4.3 APK 逆向之反调试

为了保护应用关键代码，开发者需要采用各种方式增加关键代码逆向难度。调试技术是逆向人员理解关键代码逻辑的重要手段，对应的反调试技术则是应用开发者的“铠甲”。Android 下的反调试技术大多从 Windows 平台衍生而来，可以分为以下几类。

1. 检测调试器特征

- ❖ 检测调试器端口，如 IDA 调试默认占用的 23946 端口。
- ❖ 检测常用调试器进程名，如 android_server、gdbserver 等。
- ❖ 检测/proc/pid/status、/proc/pid/task/pid/status 下的 Tracepid 是否为 0。
- ❖ 检测/proc/pid/stat、/proc/pid/task/pid/stat 的第 2 个字段是否为 t。
- ❖ 检测/proc/pid/wchan、/proc/pid/task/pid/wchan 是否为 ptrace_stop。

2. 检测进程自身运行状态

- ❖ 检测父进程是否为 zygote。
- ❖ 利用系统自带检测函数 android.os.Debug.isDebuggerConnected。
- ❖ 检测自身是否被 ptrace。
- ❖ 检测自身代码中是否包含软件断点。
- ❖ 主动发出异常信号并捕获，如果没有被正常接收说明被调试器捕获。
- ❖ 检测某段程序代码运行时间是否超出预期。

攻击者绕过上述各类检测方式最便捷的方式是定制 Android ROM，从 Android 源码层面隐藏调试器特征。比如，通过 ptrace 函数检测是否被 ptrace 时，可以修改源码，让 ptrace 函数永远返回非调试状态，即可绕过 ptrace 检测；对系统自带的 isDebuggerConnected 函数，也可以通过修改源码绕过。总之，熟悉 Android 源码，准备一套专门针对反调试定制的系统，能大大加速逆向进程。

4.4 APK 逆向之脱壳

4.4.1 注入进程 Dump 内存

下面给出一段 Android8.1 下通过 Frida Hook 完成某加固脱壳的代码：

```
http://androidxref.com/8.1.0_r33/xref/art/runtime/dex_file.cc#OpenCommon
-------------------------------------------------------------------------------------------
   Interceptor.attach(Module.findExportByName("libart.so", "_ZN3art15DexFileVerifier6
         VerifyEPKNS_7DexFileEPKhjPKcbPNSt3__112basic_stringIcNS8_11char_traitsIcEENS8
         _9allocatorIcEEEE"), {
      onEnter: function(args) {
         console.log("verify..")
         var begin = args[1]

         var dex_size = args[2]

         var file = new File("/data/data/com.xxx.xxx/"+dex_size+".dex","wb")
         console.log("dex size:"+dex_size.toInt32())
```