

**实验报告**



**题目： 缓冲区溢出攻击**

**班 级： 2023211301**

**学 号： 2023210896**

**姓 名： 王书翰**

**学 院： 计算机学院（国家示范性软件学院）**

**2024年11月16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；

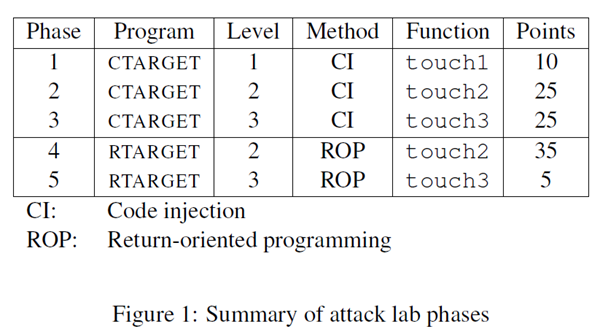
cookie.txt；

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。

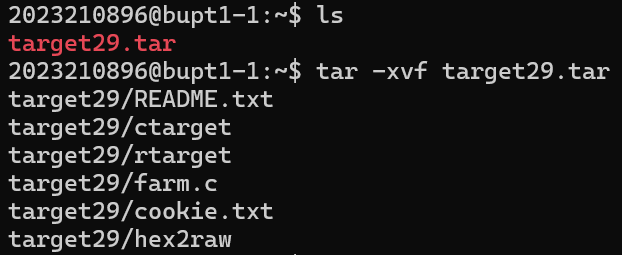
本实验包含了5个阶段（或关卡），难度逐级递增。各阶段分数如下所示：



四、实验步骤及实验分析

0、准备阶段

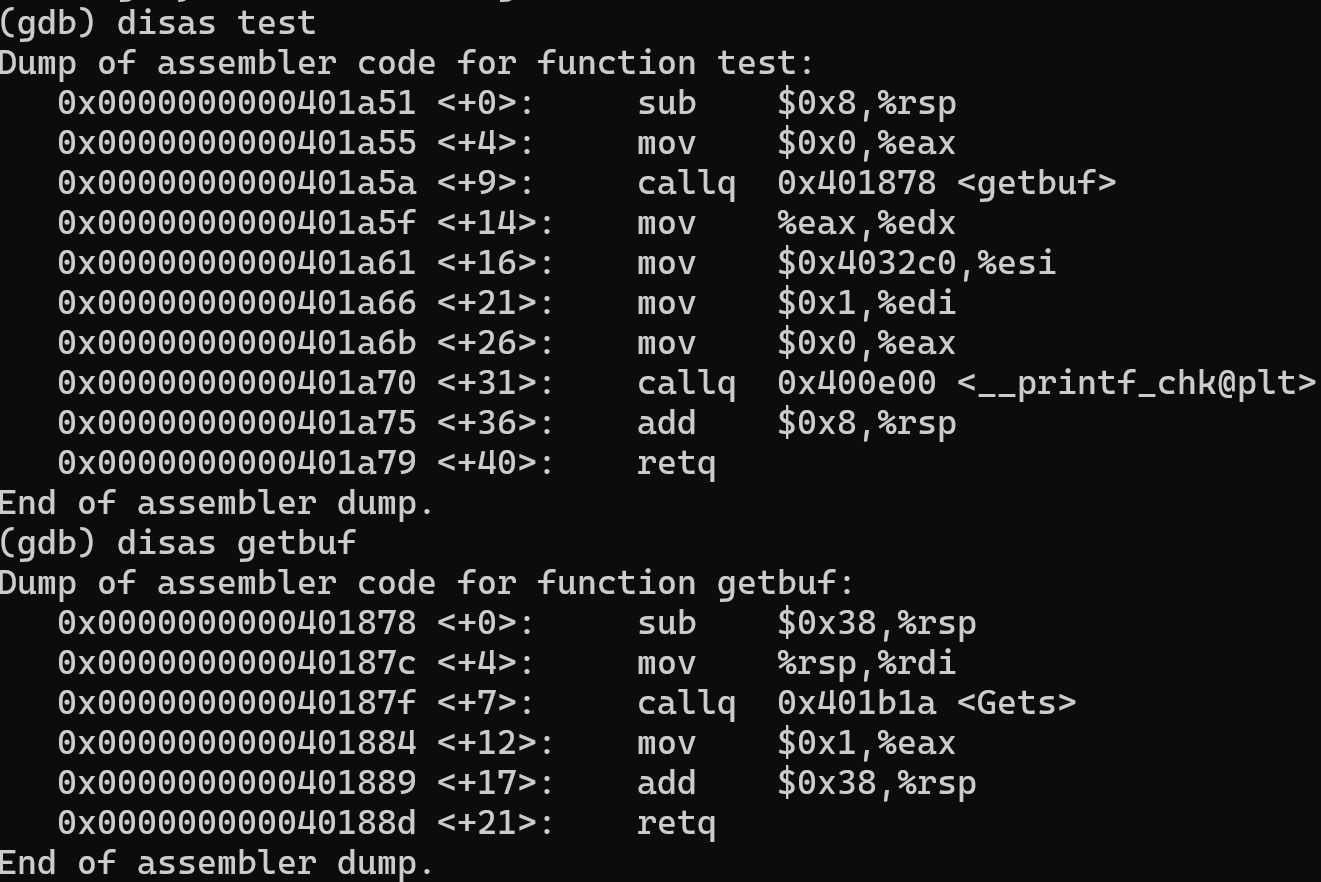
首先使用ls指令，查看压缩文件使用tar -xvf target29.tar 指令解压文件，得到6个文件



使用cd target29指令将目录更改为target29文件夹

1、第一阶段

该阶段的任务是在缓冲区溢出的部分输入指令将getbuf函数的返回值指向touch1。使用gdb打开ctarget可执行文件后先使用disas test和disas getbuf指令反汇编这两个函数



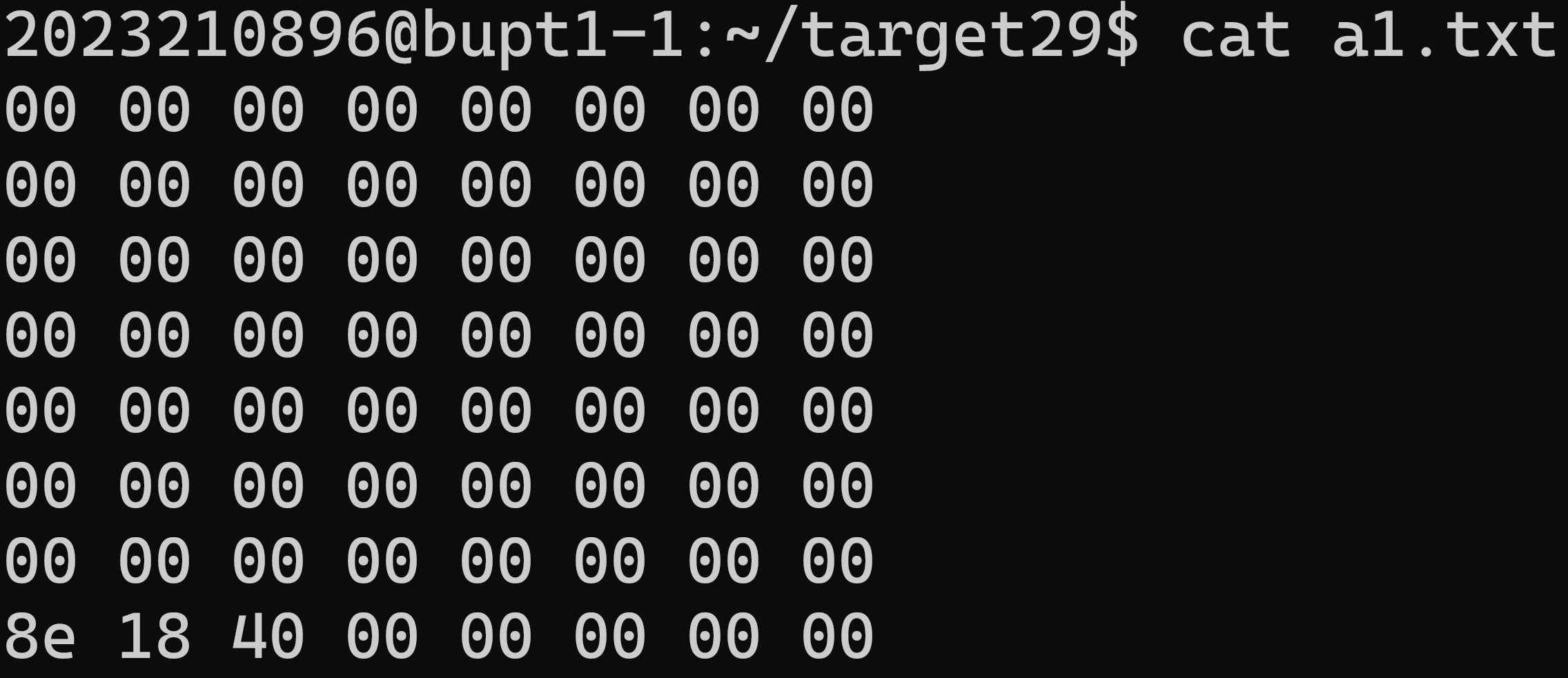
分析汇编语言可知getbuf 分配了3\*16+8=56个字节的栈帧，随后将栈顶位置作为参数调用Gets函数，读入字符串，然后正常返回，跳转到test函数继续执行。再使用disas touch1指令查看touch1函数的地址



由图可知touch1函数的首地址为0x40188e，我们需要修改getbuf结尾处的ret指令，将其指向 touch1 函数的起始地址0x40188e。要想将其准确指向0x40188e，要首先将 getbuf 的56字节内容用00填充满，使其溢出，再用0x40188e覆盖 getbuf 原来的返回地址。

用vim指令新建一个名称为a1.txt的文件，先输入8个00，用空格隔开，再按照相同的格式输入7行。因为该实验系统采用小端存储，故在最后一行倒过来输入40188e，即 8e 18 40，用00填充剩下的部分

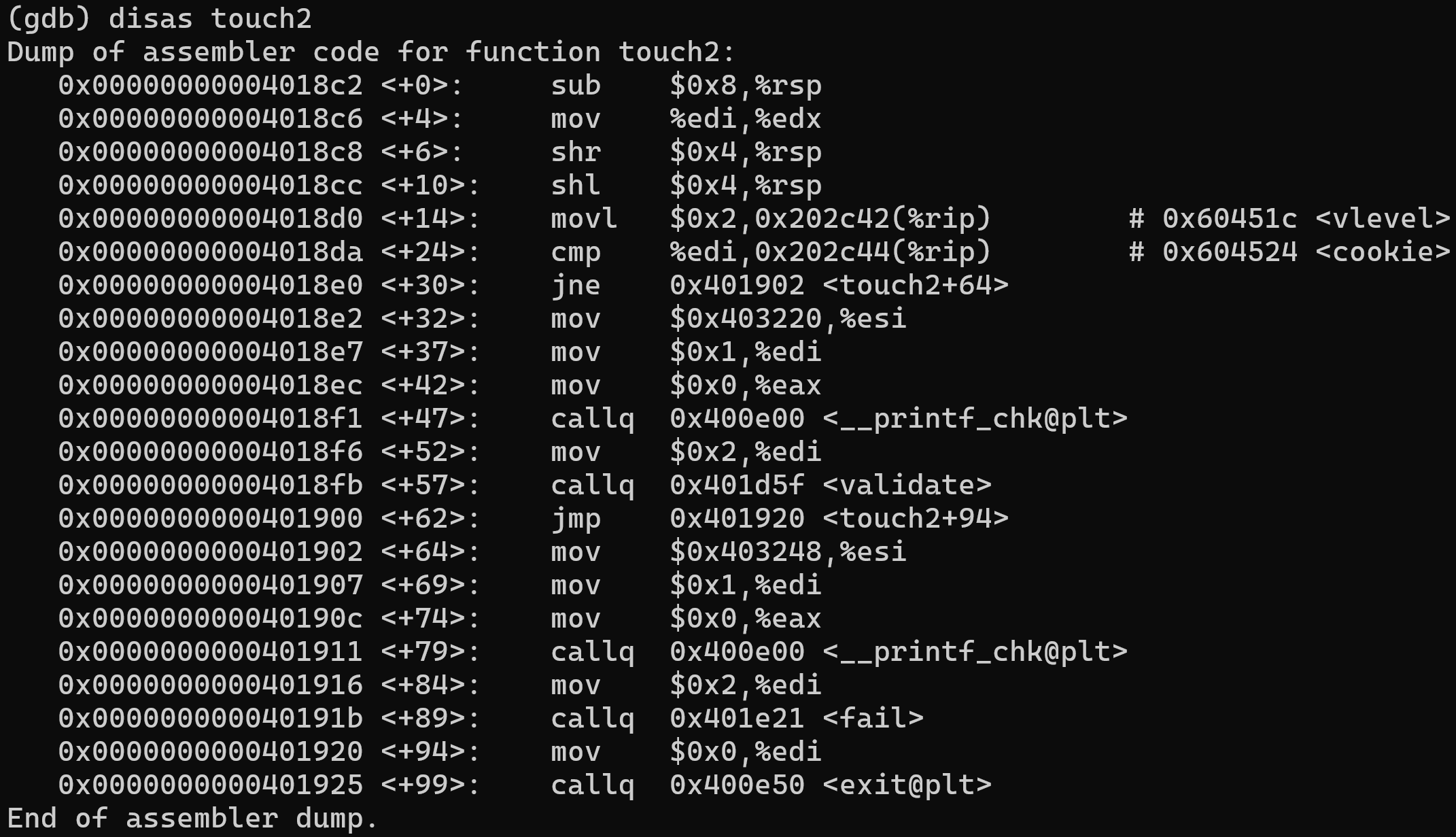
使用cat a1.txt指令查看第一阶段的输入



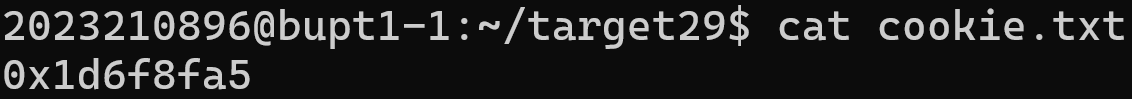
使用攻击指令 ./hex2raw < a1.txt | ./ctarget 攻击即可

2、第二阶段

该阶段的任务是将cookie识别码注入touch2函数中，之后将getbuf函数的返回值指向touch2。使用disas touch2指令查看touch2函数的地址



由图可知touch2函数的首地址为0x4018c2。再使用cat cookie.txt指令查看cookie的值



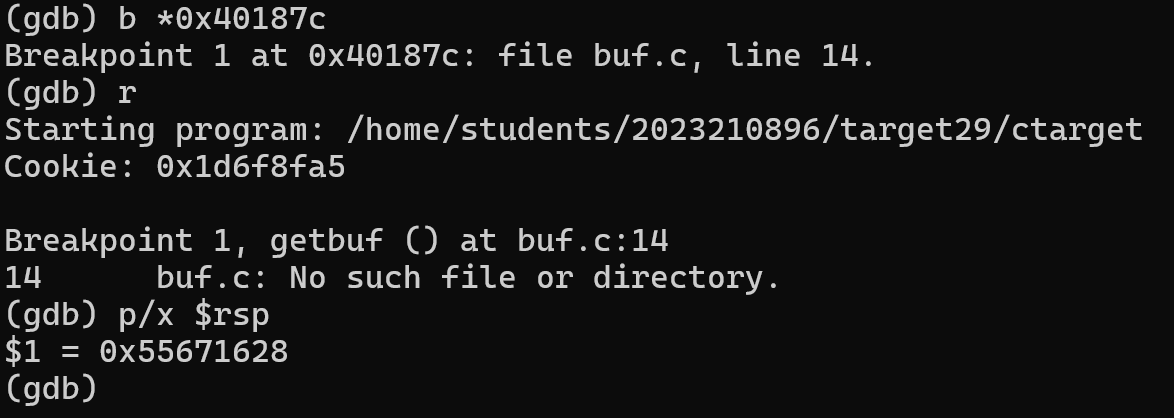
为将cookie的值注入touch2函数中，考虑构造如下指令

movq $0x1d6f8fa5, %rdi //将cookie的值存入%rdi寄存器中

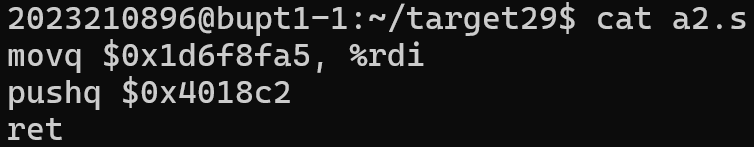
pushq $0x4018c2 //向栈中压入touch2函数的首地址

ret //调用返回地址

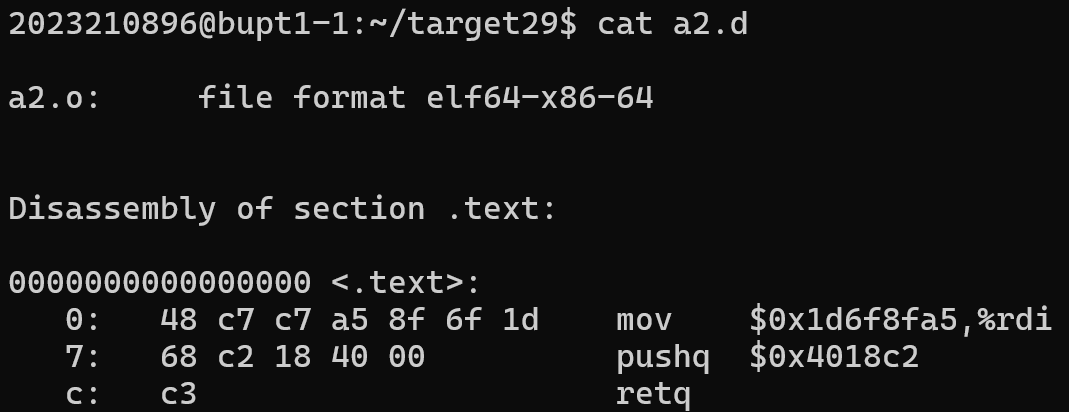
再查看%rdi寄存器的值，即栈顶指针的位置。在getbuf函数中的<+4>行即0x40187c处设置断点，随后执行函数并使用p/x $rsp查看栈顶地址

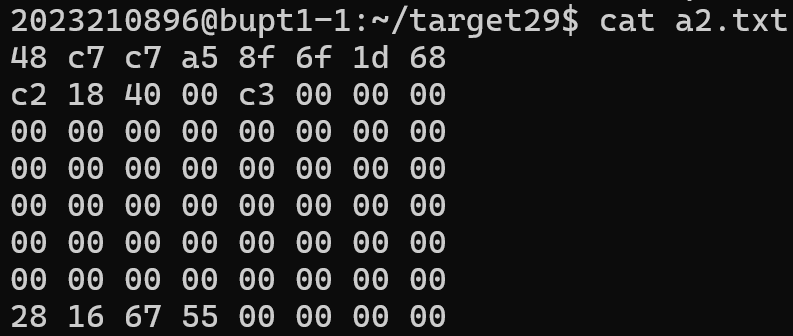


故0x55671628就是要被修改的函数的返回地址。先用vim指令新建一个名称为a2.s的文件，将构造的指令输入，后用cat a2.s指令查看输入的结果



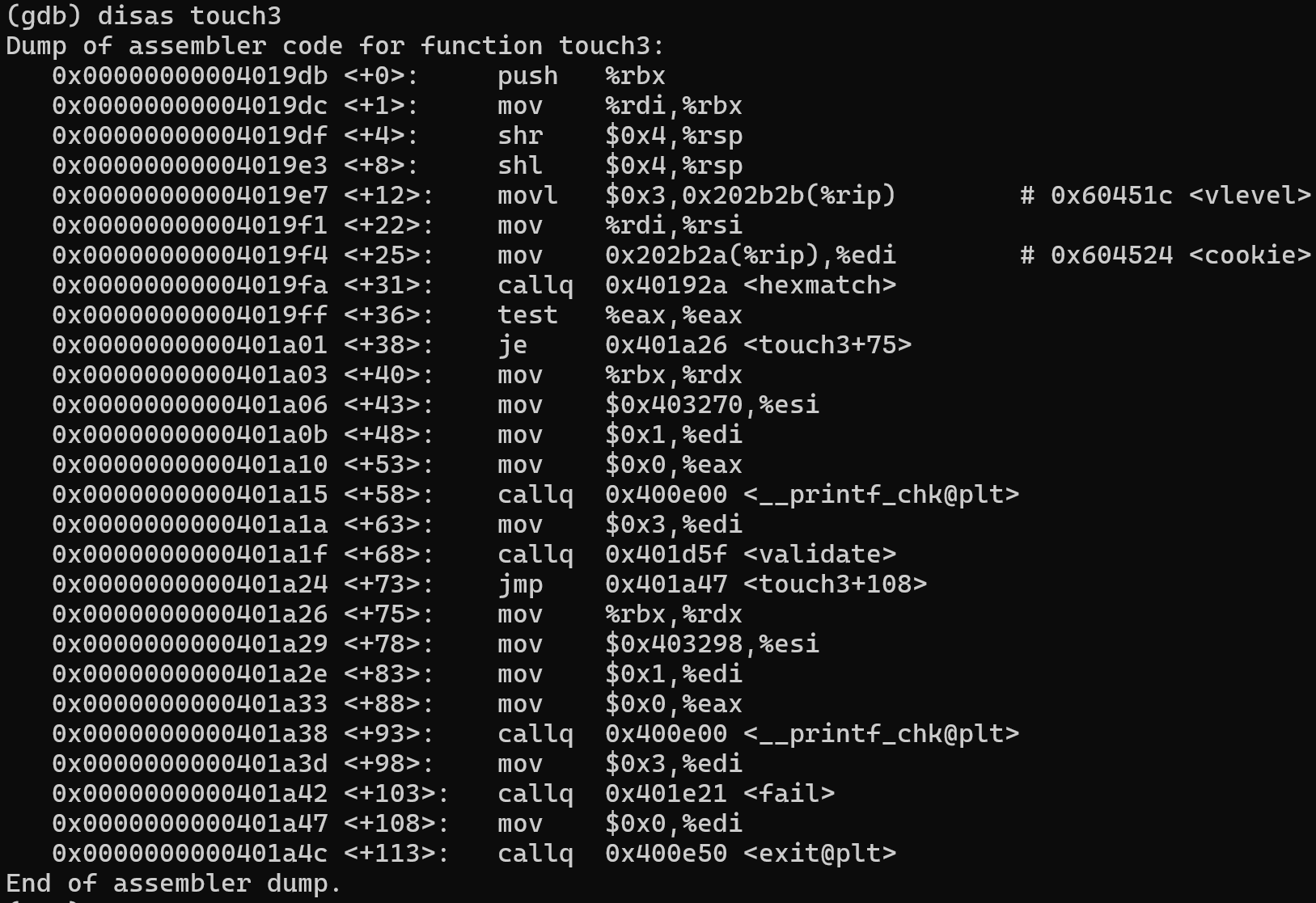
随后使用gcc -c a2.s 指令与objdump -d a2.s > a2.d 指令先将a2.s编译生成可执行文件，再将可执行文件反编译。后使用cat a2.d指令查看代码指令的字节级表示



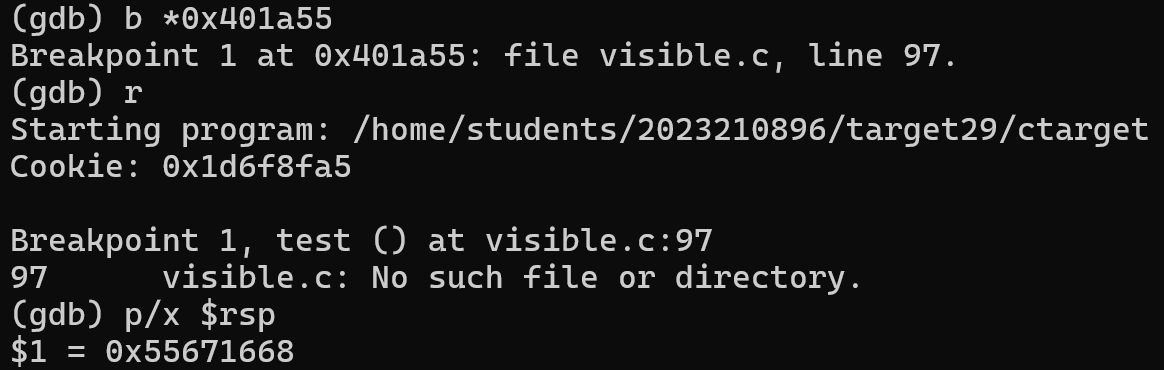
最后用vim指令新建一个名称为a2.txt的文件，先顺序输入上图中的字节序列，随后用00填满剩下的字节，最后将返回地址逆序输入放在末尾。使用cat a2.txt指令查看第二阶段的输入

使用攻击指令 ./hex2raw < a2.txt | ./ctarget 攻击即可

3、第三阶段

该阶段的任务是将一串字符串注入touch3函数中，之后将getbuf函数的返回值指向touch3。最后将cookie字符串存在test函数的栈上。使用disas touch3指令查看touch3函数的地址

即函数touch3的首地址为0x4019db。在test函数的<+4>行即0x401a55处设置断点，随后执行函数并使用p/x $rsp查看栈顶指针地址



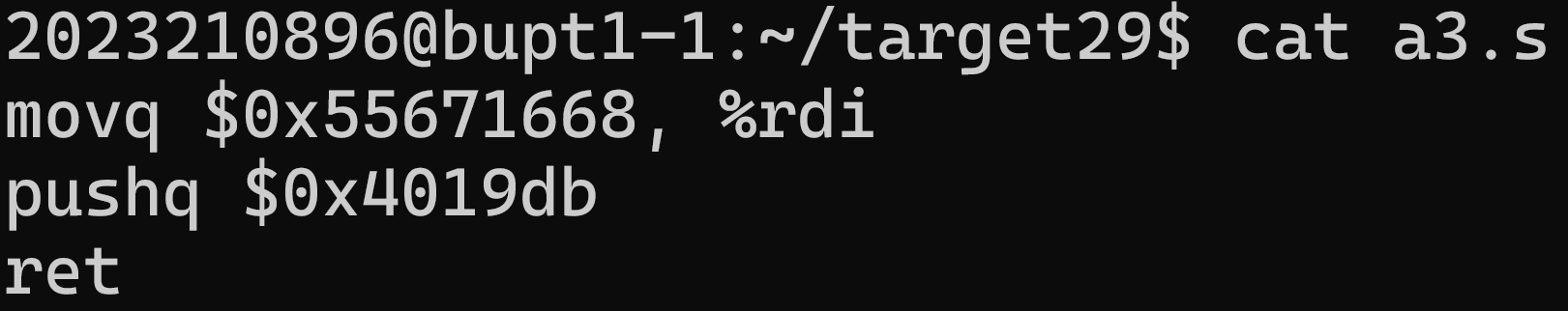
由图可知0x55671668即为touch3传入的参数，也是字符串存放的位置。考虑构造如下指令

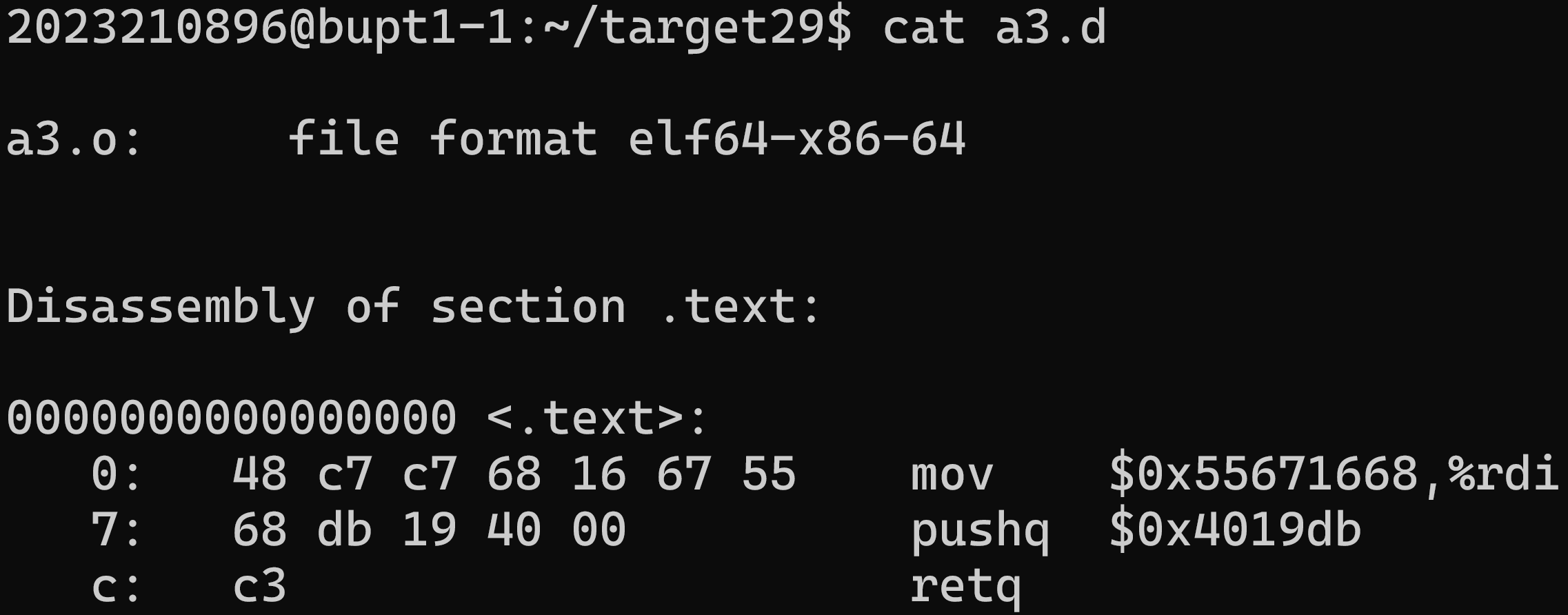
movq $0x55671668, %rdi //将字符串存放的位置存入%rdi寄存器中

pushq $0x4019db //向栈中压入touch3函数的首地址

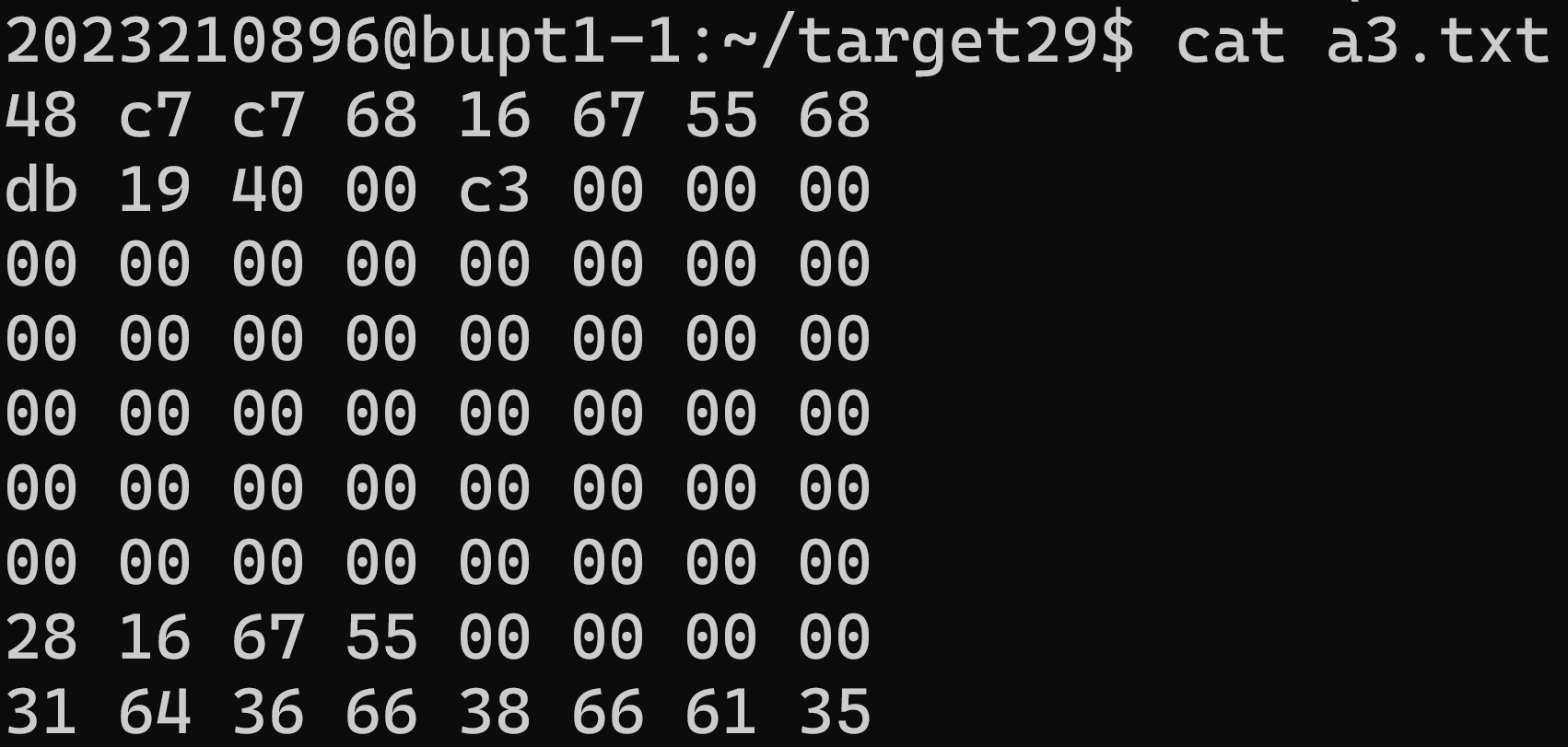
ret //调用返回地址

先用vim指令新建一个名称为a3.s的文件，将构造的指令输入，后用cat a3.s指令查看输入的结果



随后使用gcc -c a3.s 指令与objdump -d a3.s > a3.d 指令先将a3.s编译生成可执行文件，再将可执行文件反编译。后使用cat a3.d指令查看代码指令的字节级表示

最后用vim指令新建一个名称为a3.txt的文件，先顺序输入上图中的字节序列，随后用00填满剩下的字节，之后输入地址逆序输入，最后将cookie的值转化为ascii码放在末尾。使用cat a3.txt指令查看第三阶段的输入



使用攻击指令 ./hex2raw < a3.txt | ./ctarget 攻击即可

4、第四阶段

阶段四和五要求变成从文件中寻找需要的反汇编代码片段，将其地址与需要传入的函数地址或指定字符拼接后输入用于攻击。考虑构造如下指令，包含两个gadget

popq %rax //从栈中弹出返回地址 gadget1

ret //返回

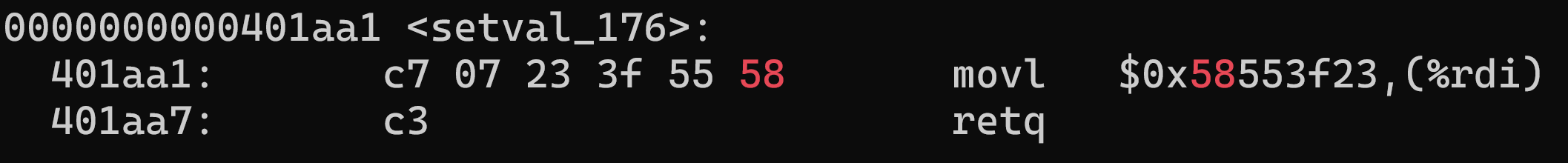
cookie值 //将%rax的值设置为cookie的值

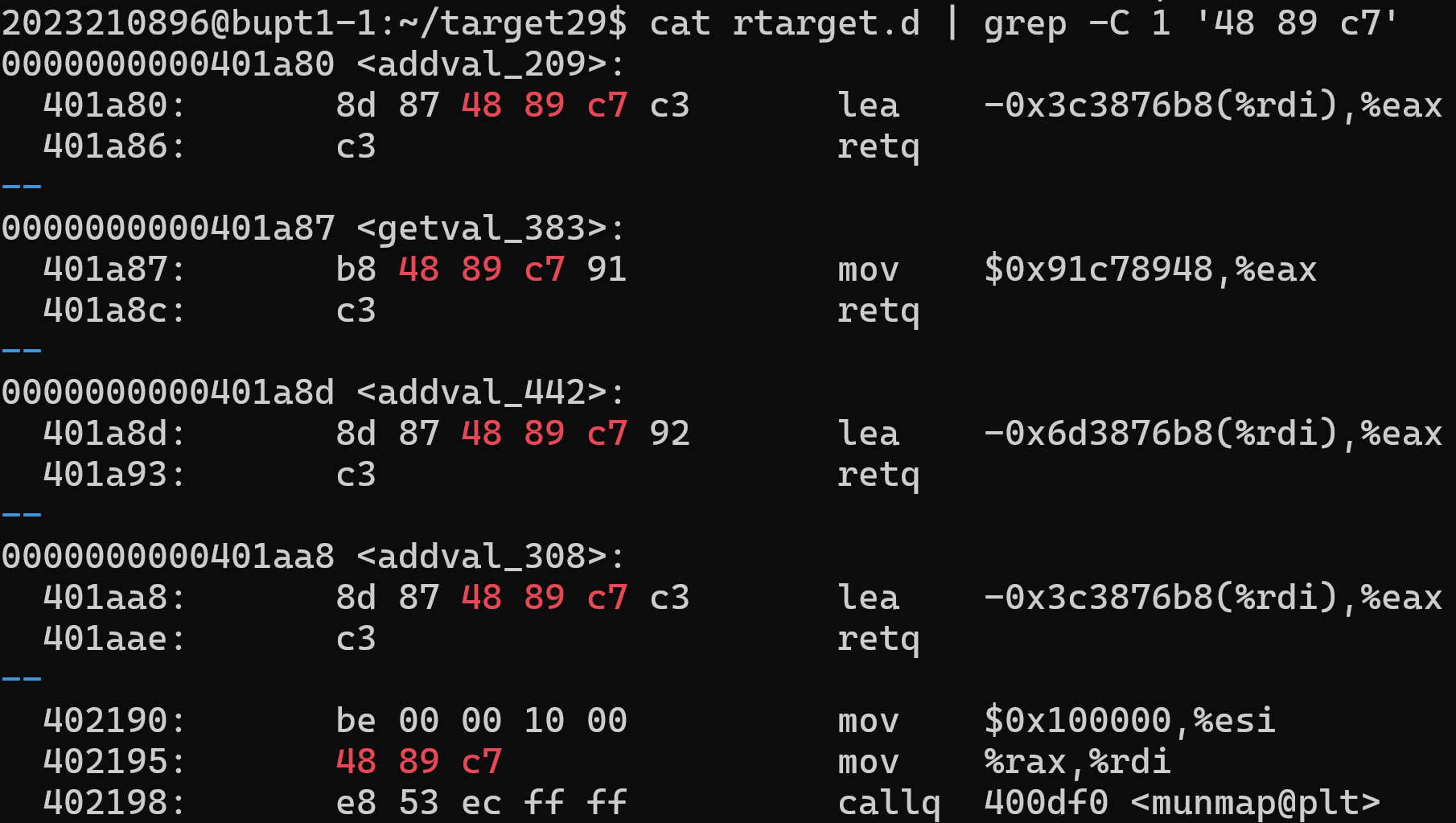
movq %rax, %rdi //将%rdi的值设置为%rax的值

Ret //从栈中弹出返回地址 gadget2

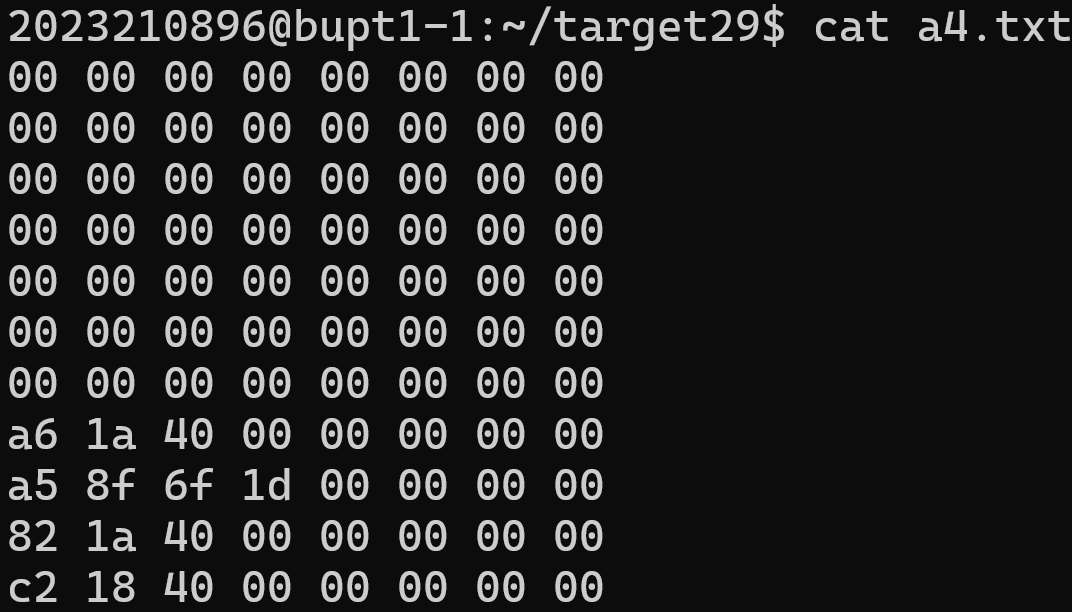
touch2函数的首地址 //调用返回的地址

查询实验指导文件中的Figure 3，发现popq %rax的字节级表示为58，movq %rax, %rdi的字节级表示为48 89 c7，故需要从rtarget的反汇编文件中寻找这两个代码片段。使用objdump rtarget > rtarget.d将rtarget函数反汇编，使用 cat rtarget.d | grep -C 1 '58' 指令与 cat rtarget.d | grep -C 1 '48 89 c7'指令查找相应内容





由以上两张图得因为401aa6地址在58后为c3，不会产生干扰，故gadget1取401aa6地址，401a82地址在48 89 c7后为c3，不会产生干扰，故gadget2取401a82地址。用vim指令新建一个名称为a4.txt的文件，先用00填满前56个字节，之后逆序输入401aa6，后逆序输入cookie的值0x1d6f8fa5，接着逆序输入401a82，最后输入touch2函数的首地址。使用cat a4.txt指令查看第四阶段的输入



使用攻击指令 ./hex2raw < a4.txt | ./rtarget 攻击即可

5、第五阶段

本阶段的任务同任务三，但不能将cookie直接存放在栈中。考虑构造如下指令

movq %rsp, %rax //48 89 e0 gadget1 将%rsp转移到%rax中

movq %rax, %rdi //48 89 c7 gadget2 将%rax转移到%rdi中

popq %rax //58 gadget3 将%rax弹出

偏移值 //cookie字符串在第121个字节，执行movq %rsp,%rax时%rsp指向第 49个字节，所以偏置量为121-49=72=0x48，存在%eax中

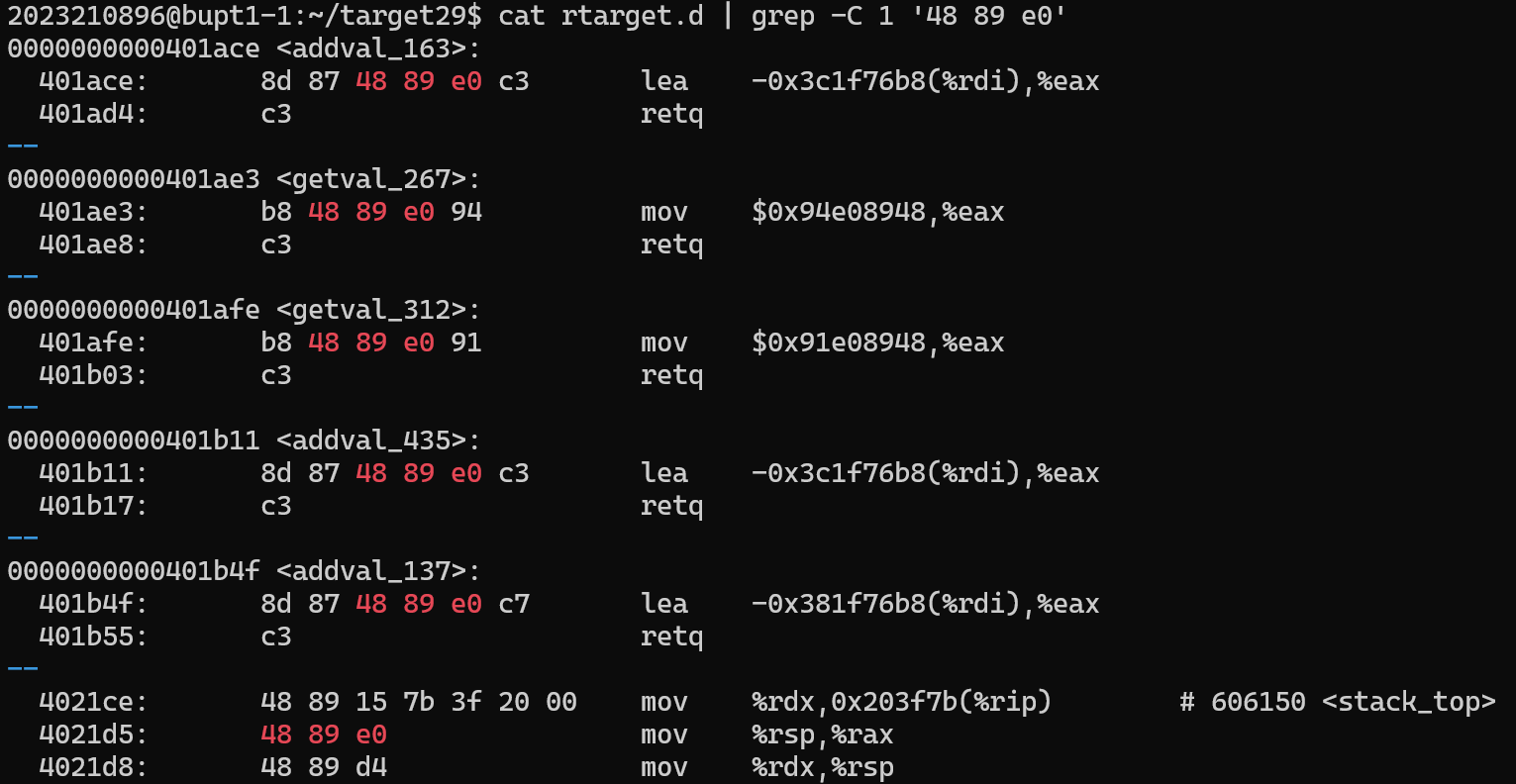
movl %eax, %esi //89 c6 gadget4 将%eax转移到%esi中

lea (%rsi, %rdi, 1) %rax //48 8d 04 37 gadget5 将%rsi+%rdi转移到%rax中

movq %rax, %rdi //48 89 c7 gadget2 将%rax转移到%rdi中

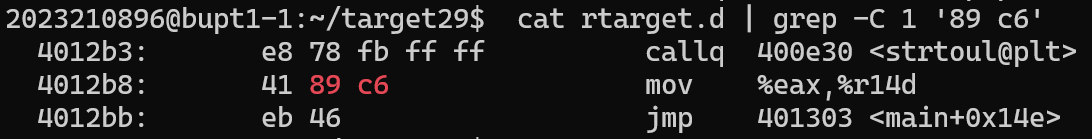
touch3函数的首地址

cookie的ascii码值

使用cat rtarget.d | grep -C 1 指令依次寻找代码片段

因为401ad0地址在48 89 e0后为c3，不会产生干扰，故gadget1取401ad0

由阶段四得gadget2取401a82，gadget3取401aa6



因为未找到89 c6代码片段，故无法直接将%eax转移到%esi中，故尝试其他途径间接将%eax转移到%esi中。经过尝试，可以通过%eax→%edx→%ecx→%esi的途径转移。故将构造更改为

movq %rsp, %rax //48 89 e0 gadget1 401ad0

movq %rax, %rdi //48 89 c7 gadget2 401a82

popq %rax //58 gadget3 401aa6

偏移值 //cookie 48

movl %eax, %edx //89 c2 gadget4

movl %edx, %ecx //89 d1 gadget5

movl %ecx, %esi //89 ce gadget6

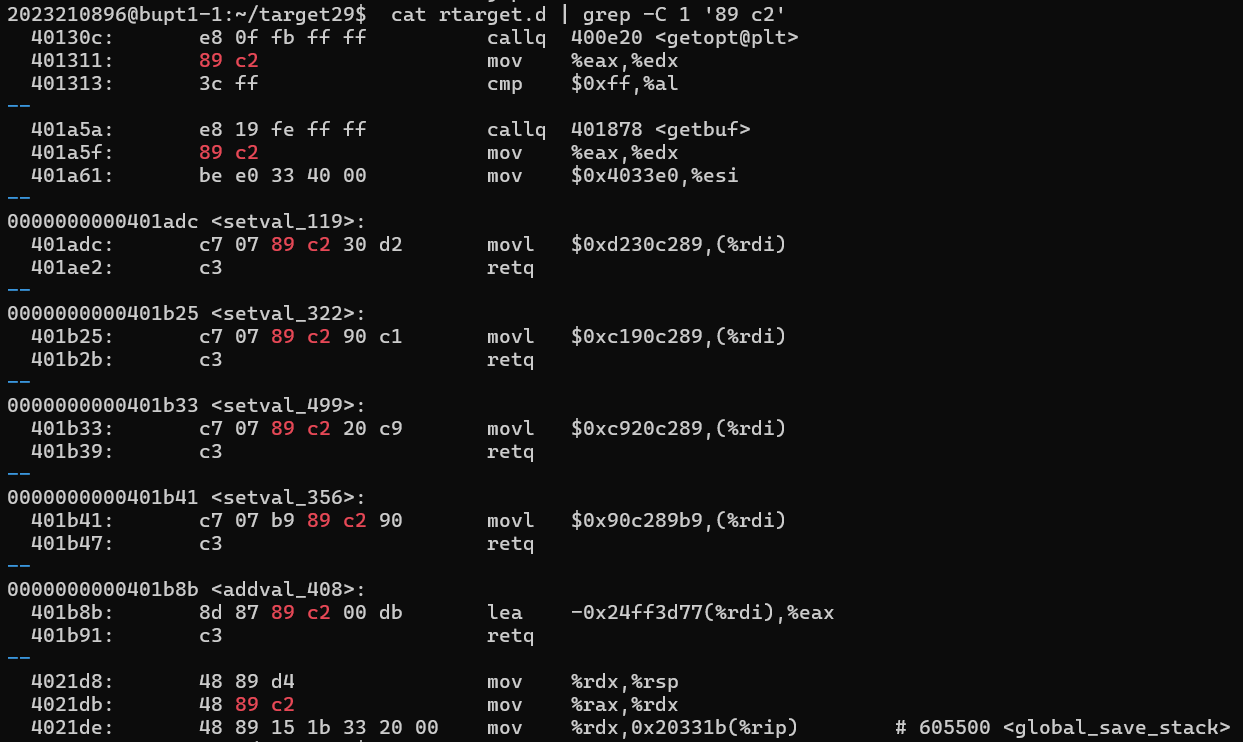
lea (%rsi, %rdi, 1) %rax //48 8d 04 37 gadget7

movq %rax, %rdi //48 89 c7 gadget2 401a82

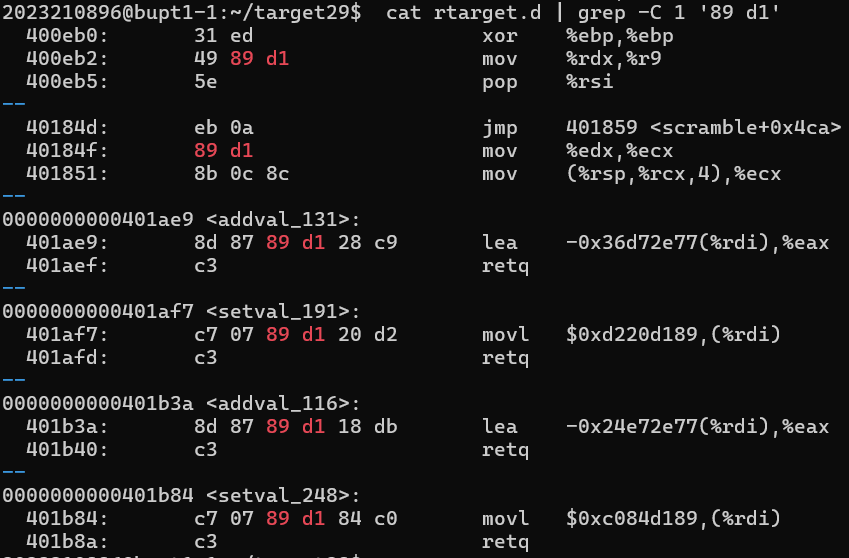
touch3函数的首地址 4019db

cookie的ascii码值 31 64 36 66 38 66 61 35

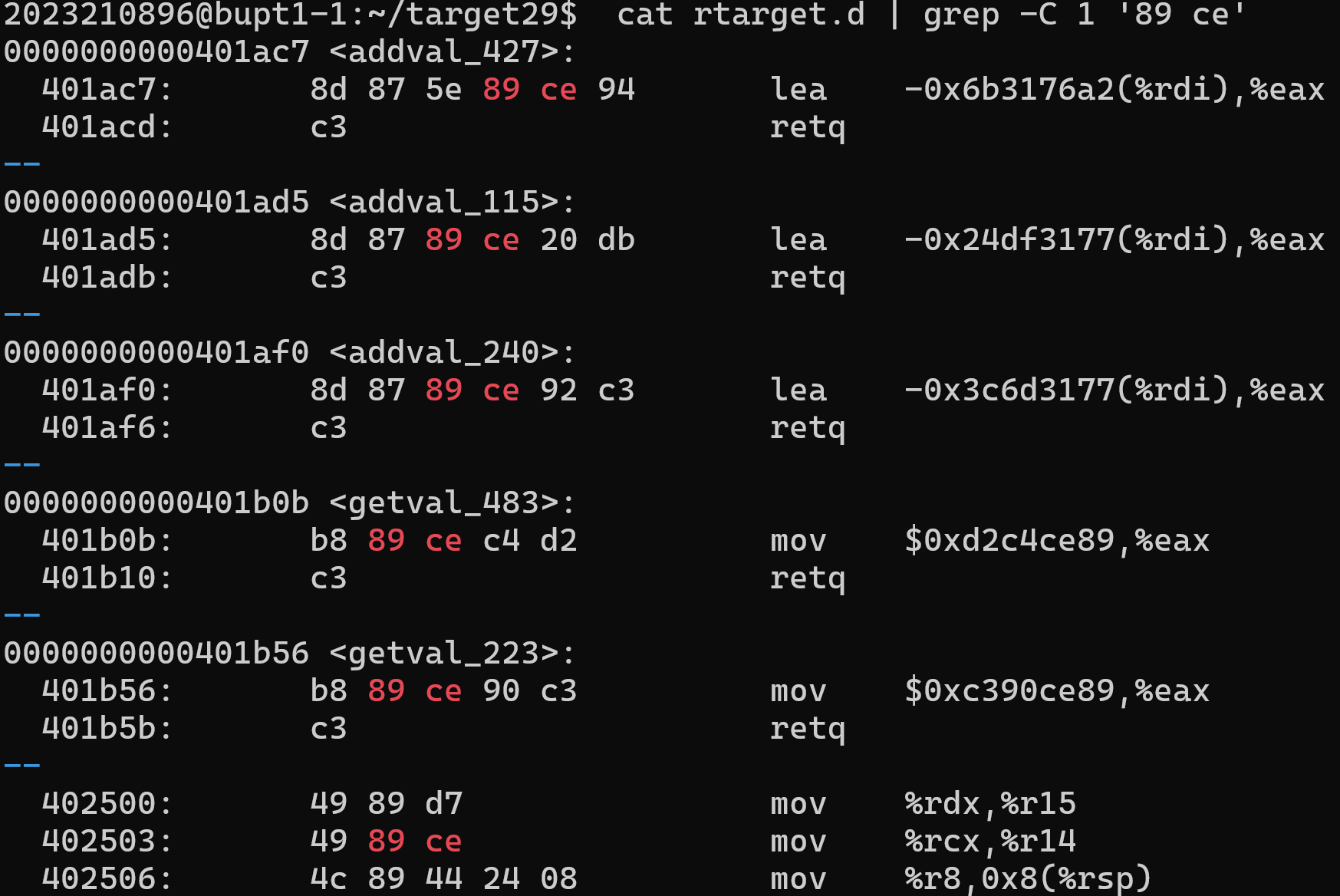
继续寻找89 c2，89 d1，89 ce，48 8d 04 37这四个代码片段



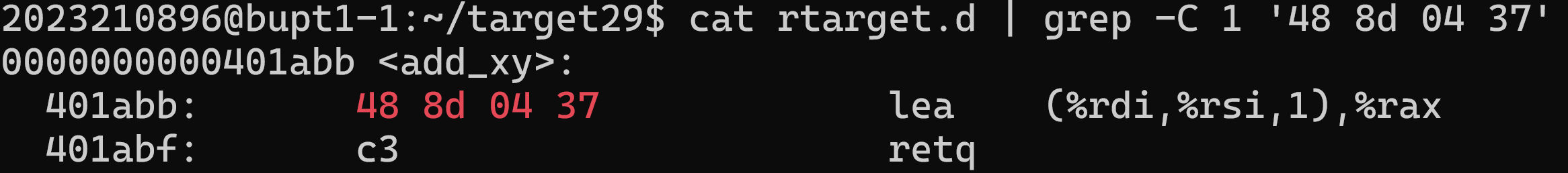
因为401b44地址在89 c2后为90 c3，不会产生干扰，故gadget4取401b44

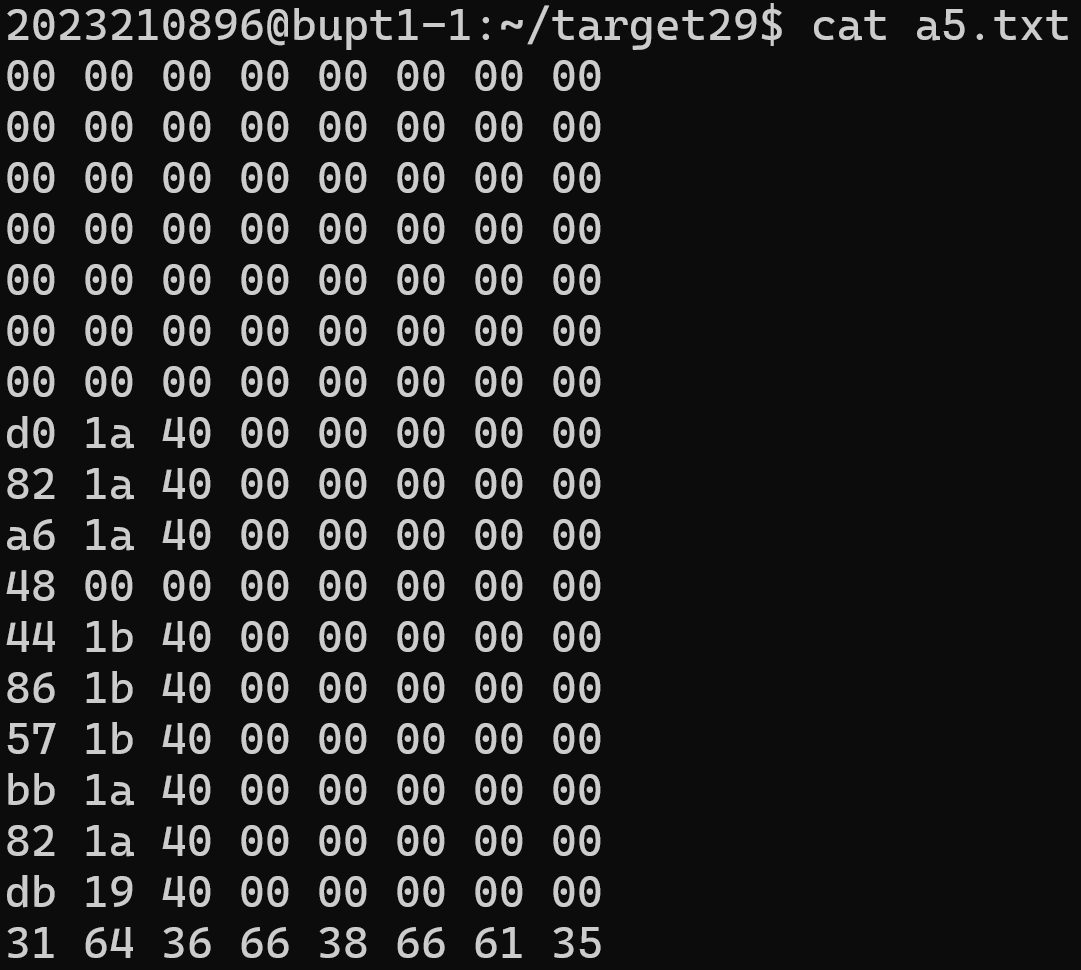


因为401b86地址在89 c2后为84 c0 c3，不会产生干扰，故gadget5取401b86

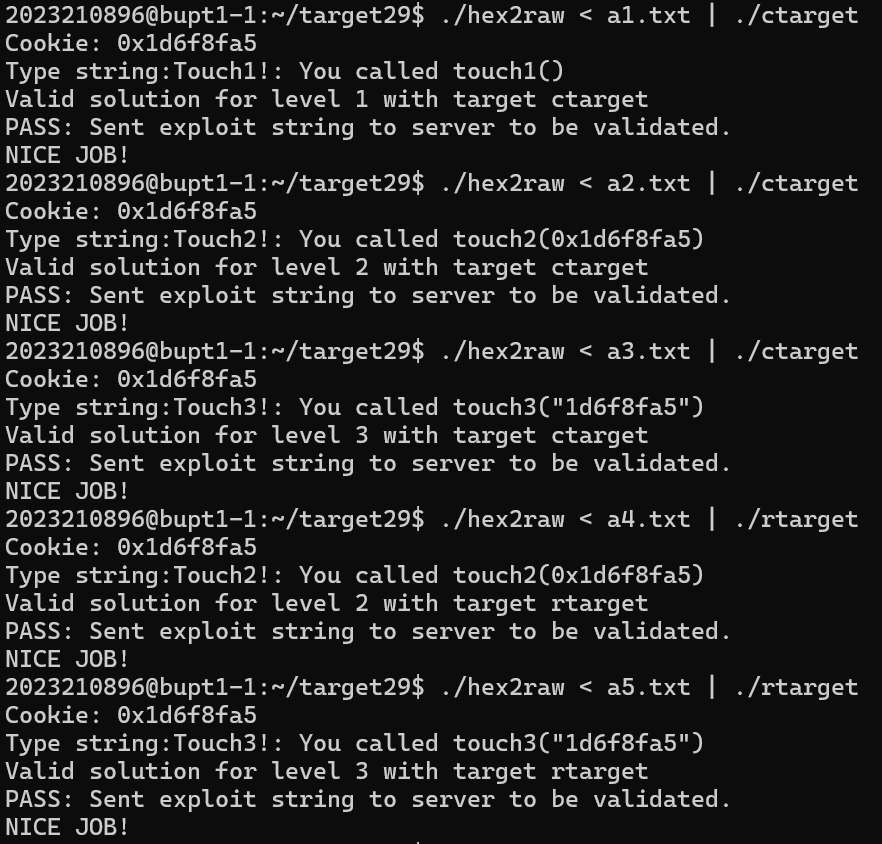


因为401b57地址在89 c2后为90 c3，不会产生干扰，故gadget6取401b57



gadget7取401abb。用vim指令新建一个名称为a5.txt的文件，先用00填满前56个字节，之后依次逆序输入gadget1~3，48，gadget4~7，gadget2，4019db，最后正序输入cookie的ascii码值。使用cat a5.txt指令查看第五阶段的输入

使用攻击指令 ./hex2raw < a5.txt | ./rtarget 攻击即可

五、答案汇总与实验截图

五、总结体会

本次实验主要内容是利用栈缓冲区实施攻击，通过先将缓冲区用00填满之后输入指令的方式对程序展开攻击。我遇到的第一个问题是全英文的实验指导手册难以理解，机翻过于晦涩难懂导致先要自己去翻译理解每一个阶段的具体内容。之后则是最后一个阶段的寻找%eax间接转移到%esi路径的过程，一开始我试图一步从%eax转移到%esi，结果没有找到代码片段；之后又试图从%eax转移到%ecx，结果在第二步又没有找到路径，最后才找到了%eax→%edx→%ecx→%esi的这一途径。在不断地寻找代码片段的过程中，常常会出现一个gadget有多种匹配的情况，此时因为无法判断哪种条件符合条件，往往是在所有gadget选择好后尝试运行失败，这样只能一个一个尝试。

通过这次实验，我对反汇编代码的生成的内在生成逻辑有了更深刻的理解，不仅能将汇编语言翻译成C语言，还能从C语言反汇编成汇编语言并生成字节级表示。同时，这次攻击实验也告诉我在今后设计程序时要注意防范潜在的缓冲区溢出攻击。第五阶段的巧妙地使用偏移量来确定cookie的地址的方法也让我学到了很多。