**深 圳 大 学 实 验 报 告**

**课 程 名 称： 计算机系统(2)**

**实验项目名称： 缓冲区溢出攻击实验**

**学 院： 计算机与软件学院**

**专 业： 计算机科学与技术**

**指 导 教 师： 刘刚**

**报告人： 吴嘉楷 学号： 2022150168 班级： 国际班**

**实 验 时 间： 2024年5月17日 ~ 5月30日**

**实验报告提交时间： 2024年5月20日**

**教务处制**

**一、 实验目标：**

1. 理解程序函数调用中参数传递机制；
2. 掌握缓冲区溢出攻击方法；
3. 进一步熟练掌握GDB调试工具和objdump反汇编工具。

**二、实验环境：**

1. 计算机（Intel CPU）
2. Linux 64位操作系统
3. GDB调试工具
4. objdump反汇编工具

**三、实验内容**

本实验设计为一个黑客利用缓冲区溢出技术进行攻击的游戏。我们仅给黑客（同学）提供一个二进制可执行文件bufbomb和部分函数的C代码，不提供每个关卡的源代码。程序运行中有3个关卡，每个关卡需要用户输入正确的缓冲区内容，否则无法通过管卡！

要求同学查看各关卡的要求，运用**GDB调试工具和objdump反汇编工具**，通过分析汇编代码和相应的栈帧结构**，**通过缓冲区溢出办法在执行了getbuf()函数返回时作攻击，使之返回到各关卡要求的指定函数中。第一关只需要返回到指定函数，第二关不仅返回到指定函数还需要为该指定函数准备好参数，最后一关要求在返回到指定函数之前执行一段汇编代码完成全局变量的修改。

实验代码bufbomb和相关工具（sendstring/makecookie）的更详细内容请参考“实验四 缓冲区溢出攻击实验.pptx”。

本实验要求解决关卡1、2、3，给出实验思路，通过截图把实验过程和结果写在实验报告上。

**四、实验步骤和结果**

**前置工作1**

因为本次实验用到的可执行文件是32位，而实验环境是64位的，需要先安装一个32位的库，在root权限下安装如下图所示：

（输入apt install lib32ncurses5-dev lib32z1）

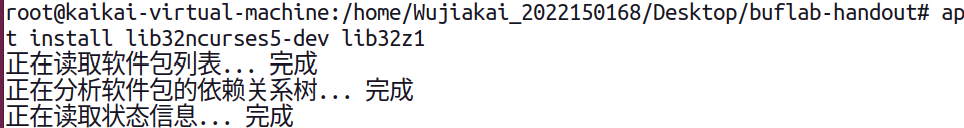


图1 安装32位的库

**前置工作2**

由于“makecookie”和“sendstring”两个包已经存在于我们获取到的“buflab-handout”文件夹里面，于是我们无需手动再去安装了。

但我们仍需要在root权限下安装“sendmail”，否则我们进行攻击输入时会报错：（使用apt install 指令安装即可）

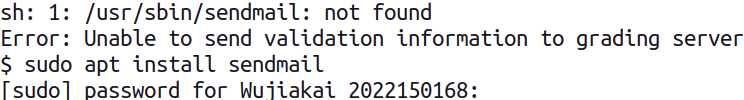


图2.1 安装sendmail

此外，我们还需要使用makecookie生成个人的身份标识：

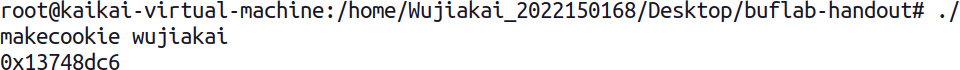


图2.2 使用makecookie生成身份标识

由图可见，我们生成的身份标识cookie为： 0x13748dc6，此标识码将在我们后续的解题中被使用！

**步骤0 分析getbuf函数**

首先利用反汇编命令查看getbuf函数的汇编代码，以便分析getbuf在调用<Gets>时的栈帧结构，汇编代码如下图1所示：

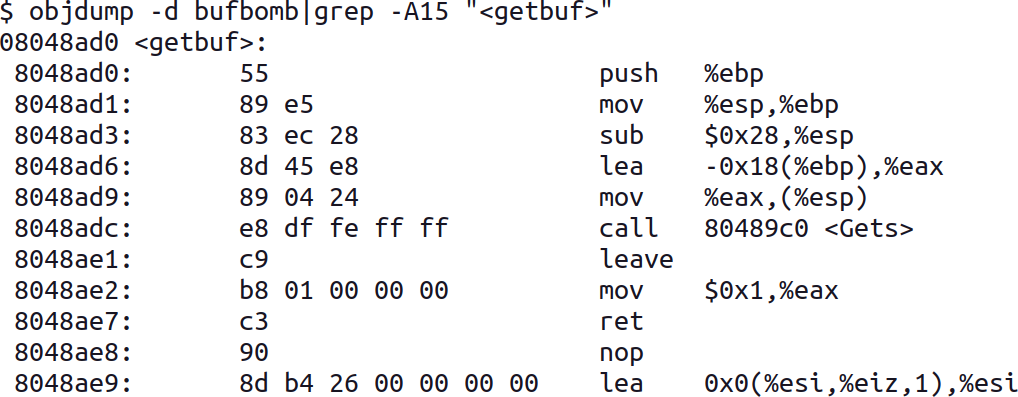


图3 getbuf函数的汇编代码

其次，分析函数的汇编代码可知，函数首先将寄存器ebp的值压栈保存，然后把esp的值复制给ebp，让esp减40开辟栈空间，再吧ebp即原esp的值减24赋值给寄存器eax（作为函数的第一个参数），进而给寄存器esp，然后调用Gets函数读入缓冲区的数据，最后返回立即数1。

（补充：在32位操作系统中，一个地址块占32/8 = 4个字节，寄存器%eax可以作为第一个参数传递给函数）

同时，系统函数gets()未进行缓冲区溢出保护。其代码如下：

int getbuf()

{

char buf[12];

Gets(buf);

return 1;

}

最后，我们画图来帮助我们理解getbuf函数：



图4 getbuf函数的栈帧结构图

为了破坏getbuf函数的返回地址，使之无法返回到test函数中，我们需要先填充28个字节空间（1个字节可以存两个十六进制数），然后输入我们想要的返回地址，总长度为64个十六进制数，即可达到目的！

**步骤1 返回到smoke()**

**1.1 解题思路**

我们的目标是使getbuf()返回时，不返回到test()，而是直接返回到指定的smoke()函数。

为此，我们可以通过构造并输入大于getbuf()中给出的数据缓冲区的字符串而破坏getbuf()的栈帧，替换其返回地址，将返回地址改成smoke()函数的地址。

**1.2 解题过程**

由步骤0我们对getbuf函数的分析可知，我们需要先填充56个十六进制数，之后我们输入的数据就会覆盖返回地址，于是我们后输入的8个十六进制数就是我们想要让getbuf（）返回的地址。

我们想让getbuf（）返回到smoke函数，那么首先就想要知道smoke函数的地址！

1. 查看smoke函数的起始地址



图5 反汇编查看smoke函数的起始地址

通过反汇编指令获得汇编代码后，查看到smoke函数的地址图5所示，为0x0848eb0。

1. 构造攻击输入语句

由于Intel的存储模式为**小端模式**，即数据的低位部分存储在低地址，所以在构造攻击语句时，需要变为b08e0408。

于是我们需要构造一个总长度为64位的十六进制串：

前56位任意 + b08e0408

例如：01234567891123456789212345678931234567894123456789512345b08e0408

1. 把答案写入attack1.txt文件

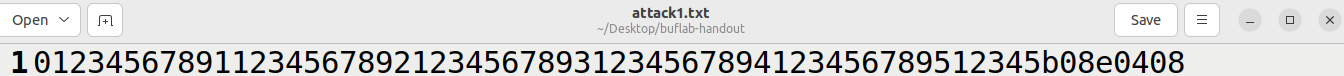


图6 attack1.txt文件内容

**1.3 最终结果截图**

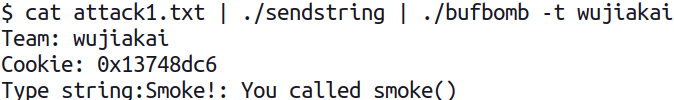


图7 通过关卡1

**步骤2 返回到fizz()并准备相应参数**

**2.1 解题思路**

我们的目标是使getbuf()返回时，不返回到test()，而是直接返回到指定的fizz()函数。

为此，我们可以通过构造并输入大于getbuf()中给出的数据缓冲区的字符串而破坏getbuf()的栈帧，替换其返回地址，将返回地址改成fizz()函数的地址。而通过对代码的分析，我们发现与一个地址的值（0x804a1d4）进行了比较，使用gdb查看该地址的值，得知其为用户名的cookie值。

破解的思路与第一关类似，只是增加了一个传入参数，所以在读入字符串时，还要把fizz()函数读取的参数的地址替换成自己的cookie值。

**2.2 解题过程**

1. **分析fizz函数读取的参数地址**

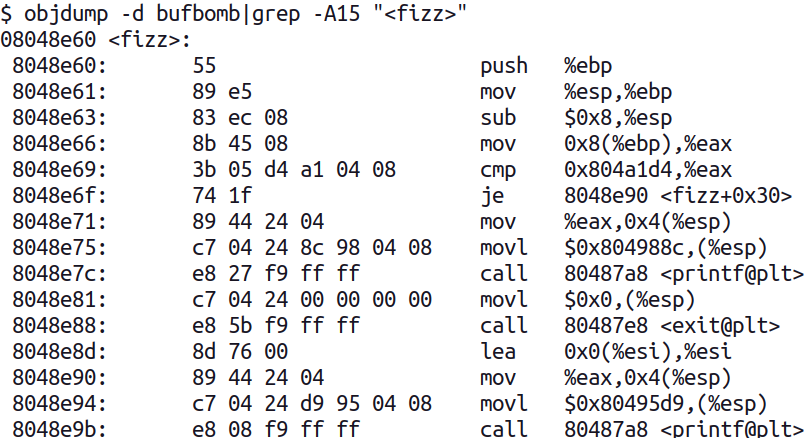


图8 fizz函数的汇编代码

在fizz函数中，首先将寄存器ebp的值压栈保存，然后把栈顶指针寄存器esp的值复制给ebp，然后esp减8以开辟新的空间，再将寄存器%ebp加8的地址值（即原%esp+8）作为传入的参数，也就是说，传入的参数应该存储在%ebp ＋ 8地址处的！

其次，程序会根据参数值与0x804a1d4处的值进行比较，只有值相等时，才可以进行正常的功能。

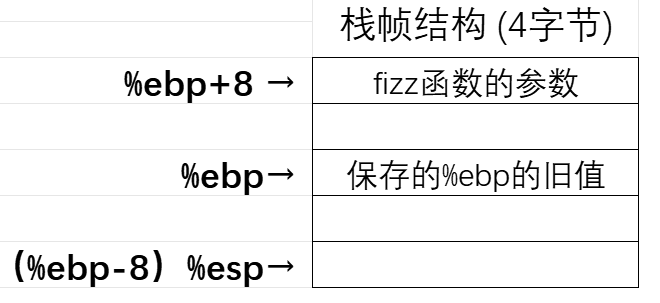


图9.1 重要的栈帧结构

我们使用gdb调试工具，输入“p （char\*） 0x804a1d4”指令查看0x804a1d4处的信息：



图9.2 0x804a1d4地址处的值

1. **分析如何将该参数地址的值替换为cookie**

对应到getbuf函数中的具体栈帧结构如下：

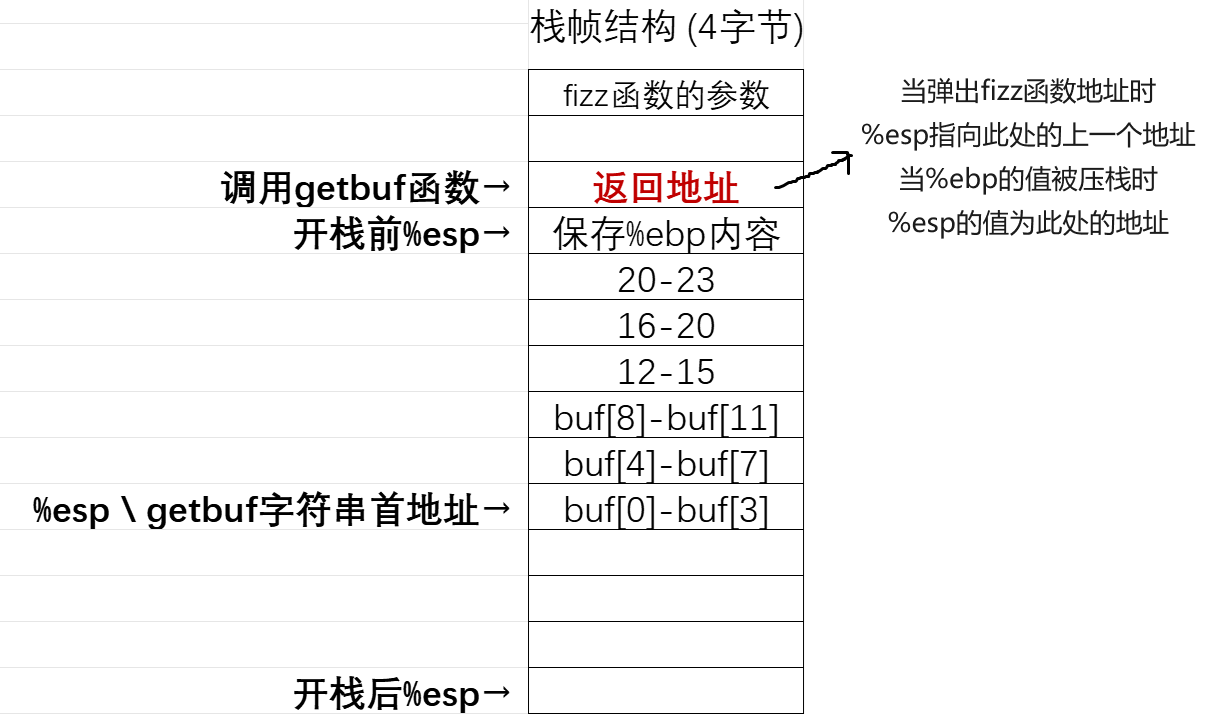


图9.3 具体栈帧结构

由以上结构不难判断出，我们需要读入buf的字符串为“28个任意字符+fizz()的地址+4个任意的字符+自己的cookie值”

1. **构造攻击输入语句**

在我们使用objdump反汇编工具查看fizz函数的信息时，我们已经知道fizz函数的地址为**08048e60**，而使用makecookie工具我们可以得到我们的cookie值为：13748dc6。

由于Intel的存储模式为**小端模式**，即数据的低位部分存储在低地址，所以在构造攻击语句时，需要分别变为608e0408和c68d7413。

于是我们需要构造一个总长度为80位的十六进制串：

前56位任意 + 608e0408 + 8个任意 + c68d7413

例如：01234567891123456789212345678931234567894123456789512345608e040812345678c68d7413

1. **把答案写入attack2.txt文件中**

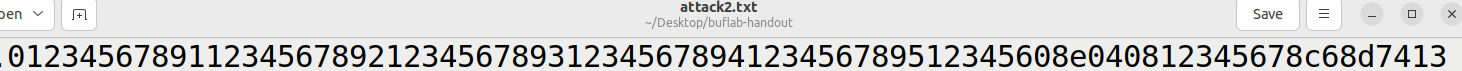


图10 attack2.txt文件内容

**2.3 最终结果截图**

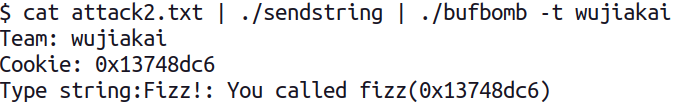


图11 通过关卡2

**步骤3 返回到bang()且修改global\_value**

**3.1 解题思路**

这一关要求先修改全局变量global\_value的值为自己的cookie值，再返回到band()。为此需要先编写一段代码，在代码中把global\_value的值改为自己的cookie后再返回到band()函数。

将这段代码通过GCC产生目标文件后读入到buf数组中，并使getbuf函数的返回到buf数组的首地址，这样程序就会执行我们写的代码，修改global\_value的值并调用band()函数。

我们的目标是使getbuf()返回时，不返回到test()，而是直接返回到我们编写在buf数组中的自定义代码块。

破解的思路与第一关类似，只是前28个字符不能全部是任意的，需要部分改成我们自定义的代码，且返回地址需替换成buf数组的首地址。

**3.2 解题过程**

1. **分析bang函数，确定global\_value**

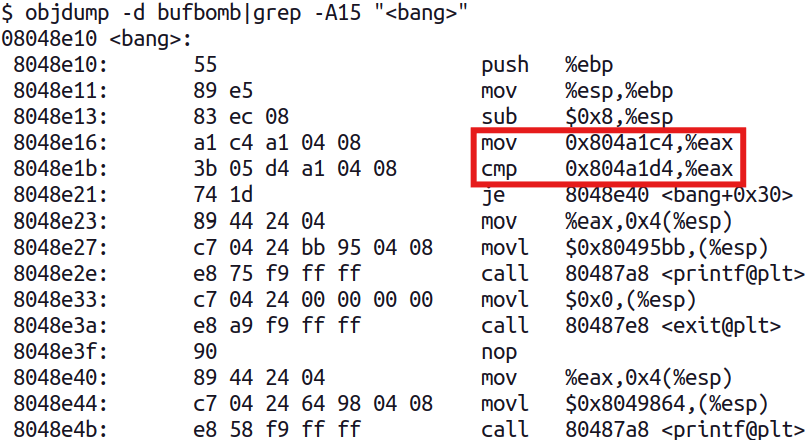


图12 bang函数的汇编代码

分析代码可知，在bang函数中，将地址0x804a1d4中存放的值与地址0x804a1c4的值进行了比较，相同则进行跳转，正常执行函数功能。

于是我们使用gdb工具查看两处地址中的内容：

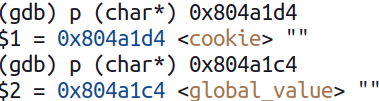


图13 两处地址的内容

由图可见，0x804a1d4中的值为我们的cookie，0x804a1c4中的值为global\_value。于是，我们应当**修改global\_value的值为我们的cookie值**！

1. **分析如何修改global\_value并返回到bang函数**

由于global\_value的值可能存在于全局数据区中，并不存在与用户栈中，况且它的地址也可能于buf的地址相距甚远。于是，我们无法直接通过缓冲区溢出来修改global\_value的值，而需要间接使用自定义代码去修改它。

而如何让程序执行我们的自定义代码，那就很简单了！我们只需要模仿关卡一的思路，将**getbuf（）的返回地址修改成我们自定义代码段的起始地址**即可。

那么自定义代码如何侵入到程序中，那就需要读入缓冲区时依靠buf数组存入即可。由于buf数组首地址距离getbuf（）的返回地址有28个字节空间，于是，该**自定义代码最多可以用28个字节的长度**！

最后，我们只需要在**自定义代码末尾无条件跳转至bang（）函数**即可。

因此，我们明确了目的：**读入缓冲区时需要输入我们的自定义代码，并修改getbuf（）的返回地址为buf数组首地址。**

1. **编写自定义汇编代码并得到其十六进制串**

由上两步的分析可知，自定义代码中，需要将地址0x804a1d4的值复制到0x804a1c4中，并且在代码末尾无条件跳转到bang（）函数处（0x08048e10)。

由于jmp指令无法使用立即数跳转，于是我们可以将0x08048e10先存到寄存器%rdx中，在利用寄存器进行跳转。

我们将汇编代码保存在mycode.s文件中，内容如下：

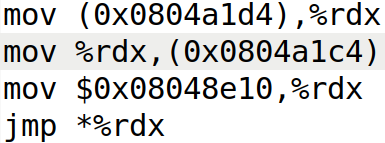


图14 自定义汇编代码内容

由于我们希望构造一个十六进制的输入语句，于是还需要把上述代码转换为十六进制，这里可以使用gcc ＋ objdump反汇编工具来实现：



图15 得到反汇编的文件

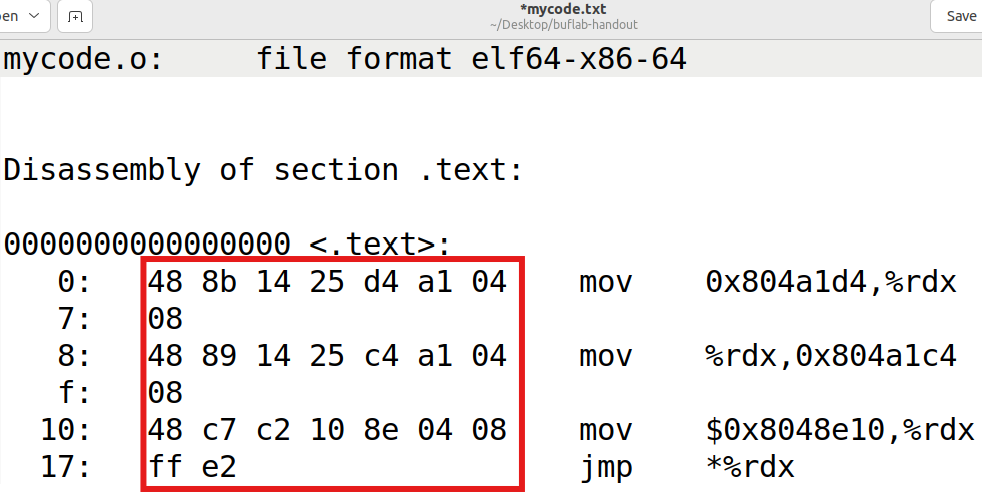


图16 通过反汇编文件查看十六进制串

由图可知，自定义代码的十六进制串为：

488b1425d4a1040848891425c4a1040848c7c2108e0408ffe2

1. **确定返回地址（即buf数组首地址）**



图17 getbuf函数调用后的栈帧结构

通过图17的栈帧结构分析，%ebp-24即为字符串的首地址。由于用户栈的内存地址具有随机性，为了确定%ebp的地址，我们首先在root权限下**关闭Linux内存地址随机化**，如图18所示：

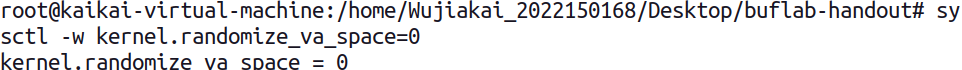


图18 关闭Linux内存地址随机化

然后，我们用gdb断点调试查看执行getbuf()时寄存器ebp的值：



图19 查看寄存器%ebp的值

由图19可见，寄存器%ebp的值为0xffffb508，于是buf数组首地址为%ebp - 0x18 = 0xffffb508 - 0x18 = 0xffffb4f0

1. **构造攻击输入语句**

已得，自定义代码的十六进制串为：

488b1425d4a1040848891425c4a1040848c7c2108e0408ffe2

buf数组首地址为：ffffb4f0

由于Intel的存储模式为**小端模式**，即数据的低位部分存储在低地址，所以在构造攻击语句时，ffffb4f0需要变为f0b4ffff

综上，我们需要构建一个总长度为64位的十六进制串：

488b1425d4a1040848891425c4a1040848c7c2108e0408ffe2 + 6个任意十六进制数 + f0b4ffff

1. **把答案写入attack3.txt文件中**

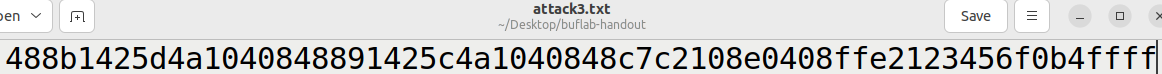


图20 attack3.txt文件内容

**3.3 最终结果截图**

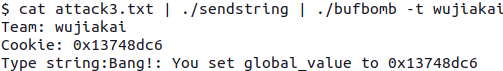


图21 通过了第三关

**五、实验总结**

**问题1：**使用指令“apt install lib32ncurses5”安装32位的库时出现报错信息：Unable to locate package lib32ncurses5

**解决：**这是由于不同版本的Kali Linux，它的源软件包名字会发生一些局部的变化，这都是很正常的。把“lib32ncurses5”换成“lib32ncurses5-dev”即可

**问题2**：在使用攻击语句破解关卡时出现报错信息：“sh: 1: /usr/sbin/sendmail: not found Error: Unable to send validation information to grading server”

**解决**：在root权限下下载安装“sendmail”即可

**问题3**：在编译mycode.s文件时出现报错：Error: unsupported syntax for `jmp'

**解决**：原因是我们的gcc默认按照64位进行编译，而我们的汇编代码中使用的是32位的寄存器%edx，我们只需把寄存器%edx更换为%rdx即可。

**收获1**：使用“gedit 文件名”命令不仅可以用图形化界面打开已存在的文件，还可以创建新文件并打开

**收获2：**objdump -d 二进制文件名|grep -A15 "<函数名>" 指令可以查看文件中对应函数的反汇编代码信息

**收获3**：cat 十六进制文件名 | ./sendstring | ./需要输入的文件名 指令可以将十六进制数据转成字符串在输送给目标文件

**收获4：**jmp无条件指令没有立即数跳转模式，需要使用寄存器或地址标签来进行跳转

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字： 刘刚    2024年 5月 日 |
| 备注： |