深入理解 Android 逆向调试原理

转载

VIP_CQCRE 于 2021-03-26 21:41:06 发布 332 收藏 3
文章标签： 网络 编程语言 java 人工智能 python
原文链接：https://bbs.pediy.com/thread-266378.htm

这是「进击的Coder」的第 372 篇技术分享

论坛作者 ID：kxliping

来源：看雪论坛、看雪学院

“

阅读本文大概需要 17 分钟。

”

一、为什么要写这篇文章

使用 IDA 远程调试 Android SO 时，有一些场景需要在 init_array、JNI_OnLoad 等函数入口处下断点，对应的调试手法是以调试模式启动应用，处理完毕之后通过 jdb 命令通知应用继续运行，这种方法的优势是能够在程序开始运行之前获取程序的控制权。

在上面的步骤中，jdb 命令有一个选项，是调试端口（port），这个端口号一般通过 DDMS 获取，常见的教程也是教大家通过 DDMS 来获取。

但是 DDMS 已经被官方的开发套件弃用，而我既不想修改已经搭建好的环境，也不想额外下载旧版本的开发套件，所以打算研究一下有没有更加便捷的方式。

我最初的思路是这样：DDMS 在调试过程中最主要的作用就是展示端口号，这个端口号由 jdb 在 host 上直接使用，说明是 host 的本地端口，jdb 使用本地端口是无法直接与远程终端（这里就是 Android）通信的，说明经过了端口转发。

所以我的初步判断是，DDMS 通过端口转发将本地端口转发到了目标机器的端口，如果能够找到目标机器的端口，那就可以通过命令行来执行转发。

经过初步的探索，我找到了 DDMS 的替代方案。但在通过搜索引擎寻找答案的过程之中，我发现两个非常普遍的问题：一个是很多做 Android 逆向的同学，虽然对逆向调试的操作过程都很熟练，但是对这些操作的原理并不了解；第二个问题，就是网络上甚至实体教材上的教程，只注重操作，不注重背后的原理，这从源头上导致后来的学习者只知其然，不知其所以然。

因此我有了写一篇科普这方面基础知识文章的想法，希望用这篇文章帮助初入 Android 逆向领域的同学更好地入门，也希望所有对 Android 逆向调试的步骤有疑问的同学，能够从这一篇文章中找到答案。

二、文本的目标读者是谁

本文的目标读者包括：

1. 刚入门 Android 逆向的同学。

2. 想要深入了解 Android 逆向调试原理的同学。

3. 从事 Android 开发相关工作的同学。

三、阅读本文的潜在收益是什么

阅读本文的潜在收获包括但不限于如下几点：

1. 对 Android 调试原理较为深入的理解；

2. 对 Android 逆向调试原理的深入理解；

3. 掌握 Android 逆向调试的操作步骤；

4. 掌握在没有 DDMS 的情况下逆向调试 Android 的操作技巧。

本文不包括的内容：

1. 对 JDWP 协议细节的介绍；

2. 对调试器内部原理的介绍；

3. 相关原理源代码的详细分析。

四、Android 逆向调试的几个关键步骤

本文介绍的主要场景是使用 IDA 远程调试 Android SO，因为这个场景是最常用、知识面覆盖最广的 Android 逆向调试场景。

使用 IDA 远程调试 Android SO 的操作方式有两种，一种是在进程起来之后直接附加，另外一种是以调试模式启动应用。本文只讨论第二种，因为第二种操作方式能够完全覆盖第一种。

使用 IDA 远程调试 Android SO 的一般步骤如下：

1. 以调试模式启动应用

Adb shell 下执行命令：

```
am start -D -n com.outdoor.debugtest/.MainActivity
```

操作结果，应用进入等待调试连接的界面：

2. 启动 android_server

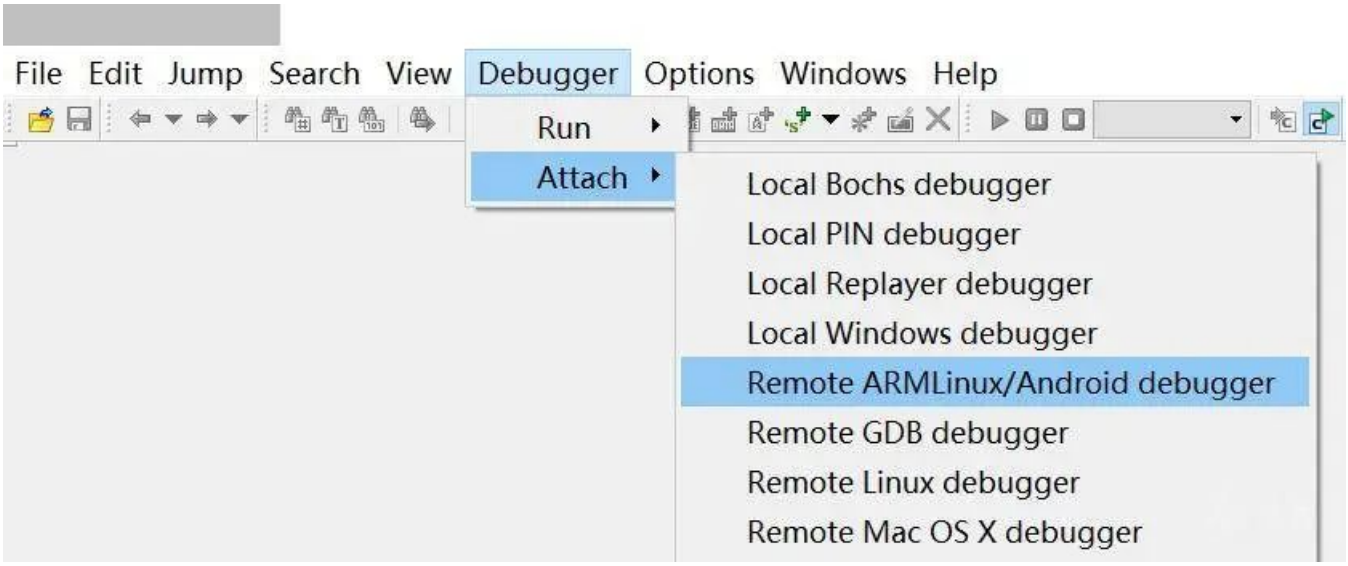
```
ringtag:/data/local/tmp # ./android_server64


IDA Android 64-bit remote debug server(ST) v1.22. Hex-Rays (c) 2004-2017
Listening on 0.0.0.0:23946...
```

3. 执行端口转发

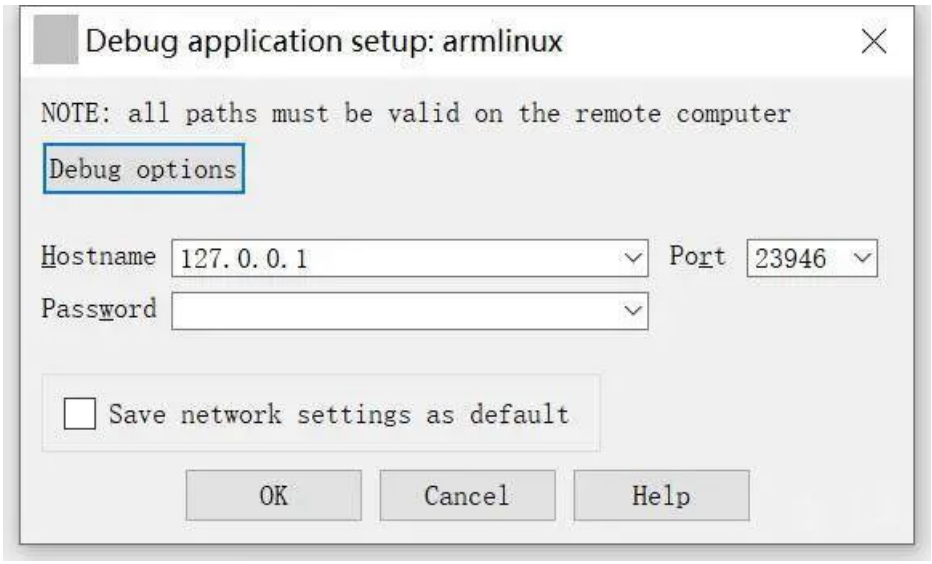
```
adb forward tcp:23946 tcp:23946
```

4. IDA 附加到调试进程

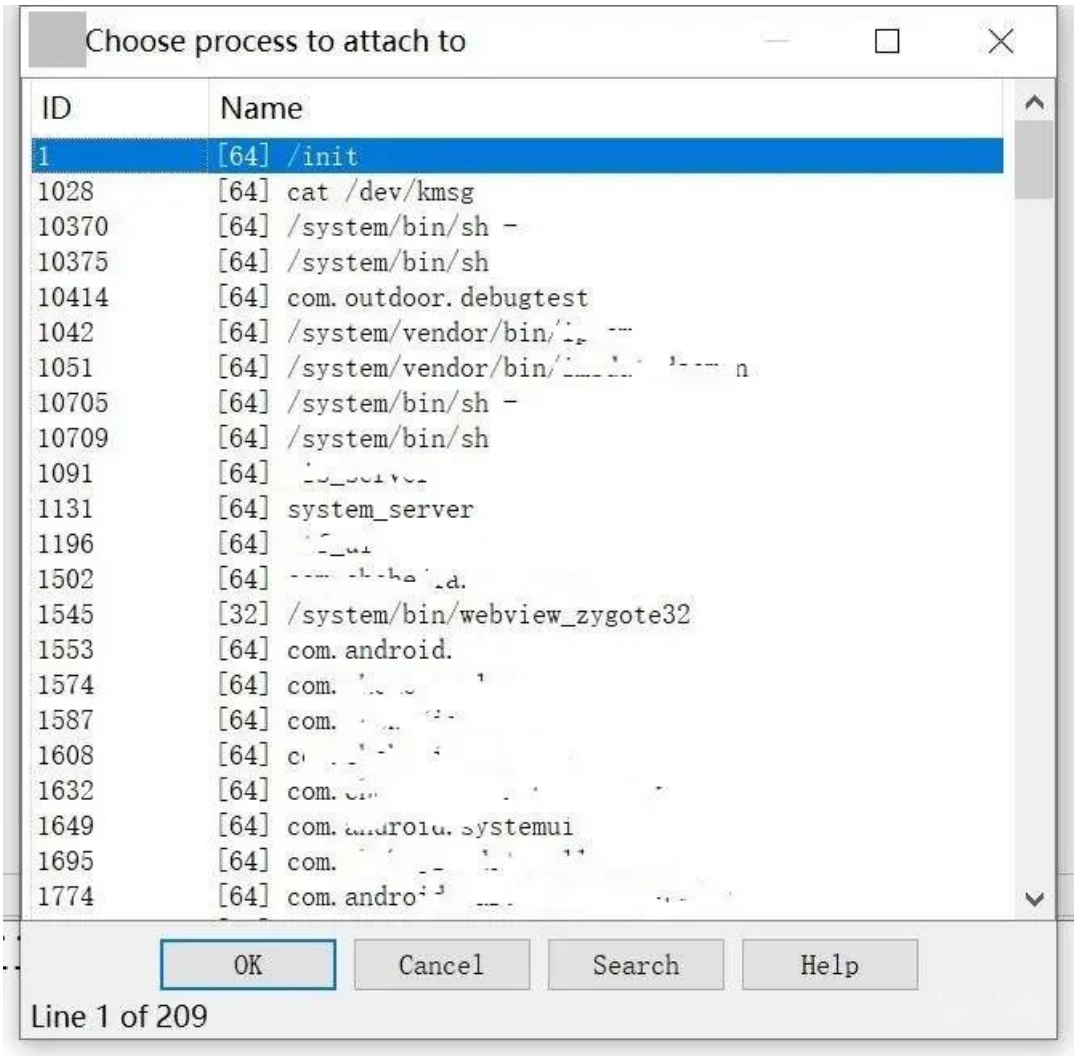
选择远程 ARMLinux/Android debugger：



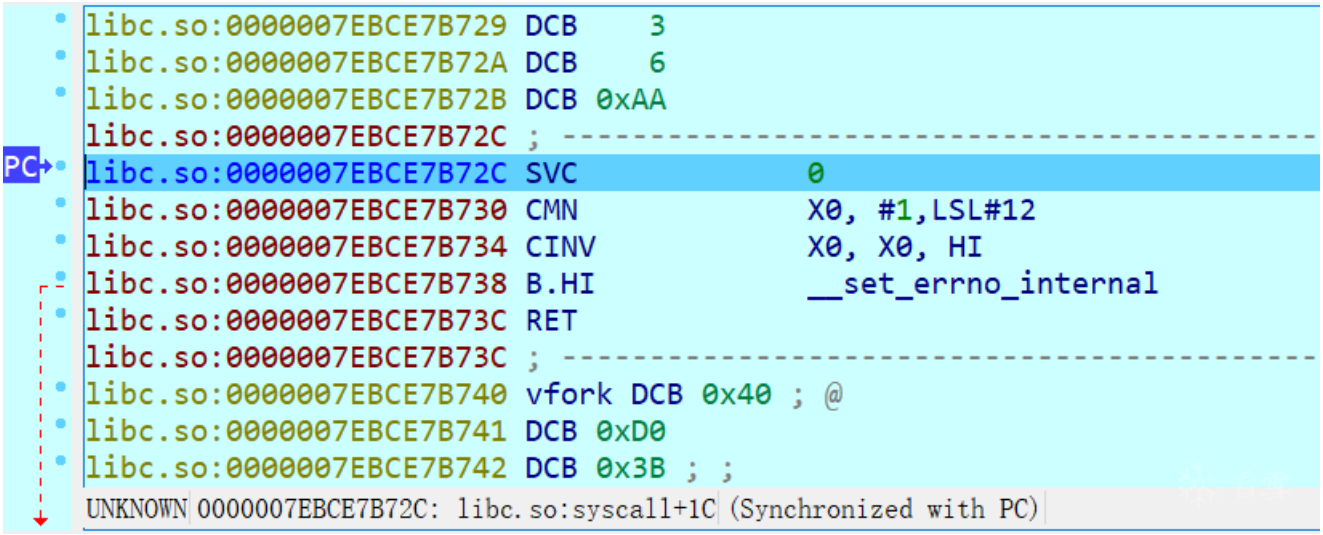
填写 hostname 为 127.0.0.1 或 localhost：



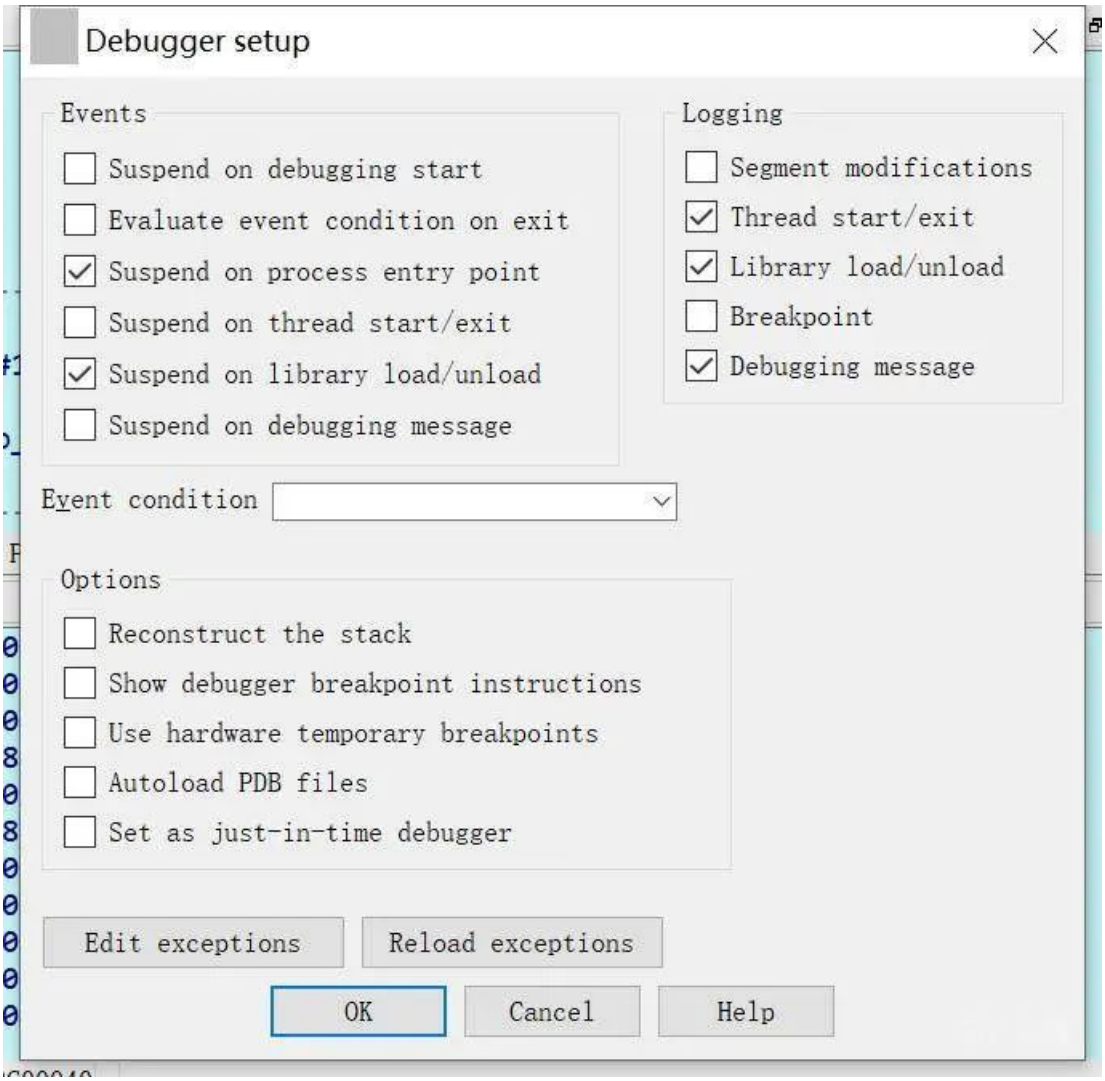
附加到调试进程：



成功附加到调试进程：



配置调试选项，勾选“Suspend on library load/unload”，以在目标 SO 加载时进程能够停下来：



5. Jdb 连接到调试进程的 VM

DDMS 查看调试进程端口号，假设为 55555，在 host 机器上执行：

```
C:\Users\reverser>jdb -connect com.sun.jdi.SocketAttach:hostname=localhost,port=55555
设置未捕获的java.lang.Throwable
设置延迟的未捕获的java.lang.Throwable
正在初始化jdb...
>
```

IDA 中按 F9 进程继续运行，直到目标 SO 被加载。

IDA output 窗口显示目标 SO 文件被加载：

```
PDBSRC: loading symbols for '/data/app/com.outdoor.debugtest-iQxBx29FyKFuLtnU6wwQag==/lib/arm64/libnative-l
```

6. 在目标 SO 上下断点

在 IDA Modules 模块中搜索目标 SO 库并双击选中，找到目标函数下断，即可开启调试流程。

以上便是 IDA 远程调试 Android SO 的关键步骤，针对如上步骤，我现在提出如下几个疑问供大家思考：

a. DDMS 展示的端口号，是怎么得来的？有没有更加便捷的方式获取这个端口号？

b. Jdb connect 命令是 Java VM 调试相关的命令，在 SO 调试中为什么要用它？它在和哪个模块进行通信？IDA 附加上进程之后为什么不能直接开始调试？

c. Adb、android_server、IDA 之间的关系是什么？IDA 与应用进程之间的调试连接到底是如何建立的？IDA 调试 Android SO 的技术原理是什么？

以上问题由浅入深，从对操作技巧层面的思考，到对背后调试原理的思考，这也是我自己探索这个问题的一个过程。在回答这些问题之前，我们需要做一些关于调试原理的铺排工作，以帮助大家全面、深入地理解问题。

五、Android 调试模型分析

本小节对Android调试模型的介绍，主要参考自源码文件 /system/core/adb/jdwp_service.cpp 中的相关注释。因为这一部分注释详细阐述了 Android 调试模型是如何建立起来的，所以我会首先翻译这个模型的建立过程，然后进行一个抽象的总结，以帮助大家更好地理解 Android 的调试模型。

1. Android 源码对调试模型建立的描述

（1）当 adbd 启动之后，它会创建一个 unix 类型的 server socket 套接字，名字叫 @jdwp-control；

（2）当一个新的 JDWP 守护线程在一个新的 VM 进程中启动时，这个新启动的 JDWP 线程会创建一个到 @jdwp-control 的连接，以宣告它自己的可用性：

```
50       JDWP thread                              @jdwp-control
51           |                                         |
52           |------------------------------->         |
53           | hello I'm in process <pid>              |
54           |                                         |
55           |                                         |
```

上面这个连接一直保持着，直到 JDWP 线程所在的进程结束；

（3）因此，adbd 维护着一个 JDWP 列表，记录着每个活跃的进程。Adbd 可以通过“device:debug-ports”服务同每一个客户进程进行通信；

（4）当有调试器想要连接时，调试器执行“adb forward tcp:<hostport> jdwp:<pid>”命令，“jdwp:<pid>”用来指定设备上的目标 JDWP 进程。

（5）Adbd 收到命令后，执行如下操作：

调用 socketpair() 创建一对“equivalent sockets”；

将第一个 socket 附加到本地 socket，本地 socket 本身又连接到远程 socket；

使用 sendmsg() 将第二个 socket 的文件描述符直接发送给 JDWP 进程。

```
JDWP thread                              @jdwp-control
    |                                          |
    |                  <-----------------------|
    |          OK, try this file descriptor    |
    |                                          |
    |                                          |
```

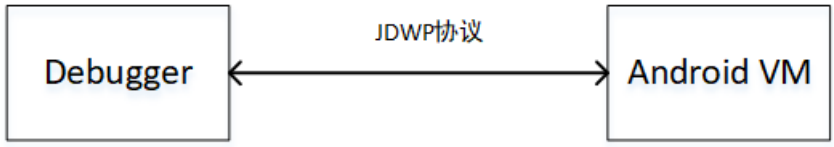
（6）接着，JDWP 线程使用这个新的 socket 描述符作为它与调试器的通信通道（接收 JDWP-Handshake 消息，应答调试器的消息等等）。

整个过程建立起来后，可以用下面这个图表示：

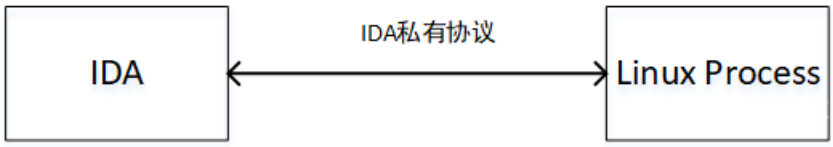
```
             ____________________________________
           |                                    |
           |          ADB Server (host)         |
           |                                    |
Debugger <---> LocalSocket <----> RemoteSocket  |
           |                              ^^    |
           |______________________________||____|
                                          ||
                                Transport ||
   (TCP for emulator - USB for device) ||
                                          ||
            ______________________________||____
           |                              ||    |
           |          ADBD  (device)      ||    |
           |                              VV    |
 JDWP <=======> LocalSocket <----> RemoteSocket  |
           |                                    |
           |____________________________________|
```

2. 对 Android 调试模型的总结

对于如上内容，我要明确阐述的第一个点是，以上调试模型是针对 Android VM 的调试模型，这个模型的顶层形式如下图，其中，调试器通过 JDWP 协议与 Android 虚拟机进行通信。



而 IDA 对 android 的支持，是在整个进程的层面，并不局限于 Android 虚拟机。这其中的区别可以用如下方法简单区分，IDA 能够支持对原生程序进程的调试，如 linker、init 进程等等，而基于 JDWP 协议的这一套原生调试框架并不支持。



归纳而言之，原生基于 JDWP 的调试框架与 IDA 的 android 调试框架，是两个层面上的调试实现，前者是虚拟机层面的实现，后者是整个进程层面的实现。理解这一点对理解 Android 逆向调试的原理至关重要。

六、逆向调试关键步骤解密

有了以上知识的铺垫，我们就可以开始研究第四章中提出的几个问题了。

1. DDMS 展示的端口号，是怎么得来的？有没有更加便捷的方式获取这个端口号？

根据前一章的阐述，调试器通过执行“adb forward tcp:<hostport> jdwp:<pid>”命令将 host 机器上的 hostport 端口转发到 Android 上的调试进程，以便调试器通过这个端口连接到目标进程。所以这个 hostport 是调试器指定的，据此推测，DDMS 便是通过这种转发方式获取到这个端口并展示到界面上。

因此，我们可以通过“adb forward tcp:<hostport> jdwp:<pid>”命令，替代使用 DDMS 的方式。

与问题 1 相关的进一步思考

当我们以调试模式启动应用时，应用会输出如下形式的日志信息:

```
--------- beginning of system03-08 10:01:05.709 6761 6761 W ActivityThread: Application com.outdoor.appcomv
```

日志信息表明应用在端口 8100 处等待调试连接，那是不是表明我们可以直接将 hostport 转发至 Android 8100 端口呢，尝试后你会发现不会成功。原因如下：

Android 应用的调试模式有“dt_socket”和“dt_android_adb”两种，即通过 socket 或者 adb 连接到 JDWP 线程。所以理论上，通过将 hostport 转发到 8100 也是可以实现的。但进一步分析源码之后发现，Android 虚拟机默认以“dt_android_adb”的模式启动，并且这个启动参数硬编码在源文件之中，不可手动配置的。

frameworks/base/core/jni/AndroidRuntime.cpp

```
601  int AndroidRuntime::startVm(JavaVM** pJavaVM, JNIEnv** pEnv, bool zygote)602  {……772 773      /*774
……995      if (JNI_CreateJavaVM(pJavaVM, pEnv, &initArgs) < 0) {996          ALOGE("JNI_CreateJavaVM failed
```

所以直接转发 8100 端口会失败，因为 Android VM 的启动模式是“dt_android_adb”，JDWP 线程也没有监听 8100 端口，这条日志给很多人带来了困扰。

最后，如果想要实现通过 socket 直接连接到 JDWP 线程，有没有办法呢？答案是有的。有两种可行的方式，一种是安装 hook 框架直接修改参数，但这种做法意义不大；另外一种是通过操作动态库的方式（dlopen、dlsym），修改调试启动参数，然后重启 JDWP 线程，这种适合实现生产环境下的远程调试。

2. Jdb connect 命令是 Java VM 调试相关的命令，在 SO 调试中为什么要用它？它在和哪个模块进行通信？IDA 附加上进程之后为什么不能直接开始调试？

在前面一章中，我们明确了通过原生 JDWP 的方式调试与使用 IDA 调试是两个层面的调试手段。

使用 Jdb 的目的，是为了通知 VM 继续运行，因为应用以调试模式启动（以调试模式启动的原因是为了在程序早期下断点），启动之后一直在等待调试连接，Jdb 通过“dt_android_adb”的方式，连接到了目标进程的 VM，VM 继续运行，这样 IDA 才可以在运行着的进程上进行调试。

归纳而言之，对 JDWP 调试手段的运用，是一种辅助手段，这种辅助手段使得我们能够在进程早期下断点。

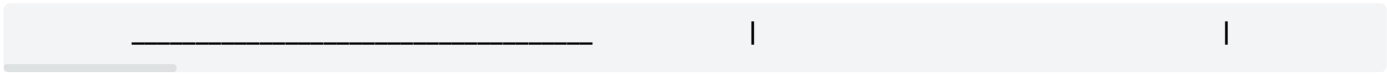
3. Adb、android_server、IDA 之间的关系是什么？IDA 与应用进程之间的调试连接到底是如何建立的？IDA 调试 Android SO 的技术原理是什么？

根据前两个小节的内容，我们知道了 adb 套件实际上是在辅助建立调试连接：

a. 在 JDWP 调试连接中，提供“dt_android_adb”模式所需的通信功能；

b. 在 IDA 调试连接中，提供端口转发、远程启动进程的手段。

而 android_server、IDA 之间的关系，基本上与原生调试模型保持着一致，它是 IDA 对 android 逆向调试的实现：



android_server 的作用类似于 adbd，区别在于，adbd 通过本地 socket 与 JDWP 进行通信，以转发调试信号；而 android_server 此处实际上是基于 ptrace 实现的一个调试器，它一端通过 socket 与 IDA 相连接，传输调试指令和数据，另一端通过 ptrace 直接操控调试进程。

七、总结

经过以上几个章节的阐述，我希望已经讲清楚我提出的那些问题，也希望这篇文章能够达到它的目的。

- END -

崔庆才丨静觅

北京 海淀

扫一扫上面的二维码图案，加我微信

看完记得关注@进击的Coder

及时收看更多好文

↓↓↓

点个在看你最好看

创作打卡挑战赛
赢取流量/现金/CSDN周边激励大奖