Android 逆向

1. APK文件解析

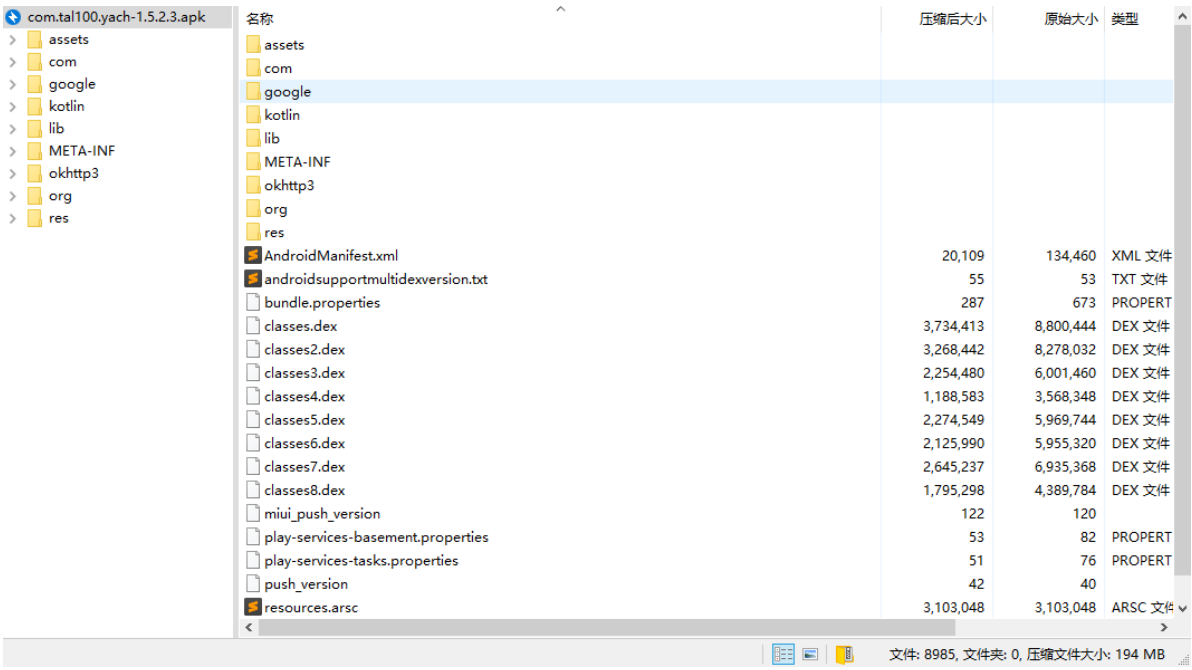
> APK（全称：Android application package，Android应用程序包）是Android操作系统使用的应用程序包文件格式，用于分发和安装移动端应用。

1.1 基本组成

```
# 资源目录
./assets

# 共享库目录(so库)
./lib

# 存放工程属性文件，签名文件
./META-INF

# 资源目录(编译后)
./res

# Android工程配置文件,记录包名、版本号、四大组件（活动、服务、广播接收者、内容提供者）、申请权限
AndroidMainfest.xml

# Android平台(Dalvik虚拟机)的可执行文件
classes*.dex

# res资源索引表文件
resources.arsc
```

- assets 和 res 目录的区别

> res目录下的文件会映射到R.java文件，可以直接使用资源id访问存储在res目录下的资源文件，例如，代码中的`InputStream is = getResources().openRawResource(R.id.filename);`等。
> 访问assets目录下文件，需要用到AssetManager类，通过open/openfile方法进行访问。



2. DEX文件解析

> DEX是 **Dalvik VM executes** 的简称，即Android Dalvik可执行程序，程序并非 **Java ME** 的字节码，而是 **Dalvik** 字节码。**Dalvik VM**是基于寄存器的，而JVM是基于栈，因此**Dalvik VM**比JVM速度快，占用空间更少。

> 在 Java 程序中，Java 类被编译成一个或多个 .class 文件，并打包成 jar 文件，而后 JVM 会通过相应的 .class 文件和 jar 文件获取对相应的字节码。
>
> 执行顺序为：.java 文件 → .class 文件 → .jar 文件
>
> DVM 会用 dx 工具将所有的 .class 文件转换为一个 .dex 文件，然后 DVM 会从该 .dex 文件读取指令和数据。
>
> 执行顺序为：.java 文件 → .class 文件 → .dex 文件。

- Class文件和DEX文件结构

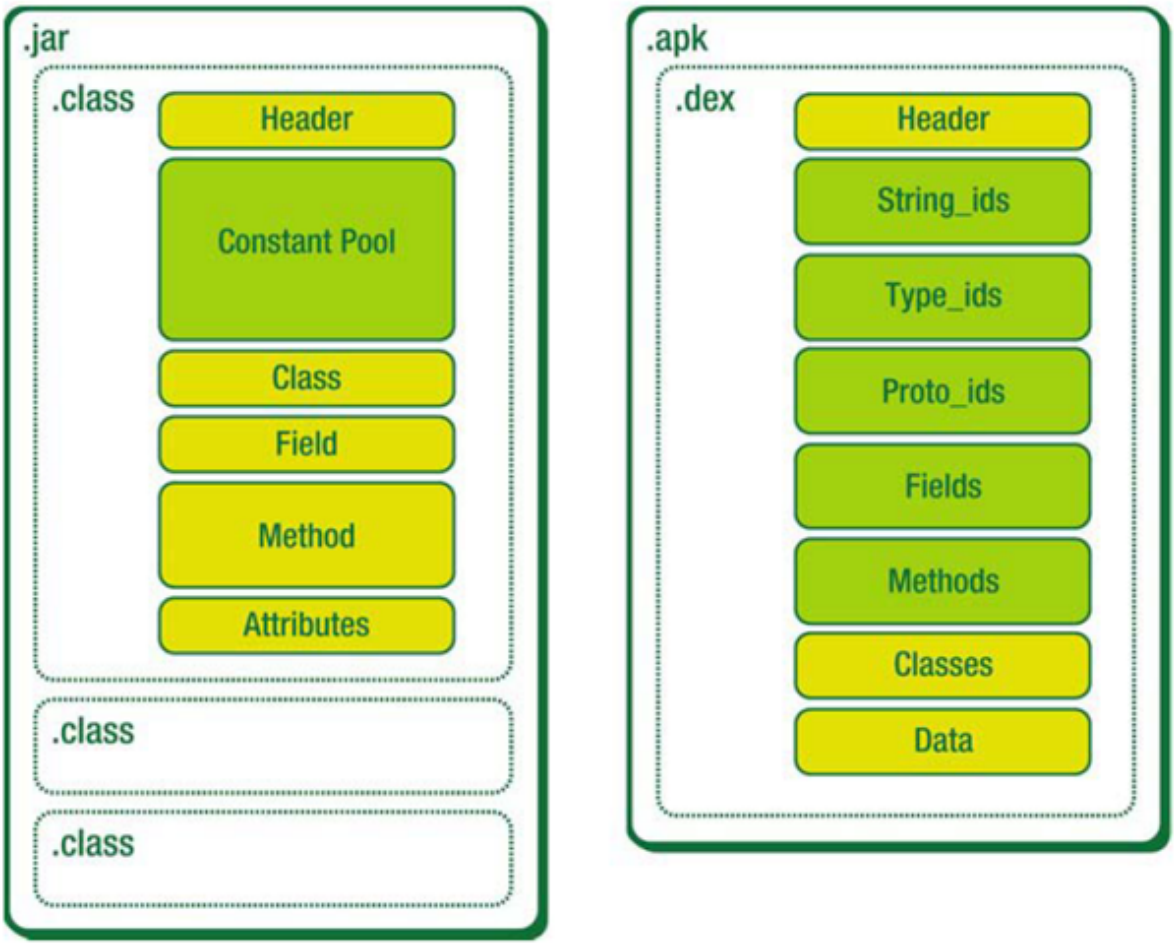


Figure 3-2. *Class file vs DEX file*

- DEX文件结构详情

名称	描述
header	dex文件头部，记录整个dex文件的相关属性
string_ids	字符串数据索引，记录了每个字符串在数据区的偏移量
type_ids	类型数据索引，记录了每个类型的字符串索引
proto_ids	原型数据索引，记录了方法声明的字符串，返回类型字符串，参数列表
field_ids	字段数据索引，记录了所属类，类型以及方法名
method_ids	类方法索引，记录方法所属类名，方法声明以及方法名等信息
class_defs	类定义数据索引，记录指定类各类信息，包括接口，父类，类数据偏移量
data	数据区，保存了各个类的真实数据
link_data	连接数据区

- DEX文件定义

```
http://androidxref.com/9.0.0_r3/xref/dalvik/libdex/DexFile.h
```

2.1 header

> 主要记录dex文件基本信息，以及所有字段大致分布。

- header定义

```
struct DexHeader {
    u1  magic[8];           /* includes version number */
```

```
    u4  checksum;           /* adler32 checksum */
    u1  signature[kSHA1DigestLen]; /* SHA-1 hash */
    u4  fileSize;           /* length of entire file */
    u4  headerSize;         /* offset to start of next section */
    u4  endianTag;
    u4  linkSize;
    u4  linkOff;
    u4  mapOff;
    u4  stringIdsSize;
    u4  stringIdsOff;
    u4  typeIdsSize;
    u4  typeIdsOff;
    u4  protoIdsSize;
    u4  protoIdsOff;
    u4  fieldIdsSize;
    u4  fieldIdsOff;
    u4  methodIdsSize;
    u4  methodIdsOff;
    u4  classDefsSize;
    u4  classDefsOff;
    u4  dataSize;
    u4  dataOff;
};
```

- header详情

名称	描述	地址	长度
magic	魔术字段，包含了dex文件标识符以及版本	0x00	8个字节
checksum	dex文件校验码	0x08	4个字节
signature	dex sha-1签名	0x0c	20个字节
file_size	dex文件大小	0x20	4个字节
header_size	dex文件头大小	0x24	4个字节
endian_tag	dex文件判断字节序是否交换，一般情况下为0x123456	0x28	4个字节
link_size	dex文件链接段大小，为0则表示为静态链接	0x2c	4个字节
link_off	dex文件链接段偏移位置	0x30	4个字节
map_off	dex文件中map数据段偏移位置	0x34	4个字节
string_ids_size	dex文件包含的字符串数量	0x38	4个字节
string_ids_off	dex文件字符串开始偏移位置	0x3c	4个字节
type_ids_size	dex文件类数量	0x40	4个字节
type_ids_off	dex文件类偏移位置	0x44	4个字节
photo_ids_size	dex文件中方法原型数量	0x48	4个字节
photo_ids_off	dex文件中方法原型偏移位置	0x4c	4个字节
field_ids_size	dex文件中字段数量	0x50	4个字节
field_ids_off	dex文件中字段偏移位置	0x54	4个字节
method_ids_size	dex文件中方法数量	0x58	4个字节
method_ids_off	dex文件中方法偏移位置	0x5c	4个字节
class_defs_size	dex文件中类定义数量	0x60	4个字节
class_defs_off	dex文件中类定义偏移位置	0x64	4个字节
data_size	dex数据段大小	0x68	4个字节
data_off	dex数据段偏移位置	0x6c	4个字节

2.2 String_ids

> 字符串数据索引，记录了每个字符串在数据区的偏移量

string_id_item			
名称	描述	格式	长度
string_data_off	从文件开头到此项的字符串数据的偏移量。	uint	4个字节

string_data_item			
名称	描述	格式	长度
utf16_size	字符串的大小	uleb128	
data	字符串数据，一系列 MUTF-8 代码单元。	ubyte[]	

- 字节码类型描述符

语法	含义
V	void；仅对返回类型有效
Z	boolean
B	byte
S	short
C	char
I	int
J	long
F	float
D	double
Lfully/qualified/Name;	类 *fully.qualified.Name*
[descriptor	*descriptor* 的数组，可递归地用于“数组的数组”，但维数不能超过 255。

e.g.

```
1. hello()V == void hello()
2. hello(III)Z == boolean hello(int, int, int)
3. hello(Z[I][ILjava/lang/String;J]Ljava/lang/String) == String hello (boolean, int[], int[], String, long)
```

2.3 Type_ids

> 类型数据索引，记录了每个类型的字符串索引

type_id_item			
名称	描述	格式	长度
descriptor_idx	类型字符串的 string_ids 列表中的索引。	uint	4个字节

2.4 Proto_ids

> 原型数据索引，记录了方法声明的字符串，返回类型字符串，参数列表

proto_id_item			
名称	描述	格式	长度
shorty_idx	字符串的 string_ids 列表中的索引。	uint	4个字节
return_type_idx	返回类型的 type_ids 列表中的索引。	uint	4个字节
parameters_off	从文件开头到此原型的参数类型列表的偏移量。	uint	4个字节

2.5 Fields

> 字段数据索引，记录了所属类，类型以及方法名

field_id_item			
名称	描述	格式	长度
class_idx	该字段的定义符的 type_ids 列表中的索引。	ushort	4个字节
type_idx	该字段的类型的 type_ids 列表中的索引。	ushort	4个字节
name_idx	该字段的名称的 string_ids 列表中的索引。	uint	4个字节

2.6 Methods

> 类方法索引，记录方法所属类名，方法声明以及方法名等信息

method_id_item			
名称	描述	格式	长度
class_idx	该方法的定义符的 type_ids 列表中的索引。	ushort	4个字节
proto_idx	该方法的原型的 proto_ids 列表中的索引。	ushort	4个字节
name_idx	该方法的名称的 string_ids 列表中的索引。	uint	4个字节

2.7 Class_defs

> 类定义数据索引，记录指定类各类信息，包括接口，父类，类数据偏移量

class_def_item			
名称	描述	格式	长度
class_idx	该类的 type_ids 列表中的索引。	uint	4个字节
access_flags	类的访问标记（public、final 等）。	uint	4个字节
superclass_idx	父类的 type_ids 列表中的索引。	uint	4个字节
interfaces_off	从文件开头到接口列表的偏移量	uint	4个字节
source_file_idx	文件名称的 string_ids 列表中的索引	uint	4个字节
annotations_off	从文件开头到此类的注释结构的偏移量	uint	4个字节
class_data_off	从文件开头到此项的关联类数据的偏移量	uint	4个字节
static_values_off	从文件开头到 static 字段初始值列表的偏移量	uint	4个字节

2.8 data

> data 数据区，保存了各个类的真实数据
>
> link_data 连接数据区(静态解析DEX时，一般都为0。)

3.Frida

3.1 简介

`Frida` 是一款基于 `Python + JavaScript` 的 `Hook` 与调试框架。

`Firda` 支持 `Java` 层到 `Native` 层的 `Hook` ，是动态的插桩工具，可以插入代码到原生 `App` 的内存空间中，动态的去监视和修改行为，但缺点是动态调试函数会遇到各种反调试、应用崩溃的情况。

3.2 用途

- 访问进程的内存
- 应用程序运行时覆盖功能
- 从导入的类调用函数
- 动态`Hook`跟踪、拦截函数等

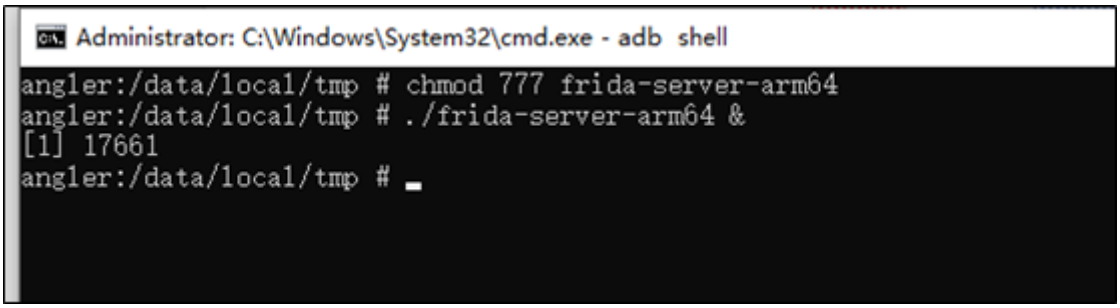
3.3 环境搭建

1. 安装python3并配置好环境变量，python安装包官方下载地址：`https://www.python.org/downloads/`。
2. 安装Frida模块，命令为`pip install frida`。

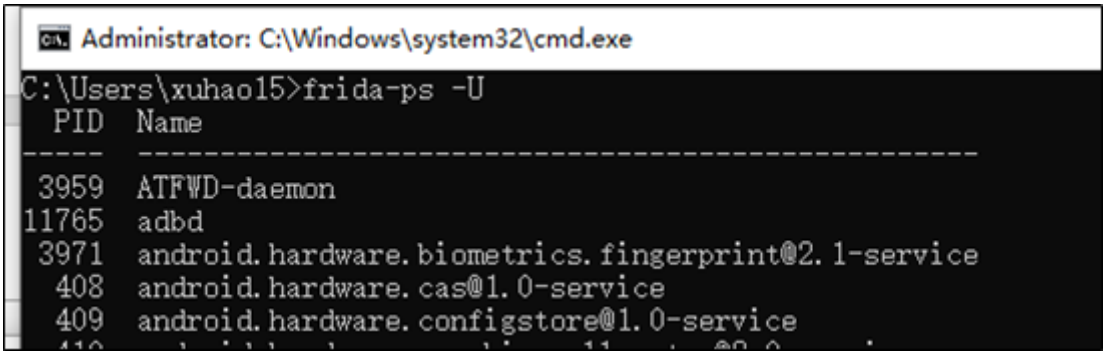
   (配置了多个python版本环境的可以使用命令`python -m pip install frida`防止用`pip install frida`命令报错)。
3. 安装frida-tools模块，`pip install frida-tools`。(或`python -m pip install frida-tools`)。

4. 下载运行在移动设备上的frida-sever端，官方下载地址：`https://github.com/frida/frida/releases`，下载时要选择对应的版本下载。

   （可以在adb使用命令`cat /proc/cpuinfo`查询，通常模拟器是x86，真机是arm）
5. 将第四步下载好的文件解压，然后通过命令`adb push frida-server /data/local/tmp`将文件传输到移动设备中，然后通过`adb shell`进入移动端，给文件赋权777。(`chmod 777 frida-server`)，并在root权限下启动(`./frida-server &`)。



6. 做完以上几步后，命令行输入命令`frida-ps -U`，能查看设备进程说明安装成功。



3.4 脱壳

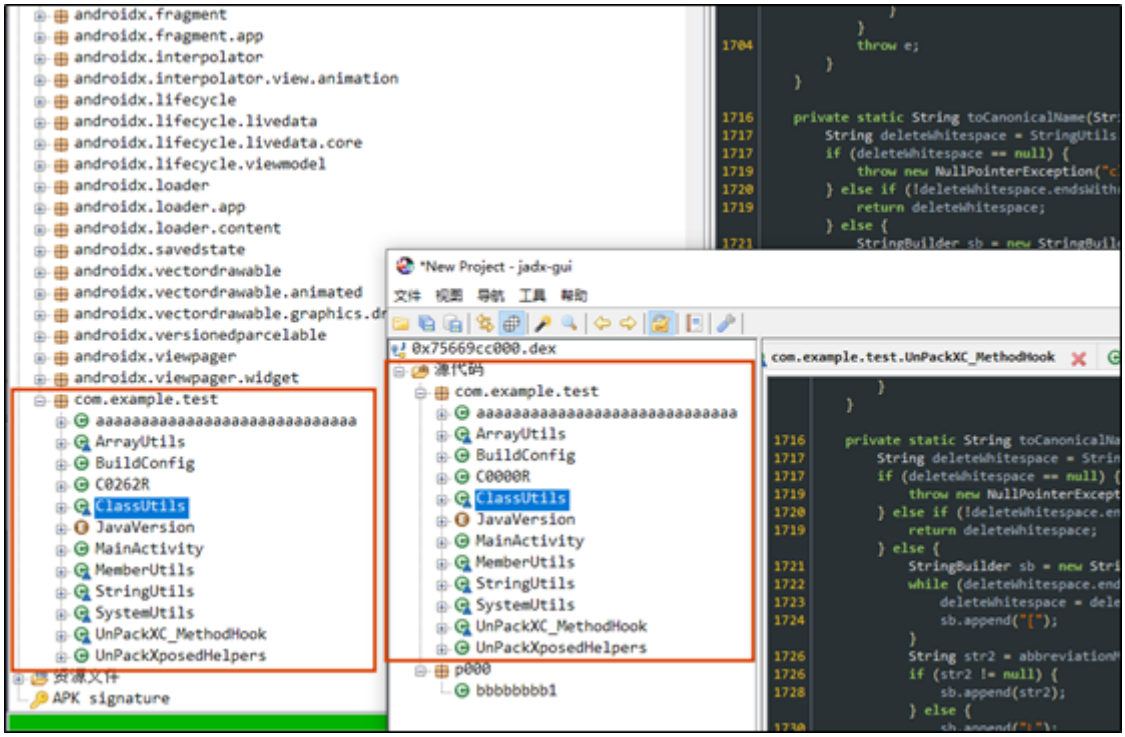
```
https://github.com/hluwa/FRIDA-DEXDump
```

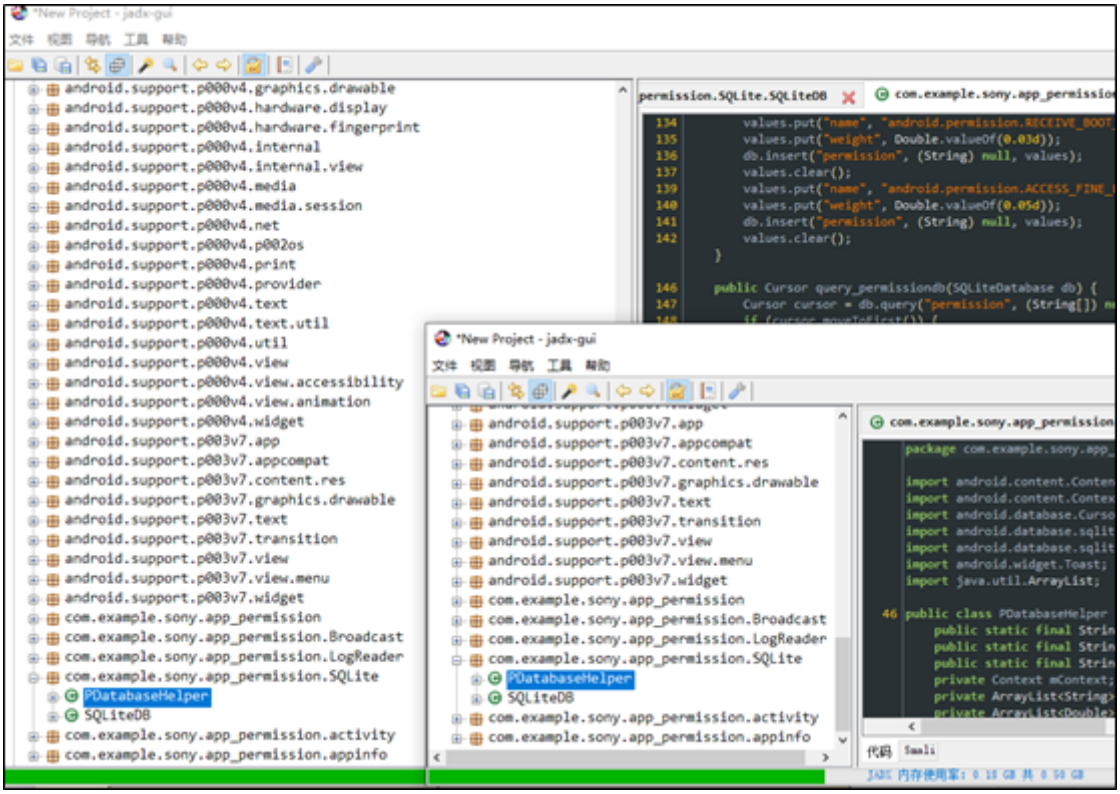
1. 移动端运行待测应用。
2. 修改main.py，pkg为待测应用包名。

```
def choose(pid=None, pkg=None, spawn=False, device=None):
    pkg = "com.example.test"
    if pid is None and pkg is None:
        target = device.get_frontmost_application()
        return target.pid, target.identifier

    for process in device.enumerate_processes():
        if (pid and process.pid == pid) or (pkg and process.name == pkg):
            if not spawn:
                return process.pid, process.name
            else:
                pkg = process.name
                break

    if pkg and spawn and device:
        pid = device.spawn(pkg)
        device.resume(pid)
        return pid, pkg
    raise Exception("Cannot found <{}> process".format(pid))
```

3. 执行main.py即进行脱壳操作。脱壳后的dex存放在运行脚本的当前目录下，jadx直接打开即可。

（测试百度免费版、梆梆安全免费版、360免费版等二代壳均可脱壳。左侧是原包，右侧是脱壳后的dex文件）





3.5 代码分析

- agent.js

```
Process.enumerateRanges('r--').forEach(function (range) {
            try {
                Memory.scanSync(range.base, range.size, "64 65 78 0a 30 ?? ??
00").forEach(function (match) {

                    if (range.file && range.file.path
                        && (// range.file.path.startsWith("/data/app/") ||
                            range.file.path.startsWith("/data/dalvik-cache/") ||
                            range.file.path.startsWith("/system/"))) {
                        return;
                    }

                    if (verify(match.address, range, false)) {
                        var dex_size = get_dex_real_size(match.address,
range.base, range.base.add(range.size));
                        result.push({
                            "addr": match.address,
                            "size": dex_size
                        });
                    }
                });
```

```
# 遍历当前进程中所有可以读的内存段
Process.enumerateRanges('r--')
# 搜索内存数据
Memory.scanSync()
# dex文件头魔术字段
"64 65 78 0a 30 ?? ?? 00"
```

- get_dex_real_size函数

```
function get_dex_real_size(dexptr, range_base, range_end) {
    var dex_size = dexptr.add(0x20).readUInt();

    var maps_address = get_maps_address(dexptr, range_base, range_end);
    if (!maps_address) {
        return dex_size;
    }

    var maps_end = get_maps_end(maps_address, range_base, range_end);
    if (!maps_end) {
        return dex_size;
    }

    return maps_end - dexptr
}
```

- verify_by_maps

```
function verify_by_maps(dexptr, mapsptr) {
    var maps_offset = dexptr.add(0x34).readUInt();
    var maps_size = mapsptr.readUInt();
    for (var i = 0; i < maps_size; i++) {
        var item_type = mapsptr.add(4 + i * 0xC).readU16();
        if (item_type === 4096) { //4096 == TYPE_MAP_LIST
            var item_offset = mapsptr.add(4 + i * 0xC + 8).readUInt();
            if (maps_offset === item_offset) {
                return true;
            }
        }
    }
    return false;
}
```

> 通过 `map_off` 找到 `DEX` 的 `dex_map_list`， 通过解析 `dex_map_list`，并得到类型为 `TYPE_MAP_LIST` 的条目。这个条目里面的索引值应该要与 `map_off` 一致，那么通过校验 `map_off` 和 `dex_map_list` 这两个地方，就可以更精确的验证内容中dex文件数据。

【参考资料】

[1] dex文件格式

[2] elf文件格式

[3] ELF文件格式解析

[4] Android so(ELF)文件解析

[5] dex文件解析(第三篇)

[6] Android so文件浅析

[7] Dalvik 可执行文件格式