Rav4 Device Provisioned issue analysis report

一、问题现象

下载正式版本软件，第一次开机做完 SetupWizard 之后无法锁屏，HOME 键和 MENU 键无效，重启后恢复正常。

Platform：MT6589

Android 版本：4.2JB

BuildType：user

系统软件版本：SWL31+UM

系统 RAM:1GB

二、第一次开机执行的流程

整个流程主要分为 3 个部分：

1、 Provision.apk 部分，此 apk 是 Android 默认的设置 DeviceProvsioned 标志的，正常来讲系统中只需要一个应用在第一次开机时进行 DeviceProvisioned 标志的置位。Provsion.apk 拥有和 Launcher 一样的 HOME 属性，并且优先级比 Launcher 要高，所以第一次开机启动时首先启动的是 provision.apk，它没有界面显示只做设置。 其配置属性如下图：

```
<application>
    <activity android:name="DefaultActivity"
            android:excludeFromRecents="true">
        <intent-filter android:priority="20">
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.HOME" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
</application>
```

它在设置完 DeviceProvsioned 标志之后会将自己禁用并结束，然后系统就会启动其他具有 HOME 属性的应用。具体代码如下：

```
/// M: SetupWizard exist
if (!setupWizardExists) {
    // Add a persistent setting to allow other apps to know the device has been provisioned.
    Settings.Secure.putInt(getContentResolver(), Settings.Secure.DEVICE_PROVISIONED, 1);
}

// remove this activity from the package manager.
ComponentName name = new ComponentName(this, DefaultActivity.class);
pm.setComponentEnabledSetting(name, PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
        PackageManager.DONT_KILL_APP);

// terminate the activity.
finish();
```

2、 SetupWizard 部分，Provision.apk 结束之后系统会再启动其他具有 HOME 属性的应用，因为 SetupWizard 的优先级比 Launcher 要高，所以此次启动的是 SetupWizard，它的属性配置如下图：

```
<activity android:name=".ui.SetupWizardStartActivityAlias"
          android:screenOrientation="portrait"
          android:clearTaskOnLaunch="false">
    <intent-filter android:priority="5" >
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.HOME" />
    </intent-filter>
</activity>
```

SetupWizard 执行完之后最后也会设置 DeviceProvisioned 标志，这一步与 provision.apk 的作用重合了，具体的设置代码如下：

```
Settings.Secure.putInt(getContentResolver(),
        Settings.Secure.DEVICE_PROVISIONED, 1);

PackageManager pm = getPackageManager();
ComponentName name = new ComponentName(this,
        SetupWizardStartActivityAlias.class);
pm.setComponentEnabledSetting(name,
        PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
        PackageManager.DONT_KILL_APP);
Intent intent = new Intent(Intent.ACTION_MAIN);
intent.addCategory(Intent.CATEGORY_HOME);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
        | Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED);
this.startActivity(intent);
```

同样的，SetupWizard 设置完成之后也会禁用自己拥有 HOME 属性的 Acitivity 组件，然后启动具有 HOME 属性的 Launcher。

3、 Launcher 部分，Launcher 的 HOME 属性 Activity 没有声明 android:priority 字段，系统默认值为 0，低于上面两个同样具有 HOME 属性的应用，所以最后被启动，Launcher 具体的属性配置如下：

```
<activity
    android:name="com.jrdcom.launcher.Launcher"
    android:launchMode="singleTask"
    android:clearTaskOnLaunch="true"
    android:stateNotNeeded="true"
    android:theme="@style/Theme"
    android:windowSoftInputMode="adjustPan"
    android:screenOrientation="nosensor">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.HOME" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.MONKEY"/>
    </intent-filter>
</activity>
```

Launcher 启动之后系统已经初始化完成，进入正常运行模式。

三、问题分析

1、初步分析

根据问题无法锁屏，HOME 键和 MENU 键无效的表面现象我们发现与系统没有成功设置 DeviceProvisioned 标志的现象非常相似和接近，以此为问题的切入点，再通过问题 log 一并进行分析，发现以下信息：

一、在 keyguard 获取 DeviceProvisioned 的值之前，provision.apk 已经将 DeviceProvisioned 的值修改为 ture（1）,同时将系统属性 sys.settings_global_version 的值 set 为 4，然后向监控 DeviceProvisioned 值改变的 keyguard 发送了通知，具体如下图：

```
00:00:00.346   648   668 I ActivityManager: Start proc com.android.provision for activity com.android.provision/.DefaultActivity
00:00:00.803   1261  1261 V Provider/Settings: Global.putString(name=device_provisioned, value=1 for 0
00:00:02.550   648   1054 V SettingsProvider: global <- value=1 name=device_provisioned for user 0
00:00:02.550   648   1054 V SettingsProvider: property: sys.settings_global_version=4
```

二、Keyguard 收到通知后是从 setting cache 中获取到的 DeviceProvisioned 的值，仍然为 false（0），而不是 provision.apk 更改后的 true（1），而且 keyguard 会将这一次通知读取到的值保存到一个成员变量中，以后都将使用这个成员变量作为 DeviceProvisioned 的判断，具体如下图：

```
00:00:02.801   648   666 V Provider/Settings:  from settings cache , name = device_provisioned , value = 0
00:00:02.801   648   666 D KeyguardUpdateMonitor: DEVICE_PROVISIONED state = false
```

三、当按键消息到达时，keyguardOn 的状态为 true，所以根据系统的代码逻辑会对 HOME 和 MENU 做特殊处理，如果 keyguardOn 并且 DeviceProvisioned 为 false 则不做响应，具体如下：

```
WindowManager: interceptKeyTi keyCode=3 down=true repeatCount=0 keyguardOn=true
```

```
if (!keyguardOn) {
        try {
                mStatusBarService.toggleRecentApps();
        } catch (RemoteException e1) {
                e1.printStackTrace();
        }
}
```

```
00:01:15.733   648   723 D KeyguardViewMediator: verifyUnlock
00:01:15.733   648   723 D KeyguardViewMediator: ignoring because device isn't provisioned
```

四、系统重启一次之后，所有功能恢复正常。

2、进一步分析

根据以上信息我们可以发现 DeviceProvisioned 的值已经成功设置，并且也及时通知到了监控此值的应用，包括 keyguard，但是 keyguard 获取到的却还是设置之前的旧值，现在问题的原因初步定位到获取的一方，获取 DeviceProvisioned 的接口最终会走到以下函数中：

```
public String getStringForUser(ContentResolver cr, String name, final int userHandle) {
    final boolean isSelf = (userHandle == UserHandle.myUserId());
    if (isSelf) {
        long newValuesVersion = SystemProperties.getLong(mVersionSystemProperty, 0);

        // Our own user's settings data uses a client-side cache
        synchronized (this) {
            if (mValuesVersion != newValuesVersion) {
                /// M: for more log to debug
                if (LOCAL_LOGV) {
                    Log.v(TAG, "invalidate [" + mUri.getLastPathSegment() + "]: current "
                            + newValuesVersion + " != cached " + mValuesVersion);
                }

                mValues.clear();
                mValuesVersion = newValuesVersion;
            }
```

这个函数会获取当前系统的 versionProperty，然后再与之前保存的 versionProperty 做比较，如果不相等则说明 settings 值有更新，之前保存的 cache 已经不能使用，所以会先清空 cache，然后保存当前的 versionProperty。
如果当前系统的 versionProperty 与之前保存的相等，则会从 cache 中查找，并将查找的结果返回，具体代码如下：

```
if (mValues.containsKey(name)) {
    /// M: for more log to debug
    String value = mValues.get(name);
    Xlog.v(TAG," from settings cache , name = " + name + " , value = " + value);
    return value;  // Could be null, that's OK -- negative caching
}
```

根据上面的初步分析我们可以知道 keyguard 没有走 versionProperty 的代码路径，而是走了 cache，这说明一个问题，就是以下代码并没有正确的获取到最新的 versionProperty：

```
long newValuesVersion = SystemProperties.getLong(mVersionSystemProperty, 0);
```

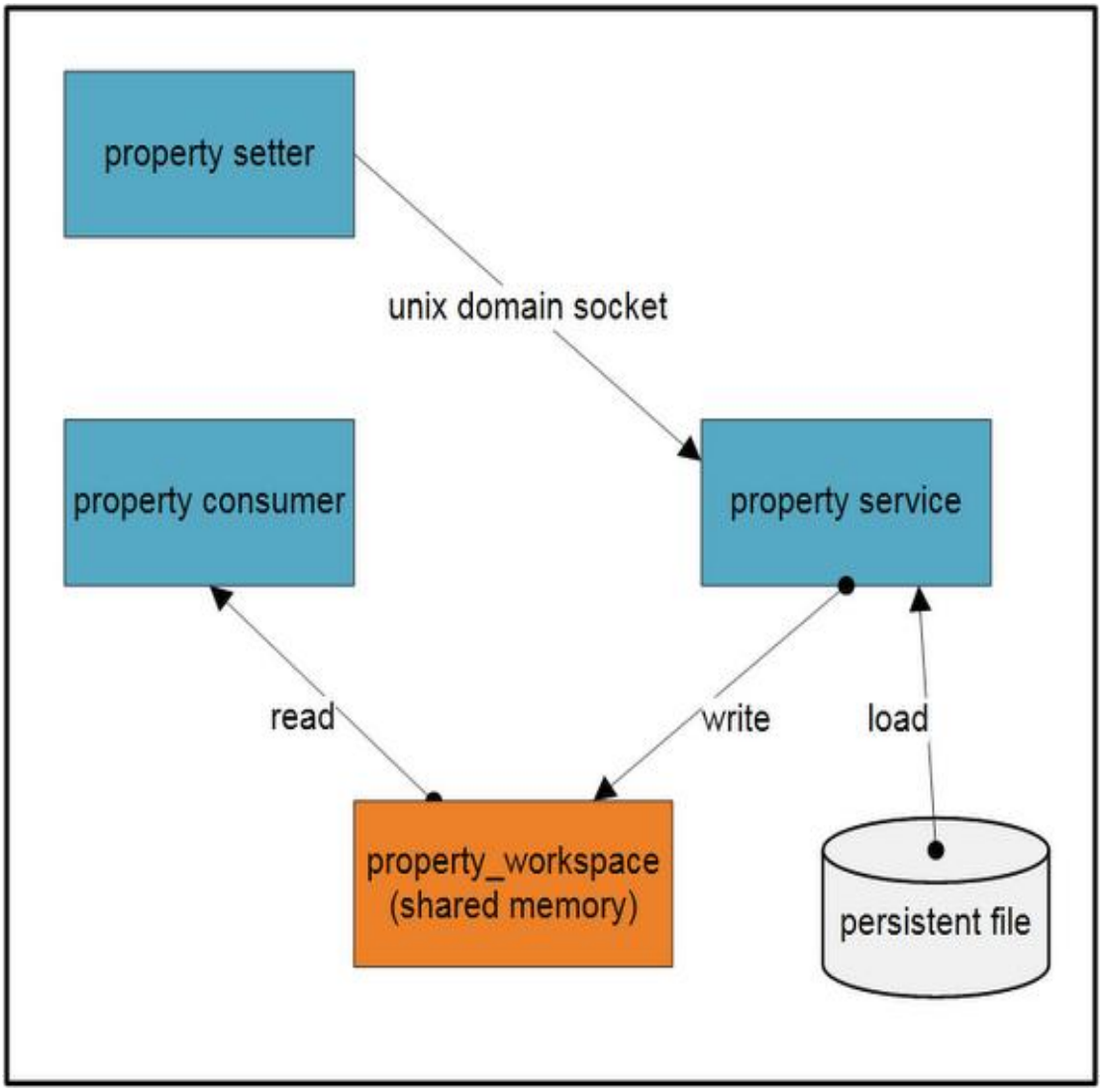
根据初步分析中的 log 以及时间点我们可以准确的知道 versionProperty 设置在前，keyguard 获取在后。
到这一步我们可以将问题的原因定位到 SystemProperties 的 set 和 get 上，就是 set 可能有存在延时，不能保证 set 之后马上 get 就能正确获取到设置后的值，下面我们通过分析 SystemProperties 的机制来验证：
1、首先 SystemProperties 的 set 和 get 实现不一样，set 时需 要通过本地 socket 与 property service（init 进程）通信，然后 property service 收到请求之后再 set 进 property workspace（共享内存方式）。但是 property service（init 进程）是单线程工作，所以有时 候它可能响应比较慢（比如它正在关闭一个子进程或者正在做其他事情），Android 在设计的时候发现了这个问题，所以他们采取的措施是在 set 之后等待 250ms， 以保证大部分时候的 read-after-write 能 够正确工作，但是 250ms 并不能保证所有时候都能正确的工作，所以我们可以认为这是一个设 计上的缺陷。在 Android 源码中的注释已经说明了这一点，具体如下：

```
r = TEMP_FAILURE_RETRY(send(s, msg, sizeof(prop_msg), 0));

if(r == sizeof(prop_msg)) {
    // We successfully wrote to the property server but now we
    // wait for the property server to finish its work.  It
    // acknowledges its completion by closing the socket so we
    // poll here (on nothing), waiting for the socket to close.
    // If you 'adb shell setprop foo bar' you'll see the POLLHUP
    // once the socket closes.  Out of paranoia we cap our poll
    // at 250 ms.
    pollfds[0].fd = s;
    pollfds[0].events = 0;
    r = TEMP_FAILURE_RETRY(poll(pollfds, 1, 250 /* ms */));
    if (r == 1 && (pollfds[0].revents & POLLHUP) != 0) {
        result = 0;
    } else {
        // Ignore the timeout and treat it like a success anyway.
        // The init process is single-threaded and its property
        // service is sometimes slow to respond (perhaps it's off
        // starting a child process or something) and thus this
        // times out and the caller thinks it failed, even though
        // it's still getting around to it.  So we fake it here,
        // mostly for ctl.* properties, but we do try and wait 250
        // ms so callers who do read-after-write can reliably see
        // what they've written.  Most of the time.
        // TODO: fix the system properties design.
        result = 0;
    }
```

2、另外 SystemProperties 的 get 是直接从映射到进程中的 property workspace（共享内存）获取的，如果此时 property service 很忙等待了 250ms 之后仍然没有 set 到 property workspace（共享内存）中，那么此时获取的就是错误的旧值，而不是刚刚发送给 property service 的新值。

SystemProperties 的实现框架如下图：



四、解决方案

通过以上分析，我们可以知道造成最终问题的原因是 SystemPropertites 机制本身的异步性和潜在的设计缺陷（不可靠性）导致的，而 keyguard 恰好在出现问题的时候收到通知并获取了错误的状态值，导致 keyguard 本身的状态混乱，最终引起一系列的按键和无法锁屏的现象。

针对此问题我们需要通过尽可能小的改动和影响来解决，通过分析当前的代码逻辑和结构，我们给出以下方案：

1、 在 SettingsProvider 中，保证 SystemProperties 在 set 成功之后再发送改变通知，从而让监控者能够正确的获取到 SystemProperties 的值。具体的实现为在 set 之后进行 get，当 get 到的值与 set 之后的值相等之后，此时就认为 set 已经成功，可以发送通知。

五、潜在问题与风险

以上方案采用的是忙等的方式，因为此问题发生的概率比较低，所以此方案只有在问题发生时此接口才会存在性能降低的可能，但是通过此方案可以保证数据的同步性和状态正确性。由于此问题的最终原因是 android 的 SystemProperties 机制造成的，所以理论上所有 android 项目都存在此问题。

#analysed by jinshi.song from SWD2 Framework team.
#jinshi.song@jrdcom.com
#201408201558