某加固逆向分析

缘起

自己稍微了解一点加固，当然只是皮毛而已。前不久发了一篇脱壳的帖子（只能脱早期的壳，本身也只是发着玩玩）为了看看现在加固技术的演进和锻炼一下自己的逆向能力，遂选择了一个业界算知名度比较高的加固产品。

本人没有装逼之意，知道我大中华牛人数量之多可以多过我的遍身毛发，也仅是对自己的一个总结，文章必会有错误和疏漏之处，还请大佬们高抬贵手。我接受任何发自肺腑的建议。

缘落

本人目前时间还算充裕，但时间毕竟珍贵，遂我仅仅分析加固流程中 dex 如何解密，如何加载到内存中，后续关于 dex 的其他流程、以及保护 app 的相关流程我并没有分析。

原理

我这里会用最简洁的文字稍作描述（具体详细流程，请到 github 上下载）。

加固特征：dexHelper、classes0.jar

如何加固：classes0.jar 是一个经过加密的 zip 压缩包，压缩包里面包含被拆分的 dex，dex 的数量可能大于 1，加密算法是 rc4（稍有改变）+ 异或或者直接异或，选择加密方式是由加密算法第一个代表偏移的参数决定的。

如何加载：系统 hook 了 libc 很多函数（open、read、pread64、mmap 等）和 libart

函数（OpenDexFilesFromOat、DexFileVerifier::Verify 等），当系统调用 OpenDexFilesFromOat 函数后，如果传入参数 dex_location 匹配.cache/classes.jar 直接调用 bool DexFile::Open(const char* filename, const char* location, std::string* error_msg, std::vector<const DexFile*>* dex_files)加载 dex。加载过程中会进入到 hook 后 libc 的函数如 mmap、read 在这里进行解密解压，最终调用下面这个函数完成加载：

```
static const DexFile* DexFile::OpenMemory(const byte* base,
                                          size_t size,
                                          const std::string& location,
                                          uint32_t location_checksum,
                                          MemMap* mem_map, std::string* error_msg)
```

如何防止 dump：我发前面帖子时，曾用这个样本进行过测试，发现可以 dump 出 dex，但是当时并没有认真看 dex 是否是完整的（后来论坛上有人指出，我仔细一看确实 dex 的 code 变成了 nop）。当然逆向完成后我了解到它 hook 了很多点，当然还有一个神奇的地方我会在下面说。

加固疑问：经过分析，我发现解密解压后的 dex 竟然是完整的 dex，这让我很疑惑（我特别和之前 dump 出来的 dex 进行了对比）。我在分析前并没有找什么特定版本，随便在网上下载了一个，或许是我分析的版本是这个样子。那它为什么通过传统方式 dump 却 dump 出一个 code 抽取的 dex 呢，猜想应该是后面做了处理 (我并没有关注这些，没做继续分析)，这也就是上面我说的神奇的地方。还有如果它家所有版本都是这样的（除 vmp 外）那不是在做幌子嘛。

彩蛋

我在 github 上提供了对应的脱壳机，一种是基于完整解密 classes0.jar 实现的，一种是基于 frida hook 实现的，第二种只能用于安装后第一次运行（加固对后续内存中的 dex 做了处理，用很多方法 dump 都是没有 code 的）

声明

此文档只用于学习交流目的，用于其他目的本人概不负责

安全既是攻防，希望我所作能对防守的（加固方）一方提供帮助，看看攻击者都怎么做

各个厂商思路同中有异，我逆向分析仅站在学习者和探测加固强度角度进行，没有针对任何特定厂商

我逆向的版本不是最新版本，甚至我都不知道是哪个版本，请大家多吸取精华，抛弃糟粕

你可以吐槽我，不过还是希望尊重我的辛苦成果，有不对的地方，可以指出，大家互相探讨

对于逆向我也是个小学生，水平有限，还请大佬们一笑而过

Github 地址：https://github.com/ylcangel/crack_dexhelper