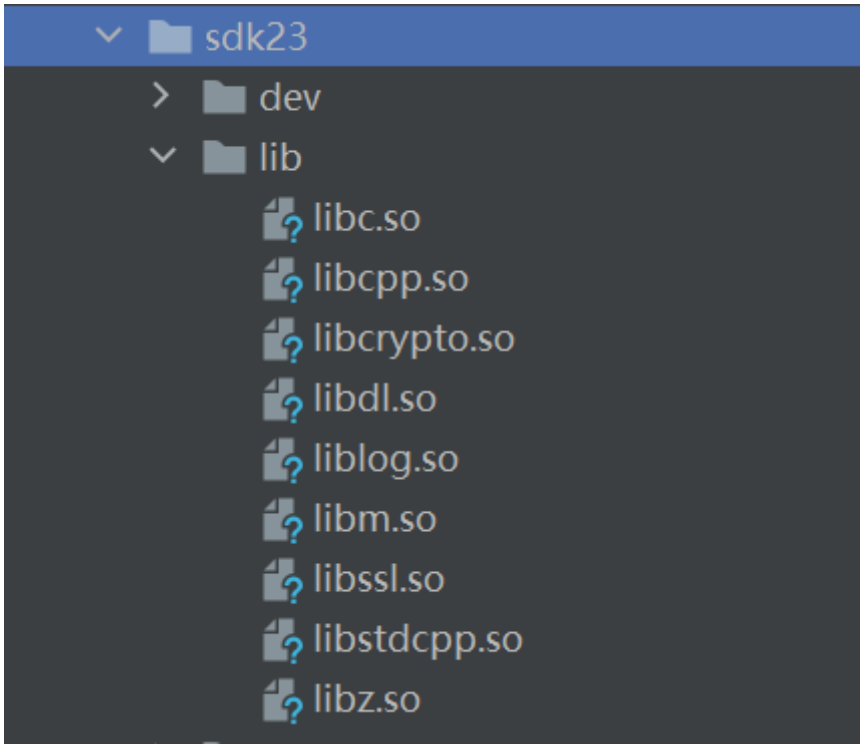
# 依赖库问题

## 一、前言

*Unidbg*仅能完美支持几个系统*SO*，使用一些*Unidbg*尚未支持的依赖库，可以极大的阻挠*Unidbg*分析者。

## 二、描述

SO常常需要依赖底层SO，Unidbg良好测试和支持了最常用的几个




不支持更多的系统SO，因为很多系统SO的依赖项很复杂，层层叠叠像个大关系网，想顺利、完美的在Unidbg中加载起来太不容易了。以极为常见的libandroid.so为例。

```
LOAD:00000000
LOAD:00000000 ; File Name   : C:\Users\pr0214\Desktop\DTA\Unidbg学习指南\Unidbg基础篇\补环境\补SO\libandroid.
LOAD:00000000 ; Format      : ELF for ARM (Shared object)
LOAD:00000000 ; Needed Library 'liblog.so'
LOAD:00000000 ; Needed Library 'libhidlbase.so'
LOAD:00000000 ; Needed Library 'libcutils.so'
LOAD:00000000 ; Needed Library 'libandroidfw.so'
LOAD:00000000 ; Needed Library 'libinput.so'
LOAD:00000000 ; Needed Library 'libutils.so'
LOAD:00000000 ; Needed Library 'libbinder.so'
LOAD:00000000 ; Needed Library 'libui.so'
LOAD:00000000 ; Needed Library 'libgui.so'
LOAD:00000000 ; Needed Library 'libharfbuzz_ng.so'
LOAD:00000000 ; Needed Library 'libsensor.so'
LOAD:00000000 ; Needed Library 'libandroid_runtime.so'
LOAD:00000000 ; Needed Library 'libminikin.so'
LOAD:00000000 ; Needed Library 'libnetd_client.so'
LOAD:00000000 ; Needed Library 'libhwui.so'
LOAD:00000000 ; Needed Library 'libxml2.so'
LOAD:00000000 ; Needed Library 'android.hardware.configstore@1.0.so'
LOAD:00000000 ; Needed Library 'android.hardware.configstore-utils.so'
LOAD:00000000 ; Needed Library 'libc++.so'
LOAD:00000000 ; Needed Library 'libc.so'
LOAD:00000000 ; Needed Library 'libm.so'
LOAD:00000000 ; Needed Library 'libdl.so'
LOAD:00000000 ; Shared Name 'libandroid.so'
```

依赖库太多了，这些对应的依赖库，怎么顺利加载并跑起来，又是很大的问题，因此Unidbg目前只支持上图的这些系统SO。如果目标SO依赖libandroid.so或者其他系统SO呢？总得有个解决方案吧？

最朴素的办法是Patch，如果目标SO依赖某个Unidbg实在无能为力的SO，那就视具体函数进行patch实现其函数功能或者直接返回。但这总有些不优雅，所以Unidbg又设计了VirtualModule的概念，即虚拟SO，可以理解为一种更优雅的Patch。比如libandroid.so 和 libJniGraphics 在Unidbg中就做了虚拟SO的实现， *unidbg/android/src/main/java/com/github/unidbg/virtualmodule/android* 下就是相关代码。Unidbg实现了这两个SO中几个常用函数。

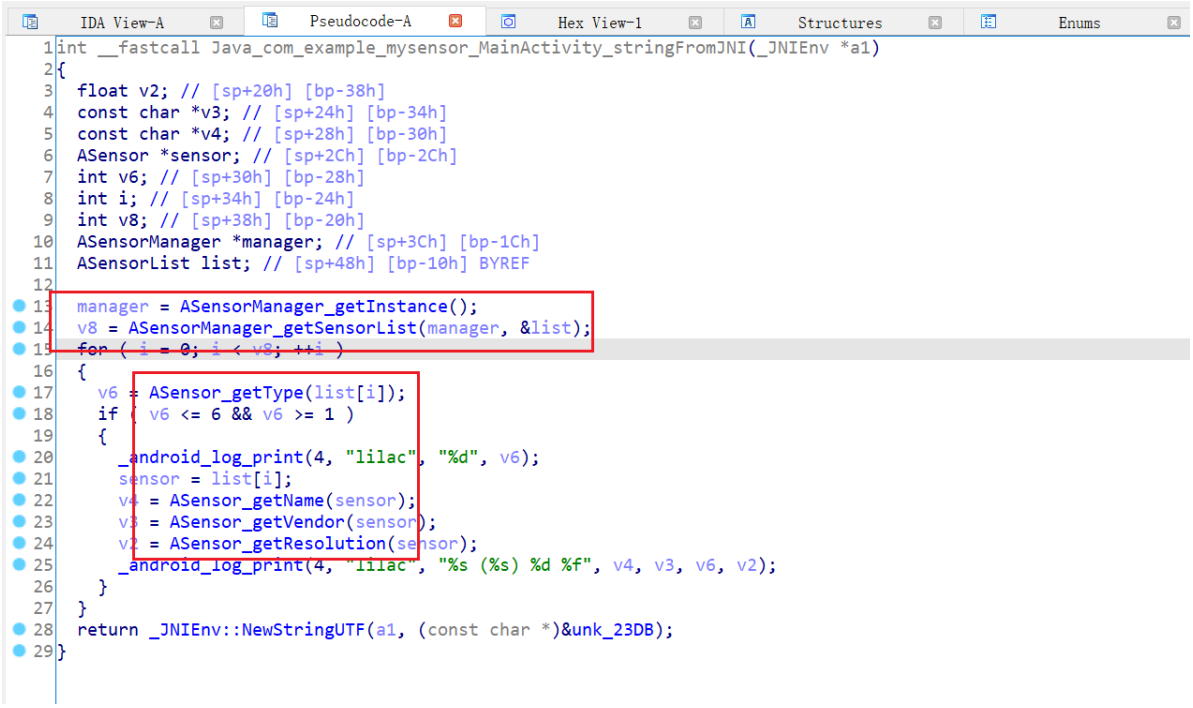
## ### 三、Anti 思路

那么anti 思路就很简单了，加载Unidbg不支持的SO，甚至只要使用Unidbg VirtualModule尚不支持的函数都行。比如libandroid.so，这个模块接近两百个导出函数，但Unidbg的AndroidModule只初步实现了不到十个函数——其中最常用的函数。

下面代码功能为NDK中获取传感器信息

```
#include <jni.h>
#include <string>
#include <android/sensor.h>
#include <android/asset_manager.h>
#include <android/log.h> // 日志

#define TAG "lilac"
// 定义info信息
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO,TAG,__VA_ARGS__)

extern "C" JNIEXPORT jstring JNICALL
Java_com_example_mysensor_MainActivity_stringFromJNI(
        JNIEnv* env,
        jobject /* this */) {

    ASensorList list;

    // 初始化传感器管理器
    ASensorManager * mpSensorManager = ASensorManager_getInstance();

    int count = ASensorManager_getSensorList(mpSensorManager, &list);
    for(int i=0;i<count;i++){
        int type = ASensor_getType(list[i]);
        const ASensor* sensor = list[i];
        // Returns this sensor's name (non localized)
        const char* name = ASensor_getName(sensor);
        // Returns this sensor's vendor's(供应商) name (non localized)
        const char* vendor = ASensor_getVendor(sensor);
        // Returns this sensors's resolution.(分辨率)
        float resolution = ASensor_getResolution(sensor);
        LOGI("%s (%s) %d %f", name, vendor, type, resolution);
    }


    return env->NewStringUTF("");
}
```

这几个方法是libandroid.so封装的，来自libsensor.so的API。




分析者需要完善Unidbg的AndroidModule源码，才可能顺利跑通。这并不是一种很容易的事，可以阻挠大部分只会使用工具的攻击者，笔者在附件中放了简易的补充代码，感兴趣的可以分析一下。

## 四、尾声

无。