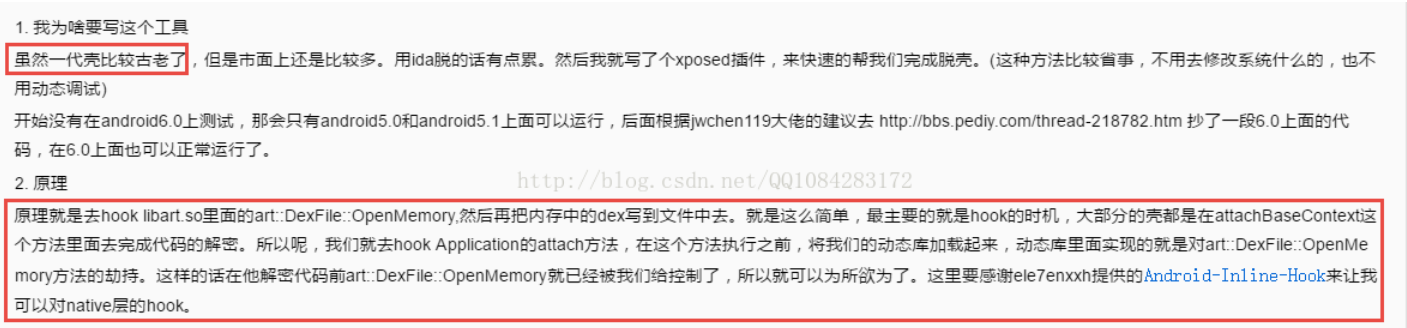
ART模式下基于Xposed Hook开发脱壳工具

原创

Fly20141201 于 2017-09-26 10:20:28 发布 4521 收藏 2
分类专栏： Android Hook学习 Android系统安全和逆向分析研究 文章标签： android脱壳 xposed Hook xposed脱壳 inlinehook 360脱壳

Dalvik模式下的Android加固技术已经很成熟了，Dalvik虚拟机模式下的Android加固技术也在不断的发展和加强，鉴于Art虚拟机比Dalvik虚拟机的设计更复杂，Art虚拟机模式下兼容性更严格，一些Dalvik虚拟机模式下的Android加固技术并不能马上移植到Art模式下以及鉴于Art虚拟机模式下的设计复杂和兼容性考虑，暂时相对来说，Art模式下的Android加固并没有Dalvik虚拟机模式下的粒度细和强。

本文给出的 Art模式下基于Xposed Hook开发脱壳工具的思路和流程不是我原创，主要是源自于看雪论坛的文章《一个基于xposed和inline hook的一代壳脱壳工具》，思路和流程有原作者smartdon提供，文中提到的Art模式下的dexdump脱壳工具源码github下载地址：https://github.com/smartdone/dexdump。原作者提供的脱壳操作步骤稍微复杂了一些，在此基础上我对原作者的代码进行了修改，使脱壳更加方便，原作者的代码是Android Studio的工程，顺手将其转化为了Eclipse下的工程。作者smartdon的Art模式下脱壳思路如下图所示：



要学习Android加固的脱壳还是需要先了解一下Dalvik模式下和Art模式下Android加固的流程和思路，熟悉一下 DexClassLoader 的代码执行流程。虽然Dalvik模式下和Art模式下DexClassLoader的java层实现代码是一样的，但是从 OpenDexFileNative函数 之后Dalvik模式下和Art模式下Native层的代码实现就不一样了，后面有空花时间整理一下Android加固相关方面的知识。Art模式下基于Xposed Hook开发的脱壳工具只对整体dex加固的Android应用脱壳才有效果，对于dex文件类方法抽离这类加固处理的Android应用就显得比较苍白了。

ART模式下基于Xposed Hook开发脱壳工具的思路整理。

1. Art模式下，Inline Hook时机 的选择

Android加固的一般思路：在外壳Apk应用调用 android.app.Application类 的成员函数 attach 时，内存解密出被保护的原始dex文件使用DexClassLoader进行内存加载，Art虚拟机模式下DexClassLoader进行dex文件的加载过程中绕不开函数 **const DexFile* DexFile::OpenMemory(const std::string& location, uint32_t location_checksum, MemMap* mem_map, std::string* error_msg)**，因此我们选择在 art::DexFile::OpenMemory函数 处进行dex文件的内存dump处理。基于Art模式下的Xposed Hook实现在外壳apk应用调用 android.app.Application类 的成员函数 attach 时，在被保护的dex文件加载之前Inline Hook OpenMemory函数，对内存解密后的原始dex文件进行拦截。

```
// 外壳dex加载器Apk应用的 ProxyApplication
public class ProxyApplication extends Application {

    private static final String appkey = "APPLICATION_CLASS_NAME";
    private String apkFileName;
    private String odexPath;
    private String libPath;

    protected void attachBaseContext(Context base) {

        super.attachBaseContext(base);

        // 省略dex文件的解密过程代码

        // 配置动态加载环境，用以后面获取当前apk的父ClassLoader
        Object currentActivityThread = RefInvoke.invokeStaticMethod("android.app.ActivityThread", "currentActivityThread",
                new Class[] {}, new Object[] {});
        // 获取当前外壳apk的包名
        String packageName = this.getPackageName();
        HashMap mPackages = (HashMap) RefInvoke.getFieldOjbect("android.app.ActivityThread", currentActivityThread, "mPackages");

        // 获取当前外壳apk的引用
        WeakReference wr = (WeakReference) mPackages.get(packageName);

        // public DexClassLoader(String dexPath, String optimizedDirectory, String libraryPath, ClassLoader parent)
        // 动态加载被加隐藏释放的原apk程序
        DexClassLoader dLoader = new DexClassLoader(apkFileName, odexPath, libPath,
                (ClassLoader)RefInvoke.getFieldOjbect("android.app.LoadedApk", wr.get(), "mClassLoader"));

        // 将动态加载的加密隐藏释放的原apk的ClassLoader和当前外壳apk的引用关联起来
        RefInvoke.setFieldOjbect("android.app.LoadedApk", "mClassLoader", wr.get(), dLoader);

    }
```


Art模式下，Xposed Hook外壳apk应用 android.app.Application类（实现的代理子类） 的成员函数 attach。

```
        // 对Android系统类android.app.Application的attach函数进行art模式下的Hook操作
        XposedHelpers.findAndHookMethod("android.app.Application",
                                        loadPackageParam.classLoader,
                                        "attach",
                                        Context.class,
                                        new XC_MethodHook() {

            private Context context;
            @Override
            protected void beforeHookedMethod(MethodHookParam param) throws Throwable {

                super.beforeHookedMethod(param);
                // 在类android.app.Application的attach函数调用之前进行dex文件的内存dump操作
                Dumpper.dump();
            }

            @Override
            protected void afterHookedMethod(MethodHookParam param) throws Throwable {
                super.afterHookedMethod(param);
                // 不处理
            }
        });
    }
```


2. Art模式下，Inline Hook函数点 的选择

Art虚拟机模式下，对 art::DexFile::OpenMemory函数 进行Inline Hook操作所采用的Hook框架为作者 Ele7enxxh 编写的Android平台的Inline Hook库。作者Ele7enxxh关于该Inline Hook库的介绍和描述可以参考作者的博文《Android Arm Inline Hook》，该Inline Hook库的github下载地址为：https://github.com/ele7enxxh/Android-Inline-Hook。由于Art虚拟机模式下，dex文件的加载DexClassLoader的代码实现流程中绕不开art::DexFile::OpenMemory函数的执行，更重要的是该函数的传入参数base表示的是dex文件所在的内存地址， size表示的是dex文件的字节长度，因此选择在art::DexFile::OpenMemory函数处进行dex文件的内存dump处理。

Android 5.0版以后ART模式下，OpenMemory函数的形式：http://androidxref.com/5.0.0_r2/xref/art/runtime/dex_file.cc#325

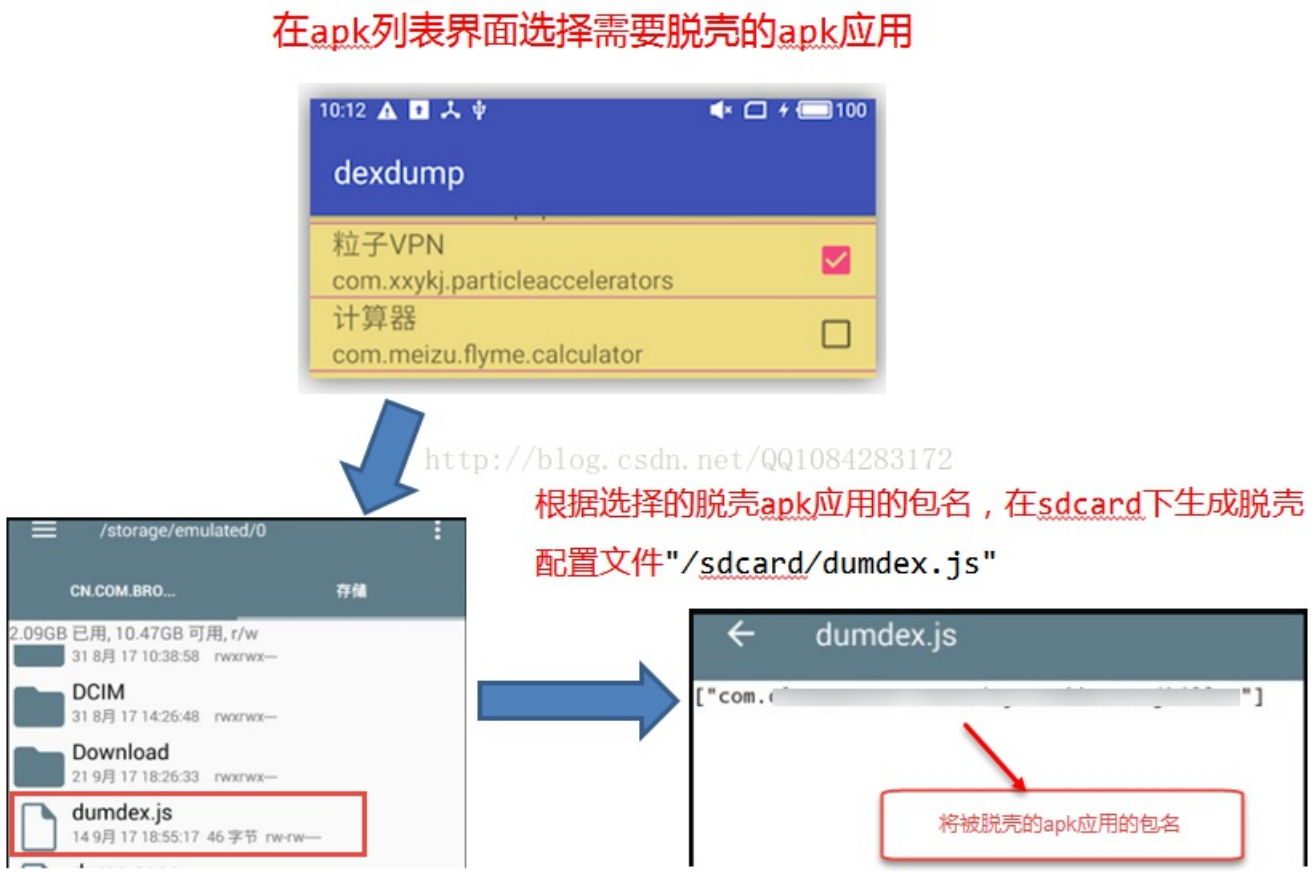
```
323
324
325 const DexFile* DexFile::OpenMemory(const byte* base,
326                                      size_t size,
327                                      const std::string& location,
328                                      uint32_t location_checksum,
329                                      MemMap* mem_map, std::string* error_msg) {
330   CHECK_ALIGNED(base, 4);  // various dex file structures must be word aligned
331   std::unique_ptr<DexFile> dex_file(new DexFile(base, size, location, location_checksum, mem_map));
332   if (!dex_file->Init(error_msg)) {
333     return nullptr;
334   } else {
335     return dex_file.release();
336   }
337 }
```

dex文件所在的内存基地址

dex文件的文件长度大小

ART模式下基于Xposed Hook和Ele7enxxh Inline Hook开发的脱壳工具dexdump的代码详细分析。

1. 作者smartdon写了个获取当前安装应用的列表界面，用以选择需要脱壳的apk应用，然后根据选择脱壳apk应用的包名，在sdcard文件夹下生成脱壳需要的配置文件dumdex.js，dumdex.js文件中保存着需要脱壳的apk应用的包名。



选择脱壳apk应用列表界面的实现代码 MainActivity.java：

```
package com.xposedhook.dexdump;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.pm.PackageInfo;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.ListView;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.List;

import com.example.com.xposedhook.dexdump.R;

public class MainActivity extends Activity {

//    static {
//
//     // 加载动态库文件libhook.so
//        System.loadLibrary("hook");
//    }
```

```
    private List<Appinfo> appinfos;
    private ListView listView;
    private AppAdapter adapter;
    private List<String> selected;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        // 设置布局文件
        setContentView(R.layout.activity_main);

        // 调用native层实现的dump函数
        // 对函数art::DexFile::OpenMemory进行Hook处理
//        Dumpper.dump();

        // 读取配置文件"/sdcard/dumdex.js"获取需要脱壳的apk应用的包名列表
        selected = Config.getConfig();

        // Apk应用信息列表
        appinfos = new ArrayList<>();

        // 用于显示apk应用的列表
        listView = (ListView) findViewById(R.id.applist);
        adapter = new AppAdapter(appinfos);
        // 设置ListView控件的适配器
        listView.setAdapter(adapter);

        // 创建线程
        new Thread(){
            @Override
            public void run() {
                super.run();

                // 获取当前Android系统安装的apk应用列表
                getInstallAppList();
            }
        }.start();
    }

    private void getInstallAppList() {

        try{

         // 获取当前安装应用的PackageInfo列表
            List<PackageInfo> packageInfos = getPackageManager().getInstalledPackages(0);
            // 遍历当前安装应用的PackageInfo列表
            for(PackageInfo packageInfo : packageInfos) {

                Appinfo info = new Appinfo();
                // 设置apk应用的名称
                info.setAppName(packageInfo.applicationInfo.loadLabel(getPackageManager()).toString());
                // 设置apk应用的包名
                info.setAppPackage(packageInfo.packageName);

                // 根据当前遍历到apk应用的包名是否在配置文件中设置选中与否的现实
                if(Config.contains(selected, info.getAppPackage())) {
```

```
                    info.setChecked(true);
                }else {

                    info.setChecked(false);
                }
                // 添加当前遍历到apk应用的信息到apk应用的现实列表中
                appinfos.addAll(info);

                // 更新适配器的现实
                adapter.notifyDataSetChanged();
            }
        }catch (Exception e) {
            e.printStackTrace();
        }
    }

    // ListView列表的适配器
    @SuppressLint({ "ViewHolder", "InflateParams" })
class AppAdapter extends BaseAdapter{

        private List<Appinfo> appinfos;

        public AppAdapter(List<Appinfo> appinfos){
            this.appinfos = appinfos;
        }

        @Override
        public int getCount() {
            return appinfos.size();
        }

        @Override
        public Object getItem(int i) {
            return appinfos.get(i);
        }

        @Override
        public long getItemId(int i) {
            return i;
        }

        @Override
        public View getView(int i, View view, ViewGroup viewGroup) {

            View v = LayoutInflater.from(MainActivity.this).inflate(R.layout.item, null);
            final int posi = i;
            final TextView appname = (TextView) v.findViewById(R.id.tv_appname);
            appname.setText(appinfos.get(i).getAppName());
            TextView appPackage = (TextView) v.findViewById(R.id.tv_apppackage);
            appPackage.setText(appinfos.get(i).getAppPackage());
            CheckBox checkBox = (CheckBox) v.findViewById(R.id.cb_select);

            if(appinfos.get(i).isChecked()) {
                checkBox.setChecked(true);
            }else {
                checkBox.setChecked(false);
            }

            // 监控apk应用列表的选中事件
            checkBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
```

```
                @Override
                public void onCheckedChanged(CompoundButton compoundButton, boolean b) {

                    if(b) {

                     // 添加选中的apk应用的包名到配置文件/sdcard/dumdex.js中
                     // 格式: ["apk包名字符串"]
                        Config.addOne(appinfos.get(posi).getAppPackage());

                    } else {

                     // 从配置文件/sdcard/dumdex.js中删除指定包名的apk引用
                        Config.removeOne(appinfos.get(posi).getAppPackage());
                    }
                }
            });

            return v;
        }
    }
}
```

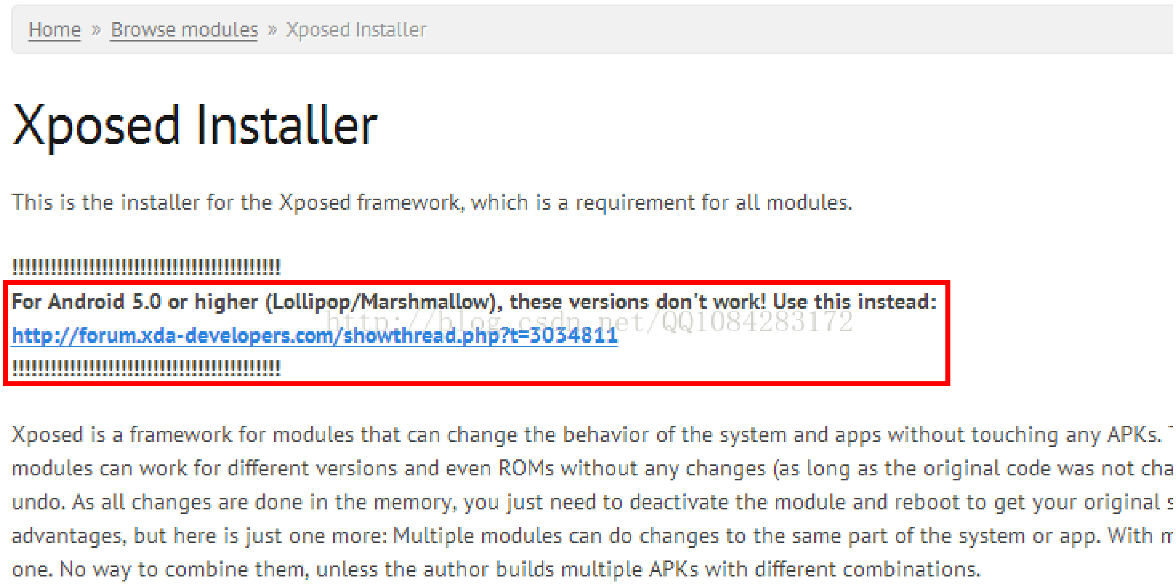
根据用户选择的脱壳apk应用的包名，生成脱壳需要的配置文件dumdex.js文件的代码 Config.java：

```
package com.xposedhook.dexdump;

import android.util.Log;

import org.json.JSONArray;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by smartdone on 2017/7/2.
 */

public class Config {

 // sdcard的问价路径最好还是通过函数来获取
 // File file=Environment.getExternalStorageDirectory();
 // 直接写死有兼容性的问题
    private static final String FILENAME = "/sdcard/dumdex.js";

    // 将JSONArray类型的数据写入到配置文件"/sdcard/dumdex.js"中
    public static void writeConfig(String s) {

        try {

         // 文件"/sdcard/dumdex.js"的文件写入流
            FileOutputStream fout = new FileOutputStream(FILENAME);
            // 将字符串写入到文件中
```

```
            fout.write(s.getBytes("utf-8"));
            // 刷新文件流
            fout.flush();
            fout.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // 添加Apk应用的包名到配置文件/sdcard/dumdex.js
    public static void addOne(String name) {

        List<String> ori = getConfig();
        if(ori == null) {

            JSONArray jsonArray = new JSONArray();
            jsonArray.put(name);
            writeConfig(jsonArray.toString());

        } else {

            ori.add(name);
            JSONArray jsonArray = new JSONArray();
            for(String o : ori) {
                jsonArray.put(o);
            }
            writeConfig(jsonArray.toString());
        }
    }

    // 从配置文件/sdcard/dumdex.js中删除指定包名的应用
    public static void removeOne(String name) {

        List<String> ori = getConfig();

        if(ori != null) {

            for(int i = 0; i < ori.size(); i++) {

                if(ori.get(i).equals(name)) {

                    ori.remove(i);
                }
            }

            JSONArray jsonArray = new JSONArray();
            for(String s : ori) {

                jsonArray.put(s);
            }

            writeConfig(jsonArray.toString());
        }
    }

    // 读取配置文件"/sdcard/dumdex.js"获取需要脱壳的apk应用的包名列表
    public static List<String> getConfig() {
```

```
     // 打开文件"/sdcard/dumdex.js"
        File file = new File(FILENAME);
        // 判断文件是否存在
        if (file.exists()) {
            try {

             // 构建内存缓冲区读取流
                BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(file)));
                // 分行读取文件数据
                String line = br.readLine();
                // 使用读取的一行文件数据构建JSONArray对象
                JSONArray jsonArray = new JSONArray(line);

                List<String> apps = new ArrayList<>();
                // 解析JSONArray数据将需要Hook的apk应用的包名添加到列表中
                for(int i = 0; i < jsonArray.length(); i++) {

                    apps.add(jsonArray.getString(i));
                }

                br.close();
//                Log.e("DEX_DUMP", "需要hook的列表: " + line);

                return apps;

            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        return null;
    }

    // 判断name是否在需要脱壳的apk应用的列表中
    public static boolean contains(List<String> lists, String name) {

        if(lists == null) {
            return false;
        }

        for(String l : lists) {

            if(l.equals(name)) {
                return true;
            }
        }

        return false;
    }

}
```

2. 基于Art虚拟机模式下的Xposed框架 Hook外壳Apk应用 android.app.Application类 的成员函数 attach，这里提到的Xposed Hook框架需要注意一下，不能使用支持Android 4.4.x版本之前的Xposed Hook框架（只支持Dalvik虚拟机模式，不支持Art虚拟机模式），需要使用支持Android 5.0版本以后的Xposed Hook框架（支持Art虚拟机模式），Xposed Hook框架的下载地址可以参考：http://repo.xposed.info/module/de.robv.android.xposed.installer。


Home » Browse modules » Xposed Installer

Xposed Installer

This is the installer for the Xposed framework, which is a requirement for all modules.

!!
For Android 5.0 or higher (Lollipop/Marshmallow), these versions don't work! Use this instead:
http://forum.xda-developers.com/showthread.php?t=3034811
!!

Xposed is a framework for modules that can change the behavior of the system and apps without touching any APKs.
modules can work for different versions and even ROMs without any changes (as long as the original code was not cha
undo. As all changes are done in the memory, you just need to deactivate the module and reboot to get your original s
advantages, but here is just one more: Multiple modules can do changes to the same part of the system or app. With m
one. No way to combine them, unless the author builds multiple APKs with different combinations.

Art虚拟机模式下，Xposed框架 Hook外壳Apk应用 **android.app.Application**类 的成员函数 **attach** 的模块代码 com.xposedhook.dexdump.Main 编写的实现：

```
package com.xposedhook.dexdump;

import android.content.Context;
import android.util.Log;

import java.util.List;

import de.robv.android.xposed.IXposedHookLoadPackage;
import de.robv.android.xposed.XC_MethodHook;
import de.robv.android.xposed.XposedBridge;
import de.robv.android.xposed.XposedHelpers;
import de.robv.android.xposed.callbacks.XC_LoadPackage;

/**
 * Created by smartdone on 2017/7/1.
 */

// art模式下的Xposed Hook
public class Main implements IXposedHookLoadPackage {

    private static final String TAG = "DEX_DUMP";

    @Override
    public void handleLoadPackage(final XC_LoadPackage.LoadPackageParam loadPackageParam) throws Throwable

     // 从配置文件"/sdcard/dumdex.js"中获取需要脱壳的apk应用列表
        List<String> hooklist = Config.getConfig();

        // 判断当前应用是否在需要脱壳的apk应用列表中
        if(!Config.contains(hooklist, loadPackageParam.packageName))
            return;

        XposedBridge.log("对" + loadPackageParam.packageName + "进行处理");
        Log.e(TAG, "开始处理: " + loadPackageParam.packageName);

        try{
```

```
         // 自定义加载动态库文件libhook.so
         // 可以试着使用兼容性好的Android系统函数来处理路径问题
            System.load("/data/data/com.xposedHook.dexdump/lib/libhook.so");

        } catch (Exception e) {
            Log.e(TAG, "加载动态库失败：" + e.getMessage());
        }
        Log.e(TAG, "加载动态库成功");

        // 对Android系统类android.app.Application的attach函数进行art模式下的Hook操作
        XposedHelpers.findAndHookMethod("android.app.Application",
                loadPackageParam.classLoader,
                "attach",
                Context.class,
                new XC_MethodHook() {

            private Context context;
            @Override
            protected void beforeHookedMethod(MethodHookParam param) throws Throwable {

                super.beforeHookedMethod(param);

                // 在类android.app.Application的attach函数调用之前进行dex文件的内存dump操作
                Dumpper.dump();
            }

            @Override
            protected void afterHookedMethod(MethodHookParam param) throws Throwable {
                super.afterHookedMethod(param);
                // 不处理
            }
        });
    }
}
```

3.在需要脱壳的apk应用进程里动态加载动态库文件/data/data/com.xposedHook.dexdump/lib/libhook.so，实现Art模式下对 **const DexFile* DexFile::OpenMemory(const std::string& location, uint32_t location_checksum, MemMap* mem_map, std::string* error_msg)函数** 的Inline Hook操作，在Inline Hook操作的自定义实现函数里进行dex文件的内存dump处理。使用Ele7enxxh Inline Hook框架对Art模式下的OpenMemory函数进行Hook操作的实现代码如下：

```
//
// Created by 袁东明 on 2017/7/1.
//

extern "C" {
#include "include/inlineHook.h"
}

#include "dump.h"
#include <unistd.h>
#include <android/log.h>
#include <sys/system_properties.h>
#include <stdlib.h>
#include <fcntl.h>
#include <time.h>
#include <string>
#include <dlfcn.h>
#include <dlfcn.h>
```

```
#define TAG "DEX_DUMP"

int isArt();
void getProcessName(int pid, char *name, int len);
void dumpFileName(char *name, int len, const char *pname, int dexlen);

// 保存当前apk进程的进程名字
static char pname[256];

// 判断当前所处环境是否是Android art虚拟机模式
int isArt() {

    char version[10];

    // 获取ro.build.version.sdk的属性值
    __system_property_get("ro.build.version.sdk", version);
    // 打印当前Android系统的api版本信息
    __android_log_print(ANDROID_LOG_INFO, TAG, "api level %s", version);

    // 将api版本转换成int型版本号
    int sdk = atoi(version);
    // 判断api版本是否是大于21（要求Android系统的版本为 Android 5.0以上 才可以）
    if (sdk >= 21) {

     // art虚拟机模式
        return 1;
    }

    return 0;
}


// 读取/proc/self/cmdline文件的数据，获取当前apk进程的进程名字
void getProcessName(int pid, char *name, int len) {

    int fp = open("/proc/self/cmdline", O_RDONLY);
    memset(name, 0, len);
    read(fp, name, len);
    close(fp);
}


// 格式字符串构建dump的dex文件的路径字符串
void dumpFileName(char *name, int len, const char *pname, int dexlen) {

    time_t now;
    struct tm *timenow;
    time(&now);
    // 获取当前时间（值得借鉴和学习）
    timenow = localtime(&now);

    memset(name, 0, len);
    // 格式化字符串得到当前dump的dex文件路径字符串
    sprintf(name, "/data/data/%s/dump_size_%u_time_%d_%d_%d_%d_%d_%d.dex", pname, dexlen,
            timenow->tm_year + 1900,
            timenow->tm_mon + 1,
            timenow->tm_mday,
            timenow->tm_hour,
```

```
            timenow->tm_min,
            timenow->tm_sec);
}

void writeToFile(const char *pname, u_int8_t *data, size_t length) {

    char dname[1024];

    // pname为当前进程的名称
    // 格式字符串构建dump的dex文件的路径字符串dname
    dumpFileName(dname, sizeof(dname), pname, length);
    __android_log_print(ANDROID_LOG_ERROR, TAG, "dump dex file name is : %s", dname);

    __android_log_print(ANDROID_LOG_ERROR, TAG, "start dump");
    // 根据dname创建新文件用于保存内存dump的dex文件
    int dex = open(dname, O_CREAT | O_WRONLY, 0644);
    if (dex < 0) {

        __android_log_print(ANDROID_LOG_ERROR, TAG, "open or create file error");
        return;
    }

    // 将内存dex文件的数据写入到新的dname文件中
    int ret = write(dex, data, length);
    if (ret < 0) {

        __android_log_print(ANDROID_LOG_ERROR, TAG, "write file error");
    } else {

        __android_log_print(ANDROID_LOG_ERROR, TAG, "dump dex file success `%s`", dname);
    }

    // 关闭文件
    close(dex);
}

// 保存openmemory函数旧的地址
art::DexFile *(*old_openmemory)(const byte *base, size_t size, const std::string &location,
                                uint32_t location_checksum, art::MemMap *mem_map,
                                const art::OatDexFile *oat_dex_file, std::string *error_msg) = NULL;

art::DexFile *new_openmemory(const byte *base, size_t size, const std::string &location,
                             uint32_t location_checksum, art::MemMap *mem_map,
                             const art::OatDexFile *oat_dex_file, std::string *error_msg) {

    __android_log_print(ANDROID_LOG_ERROR, TAG, "art::DexFile::OpenMemory is called");

    writeToFile(pname, (uint8_t *) base, size);

    // 调用原art::DexFile::OpenMemory函数
    return (*old_openmemory)(base, size, location, location_checksum, mem_map, oat_dex_file,
                             error_msg);
}

void hook() {

 // 加载动态库文件libart.so
    void *handle = dlopen("libart.so", RTLD_GLOBAL | RTLD_LAZY);
    if (handle == NULL) {
```

```
        __android_log_print(ANDROID_LOG_ERROR, TAG, "Error: unable to find the SO : libart.so");
        return;
    }

    // 获取导出函数const DexFile* DexFile::OpenMemory(const std::string& location, uint32_t location_checksum
    // 的调用地址,http://androidxref.com/5.0.0_r2/xref/art/runtime/dex_file.cc#325
    // 在不同的Android版本上OpenMemory函数的名称粉碎稍有不同，需要根据实际Android系统版本进行修改
    void *addr = dlsym(handle,
                       "_ZN3art7DexFile10OpenMemoryEPKhjRKNSt3__112basic_stringIcNS3_11char_traitsIcEENS3_9
    if (addr == NULL) {

        __android_log_print(ANDROID_LOG_ERROR, TAG,
                            "Error: unable to find the Symbol : _ZN3art7DexFile10OpenMemoryEPKhjRKNSt3__112
        return;
    }

    // 使用ele7enxxh写的inline Hook框架对art模式下的art::DexFile::OpenMemory函数进行inline Hook操作
    // 进行art::DexFile::OpenMemory函数inline Hook操作的Hook注册
    if (registerInlineHook((uint32_t) addr, (uint32_t) new_openmemory,
                           (uint32_t **) &old_openmemory) != ELE7EN_OK) {

        __android_log_print(ANDROID_LOG_ERROR, TAG, "register inline hook failed");
        return;
    }

    // 对art模式下的art::DexFile::OpenMemory函数进行inline Hook操作
    if (inlineHook((uint32_t) addr) != ELE7EN_OK) {

        __android_log_print(ANDROID_LOG_ERROR, TAG, "inline hook failed");
        return;
    }

    __android_log_print(ANDROID_LOG_INFO, TAG, "inline hook success");
}


// java方法Dumpper.dump()的native层实现
// com.xposedhook.dexdump.Dumpper.dump
JNIEXPORT void JNICALL Java_com_xposedhook_dexdump_Dumpper_dump(JNIEnv *env, jclass clazz) {

 // 获取当前apk进程的进程名字
    getProcessName(getpid(), pname, sizeof(pname));

    // 判断当前Android虚拟机是否是art模式
    if (isArt()) {

     // 当前Android系统运行在art模式下
     // 执行inline Hook操作
        hook();
    }
}
```

4. 使用当前脱壳工具进行Art虚拟机模式下的Android加固脱壳需要注意的地方：

A. 移动设备的Android系统必须是 Api 21以上即Android 5.0以上版本的Android系统并且要运行在Art虚拟机模式下，不能运行在Dalvik虚拟机模式下。

B. 移动设备上安装的Xposed Hook框架必须是支持Android 5.0以上版本的ART虚拟机模式下的Xposed Hook框架。

C. 第1次使用该脱壳工具com.xposedhook.dexdump.apk时，先使用apk应用列表界面选择需要脱壳的apk应用，生成后面脱壳需要的配置文件"/sdcard/dumdex.js"。

D. 重启移动设备使Xposed框架的Hook模块com.xposedhook.dexdump.Main生效，运行需要脱壳的apk应用等待脱壳完成，脱壳后的dex文件在 /data/data/脱壳apk应用的包名/ 文件夹下。

注释版完整的代码下载地址：http://download.csdn.net/download/qq1084283172/9996172