Contents

Preface xi
Acknowledgments xiii
Authors xv
Conventions xix

Chapter 1 Introduction to the Android Operating System and Threats 1
Android Development Tools 2
Risky Apps 3
Looking Closer at Android Apps 5

Chapter 2 Malware Threats, Hoaxes, and Taxonomy 7
2010 7
FakePlayer 7
DroidSMS 8
FakeInst 8
TapSnake 8
SMSReplicator 9
Geinimi 9
2011 10
ADRD 10
Pjapps 11
BgServ 11
DroidDream 11
Walkinwat 12
zHash 13
DroidDreamLight 13
Zsone 14
BaseBridge 14
DroidKungFu1 15
GGTracker 16
jSMSHider 16
Plankton 17
GoldDream 18
DroidKungFu2 18
GamblerSMS 19
HippoSMS 19
LoveTrap 19
Nickyspy 20
SndApps 20
Zitmo 21
DogWars 21
DroidKungFu3 22
GingerMaster 22
AnserverBot 23
DroidCoupon 23
Spitmo 24
JiFake 24
Batterydoctor 24
2012 25
AirPush 25
Boxer 25
Gappusin 26
Leadbolt 26
Adwo 26
Counterclank 27
SMSZombie 27
NotCompatible 27
Bmaster 27
LuckyCat 28
DrSheep 28
2013 28
GGSmart 28
Defender 29
Qadars 29
MisoSMS 29
FakeRun 30
TechnoReaper 30
BadNews 31
Obad 31
2014 32
DriveGenie 32
Torec 32
OldBoot 33
DroidPack 33

Click here to buy Android Malware and Analysis
## Chapter 3  Open Source Tools

Locating and Downloading Android Packages 36  
Vulnerability Research for Android OS 37  
Antivirus Scans 37  
Static Analysis 38  
  - Linux File Command 38  
  - Unzip the APK 38  
  - Strings 39  
  - Keytool Key and Certificate Management Utility 39  
  - DexID 39  
  - DARE 40  
  - Dex2Jar 40  
  - JD-GUI 41  
  - JAD 41  
  - APKTool 41  
  - AndroWarn 41  
  - Dexter 42  
  - VisualThreat 43  
Sandbox Analysis 43  
  - AndroTotal 45  
  - APKScan 45  
  - Mobile Malware Sandbox 45  
  - Mobile Sandbox 45  
Emulation Analysis 45  
  - Eclipse 45  
  - DroidBox 46  
  - AppsPlayground 46  
Native Analysis 46  
  - Logcat 46  
  - Traceview and Dmtracedump 46  
  - Tcpdump 47  
Reverse Engineering 47  
  - Androguard 47  
  - AndroidAuditTools 48  
  - Smali/Baksmali 48  
  - AndBug 48  
Memory Analysis 48  
  - LiME 49  
  - Memfetch 49  
  - Volatility for Android 49  
  - Volatilitux 49  

## Chapter 4  Static Analysis

Collections: Where to Find Apps for Analysis 52  
  - Google Play Marketplace 52  
  - Marketplace Mirrors and Cache 53  
  - Contagio Mobile 53
<table>
<thead>
<tr>
<th>Chapter 5</th>
<th><strong>Android Malware Evolution</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Chapter 6</th>
<th><strong>Android Malware Trends and Reversing Tactics</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Chapter 7</th>
<th><strong>Behavioral Analysis</strong></th>
</tr>
</thead>
</table>

Advanced Internet Queries 53
Private Groups and Rampart Research Inc. 53
Android Malware Genome Project 54
File Data 54
Cryptographic Hash Types and Queries 55
Other Metadata 56
Antivirus Scans and Aliases 57
Unzipping an APK 57
Common Elements of an Unpacked APK File 57
Certificate Information 58
Permissions 59
Strings 60
Other Content of Interest within an APK 61
Creating a JAR File 62
Visual Threat Modeling 62
Automation 62
(Fictional) Case Study 63

Click here to buy Android Malware and Analysis
## Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installing Samples to Devices and Emulators</td>
<td>111</td>
</tr>
<tr>
<td>Application Storage and Data Locations</td>
<td>112</td>
</tr>
<tr>
<td>Getting Samples Off Devices</td>
<td>112</td>
</tr>
<tr>
<td>The Eclipse DDMS Perspective</td>
<td>113</td>
</tr>
<tr>
<td>Devices View</td>
<td>113</td>
</tr>
<tr>
<td>- Network Statistics</td>
<td>116</td>
</tr>
<tr>
<td>- File Explorer</td>
<td>116</td>
</tr>
<tr>
<td>- Emulator Control</td>
<td>117</td>
</tr>
<tr>
<td>- System Information</td>
<td>117</td>
</tr>
<tr>
<td>LogCat View</td>
<td>117</td>
</tr>
<tr>
<td>- Filtering LogCat Output</td>
<td>117</td>
</tr>
<tr>
<td>Application Tracing</td>
<td>118</td>
</tr>
<tr>
<td>Analysis of Results</td>
<td>118</td>
</tr>
<tr>
<td>Data Wiping Method</td>
<td>122</td>
</tr>
<tr>
<td>Application Tracing on a Physical Device</td>
<td>122</td>
</tr>
<tr>
<td>Imaging the Device</td>
<td>124</td>
</tr>
<tr>
<td>Other Items of Interest</td>
<td>126</td>
</tr>
<tr>
<td>- Using Google Services Accounts</td>
<td>126</td>
</tr>
<tr>
<td>- Sending SMS Messages</td>
<td>126</td>
</tr>
<tr>
<td>- Getting Apps from Google Play</td>
<td>127</td>
</tr>
<tr>
<td>- Working with Databases</td>
<td>127</td>
</tr>
<tr>
<td>Conclusion</td>
<td>128</td>
</tr>
</tbody>
</table>

### Chapter 8 Building Your Own Sandbox  

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Analysis</td>
<td>130</td>
</tr>
<tr>
<td>Dynamic Analysis</td>
<td>131</td>
</tr>
<tr>
<td>Working Terminology for an Android Sandbox</td>
<td>131</td>
</tr>
<tr>
<td>- Android Internals Overview</td>
<td>131</td>
</tr>
<tr>
<td>- Android Architecture</td>
<td>132</td>
</tr>
<tr>
<td>- Applications</td>
<td>133</td>
</tr>
<tr>
<td>- Applications Framework</td>
<td>133</td>
</tr>
<tr>
<td>- Libraries</td>
<td>134</td>
</tr>
<tr>
<td>Android Runtime</td>
<td>135</td>
</tr>
<tr>
<td>The Android Kernel</td>
<td>139</td>
</tr>
<tr>
<td>Build Your Own Sandbox</td>
<td>144</td>
</tr>
<tr>
<td>Tools for Static Analysis</td>
<td>144</td>
</tr>
<tr>
<td>Androguard</td>
<td>144</td>
</tr>
<tr>
<td>- Radare2</td>
<td>146</td>
</tr>
<tr>
<td>- Dex2Jar and JD-GUI</td>
<td>147</td>
</tr>
<tr>
<td>- APKInspector</td>
<td>148</td>
</tr>
<tr>
<td>- Keytool</td>
<td>148</td>
</tr>
<tr>
<td>Tools for Dynamic Analysis</td>
<td>149</td>
</tr>
<tr>
<td>TaintDroid</td>
<td>149</td>
</tr>
<tr>
<td>DroidBox</td>
<td>150</td>
</tr>
<tr>
<td>DECAF</td>
<td>151</td>
</tr>
<tr>
<td>TraceDroid Analysis Platform</td>
<td>151</td>
</tr>
<tr>
<td>Volatility Framework</td>
<td>152</td>
</tr>
</tbody>
</table>

Click here to buy Android Malware and Analysis
## CONTENTS

Sandbox Lab (Codename AMA) 152  
Architecture 153  
Host Requirements 154  
Operating System 154  
Configuration 158  
Running Sandbox 162  
Static Analysis of Uploaded Malware Samples 164  
Dynamic Analysis of Uploaded Malware Samples 168  
Conclusions about AMA 173

### Chapter 9  Case Study Examples 175

Usbcleaver 175  
Checkpoint 180  
Static Analysis 180  
Checkpoint 185  
Dynamic Analysis 185  
Launch of the APK 187  
Summary 195  
Torec 196

### Bibliography

205

### Index

209
Preface

Updated information, tutorials, a private forum, code, scripts and tools, and author assistance are available on http://AndroidRisk.com for first-time owners of each copy of this book.

Everyone just starting in a technical field, from the student in college to a seasoned security professional who wishes to add another skill to his or her seasoned career, can benefit from this actionable and tactical book. Within minutes, the reader can start analyzing Android malware. This is not a book on Android OS, fuzzy testing, or social engineering; it is, however, on tearing apart Android malware threats. You can quickly become the local expert with just a few tools and tips outlined in this book. This book contains a voice of authority from leading global experts in the field who have already sized up the best tools, tactics, and procedures for recognizing and analyzing Android malware threats quickly and effectively.

Global growth and development of Android-based devices has resulted in a wealth of assets on mobile solutions. In 2014, a person’s phone may contain more information than a personal computer did at the turn of the century, with sensitive contacts, banking information, online searches and habits, personal voice and text data, recorded geolocations at all times, a camera, voice monitoring and recording, personal information, and more. Malware naturally follows areas of opportunity for a variety of motives including eCrime, espionage, and
hacktivism. Rapid adoption and changes in the Android operating system, apps, and real-world implementation have resulted in widespread use with little to no malware protection in many cases. Most security professionals have little understanding of how to approach the complex subject of Android malware threats and analysis.

Advanced topics, such as reverse engineering, do require the reader to have some prior experience with the topic to properly understand the tools and tactics explained.
The evolution of Android malware, while mapping closely to the desktop trends, is often viewed at an accelerated pace. Malware and botnets have had time to grow and trial different methods of infections and potential uses, and the authors of the mobile counterparts are definitely applying these learned lessons. There are clear indicators that these are often the same groups working toward extending their list of infected machines to the Android world.

Android also provides an extra interesting launching point for these actors. Although broadband connection PCs were often considered golden, with the always-on connection and almost never being shut off, the mobile phone provides even more perks: access to telephony systems, the ability to dial or text numbers, location-aware services, and access to high-speed segmented systems. Although with some of these features there are clear monetization methods, such as premium text messaging, others like the Internet may seem questionable. One could assume a malicious actor would rather have unchanging Internet connection from a desktop machine, however this would not give them the possibility for roaming. A cell phone could drift from 3G to 4G, offering an interesting proxy scenario. Add in the fact that this device might then connect to a sensitive network at some point, it could exfiltrate or gain intimate knowledge that a PC might never have access to.

The first Android malware to come into existence in early August 2010 was dubbed FakePlayer. There was really no magic to this malware; it purported to be a video player for viewing porn on Android. Since the code was compiled with debug information left in, we could estimate how many lines the original Java code would have been. This trick is actually quite easy. The Dalvik code allows us to see which opcodes originated from which Java code, so that if an error occurs the stack trace can give you useful information about which line the
error occurred at. FakePlayer only consists of three main classes—MoviePlayer, HelloWorld, and DataHelper—so focusing on these classes after using baksmali on the APK file we can look for the .line operation. If we then only look at the highest line count, we should be able to get an accurate estimation of how many lines of Java it originated from. Grepping (Linux tool grep) through we can see that DataHelper has 69 lines, HelloWorld has 55, and MoviePlayer has 210 lines; this leads us to a total of 334 lines of code. This would include empty lines, comments, and other nonfunctional pieces of code. If we look at the following excerpts from the MoviePlayer class in smali code, we can quickly and easily translate it to Java pseudocode:

```smali
.line 35
 invoke-static {}, Landroid/telephony/SmsManager;->getDefault()Landroid/telephony/SmsManager;
 move-result-object v0
 .line 54
 .local v0, "m":Landroid/telephony/SmsManager;
 const-string v1, "3353"
 .line 55
 .local v1, "destination":Ljava/lang/String;
 const-string v3, "798657"
 .line 57
 .local v3, "text":Ljava/lang/String;
 const/4 v2, 0x0
 const/4 v4, 0x0
 const/4 v5, 0x0
 :try_start_2a
 invoke-virtual/range {v0.. v5}, Landroid/telephony/SmsManager;->sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V
 :try_end_2d
 .catch Ljava/lang/Exception; {:try_start_2a.. :try_end_2d} :catch_44
```

This code essentially will just take the SmsManager object and use it to send a text message to the 3353 number with a body of 798657. The rest of the registers are loaded with 0x0, which is interpreted as null in this case, and not actually required for the sendTextMessage method. Immediately before this, a TextView is set to read “Подождите,
“Wait, requested access to the video library...” which roughly translates to “Wait, requested access to the video library...” After the first text message is sent, the same message as before will be sent to the short code 3354. That sums up the first Android malware, a little less than 350 lines of code, showing only a small blurb of Russian text and sending off two text messages.

Although this is a relatively simple example of malware, it showed initiative in malware. It also showed the immediate motivation to try and monetize the mobile space. It is not a large surprise that the first targets were Russian consumers. At this point in time, the Google Play store (former Android Market) was not accessible by all countries, nor did all countries have access to Google Experience devices, which came bundled with the store. This, combined with the ease of sending money on Eastern European telecom carriers, made it a likely target for the Russian malware actors.

As time progressed, we have seen the Russian actors step up their game—both in the sophistication of distribution and coding. Although reports in the news and from vendors can often be misleading, it is clear that there was a significant push in automation from the Russian actors. Most followed the FakeInstaller game plan, which is essentially to somehow get a user to download their application thinking that it is the application that they actually wanted. Favorite Russian targets are often Opera, Skype, Google Play, or some type of pornography. Upon installing the malicious application, the user can sometimes be prompted with Terms of Service (TOS), which has details buried inside of it about payment. They often say that an update is required or you must acknowledge the TOS prior to using the application. After accepting (or sometimes no matter what is clicked) the user is charged and then the application that they were looking for is sometimes delivered.

One of these Russian families is called AlphaSMS. This family exhibits another common Russian trait: server side polymorphism. Although this is not the fascinating polymorphism we see in highly sophisticated bots on PCs, it does present an interesting issue. The server side polymorphism results in a new SHA1 for everything that is delivered to the user, mainly due to the back end systems generating and bundling the packages to meet what the victim is looking for. The back panels for AlphaSMS take in arguments, such as application
name, icon to use, and other resources, while the code remains the same. This is easily seen next:

```
bebop:alphasms tstrazzere$ shasum *apk
8263d3a255fe75f4d02d08e928a3113fa2f9e17 mwlqythh.
rwbkulojmti-1.apk
521d3734e927f47af62e15e9880017609c018373 mwlqythh.
rwbkulojmti-2.apk
bebop:alphasms tstrazzere$ shasum *.dex*
14e46f0330535cb5e8f377a6c2bb2c858de6f414 classes.dex-1
14e46f0330535cb5e8f377a6c2bb2c858de6f414 classes.dex-2
14e46f0330535cb5e8f377a6c2bb2c858de6f414 classes.dex-3
```

When inspecting the actual ZIP files we see that one of the only differences is when the files have been last touched (see Image 5.1).

This is one of the tactics that led to the mischaracterizing of malware in the wild. Although those three samples have different SHA1s, the internal code is identical as shown by the second SHA1. If we rely solely on unique containers and do not even bother inspecting the code, it is easy to incorrectly assert variants and how the code has evolved.

Another interesting trend that has grown in the Russian malware space is custom obfuscation. Although commercial obfuscators exist and are sometimes employed, the FakeInstaller organizations often employ their own obfuscation tools; AlphaSMS is no exception. This tactic can fool the classification tools, analysts, and detection techniques. An excerpt from an AlphaSMS highlights this (Image 5.2).
This code shows the use of reflection combined with encoding the strings. This pattern continues across the variants of AlphaSMS, slightly morphing the encoding used for the strings of each variant while keeping the underlying code the same. Upon a closer investigation of the AlphaSMS family, there was an even more interesting trend when looking at a massive amount of detection data.

Here, we can see the average number of detections over time of the Russian family AlphaSMS (see Image 5.3). Each different color is a different variant of the malware, distinguished by difference in the code, while taking into consideration obfuscation. What this immediately shows us is the level of sophistication in distributing and iterating on the malware. It models much like an agile coding shop. There appears to be a new “release” of the malware approximately every week and a half, meaning the operation would push a new code-base on a schedule while halting the distribution of the older versions.
Image 5.3  AlphaSMS distribution/infection trends over time.