看雪

SDC·2019

2019 安全开发者峰会

Security
Development
Conference

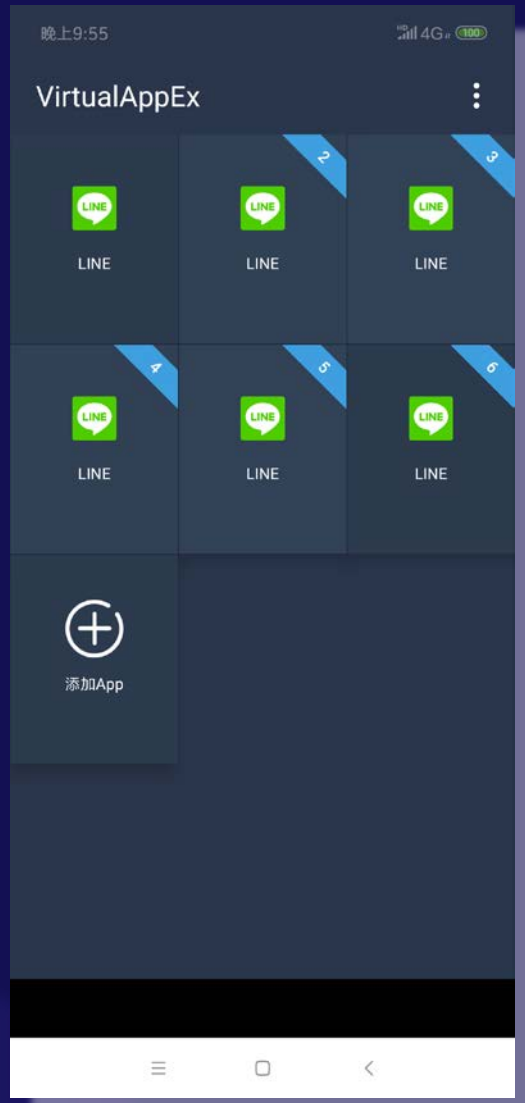
Android容器和虚拟化

邓维佳

声明：演示内容使用了部分VA代码，仅在研究中使用

目录

VA简介



容器作用

无限多开

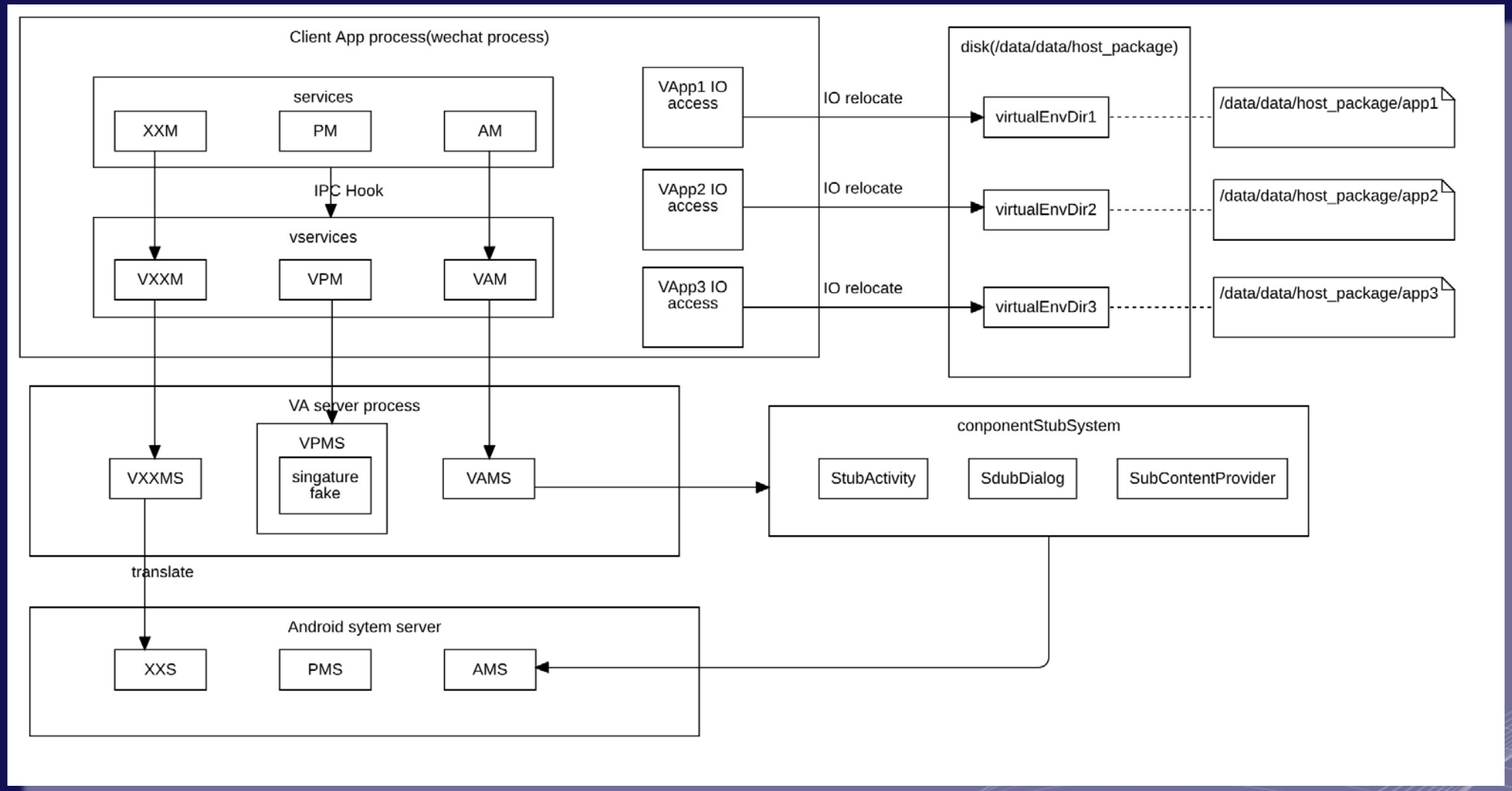
VA简介


```
@Override
public void beforeActivityDestroy(Activity activity) {

}

@Override
public void afterActivityCreate(Activity activity) {
    new AlertDialog.Builder(activity)
            .setTitle("app注入测试")
            .setMessage("当前activity: "
                    + activity.getClass().getName()
                    + "\n当前进程: "
                    + activity.getPackageName()
            )
            .setNeutralButton( text: "ok",
                    (dialog, which) -> {

                    })
            .create().show();
}

@Override
public void afterActivityResume(Activity activity) {

}
```

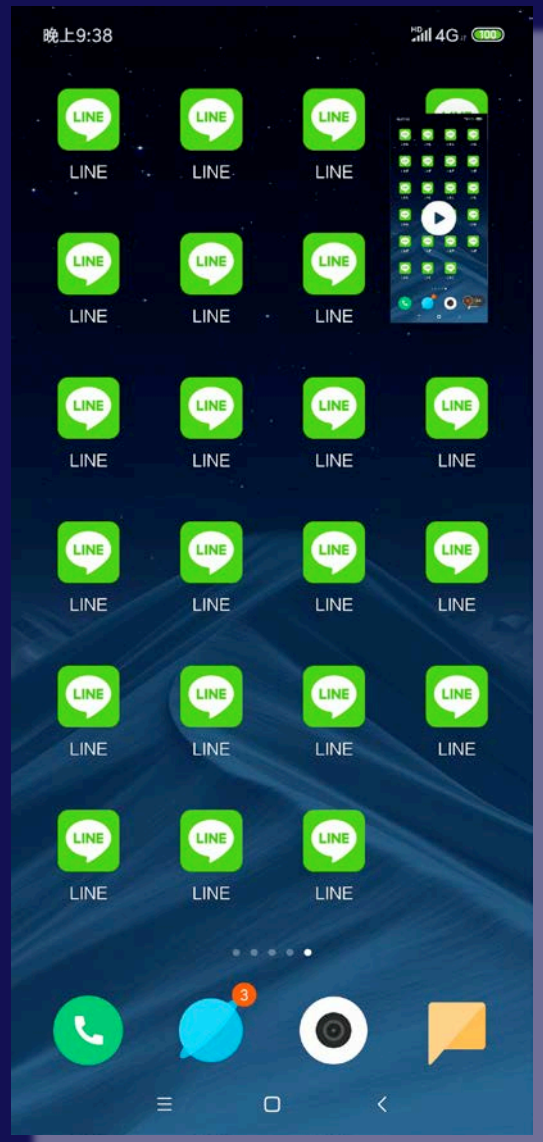
容器作用

注入控制

VA简介


Client App process(wechat process)
services
XXM
PM
AM
IPC Hook
vservices
VXXM
VPM
VAM
VApp1 IO access
VApp2 IO access
VApp3 IO access
IO relocate
IO relocate
IO relocate
disk(/data/data/host_package)
virtualEnvDir1
virtualEnvDir2
virtualEnvDir3
/data/data/host_package/app1
/data/data/host_package/app2
/data/data/host_package/app3
VA server process
VPMS
singature fake
VXXMS
VAMS
conponentStubSystem
StubActivity
SdubDialog
SubContentProvider
translate
Android sytem server
XXS
PMS
AMS

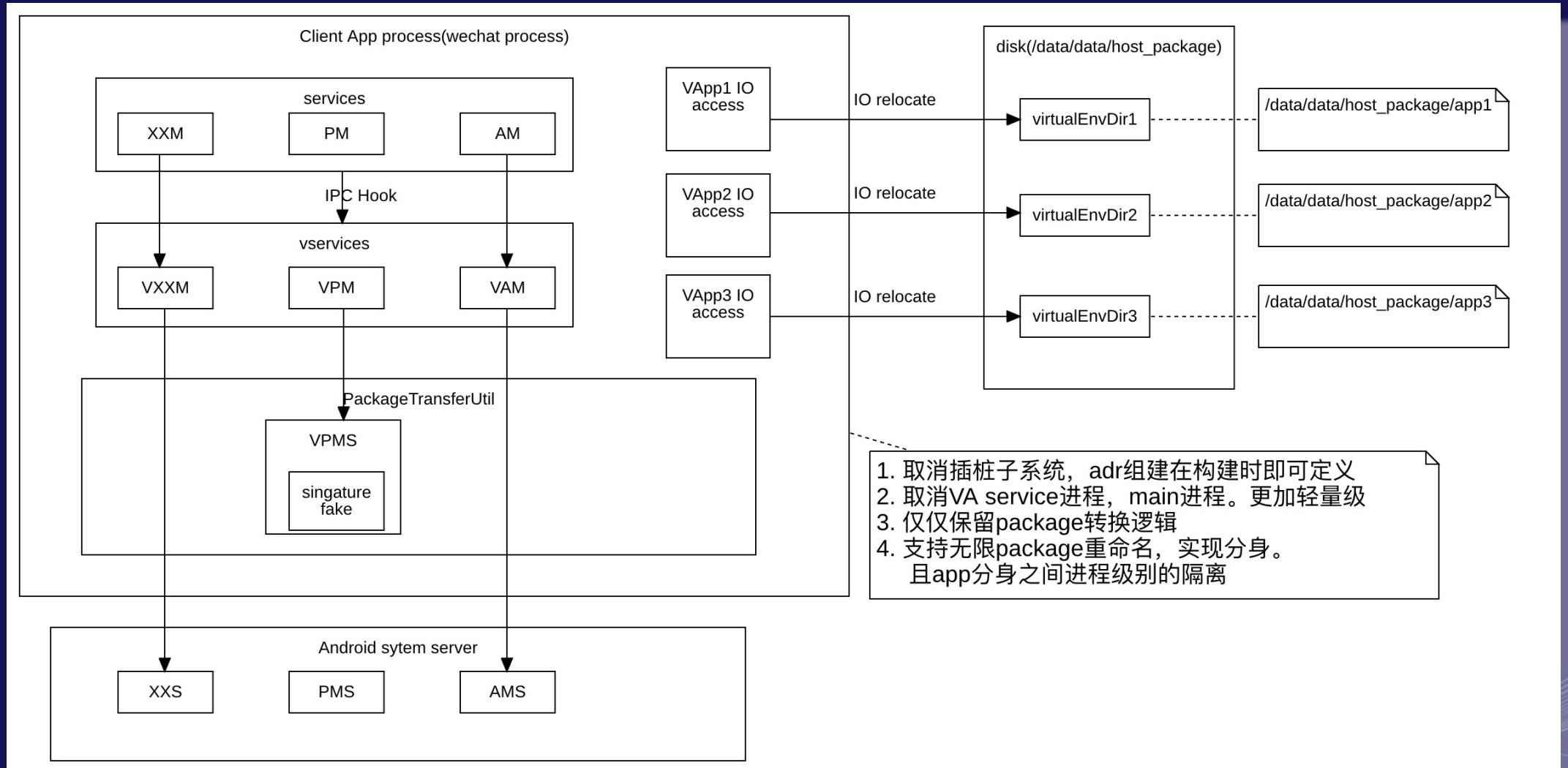
VA变种-基于VA打包



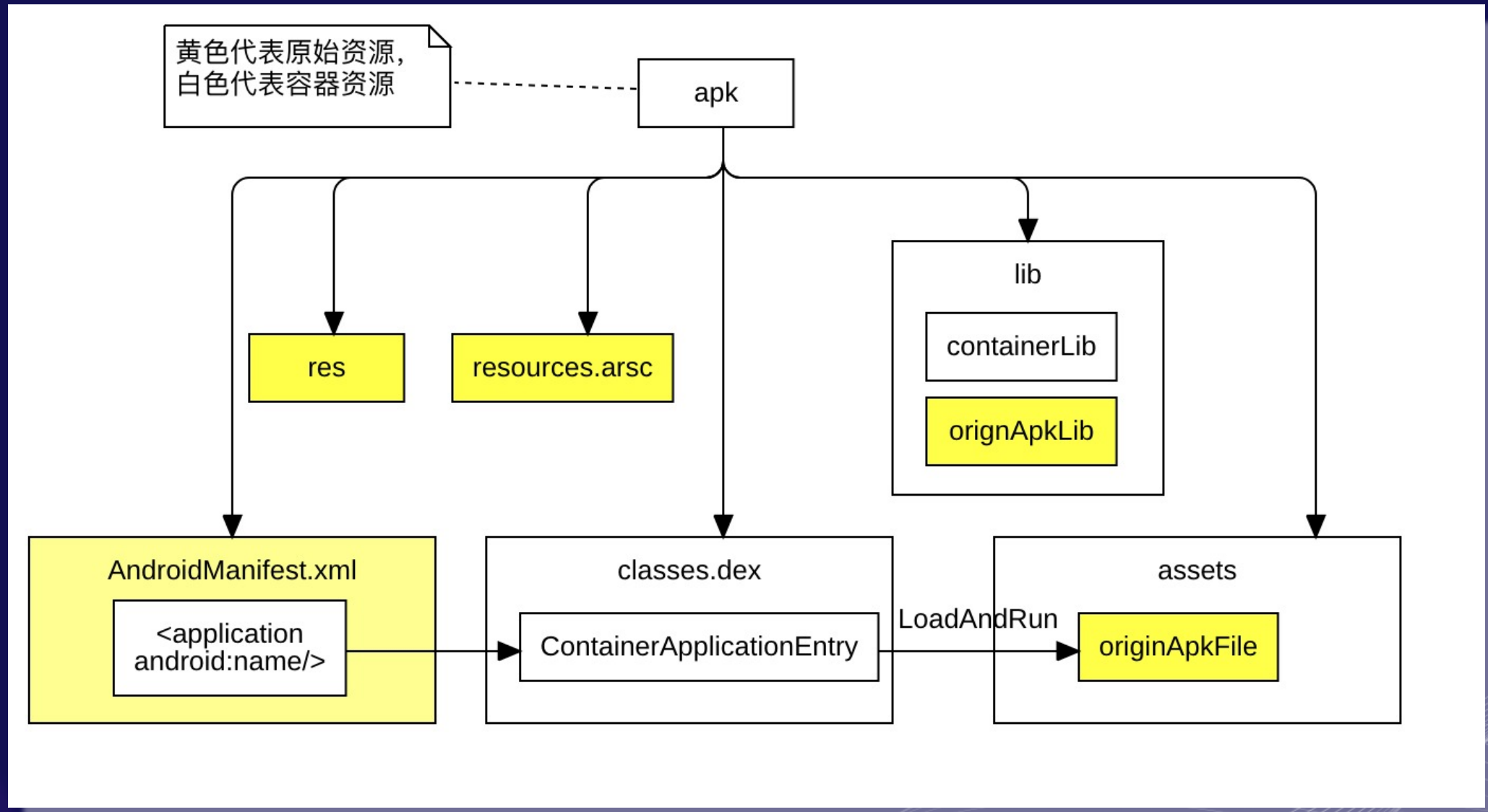
Package Rename

- 容器实例级别无限分身
- 更好的兼容性
- 更少的分身特征
- 适合向普通C端用户分包
- 病毒特征隐藏
- Hook控制

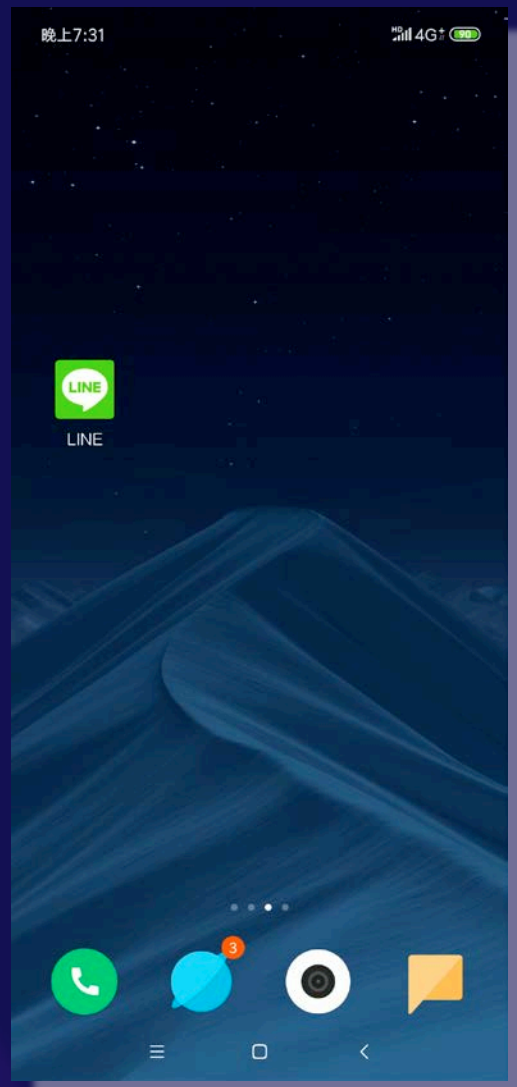
VA变种-基于VA打包


Client App process(wechat process)
services
XXM
PM
AM
IPC Hook
vservices
VXXM
VPM
VAM
PackageTransferUtil
VPMS
singature
fake
Android sytem server
XXS
PMS
AMS
VApp1 IO access
VApp2 IO access
VApp3 IO access
IO relocate
IO relocate
IO relocate
disk(/data/data/host_package)
virtualEnvDir1
virtualEnvDir2
virtualEnvDir3
/data/data/host_package/app1
/data/data/host_package/app2
/data/data/host_package/app3
1. 取消插桩子系统，adr组建在构建时即可定义
2. 取消VA service进程，main进程。更加轻量级
3. 仅仅保留package转换逻辑
4. 支持无限package重命名，实现分身。
且app分身之间进程级别的隔离

套壳容器

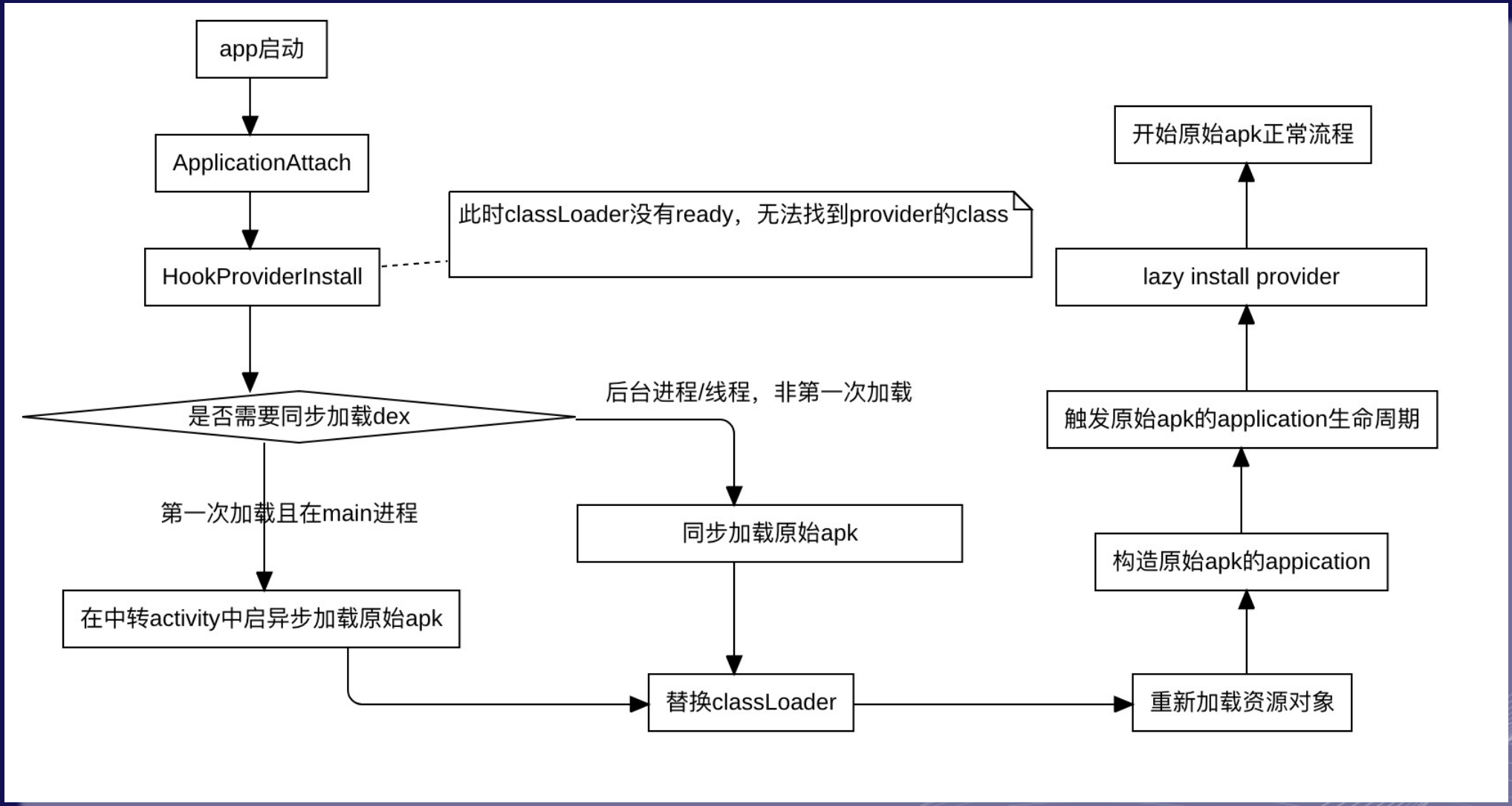

黄色代表原始资源，
白色代表容器资源
apk
res
resources.arsc
lib
containerLib
orignApkLib
AndroidManifest.xml
<application
android:name/>
classes.dex
ContainerApplicationEntry
LoadAndRun
assets
originApkFile

套壳容器

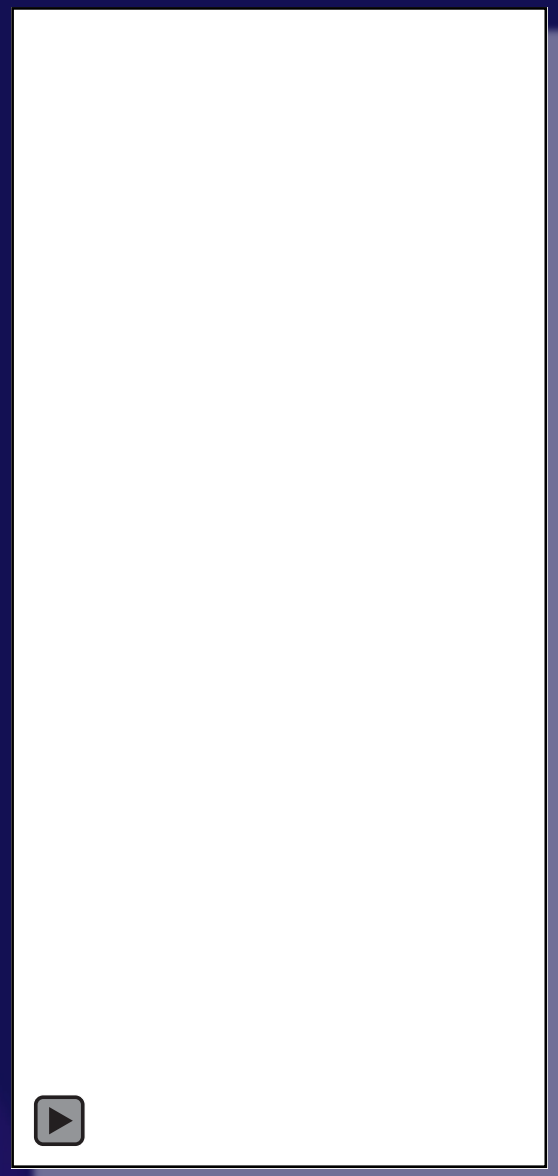


- 介于VA和重打包之间的一种方案
- Apk体积小，无需存在双份dex文件
- 没有dex修改痕迹
- 不需要对抗资源混淆
- Dex2Oat延时问题
- contentProvider无法自启动

套壳容器

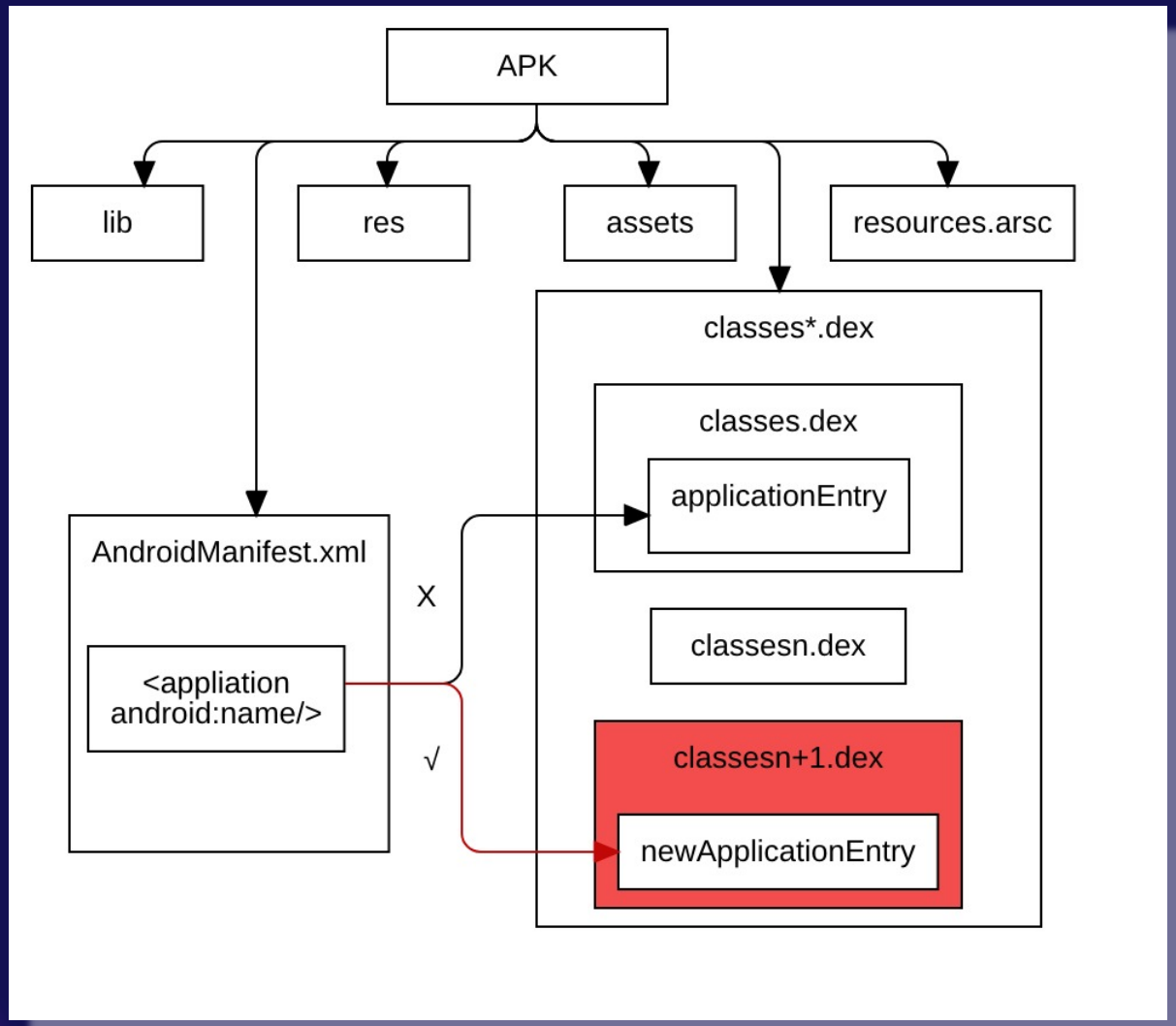

app启动
ApplicationAttach
HookProviderInstall
此时classLoader没有ready，无法找到provider的class
是否需要同步加载dex
后台进程/线程，非第一次加载
第一次加载且在main进程
同步加载原始apk
在中转activity中启异步加载原始apk
替换classLoader
重新加载资源对象
构造原始apk的appication
触发原始apk的application生命周期
lazy install provider
开始原始apk正常流程

重打包 append MultiDex



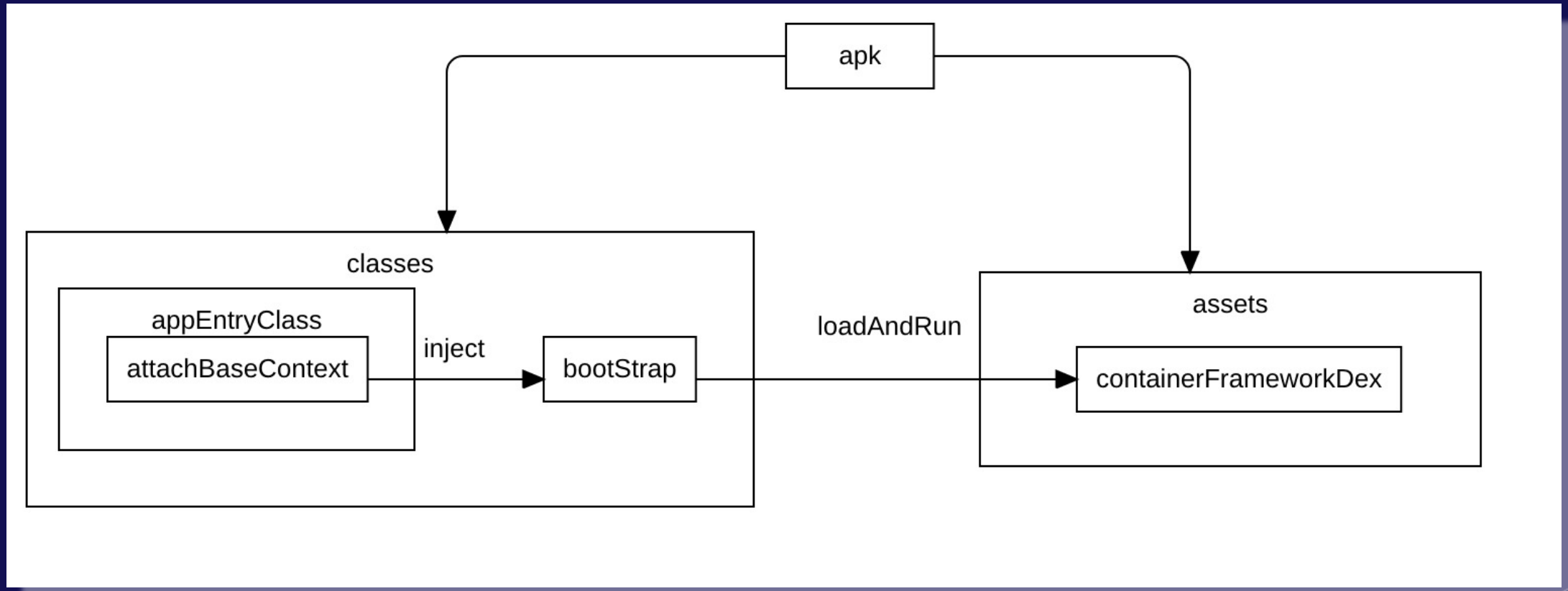
```
public static void applicationOnCreateWithMultiMode() {
    Instrumentation instrumentation = (Instrumentation) XposedHelpers.getObjectField(
            mainThread, fieldName: "mInstrumentation");
    instrumentation.callApplicationOnCreate(realApplication);
    XposedHelpers.findAndHookMethod(Activity.class, methodName: "onResume", new XC_Met
        @Override
        protected void afterHookedMethod(MethodHookParam param) throws Throwable {
            Log.i(Constants.TAG, msg: "Activity onResume", new Throwable());
            new AlertDialog.Builder((Context) param.thisObject)
                    .setTitle("injectTest")
                    .setMessage("injected by virjar")
                    .setNeutralButton( text: "ok", (dialog, which) -> {
                    }).show();
        }
    });
}

//shell engine entry
public static void applicationAttachWithShellMode(Context context) throws Exception {
    init( applicationContext: null, context);
    // prevent installContentProviders because of classloader not ready
```

重打包 append MultiDex

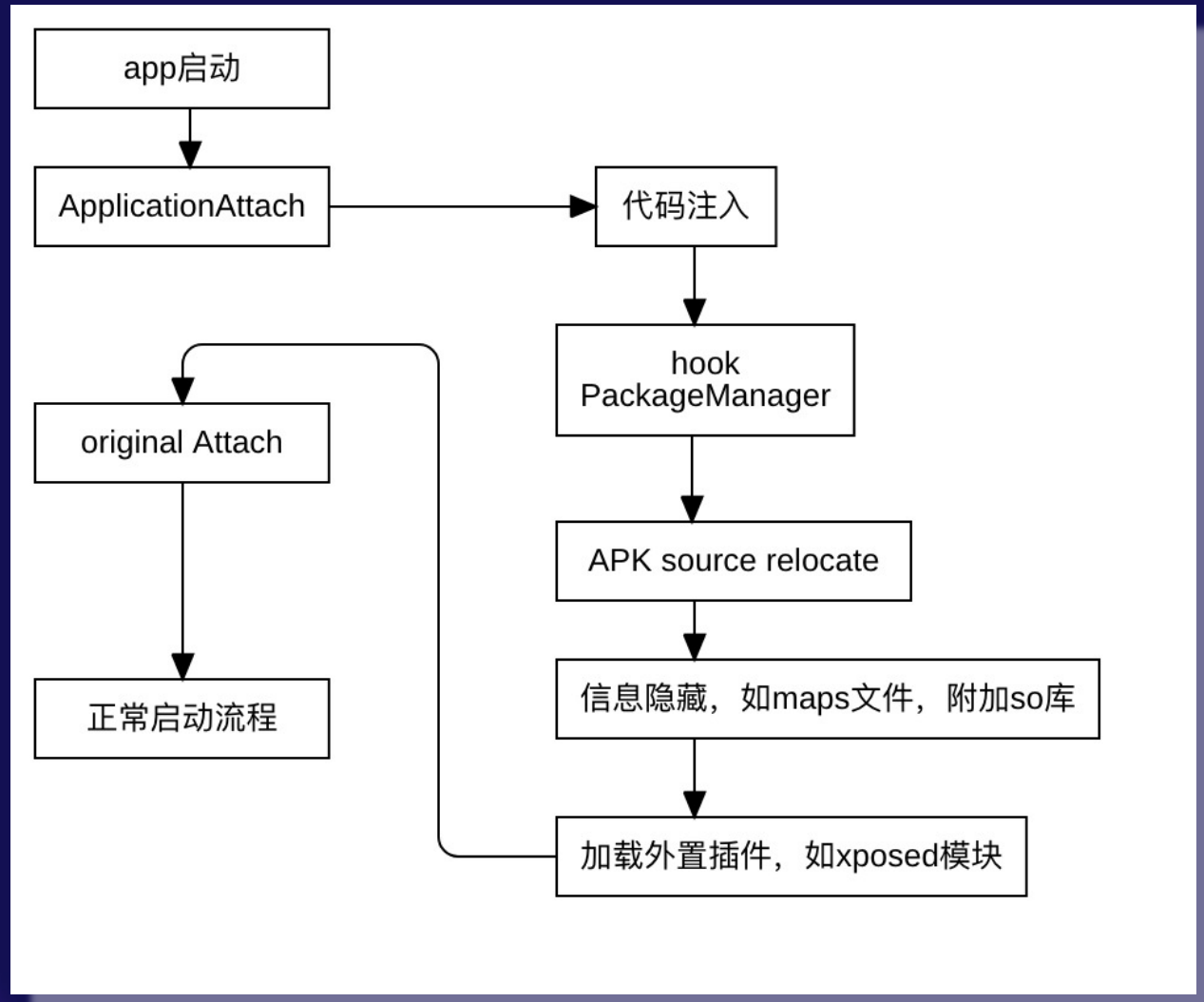


- 无dex修改痕迹
- 需要绕过资源混淆对抗
- Apk两倍膨胀

Dex入口修改


apk
classes
appEntryClass
attachBaseContext
inject
bootStrap
loadAndRun
assets
containerFrameworkDex

Dex入口修改



- 支持加壳apk
- 无代理痕迹
- Apk两倍膨胀
- Maps中dex特征将发生变化
- 65535限制

常见痕迹隐藏和对抗方案

1. 动态代理：Package Manager fake
2. IO重定向：文件内容签名检测
3. ARTHook：所有java函数拦截
4. Maps:maps重定向
5. OAT文件头部：OAT伪造
6. AXmlEditor:基于二进制格式层面修改AndroidManifest.xml,对抗资源混淆
7. 单指令注入：考虑单dex65535限制

1. 代理痕迹：$Poxyxxx
2. classLoader：资源路径&dexList&PathClassLoader&classLoader父子关系
3. Class.getDex
4. Maps OAT内容

常见痕迹隐藏和对抗方案

	优点	缺点	案例
Rom定制	App感知不到任何痕迹	拿不到rom源码 需要解BL锁 无法作用于流行机型	云手机 Google手机 模拟器
Root注入	可以侵入system-server App侧痕迹少	多需要修改系统，难度较大，root痕迹和注入痕迹隐藏较为麻烦	Xposed/太极 **Magisk**
套壳注入	可移植性强 无系统全局修改痕迹 可以实现分身	注入逻辑再apk侧，运行上下文有痕迹(如classloader、maps文件、进程文件、动态代理痕迹)。需要对android框架层有比较深入了解	**VirtualApp**
重打包	兼容性最好（非对抗兼容性） 普通用户无感知（输出APK和原始APK参数一致）	重打包痕迹对抗难度较大 需hack底层函数，和各种加固方案冲突 跨进程API检查失败(外部签名无法影响)	**太极阴** **Xpatch**