简述基本原理

在进程启动的时候通过双亲委派机制遍历所有classloader，然后遍历里面的所有class，取出所有函数，直接调用。然后在ArtMethod的Invoke函数这里根据参数判断出这是主动调用触发的，然后就取消函数的正常执行，并执行脱壳操作

fartext改为安卓8实现步骤

修改文件 frameworks/base/core/java/android/app/ActivityThread.java

添加如下代码

import cn.mik.Fartext;

这是自实现的一个类，首先导入

然后在handleBindApplication 合适位置插入 Fartext.fartthread()

private void handleBindApplication(AppBindData data) {

// Register the UI Thread as a sensitive thread to the runtime.

VMRuntime.registerSensitiveThread();

// In the case the stack depth property exists, pass it down to the runtime.

String property = SystemProperties.get("debug.allocTracker.stackDepth");

if (property.length() != 0) {

VMDebug.setAllocTrackerStackDepth(Integer.parseInt(property));

}

if (data.trackAllocation) {

DdmVmInternal.enableRecentAllocations(true);

}

// Note when this process has started.

Process.setStartTimes(SystemClock.elapsedRealtime(), SystemClock.uptimeMillis());

mBoundApplication = data;

mConfiguration = new Configuration(data.config);

mCompatConfiguration = new Configuration(data.config);

mProfiler = new Profiler();

String agent = null;

if (data.initProfilerInfo != null) {

mProfiler.profileFile = data.initProfilerInfo.profileFile;

mProfiler.profileFd = data.initProfilerInfo.profileFd;

mProfiler.samplingInterval = data.initProfilerInfo.samplingInterval;

mProfiler.autoStopProfiler = data.initProfilerInfo.autoStopProfiler;

mProfiler.streamingOutput = data.initProfilerInfo.streamingOutput;

if (data.initProfilerInfo.attachAgentDuringBind) {

agent = data.initProfilerInfo.agent;

}

}

// send up app name; do this \*before\* waiting for debugger

Process.setArgV0(data.processName);

android.ddm.DdmHandleAppName.setAppName(data.processName,

UserHandle.myUserId());

VMRuntime.setProcessPackageName(data.appInfo.packageName);

// Pass data directory path to ART. This is used for caching information and

// should be set before any application code is loaded.

VMRuntime.setProcessDataDirectory(data.appInfo.dataDir);

if (mProfiler.profileFd != null) {

mProfiler.startProfiling();

}

// If the app is Honeycomb MR1 or earlier, switch its AsyncTask

// implementation to use the pool executor. Normally, we use the

// serialized executor as the default. This has to happen in the

// main thread so the main looper is set right.

if (data.appInfo.targetSdkVersion <= android.os.Build.VERSION\_CODES.HONEYCOMB\_MR1) {

AsyncTask.setDefaultExecutor(AsyncTask.THREAD\_POOL\_EXECUTOR);

}

// Let the util.\*Array classes maintain "undefined" for apps targeting Pie or earlier.

UtilConfig.setThrowExceptionForUpperArrayOutOfBounds(

data.appInfo.targetSdkVersion >= Build.VERSION\_CODES.Q);

Message.updateCheckRecycle(data.appInfo.targetSdkVersion);

// Prior to P, internal calls to decode Bitmaps used BitmapFactory,

// which may scale up to account for density. In P, we switched to

// ImageDecoder, which skips the upscale to save memory. ImageDecoder

// needs to still scale up in older apps, in case they rely on the

// size of the Bitmap without considering its density.

ImageDecoder.sApiLevel = data.appInfo.targetSdkVersion;

/\*

\* Before spawning a new process, reset the time zone to be the system time zone.

\* This needs to be done because the system time zone could have changed after the

\* the spawning of this process. Without doing this this process would have the incorrect

\* system time zone.

\*/

TimeZone.setDefault(null);

/\*

\* Set the LocaleList. This may change once we create the App Context.

\*/

LocaleList.setDefault(data.config.getLocales());

synchronized (mResourcesManager) {

/\*

\* Update the system configuration since its preloaded and might not

\* reflect configuration changes. The configuration object passed

\* in AppBindData can be safely assumed to be up to date

\*/

mResourcesManager.applyConfigurationToResourcesLocked(data.config, data.compatInfo);

mCurDefaultDisplayDpi = data.config.densityDpi;

// This calls mResourcesManager so keep it within the synchronized block.

applyCompatConfiguration(mCurDefaultDisplayDpi);

}

data.info = getPackageInfoNoCheck(data.appInfo, data.compatInfo);

if (agent != null) {

handleAttachAgent(agent, data.info);

}

/\*\*

\* Switch this process to density compatibility mode if needed.

\*/

if ((data.appInfo.flags&ApplicationInfo.FLAG\_SUPPORTS\_SCREEN\_DENSITIES)

== 0) {

mDensityCompatMode = true;

Bitmap.setDefaultDensity(DisplayMetrics.DENSITY\_DEFAULT);

}

updateDefaultDensity();

final String use24HourSetting = mCoreSettings.getString(Settings.System.TIME\_12\_24);

Boolean is24Hr = null;

if (use24HourSetting != null) {

is24Hr = "24".equals(use24HourSetting) ? Boolean.TRUE : Boolean.FALSE;

}

// null : use locale default for 12/24 hour formatting,

// false : use 12 hour format,

// true : use 24 hour format.

DateFormat.set24HourTimePref(is24Hr);

updateDebugViewAttributeState();

StrictMode.initThreadDefaults(data.appInfo);

StrictMode.initVmDefaults(data.appInfo);

if (data.debugMode != ApplicationThreadConstants.DEBUG\_OFF) {

// XXX should have option to change the port.

Debug.changeDebugPort(8100);

if (data.debugMode == ApplicationThreadConstants.DEBUG\_WAIT) {

Slog.w(TAG, "Application " + data.info.getPackageName()

+ " is waiting for the debugger on port 8100...");

IActivityManager mgr = ActivityManager.getService();

try {

mgr.showWaitingForDebugger(mAppThread, true);

} catch (RemoteException ex) {

throw ex.rethrowFromSystemServer();

}

Debug.waitForDebugger();

try {

mgr.showWaitingForDebugger(mAppThread, false);

} catch (RemoteException ex) {

throw ex.rethrowFromSystemServer();

}

} else {

Slog.w(TAG, "Application " + data.info.getPackageName()

+ " can be debugged on port 8100...");

}

}

// Allow binder tracing, and application-generated systrace messages if we're profileable.

boolean isAppProfileable = data.appInfo.isProfileableByShell();

Trace.setAppTracingAllowed(isAppProfileable);

if (isAppProfileable && data.enableBinderTracking) {

Binder.enableTracing();

}

// Initialize heap profiling.

if (isAppProfileable || Build.IS\_DEBUGGABLE) {

nInitZygoteChildHeapProfiling();

}

// Allow renderer debugging features if we're debuggable.

boolean isAppDebuggable = (data.appInfo.flags & ApplicationInfo.FLAG\_DEBUGGABLE) != 0;

HardwareRenderer.setDebuggingEnabled(isAppDebuggable || Build.IS\_DEBUGGABLE);

HardwareRenderer.setPackageName(data.appInfo.packageName);

/\*\*

\* Initialize the default http proxy in this process for the reasons we set the time zone.

\*/

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "Setup proxies");

final IBinder b = ServiceManager.getService(Context.CONNECTIVITY\_SERVICE);

if (b != null) {

// In pre-boot mode (doing initial launch to collect password), not

// all system is up. This includes the connectivity service, so don't

// crash if we can't get it.

final IConnectivityManager service = IConnectivityManager.Stub.asInterface(b);

try {

Proxy.setHttpProxySystemProperty(service.getProxyForNetwork(null));

} catch (RemoteException e) {

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

throw e.rethrowFromSystemServer();

}

}

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

// Instrumentation info affects the class loader, so load it before

// setting up the app context.

final InstrumentationInfo ii;

if (data.instrumentationName != null) {

try {

ii = new ApplicationPackageManager(null, getPackageManager())

.getInstrumentationInfo(data.instrumentationName, 0);

} catch (PackageManager.NameNotFoundException e) {

throw new RuntimeException(

"Unable to find instrumentation info for: " + data.instrumentationName);

}

// Warn of potential ABI mismatches.

if (!Objects.equals(data.appInfo.primaryCpuAbi, ii.primaryCpuAbi)

|| !Objects.equals(data.appInfo.secondaryCpuAbi, ii.secondaryCpuAbi)) {

Slog.w(TAG, "Package uses different ABI(s) than its instrumentation: "

+ "package[" + data.appInfo.packageName + "]: "

+ data.appInfo.primaryCpuAbi + ", " + data.appInfo.secondaryCpuAbi

+ " instrumentation[" + ii.packageName + "]: "

+ ii.primaryCpuAbi + ", " + ii.secondaryCpuAbi);

}

mInstrumentationPackageName = ii.packageName;

mInstrumentationAppDir = ii.sourceDir;

mInstrumentationSplitAppDirs = ii.splitSourceDirs;

mInstrumentationLibDir = getInstrumentationLibrary(data.appInfo, ii);

mInstrumentedAppDir = data.info.getAppDir();

mInstrumentedSplitAppDirs = data.info.getSplitAppDirs();

mInstrumentedLibDir = data.info.getLibDir();

} else {

ii = null;

}

final ContextImpl appContext = ContextImpl.createAppContext(this, data.info);

updateLocaleListFromAppContext(appContext,

mResourcesManager.getConfiguration().getLocales());

if (!Process.isIsolated()) {

final int oldMask = StrictMode.allowThreadDiskWritesMask();

try {

setupGraphicsSupport(appContext);

} finally {

StrictMode.setThreadPolicyMask(oldMask);

}

} else {

HardwareRenderer.setIsolatedProcess(true);

}

// Install the Network Security Config Provider. This must happen before the application

// code is loaded to prevent issues with instances of TLS objects being created before

// the provider is installed.

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "NetworkSecurityConfigProvider.install");

NetworkSecurityConfigProvider.install(appContext);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

// Continue loading instrumentation.

if (ii != null) {

ApplicationInfo instrApp;

try {

instrApp = getPackageManager().getApplicationInfo(ii.packageName, 0,

UserHandle.myUserId());

} catch (RemoteException e) {

instrApp = null;

}

if (instrApp == null) {

instrApp = new ApplicationInfo();

}

ii.copyTo(instrApp);

instrApp.initForUser(UserHandle.myUserId());

final LoadedApk pi = getPackageInfo(instrApp, data.compatInfo,

appContext.getClassLoader(), false, true, false);

// The test context's op package name == the target app's op package name, because

// the app ops manager checks the op package name against the real calling UID,

// which is what the target package name is associated with.

final ContextImpl instrContext = ContextImpl.createAppContext(this, pi,

appContext.getOpPackageName());

try {

final ClassLoader cl = instrContext.getClassLoader();

mInstrumentation = (Instrumentation)

cl.loadClass(data.instrumentationName.getClassName()).newInstance();

} catch (Exception e) {

throw new RuntimeException(

"Unable to instantiate instrumentation "

+ data.instrumentationName + ": " + e.toString(), e);

}

final ComponentName component = new ComponentName(ii.packageName, ii.name);

mInstrumentation.init(this, instrContext, appContext, component,

data.instrumentationWatcher, data.instrumentationUiAutomationConnection);

if (mProfiler.profileFile != null && !ii.handleProfiling

&& mProfiler.profileFd == null) {

mProfiler.handlingProfiling = true;

final File file = new File(mProfiler.profileFile);

file.getParentFile().mkdirs();

Debug.startMethodTracing(file.toString(), 8 \* 1024 \* 1024);

}

} else {

mInstrumentation = new Instrumentation();

mInstrumentation.basicInit(this);

}

if ((data.appInfo.flags&ApplicationInfo.FLAG\_LARGE\_HEAP) != 0) {

dalvik.system.VMRuntime.getRuntime().clearGrowthLimit();

} else {

// Small heap, clamp to the current growth limit and let the heap release

// pages after the growth limit to the non growth limit capacity. b/18387825

dalvik.system.VMRuntime.getRuntime().clampGrowthLimit();

}

// Allow disk access during application and provider setup. This could

// block processing ordered broadcasts, but later processing would

// probably end up doing the same disk access.

Application app;

final StrictMode.ThreadPolicy savedPolicy = StrictMode.allowThreadDiskWrites();

final StrictMode.ThreadPolicy writesAllowedPolicy = StrictMode.getThreadPolicy();

try {

// If the app is being launched for full backup or restore, bring it up in

// a restricted environment with the base application class.

app = data.info.makeApplication(data.restrictedBackupMode, null);

// Propagate autofill compat state

app.setAutofillOptions(data.autofillOptions);

// Propagate Content Capture options

app.setContentCaptureOptions(data.contentCaptureOptions);

mInitialApplication = app;

//add

Fartext.fartthread();

//add end

自实现Fartext类

package cn.mik;

import android.app.ActivityThread;

import android.app.Application;

import android.util.Log;

import java.io.BufferedReader;

import java.io.FileReader;

import java.lang.reflect.Constructor;

import java.lang.reflect.Field;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Fartext {

//为了反射封装，根据类名和字段名，反射获取字段

public static Field getClassField(ClassLoader classloader, String class\_name,

String filedName) {

try {

Class obj\_class = classloader.loadClass(class\_name);//Class.forName(class\_name);

Field field = obj\_class.getDeclaredField(filedName);

field.setAccessible(true);

return field;

} catch (SecurityException e) {

e.printStackTrace();

} catch (NoSuchFieldException e) {

e.printStackTrace();

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

return null;

}

public static Object getClassFieldObject(ClassLoader classloader, String class\_name, Object obj,

String filedName) {

try {

Class obj\_class = classloader.loadClass(class\_name);//Class.forName(class\_name);

Field field = obj\_class.getDeclaredField(filedName);

field.setAccessible(true);

Object result = null;

result = field.get(obj);

return result;

//field.setAccessible(true);

//return field;

} catch (SecurityException e) {

e.printStackTrace();

} catch (NoSuchFieldException e) {

e.printStackTrace();

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

}

return null;

}

public static Object invokeStaticMethod(String class\_name,

String method\_name, Class[] pareTyple, Object[] pareVaules) {

try {

Class obj\_class = Class.forName(class\_name);

Method method = obj\_class.getMethod(method\_name, pareTyple);

return method.invoke(null, pareVaules);

} catch (SecurityException e) {

e.printStackTrace();

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (NoSuchMethodException e) {

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

return null;

}

public static Object getFieldObject(String class\_name, Object obj,

String filedName) {

try {

Class obj\_class = Class.forName(class\_name);

Field field = obj\_class.getDeclaredField(filedName);

field.setAccessible(true);

return field.get(obj);

} catch (SecurityException e) {

e.printStackTrace();

} catch (NoSuchFieldException e) {

e.printStackTrace();

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (NullPointerException e) {

e.printStackTrace();

}

return null;

}

public static Application getCurrentApplication(){

Object currentActivityThread = invokeStaticMethod(

"android.app.ActivityThread", "currentActivityThread",

new Class[]{}, new Object[]{});

Object mBoundApplication = getFieldObject(

"android.app.ActivityThread", currentActivityThread,

"mBoundApplication");

Application mInitialApplication = (Application) getFieldObject("android.app.ActivityThread",

currentActivityThread, "mInitialApplication");

Object loadedApkInfo = getFieldObject(

"android.app.ActivityThread$AppBindData",

mBoundApplication, "info");

Application mApplication = (Application) getFieldObject("android.app.LoadedApk", loadedApkInfo, "mApplication");

return mApplication;

}

public static ClassLoader getClassloader() {

ClassLoader resultClassloader = null;

Object currentActivityThread = invokeStaticMethod(

"android.app.ActivityThread", "currentActivityThread",

new Class[]{}, new Object[]{});

Object mBoundApplication = getFieldObject(

"android.app.ActivityThread", currentActivityThread,

"mBoundApplication");

Application mInitialApplication = (Application) getFieldObject("android.app.ActivityThread",

currentActivityThread, "mInitialApplication");

Object loadedApkInfo = getFieldObject(

"android.app.ActivityThread$AppBindData",

mBoundApplication, "info");

Application mApplication = (Application) getFieldObject("android.app.LoadedApk", loadedApkInfo, "mApplication");

Log.e("fartext", "go into app->" + "packagename:" + mApplication.getPackageName());

resultClassloader = mApplication.getClassLoader();

return resultClassloader;

}

//取指定类的所有构造函数，和所有函数，使用dumpMethodCode函数来把这些函数给保存出来

public static void loadClassAndInvoke(ClassLoader appClassloader, String eachclassname, Method dumpMethodCode\_method) {

Class resultclass = null;

Log.e("fartext", "go into loadClassAndInvoke->" + "classname:" + eachclassname);

try {

resultclass = appClassloader.loadClass(eachclassname);

} catch (Exception e) {

e.printStackTrace();

return;

} catch (Error e) {

e.printStackTrace();

return;

}

if (resultclass != null) {

try {

Constructor<?> cons[] = resultclass.getDeclaredConstructors();

for (Constructor<?> constructor : cons) {

if (dumpMethodCode\_method != null) {

try {

if(constructor.getName().contains("cn.mik.")){

continue;

}

Log.e("fartext", "classname:" + eachclassname+ " constructor->invoke "+constructor.getName());

dumpMethodCode\_method.invoke(null, constructor);

} catch (Exception e) {

e.printStackTrace();

continue;

} catch (Error e) {

e.printStackTrace();

continue;

}

} else {

Log.e("fartext", "dumpMethodCode\_method is null ");

}

}

} catch (Exception e) {

e.printStackTrace();

} catch (Error e) {

e.printStackTrace();

}

try {

Method[] methods = resultclass.getDeclaredMethods();

if (methods != null) {

Log.e("fartext", "classname:" + eachclassname+ " start invoke");

for (Method m : methods) {

if (dumpMethodCode\_method != null) {

try {

if(m.getName().contains("cn.mik.")){

continue;

}

Log.e("fartext", "classname:" + eachclassname+ " method->invoke:" + m.getName());

dumpMethodCode\_method.invoke(null, m);

} catch (Exception e) {

e.printStackTrace();

continue;

} catch (Error e) {

e.printStackTrace();

continue;

}

} else {

Log.e("fartext", "dumpMethodCode\_method is null ");

}

}

Log.e("fartext", "go into loadClassAndInvoke->" + "classname:" + eachclassname+ " end invoke");

}

} catch (Exception e) {

e.printStackTrace();

} catch (Error e) {

e.printStackTrace();

}

}

}

//根据classLoader->pathList->dexElements拿到dexFile

//然后拿到mCookie后，使用getClassNameList获取到所有类名。

//loadClassAndInvoke处理所有类名导出所有函数

//dumpMethodCode这个函数是fart自己加在DexFile中的

public static void fartWithClassLoader(ClassLoader appClassloader) {

Log.e("fartext", "fartWithClassLoader "+appClassloader.toString());

List<Object> dexFilesArray = new ArrayList<Object>();

Field paist\_Field = (Field) getClassField(appClassloader, "dalvik.system.BaseDexClassLoader", "pathList");

Object pathList\_object = getFieldObject("dalvik.system.BaseDexClassLoader", appClassloader, "pathList");

Object[] ElementsArray = (Object[]) getFieldObject("dalvik.system.DexPathList", pathList\_object, "dexElements");

Field dexFile\_fileField = null;

try {

dexFile\_fileField = (Field) getClassField(appClassloader, "dalvik.system.DexPathList$Element", "dexFile");

} catch (Exception e) {

e.printStackTrace();

} catch (Error e) {

e.printStackTrace();

}

Class DexFileClazz = null;

try {

DexFileClazz = appClassloader.loadClass("dalvik.system.DexFile");

} catch (Exception e) {

e.printStackTrace();

} catch (Error e) {

e.printStackTrace();

}

Method getClassNameList\_method = null;

Method defineClass\_method = null;

Method dumpDexFile\_method = null;

Method dumpMethodCode\_method = null;

for (Method field : DexFileClazz.getDeclaredMethods()) {

if (field.getName().equals("getClassNameList")) {

getClassNameList\_method = field;

getClassNameList\_method.setAccessible(true);

}

if (field.getName().equals("defineClassNative")) {

defineClass\_method = field;

defineClass\_method.setAccessible(true);

}

if (field.getName().equals("dumpDexFile")) {

dumpDexFile\_method = field;

dumpDexFile\_method.setAccessible(true);

}

if (field.getName().equals("fartextMethodCode")) {

dumpMethodCode\_method = field;

dumpMethodCode\_method.setAccessible(true);

}

}

Field mCookiefield = getClassField(appClassloader, "dalvik.system.DexFile", "mCookie");

Log.e("fartext->methods", "dalvik.system.DexPathList.ElementsArray.length:" + ElementsArray.length);

for (int j = 0; j < ElementsArray.length; j++) {

Object element = ElementsArray[j];

Object dexfile = null;

try {

dexfile = (Object) dexFile\_fileField.get(element);

} catch (Exception e) {

e.printStackTrace();

} catch (Error e) {

e.printStackTrace();

}

if (dexfile == null) {

Log.e("fartext", "dexfile is null");

continue;

}

if (dexfile != null) {

dexFilesArray.add(dexfile);

Object mcookie = getClassFieldObject(appClassloader, "dalvik.system.DexFile", dexfile, "mCookie");

if (mcookie == null) {

Object mInternalCookie = getClassFieldObject(appClassloader, "dalvik.system.DexFile", dexfile, "mInternalCookie");

if(mInternalCookie!=null)

{

mcookie=mInternalCookie;

}else{

Log.e("fartext->err", "get mInternalCookie is null");

continue;

}

}

String[] classnames = null;

try {

classnames = (String[]) getClassNameList\_method.invoke(dexfile, mcookie);

} catch (Exception e) {

e.printStackTrace();

continue;

} catch (Error e) {

e.printStackTrace();

continue;

}

if (classnames != null) {

Log.e("fartext", "all classes "+String.join(",",classnames));

for (String eachclassname : classnames) {

loadClassAndInvoke(appClassloader, eachclassname, dumpMethodCode\_method);

}

}

}

}

return;

}

public static void fart() {

Log.e("fartext", "fart");

ClassLoader appClassloader = getClassloader();

if(appClassloader==null){

Log.e("fartext", "appClassloader is null");

return;

}

ClassLoader tmpClassloader=appClassloader;

ClassLoader parentClassloader=appClassloader.getParent();

if(appClassloader.toString().indexOf("java.lang.BootClassLoader")==-1)

{

fartWithClassLoader(appClassloader);

}

while(parentClassloader!=null){

if(parentClassloader.toString().indexOf("java.lang.BootClassLoader")==-1)

{

fartWithClassLoader(parentClassloader);

}

tmpClassloader=parentClassloader;

parentClassloader=parentClassloader.getParent();

}

}

public static boolean shouldUnpack() {

boolean should\_unpack = false;

String processName = ActivityThread.currentProcessName();

BufferedReader br = null;

String configPath="/data/local/tmp/fext.config";

Log.e("fartext", "shouldUnpack processName:"+processName);

try {

br = new BufferedReader(new FileReader(configPath));

String line;

while ((line = br.readLine()) != null) {

if (processName.equals(line)) {

should\_unpack = true;

break;

}

}

br.close();

}

catch (Exception ex) {

Log.e("fartext", "shouldUnpack err:"+ex.getMessage());

}

return should\_unpack;

}

public static String getClassList() {

String processName = ActivityThread.currentProcessName();

BufferedReader br = null;

String configPath="/data/local/tmp/"+processName;

Log.e("fartext", "getClassList processName:"+processName);

StringBuilder sb=new StringBuilder();

try {

br = new BufferedReader(new FileReader(configPath));

String line;

while ((line = br.readLine()) != null) {

if(line.length()>=2){

sb.append(line+"\n");

}

}

br.close();

}

catch (Exception ex) {

Log.e("fartext", "getClassList err:"+ex.getMessage());

return "";

}

return sb.toString();

}

public static void fartWithClassList(String classlist){

ClassLoader appClassloader = getClassloader();

if(appClassloader==null){

Log.e("fartext", "appClassloader is null");

return;

}

Class DexFileClazz = null;

try {

DexFileClazz = appClassloader.loadClass("dalvik.system.DexFile");

} catch (Exception e) {

e.printStackTrace();

} catch (Error e) {

e.printStackTrace();

}

Method dumpMethodCode\_method = null;

for (Method field : DexFileClazz.getDeclaredMethods()) {

if (field.getName().equals("fartextMethodCode")) {

dumpMethodCode\_method = field;

dumpMethodCode\_method.setAccessible(true);

}

}

String[] classes=classlist.split("\n");

for(String clsname : classes){

String line=clsname;

if(line.startsWith("L")&&line.endsWith(";")&&line.contains("/")){

line=line.substring(1,line.length()-1);

line=line.replace("/",".");

}

loadClassAndInvoke(appClassloader, line, dumpMethodCode\_method);

}

}

public static void fartthread() {

if (!shouldUnpack()) {

return;

}

String classlist=getClassList();

if(!classlist.equals("")){

fartWithClassList(classlist);

return;

}

new Thread(new Runnable() {

@Override

public void run() {

// TODO Auto-generated method stub

try {

Log.e("fartext", "start sleep......");

Thread.sleep(1 \* 60 \* 1000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

Log.e("fartext", "sleep over and start fart");

fart();

Log.e("fartext", "fart run over");

}

}).start();

}

}

在libcore/DexFile.java文件下添加代码

private static native void fartextMethodCode(Object m);

接下来在art/runtime/native/dalvik\_system\_DexFile.cc文件中实现fartextMethodCode方法并注册

#include "scoped\_fast\_native\_object\_access.h"

extern "C" void fartextInvoke(ArtMethod\* artmethod);

extern "C" ArtMethod\* jobject2ArtMethod(JNIEnv\* env, jobject javaMethod);

static void DexFile\_fartextMethodCode(JNIEnv\* env, jclass,jobject method) {

if(method!=nullptr)

{

ArtMethod\* proxy\_method = jobject2ArtMethod(env, method);

fartextInvoke(proxy\_method);

}

return;

}

在JNINativeMethod gMethods[]添加注册

NATIVE\_METHOD(DexFile, fartextMethodCode,

"(Ljava/lang/Object;)V")

在art/runtime/art\_method.cc 中实现fartextInvoke并

添加头文件

#include <sys/syscall.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include "runtime.h"

#include <android/log.h>

#include <assert.h>

#include <errno.h>

#include <fcntl.h>

#include <pthread.h>

#include <stdarg.h>

#include <stddef.h>

#include <stdlib.h>

#include <fstream>

#include <iostream>

#include <string>

#include <sys/mman.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <sys/uio.h>

#include <sys/un.h>

#include <time.h>

#include <unistd.h>

#define gettidv1() syscall(\_\_NR\_gettid)

#define LOG\_TAG "ActivityThread"

#define ALOGI(...) \_\_android\_log\_print(ANDROID\_LOG\_INFO, LOG\_TAG, \_\_VA\_ARGS\_\_)

实现fartextinvoke以及一些其他的方法

uint8\_t\* codeitem\_end(const uint8\_t \*\*pData)

{

uint32\_t num\_of\_list = DecodeUnsignedLeb128(pData);

for (;num\_of\_list>0;num\_of\_list--) {

int32\_t num\_of\_handlers=DecodeSignedLeb128(pData);

int num=num\_of\_handlers;

if (num\_of\_handlers<=0) {

num=-num\_of\_handlers;

}

for (; num > 0; num--) {

DecodeUnsignedLeb128(pData);

DecodeUnsignedLeb128(pData);

}

if (num\_of\_handlers<=0) {

DecodeUnsignedLeb128(pData);

}

}

return (uint8\_t\*)(\*pData);

}

extern "C" char \*base64\_encode(char \*str,long str\_len,long\* outlen){

long len;

char \*res;

int i,j;

const char \*base64\_table="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

if(str\_len % 3 == 0)

len=str\_len/3\*4;

else

len=(str\_len/3+1)\*4;

res=(char\*)malloc(sizeof(char)\*(len+1));

res[len]='\0';

\*outlen=len;

for(i=0,j=0;i<len-2;j+=3,i+=4)

{

res[i]=base64\_table[str[j]>>2];

res[i+1]=base64\_table[(str[j]&0x3)<<4 | (str[j+1]>>4)];

res[i+2]=base64\_table[(str[j+1]&0xf)<<2 | (str[j+2]>>6)];

res[i+3]=base64\_table[str[j+2]&0x3f];

}

switch(str\_len % 3)

{

case 1:

res[i-2]='=';

res[i-1]='=';

break;

case 2:

res[i-1]='=';

break;

}

return res;

}

//在函数即将调用解释器执行前进行dump。

extern "C" void dumpdexfilebyExecute(ArtMethod\* artmethod) REQUIRES\_SHARED(Locks::mutator\_lock\_) {

char \*dexfilepath=(char\*)malloc(sizeof(char)\*1000);

if(dexfilepath==nullptr)

{

return;

}

int result=0;

int fcmdline =-1;

char szCmdline[64]= {0};

char szProcName[256] = {0};

int procid = getpid();

sprintf(szCmdline,"/proc/%d/cmdline", procid);

fcmdline = open(szCmdline, O\_RDONLY,0644);

if(fcmdline >0)

{

result=read(fcmdline, szProcName,256);

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,open cmdline file error";

}

close(fcmdline);

}

if(szProcName[0])

{

const DexFile\* dex\_file = artmethod->GetDexFile();

const uint8\_t\* begin\_=dex\_file->Begin(); // Start of data.

size\_t size\_=dex\_file->Size(); // Length of data.

memset(dexfilepath,0,1000);

int size\_int\_=(int)size\_;

memset(dexfilepath,0,1000);

sprintf(dexfilepath,"%s","/sdcard/fext");

mkdir(dexfilepath,0777);

memset(dexfilepath,0,1000);

sprintf(dexfilepath,"/sdcard/fext/%s",szProcName);

mkdir(dexfilepath,0777);

memset(dexfilepath,0,1000);

sprintf(dexfilepath,"/sdcard/fext/%s/%d\_dexfile\_execute.dex",szProcName,size\_int\_);

int dexfilefp=open(dexfilepath,O\_RDONLY,0666);

if(dexfilefp>0){

close(dexfilefp);

dexfilefp=0;

}else{

int fp=open(dexfilepath,O\_CREAT|O\_APPEND|O\_RDWR,0666);

if(fp>0)

{

result=write(fp,(void\*)begin\_,size\_);

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,open dexfilepath error";

}

fsync(fp);

close(fp);

memset(dexfilepath,0,1000);

sprintf(dexfilepath,"/sdcard/fext/%s/%d\_classlist\_execute.txt",szProcName,size\_int\_);

int classlistfile=open(dexfilepath,O\_CREAT|O\_APPEND|O\_RDWR,0666);

if(classlistfile>0)

{

for (size\_t ii= 0; ii< dex\_file->NumClassDefs(); ++ii)

{

const dex::ClassDef& class\_def = dex\_file->GetClassDef(ii);

const char\* descriptor = dex\_file->GetClassDescriptor(class\_def);

result=write(classlistfile,(void\*)descriptor,strlen(descriptor));

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,write classlistfile file error";

}

const char\* temp="\n";

result=write(classlistfile,(void\*)temp,1);

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,write classlistfile file error";

}

}

fsync(classlistfile);

close(classlistfile);

}

}

}

}

if(dexfilepath!=nullptr)

{

free(dexfilepath);

dexfilepath=nullptr;

}

}

extern "C" bool ShouldUnpack() {

int result=0;

int fcmdline =-1;

char szCmdline[64]= {0};

char szProcName[256] = {0};

int procid = getpid();

sprintf(szCmdline,"/proc/%d/cmdline", procid);

fcmdline = open(szCmdline, O\_RDONLY,0644);

if(fcmdline >0)

{

result=read(fcmdline, szProcName,256);

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::ShouldUnpack,open cmdline file file error";

}

close(fcmdline);

}

if(szProcName[0]){

const char\* UNPACK\_CONFIG = "/data/local/tmp/fext.config";

std::ifstream config(UNPACK\_CONFIG);

std::string line;

if(config) {

while (std::getline(config, line)) {

std::string package\_name = line.substr(0, line.find(':'));

if (strstr(package\_name.c\_str(),szProcName)) {

return true;

}

}

}

return false;

}

return false;

}

//主动调用函数的dump处理

extern "C" void dumpArtMethod(ArtMethod\* artmethod) REQUIRES\_SHARED(Locks::mutator\_lock\_) {

LOG(ERROR) << "fartext ArtMethod::dumpArtMethod enter "<<artmethod->PrettyMethod().c\_str();

char \*dexfilepath=(char\*)malloc(sizeof(char)\*1000);

if(dexfilepath==nullptr)

{

return;

}

int result=0;

int fcmdline =-1;

char szCmdline[64]= {0};

char szProcName[256] = {0};

int procid = getpid();

sprintf(szCmdline,"/proc/%d/cmdline", procid);

fcmdline = open(szCmdline, O\_RDONLY,0644);

if(fcmdline >0)

{

result=read(fcmdline, szProcName,256);

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,open cmdline file file error";

}

close(fcmdline);

}

if(szProcName[0])

{

const DexFile\* dex\_file = artmethod->GetDexFile();

const uint8\_t\* begin\_=dex\_file->Begin(); // Start of data.

size\_t size\_=dex\_file->Size(); // Length of data.

memset(dexfilepath,0,1000);

int size\_int\_=(int)size\_;

memset(dexfilepath,0,1000);

sprintf(dexfilepath,"%s","/sdcard/fext");

mkdir(dexfilepath,0777);

memset(dexfilepath,0,1000);

sprintf(dexfilepath,"/sdcard/fext/%s",szProcName);

mkdir(dexfilepath,0777);

memset(dexfilepath,0,1000);

sprintf(dexfilepath,"/sdcard/fext/%s/%d\_dexfile.dex",szProcName,size\_int\_);

int dexfilefp=open(dexfilepath,O\_RDONLY,0666);

if(dexfilefp>0){

close(dexfilefp);

dexfilefp=0;

}else{

int fp=open(dexfilepath,O\_CREAT|O\_APPEND|O\_RDWR,0666);

if(fp>0)

{

result=write(fp,(void\*)begin\_,size\_);

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,open dexfilepath file error";

}

fsync(fp);

close(fp);

memset(dexfilepath,0,1000);

sprintf(dexfilepath,"/sdcard/fext/%s/%d\_classlist.txt",szProcName,size\_int\_);

int classlistfile=open(dexfilepath,O\_CREAT|O\_APPEND|O\_RDWR,0666);

if(classlistfile>0)

{

for (size\_t ii= 0; ii< dex\_file->NumClassDefs(); ++ii)

{

const dex::ClassDef& class\_def = dex\_file->GetClassDef(ii);

const char\* descriptor = dex\_file->GetClassDescriptor(class\_def);

result=write(classlistfile,(void\*)descriptor,strlen(descriptor));

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,write classlistfile file error";

}

const char\* temp="\n";

result=write(classlistfile,(void\*)temp,1);

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,write classlistfile file error";

}

}

fsync(classlistfile);

close(classlistfile);

}

}

}

const dex::CodeItem\* code\_item = artmethod->GetCodeItem();

//todo 注意这下面，aosp10修改的 对于CodeItem的成员访问方式发生了变化，需要参考aosp8

const DexFile\* dex\_=artmethod->GetDexFile();

CodeItemDataAccessor accessor(\*dex\_, dex\_->GetCodeItem(artmethod->GetCodeItemOffset()));

if (LIKELY(code\_item != nullptr))

{

int code\_item\_len = 0;

uint8\_t \*item=(uint8\_t \*) code\_item;

if (accessor.TriesSize()>0) {

const uint8\_t \*handler\_data = accessor.GetCatchHandlerData();

uint8\_t \* tail = codeitem\_end(&handler\_data);

code\_item\_len = (int)(tail - item);

}else{

code\_item\_len = 16+accessor.InsnsSizeInCodeUnits()\*2;

}

//todo 结束

memset(dexfilepath,0,1000);

int size\_int=(int)dex\_file->Size();

uint32\_t method\_idx=artmethod->GetDexMethodIndex();

sprintf(dexfilepath,"/sdcard/fext/%s/%d\_ins\_%d.bin",szProcName,size\_int,(int)gettidv1());

int fp2=open(dexfilepath,O\_CREAT|O\_APPEND|O\_RDWR,0666);

if(fp2>0){

lseek(fp2,0,SEEK\_END);

memset(dexfilepath,0,1000);

int offset=(int)(item - begin\_);

sprintf(dexfilepath,"{name:%s,method\_idx:%d,offset:%d,code\_item\_len:%d,ins:",artmethod->PrettyMethod().c\_str(),method\_idx,offset,code\_item\_len);

int contentlength=0;

while(dexfilepath[contentlength]!=0) contentlength++;

result=write(fp2,(void\*)dexfilepath,contentlength);

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,write ins file error";

}

long outlen=0;

char\* base64result=base64\_encode((char\*)item,(long)code\_item\_len,&outlen);

result=write(fp2,base64result,outlen);

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,write ins file error";

}

result=write(fp2,"};",2);

if(result<0)

{

LOG(ERROR) << "fartext ArtMethod::dumpdexfilebyArtMethod,write ins file error";

}

fsync(fp2);

close(fp2);

if(base64result!=nullptr){

free(base64result);

base64result=nullptr;

}

}

}

}

if(dexfilepath!=nullptr)

{

free(dexfilepath);

dexfilepath=nullptr;

}

LOG(ERROR) << "fartext ArtMethod::dumpArtMethod over "<<artmethod->PrettyMethod().c\_str();

}

extern "C" void fartextInvoke(ArtMethod\* artmethod) REQUIRES\_SHARED(Locks::mutator\_lock\_) {

if(artmethod->IsNative()||artmethod->IsAbstract()){

return;

}

JValue result;

Thread \*self=Thread::Current();

uint32\_t temp[100]={0};

uint32\_t\* args=temp;

uint32\_t args\_size = (uint32\_t)ArtMethod::NumArgRegisters(artmethod->GetShorty());

if (!artmethod->IsStatic()) {

args\_size += 1;

}

result.SetI(111111);

LOG(ERROR) << "fartext fartextInvoke";

artmethod->Invoke(self, args, args\_size, &result,artmethod->GetShorty());

}

在void ArtMethod::Invoke函数中添加判断是否时主动调用

Runtime\* runtime = Runtime::Current();

// Call the invoke stub, passing everything as arguments.

// If the runtime is not yet started or it is required by the debugger, then perform the

// Invocation by the interpreter, explicitly forcing interpretation over JIT to prevent

// cycling around the various JIT/Interpreter methods that handle method invocation.

//add

if ((result!=nullptr && result->GetI()==111111)&&!IsNative()){

const dex::CodeItem\* code\_item =this->GetCodeItem();

if(LIKELY(code\_item!=nullptr)){

if (IsStatic()) {

LOG(ERROR) << "fartext artMethod::Invoke Static Method "<<this->PrettyMethod().c\_str();

art::interpreter::EnterInterpreterFromInvoke(

self, this, nullptr, args, result, /\*stay\_in\_interpreter=\*/ true);

}else{

LOG(ERROR) << "fartext artMethod::Invoke Method "<<this->PrettyMethod().c\_str();

art::interpreter::EnterInterpreterFromInvoke(

self, this, nullptr, args + 1, result, /\*stay\_in\_interpreter=\*/ true);

}

self->PopManagedStackFragment(fragment);

}

return;

}

//add end

if (UNLIKELY(!runtime->IsStarted() ||

Dbg::IsForcedInterpreterNeededForCalling(self, this))) {

if (IsStatic()) {

art::interpreter::EnterInterpreterFromInvoke(

self, this, nullptr, args, result, /\*stay\_in\_interpreter=\*/ true);

} else {

mirror::Object\* receiver =

reinterpret\_cast<StackReference<mirror::Object>\*>(&args[0])->AsMirrorPtr();

art::interpreter::EnterInterpreterFromInvoke(

self, this, receiver, args + 1, result, /\*stay\_in\_interpreter=\*/ true);

}

} else {

if (result!=nullptr && result->GetI()==111111){

LOG(ERROR) << "fartext artMethod::Invoke return Native Method "<<this->PrettyMethod().c\_str();

return;

}

修改art/runtime/native/java\_lang\_reflect\_Method.cc 实现 jobject2ArtMethod

namespace art {

//add

extern "C" ArtMethod\* jobject2ArtMethod(JNIEnv\* env, jobject javaMethod) {

ScopedFastNativeObjectAccess soa(env);

ArtMethod\* method = ArtMethod::FromReflectedMethod(soa, javaMethod);

return method;

}

//add end

接下来修改art/runtime/interpreter/interpreter.cc 更深层的调用

导入函数

extern "C" bool ShouldUnpack();

namespace interpreter {

//add

extern "C" void dumpdexfilebyExecute(ArtMethod\* artmethod);

//addend

判断是否时主动调用

static inline JValue Execute(

Thread\* self,

const CodeItemDataAccessor& accessor,

ShadowFrame& shadow\_frame,

JValue result\_register,

bool stay\_in\_interpreter = false,

bool from\_deoptimize = false) REQUIRES\_SHARED(Locks::mutator\_lock\_) {

DCHECK(!shadow\_frame.GetMethod()->IsAbstract());

DCHECK(!shadow\_frame.GetMethod()->IsNative());

//add

if(result\_register.GetI()==111111){

LOG(ERROR) << "fartext Execute start "<<shadow\_frame.GetMethod()->PrettyMethod().c\_str();

}

if(strstr(shadow\_frame.GetMethod()->PrettyMethod().c\_str(),"<clinit>"))

{

if(ShouldUnpack()){

dumpdexfilebyExecute(shadow\_frame.GetMethod());

}

}

在void EnterInterpreterFromInvoke 函数下添加 （有个问题是上面这种模拟参数的方式，碰到引用类型的参数会报错。所以在处理参数入栈的时候，也要进行判断处理一下）

self->PushShadowFrame(shadow\_frame);

size\_t cur\_reg = num\_regs - num\_ins;

if (!method->IsStatic()) {

//add

if(result!=nullptr&&result->GetI()==111111){

shadow\_frame->SetVReg(cur\_reg, args[0]);

}else{

CHECK(receiver != nullptr);

shadow\_frame->SetVRegReference(cur\_reg, receiver);

}

//add end

//shadow\_frame->SetVRegReference(cur\_reg, receiver);

++cur\_reg;

}

uint32\_t shorty\_len = 0;

const char\* shorty = method->GetShorty(&shorty\_len);

for (size\_t shorty\_pos = 0, arg\_pos = 0; cur\_reg < num\_regs; ++shorty\_pos, ++arg\_pos, cur\_reg++) {

DCHECK\_LT(shorty\_pos + 1, shorty\_len);

switch (shorty[shorty\_pos + 1]) {

case 'L': {

//add

if(result!=nullptr&&result->GetI()==111111){

shadow\_frame->SetVReg(cur\_reg, args[0]);

break;

}

//add end

处理art/runtime/interpreter/interpreter\_switch\_impl-inl.h 指令集判断进行更深层次的调用

发现没有这个文件，这可咋办？？？？