验目的  
1.理解C语言程序的函数调用机制，栈帧的结构。

2.理解x86-64的栈和参数传递机制

3.初步掌握如何编写更加安全的程序，了解编译器和操作系统提供的防攻击手段。  
3.进一步理解x86-64机器指令及指令编码。

1. 实验环境
2. Linux
3. Objdump命令反汇编
4. GDB调试工具
5. 。。。。。

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；

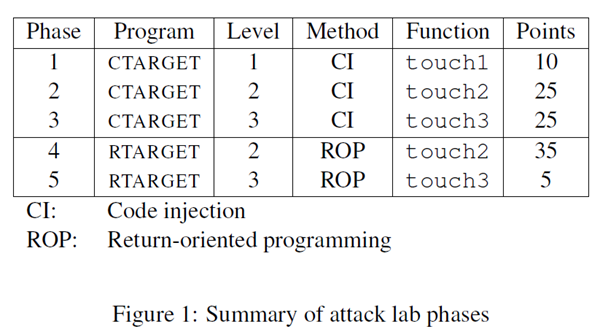
cookie.txt；

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2的具体内容见实验2说明，尤其需要认真阅读各阶段的Some Advice提示。

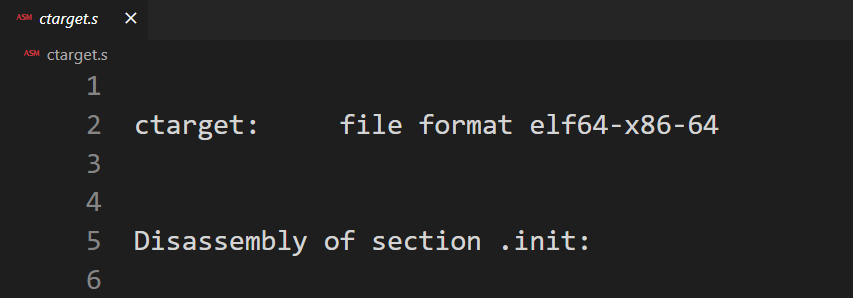
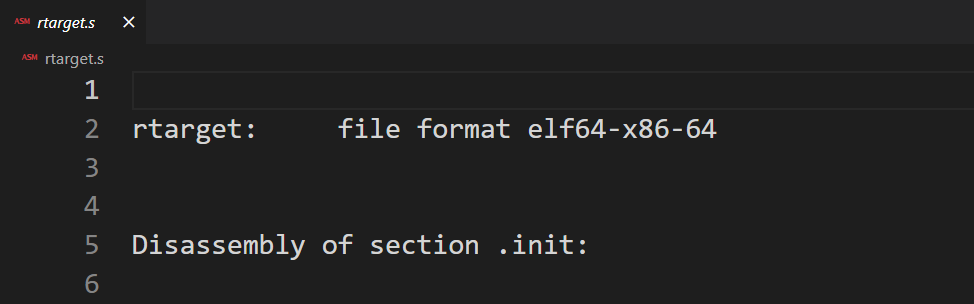
本实验包含了5个阶段（或关卡），难度逐级递增。各阶段分数如下所示：



四、实验步骤及实验分析

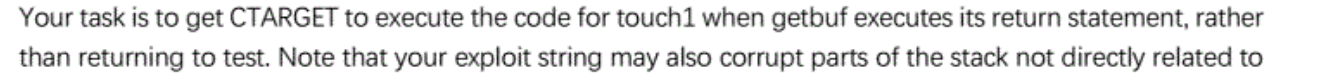
**准备工作：**

利用objdump生成反汇编文件：

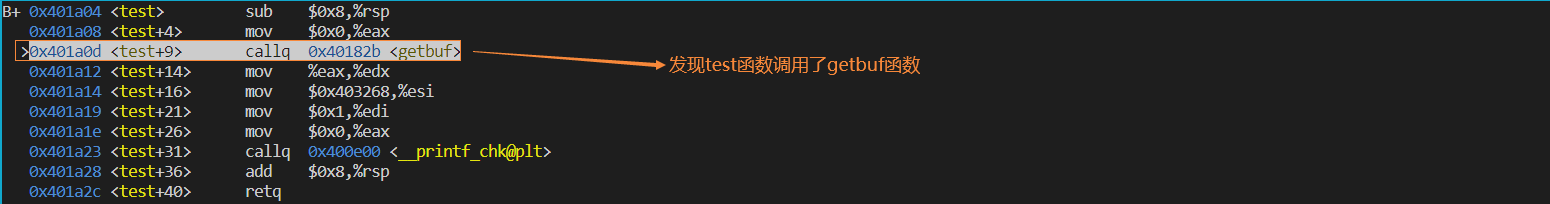
**阶段1：**

根据题目指引:

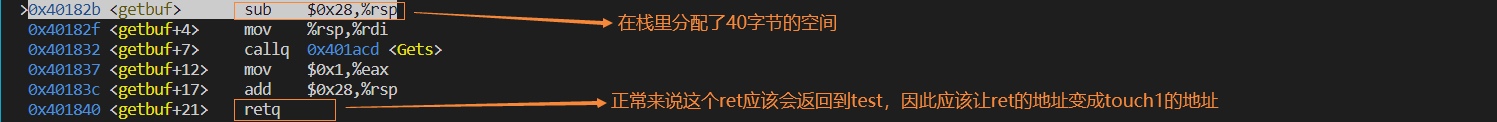


应该是要让getbuf函数返回到touch1而不是test

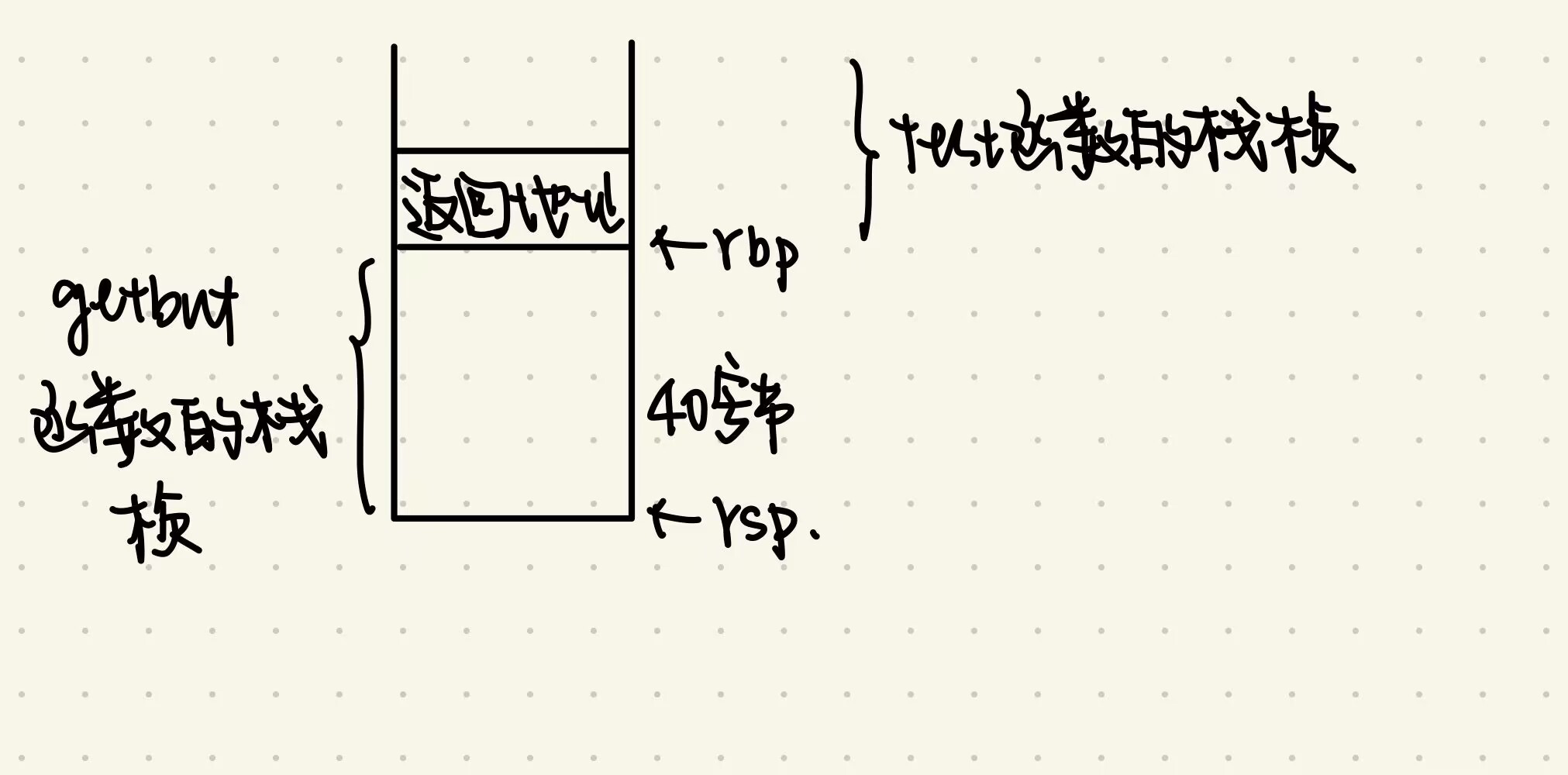
在test函数打断点查看机器指令：



进入getbuf:

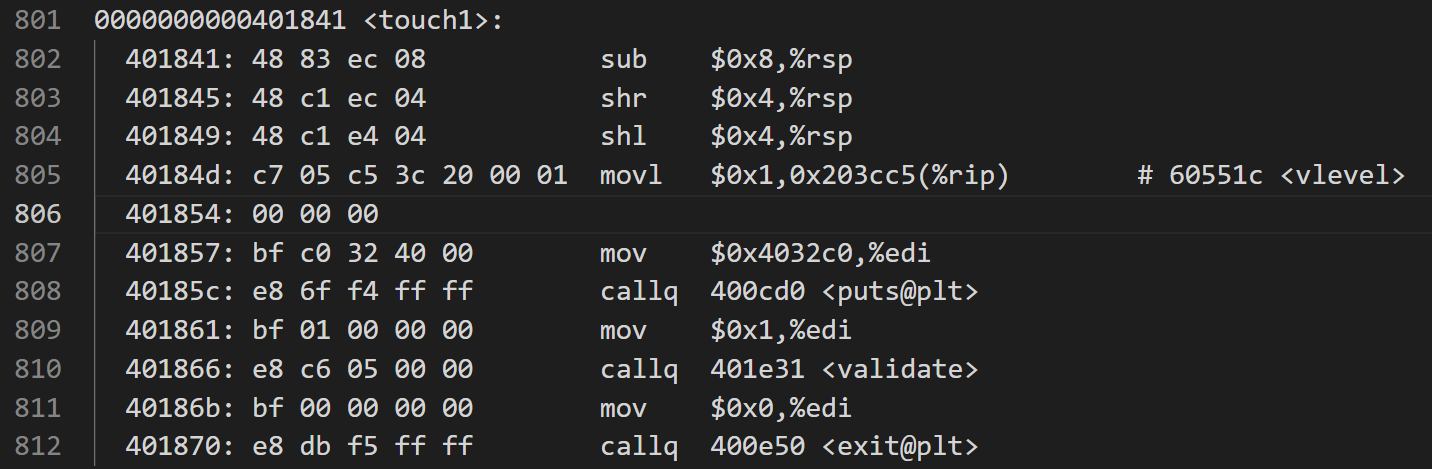


这时候的栈：

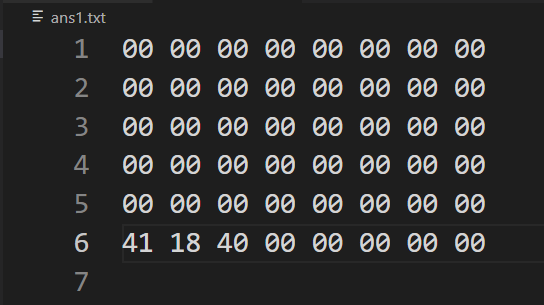
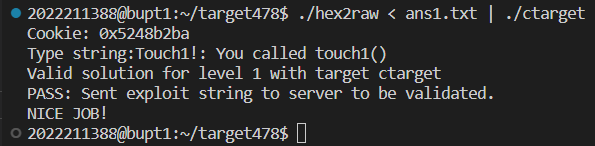


因此只要输入40个字节后，再输入3字节把返回地址覆盖为touch1的地址即可

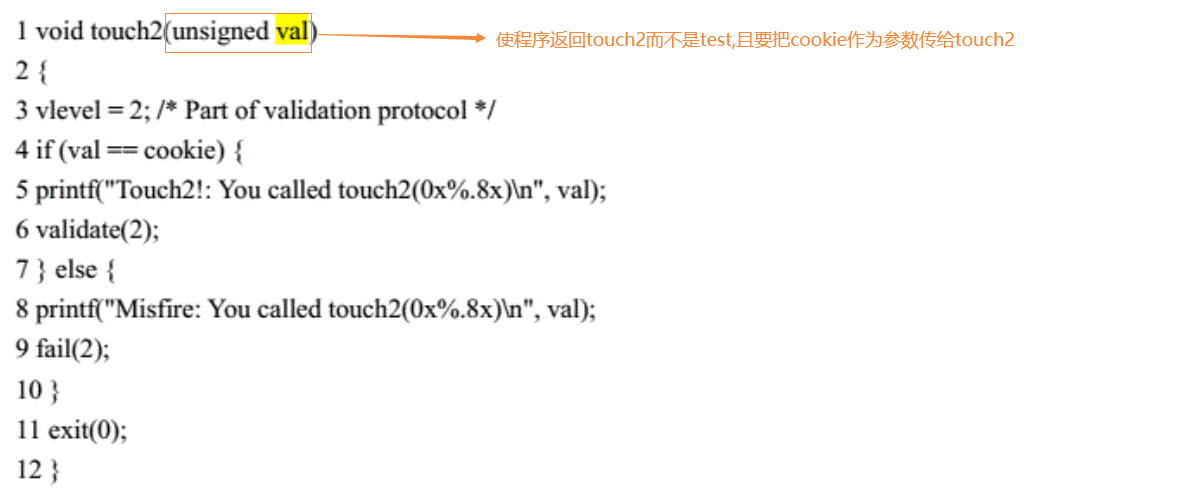
查看touch1的地址：



输入:

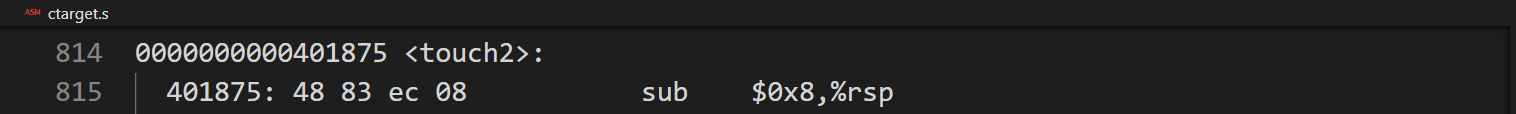
 

**阶段2：**

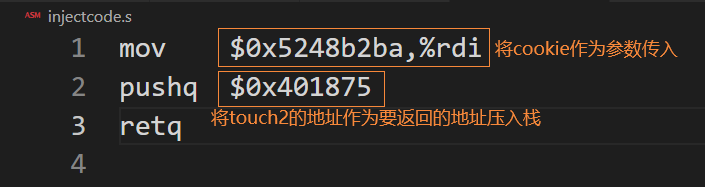


所以这次应该首先把getbuf的返回值地址改成touch2，再把cookie的值传进寄存器%rdi，使它成为touch2的参数

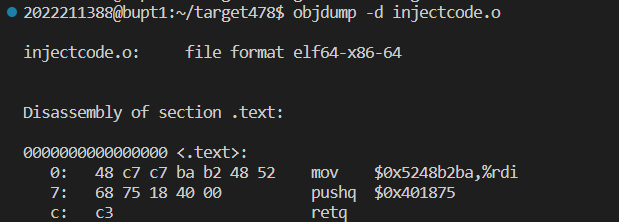
查看touch2的地址:



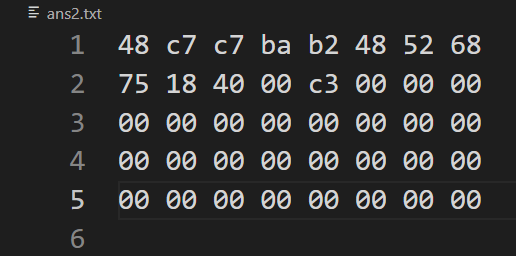
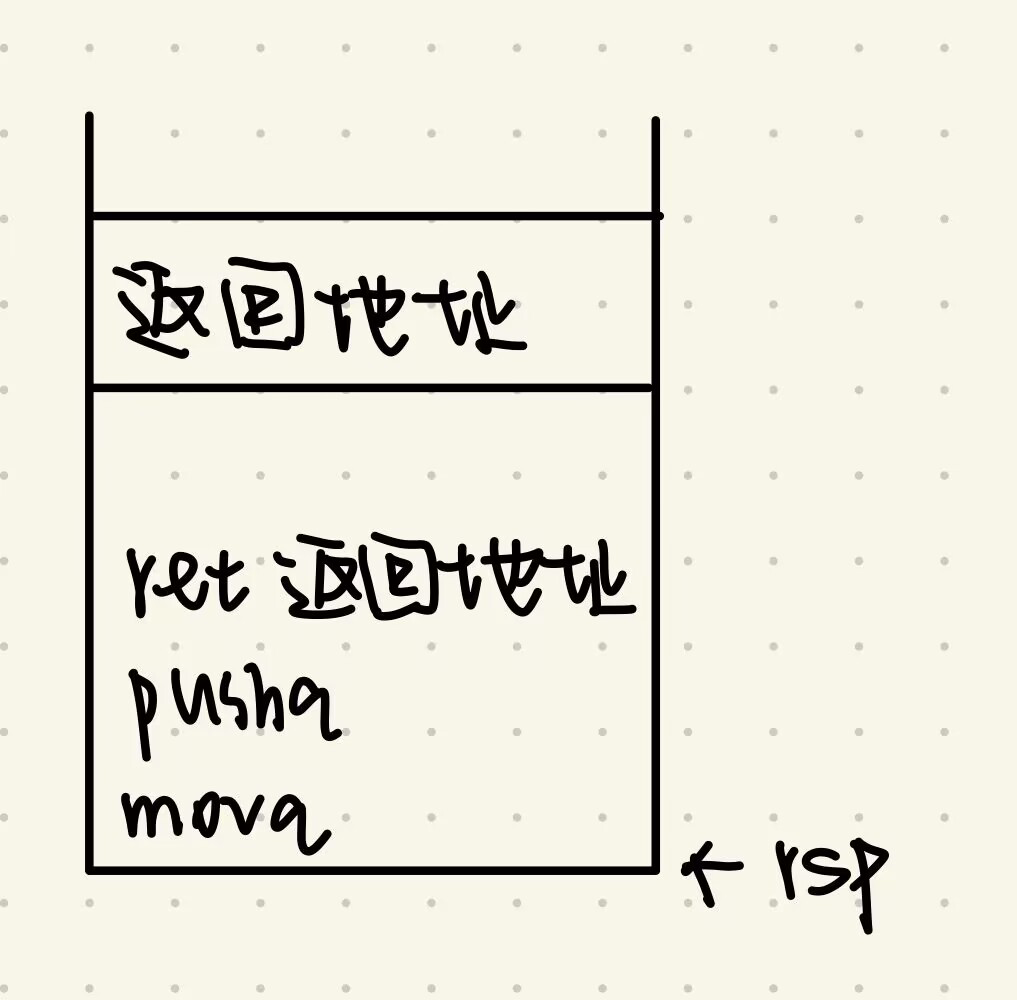
编写要植入的代码并将其转换成计算机可执行的指令序列：







把这一段代码放在buf里，想办法把它执行

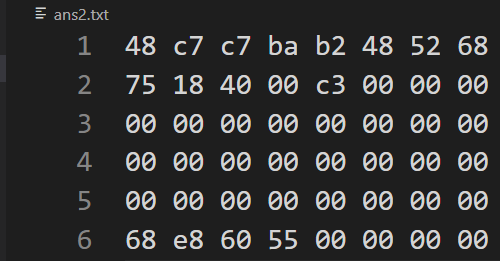
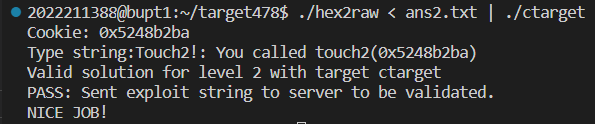
如图所示，此时的getbuf的栈是这样

在buf里面，首先存储了能让%rdi=cookie的代码，然后将touch2的地址入栈，然后执行到ret指令时，程序就会跳转开始执行touch2。然后再因为缓冲区溢出改变了返回地址，getbuf返回时将开始执行buf栈顶的这段代码。

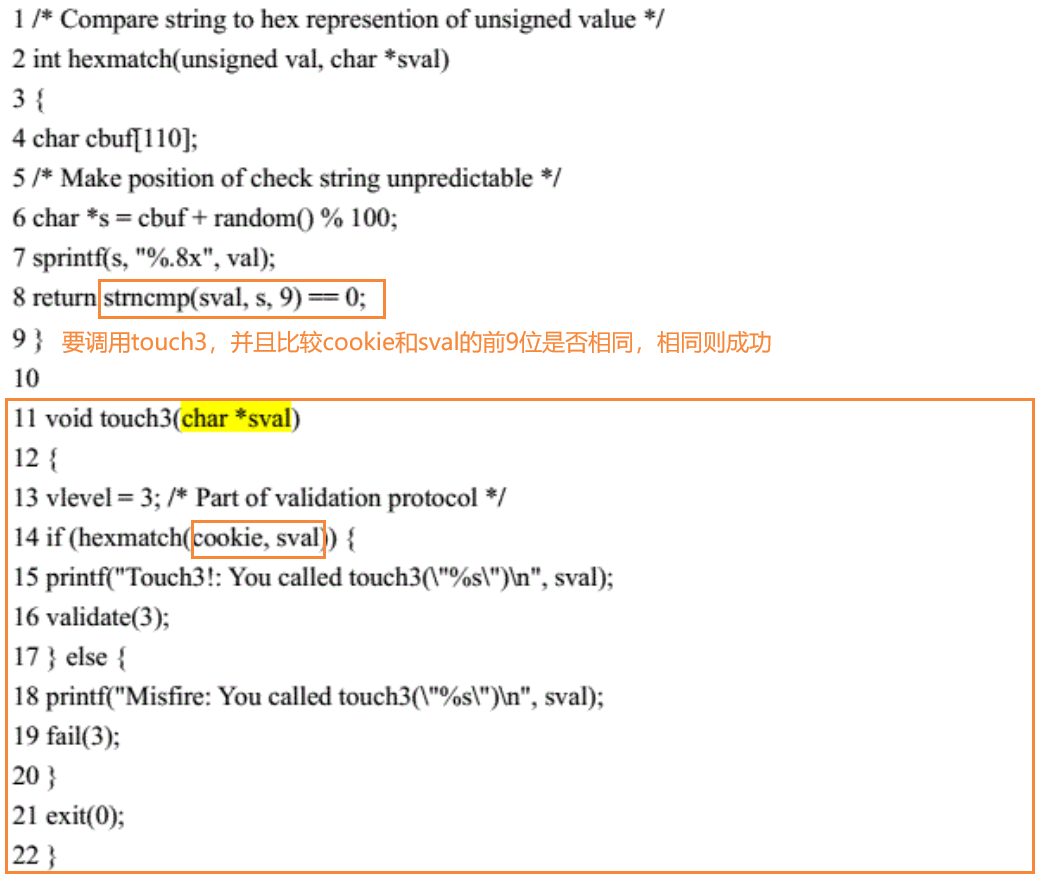
运行代码至给buf分配完栈里的空间，查看此时rsp的地址：



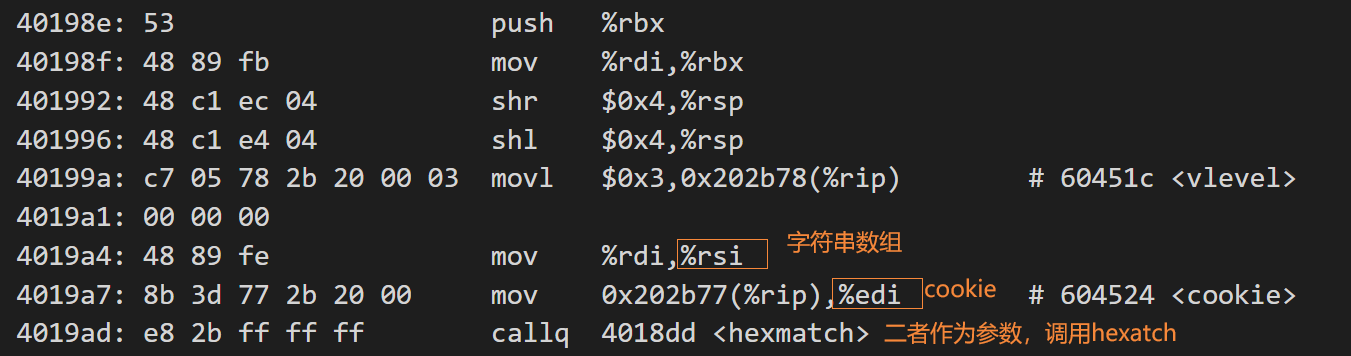
因此利用缓冲区溢出修改getbuf的返回地址，使getbuf返回之后执行放在buf里面的代码：

**阶段3：**

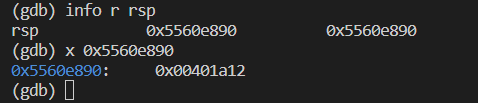


分析touch3:



因为hexmatch里面s的位置是随机的，且题目提示有可能hexmatch和strncmp会覆盖getbuf的缓冲区，因此应该把输入的字符串数组存在test的栈里面。

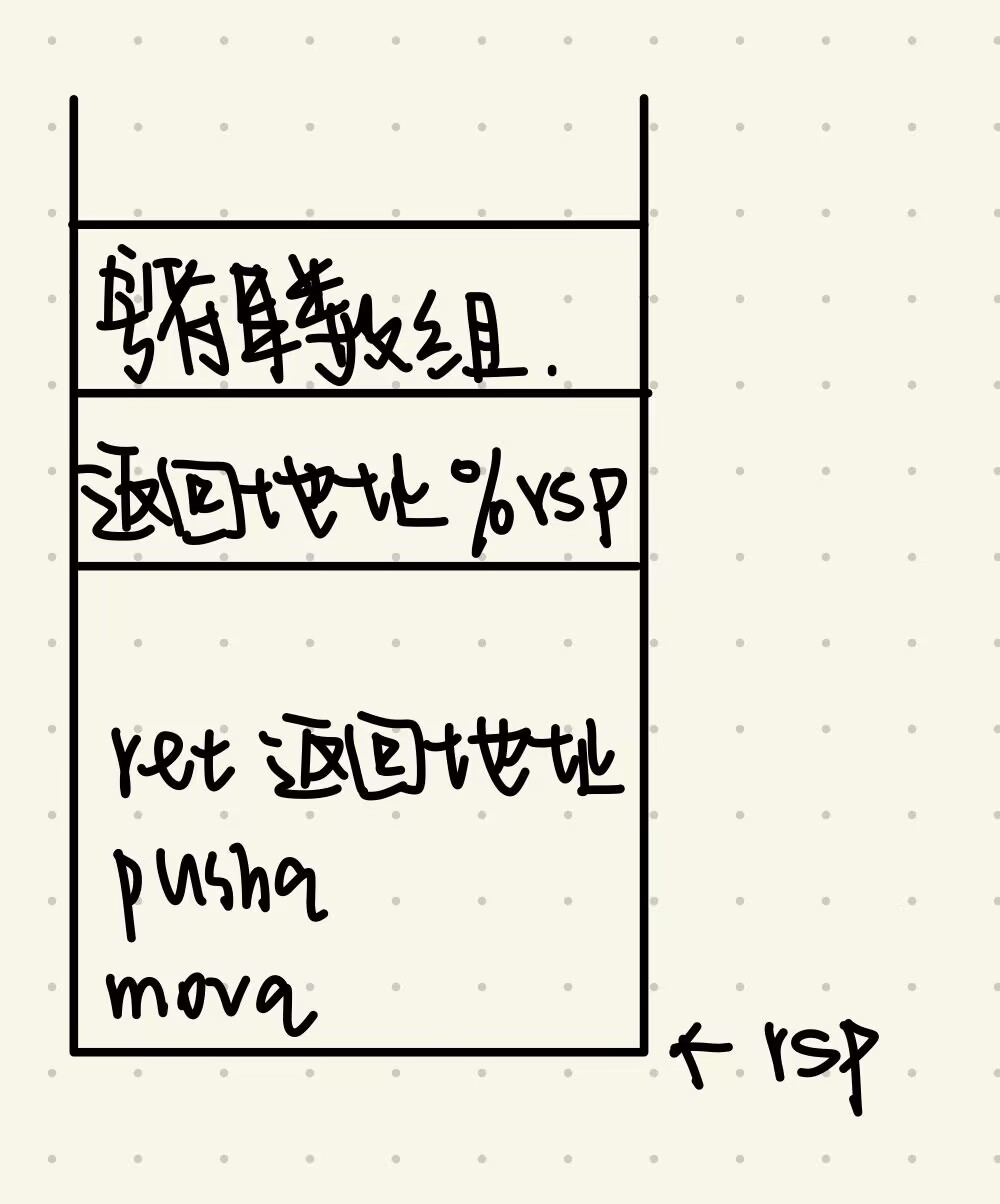
查看在getbuf分配栈之前栈指针rsp的情况：



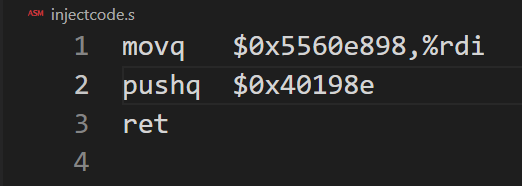
此时栈里存的就是调用getbuf的下一条指令的位置

因此注入代码的思路应该是：把作为touch3参数的字符串数组的起始地址存在0x5560e898处，即存储getbuf返回值的后8个字节处。再把touch3的地址入栈，利用ret使程序跳转到touch3。

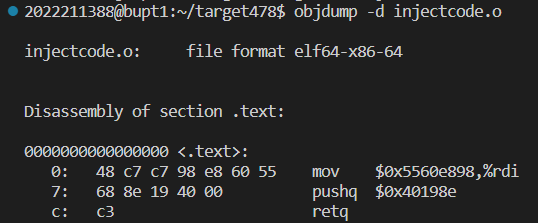
此时栈里的情况大致如下：



编写汇编指令：

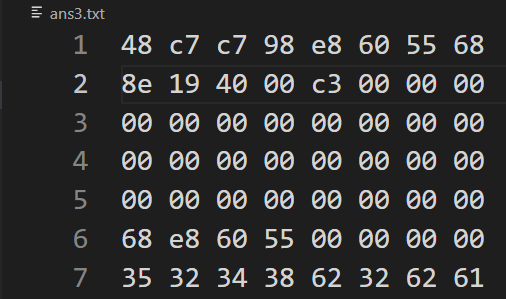
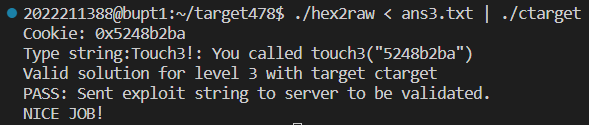


获取可执行的指令序列：



与阶段2一样，getbuf的栈顶指针应该是：0x5560e868

将cookie转换为16进制表示的字符串，编写输入：

**阶段4：**

思路：把cookie的值赋值给%rdi，再利用ret进入touch2  
 因此把cookie存在栈上，再利用指令pop %rdi就可以把cookie的值赋给%rdi，但由于farm中没有这个指令，因此使用：

pop %rax

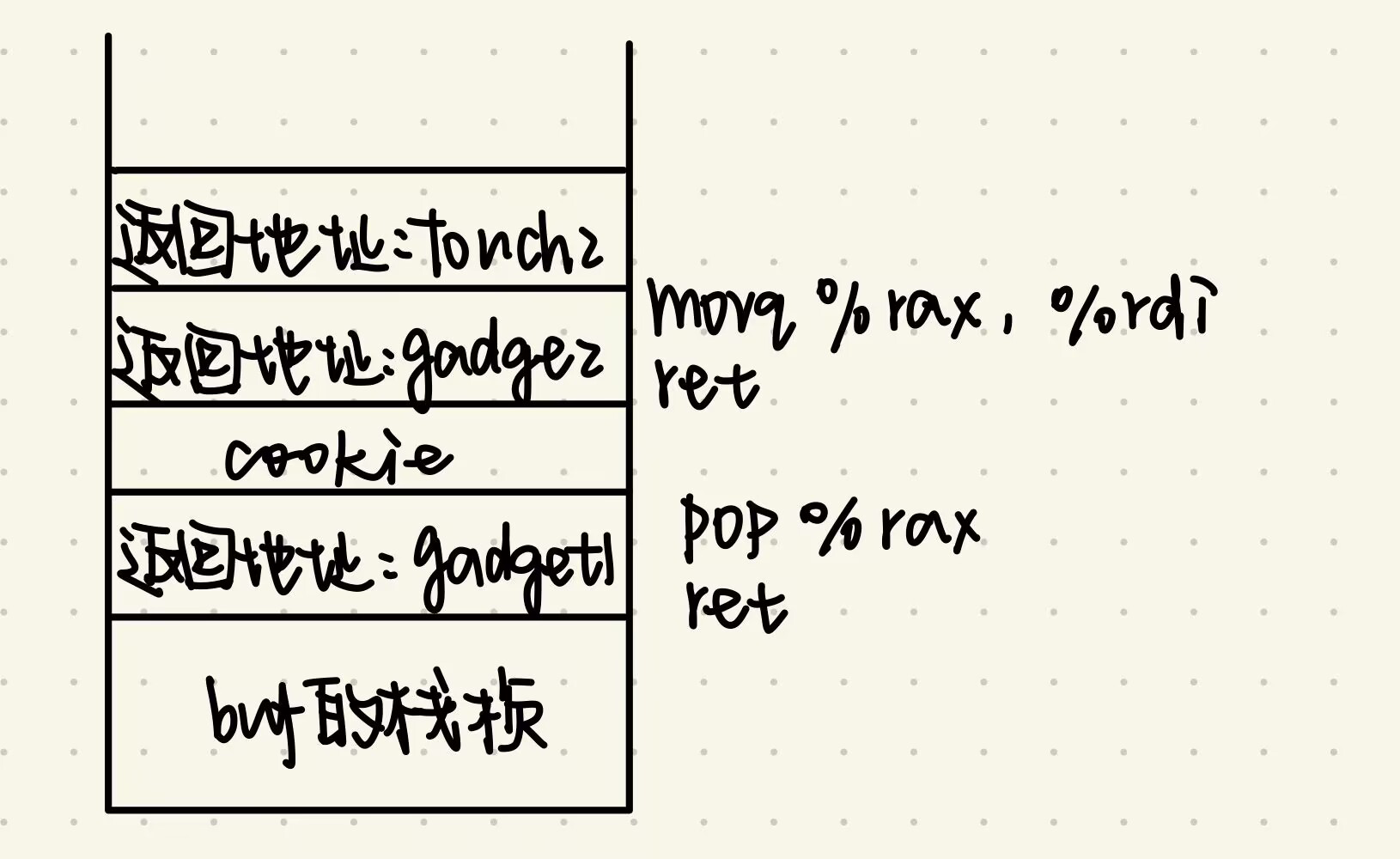
ret

movq %rax,%rdi

ret

每次ret都会使代码跳到下一个要执行的代码段

此时栈的内容如图所示:



要输入的字符串：前40字节是正常存储在buf栈帧里的，41-48字节存储gadget1的地址，49-56字节存储cookie的值，57-64字节存储gadget2的地址，65-72字节存储touch2的地址

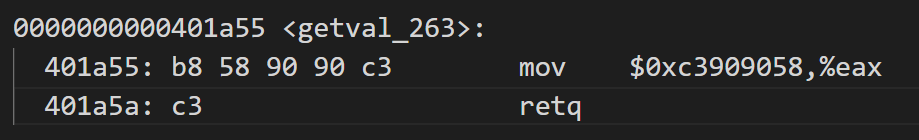
查表得：

58 popq %rax

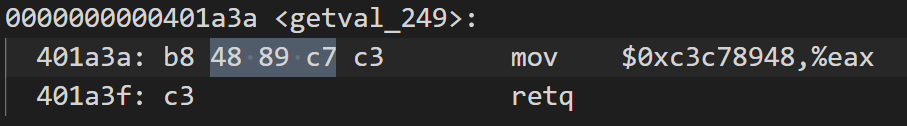
48 89 c7 movq %rax,%rdi

其中90表示空可以忽略，因此要找的几个指令必须与90和c3连续：

Gadget1：地址为0x401a56

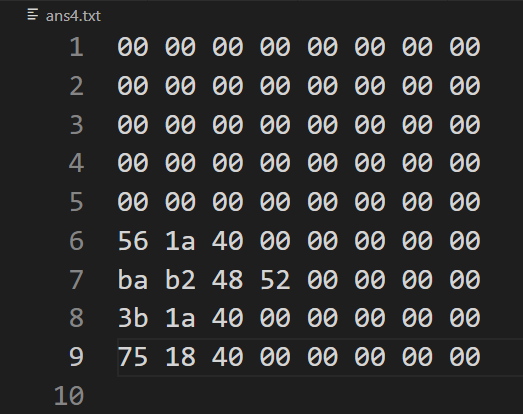
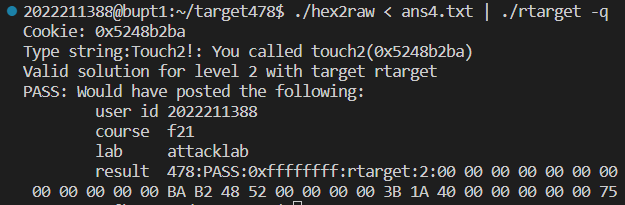


Gadget2：地址为0x401a3b



Touch2：地址为0x401875

编写输入字符串：

五、总结体会

总结心得（包括实验过程中遇到的问题、如何解决的、过关或挫败的感受、实验投入的时间和精力、意见和建议等）

1. 不明白怎么使用各个文件

仔细阅读说明文档，发现如何使用hex2raw以及每个文件的用途都写得比较清楚。

2. 在第二关的时候卡了很久，因为不知道原来要自己写代码来反汇编获得指令编号，但其实这个在书上和说明文档也提到了，应该加强对这两个东西的理解。