# VirtualAPP - 流程分析

# VirtualAPP - 简介

[VirtualAPP](https://github.com/asLody/VirtualApp) [new VirtualAPP](https://github.com/ServenScorpion/VirtualApp.git)

VirtualAPP(简称：VA)是一款运行于Android系统的沙盒产品，可以理解为轻量级的“Android虚拟机”。其产品形态为高可扩展，可定制的集成SDK，您可以基于VA或者使用VA定制开发各种看似不可能完成的项目。VA目前被广泛应用于APP多开、小游戏合集、手游加速器、手游租号、手游手柄免激活、VR程序移植、区块链、移动办公安全、军队政府数据隔离、手机模拟信息、脚本自动化、插件化开发、无感知热更新、云控等技术领域。  
**Github上代码已在2017年12月份停止更新，商业版代码在持续更新中**

## VirtualAPP - 核心原理

* 安装到VA内部的APP实际上并**没有安装到系统**中,所以正常情况下是无法运行的。
* **欺骗**系统，让系统认为已经安装。而这个“欺骗”过程就是VA Framework的核心工作内容，也是整个VA的核心技术原理。

# VirtualAPP - 基础架构



## VirtualAPP - VA Space

* 由VA提供了一个内部的空间，用于**安装要在其内部运行的APP**，这个空间是**系统隔离**的。

## VirtualAPP - VA Framework

* 这一层主要给Android Framework和VAPP做代理，这也是VA的核心。
* **VA提供了一套自己的VA Framework**，处于Android **Framework与VA APP**之间。

1. 对于VAPP，其访问的**所有系统Service均已被 VA Framework 代理**，它会**修改VAPP的请求参数**，将其中与VAPP安装信息相关的全部参数修改为宿主的参数之后发送给Android Framework（有部分请求会发送给自己的VA Server直接处理而不再发送给Android系统）。这样Android Framework收到VAPP请求后检查参数就会认为没有问题。
2. 待Android系统对该请求处理完成返回结果时，VA Framework同样也会**拦截住该返回结果**，此时再将原来修改过的参数全部还原为VAPP请求时发送的。
3. 这样VAPP与Android系统的交互也就能跑通了。

## VirtualAPP - VA Native

* 这一层主要为了完成2个工作，**IO重定向**和**VA APP与Android系统交互的请求修改**。

1. IO重定向是因为可能有部分APP会通过写死的绝对路径访问，但是如果APP没有安装到系统，这个路径是不存在的，通过IO重定向，则将其转向VA内部安装的路径。(**重定向io路径**)
2. 另外有部分jni函数在VA Framework中无法hook的，所以需要在native层来做hook。

## VirtualAPP - 架构总结

1. app并没有被安装到系统.
2. VA app 和framework之间做了一层代理,劫持了app所访问的系统服务处理输入输出.
3. VA 做了io重定向,实现app与系统隔离.

# VirtualAPP - 源码分析

* 由3个VA接口为切入点.

1. VirtualCore.get().startup(); // 启动VA引擎
2. VirtualCore.get().installPackageAsUser(userId, packageName); // 将目标APP安装到VA中
3. VActivityManager.get().launchApp(userId, packageName); // 启动APP

## VirtualAPP - Java层Hook 实现原理

* **核心是使用Java动态代理去实现Java层的hook**
* **前置知识,需要对Java动态代理有一定了解**

1. 定义一些接口,使用动态代理:lib\src\main\java\com\lody\virtual\client\hook\base\MethodProxy.java

public boolean beforeCall(Object who, Method method, Object... args) {

return true;

}

public Object call(Object who, Method method, Object... args) throws Throwable {

return method.invoke(who, args);

}

public Object afterCall(Object who, Method method, Object[] args, Object result) throws Throwable {

return result;

}

1. 一些需要代理的方法: lib\src\main\java\com\lody\virtual\client\hook\proxies\am\MethodProxies.java

// 案例一

static class GetServices extends MethodProxy {

@Override

public String getMethodName() {

// 要 Hook 的方法名

return "getServices";

}

@Override

public Object call(Object who, Method method, Object... args) throws Throwable {

int maxNum = (int) args[0];

int flags = (int) args[1];

// 返回由VAMS 去获取的服务，而不是原始的AMS ActivityManager 去获取的服务

return VActivityManager.get().getServices(maxNum, flags).getList();

}

@Override

public boolean isEnable() {

// 是否需要hook 这里返回的是 当前app是否为VASpace 的app 如果返回false 则不hook该方法

return isAppProcess();

}

}

// 案例二

static class CheckPermission extends MethodProxy {

@Override

public String getMethodName() {

return "checkPermission";

}

// 自己去实现想要执行的代码和逻辑

@Override

public Object call(Object who, Method method, Object... args) throws Throwable {

String permission = (String) args[0];

if (SpecialComponentList.isWhitePermission(permission)) {

return PackageManager.PERMISSION\_GRANTED;

}

if (permission.startsWith("com.google")) {

return PackageManager.PERMISSION\_GRANTED;

}

args[args.length - 1] = getRealUid();

return method.invoke(who, args);

}

@Override

public boolean isEnable() {

return isAppProcess();

}

}

// startActivity 方法 hook

static class StartActivity extends MethodProxy {

private static final String SCHEME\_FILE = "file";

private static final String SCHEME\_PACKAGE = "package";

private static final String SCHEME\_CONTENT = "content";

@Override

public String getMethodName() {

return "startActivity";

}

@Override

public Object call(Object who, Method method, Object... args) throws Throwable {

Log.d("Q\_M", "---->StartActivity 类");

int intentIndex = ArrayUtils.indexOfObject(args, Intent.class, 1);

if (intentIndex < 0) {

return ActivityManagerCompat.START\_INTENT\_NOT\_RESOLVED;

}

int resultToIndex = ArrayUtils.indexOfObject(args, IBinder.class, 2);

String resolvedType = (String) args[intentIndex + 1];

Intent intent = (Intent) args[intentIndex];

intent.setDataAndType(intent.getData(), resolvedType);

IBinder resultTo = resultToIndex >= 0 ? (IBinder) args[resultToIndex] : null;

int userId = VUserHandle.myUserId();

if (ComponentUtils.isStubComponent(intent)) {

return method.invoke(who, args);

}

if (Intent.ACTION\_INSTALL\_PACKAGE.equals(intent.getAction())

|| (Intent.ACTION\_VIEW.equals(intent.getAction())

&& "application/vnd.android.package-archive".equals(intent.getType()))) {

if (handleInstallRequest(intent)) {

return 0;

}

} else if ((Intent.ACTION\_UNINSTALL\_PACKAGE.equals(intent.getAction())

|| Intent.ACTION\_DELETE.equals(intent.getAction()))

&& "package".equals(intent.getScheme())) {

if (handleUninstallRequest(intent)) {

return 0;

}

}

String resultWho = null;

int requestCode = 0;

Bundle options = ArrayUtils.getFirst(args, Bundle.class);

if (resultTo != null) {

resultWho = (String) args[resultToIndex + 1];

requestCode = (int) args[resultToIndex + 2];

}

// chooser

if (ChooserActivity.check(intent)) {

intent.setComponent(new ComponentName(getHostContext(), ChooserActivity.class));

intent.putExtra(Constants.EXTRA\_USER\_HANDLE, userId);

intent.putExtra(ChooserActivity.EXTRA\_DATA, options);

intent.putExtra(ChooserActivity.EXTRA\_WHO, resultWho);

intent.putExtra(ChooserActivity.EXTRA\_REQUEST\_CODE, requestCode);

return method.invoke(who, args);

}

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.JELLY\_BEAN\_MR2) {

args[intentIndex - 1] = getHostPkg();

}

if (intent.getScheme() != null && intent.getScheme().equals(SCHEME\_PACKAGE) && intent.getData() != null) {

if (intent.getAction() != null && intent.getAction().startsWith("android.settings.")) {

intent.setData(Uri.parse("package:" + getHostPkg()));

}

}

ActivityInfo activityInfo = VirtualCore.get().resolveActivityInfo(intent, userId);

if (activityInfo == null) {

VLog.e("VActivityManager", "Unable to resolve activityInfo : " + intent);

Log.d("Q\_M", "---->StartActivity who=" + who);

Log.d("Q\_M", "---->StartActivity intent=" + intent);

Log.d("Q\_M", "---->StartActivity resultTo=" + resultTo);

if (intent.getPackage() != null && isAppPkg(intent.getPackage())) {

return ActivityManagerCompat.START\_INTENT\_NOT\_RESOLVED;

}

if (INTERCEPT\_BACK\_HOME && Intent.ACTION\_MAIN.equals(intent.getAction())

&& intent.getCategories().contains("android.intent.category.HOME")

&& resultTo != null) {

VActivityManager.get().finishActivity(resultTo);

return 0;

}

return method.invoke(who, args);

}

int res = VActivityManager.get().startActivity(intent, activityInfo, resultTo, options, resultWho, requestCode, VUserHandle.myUserId());

if (res != 0 && resultTo != null && requestCode > 0) {

VActivityManager.get().sendActivityResult(resultTo, resultWho, requestCode);

}

if (resultTo != null) {

ActivityClientRecord r = VActivityManager.get().getActivityRecord(resultTo);

if (r != null && r.activity != null) {

try {

TypedValue out = new TypedValue();

Resources.Theme theme = r.activity.getResources().newTheme();

theme.applyStyle(activityInfo.getThemeResource(), true);

if (theme.resolveAttribute(android.R.attr.windowAnimationStyle, out, true)) {

TypedArray array = theme.obtainStyledAttributes(out.data,

new int[]{

android.R.attr.activityOpenEnterAnimation,

android.R.attr.activityOpenExitAnimation

});

r.activity.overridePendingTransition(array.getResourceId(0, 0), array.getResourceId(1, 0));

array.recycle();

}

} catch (Throwable e) {

// Ignore

}

}

}

return res;

}

1. 用于批量管理代理类:lib\src\main\java\com\lody\virtual\client\hook\base\MethodInvocationStub.java

// 保存所有hook的方法对象

public Map<String, MethodProxy> getAllHooks() {

return mInternalMethodProxies;

}

// 使用addMethodProxy 方法去添加需要代理的 方法对象.

public MethodProxy addMethodProxy(MethodProxy methodProxy) {

// 防止重复添加

if (methodProxy != null && !TextUtils.isEmpty(methodProxy.getMethodName())) {

if (mInternalMethodProxies.containsKey(methodProxy.getMethodName())) {

VLog.w(TAG, "The Hook(%s, %s) you added has been in existence.", methodProxy.getMethodName(),

methodProxy.getClass().getName());

return methodProxy;

}

mInternalMethodProxies.put(methodProxy.getMethodName(), methodProxy);

}

return methodProxy;

}

// 初始化时 创建代理对象并统一处理回调HookInvocationHandler

public MethodInvocationStub(T baseInterface, Class<?>... proxyInterfaces) {

this.mBaseInterface = baseInterface;

if (baseInterface != null) {

if (proxyInterfaces == null) {

proxyInterfaces = MethodParameterUtils.getAllInterface(baseInterface.getClass());

}

// 创建代理对象

mProxyInterface = (T) Proxy.newProxyInstance(baseInterface.getClass().getClassLoader(), proxyInterfaces, new HookInvocationHandler());

} else {

VLog.d(TAG, "Unable to build HookDelegate: %s.", getIdentityName());

}

}

// 核心回调HookInvocationHandler处理

private class HookInvocationHandler implements InvocationHandler {

@Override

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

// 获取代理的对象

MethodProxy methodProxy = getMethodProxy(method.getName());

// 判断是否代理

boolean useProxy = (methodProxy != null && methodProxy.isEnable());

boolean mightLog = (mInvocationLoggingCondition != LogInvocation.Condition.NEVER) ||

(methodProxy != null && methodProxy.getInvocationLoggingCondition() != LogInvocation.Condition.NEVER);

String argStr = null;

Object res = null;

Throwable exception = null;

if (mightLog) {

// Arguments to string is done before the method is called because the method might actually change it

argStr = Arrays.toString(args);

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

}

try {

// 这里判断走代理自己实现的逻辑 还是走原方法的逻辑

if (useProxy && methodProxy.beforeCall(mBaseInterface, method, args)) {

res = methodProxy.call(mBaseInterface, method, args);

res = methodProxy.afterCall(mBaseInterface, method, args, res);

} else {

// 这里走原方法的逻辑

res = method.invoke(mBaseInterface, args);

}

return res;

* 另一种方式注解 hook java lib\src\main\java\com\lody\virtual\client\hook\proxies\libcore\LibCoreStub.java

@Inject(MethodProxies.class)

public class LibCoreStub extends MethodInvocationProxy<MethodInvocationStub<Object>> {

public LibCoreStub() {

super(new MethodInvocationStub<Object>(getOs()));

}

private static Object getOs() {

Object os = Libcore.os.get();

if (ForwardingOs.os != null) {

Object posix = ForwardingOs.os.get(os);

if (posix != null) {

os = posix;

}

}

return os;

}

@Override

protected void onBindMethods() {

// 调用 父类 MethodInvocationProxy -> onBindMethods()

super.onBindMethods();

addMethodProxy(new ReplaceUidMethodProxy("chown", 1));

addMethodProxy(new ReplaceUidMethodProxy("fchown", 1));

addMethodProxy(new ReplaceUidMethodProxy("getpwuid", 0));

addMethodProxy(new ReplaceUidMethodProxy("lchown", 1));

addMethodProxy(new ReplaceUidMethodProxy("setuid", 0));

}

// 父类 MethodInvocationProxy

protected void onBindMethods() {

if (mInvocationStub == null) {

return;

}

Class<? extends MethodInvocationProxy> clazz = getClass();

Inject inject = clazz.getAnnotation(Inject.class);

if (inject != null) {

Class<?> proxiesClass = inject.value();

Class<?>[] innerClasses = proxiesClass.getDeclaredClasses();

// 遍历内部类

for (Class<?> innerClass : innerClasses) {

if (!Modifier.isAbstract(innerClass.getModifiers())

&& MethodProxy.class.isAssignableFrom(innerClass)

&& innerClass.getAnnotation(SkipInject.class) == null) {

// 最终会 添加需要代理的 方法对象.

addMethodProxy(innerClass);

}

}

}

}

## VirtualAPP - 添加所有hook

* 在 VirtualCore.get().startup(base);中
* invocationStubManager.init(); .injectAll();
* 完成所有VAService服务 framework 的hook

public void startup(Context context) throws Throwable {

InvocationStubManager invocationStubManager = InvocationStubManager.getInstance();

invocationStubManager.init();

invocationStubManager.injectAll();

.....................................................

private void injectInternal() throws Throwable {

if (VirtualCore.get().isMainProcess()) {

return;

}

if (VirtualCore.get().isServerProcess()) {

addInjector(new ActivityManagerStub());

addInjector(new PackageManagerStub());

return;

}

if (VirtualCore.get().isVAppProcess()) {

addInjector(new LibCoreStub());

addInjector(new ActivityManagerStub());

addInjector(new PackageManagerStub());

addInjector(HCallbackStub.getDefault());

addInjector(new ISmsStub());

addInjector(new ISubStub());

addInjector(new DropBoxManagerStub());

addInjector(new NotificationManagerStub());

addInjector(new LocationManagerStub());

addInjector(new WindowManagerStub());

addInjector(new ClipBoardStub());

addInjector(new MountServiceStub());

addInjector(new BackupManagerStub());

addInjector(new TelephonyStub());

addInjector(new TelephonyRegistryStub());

addInjector(new PhoneSubInfoStub());

addInjector(new PowerManagerStub());

addInjector(new AppWidgetManagerStub());

addInjector(new AccountManagerStub());

addInjector(new AudioManagerStub());

addInjector(new SearchManagerStub());

addInjector(new ContentServiceStub());

addInjector(new ConnectivityStub());

if (Build.VERSION.SDK\_INT >= JELLY\_BEAN\_MR2) {

addInjector(new VibratorStub());

addInjector(new WifiManagerStub());

addInjector(new BluetoothStub());

addInjector(new ContextHubServiceStub());

}

if (Build.VERSION.SDK\_INT >= JELLY\_BEAN\_MR1) {

addInjector(new UserManagerStub());

}

if (Build.VERSION.SDK\_INT >= JELLY\_BEAN\_MR1) {

addInjector(new DisplayStub());

}

if (Build.VERSION.SDK\_INT >= LOLLIPOP) {

addInjector(new PersistentDataBlockServiceStub());

addInjector(new InputMethodManagerStub());

addInjector(new MmsStub());

addInjector(new SessionManagerStub());

addInjector(new JobServiceStub());

addInjector(new RestrictionStub());

}

..............................................

void injectAll() throws Throwable {

for (IInjector injector : mInjectors.values()) {

injector.inject();

}

// XXX: Lazy inject the Instrumentation,

addInjector(AppInstrumentation.getDefault());

}

# VirtualAPP - NativeHook分析

[参考文档](https://toutiao.io/posts/ryt03gi/preview)

# VirtualAPP - 安装APP分析

[参考文档](https://blog.csdn.net/ganyao939543405/article/details/76150725)

# VirtualAPP - startActivity

* 在java层hook中有提到startActivity hook
* 在重写的call方法中可以看到VActivityManager.get().startActivity() 调用了VAMS的同名方法
* 这里拦截了安装卸载在VA中处理

// 在 com.lody.virtual.client.hook.proxies.am.MethodProxies中

static class StartActivity extends MethodProxy {

private static final String SCHEME\_FILE = "file";

private static final String SCHEME\_PACKAGE = "package";

private static final String SCHEME\_CONTENT = "content";

@Override

public String getMethodName() {

return "startActivity";

}

@Override

public Object call(Object who, Method method, Object... args) throws Throwable {

Log.d("Q\_M", "---->StartActivity 类");

int intentIndex = ArrayUtils.indexOfObject(args, Intent.class, 1);

if (intentIndex < 0) {

return ActivityManagerCompat.START\_INTENT\_NOT\_RESOLVED;

}

int resultToIndex = ArrayUtils.indexOfObject(args, IBinder.class, 2);

String resolvedType = (String) args[intentIndex + 1];

Intent intent = (Intent) args[intentIndex];

intent.setDataAndType(intent.getData(), resolvedType);

IBinder resultTo = resultToIndex >= 0 ? (IBinder) args[resultToIndex] : null;

int userId = VUserHandle.myUserId();

if (ComponentUtils.isStubComponent(intent)) {

return method.invoke(who, args);

}

if (Intent.ACTION\_INSTALL\_PACKAGE.equals(intent.getAction())

|| (Intent.ACTION\_VIEW.equals(intent.getAction())

&& "application/vnd.android.package-archive".equals(intent.getType()))) {

if (handleInstallRequest(intent)) {

return 0;

}

} else if ((Intent.ACTION\_UNINSTALL\_PACKAGE.equals(intent.getAction())

|| Intent.ACTION\_DELETE.equals(intent.getAction()))

&& "package".equals(intent.getScheme())) {

if (handleUninstallRequest(intent)) {

return 0;

}

}

String resultWho = null;

int requestCode = 0;

Bundle options = ArrayUtils.getFirst(args, Bundle.class);

if (resultTo != null) {

resultWho = (String) args[resultToIndex + 1];

requestCode = (int) args[resultToIndex + 2];

}

// chooser

if (ChooserActivity.check(intent)) {

intent.setComponent(new ComponentName(getHostContext(), ChooserActivity.class));

intent.putExtra(Constants.EXTRA\_USER\_HANDLE, userId);

intent.putExtra(ChooserActivity.EXTRA\_DATA, options);

intent.putExtra(ChooserActivity.EXTRA\_WHO, resultWho);

intent.putExtra(ChooserActivity.EXTRA\_REQUEST\_CODE, requestCode);

return method.invoke(who, args);

}

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.JELLY\_BEAN\_MR2) {

args[intentIndex - 1] = getHostPkg();

}

if (intent.getScheme() != null && intent.getScheme().equals(SCHEME\_PACKAGE) && intent.getData() != null) {

if (intent.getAction() != null && intent.getAction().startsWith("android.settings.")) {

intent.setData(Uri.parse("package:" + getHostPkg()));

}

}

ActivityInfo activityInfo = VirtualCore.get().resolveActivityInfo(intent, userId);

if (activityInfo == null) {

VLog.e("VActivityManager", "Unable to resolve activityInfo : " + intent);

Log.d("Q\_M", "---->StartActivity who=" + who);

Log.d("Q\_M", "---->StartActivity intent=" + intent);

Log.d("Q\_M", "---->StartActivity resultTo=" + resultTo);

if (intent.getPackage() != null && isAppPkg(intent.getPackage())) {

return ActivityManagerCompat.START\_INTENT\_NOT\_RESOLVED;

}

if (INTERCEPT\_BACK\_HOME && Intent.ACTION\_MAIN.equals(intent.getAction())

&& intent.getCategories().contains("android.intent.category.HOME")

&& resultTo != null) {

VActivityManager.get().finishActivity(resultTo);

return 0;

}

return method.invoke(who, args);

}

int res = VActivityManager.get().startActivity(intent, activityInfo, resultTo, options, resultWho, requestCode, VUserHandle.myUserId());

if (res != 0 && resultTo != null && requestCode > 0) {

VActivityManager.get().sendActivityResult(resultTo, resultWho, requestCode);

}

if (resultTo != null) {

ActivityClientRecord r = VActivityManager.get().getActivityRecord(resultTo);

if (r != null && r.activity != null) {

try {

TypedValue out = new TypedValue();

Resources.Theme theme = r.activity.getResources().newTheme();

theme.applyStyle(activityInfo.getThemeResource(), true);

if (theme.resolveAttribute(android.R.attr.windowAnimationStyle, out, true)) {

TypedArray array = theme.obtainStyledAttributes(out.data,

new int[]{

android.R.attr.activityOpenEnterAnimation,

android.R.attr.activityOpenExitAnimation

});

r.activity.overridePendingTransition(array.getResourceId(0, 0), array.getResourceId(1, 0));

array.recycle();

}

} catch (Throwable e) {

// Ignore

}

}

}

return res;

}

private boolean handleInstallRequest(Intent intent) {

IAppRequestListener listener = VirtualCore.get().getAppRequestListener();

if (listener != null) {

Uri packageUri = intent.getData();

if (SCHEME\_FILE.equals(packageUri.getScheme())) {

File sourceFile = new File(packageUri.getPath());

try {

listener.onRequestInstall(sourceFile.getPath());

return true;

} catch (RemoteException e) {

e.printStackTrace();

}

} else if (SCHEME\_CONTENT.equals(packageUri.getScheme())) {

InputStream inputStream = null;

OutputStream outputStream = null;

File sharedFileCopy = new File(getHostContext().getCacheDir(), packageUri.getLastPathSegment());

try {

inputStream = getHostContext().getContentResolver().openInputStream(packageUri);

outputStream = new FileOutputStream(sharedFileCopy);

byte[] buffer = new byte[1024];

int count;

while ((count = inputStream.read(buffer)) > 0) {

outputStream.write(buffer, 0, count);

}

outputStream.flush();

} catch (IOException e) {

e.printStackTrace();

} finally {

FileUtils.closeQuietly(inputStream);

FileUtils.closeQuietly(outputStream);

}

try {

listener.onRequestInstall(sharedFileCopy.getPath());

return true;

} catch (RemoteException e) {

e.printStackTrace();

}

}

}

return false;

}

private boolean handleUninstallRequest(Intent intent) {

IAppRequestListener listener = VirtualCore.get().getAppRequestListener();

if (listener != null) {

Uri packageUri = intent.getData();

if (SCHEME\_PACKAGE.equals(packageUri.getScheme())) {

String pkg = packageUri.getSchemeSpecificPart();

try {

listener.onRequestUninstall(pkg);

return true;

} catch (RemoteException e) {

e.printStackTrace();

}

}

}

return false;

}

}

* 在com.lody.virtual.server.am.VActivityManagerService 的startActivity 最终调用了startActivityLocked -> startActivityProcess

@Override

public int startActivity(Intent intent, ActivityInfo info, IBinder resultTo, Bundle options, String resultWho, int requestCode, int userId) {

synchronized (this) {

return mMainStack.startActivityLocked(userId, intent, info, resultTo, options, resultWho, requestCode);

}

}

* 继续到startActivityProcess
* startProcessIfNeedLocked: 启动并分配进程
* fetchStubActivity: 会根据当前的情况，根据vpid去清单文件寻找合适的StubActivity，同一进程下的StubActivity信息会被多次复用
* saveToIntent(targetIntent);将ClientAPP要启动的目标Activity的信息保存在intent中，而此时的intent已经被包装为启动StubActivity的intent。

private Intent startActivityProcess(int userId, ActivityRecord sourceRecord, Intent intent, ActivityInfo info) {

intent = new Intent(intent);

ProcessRecord targetApp = mService.startProcessIfNeedLocked(info.processName, userId, info.packageName);

if (targetApp == null) {

return null;

}

Intent targetIntent = new Intent();

targetIntent.setClassName(VirtualCore.get().getHostPkg(), fetchStubActivity(targetApp.vpid, info));

ComponentName component = intent.getComponent();

if (component == null) {

component = ComponentUtils.toComponentName(info);

}

targetIntent.setType(component.flattenToString());

StubActivityRecord saveInstance = new StubActivityRecord(intent, info,

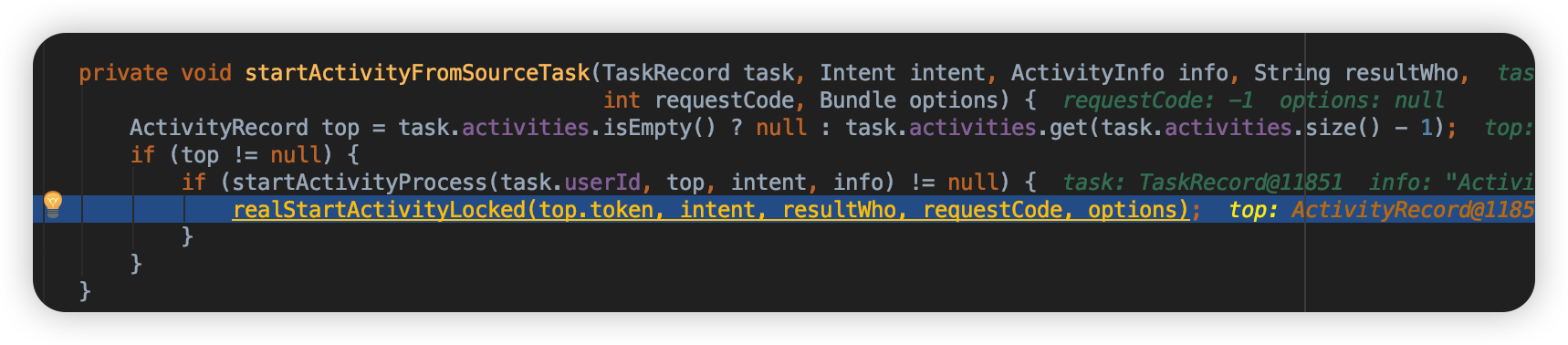
sourceRecord != null ? sourceRecord.component : null, userId);

saveInstance.saveToIntent(targetIntent);

return targetIntent;

}

* 后面走到 realStartActivityLocked



* 经过对参数的替换处理，最终将伪造的StubActivity启动intent交给了系统AMS。

private void realStartActivityLocked(IBinder resultTo, Intent intent, String resultWho, int requestCode,

Bundle options) {

Class<?>[] types = mirror.android.app.IActivityManager.startActivity.paramList();

Object[] args = new Object[types.length];

if (types[0] == IApplicationThread.TYPE) {

args[0] = ActivityThread.getApplicationThread.call(VirtualCore.mainThread());

}

int intentIndex = ArrayUtils.protoIndexOf(types, Intent.class);

int resultToIndex = ArrayUtils.protoIndexOf(types, IBinder.class, 2);

int optionsIndex = ArrayUtils.protoIndexOf(types, Bundle.class);

int resolvedTypeIndex = intentIndex + 1;

int resultWhoIndex = resultToIndex + 1;

int requestCodeIndex = resultToIndex + 2;

// 替换参数

args[intentIndex] = intent;

args[resultToIndex] = resultTo;

args[resultWhoIndex] = resultWho;

args[requestCodeIndex] = requestCode;

if (optionsIndex != -1) {

args[optionsIndex] = options;

}

args[resolvedTypeIndex] = intent.getType();

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.JELLY\_BEAN\_MR2) {

args[intentIndex - 1] = VirtualCore.get().getHostPkg();

}

ClassUtils.fixArgs(types, args);

// 调用系统原本的startActivity

mirror.android.app.IActivityManager.startActivity.call(ActivityManagerNative.getDefault.call(),

(Object[]) args);

}

* 经过系统AMS处理后又回到Clientapp主线程的Handle中
* handleLaunchActivity:

@Override

public boolean handleMessage(Message msg) {

if (!mCalling) {

mCalling = true;

try {

if (LAUNCH\_ACTIVITY == msg.what) {

if (!handleLaunchActivity(msg)) {

return true;

}

..................

private boolean handleLaunchActivity(Message msg) {

Object r = msg.obj;

// StubActivity的intent

Intent stubIntent = ActivityThread.ActivityClientRecord.intent.get(r);

StubActivityRecord saveInstance = new StubActivityRecord(stubIntent);

if (saveInstance.intent == null) {

return true;

}

// clientapp 的intent

Intent intent = saveInstance.intent;

ComponentName caller = saveInstance.caller;

IBinder token = ActivityThread.ActivityClientRecord.token.get(r);

ActivityInfo info = saveInstance.info;

if (VClientImpl.get().getToken() == null) {

InstalledAppInfo installedAppInfo = VirtualCore.get().getInstalledAppInfo(info.packageName, 0);

if(installedAppInfo == null){

return true;

}

VActivityManager.get().processRestarted(info.packageName, info.processName, saveInstance.userId);

getH().sendMessageAtFrontOfQueue(Message.obtain(msg));

return false;

}

if (!VClientImpl.get().isBound()) {

VClientImpl.get().bindApplication(info.packageName, info.processName);

getH().sendMessageAtFrontOfQueue(Message.obtain(msg));

return false;

}

int taskId = IActivityManager.getTaskForActivity.call(

ActivityManagerNative.getDefault.call(),

token,

false

);

VActivityManager.get().onActivityCreate(ComponentUtils.toComponentName(info), caller, token, info, intent, ComponentUtils.getTaskAffinity(info), taskId, info.launchMode, info.flags);

ClassLoader appClassLoader = VClientImpl.get().getClassLoader(info.applicationInfo);

intent.setExtrasClassLoader(appClassLoader);

//真正替换intent信息的地方，此时intent信息为原版目标信息，又交还给了系统处理

ActivityThread.ActivityClientRecord.intent.set(r, intent);

ActivityThread.ActivityClientRecord.activityInfo.set(r, info);

return true;

}

@Override

public void inject() throws Throwable {

otherCallback = getHCallback();

mirror.android.os.Handler.mCallback.set(getH(), this);

}

# VirtualAPP - Service

## VirtualAPP - VAService 管理

* 所有 VAService 直接继承与 XXX.Stub
* 通过 ServiceCache 保存所有服务

public class ServiceCache {

private static final Map<String, IBinder> sCache = new ArrayMap<>(5);

public static void addService(String name, IBinder service) {

sCache.put(name, service);

}

public static IBinder removeService(String name) {

return sCache.remove(name);

}

public static IBinder getService(String name) {

return sCache.get(name);

}

}

* 在 BinderProvider 的 onCreate() 回掉中被addService 添加
* lib/src/main/java/com/lody/virtual/server/BinderProvider.java

@Override

public boolean onCreate() {

Context context = getContext();

// 这是一个空前台服务，目的是为了保活 VAService 进程，即 :x 进程

DaemonService.startup(context);

if (!VirtualCore.get().isStartup()) {

return true;

}

VPackageManagerService.systemReady();

addService(ServiceManagerNative.PACKAGE, VPackageManagerService.get());

VActivityManagerService.systemReady(context);

addService(ServiceManagerNative.ACTIVITY, VActivityManagerService.get());

addService(ServiceManagerNative.USER, VUserManagerService.get());

VAppManagerService.systemReady();

addService(ServiceManagerNative.APP, VAppManagerService.get());

BroadcastSystem.attach(VActivityManagerService.get(), VAppManagerService.get());

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.LOLLIPOP) {

addService(ServiceManagerNative.JOB, VJobSchedulerService.get());

}

VNotificationManagerService.systemReady(context);

addService(ServiceManagerNative.NOTIFICATION, VNotificationManagerService.get());

VAppManagerService.get().scanApps();

VAccountManagerService.systemReady();

addService(ServiceManagerNative.ACCOUNT, VAccountManagerService.get());

addService(ServiceManagerNative.VS, VirtualStorageService.get());

addService(ServiceManagerNative.DEVICE, VDeviceManagerService.get());

return true;

}

private void addService(String name, IBinder service) {

ServiceCache.addService(name, service);

}

* 使用ServiceFetcher 向外部暴露VAService 中所有服务的IBinder句柄 本身也是Binder服务.
* ServiceFetcher 的 Ibinder 句柄是拿到其他 VAServic IBinder 的钥匙

private class ServiceFetcher extends IServiceFetcher.Stub {

@Override

public IBinder getService(String name) throws RemoteException {

if (name != null) {

return ServiceCache.getService(name);

}

return null;

}

@Override

public void addService(String name, IBinder service) throws RemoteException {

if (name != null && service != null) {

ServiceCache.addService(name, service);

}

}

@Override

public void removeService(String name) throws RemoteException {

if (name != null) {

ServiceCache.removeService(name);

}

}

}

1. ServiceFetcher 自身的 IBnder 则通过 BinderProvider 获取

public final class BinderProvider extends ContentProvider {

private final ServiceFetcher mServiceFetcher = new ServiceFetcher();

public Bundle call(String method, String arg, Bundle extras) {

if ("@".equals(method)) {

Bundle bundle = new Bundle();

BundleCompat.putBinder(bundle, "\_VA\_|\_binder\_", mServiceFetcher);

return bundle;

}

return null;

}

// ServiceManagerNative -> getServiceFetcher() 获取一个IServiceFetcher

public class ServiceManagerNative {

public static final String PACKAGE = "package";

public static final String ACTIVITY = "activity";

public static final String USER = "user";

public static final String APP = "app";

public static final String ACCOUNT = "account";

public static final String JOB = "job";

public static final String NOTIFICATION = "notification";

public static final String VS = "vs";

public static final String DEVICE = "device";

public static final String SERVICE\_DEF\_AUTH = "virtual.service.BinderProvider";

private static final String TAG = ServiceManagerNative.class.getSimpleName();

public static String SERVICE\_CP\_AUTH = "virtual.service.BinderProvider";

// 通过 ContentProvider 传递一个

private static IServiceFetcher sFetcher;

private static IServiceFetcher getServiceFetcher() {

if (sFetcher == null || !sFetcher.asBinder().isBinderAlive()) {

synchronized (ServiceManagerNative.class) {

Context context = VirtualCore.get().getContext();

Bundle response = new ProviderCall.Builder(context, SERVICE\_CP\_AUTH).methodName("@").call();

if (response != null) {

IBinder binder = BundleCompat.getBinder(response, "\_VA\_|\_binder\_");

linkBinderDied(binder);

sFetcher = IServiceFetcher.Stub.asInterface(binder);

}

}

}

return sFetcher;

}

// 返回服务的 IBinder 句柄

public static IBinder getService(String name) {

// 如果是本地服务，直接本地返回

if (VirtualCore.get().isServerProcess()) {

return ServiceCache.getService(name);

}

// 通过 ServiceFetcher 的句柄找到远程 Service 的句柄

IServiceFetcher fetcher = getServiceFetcher();

if (fetcher != null) {

try {

return fetcher.getService(name);

} catch (RemoteException e) {

e.printStackTrace();

}

}

VLog.e(TAG, "GetService(%s) return null.", name);

return null;

}

## VirtualAPP - service简介

由于Service并不像Activity那样有交互有页面，它的生命周期非常简单，并且ClientApp中的Service并不需要暴露给外部（指沙盒外）App使用，因此在VA中，Service不需要让AMS等系统服务知晓。

通过hook startService方法将逻辑直接引导到VAMS中，利用ApplicationThread.scheduleCreateService对目标Service直接进行创建（如果目标进程不存在则拉起进程），如果是bindService则使用ApplicationThread.scheduleBindService等待bind完成。

简而言之就是Client App中Service的创建运行过程，对系统服务屏蔽，利用mirror手段直接调用其生命周期。

### VirtualAPP - startService 分析

* startService 和startActivity一样都是在 MethodProxies.java 去代理实现hook
* 最终调用VAMS VActivityManager.get().startService

static class StartService extends MethodProxy {

@Override

public String getMethodName() {

return "startService";

}

@Override

public Object call(Object who, Method method, Object... args) throws Throwable {

IInterface appThread = (IInterface) args[0];

Intent service = (Intent) args[1];

String resolvedType = (String) args[2];

if (service.getComponent() != null

&& getHostPkg().equals(service.getComponent().getPackageName())) {

// for server process

return method.invoke(who, args);

}

int userId = VUserHandle.myUserId();

if (service.getBooleanExtra("\_VA\_|\_from\_inner\_", false)) {

userId = service.getIntExtra("\_VA\_|\_user\_id\_", userId);

service = service.getParcelableExtra("\_VA\_|\_intent\_");

} else {

if (isServerProcess()) {

userId = service.getIntExtra("\_VA\_|\_user\_id\_", VUserHandle.USER\_NULL);

}

}

service.setDataAndType(service.getData(), resolvedType);

ServiceInfo serviceInfo = VirtualCore.get().resolveServiceInfo(service, VUserHandle.myUserId());

if (serviceInfo != null) {

// 最终调用VAMS的startService实现

return VActivityManager.get().startService(appThread, service, resolvedType, userId);

}

return method.invoke(who, args);

}

@Override

public boolean isEnable() {

return isAppProcess() || isServerProcess();

}

}

* 再来看一下VAMS 的startService
* com.lody.virtual.server.am.VActivityManagerService
* 通过 scheduleCreateService创建service后将 ServiceRecord 记录到 Service 列表

@Override

public ComponentName startService(IBinder caller, Intent service, String resolvedType, int userId) {

synchronized (this) {

return startServiceCommon(service, true, userId);

}

}

private ComponentName startServiceCommon(Intent service,

boolean scheduleServiceArgs, int userId) {

ServiceInfo serviceInfo = resolveServiceInfo(service, userId);

if (serviceInfo == null) {

return null;

}

// 检查 Application 是否初始化，没有则开始初始化 Application 流程

ProcessRecord targetApp = startProcessIfNeedLocked(ComponentUtils.getProcessName(serviceInfo),

userId,

serviceInfo.packageName);

if (targetApp == null) {

VLog.e(TAG, "Unable to start new Process for : " + ComponentUtils.toComponentName(serviceInfo));

return null;

}

IInterface appThread = targetApp.appThread;

ServiceRecord r = findRecordLocked(userId, serviceInfo);

if (r == null) {

r = new ServiceRecord();

r.startId = 0;

r.activeSince = SystemClock.elapsedRealtime();

r.process = targetApp;

r.serviceInfo = serviceInfo;

try {

// 调用 ApplicationThread.scheduleCreateService 直接创建 Service

IApplicationThreadCompat.scheduleCreateService(appThread, r, r.serviceInfo, 0);

} catch (RemoteException e) {

e.printStackTrace();

}

addRecord(r);

}

r.lastActivityTime = SystemClock.uptimeMillis();

if (scheduleServiceArgs) {

r.startId++;

boolean taskRemoved = serviceInfo.applicationInfo != null

&& serviceInfo.applicationInfo.targetSdkVersion < Build.VERSION\_CODES.ECLAIR;

try {

IApplicationThreadCompat.scheduleServiceArgs(appThread, r, taskRemoved, r.startId, 0, service);

} catch (RemoteException e) {

e.printStackTrace();

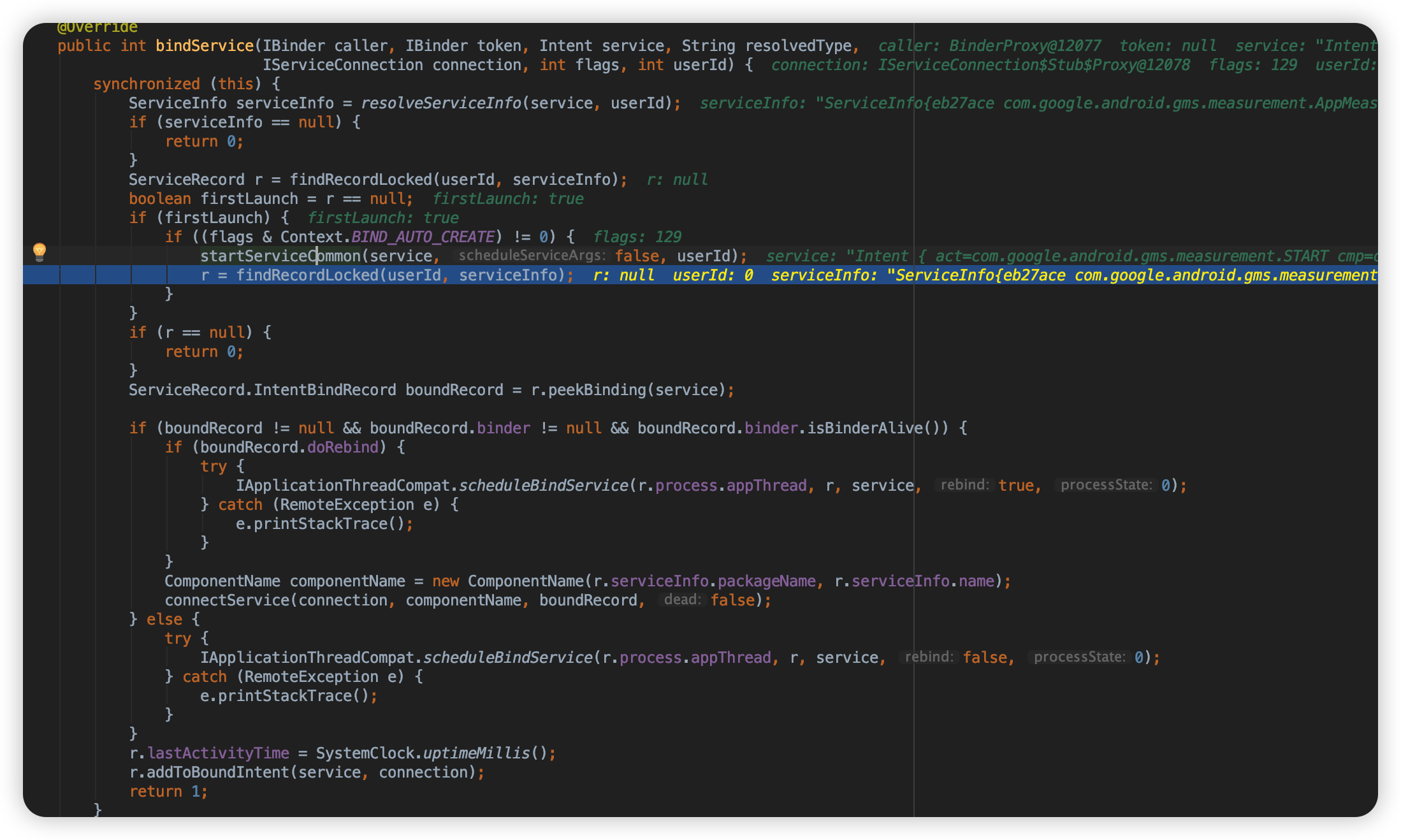
}

}

return ComponentUtils.toComponentName(serviceInfo);

}

* 之后进入bindService流程



# VirtualAPP - BroadcastReceiver

由于广播分静态注册和动态注册，而ClientApp中的静态广播无法被系统AMS所知晓，因此VA使用动态注册来代替静态注册。

当VA Service进程拉起时，VA对所有已安装的APP进行扫描，遍历其所有的Reveiver信息，通过新建StaticBroadcastReceiver代理来接收每个IntentFilter。

然后hook了ClientApp中的broadcastIntent方法，这里其实也做了一个intent包裹，和Activity逻辑类似，当代理广播StaticBroadcastReceiver接收到包装后的intent信息时，解包出真正的intent，回调到ClientApp空间，实例化目标 Receiver（如果目标进程不存在则拉起进程），最终手动进行对其OnCreate和finish的调用。

## VirtualAPP - Broadcast接收者

* 在安装clinetapp的时候 com.lody.virtual.server.pm.VAppManagerService#installPackage(java.lang.String, int)
* 最后会调用BroadcastSystem.get().startApp(pkg);
* 扫描所有clientapp 静态注册的接受者,使用动态注册去实现.

public void startApp(VPackage p) {

PackageSetting setting = (PackageSetting) p.mExtras;

// 遍历 Client App 的 Receiver

for (VPackage.ActivityComponent receiver : p.receivers) {

ActivityInfo info = receiver.info;

// 得到对应 Client App 在 VAService 中的记录列表

List<BroadcastReceiver> receivers = mReceivers.get(p.packageName);

if (receivers == null) {

receivers = new ArrayList<>();

mReceivers.put(p.packageName, receivers);

}

// 拼接自定义的 IntentFilter 根据包名和原接收者

String componentAction = String.format("\_VA\_%s\_%s", info.packageName, info.name);

IntentFilter componentFilter = new IntentFilter(componentAction);

//

BroadcastReceiver r = new StaticBroadcastReceiver(setting.appId, info, componentFilter);

// 调用hook过的 动态注册广播接收者 注册.

mContext.registerReceiver(r, componentFilter, null, mScheduler);

// 添加记录

receivers.add(r);

// 遍历 Intent-Filter，每个 Intent-Filter 注册一个 StaticBroadcastReceiver 代理

for (VPackage.ActivityIntentInfo ci : receiver.intents) {

IntentFilter cloneFilter = new IntentFilter(ci.filter);

SpecialComponentList.protectIntentFilter(cloneFilter);

r = new StaticBroadcastReceiver(setting.appId, info, cloneFilter);

mContext.registerReceiver(r, cloneFilter, null, mScheduler);

receivers.add(r);

}

}

}

* StaticBroadcastReceiver 通过handleStaticBroadcast 分发数据

private final class StaticBroadcastReceiver extends BroadcastReceiver {

private int appId;

private ActivityInfo info;

@SuppressWarnings("unused")

private IntentFilter filter;

private StaticBroadcastReceiver(int appId, ActivityInfo info, IntentFilter filter) {

this.appId = appId;

this.info = info;

this.filter = filter;

}

@Override

public void onReceive(Context context, Intent intent) {

if (mApp.isBooting()) {

return;

}

if ((intent.getFlags() & FLAG\_RECEIVER\_REGISTERED\_ONLY) != 0 || isInitialStickyBroadcast()) {

return;

}

String privilegePkg = intent.getStringExtra("\_VA\_|\_privilege\_pkg\_");

if (privilegePkg != null && !info.packageName.equals(privilegePkg)) {

return;

}

PendingResult result = goAsync();

if (!mAMS.handleStaticBroadcast(appId, info, intent, new PendingResultData(result))) {

result.finish();

}

}

}

}

* handleStaticBroadcast:从这个方法开始取出真正的Intent componet 一直回到clientclient.scheduleReceiver发送RECEIVER 消息

boolean handleStaticBroadcast(int appId, ActivityInfo info, Intent intent,

PendingResultData result) {

Intent realIntent = intent.getParcelableExtra("\_VA\_|\_intent\_");

ComponentName component = intent.getParcelableExtra("\_VA\_|\_component\_");

int userId = intent.getIntExtra("\_VA\_|\_user\_id\_", VUserHandle.USER\_NULL);

if (realIntent == null) {

return false;

}

if (userId < 0) {

VLog.w(TAG, "Sent a broadcast without userId " + realIntent);

return false;

}

int vuid = VUserHandle.getUid(userId, appId);

return handleUserBroadcast(vuid, info, component, realIntent, result);

}

private boolean handleUserBroadcast(int vuid, ActivityInfo info, ComponentName component, Intent realIntent, PendingResultData result) {

if (component != null && !ComponentUtils.toComponentName(info).equals(component)) {

// Verify the component.

return false;

}

String originAction = SpecialComponentList.unprotectAction(realIntent.getAction());

if (originAction != null) {

// restore to origin action.

realIntent.setAction(originAction);

}

handleStaticBroadcastAsUser(vuid, info, realIntent, result);

return true;

}

private void handleStaticBroadcastAsUser(int vuid, ActivityInfo info, Intent intent,

PendingResultData result) {

synchronized (this) {

ProcessRecord r = findProcessLocked(info.processName, vuid);

if (BROADCAST\_NOT\_STARTED\_PKG && r == null) {

r = startProcessIfNeedLocked(info.processName, getUserId(vuid), info.packageName);

}

if (r != null && r.appThread != null) {

performScheduleReceiver(r.client, vuid, info, intent,

result);

}

}

}

private void performScheduleReceiver(IVClient client, int vuid, ActivityInfo info, Intent intent,

PendingResultData result) {

ComponentName componentName = ComponentUtils.toComponentName(info);

BroadcastSystem.get().broadcastSent(vuid, info, result);

try {

client.scheduleReceiver(info.processName, componentName, intent, result);

} catch (Throwable e) {

if (result != null) {

result.finish();

}

}

}

* 跟到消息处理

public void handleMessage(Message msg) {

switch (msg.what) {

case NEW\_INTENT: {

handleNewIntent((NewIntentData) msg.obj);

}

break;

case RECEIVER: {

handleReceiver((ReceiverData) msg.obj);

}

}

}

* handleReceiver: 通过数据获取clientapp接收广播的类加载

private void handleReceiver(ReceiverData data) {

BroadcastReceiver.PendingResult result = data.resultData.build();

try {

// 依然是检测 Application 是否初始化，没有则初始化

if (!isBound()) {

bindApplication(data.component.getPackageName(), data.processName);

}

Context context = mInitialApplication.getBaseContext();

Context receiverContext = ContextImpl.getReceiverRestrictedContext.call(context);

String className = data.component.getClassName();

BroadcastReceiver receiver = (BroadcastReceiver) context.getClassLoader().loadClass(className).newInstance();

mirror.android.content.BroadcastReceiver.setPendingResult.call(receiver, result);

data.intent.setExtrasClassLoader(context.getClassLoader());

if (data.intent.getComponent() == null) {

data.intent.setComponent(data.component);

}

// 主动调用接收方法处理

receiver.onReceive(receiverContext, data.intent);

if (mirror.android.content.BroadcastReceiver.getPendingResult.call(receiver) != null) {

result.finish();

}

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException(

"Unable to start receiver " + data.component

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

}

// 远程通知 VAService 广播已送到

VActivityManager.get().broadcastFinish(data.resultData);

}

* com.lody.virtual.server.am.BroadcastSystem#broadcastFinish

void broadcastSent(int vuid, ActivityInfo receiverInfo, PendingResultData res) {

BroadcastRecord record = new BroadcastRecord(vuid, receiverInfo, res);

synchronized (mBroadcastRecords) {

mBroadcastRecords.put(res.mToken, record);

}

Message msg = new Message();

msg.obj = res.mToken;

// 超时机制

mTimeoutHandler.sendMessageDelayed(msg, BROADCAST\_TIME\_OUT);

}

private final class TimeoutHandler extends Handler {

TimeoutHandler(Looper looper) {

super(looper);

}

@Override

public void handleMessage(Message msg) {

IBinder token = (IBinder) msg.obj;

BroadcastRecord r = mBroadcastRecords.remove(token);

if (r != null) {

VLog.w(TAG, "Broadcast timeout, cancel to dispatch it.");

// 通知 PendingResult 结束，告诉发送方广播结束了

r.pendingResult.finish();

}

}

}

## VirtualAPP - Broadcast 发送处理

广播的发送，其实也是交给AMS来完成的，首先调用context.sendBroadcast将广播发给AMS的相应函数，AMS再和之前注册的Receiver进行匹配，匹配成功后，就发送给对应的进程。

* ContextImpl.java 通过 sendBroadcast 发送广播,实际调用AMS的broadcastIntent

@Override

public void sendBroadcast(Intent intent, String receiverPermission) {

...........................

try {

ActivityManagerNative.getDefault().broadcastIntent(

mMainThread.getApplicationThread(), intent, resolvedType, null,

Activity.RESULT\_OK, null, null, receiverPermissions, AppOpsManager.OP\_NONE,

null, false, false, getUserId());

}

........................

}

* 而VA 和startActivity等相同 hook了 broadcastIntent
* com.lody.virtual.client.hook.proxies.am.MethodProxies.BroadcastIntent
* handleIntent 拦截了创建卸载快捷方式的Intent

static class BroadcastIntent extends MethodProxy {

@Override

public String getMethodName() {

return "broadcastIntent";

}

@Override

public Object call(Object who, Method method, Object... args) throws Throwable {

Intent intent = (Intent) args[1];

String type = (String) args[2];

intent.setDataAndType(intent.getData(), type);

if (VirtualCore.get().getComponentDelegate() != null) {

VirtualCore.get().getComponentDelegate().onSendBroadcast(intent);

}

Intent newIntent = handleIntent(intent);

if (newIntent != null) {

args[1] = newIntent;

} else {

return 0;

}

if (args[7] instanceof String || args[7] instanceof String[]) {

// clear the permission

args[7] = null;

}

// 调用原始的方法

return method.invoke(who, args);

}

// 拦截了创建卸载快捷方式的Intent

private Intent handleIntent(final Intent intent) {

final String action = intent.getAction();

if ("android.intent.action.CREATE\_SHORTCUT".equals(action)

|| "com.android.launcher.action.INSTALL\_SHORTCUT".equals(action)) {

return VASettings.ENABLE\_INNER\_SHORTCUT ? handleInstallShortcutIntent(intent) : null;

} else if ("com.android.launcher.action.UNINSTALL\_SHORTCUT".equals(action)) {

handleUninstallShortcutIntent(intent);

} else if (BadgerManager.handleBadger(intent)) {

return null;

} else {

// 利用subIntent打包了真正的Intent 和activity类似

return ComponentUtils.redirectBroadcastIntent(intent, VUserHandle.myUserId());

}

return intent;

}

private Intent handleInstallShortcutIntent(Intent intent) {

Intent shortcut = intent.getParcelableExtra(Intent.EXTRA\_SHORTCUT\_INTENT);

if (shortcut != null) {

ComponentName component = shortcut.resolveActivity(VirtualCore.getPM());

if (component != null) {

String pkg = component.getPackageName();

Intent newShortcutIntent = new Intent();

newShortcutIntent.setClassName(getHostPkg(), Constants.SHORTCUT\_PROXY\_ACTIVITY\_NAME);

newShortcutIntent.addCategory(Intent.CATEGORY\_DEFAULT);

newShortcutIntent.putExtra("\_VA\_|\_intent\_", shortcut);

newShortcutIntent.putExtra("\_VA\_|\_uri\_", shortcut.toUri(0));

newShortcutIntent.putExtra("\_VA\_|\_user\_id\_", VUserHandle.myUserId());

intent.removeExtra(Intent.EXTRA\_SHORTCUT\_INTENT);

intent.putExtra(Intent.EXTRA\_SHORTCUT\_INTENT, newShortcutIntent);

Intent.ShortcutIconResource icon = intent.getParcelableExtra(Intent.EXTRA\_SHORTCUT\_ICON\_RESOURCE);

if (icon != null && !TextUtils.equals(icon.packageName, getHostPkg())) {

try {

Resources resources = VirtualCore.get().getResources(pkg);

int resId = resources.getIdentifier(icon.resourceName, "drawable", pkg);

if (resId > 0) {

//noinspection deprecation

Drawable iconDrawable = resources.getDrawable(resId);

Bitmap newIcon = BitmapUtils.drawableToBitmap(iconDrawable);

if (newIcon != null) {

intent.removeExtra(Intent.EXTRA\_SHORTCUT\_ICON\_RESOURCE);

intent.putExtra(Intent.EXTRA\_SHORTCUT\_ICON, newIcon);

}

}

} catch (Throwable e) {

e.printStackTrace();

}

}

}

}

return intent;

}

private void handleUninstallShortcutIntent(Intent intent) {

Intent shortcut = intent.getParcelableExtra(Intent.EXTRA\_SHORTCUT\_INTENT);

if (shortcut != null) {

ComponentName componentName = shortcut.resolveActivity(getPM());

if (componentName != null) {

Intent newShortcutIntent = new Intent();

newShortcutIntent.putExtra("\_VA\_|\_uri\_", shortcut.toUri(0));

newShortcutIntent.setClassName(getHostPkg(), Constants.SHORTCUT\_PROXY\_ACTIVITY\_NAME);

newShortcutIntent.removeExtra(Intent.EXTRA\_SHORTCUT\_INTENT);

intent.putExtra(Intent.EXTRA\_SHORTCUT\_INTENT, newShortcutIntent);

}

}

}

@Override

public boolean isEnable() {

return isAppProcess();

}

}

# VirtualAPP - ContentProvider

相比较于StubActivity的占位作用，StubContentProvider却并不是为了占位，其作用是为了调用时可以带起p进程，而在进程拉起后，会进行application的初始化，bindApplicationNoCheck方法中会真正对Client App中的ContentProvider进行安装注册。

当进程A调用进程B（或应用B）中的ContentProvider时，会hook进程A中的getContentProvider方法，判断目标Provider所在进程B是否存在，如果不存在则拉起，这样进程B的Provider安装完毕，返回进程A所需要的目标Provider句柄，完成调用。

* 在和activity一样也有很对预定义好的StubContentProvider
* 不过和activity不同的是 StubContentProvider 的真正目的是为了让 AMS 通过 system\_process 带起 “p(n)”进程，然后 VAMS 用过远程调用 StubProvider.call() 回插件 IClient 的 IBinder 句柄给 VAMS 持有。这样 VAMS 就可以远程调用插件进程 “p(n)”中的方法了。

public class StubContentProvider extends ContentProvider {

@Override

public boolean onCreate() {

return true;

}

@Override

public Bundle call(String method, String arg, Bundle extras) {

if ("\_VA\_|\_init\_process\_".equals(method)) {

return initProcess(extras);

}

return null;

}

private Bundle initProcess(Bundle extras) {

ConditionVariable lock = VirtualCore.get().getInitLock();

if (lock != null) {

lock.block();

}

IBinder token = BundleCompat.getBinder(extras,"\_VA\_|\_binder\_");

int vuid = extras.getInt("\_VA\_|\_vuid\_");

VClientImpl client = VClientImpl.get();

client.initProcess(token, vuid);

Bundle res = new Bundle();

BundleCompat.putBinder(res, "\_VA\_|\_client\_", client.asBinder());

res.putInt("\_VA\_|\_pid\_", Process.myPid());

return res;

}

@Override

public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {

return null;

}

@Override

public String getType(Uri uri) {

return null;

}

@Override

public Uri insert(Uri uri, ContentValues values) {

return null;

}

@Override

public int delete(Uri uri, String selection, String[] selectionArgs) {

return 0;

}

@Override

public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {

return 0;

}

public static class C0 extends StubContentProvider {

}

public static class C1 extends StubContentProvider {

}

public static class C2 extends StubContentProvider {

}

public static class C3 extends StubContentProvider {

}

public static class C4 extends StubContentProvider {

}

public static class C5 extends StubContentProvider {

* 同样VA hook了 getContentProvider

static class GetContentProvider extends MethodProxy {

@Override

public String getMethodName() {

return "getContentProvider";

}

@Override

public Object call(Object who, Method method, Object... args) throws Throwable {

int nameIdx = getProviderNameIndex();

String name = (String) args[nameIdx];

int userId = VUserHandle.myUserId();

// 获取解析好的目标ProviderInfo

ProviderInfo info = VPackageManager.get().resolveContentProvider(name, 0, userId);

if (info != null && info.enabled && isAppPkg(info.packageName)) {

// 如果为空 启动Provider 所在的进程

int targetVPid = VActivityManager.get().initProcess(info.packageName, info.processName, userId);

if (targetVPid == -1) {

return null;

}

args[nameIdx] = VASettings.getStubAuthority(targetVPid);

Object holder = method.invoke(who, args);

if (holder == null) {

return null;

}

if (BuildCompat.isOreo()) {

IInterface provider = ContentProviderHolderOreo.provider.get(holder);

if (provider != null) {

// 远程调用了 VAMS 的 acquireProviderClient

provider = VActivityManager.get().acquireProviderClient(userId, info);

}

ContentProviderHolderOreo.provider.set(holder, provider);

ContentProviderHolderOreo.info.set(holder, info);

} else {

IInterface provider = IActivityManager.ContentProviderHolder.provider.get(holder);

if (provider != null) {

//

provider = VActivityManager.get().acquireProviderClient(userId, info);

}

IActivityManager.ContentProviderHolder.provider.set(holder, provider);

IActivityManager.ContentProviderHolder.info.set(holder, info);

}

return holder;

* acquireProviderClient :

public IBinder acquireProviderClient(int userId, ProviderInfo info) {

ProcessRecord callerApp;

synchronized (mPidsSelfLocked) {

callerApp = findProcessLocked(getCallingPid());

}

if (callerApp == null) {

throw new SecurityException("Who are you?");

}

String processName = info.processName;

ProcessRecord r;

synchronized (this) {

r = startProcessIfNeedLocked(processName, userId, info.packageName);

}

if (r != null && r.client.asBinder().isBinderAlive()) {

try {

// 回到目标 Provider 的 Client App 进程中

return r.client.acquireProviderClient(info);

} catch (RemoteException e) {

e.printStackTrace();

}

}

return null;

}

* com.lody.virtual.client.VClientImpl#acquireProviderClient

public IBinder acquireProviderClient(ProviderInfo info) {

if (mTempLock != null) {

mTempLock.block();

}

if (!isBound()) {

// 这里检查 Application 是否启动，注册 Provider 的逻辑也在里面

VClientImpl.get().bindApplication(info.packageName, info.processName);

}

// 准备 ContentProviderClient

IInterface provider = null;

String[] authorities = info.authority.split(";");

String authority = authorities.length == 0 ? info.authority : authorities[0];

ContentResolver resolver = VirtualCore.get().getContext().getContentResolver();

ContentProviderClient client = null;

try {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.JELLY\_BEAN) {

client = resolver.acquireUnstableContentProviderClient(authority);

} else {

client = resolver.acquireContentProviderClient(authority);

}

} catch (Throwable e) {

e.printStackTrace();

}

if (client != null) {

// 通过反射获取provider

provider = mirror.android.content.ContentProviderClient.mContentProvider.get(client);

client.release();

}

return provider != null ? provider.asBinder() : null;

}

## VirtualAPP - provider install

VClientImpl.get().bindApplication

installContentProviders(mInitialApplication, data.providers);

private void installContentProviders(Context app, List<ProviderInfo> providers) {

long origId = Binder.clearCallingIdentity();

Object mainThread = VirtualCore.mainThread();

try {

for (ProviderInfo cpi : providers) {

try {

ActivityThread.installProvider(mainThread, app, cpi, null);

} catch (Throwable e) {

e.printStackTrace();

}

}

} finally {

Binder.restoreCallingIdentity(origId);

}

}

# VirtualAPP - 总结

1. 在va启动时就会对服务进行代理hook,对native做io文件重定位.
2. 安装app实际并没有安装,会分析app的manifest resources等保存备用.
3. VA app 和framework之间做了一层代理,劫持了app所访问的系统服务主要是AMS
4. va会提前注册一些activity用于支持clientapp启动界面,hook startActivity解析activity信息查询可用StubActivity 新建对应目标intent封装targetintent,最后调用经过代理的startActivity.
5. 对于service的处理相对activity来说,service的生命周期比较简单,直接利用scheduleCreateService创建,当然这些也是在VAMS中进行.
6. 在处理广播的时候对于静态注册的广播是无法直接被系统AMS感知的,根本没安装,由VA启动进程的时候扫描遍历所有Receiver,通过新建StaticBroadcastReceiver代理来接收每个IntentFilter。然后hook ClientApp中的发送广播的方法，这里其实也做了一个intent包裹，和Activity逻辑类似，当代理广播StaticBroadcastReceiver接收到包装后的intent信息时，解包出真正的intent，回调到ClientApp空间，实例化目标 Receiver（如果目标进程不存在则拉起进程），最终手动进行对其OnCreate和finish的调用。
7. 关注分析过程和熟悉架构。