"The Android Project"
Incident Response
And Forensics

A project by -
Dhruv Mohindra
Incident Response
(Spring 2008)

INDEX

1. Introduction to Android

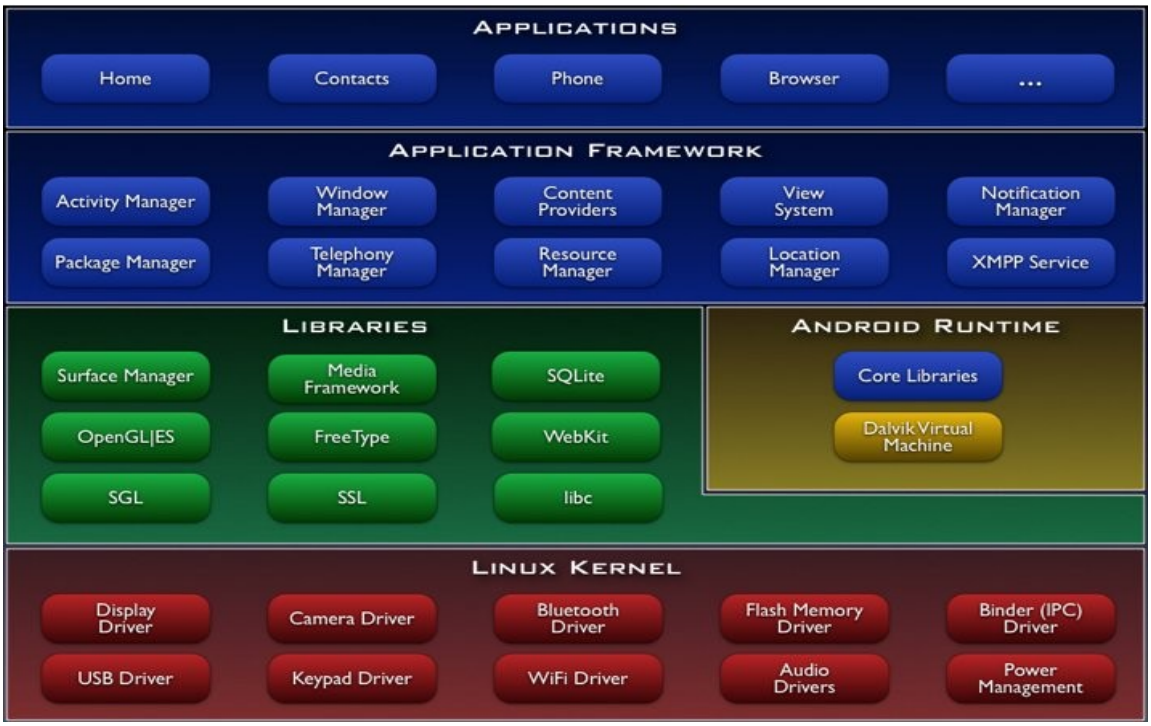
The open handset alliance, a group of more then 30 technology and mobile companies, was formed in 2007 to foster creativity and innovation in the mobile arena. This group includes members like Sprint, T-Mobile, Intel, Broadcom, Motorola, Samsung, Google, eBay, amongst others.

The main motive of this move was to facilitate openness in mobile software and thus give customers a diversified software base which is not only less expensive but also convenient and easy to use. The Android platform has emerged from such a move. Android is an Operating System that is designed to be used with a variety of handsets that will be launched in the second half of 2008.

Currently an SDK is hosted by Google, which is a leading player in the initiative. This SDK allows creation of various Java based programs by using a special Eclipse plug-in that interfaces with the kit [1]. Android is not only an OS but also includes a middleware and an array of applications for the users. Some of the supported features are – a Dalvik Virtual Machine, built in browser and database support, media, camera, GPS, map and other features.

2. Overall Architecture

The Android platform is divided into the following components -
1. Applications 2. Application Framework 3. Libraries 4. Android Runtime 5. Linux Kernel



Applications include a bundle of programs like the SMS manager, calendar, web-browser, contact

manager etc. which will be shipped as core programs, with the handset. The application framework is designed to promote reuse of components so that any application can export its interfaces to other applications with the intent of publishing. These include Activities, Content Providers, Activity Manager and so on. Libraries are meant for developers and include a BSD derived System-C library. The Android Runtime provides most of the Java programming features. Android makes use of Linux 2.6 kernel for low level system processes such as memory management, process management and network stack.

3. Security Architecture

Security between applications is enforced at process level through Linux facilities such as user and Group Ids that are unique to applications. A permission mechanism exists which helps refine and add more granularity to the security model.

3.1 User IDs and file accesses

An Android application is called app-name.apk and is given a unique user ID so that a sandbox is created that prevents it from interfering with other applications and gives it the independence. This is decided at install time and remains the same throughout the life of the application. If two programs need to share process code, the sharedUserId in AndroidManifest.xml's manifest tag has to be specified accordingly.

3.2 Permissions

By default, no permissions are associated with an application. At least one <user-permission> tag must be included that declares the permissions that an application needs. Here is an example -
<uses-permission android:name="android.permission.RECEIVE_SMS" />
which allows monitoring of incoming SMS messages. Permissions that are requested are granted based on two things -

1. Some trusted authority
2. User interaction

It should be noted that once the application is installed, no checks are done at runtime. In certain cases a SecurityException may not even be thrown. The Manifest.permission file lists all the permissions in Android.

A particular permission may be enforced at a number of places during your program's operation [2]:

- At the time of a call into the system, to prevent an application from executing certain functions.
- When starting an activity, to prevent applications from launching activities of other applications.
- Both sending and receiving Intent broadcasts, to control who can receive your broadcast or who can send a broadcast to you.
- When accessing and operating on a content provider.
- Binding or starting a service.

3.3 Miscellaneous

It is possible that both the sender and receiver, while broadcasting intents, can require a permission. In this case, both permission checks should pass for the target to get the message. Both a label and description are important while specifying permissions.

4. The Proof of Concept Attack

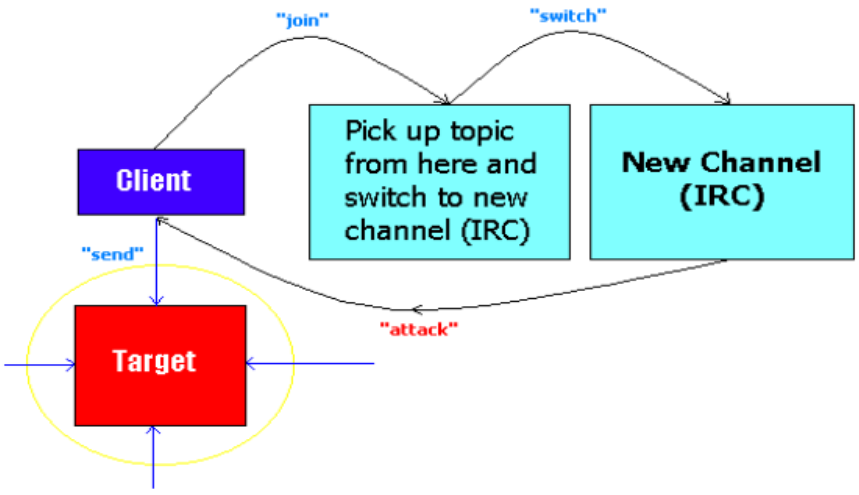
A novel attack that uses the Android Platform to form a *botnet* and then block until the attacker issues a command, has been proposed in this text. The following sections will describe this in detail.

4.1 Distributed Denial of Service Attacks

The goal of a Denial of Service attack, popularly abbreviated as DoS is to disrupt some legitimate activity such as browsing Web pages, listening to an online radio, transferring money from your bank account, or even docking ships communicating with a naval port. The target application, machine, or network spends all of its critical resources on handling the attack traffic and cannot attend to its legitimate clients.

The destination address field of a UDP packet contains the address of the target which is to be attacked. The length of the UDP is generally fixed but to circumvent filtering programs it can be varied. If large packets are streamed higher than the maximum bandwidth available to the target, then a DDoS occurs.

4.2 The Attack



A piece of Java code [3] was written which when runs on the SDK goes through the following sequence of steps -

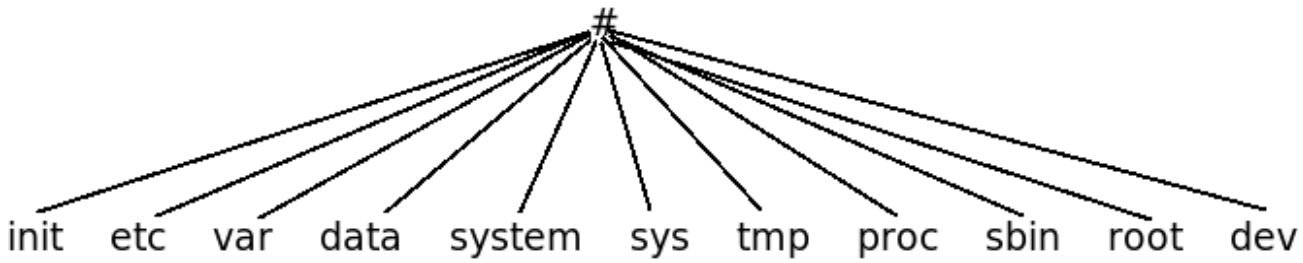
1. It connects itself to a predefined Internet Relay Chat (IRC) channel (#report) when the network interface is available. On mobile phones, this is typically always ON unless the user disables the connection.

2. After joining the channel #report, it picks the topic of channel that is specified by the attacker. After this, it makes a transition to a new channel whose name was specified in the topic of previous channel. Let this new channel be #newchannel for simplicity.
3. If the attacker pleases these connections can be encrypted, for added stealth. The bot now waits for an attacker to issue a command. The mobile phone has been completely compromised at this point.
4. The attacker can chat as usual making it seem like a normal IRC channel. However, if at any time he includes the word “attack” or a custom chosen word, the bot gets the go ahead to attack a website that the attacker may choose. This takes the form of a UDP flood against the target.
5. The attacker may stop the attack at any point of time by issuing appropriate commands.

5. Live Incident Response

5.1 File System Architecture

It is important to understand the workings of the Filesystem if one has to perform any kind of incident response. The diagram given below illustrates the root directories. A high level description follows.



The features discussed below are largely undocumented and observations of the author. Only the directories that are important from response point of view have been highlighted.

1. /etc : This folder contains the USB device configuration file, file to control the systemwide message bus (system.conf), ppp connection status files and most importantly the hosts file. The hosts file is important since it can give details about what computers have been added to the local network.

2. /data: This directory contains all databases and caches. The packages.xml file contained within the /data/system subdir contains information about what packages are installed on the system. The dalvik-cache file contains application specific cache information. These are .dex files and can be disassembled using dexdump, as will be described later.

   There is a Digital Rights Management (drm) folder which is natively empty but in future it may be used by key applications. The log subdir is also empty by default but it may be possible for logging applications to write to this folder in the future.

   The misc/location/gps directory has a file called location which contains the exact location of

the phone. This can be vital for recording the place where the mobile phone was seized from, in order to produce evidence in court. The data directory has all the databases which can be queries using a light-weight database utility called sqlite3. A snapshot is given below.

```
drwxrwx--x system   system           2008-01-20 21:40 android
drwxrwx--x app_0    app_0            2008-01-20 21:40 com.google.android.providers.contacts
drwxrwx--x app_0    app_0            2008-01-20 21:40 com.google.android.providers.googleapps
drwxrwx--x app_0    app_0            2008-01-20 21:40 com.google.android.providers.im
drwxrwx--x app_0    app_0            2008-01-20 21:40 com.google.android.providers.media
drwxrwx--x system   system           2008-01-20 21:40 com.google.android.providers.settings
drwxrwx--x phone    phone            2008-01-20 21:40 com.google.android.providers.telephony
drwxrwx--x app_1    app_1            2008-01-20 21:40 com.google.android.browser
drwxrwx--x system   system           2008-01-20 21:40 com.google.android.development
drwxrwx--x app_2    app_2            2008-01-20 21:40 com.google.android.contacts
drwxrwx--x app_3    app_3            2008-01-20 21:40 com.google.android.fallback
drwxrwx--x app_0    app_0            2008-01-20 21:40 com.google.android.googleapps
drwxrwx--x app_4    app_4            2008-01-20 21:40 com.google.android.home
drwxrwx--x app_0    app_0            2008-01-20 21:40 com.google.android.xmppService
drwxrwx--x app_0    app_0            2008-01-20 21:40 com.google.android.maps
drwxrwx--x phone    phone            2008-01-20 21:40 com.google.android.phone
drwxrwx--x app_0    app_0            2008-01-20 21:40 com.google.android.xmppSettings
drwxrwx--x app_5    app_5            2008-01-20 21:40 com.google.android.samples
drwxrwx--x app_6    app_6            2008-01-20 21:40 com.google.android.lunarlander
drwxrwx--x app_7    app_7            2008-01-21 00:16 com.google.android.hello
drwxrwx--x app_8    app_8            2008-01-21 01:48 org.apache.gtalk
```

All the folders contain a sub folder called databases which in turn have one or more .db files. These db files can be opened with #sqlite3 filename.db and can be observed using standard SQL queries (described later).

3. /app: This folder lists all the currently installed applications (.apk format) and is one of the first few places to look for malicious programs.

4. 'timezone' file: This file lists the current timezone such as EST for Eastern Standard Time.

5.2 Useful Native Commands

This section will provide an overview of built-in system commands that can help the responser in establishing the initial chain of custody and subsequently collect evidence. More specific process, memory and network related commands will be introduced later.

5.2.1 General Commands

1. #date: This command gives the current system date. It should be the first step in incident response.

```
# date
Sat Feb 16 20:34:42 EST 2008
#
```

2. #cat uptime (in /proc): Get the time since when the system is up.

```
# cat uptime
7031.58 6771.70
#
```

3. #cat filesystems (in /proc): Display supported file systems.

5.2.2 Process, Memory and Network

1. #ps: This command lists all the running processes and is useful for detecting any rogue elements.
2. #netstat: Use this to list all the open connections and connection states. This is one of the best methods for detecting if a malicious application is trying to connect to the outside world.
3. #netcfg: This is a substitute command for *ifconfig* which does not produce any output on Android. The screenshot below lists a typical output which shows the network interface and the corresponding IP address.

```
# netcfg
gre0     DOWN  0.0.0.0         0.0.0.0         0x00000080
tunl0    DOWN  0.0.0.0         0.0.0.0         0x00000080
eth0     UP    10.0.2.15       255.255.255.0   0x00001043
lo       UP    127.0.0.1       255.0.0.0       0x00000049
```

4. #mem_profiler: This utility lists the memory that is occupied by each program. More specifics will be discussed in attack analysis.

5.3 Useful Built-in Tools

1. Dumpsys and dumpstate: They print system state information for the purpose of bug reporting. Dumpsys simply dumps the data to the screen while Dumpstate writes it to a file.

2. Showmap: This tool is like Dependency Walker or PEView. It allows one to see the dynamic files (shared objects - .so) required for an application's successful operation.

3. Logcat: It is a debug feature which displays logging information on the terminal.

4. Dd: This is like the usual Disk Dumper utility which allows making an image of a partition and then transferring it to the host computer. (dd if=src off=destination)

5.4 Attack Analysis

For analyzing the proposed attack, all the native tools were used. The subsections will describe host and network based signatures.

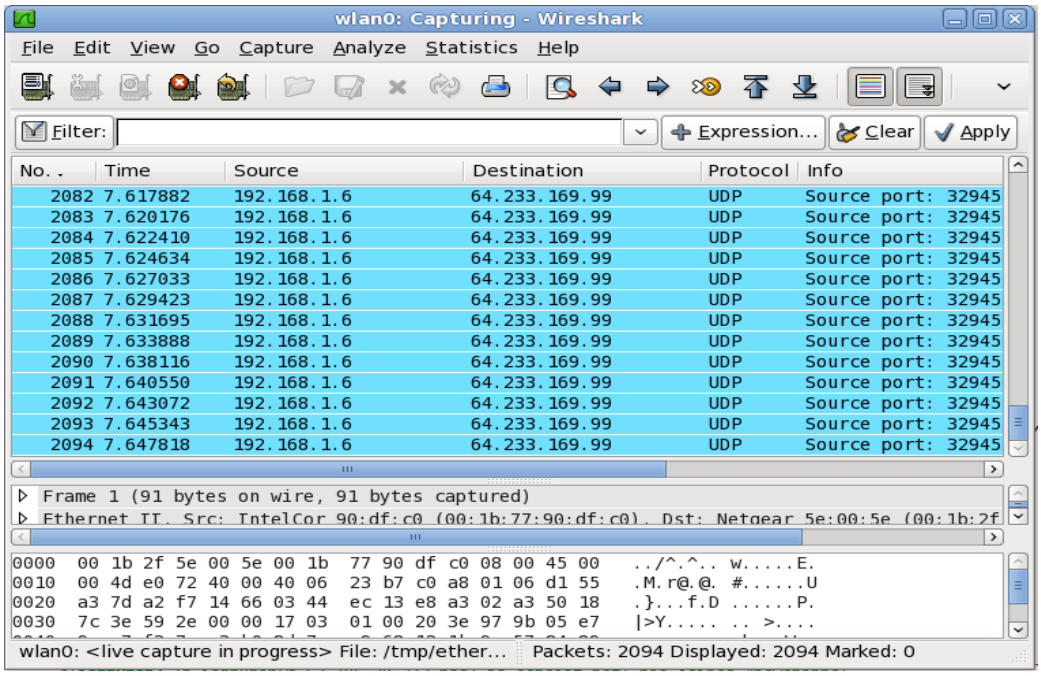
5.4.1 Network Based Signatures

1. #netstat: This command gives vital information about network connections. The screenshot below shows the output obtained when this was run on a compromised mobile phone.

```
# netstat
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:8000          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:8001          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:8002          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:8003          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:8004          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:5900          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:5037          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5555            0.0.0.0:*               LISTEN
tcp        0    158 10.0.2.15:49514         208.71.169.36:6667      ESTABLISHED
tcp        0      0 127.0.0.1:42516         127.0.0.1:8003          ESTABLISHED
tcp        0      0 127.0.0.1:8004          127.0.0.1:37350         ESTABLISHED
tcp        0      0 127.0.0.1:8003          127.0.0.1:42516         ESTABLISHED
tcp       98      0 10.0.2.15:5555          10.0.2.2:33290          ESTABLISHED
tcp        0      0 127.0.0.1:8001          127.0.0.1:52901         ESTABLISHED
tcp        0      0 127.0.0.1:37350         127.0.0.1:8004          ESTABLISHED
tcp        0      0 127.0.0.1:8002          127.0.0.1:34690         ESTABLISHED
tcp        0      0 10.0.2.15:44477         209.85.163.125:5222     ESTABLISHED
tcp        0      0 127.0.0.1:34690         127.0.0.1:8002          ESTABLISHED
tcp        0      0 127.0.0.1:52901         127.0.0.1:8001          ESTABLISHED
tcp        0      0 127.0.0.1:8000          127.0.0.1:48812         ESTABLISHED
tcp        0      0 127.0.0.1:48812         127.0.0.1:8000          ESTABLISHED
```

A suspicious TCP connection to 208.71.169.36 on port 6667 was noted as highlighted. The port 6667 is used by IRC in Internet nomenclature. The state later changed from ESTABLISHED to CLOSE_WAIT. This hinted that this connection was waiting for some input from the server.

2. Since the malicious application was running on the SDK, it was possible to sniff the packets that were being transmitted to the unknown location. Wireshark in action -



Note that a continuous stream of UDP packets is observed when the malicious code denoted by *Hello Android* is run. The destination resolves to yo-in-f99.google.com when **nslookup** is used. This means that the machine is waging a Distributed Denial of Service attack against the Google webservers.

5.4.2 Host Based Signatures

1. #ps: This command should be the first in sequence to determine if any rogue processes are running on the system. It lists all the running processes along with their PIDs and other details. See the screenshot for more details.

   #ps (List all running processes)

USER	PID	PPID	VSIZE	RSS	WCHAN	PC		NAME
app_7	547	454	76360	13920	ffffffff	afe09604	S	com.google.android.hello

   This technique reveals the suspicious filename that we must look out for, while analyzing.

2. #pm -lf : List all the installed packages.

```
# pm -lf
/system/app/Browser.apk=com.google.android.browser
/data/app/ApiDemos.apk=com.google.android.samples
/system/app/TelephonyProvider.apk=com.google.android.providers.telephony
/system/app/Home.apk=com.google.android.home
/system/app/ContactsProvider.apk=com.google.android.providers.contacts
/data/app/LunarLander.apk=com.google.android.lunarlander
/system/app/Phone.apk=com.google.android.phone
/system/app/Fallback.apk=com.google.android.fallback
/data/app/HelloAndroid.apk=com.google.android.hello
```

   The last line in the above output shows that a malicious application is installed under /data/app and is called HelloAndroid.apk.

3. Output from Dumpstate also reveals signatures as is demonstrated below.

```
02-16 18:17:35.601 D/PackageManager(  465): Removing package com.google.android.hello
02-16 18:17:35.631 D/PackageManager(  465):   Activities: com.google.android.hello.HelloAndroid
02-16 18:17:36.561 D/AndroidRuntime(  531):
02-16 18:17:36.561 D/AndroidRuntime(  531): >>>>>>>>>>>>>> AndroidRuntime START <<<<<<<<<<<<<<
02-16 18:17:36.601 D/AndroidRuntime(  532):
02-16 18:17:36.601 D/AndroidRuntime(  532): >>>>>>>>>>>>>> AndroidRuntime START <<<<<<<<<<<<<<
02-16 18:17:36.871 D/ActivityManager(  465): Force removing app null (com.google.android.hello)
02-16 18:17:36.871 D/PackageParser(  465): Scanning package: /data/app/HelloAndroid.apk
02-16 18:17:36.881 I/ActivityThread(  522): Loading code package com.google.android.providers.contacts (in com.google.process.content)
02-16 18:17:36.961 D/PackageManager(  465): Adding package com.google.android.hello
02-16 18:17:36.961 D/PackageManager(  465):   Activities: com.google.android.hello.HelloAndroid
02-16 18:17:37.101 I/ActivityThread(  522): Publishing provider googleaccounts: com.google.android.providers.googleapps.GoogleAccountsProvider
```

   The name of the malicious program was known from the **ps** command and thus we can easily confirm the time(s) at which this trojan was started and what was the sequence of actions that it

took to do the damage. Logic bombs that are resident in the system can also be detected in a similar fashion.

1. Memory Profiler: This tool can be useful in differentiating between two applications, one of which is a rogue. Many a times, two file names look similar like svchost.dll and svchosI.dll. In such cases, it is easy to miss the critical file.

   Memory profiler immediately shows the amount of system memory consumed by each application and thus the same two applications may be reported with different memory usage. When compared to the baseline image of the system, one can then easily find out the rogue. A screenshot from the output is shown below.

```
PROCESS NAME        MEMORY USED
-----------------------------------------
system_server                    19588 KB
com.google.process.content       19016 KB
com.google.android.phone         14936 KB
zygote                           14776 KB
com.google.android.hello         13992 KB
```

6. Collecting Persistent Data

The *adb* tool was mentioned earlier, so that the native commands could be run. It has further functionality, especially when files are need to be transferred to and fro from the Operating System. This section deals with acquisition of persistent data and provides some inputs on how it can be analyzed, for starters.

6.1 Data Acquisition

To copy a file or directory (recursively) *from* the emulator or device [4], use

```
$adb pull <remote> <local>
```

To copy a file or directory (recursively) *to* the emulator or device [4], use

```
$adb push <local> <remote>
```

<local> denotes the directory on the host, in this case a computer that runs the SDK
<remote> denotes the directory on the guest, in this case the Android SDK

For instance, if one has to acquire the contact information, the following command should be issued -

```
$./adb pull /data/data/com.google.android.providers.contacts/databases/contacts.db hostdir
```

NOTE: Copying remote directories is not supported yet, however, individual files can be copied.

After obtaining the file(s) forensic study may be conducted on an appropriate copy of the file using a database utility like Sqlite3.

6.2 Using Sqlite3

A short guide on Sqlite3 is presented, which helps in examining vital evidence, without hampering the device.

1. #sqlite3 <path-to-db-file>, fires up the program and throws a *sqlite3*> prompt.
2. .tables, shows the tables that are contained within the loaded database.
3. .help, can be entered anytime to see the lost of available commands.
4. .schema <table-name>, shows the structure of a table
5. .quit, quits the program.

6.3 Important Databases

The important databases which should be collected, along with their contained tables are described below.

1. Contacts.db: It has three tables – calls, people and phones.

Calls table structure:
(_id, number, number_key, number_type, date, duration, type, person, new)

People table structure:
(_id,_sync_account,_sync_id,_sync_time,_sync_version,_sync_local_id,_sync_dirty ,_sync_mark ,name, notes, photo, company,title, preferred_phone , preferred_email)

Phones table structure:
<empty>

2. Im.db: It has all the Instant Messenger contact details and messaging history. The interesting tables are shown below.

Accounts table structure:
(_id, number, number_key, number_type, date, duration, type, person, new)

Messages table structure:
(_id, contact, provider, account, body, date, incoming);

3. sms.db: Contains history about SMS messages. Mms.db can also be queried if required.

Attachments table structure:
(sms_id, content_url, offset);

Sms table structure:
(_id , thread_id, address, person, date, read, type, reply_path_present, subject, body, service_center)

4. browser.db:

Bookmarks table structure:
(_id, title, url,visits, date, created, bookmark);

Cache table structure:
(_id, url, filepath, lastmodify, etag, expires, mimetype, encoding);

FormUrls table structure: #tells what URLs were visited
(_id, url)

Cache table structure:
(_id, host, username, password)

Searches table structure:
(_id, search, date);

7. Analyzing Persistent Data

This section will introduce a new tool called dexdump that can be used to disassemble a .apk application file. Dexdump utility can be used to tear apart a dex file, which may be obtained form the cache directory.

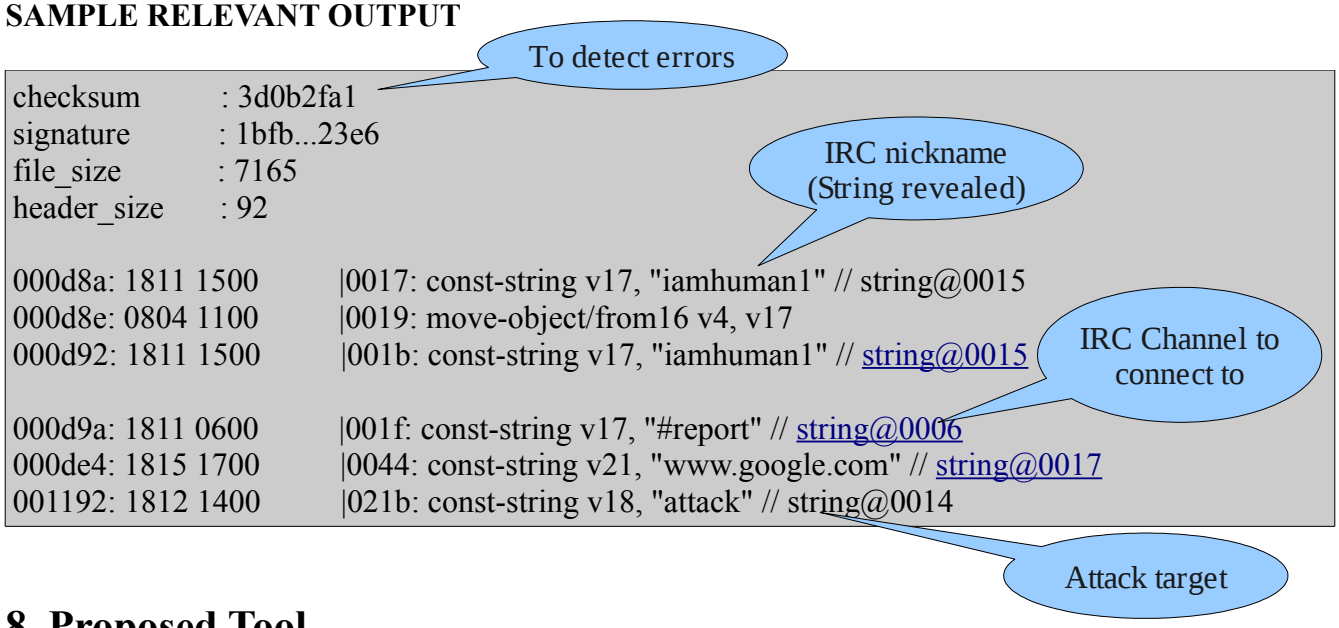
NOTE: So far we had neglected calculating checksums for the files before transferring them. Dexdump allows the calculation of a checksum which should be verified on the host machine to ensure that the data was transferred correctly.

The important categories listed when #dexdump <file-name.dex> is run are highlighted below. (We will analyze the proposed trojan 'Hello.Android' as the dex file. For this pipe the output of dexdump to a file and then pull this file to the host using adb, for analysis.)

Command: #dexdump -d -h -f -C data@app@HelloAndroid.apk@classes.dex > output.txt

Option,
-d : disassemble code sections
-f : display summary information from file header
-h : display file header details
-C : decode (demangle) low-level symbol names

SAMPLE RELEVANT OUTPUT




```
checksum        : 3d0b2fa1
signature       : 1bfb...23e6
file_size       : 7165
header_size     : 92

000d8a: 1811 1500       |0017: const-string v17, "iamhuman1" // string@0015
000d8e: 0804 1100       |0019: move-object/from16 v4, v17
000d92: 1811 1500       |001b: const-string v17, "iamhuman1" // string@0015

000d9a: 1811 0600       |001f: const-string v17, "#report" // string@0006
000de4: 1815 1700       |0044: const-string v21, "www.google.com" // string@0017
001192: 1812 1400       |021b: const-string v18, "attack" // string@0014
```

8. Proposed Tool

Apart from the collection techniques analyzed so far, it is desired that the whole collection process be automated so that more time is available for analysis of actual data. For this reason a new tool is proposed which can help collect forensic data from phones that would run the Android platform.

8.1 Features

The forensic tool can be used to collect information such as -

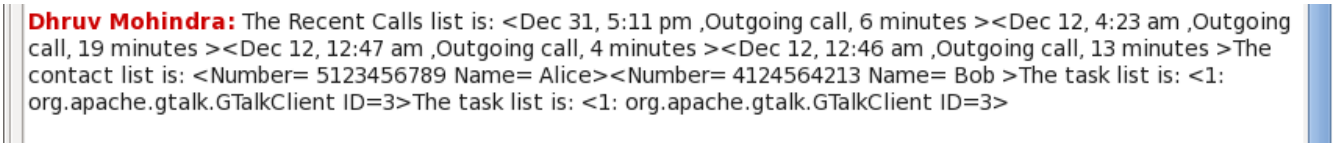
1. The Task List (Helps detect any rogue programs that are currently executing)
2. Call logs from the Recent Calls List
3. Information from the Contacts List
4. Instant Messaging Logs

It is possible to recover SMS messages as well but this functionality is not exported in Google's SDK yet. The XMPP protocol has been used to transfer messages from the device to the host computer.

Bluetooth and Wi-fi capability is yet to be added to the SDK, hence the whole process is demonstrated using Instant Messaging.

The screenshot below shows the tool in collection mode.

The screenshot below shows the output that the host receives.



LOG Formats

<> denotes one entry.

Recent Calls list is composed of the tuple - <Date and time of call ,Type of call (Outgoing, Incoming or Missed,), Duration of call>

Contact List is composed of the tuple - <Phone Number, Name>

Task List comprises the tuple <Program/task name that is currently executing>

8.2 Code Snippets

The basic source code of the application is in Java that is specially crafted for the Android Platform. This section will cover the data acquisition code rather than the entire module, for the sake of brevity.

Data is stored and retrieved to/from a database using content providers. Content provider is an object that can perform these functions for all applications. It provides a sole way to share data between packages, in absence of all other methods. "Each content provider exposes a unique string (a URI) identifying the type of data that it will handle, and the client must use that string to store or retrieve data of that type"[5].

For example,
content://contacts/people/23 is the URI string that would return a single result row, the contact with ID = 23

An application specified a query that returns some data item and Cursor over the result set is returned. An example from the tool is listed below.

```
Cursor c2 = managedQuery( android.provider.Contacts.Phones.CONTENT_URI,
                projection2, //Which columns to return.
                null,       // WHERE clause--we won't specify.
                android.provider.Contacts.PeopleColumns.NAME + " ASC"); // Order-
by clause.

        if(!c2.first())
            return;
String name;
String phoneNumber;

int nameColumn = c2.getColumnIndex(android.provider.Contacts.PeopleColumns.NAME);
int phoneColumn=
c2.getColumnIndex(android.provider.Contacts.PhonesColumns.NUMBER);

    do {
         // Get the field values
         name = c2.getString(nameColumn);
         phoneNumber = c2.getString(phoneColumn);
         mSendText.append("<" + "Number= "+phoneNumber + " Name= " + name + ">");

       } while (c2.next());

private static String[] projection2 = new String[] {
       android.provider.Contacts.PeopleColumns.NAME,
       android.provider.Contacts.PhonesColumns.NUMBER

    };
```

The first argument to managedQuery() function is the URI and the second (projection2) is a list of fields that need to be returned. In this case we are interested in the NAME and NUMBER fields in the database.

NameColumn and phoneColumn store the returned values of indices within the database. To obtain the actual data, the cursor c2 is queried according to the indices and the results returned in name and phoneNumber. Finally the loop iterates till the cursor becomes empty. The gathered data is then sent across to the host computer using the XMPP protocol.

8.3 Following Forensic Practices

The tool can be installed on an external SD card and thus run from it directly, while collecting forensic data from the device. This preserves the integrity of the device's file system and also forms a first responder's trusted toolkit. It is highly recommended that the examiner use an md5sum application written specially for Dalvik to establish the chain of custody.

9. Mitigation Strategies

The simplest way to remove the trojan is to do it conventionally using the *adb* tool. Once the compromise has been recognized, the best thing is to remove the application from the /app directory. However, in some scenarios it is vital to gauge the extent of damage or compromise. Thus it may be a good idea to pull the plug after the Live Response has been conducted.

Some precautionary measures that can be taken to avoid installation of such software -

1. Avoid installing unauthorized software and pay attention to any suspicious activities.
2. It is possible that bugs in the platform can reveal holes in the sandbox and the overall security model. Thus it is vital to update the Operating System from time to time. Patches or firmware upgrades will increasingly become important in the mobile phone community.
3. Recognize compromise early if possible so that the attack can be contained.
4. If you ever seize a phone, do not ever make any calls or play with the files. It could trigger some logic bomb or could completely destroy the evidence.

10. Future Work

Many of the features could not be demonstrated at this time due to unavailability of APIs that have been documented, however, not implemented with the m3 version of the SDK. Interfaces change every now and then and often break working components, thus maintainability of the code is a potential issue.

Among future improvements, transferring data to the host using bluetooth or wireless is very desirable. This will improve efficiency and will allow for easy persistent storage. For GSM capable handsets, the SIM card will need to be queried for information. The code has been written, however again, the API remains unimplemented.

Binary analysis using dexdump has been introduced in this module and more will be covered later. The analysis of apk files can reveal necessary information, however if a suspect uses his/her phone to hide some data, then other tools will be needed. This is an unexplored arena but with the advent of large scale storage systems in mobiles, it is a significant plausibility.

11. Conclusion

Mobile phones have become pervasive in todays world. The era where every phone runs the Internet may not be too far. We will see increasing ports of computer hacks and attacks, being simulated on mobile phones. While individual vendors already have solutions for many devices, the Android Platform is still relatively new. Many companies have adopted security in their Software Development LifeCycle and such should be the approach while designing applications and interfaces for Android. Even when an application lives in a sandbox, it is possible that sensitive data can leak out, as seen.

This report aims at creating awareness by demonstrating the proof-of-concept mobile phone 0-day DDoS attack, which grants an attacker control over the victim's phone. At the same time, a tool has been proposed to gather the most important data, when a mobile phone is seized. A data collection procedure was then established. A live response strategy for the Android Platform was formed and analyzing persistent data was covered in brief. In the end, mitigation strategies were enumerated and it is hoped that the end user pays due attention to these notes.

The days of waging malicious attacks from your pocket are fast approaching...

12. References

[1] Android SDK
http://code.google.com/android/documentation.html

[2] Android Security Model
http://code.google.com/android/devel/security.html

[3] Source-side Defenses Against Distributed Denial of Service Attacks
http://www1.webng.com/dhruv/material/secuzilla.pdf

[4]Android Debug Bridge (adb)
http://code.google.com/android/reference/adb.html

[5] Android, Content Providers
http://code.google.com/android/devel/data/contentproviders.html

[6] IRC Hacks book, by Paul Mutton (Some help with coding IRC clients – used in bot)
http://www.amazon.com/IRC-Hacks-Paul-Mutton/dp/059600687X