一、前言

Hi大家好，前面我们已经讨论过许多Unidbg补JAVA环境的技巧与实例，也遭遇过补文件访问、补依赖SO、补系统调用等场景。我从朋友们的反馈中感受到，大家普遍认为，Unidbg的学习门槛并不搞，看几个案例就能上手，问题在于Unidbg和样本结合时遇到的各种报错，报错时如何处理，需要经验和技法的支持。今天我们讨论的这个样本，主要聚焦在这样一个场景上——“样本获取系统属性怎么补？ ”

二、准备

什么叫样本获取系统属性？NDK中最常见的方式是通过JNI调用，比如如下代码获取SERIAL

```
jclass androidBuildClass = env->FindClass("android/os/Build");
jfieldID SERIAL = env->GetStaticFieldID(androidBuildClass, "SERIAL",
"Ljava/lang/String;");
jstring serialNum = (jstring) env->GetStaticObjectField(androidBuildClass,
SERIAL);
```

通过JNI调用JAVA方法获取本机的属性和信息，是最常见的做法，除了Build类，常见的还有System.getProperty和Systemproperties.get等API。Unidbg补环境过程中，最好补而且不会遗漏的就是这一类，因为Unidbg会给出清楚的报错，你没法对它置之不理。

第二个常见方式是通过system_property_get 函数获取系统属性也是常见做法

```
char *key = "ro.build.id";
char value[PROP_VALUE_MAX] = {0};
__system_property_get(key, value);
```

这类环境缺失容易被大家忽视，因为没有日志提示，即使src/test/resources/log4j.properties中日志全开，也不会打印相关信息。

第三个常见方式是通过文件访问，比如读取/proc/pid/maps，此种情况，Unidbg会提供日志输出，但经常被大家忽视，事实上，不少朋友初学Unidbg时除了JAVA环境的报错，其他日志信息都不去管。

第四个常见方式是通过popen()管道从shell中获取系统属性，其效果可以理解成在NDK中使用adb shell，popen参数一就是shell命令，返回值是一个fd文件描述符，可以read其内容，其中内容就是adb shell执行该命令应该返回的内容。

```
char value[PROP_VALUE_MAX] = {0};
std::string cmd = "getprop ro.build.id";
FILE* file = popen(cmd.c_str(), "r");
fread(value, PROP_VALUE_MAX, 1, file);
pclose(file);
```

除此之外，system函数也可以做这一件事，两者在底层都依赖于execve系统调用。

第五个常见方式是通过 getenv函数 获取进程环境变量，首先，Android系统层面存在一些默认的环境变量，除此之外，样本可以设置自己进程内的环境变量。因此，样本可以在Native层获取系统环境变量或者自身JAVA层设置的环境变量。

> getenv()用来取得环境变量的内容。参数为环境变量的名称，如果该变量存在则会返回指向该内容的指针，如果不存在则返回null。

我们可以通过ADB 查看环境变量有哪些，也可以查看环境变量的值。

```
C:\Users\pr0214>adb shell
bullhead:/ $ export
ANDROID_ASSETS
ANDROID_BOOTLOGO
ANDROID_DATA
ANDROID_ROOT
ANDROID_SOCKET_adbd
ANDROID_STORAGE
ASEC_MOUNTPOINT
BOOTCLASSPATH
DOWNLOAD_CACHE
EXTERNAL_STORAGE
HOME
HOSTNAME
LOGNAME
PATH
SHELL
SYSTEMSERVERCLASSPATH
TERM
TMPDIR
USER
bullhead:/ $ echo $HOME
/
bullhead:/ $ echo $ANDROID_DATA
/data
bullhead:/ $ echo $SYSTEMSERVERCLASSPATH
/system/framework/services.jar:/system/framework/ethernet-
service.jar:/system/framework/wifi-
service.jar:/system/framework/com.android.location.provider.jar
bullhead:/ $ echo $PATH
/sbin:/system/sbin:/system/bin:/system/xbin:/vendor/bin:/vendor/xbin
bullhead:/ $
```

第六个常见方式是使用系统调用获取相关属性，不管是通过syscall函数还是内联汇编，都属此类。

常见的比如uname系统调用

> uname - 获取当前内核的名称和信息
>
> 返回的信息是一个结构体
>
> ```
> struct utsname {
>        char sysname[];    /* 操作系统名称 (例如 "Linux") */
>        char nodename[];   /* "一些实现了的网络"内的名称*/
>        char release[];    /* 操作系统版本 (例如 "2.6.28")*/
>        char version[];    /* 操作系统发布日期 */
>        char machine[];    /* 硬件标识符 */
> #ifdef _GNU_SOURCE
>        char domainname[]; /* NIS或YP域名 */
> #endif
> };
> ```

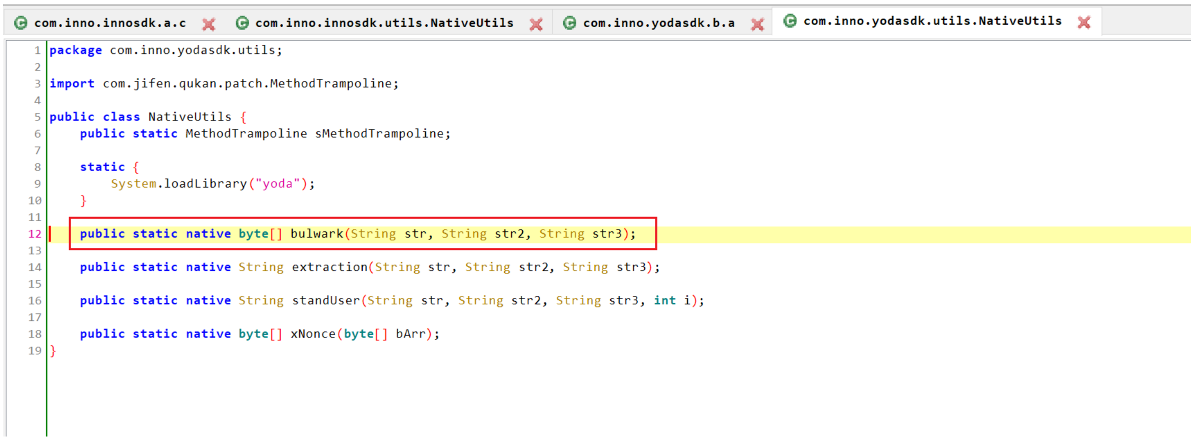
日志全开的情况下，系统调用的相关调用会被全部打印，大家看仔细一些就没什么问题。值得一提的是，Unidbg的uname系统调用实现是个很好也很简单的检测点，十分规范的表明了自己是Unidbg。

```
protected int uname(Emulator<?> emulator) {
    Pointer buf = UnidbgPointer.register(emulator, ArmConst.UC_ARM_REG_R0);
```

```
    if (log.isDebugEnabled()) {
        log.debug("uname buf=" + buf);
    }

    final int SYS_NMLN = 65;

    Pointer sysname = buf.share(0);
    sysname.setString(0, "Linux");

    Pointer nodename = sysname.share(SYS_NMLN);
    nodename.setString(0, "localhost");

    Pointer release = nodename.share(SYS_NMLN);
    release.setString(0, "1.0.0-unidbg");

    Pointer version = release.share(SYS_NMLN);
    version.setString(0, "#1 SMP PREEMPT Thu Apr 19 14:36:58 CST 2018");

    Pointer machine = version.share(SYS_NMLN);
    machine.setString(0, "armv7l");

    Pointer domainname = machine.share(SYS_NMLN);
    domainname.setString(0, "localdomain");

    return 0;
}
```

以上这些是较为常见的获取系统属性的方式，其中有一些方式和API，容易被刚使用Unidbg的新手忽视，我们需要一个办法帮助大家意识到需要补充它们。我们到第三节讨论一下这个问题。

三、Unidbg 实战



com.inno.innosdk.a.c
com.inno.innosdk.utils.NativeUtils
com.inno.yodasdk.b.a
com.inno.yodasdk.utils.NativeUtils

```
package com.inno.yodasdk.utils;

import com.jifen.qukan.patch.MethodTrampoline;

public class NativeUtils {
    public static MethodTrampoline sMethodTrampoline;

    static {
        System.loadLibrary("yoda");
    }

    public static native byte[] bulwark(String str, String str2, String str3);

    public static native String extraction(String str, String str2, String str3);

    public static native String standUser(String str, String str2, String str3, int i);

    public static native byte[] xNonce(byte[] bArr);
}
```

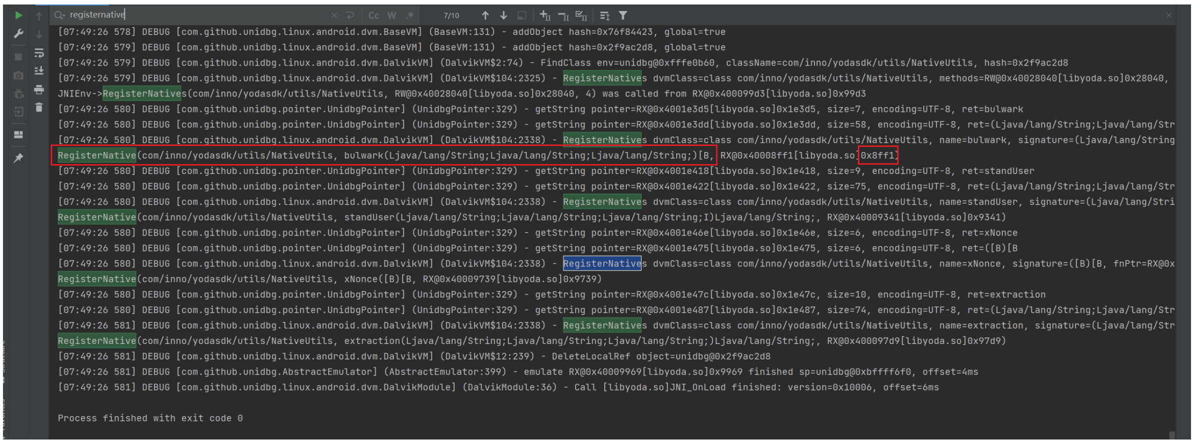
如图是目标函数，Frida hook 获取入参。这个函数看着没有初始化函数，一副可以单独调用的样子。

我们首先写一些基础性的代码，如果按照平时，会是这样的

```
package com.lession5;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
import com.github.unidbg.arm.backend.Backend;
import com.github.unidbg.arm.backend.BlockHook;
```

```java
import com.github.unidbg.arm.context.RegisterContext;
import com.github.unidbg.debugger.BreakPointCallback;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.SystemPropertyHook;
import com.github.unidbg.linux.android.SystemPropertyProvider;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;
import unicorn.Unicorn;

import java.io.File;

public class yoda extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    yoda(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机
        vm = emulator.createDalvikVM(new File("unidbg-android/src/test/resources/qdd/qtt_new.apk"));
        // 设置是否打印相关调用细节
        vm.setVerbose(true);
        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File("unidbg-android/src/test/resources/qdd/libyoda.so"), true);
        module = dm.getModule();
        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }

    public static void main(String[] args) {
        yoda demo = new yoda();
    }
}
```

然后运行

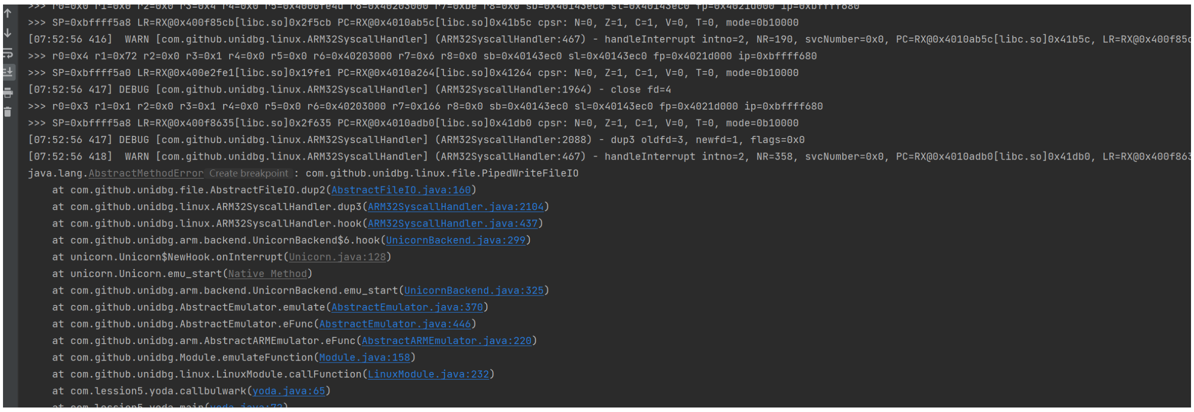


```
[07:49:26 578] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x76f84423, global=true
[07:49:26 579] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x2f9ac2d8, global=true
[07:49:26 579] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$2:74) - FindClass env=unidbg@0xfffe0b60, className=com/inno/yodasdk/utils/NativeUtils, hash=0x2f9ac2d8
[07:49:26 579] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$104:2325) - RegisterNatives dvmClass=class com/inno/yodasdk/utils/NativeUtils, methods=RW@0x40028040[libyoda.so]0x28040,
JNIEnv->RegisterNatives(com/inno/yodasdk/utils/NativeUtils, RW@0x40028040[libyoda.so]0x28040, 4) was called from RX@0x400099d3[libyoda.so]0x99d3
[07:49:26 580] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e3d5[libyoda.so]0x1e3d5, size=7, encoding=UTF-8, ret=bulwark
[07:49:26 580] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e3dd[libyoda.so]0x1e3dd, size=58, encoding=UTF-8, ret=(Ljava/lang/String;Ljava/lang/Str
[07:49:26 580] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$104:2338) - RegisterNatives dvmClass=class com/inno/yodasdk/utils/NativeUtils, name=bulwark, signature=(Ljava/lang/String
RegisterNative(com/inno/yodasdk/utils/NativeUtils, bulwark(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)[B, RX@0x40008ff1[libyoda.so]0x8ff1)
[07:49:26 580] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e418[libyoda.so]0x1e418, size=9, encoding=UTF-8, ret=standUser
[07:49:26 580] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e422[libyoda.so]0x1e422, size=75, encoding=UTF-8, ret=(Ljava/lang/String;Ljava/lang/Str
[07:49:26 580] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$104:2338) - RegisterNatives dvmClass=class com/inno/yodasdk/utils/NativeUtils, name=standUser, signature=(Ljava/lang/Stri
RegisterNative(com/inno/yodasdk/utils/NativeUtils, standUser(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;I)Ljava/lang/String;, RX@0x40009341[libyoda.so]0x9341)
[07:49:26 580] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e46e[libyoda.so]0x1e46e, size=6, encoding=UTF-8, ret=xNonce
[07:49:26 580] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e475[libyoda.so]0x1e475, size=6, encoding=UTF-8, ret=([B)[B
[07:49:26 580] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$104:2338) - RegisterNatives dvmClass=class com/inno/yodasdk/utils/NativeUtils, name=xNonce, signature=([B)[B, fnPtr=RX@0x
RegisterNative(com/inno/yodasdk/utils/NativeUtils, xNonce([B)[B, RX@0x40009739[libyoda.so]0x9739)
[07:49:26 580] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e47c[libyoda.so]0x1e47c, size=10, encoding=UTF-8, ret=extraction
[07:49:26 580] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e487[libyoda.so]0x1e487, size=74, encoding=UTF-8, ret=(Ljava/lang/String;Ljava/lang/Str
[07:49:26 581] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$104:2338) - RegisterNatives dvmClass=class com/inno/yodasdk/utils/NativeUtils, name=extraction, signature=(Ljava/lang/Str
RegisterNative(com/inno/yodasdk/utils/NativeUtils, extraction(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;, RX@0x400097d9[libyoda.so]0x97d9)
[07:49:26 581] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$12:239) - DeleteLocalRef object=unidbg@0x2f9ac2d8
[07:49:26 581] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x40009969[libyoda.so]0x9969 finished sp=unidbg@0xbffff6f0, offset=4ms
[07:49:26 581] DEBUG [com.github.unidbg.linux.android.dvm.DalvikModule] (DalvikModule:36) - Call [libyoda.so]JNI_OnLoad finished: version=0x10006, offset=6ms

Process finished with exit code 0
```

日志中发现它是动态注册的JNI函数，使用这个地址call 函数。参数通过frida hook得到。这是最新的代码

```java
package com.lession5;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
import com.github.unidbg.arm.backend.Backend;
import com.github.unidbg.arm.backend.BlockHook;
import com.github.unidbg.arm.context.RegisterContext;
import com.github.unidbg.debugger.BreakPointCallback;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.SystemPropertyHook;
import com.github.unidbg.linux.android.SystemPropertyProvider;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.StringObject;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;
import com.github.unidbg.utils.Inspector;
import unicorn.Unicorn;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class yoda extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    yoda(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机
        vm = emulator.createDalvikVM(new File("unidbg-android/src/test/resources/qdd/qtt_new.apk"));
        // 设置是否打印相关调用细节
        vm.setVerbose(true);
        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/qdd/libyoda.so"), true);
        module = dm.getModule();
        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }

    public void callbulwark() {
        List<Object> list = new ArrayList<>(10);
        list.add(vm.getJNIEnv());
        list.add(0);
        String str1 = "
{\"screen_brightness\":\"82\",\"tk\":\"ACHaSnpgYUzPHTlyVie2s7LThdzXV_vhfZ40NzUxN
Dk1MDg5NTIyNQ\",\"cpu_model\":\"AArch64 Processor rev 3 (aarch64)
,8,2016000\",\"carrier\":\"46007\",\"instance\":\"com.inno.yodasdk.info.Infos@ac
3ab7c\",\"sim_state\":\"5\",\"sid\":\"56a91d6a-204d-48ea-b170-
4c5cd713e05e\",\"imei\":\"869593867257804\",\"gyro\":\"0.02,0.0,1.0\",\"manufact
urer\":\"HUAWEI\",\"screen_scale\":\"5.2\",\"android_id\":\"86ee835487a1f4e4\",\
"boot_time\":\"1626514060336\",\"volume\":\"4,5,5,11,6\",\"serial_number\":\"LNX
11WPJ5M627459\",\"bt_mac\":\"14:09:DC:99:DB:89\",\"wifi_mac\":\"08:40:f3:f6:9a:2
1\",\"mac\":\"14:09:dc:9b:1c:60\",\"cid\":\"47514950895225\",\"charge_state\":\"
2\",\"apps_count\":\"2,120\",\"package_name\":\"com.jifen.qukan\",\"ext\":\"
{\\\"author_id\\\":\\\"2328110\\\",\\\"content_id\\\":\\\"1624220959\\\",\\\"mem
ber_id\\\":\\\"1453484970\\\"}\",\"platform\":\"android\",\"sensor_count\":\"11\
",\"app_version\":\"3.10.48.000.0714.1521\",\"screen_size\":\"1080,1920,3.0\",\"
brand\":\"HUAWEI\",\"sdk_version\":\"1.0.7.210128\",\"wifi_name\":\"123\",\"os_v
ersion\":\"23\",\"hardware\":\"hi3635\",\"adb\":\"1\",\"scene\":\"qtt_article_re
adtimerreport\",\"model\":\"HUAWEI GRA-TL00\"}";
        StringObject stringObject1 = new StringObject(vm, str1);
        list.add(vm.addLocalObject(stringObject1));
        String str2 = "dubo";
        StringObject stringObject2 = new StringObject(vm, str2);
        list.add(vm.addLocalObject(stringObject2));
        String str3 = "1629280231";
        StringObject stringObject3 = new StringObject(vm, str3);
        list.add(vm.addLocalObject(stringObject3));

        Number number = module.callFunction(emulator, 0x8ff1, list.toArray())
[0];
        byte[] result = (byte[]) vm.getObject(number.intValue()).getValue();
        Inspector.inspect(result, "result");
    }

    public static void main(String[] args) {
        yoda demo = new yoda();
        demo.callbulwark();
    }
}
```

也可以尝试运行一下报错

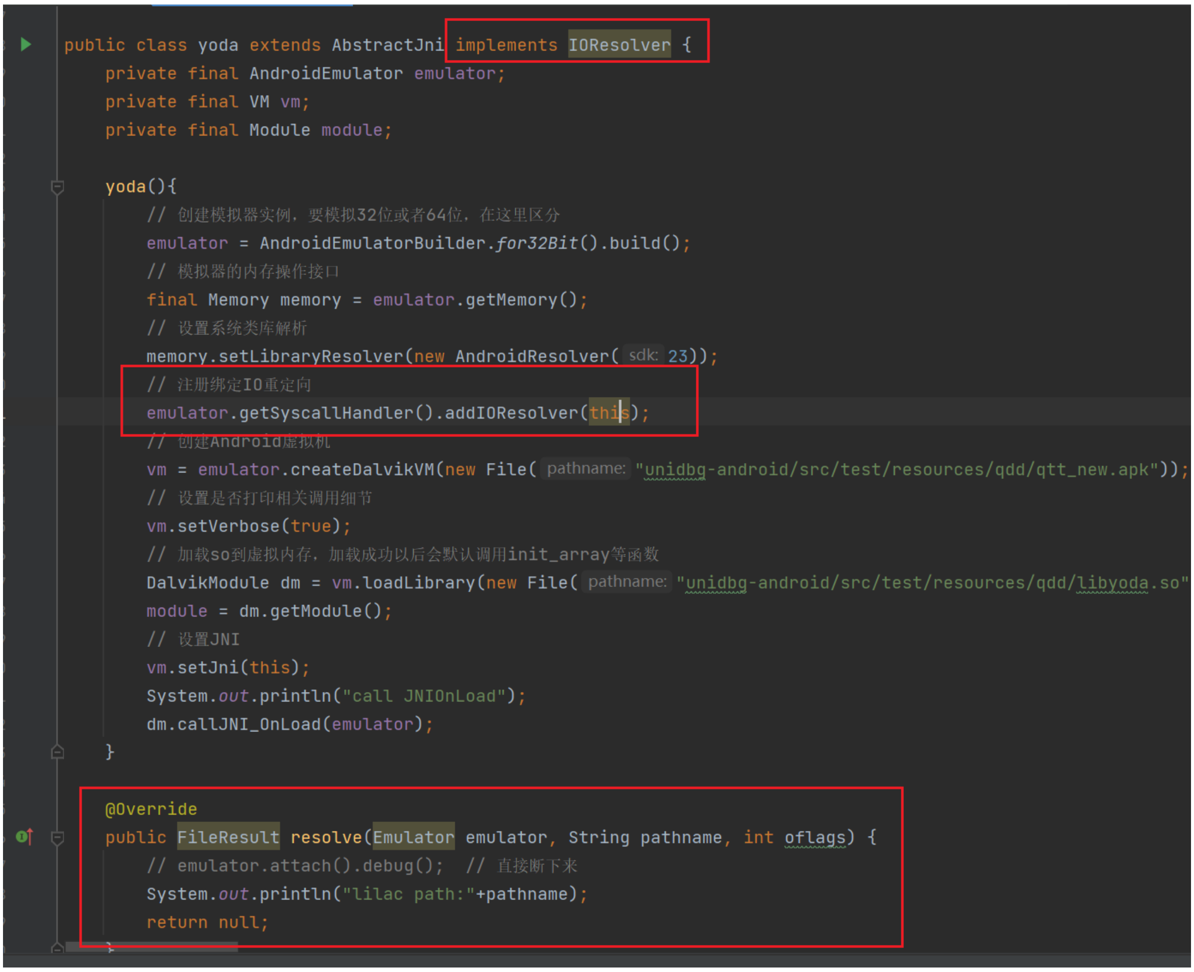


```
>>> r0=0x0 r1=0x0 r2=0x0 r3=0x4 r4=0x0 r5=0x4000fe4d r6=0x40203000 r7=0xbe r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021d000 ip=0xbffff680
>>> SP=0xbffff5a8 LR=RX@0x400f85cb[libc.so]0x2f5cb PC=RX@0x4010ab5c[libc.so]0x41b5c cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[07:52:56 416]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=190, svcNumber=0x0, PC=RX@0x4010ab5c[libc.so]0x41b5c, LR=RX@0x400f85c
>>> r0=0x4 r1=0x72 r2=0x0 r3=0x1 r4=0x0 r5=0x0 r6=0x40203000 r7=0x6 r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021d000 ip=0xbffff680
>>> SP=0xbffff5a0 LR=RX@0x400e2fe1[libc.so]0x19fe1 PC=RX@0x4010a264[libc.so]0x41264 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[07:52:56 417] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1964) - close fd=4
>>> r0=0x3 r1=0x1 r2=0x0 r3=0x1 r4=0x0 r5=0x0 r6=0x40203000 r7=0x166 r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021d000 ip=0xbffff680
>>> SP=0xbffff5a8 LR=RX@0x400f8635[libc.so]0x2f635 PC=RX@0x4010adb0[libc.so]0x41db0 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[07:52:56 417] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:2088) - dup3 oldfd=3, newfd=1, flags=0x0
[07:52:56 418]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=358, svcNumber=0x0, PC=RX@0x4010adb0[libc.so]0x41db0, LR=RX@0x400f863
java.lang.AbstractMethodError Create breakpoint : com.github.unidbg.linux.file.PipedWriteFileIO
    at com.github.unidbg.file.AbstractFileIO.dup2(AbstractFileIO.java:160)
    at com.github.unidbg.linux.ARM32SyscallHandler.dup3(ARM32SyscallHandler.java:2104)
    at com.github.unidbg.linux.ARM32SyscallHandler.hook(ARM32SyscallHandler.java:437)
    at com.github.unidbg.arm.backend.UnicornBackend$6.hook(UnicornBackend.java:299)
    at unicorn.Unicorn$NewHook.onInterrupt(Unicorn.java:128)
    at unicorn.Unicorn.emu_start(Native Method)
    at com.github.unidbg.arm.backend.UnicornBackend.emu_start(UnicornBackend.java:325)
    at com.github.unidbg.AbstractEmulator.emulate(AbstractEmulator.java:370)
    at com.github.unidbg.AbstractEmulator.eFunc(AbstractEmulator.java:446)
    at com.github.unidbg.arm.AbstractARMEmulator.eFunc(AbstractARMEmulator.java:220)
    at com.github.unidbg.Module.emulateFunction(Module.java:158)
    at com.github.unidbg.linux.LinuxModule.callFunction(LinuxModule.java:232)
    at com.lession5.yoda.callbulwark(yoda.java:65)
    at com.lession5.yoda.main(yoda.java:72)
```

报错了，报了一个平平无奇的错。接下来我们展示如何初始化Hook一些系统属性函数，当你认为样本可能会大量获取系统属性的时候，就可以像我如下这么做，它有两个好处

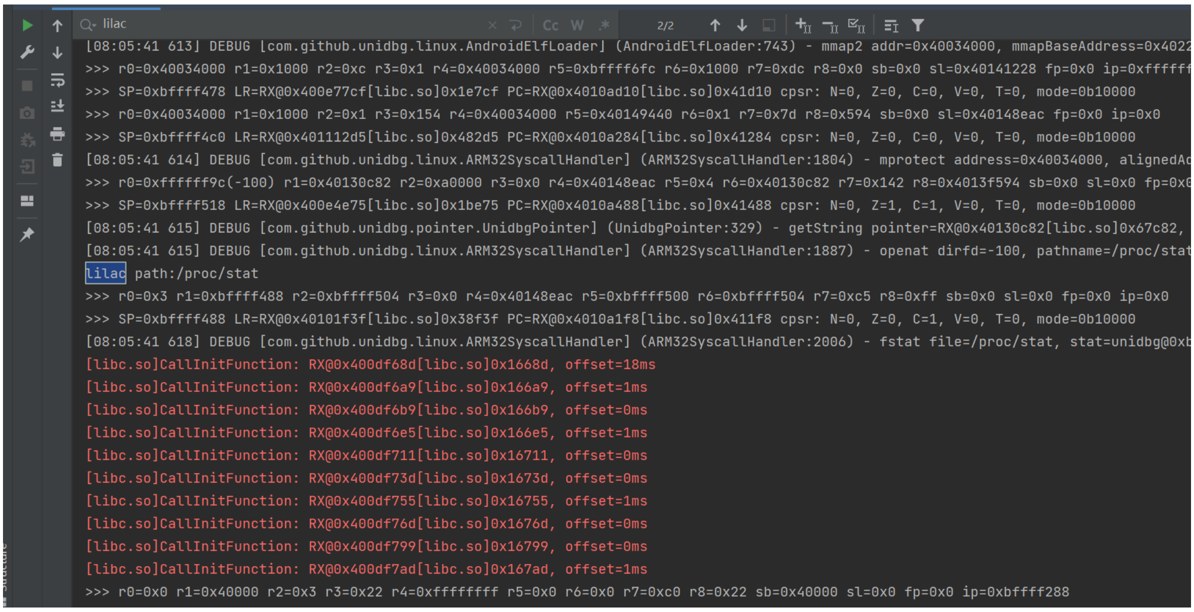
- 如果样本使用了它们，能被我们迅速发现。
- 如果这些API调用过程中出了错误，报错结果可能很晦涩，典型就是popen，但我们主动Hook这些函数后，就会提前意识到是哪个函数出了问题。

首先是文件访问，我们实现文件重定向接口，打上自己的日志，甚至可以像我注释里那样直接断下来。千万别忘了绑定文件重定向，没有这一句代码重定向就不会生效。



```java
public class yoda extends AbstractJni implements IOResolver {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    yoda(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver( sdk: 23));
        // 注册绑定IO重定向
        emulator.getSyscallHandler().addIOResolver(this);
        // 创建Android虚拟机
        vm = emulator.createDalvikVM(new File( pathname: "unidbg-android/src/test/resources/qdd/qtt_new.apk"));
        // 设置是否打印相关调用细节
        vm.setVerbose(true);
        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File( pathname: "unidbg-android/src/test/resources/qdd/libyoda.so"
        module = dm.getModule();
        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }

    @Override
    public FileResult resolve(Emulator emulator, String pathname, int oflags) {
        // emulator.attach().debug();  // 直接断下来
        System.out.println("lilac path:"+pathname);
        return null;
    }
```

运行代码，发现有两处调用



```
[08:05:41 613] DEBUG [com.github.unidbg.linux.AndroidElfLoader] (AndroidElfLoader:743) - mmap2 addr=0x40034000, mmapBaseAddress=0x4022
>>> r0=0x40034000 r1=0x1000 r2=0xc r3=0x1 r4=0x40034000 r5=0xbffff6fc r6=0x1000 r7=0xdc r8=0x0 sb=0x0 sl=0x40141228 fp=0x0 ip=0xfffff
>>> SP=0xbffff478 LR=RX@0x400e77cf[libc.so]0x1e7cf PC=RX@0x4010ad10[libc.so]0x41d10 cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
>>> r0=0x40034000 r1=0x1000 r2=0x1 r3=0x154 r4=0x40034000 r5=0x40149440 r6=0x1 r7=0x7d r8=0x594 sb=0x0 sl=0x40148eac fp=0x0 ip=0x0
>>> SP=0xbffff4c0 LR=RX@0x401112d5[libc.so]0x482d5 PC=RX@0x4010a284[libc.so]0x41284 cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
[08:05:41 614] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1804) - mprotect address=0x40034000, alignedAc
>>> r0=0xffffff9c(-100) r1=0x40130c82 r2=0xa0000 r3=0x0 r4=0x40148eac r5=0x4 r6=0x40130c82 r7=0x142 r8=0x4013f594 sb=0x0 sl=0x0 fp=0x0
>>> SP=0xbffff518 LR=RX@0x400e4e75[libc.so]0x1be75 PC=RX@0x4010a488[libc.so]0x41488 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:05:41 615] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x40130c82[libc.so]0x67c82,
[08:05:41 615] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1887) - openat dirfd=-100, pathname=/proc/stat
lilac path:/proc/stat
>>> r0=0x3 r1=0xbffff488 r2=0xbffff504 r3=0x0 r4=0x40148eac r5=0xbffff500 r6=0xbffff504 r7=0xc5 r8=0xff sb=0x0 sl=0x0 fp=0x0 ip=0x0
>>> SP=0xbffff488 LR=RX@0x40101f3f[libc.so]0x38f3f PC=RX@0x4010a1f8[libc.so]0x411f8 cpsr: N=0, Z=0, C=1, V=0, T=0, mode=0b10000
[08:05:41 618] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:2006) - fstat file=/proc/stat, stat=unidbg@0xb
[libc.so]CallInitFunction: RX@0x400df68d[libc.so]0x1668d, offset=18ms
[libc.so]CallInitFunction: RX@0x400df6a9[libc.so]0x166a9, offset=1ms
[libc.so]CallInitFunction: RX@0x400df6b9[libc.so]0x166b9, offset=0ms
[libc.so]CallInitFunction: RX@0x400df6e5[libc.so]0x166e5, offset=1ms
[libc.so]CallInitFunction: RX@0x400df711[libc.so]0x16711, offset=0ms
[libc.so]CallInitFunction: RX@0x400df73d[libc.so]0x1673d, offset=0ms
[libc.so]CallInitFunction: RX@0x400df755[libc.so]0x16755, offset=1ms
[libc.so]CallInitFunction: RX@0x400df76d[libc.so]0x1676d, offset=0ms
[libc.so]CallInitFunction: RX@0x400df799[libc.so]0x16799, offset=0ms
[libc.so]CallInitFunction: RX@0x400df7ad[libc.so]0x167ad, offset=1ms
>>> r0=0x0 r1=0x40000 r2=0x3 r3=0x22 r4=0xffffffff r5=0x0 r6=0x0 r7=0xc0 r8=0x22 sb=0x40000 sl=0x0 fp=0x0 ip=0xbffff288
```

> lilac path:/dev/**properties**
>
> lilac path:/proc/stat

需要注意的是，这前两个文件访问，并不需要我们管，这是libc初始化的内部逻辑，与样本无关。

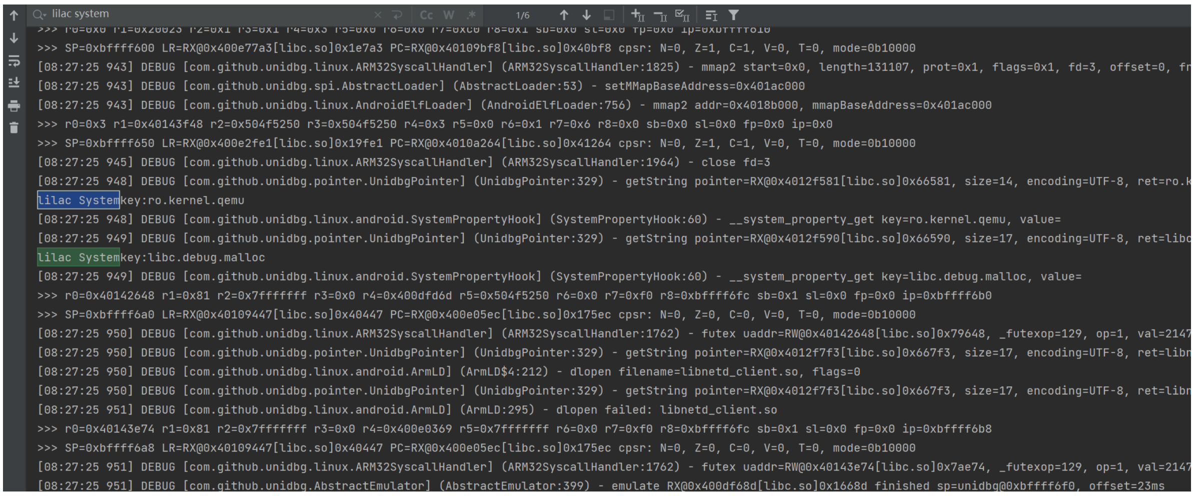
文件访问处理好了，接下来是__system_property_get 这个函数的处理，该函数在libc里，实现比较复杂，但使用上又很频繁。因此Unidbg 在src/main/java/com/github/unidbg/linux/android 目录下有相关类对它进行了Hook和封装，我们可以直接拿来用

```
SystemPropertyHook systemPropertyHook = new SystemPropertyHook(emulator);
systemPropertyHook.setPropertyProvider(new SystemPropertyProvider() {
    @Override
    public String getProperty(String key) {
        System.out.println("lilac Systemkey:"+key);
        switch (key){

        }
        return "";
    };
});
memory.addHookListener(systemPropertyHook);
```

看一下完整代码

```
package com.lession5;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
import com.github.unidbg.arm.backend.Backend;
import com.github.unidbg.arm.backend.BlockHook;
import com.github.unidbg.arm.context.RegisterContext;
import com.github.unidbg.debugger.BreakPointCallback;
import com.github.unidbg.file.FileResult;
import com.github.unidbg.file.IOResolver;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.SystemPropertyHook;
import com.github.unidbg.linux.android.SystemPropertyProvider;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.StringObject;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;
import com.github.unidbg.utils.Inspector;
import unicorn.Unicorn;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class yoda extends AbstractJni implements IOResolver {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    yoda(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 注册绑定IO重定向
        emulator.getSyscallHandler().addIOResolver(this);
        SystemPropertyHook systemPropertyHook = new
SystemPropertyHook(emulator);
        systemPropertyHook.setPropertyProvider(new SystemPropertyProvider() {
            @Override
            public String getProperty(String key) {
                System.out.println("lilac Systemkey:"+key);
                switch (key){

                }
                return "";
            };
        });
        memory.addHookListener(systemPropertyHook);
        // 创建Android虚拟机
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/qdd/qtt_new.apk"));
        // 设置是否打印相关调用细节
        vm.setVerbose(true);
        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/qdd/libyoda.so"), true);
        module = dm.getModule();
        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }

    @Override
    public FileResult resolve(Emulator emulator, String pathname, int oflags) {
        // emulator.attach().debug();  // 直接断下来
        System.out.println("lilac path:"+pathname);
        return null;
    }

    public void callbulwark() {
        List<Object> list = new ArrayList<>(10);
        list.add(vm.getJNIEnv());
        list.add(0);
        String str1 = "
{\"screen_brightness\":\"82\",\"tk\":\"ACHaSnpgYUzPHTlyVie2s7LThdzXV_vhfZ40NzUxN
Dk1MDg5NTIyNQ\",\"cpu_model\":\"AArch64 Processor rev 3 (aarch64)
,8,2016000\",\"carrier\":\"46007\",\"instance\":\"com.inno.yodasdk.info.Infos@ac
3ab7c\",\"sim_state\":\"5\",\"sid\":\"56a91d6a-204d-48ea-b170-
4c5cd713e05e\",\"imei\":\"869593867257804\",\"gyro\":\"0.02,0.0,1.0\",\"manufact
urer\":\"HUAWEI\",\"screen_scale\":\"5.2\",\"android_id\":\"86ee835487a1f4e4\",\
"boot_time\":\"1626514060336\",\"volume\":\"4,5,5,11,6\",\"serial_number\":\"LNX
11WPJ5M627459\",\"bt_mac\":\"14:09:DC:99:DB:89\",\"wifi_mac\":\"08:40:f3:f6:9a:2
1\",\"mac\":\"14:09:dc:9b:1c:60\",\"cid\":\"47514950895225\",\"charge_state\":\"
2\",\"apps_count\":\"2,120\",\"package_name\":\"com.jifen.qukan\",\"ext\":\"
{\\\"author_id\\\":\\\"2328110\\\",\\\"content_id\\\":\\\"1624220959\\\",\\\"mem
ber_id\\\":\\\"1453484970\\\"}\",\"platform\":\"android\",\"sensor_count\":\"11\
",\"app_version\":\"3.10.48.000.0714.1521\",\"screen_size\":\"1080,1920,3.0\",\"
brand\":\"HUAWEI\",\"sdk_version\":\"1.0.7.210128\",\"wifi_name\":\"123\",\"os_v
ersion\":\"23\",\"hardware\":\"hi3635\",\"adb\":\"1\",\"scene\":\"qtt_article_re
adtimerreport\",\"model\":\"HUAWEI GRA-TL00\"}";
        StringObject stringObject1 = new StringObject(vm, str1);
        list.add(vm.addLocalObject(stringObject1));
        String str2 = "dubo";
        StringObject stringObject2 = new StringObject(vm, str2);
        list.add(vm.addLocalObject(stringObject2));
        String str3 = "1629280231";
        StringObject stringObject3 = new StringObject(vm, str3);
        list.add(vm.addLocalObject(stringObject3));

        Number number = module.callFunction(emulator, 0x8ff1, list.toArray())
[0];
        byte[] result = (byte[]) vm.getObject(number.intValue()).getValue();
        Inspector.inspect(result, "result");
    }

    public static void main(String[] args) {
        yoda demo = new yoda();
        demo.callbulwark();
    }

}
```

运行测试一下，一共有六处访问



```
>>> r0=0x0 r1=0x20023 r2=0x1 r3=0x1 r4=0x3 r5=0x0 r6=0x0 r7=0xc0 r8=0x1 sb=0x0 sl=0x0 fp=0x0 ip=0xbffff610
>>> SP=0xbffff600 LR=RX@0x400e77a3[libc.so]0x1e7a3 PC=RX@0x40109bf8[libc.so]0x40bf8 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:27:25 943] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1825) - mmap2 start=0x0, length=131107, prot=0x1, flags=0x1, fd=3, offset=0, fr
[08:27:25 943] DEBUG [com.github.unidbg.spi.AbstractLoader] (AbstractLoader:53) - setMMapBaseAddress=0x401ac000
[08:27:25 943] DEBUG [com.github.unidbg.linux.AndroidElfLoader] (AndroidElfLoader:756) - mmap2 addr=0x4018b000, mmapBaseAddress=0x401ac000
>>> r0=0x3 r1=0x40143f48 r2=0x504f5250 r3=0x504f5250 r4=0x3 r5=0x0 r6=0x1 r7=0x6 r8=0x0 sb=0x0 sl=0x0 fp=0x0 ip=0x0
>>> SP=0xbffff650 LR=RX@0x400e2fe1[libc.so]0x19fe1 PC=RX@0x4010a264[libc.so]0x41264 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:27:25 945] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1964) - close fd=3
[08:27:25 948] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4012f581[libc.so]0x66581, size=14, encoding=UTF-8, ret=ro.k
lilac Systemkey:ro.kernel.qemu
[08:27:25 948] DEBUG [com.github.unidbg.linux.android.SystemPropertyHook] (SystemPropertyHook:60) - __system_property_get key=ro.kernel.qemu, value=
[08:27:25 949] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4012f590[libc.so]0x66590, size=17, encoding=UTF-8, ret=libc
lilac Systemkey:libc.debug.malloc
[08:27:25 949] DEBUG [com.github.unidbg.linux.android.SystemPropertyHook] (SystemPropertyHook:60) - __system_property_get key=libc.debug.malloc, value=
>>> r0=0x40142648 r1=0x81 r2=0x7fffffff r3=0x0 r4=0x400dfd6d r5=0x504f5250 r6=0x0 r7=0xf0 r8=0xbffff6fc sb=0x1 sl=0x0 fp=0x0 ip=0xbffff6b0
>>> SP=0xbffff6a0 LR=RX@0x40109447[libc.so]0x40447 PC=RX@0x400e05ec[libc.so]0x175ec cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
[08:27:25 950] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1762) - futex uaddr=RW@0x40142648[libc.so]0x79648, _futexop=129, op=1, val=2147
[08:27:25 950] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4012f7f3[libc.so]0x667f3, size=17, encoding=UTF-8, ret=libn
[08:27:25 950] DEBUG [com.github.unidbg.linux.android.ArmLD] (ArmLD$4:212) - dlopen filename=libnetd_client.so, flags=0
[08:27:25 950] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4012f7f3[libc.so]0x667f3, size=17, encoding=UTF-8, ret=libn
[08:27:25 951] DEBUG [com.github.unidbg.linux.android.ArmLD] (ArmLD:295) - dlopen failed: libnetd_client.so
>>> r0=0x40143e74 r1=0x81 r2=0x7fffffff r3=0x0 r4=0x400e0369 r5=0x7fffffff r6=0x0 r7=0xf0 r8=0xbffff6fc sb=0x1 sl=0x0 fp=0x0 ip=0xbffff6b8
>>> SP=0xbffff6a8 LR=RX@0x40109447[libc.so]0x40447 PC=RX@0x400e05ec[libc.so]0x175ec cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
[08:27:25 951] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1762) - futex uaddr=RW@0x40143e74[libc.so]0x7ae74, _futexop=129, op=1, val=2147
[08:27:25 951] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x400df68d[libc.so]0x1668d finished sp=unidbg@0xbffff6f0, offset=23ms
```

```
lilac Systemkey:ro.kernel.qemu
lilac Systemkey:libc.debug.malloc
lilac Systemkey:ro.serialno
lilac Systemkey:ro.product.manufacturer
lilac Systemkey:ro.product.brand
lilac Systemkey:ro.product.model
```

这里同样需要注意，系统属性获取的前两次不需要我们管，也是libc里的初始化，只需要记住就行了= =。

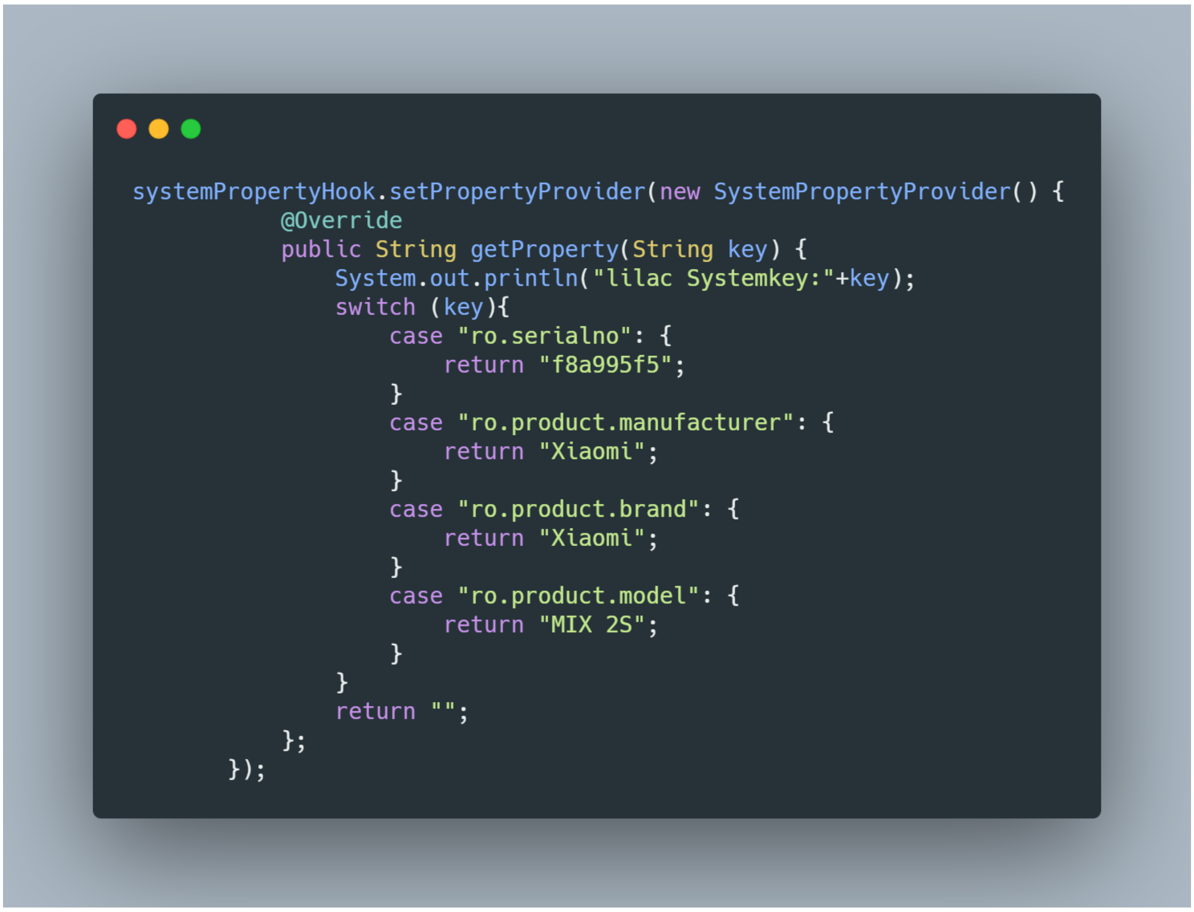
所以我们需要考虑的是这些

```
lilac Systemkey:ro.serialno
lilac Systemkey:ro.product.manufacturer
lilac Systemkey:ro.product.brand
lilac Systemkey:ro.product.model
```

通过adb shell 获取这些信息，一一填入正确的值，建议使用Unidbg时，对应的测试机Android版本为6.0，这样或许可以避免潜在的麻烦。

```
polaris:/ $ su
polaris:/ # getprop ro.serialno
f8a995f5
polaris:/ # getprop ro.product.manufacturer
Xiaomi
polaris:/ # getprop ro.product.brand
Xiaomi
polaris:/ # getprop ro.product.model
MIX 2S
polaris:/ #
```

填入switch语句中去



```
systemPropertyHook.setPropertyProvider(new SystemPropertyProvider() {
           @Override
           public String getProperty(String key) {
               System.out.println("lilac Systemkey:"+key);
               switch (key){
                   case "ro.serialno": {
                       return "f8a995f5";
                   }
                   case "ro.product.manufacturer": {
                       return "Xiaomi";
                   }
                   case "ro.product.brand": {
                       return "Xiaomi";
                   }
                   case "ro.product.model": {
                       return "MIX 2S";
                   }
               }
               return "";
           };
       });
```

接着我们要管popen和getenv，它俩都是libc里的函数，所以放一起说。我的想法是Hook这两个函数，如果产生调用就打印日志

我们先前介绍过很多种Unidbg Hook方案了，总体分成两派，以HookZz为代表的Hook框架，以及基于Unidbg原生Hook封装的各种Hook。我们这里选择后者，在大多数情况下我都更建议后者，因为复杂度更低，更不容易出BUG。

看一下代码

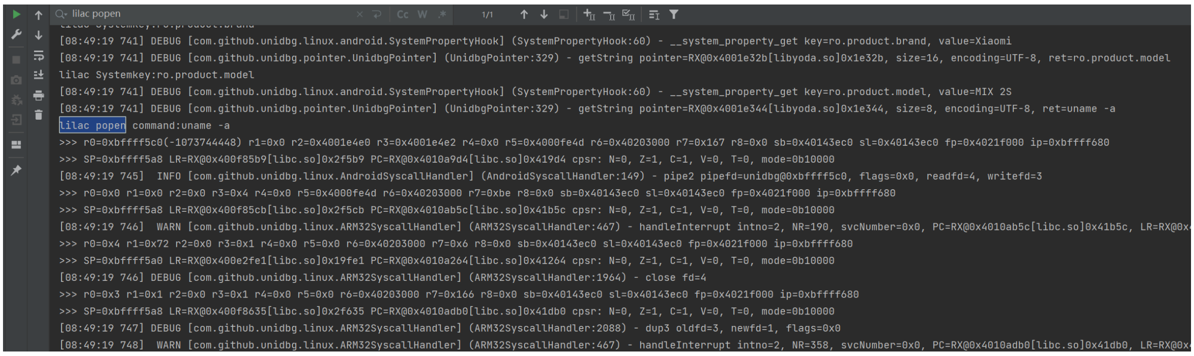
```
// HOOK popen
int popenAddress = (int) module.findSymbolByName("popen").getAddress();
// 函数原型：FILE *popen(const char *command, const char *type);
emulator.attach().addBreakPoint(popenAddress, new BreakPointCallback() {
    @Override
    public boolean onHit(Emulator<?> emulator, long address) {
        RegisterContext registerContext = emulator.getContext();
        String command = registerContext.getPointerArg(0).getString(0);
        System.out.println("lilac popen command:"+command);
        return true;
    }
});
```

addBreakPoint 我们一般用于下断点，添加回调，在命中断点时打印输出popen的参数1(即传给shell的命令)，并设置返回值为true，即做完打印程序继续跑，不用真断下来。

看一下完整代码

```
package com.lession5;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
import com.github.unidbg.arm.backend.Backend;
import com.github.unidbg.arm.backend.BlockHook;
import com.github.unidbg.arm.context.RegisterContext;
import com.github.unidbg.debugger.BreakPointCallback;
import com.github.unidbg.file.FileResult;
import com.github.unidbg.file.IOResolver;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.SystemPropertyHook;
import com.github.unidbg.linux.android.SystemPropertyProvider;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.StringObject;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;
import com.github.unidbg.utils.Inspector;
import unicorn.Unicorn;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class yoda extends AbstractJni implements IOResolver {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    yoda(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 注册绑定IO重定向
        emulator.getSyscallHandler().addIOResolver(this);
        SystemPropertyHook systemPropertyHook = new
SystemPropertyHook(emulator);
        systemPropertyHook.setPropertyProvider(new SystemPropertyProvider() {
            @Override
            public String getProperty(String key) {
                System.out.println("lilac Systemkey:"+key);
                switch (key){
                    case "ro.serialno": {
                        return "f8a995f5";
                    }
                    case "ro.product.manufacturer": {
                        return "Xiaomi";
                    }
                    case "ro.product.brand": {
                        return "Xiaomi";
                    }
                    case "ro.product.model": {
                        return "MIX 2S";
                    }
                }
                return "";
            };
        });
        memory.addHookListener(systemPropertyHook);
        // 创建Android虚拟机
        vm = emulator.createDalvikVM(new File("unidbg-android/src/test/resources/qdd/qtt_new.apk"));
        // 设置是否打印相关调用细节
        vm.setVerbose(true);
        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File("unidbg-android/src/test/resources/qdd/libyoda.so"), true);
        module = dm.getModule();

        // HOOK popen
        int popenAddress = (int) module.findSymbolByName("popen").getAddress();
        // 函数原型：FILE *popen(const char *command, const char *type);
        emulator.attach().addBreakPoint(popenAddress, new BreakPointCallback() {
            @Override
            public boolean onHit(Emulator<?> emulator, long address) {
                RegisterContext registerContext = emulator.getContext();
                String command = registerContext.getPointerArg(0).getString(0);
                System.out.println("lilac popen command:"+command);
                return true;
            }
        });


        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }

    @Override
    public FileResult resolve(Emulator emulator, String pathname, int oflags) {
        // emulator.attach().debug();  // 直接断下来
        System.out.println("lilac path:"+pathname);
        return null;
    }

    public void callbulwark() {
        List<Object> list = new ArrayList<>(10);
        list.add(vm.getJNIEnv());
        list.add(0);
        String str1 = "
{\"screen_brightness\":\"82\",\"tk\":\"ACHaSnpgYUzPHTlyVie2s7LThdzXV_vhfZ40NzUxN
Dk1MDg5NTIyNQ\",\"cpu_model\":\"AArch64 Processor rev 3 (aarch64)
,8,2016000\",\"carrier\":\"46007\",\"instance\":\"com.inno.yodasdk.info.Infos@ac
3ab7c\",\"sim_state\":\"5\",\"sid\":\"56a91d6a-204d-48ea-b170-
4c5cd713e05e\",\"imei\":\"869593867257804\",\"gyro\":\"0.02,0.0,1.0\",\"manufact
urer\":\"HUAWEI\",\"screen_scale\":\"5.2\",\"android_id\":\"86ee835487a1f4e4\",\
"boot_time\":\"1626514060336\",\"volume\":\"4,5,5,11,6\",\"serial_number\":\"LNX
11WPJ5M627459\",\"bt_mac\":\"14:09:DC:99:DB:89\",\"wifi_mac\":\"08:40:f3:f6:9a:2
1\",\"mac\":\"14:09:dc:9b:1c:60\",\"cid\":\"47514950895225\",\"charge_state\":\"
2\",\"apps_count\":\"2,120\",\"package_name\":\"com.jifen.qukan\",\"ext\":\"
{\\\"author_id\\\":\\\"2328110\\\",\\\"content_id\\\":\\\"1624220959\\\",\\\"mem
ber_id\\\":\\\"1453484970\\\"}\",\"platform\":\"android\",\"sensor_count\":\"11\
",\"app_version\":\"3.10.48.000.0714.1521\",\"screen_size\":\"1080,1920,3.0\",\"
brand\":\"HUAWEI\",\"sdk_version\":\"1.0.7.210128\",\"wifi_name\":\"123\",\"os_v
ersion\":\"23\",\"hardware\":\"hi3635\",\"adb\":\"1\",\"scene\":\"qtt_article_re
adtimerreport\",\"model\":\"HUAWEI GRA-TL00\"}";
        StringObject stringObject1 = new StringObject(vm, str1);
        list.add(vm.addLocalObject(stringObject1));
        String str2 = "dubo";
        StringObject stringObject2 = new StringObject(vm, str2);
        list.add(vm.addLocalObject(stringObject2));
        String str3 = "1629280231";
        StringObject stringObject3 = new StringObject(vm, str3);
        list.add(vm.addLocalObject(stringObject3));

        Number number = module.callFunction(emulator, 0x8ff1, list.toArray())
[0];
        byte[] result = (byte[]) vm.getObject(number.intValue()).getValue();
        Inspector.inspect(result, "result");
    }

    public static void main(String[] args) {
        yoda demo = new yoda();
        demo.callbulwark();
    }

}
```

运行测试一下



```
[08:49:19 741] DEBUG [com.github.unidbg.linux.android.SystemPropertyHook] (SystemPropertyHook:60) - __system_property_get key=ro.product.brand, value=Xiaomi
[08:49:19 741] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e32b[libyoda.so]0x1e32b, size=16, encoding=UTF-8, ret=ro.product.model
lilac Systemkey:ro.product.model
[08:49:19 741] DEBUG [com.github.unidbg.linux.android.SystemPropertyHook] (SystemPropertyHook:60) - __system_property_get key=ro.product.model, value=MIX 2S
[08:49:19 741] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e344[libyoda.so]0x1e344, size=8, encoding=UTF-8, ret=uname -a
lilac popen command:uname -a
>>> r0=0xbffff5c0(-1073744448) r1=0x0 r2=0x4001e4e0 r3=0x4001e4e2 r4=0x0 r5=0x4000fe4d r6=0x40203000 r7=0x167 r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021f000 ip=0xbffff680
>>> SP=0xbffff5a8 LR=RX@0x400f85b9[libc.so]0x2f5b9 PC=RX@0x4010a9d4[libc.so]0x419d4 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:49:19 745]  INFO [com.github.unidbg.linux.AndroidSyscallHandler] (AndroidSyscallHandler:149) - pipe2 pipefd=unidbg@0xbffff5c0, flags=0x0, readfd=4, writefd=3
>>> r0=0x0 r1=0x0 r2=0x0 r3=0x4 r4=0x0 r5=0x4000fe4d r6=0x40203000 r7=0xbe r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021f000 ip=0xbffff680
>>> SP=0xbffff5a8 LR=RX@0x400f85cb[libc.so]0x2f5cb PC=RX@0x4010ab5c[libc.so]0x41b5c cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:49:19 746]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=190, svcNumber=0x0, PC=RX@0x4010ab5c[libc.so]0x41b5c, LR=RX@0x
>>> r0=0x4 r1=0x72 r2=0x0 r3=0x1 r4=0x0 r5=0x0 r6=0x40203000 r7=0x6 r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021f000 ip=0xbffff680
>>> SP=0xbffff5a0 LR=RX@0x400e2fe1[libc.so]0x19fe1 PC=RX@0x4010a264[libc.so]0x41264 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:49:19 746] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1964) - close fd=4
>>> r0=0x3 r1=0x1 r2=0x0 r3=0x1 r4=0x0 r5=0x0 r6=0x40203000 r7=0x166 r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021f000 ip=0xbffff680
>>> SP=0xbffff5a8 LR=RX@0x400f8635[libc.so]0x2f635 PC=RX@0x4010adb0[libc.so]0x41db0 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:49:19 747] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:2088) - dup3 oldfd=3, newfd=1, flags=0x0
[08:49:19 748]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=358, svcNumber=0x0, PC=RX@0x4010adb0[libc.so]0x41db0, LR=RX@0x
```

确实有效，通过它发现样本在通过popen执行uname -a命令

uname -a 返回的是一些系统信息，其实内容就是系统调用uname返回的那些。

```
polaris:/ # uname -a
Linux localhost 4.9.186-perf-gd3d6708 #1 SMP PREEMPT Wed Nov 4 01:05:59 CST 2020
aarch64
```

但接下来有两个大问题在等着我们，思考一下

- 我们的HOOK时机够早吗？这个样本的popen调用发生在目标函数中，如果发生在init中呢？
- 我们通过HOOK得到了其参数，那怎么给它返回正确的值呢？

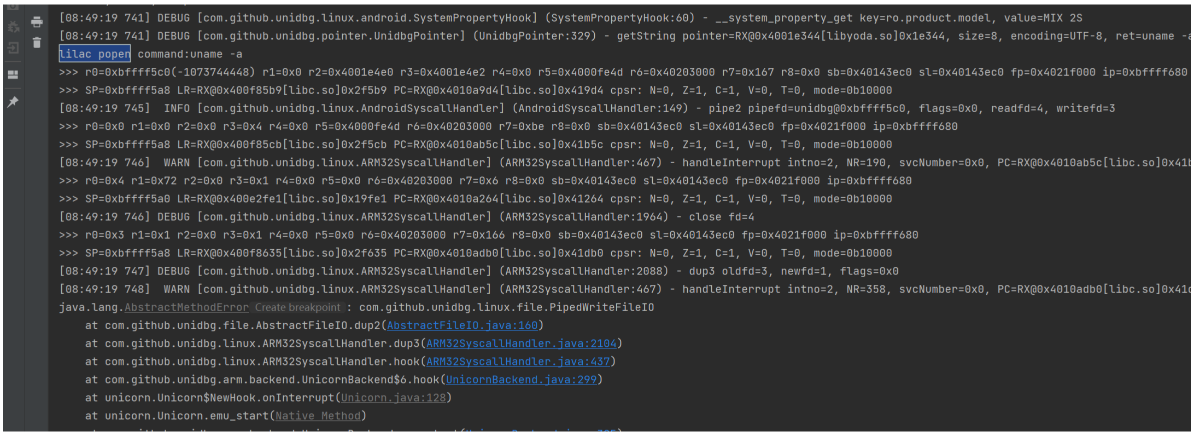
先考虑第一个问题，我们现在的HOOK时机是Loadlibrary之后，JNIOnLoad之前，如果SO存在init_proc函数，或者init_array非空，都会在Loadlibrary的过程中执行，我们的时机晚于这些初始化函数，这是绝对不能接受的。

我们需要在Loadlibrary前面开始Hook，为了实现这个目标，我们提前将libc加载进Unidbg内存中，看一下完整代码

```
package com.lession5;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
import com.github.unidbg.arm.backend.Backend;
import com.github.unidbg.arm.backend.BlockHook;
import com.github.unidbg.arm.context.RegisterContext;
import com.github.unidbg.debugger.BreakPointCallback;
import com.github.unidbg.file.FileResult;
import com.github.unidbg.file.IOResolver;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.SystemPropertyHook;
import com.github.unidbg.linux.android.SystemPropertyProvider;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.StringObject;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;
import com.github.unidbg.utils.Inspector;
import unicorn.Unicorn;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class yoda extends AbstractJni implements IOResolver {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    yoda(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 注册绑定IO重定向
        emulator.getSyscallHandler().addIOResolver(this);
        SystemPropertyHook systemPropertyHook = new
SystemPropertyHook(emulator);
        systemPropertyHook.setPropertyProvider(new SystemPropertyProvider() {
            @Override
            public String getProperty(String key) {
                System.out.println("lilac Systemkey:"+key);
                switch (key){
                    case "ro.serialno": {
                        return "f8a995f5";
                    }
                    case "ro.product.manufacturer": {
                        return "Xiaomi";
                    }
                    case "ro.product.brand": {
                        return "Xiaomi";
                    }
                    case "ro.product.model": {
                        return "MIX 2S";
                    }
                }
                return "";
            };
        });
        memory.addHookListener(systemPropertyHook);
        // 创建Android虚拟机
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/qdd/qtt_new.apk"));
        // 设置是否打印相关调用细节
        vm.setVerbose(true);

        DalvikModule dmLibc = vm.loadLibrary(new File("unidbg-
android/src/main/resources/android/sdk23/lib/libc.so"), true);
        Module moduleLibc = dmLibc.getModule();
        // HOOK popen
        int popenAddress = (int)
moduleLibc.findSymbolByName("popen").getAddress();
        // 函数原型：FILE *popen(const char *command, const char *type);
        emulator.attach().addBreakPoint(popenAddress, new BreakPointCallback() {
            @Override
            public boolean onHit(Emulator<?> emulator, long address) {
                RegisterContext registerContext = emulator.getContext();
                String command = registerContext.getPointerArg(0).getString(0);
                System.out.println("lilac popen command:"+command);
                return true;
            }
        });

        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/qdd/libyoda.so"), true);
        module = dm.getModule();

        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }
    @Override
    public FileResult resolve(Emulator emulator, String pathname, int oflags) {
        // emulator.attach().debug();  // 直接断下来
        System.out.println("lilac path:"+pathname);
        return null;
    }

    public void callbulwark() {
        List<Object> list = new ArrayList<>(10);
        list.add(vm.getJNIEnv());
        list.add(0);
        String str1 = "{\"screen_brightness\":\"82\",\"tk\":\"ACHaSnpgYUzPHTlyVie2s7LThdzXV_vhfZ40NzUxNDk1MDg5NTIyNQ\",\"cpu_model\":\"AArch64 Processor rev 3 (aarch64),8,2016000\",\"carrier\":\"46007\",\"instance\":\"com.inno.yodasdk.info.Infos@ac3ab7c\",\"sim_state\":\"5\",\"sid\":\"56a91d6a-204d-48ea-b170-4c5cd713e05e\",\"imei\":\"869593867257804\",\"gyro\":\"0.02,0.0,1.0\",\"manufacturer\":\"HUAWEI\",\"screen_scale\":\"5.2\",\"android_id\":\"86ee835487a1f4e4\",\"boot_time\":\"1626514060336\",\"volume\":\"4,5,5,11,6\",\"serial_number\":\"LNX11WPJ5M627459\",\"bt_mac\":\"14:09:DC:99:DB:89\",\"wifi_mac\":\"08:40:f3:f6:9a:21\",\"mac\":\"14:09:dc:9b:1c:60\",\"cid\":\"47514950895225\",\"charge_state\":\"2\",\"apps_count\":\"2,120\",\"package_name\":\"com.jifen.qukan\",\"ext\":\"{\\\"author_id\\\":\\\"2328110\\\",\\\"content_id\\\":\\\"1624220959\\\",\\\"member_id\\\":\\\"1453484970\\\"}\",\"platform\":\"android\",\"sensor_count\":\"11\",\"app_version\":\"3.10.48.000.0714.1521\",\"screen_size\":\"1080,1920,3.0\",\"brand\":\"HUAWEI\",\"sdk_version\":\"1.0.7.210128\",\"wifi_name\":\"123\",\"os_version\":\"23\",\"hardware\":\"hi3635\",\"adb\":\"1\",\"scene\":\"qtt_article_readtimerreport\",\"model\":\"HUAWEI GRA-TL00\"}";
        StringObject stringObject1 = new StringObject(vm, str1);
        list.add(vm.addLocalObject(stringObject1));
        String str2 = "dubo";
        StringObject stringObject2 = new StringObject(vm, str2);
        list.add(vm.addLocalObject(stringObject2));
        String str3 = "1629280231";
        StringObject stringObject3 = new StringObject(vm, str3);
        list.add(vm.addLocalObject(stringObject3));

        Number number = module.callFunction(emulator, 0x8ff1, list.toArray())[0];
        byte[] result = (byte[]) vm.getObject(number.intValue()).getValue();
        Inspector.inspect(result, "result");
    }

    public static void main(String[] args) {
        yoda demo = new yoda();
        demo.callbulwark();
    }

}
```

第一个问题就这么解决了，但它并非好无副作用，我们的样本SO的基地址就并非0x40000000了，做算法分析时需要注意一下。

第二个问题，怎么给它合适的返回值呢？



```
[08:49:19 741] DEBUG [com.github.unidbg.linux.android.SystemPropertyHook] (SystemPropertyHook:60) - __system_property_get key=ro.product.model, value=MIX 2S
[08:49:19 741] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4001e344[libyoda.so]0x1e344, size=8, encoding=UTF-8, ret=uname -a
lilac popen command:uname -a
>>> r0=0xbffff5c0(-1073744448) r1=0x0 r2=0x4001e4e0 r3=0x4001e4e2 r4=0x0 r5=0x4000fe4d r6=0x40203000 r7=0x167 r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021f000 ip=0xbffff680
>>> SP=0xbffff5a8 LR=RX@0x400f85b9[libc.so]0x2f5b9 PC=RX@0x4010a9d4[libc.so]0x419d4 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:49:19 745]  INFO [com.github.unidbg.linux.AndroidSyscallHandler] (AndroidSyscallHandler:149) - pipe2 pipefd=unidbg@0xbffff5c0, flags=0x0, readfd=4, writefd=3
>>> r0=0x0 r1=0x0 r2=0x0 r3=0x4 r4=0x0 r5=0x4000fe4d r6=0x40203000 r7=0xbe r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021f000 ip=0xbffff680
>>> SP=0xbffff5a8 LR=RX@0x400f85cb[libc.so]0x2f5cb PC=RX@0x4010ab5c[libc.so]0x41b5c cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:49:19 746]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=190, svcNumber=0x0, PC=RX@0x4010ab5c[libc.so]0x41b
>>> r0=0x4 r1=0x72 r2=0x0 r3=0x1 r4=0x0 r5=0x0 r6=0x40203000 r7=0x6 r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021f000 ip=0xbffff680
>>> SP=0xbffff5a0 LR=RX@0x400e2fe1[libc.so]0x19fe1 PC=RX@0x4010a264[libc.so]0x41264 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:49:19 746] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1964) - close fd=4
>>> r0=0x3 r1=0x1 r2=0x0 r3=0x1 r4=0x0 r5=0x0 r6=0x40203000 r7=0x166 r8=0x0 sb=0x40143ec0 sl=0x40143ec0 fp=0x4021f000 ip=0xbffff680
>>> SP=0xbffff5a8 LR=RX@0x400f8635[libc.so]0x2f635 PC=RX@0x4010adb0[libc.so]0x41db0 cpsr: N=0, Z=1, C=1, V=0, T=0, mode=0b10000
[08:49:19 747] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:2088) - dup3 oldfd=3, newfd=1, flags=0x0
[08:49:19 748]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=358, svcNumber=0x0, PC=RX@0x4010adb0[libc.so]0x41d
java.lang.AbstractMethodError Create breakpoint : com.github.unidbg.linux.file.PipedWriteFileIO
    at com.github.unidbg.file.AbstractFileIO.dup2(AbstractFileIO.java:160)
    at com.github.unidbg.linux.ARM32SyscallHandler.dup3(ARM32SyscallHandler.java:2104)
    at com.github.unidbg.linux.ARM32SyscallHandler.hook(ARM32SyscallHandler.java:437)
    at com.github.unidbg.arm.backend.UnicornBackend$6.hook(UnicornBackend.java:299)
    at unicorn.Unicorn$NewHook.onInterrupt(Unicorn.java:128)
    at unicorn.Unicorn.emu_start(Native Method)
```

可以发现，其实下面奇怪的报错就是popen导致的，这是因为popen内部实现较为复杂，调用了wait4、fork等Unidbg尚不能良好实现的系统调用。理论上我们有两条路可以走，1是既然内部实现复杂，那么直接Hook替换了这个函数，我们自己根据参数，返回合适的值，SystemPropertyHook不就是一个现成的例子吗？

可是还有点儿不太一样，SystemPropertyHook函数返回的是字符串，可以方便处理，而popen返回的是文件描述符，Unidbg中相关的实现在UnixSyscallHandler类中，使用起来好像有点儿不方便。

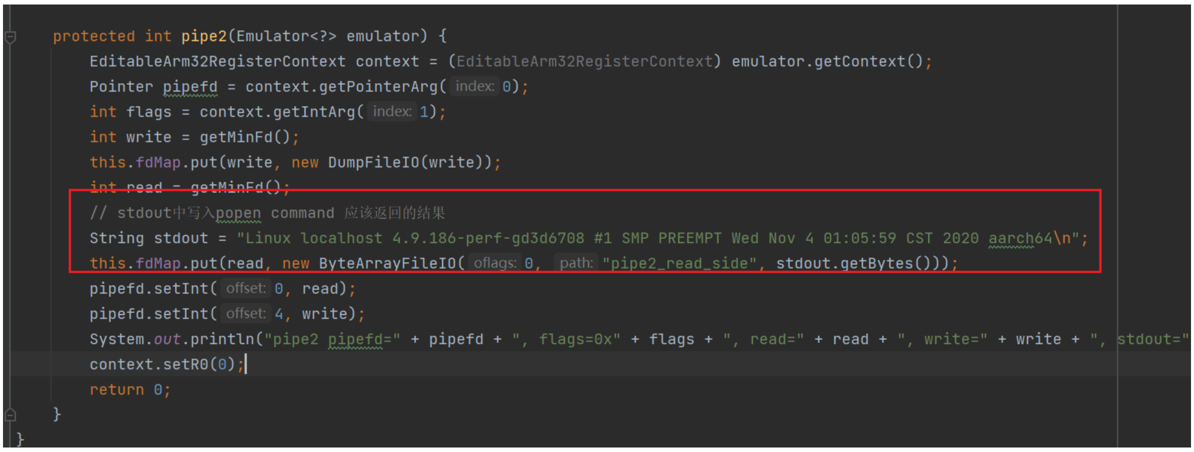
所以我们采用第二种方式，Unidbg提供了一种在底层修复和实现popen函数的法子。

首先我们要实现自己的ARM32SyscallHandler，完整代码如下，你可以把它当成固定讨论，它是针对popen报错的官方解决方案。

```java
package com.lession5;

import com.github.unidbg.Emulator;
import com.github.unidbg.arm.context.EditableArm32RegisterContext;
import com.github.unidbg.linux.ARM32SyscallHandler;
import com.github.unidbg.linux.file.ByteArrayFileIO;
import com.github.unidbg.linux.file.DumpFileIO;
import com.github.unidbg.memory.SvcMemory;
import com.sun.jna.Pointer;

import java.util.concurrent.ThreadLocalRandom;

class yodaSyscallHandler extends ARM32SyscallHandler {

    public yodaSyscallHandler(SvcMemory svcMemory) {
        super(svcMemory);
    }


    @Override
    protected boolean handleUnknownSyscall(Emulator emulator, int NR) {
        switch (NR) {
            case 190:
                vfork(emulator);
                return true;
            case 114:
                wait4(emulator);
                return true;
        }

        return super.handleUnknownSyscall(emulator, NR);
    }
    private void vfork(Emulator<?> emulator) {
        EditableArm32RegisterContext context = (EditableArm32RegisterContext) emulator.getContext();
        int childPid = emulator.getPid() + ThreadLocalRandom.current().nextInt(256);
        int r0 = childPid;
        System.out.println("vfork pid=" + r0);
        context.setR0(r0);
    }

    private void wait4(Emulator emulator) {
        EditableArm32RegisterContext context = (EditableArm32RegisterContext) emulator.getContext();
        int pid = context.getR0Int();
        Pointer wstatus = context.getR1Pointer();
        int options = context.getR2Int();
        Pointer rusage = context.getR3Pointer();
        System.out.println("wait4 pid=" + pid + ", wstatus=" + wstatus + ", options=0x" + Integer.toHexString(options) + ", rusage=" + rusage);
    }

    protected int pipe2(Emulator<?> emulator) {
        EditableArm32RegisterContext context = (EditableArm32RegisterContext) emulator.getContext();
        Pointer pipefd = context.getPointerArg(0);
        int flags = context.getIntArg(1);
        int write = getMinFd();
        this.fdMap.put(write, new DumpFileIO(write));
        int read = getMinFd();
        // stdout中写入popen command 应该返回的结果
        String stdout = "Linux localhost 4.9.186-perf-gd3d6708 #1 SMP PREEMPT Wed Nov 4 01:05:59 CST 2020 aarch64\n";
        this.fdMap.put(read, new ByteArrayFileIO(0, "pipe2_read_side", stdout.getBytes()));
        pipefd.setInt(0, read);
        pipefd.setInt(4, write);
        System.out.println("pipe2 pipefd=" + pipefd + ", flags=0x" + flags + ", read=" + read + ", write=" + write + ", stdout=" + stdout);
        context.setR0(0);
        return 0;
    }
}
```

解释一下为什么不直接补在ARM32SyscallHandler中？因为Unidbg并没有真正实现wait4和fork这两个系统调用，只不过对于popen而言，用上述方式可以“凑合用”，既然不是完美的实现，自然不能放到ARM32SyscallHandler中去，免得出大问题。

在pipe2中注释下的stdout中传入正确返回值即可，比如uname -a就是，需要注意，结果都要加换行符，这是shell结果的返回习惯。

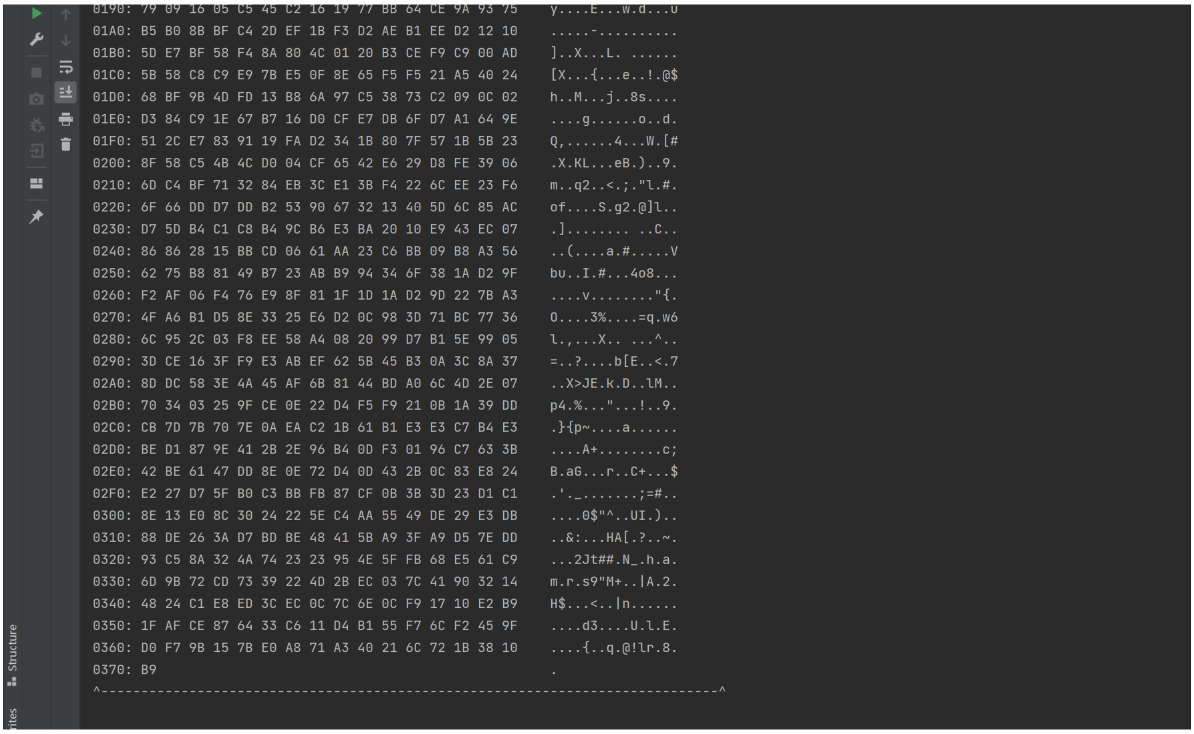


```
    protected int pipe2(Emulator<?> emulator) {
        EditableArm32RegisterContext context = (EditableArm32RegisterContext) emulator.getContext();
        Pointer pipefd = context.getPointerArg( index: 0);
        int flags = context.getIntArg( index: 1);
        int write = getMinFd();
        this.fdMap.put(write, new DumpFileIO(write));
        int read = getMinFd();
        // stdout中写入popen command 应该返回的结果
        String stdout = "Linux localhost 4.9.186-perf-gd3d6708 #1 SMP PREEMPT Wed Nov 4 01:05:59 CST 2020 aarch64\n";
        this.fdMap.put(read, new ByteArrayFileIO( oflags: 0,  path: "pipe2_read_side", stdout.getBytes()));
        pipefd.setInt( offset: 0, read);
        pipefd.setInt( offset: 4, write);
        System.out.println("pipe2 pipefd=" + pipefd + ", flags=0x" + flags + ", read=" + read + ", write=" + write + ", stdout="
        context.setR0(0);
        return 0;
    }
}
```

接下来让我们的emulator使用我们自己的syscallHandler，**emulator = new AndroidARMEmulator(new File("target/rootfs"));** 由如下洋洋洒洒十来行取代。

```
    AndroidEmulatorBuilder builder = new AndroidEmulatorBuilder(false) {
        public AndroidEmulator build() {
            return new AndroidARMEmulator(processName, rootDir,
                    backendFactories) {
                @Override
                protected UnixSyscallHandler<AndroidFileIO>
                createSyscallHandler(SvcMemory svcMemory) {
                    return new yodaSyscallHandler(svcMemory);
                }
            };
        }
    };

    emulator = builder.setRootDir(new File("target/rootfs")).build();
```

接下来运行代码



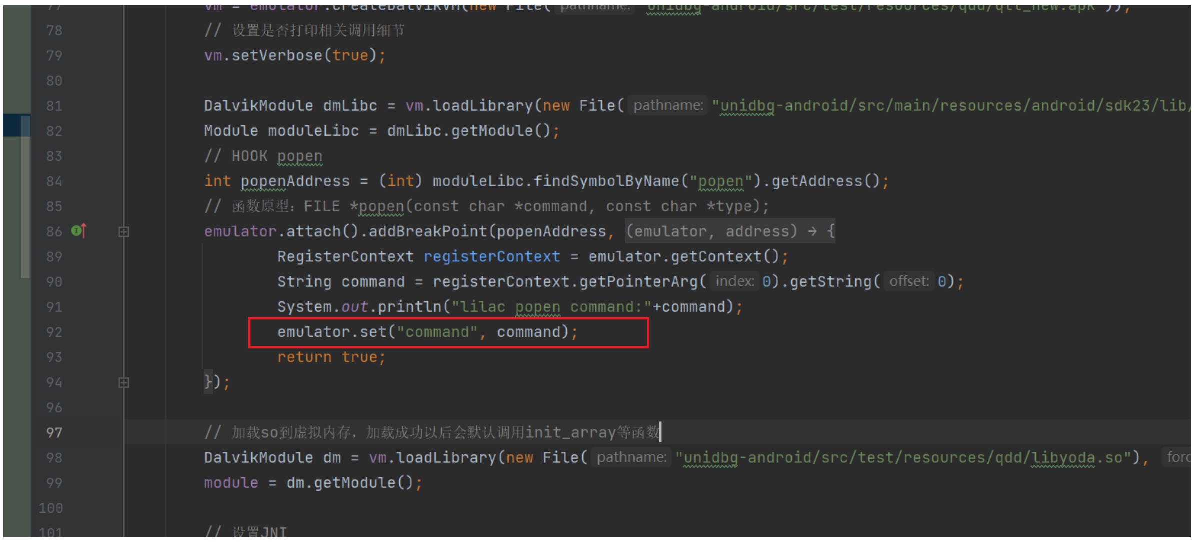
```
0190: 79 09 16 05 C5 45 C2 16 19 77 BB 64 CE 9A 93 75    y....E...w.d...u
01A0: B5 B0 8B BF C4 2D EF 1B F3 D2 AE B1 EE D2 12 10    .....-..........
01B0: 5D E7 BF 58 F4 8A 80 4C 01 20 B3 CE F9 C9 00 AD    ]..X...L. ......
01C0: 5B 58 C8 C9 E9 7B E5 0F 8E 65 F5 F5 21 A5 40 24    [X...{...e..!.@$
01D0: 68 BF 9B 4D FD 13 B8 6A 97 C5 38 73 C2 09 0C 02    h..M...j..8s....
01E0: D3 84 C9 1E 67 B7 16 D0 CF E7 DB 6F D7 A1 64 9E    ....g......o..d.
01F0: 51 2C E7 83 91 19 FA D2 34 1B 80 7F 57 1B 5B 23    Q,......4...W.[#
0200: 8F 58 C5 4B 4C D0 04 CF 65 42 E6 29 D8 FE 39 06    .X.KL...eB.)..9.
0210: 6D C4 BF 71 32 84 EB 3C E1 3B F4 22 6C EE 23 F6    m..q2..<.;."l.#.
0220: 6F 66 DD D7 DD B2 53 90 67 32 13 40 5D 6C 85 AC    of....S.g2.@]l..
0230: D7 5D B4 C1 C8 B4 9C B6 E3 BA 20 10 E9 43 EC 07    .]........ ..C..
0240: 86 86 28 15 BB CD 06 61 AA 23 C6 BB 09 B8 A3 56    ..(....a.#.....V
0250: 62 75 B8 81 49 B7 23 AB B9 94 34 6F 38 1A D2 9F    bu..I.#...4o8...
0260: F2 AF 06 F4 76 E9 8F 81 1F 1D 1A D2 9D 22 7B A3    ....v........"{.
0270: 4F A6 B1 D5 8E 33 25 E6 D2 0C 98 3D 71 BC 77 36    O....3%....=q.w6
0280: 6C 95 2C 03 F8 EE 58 A4 08 20 99 D7 B1 5E 99 05    l.,...X.. ...^..
0290: 3D CE 16 3F F9 E3 AB EF 62 5B 45 B3 0A 3C 8A 37    =..?....b[E..<.7
02A0: 8D DC 58 3E 4A 45 AF 6B 81 44 BD A0 6C 4D 2E 07    ..X>JE.k.D..lM..
02B0: 70 34 03 25 9F CE 0E 22 D4 F5 F9 21 0B 1A 39 DD    p4.%..."...!..9.
02C0: CB 7D 7B 70 7E 0A EA C2 1B 61 B1 E3 E3 C7 B4 E3    .}{p~....a......
02D0: BE D1 87 9E 41 2B 2E 96 B4 0D F3 01 96 C7 63 3B    ....A+........c;
02E0: 42 BE 61 47 DD 8E 0E 72 D4 0D 43 2B 0C 83 E8 24    B.aG...r..C+...$
02F0: E2 27 D7 5F B0 C3 BB FB 87 CF 0B 3B 3D 23 D1 C1    .'._.......;=#..
0300: 8E 13 E0 8C 30 24 22 5E C4 AA 55 49 DE 29 E3 DB    ....0$"^..UI.)..
0310: 88 DE 26 3A D7 BD BE 48 41 5B A9 3F A9 D5 7E DD    ..&:...HA[.?..~.
0320: 93 C5 8A 32 4A 74 23 23 95 4E 5F FB 68 E5 61 C9    ...2Jt##.N_.h.a.
0330: 6D 9B 72 CD 73 39 22 4D 2B EC 03 7C 41 90 32 14    m.r.s9"M+..|A.2.
0340: 48 24 C1 E8 ED 3C EC 0C 7C 6E 0C F9 17 10 E2 B9    H$...<..|n......
0350: 1F AF CE 87 64 33 C6 11 D4 B1 55 F7 6C F2 45 9F    ....d3....U.l.E.
0360: D0 F7 9B 15 7B E0 A8 71 A3 40 21 6C 72 1B 38 10    ....{..q.@!lr.8.
0370: B9                                                 .
^-------------------------------------------------------------------------------^
```

直接跑出了结果，但我们的任务其实还没有完成= =，tag中搜索lilac popen，发现一共调用了三次

```
lilac popen command:uname -a
lilac popen command:cd /system/bin && ls -l
lilac popen command:stat /root
```

问题来了，我们上面的代码，似乎只处理了uname -a应该返回的值，后面两次呢？怎么在pipe2中根据popen输入的command返回合适的输出呢？

我们可以使用emulator的全局变量来完成这一点



```
vm = emulator.createDalvikVM(new File( pathname: "unidbg-android/src/test/resources/qdd/qtt_new.apk"));
// 设置是否打印相关调用细节
vm.setVerbose(true);

DalvikModule dmLibc = vm.loadLibrary(new File( pathname: "unidbg-android/src/main/resources/android/sdk23/lib/
Module moduleLibc = dmLibc.getModule();
// HOOK popen
int popenAddress = (int) moduleLibc.findSymbolByName("popen").getAddress();
// 函数原型：FILE *popen(const char *command, const char *type);
emulator.attach().addBreakPoint(popenAddress, (emulator, address) → {
        RegisterContext registerContext = emulator.getContext();
        String command = registerContext.getPointerArg( index: 0).getString( offset: 0);
        System.out.println("lilac popen command:"+command);
        emulator.set("command", command);
        return true;
});

// 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
DalvikModule dm = vm.loadLibrary(new File( pathname: "unidbg-android/src/test/resources/qdd/libyoda.so"), forc
module = dm.getModule();

// 设置JNI
```

对应的yodaSyscallHandler代码，其中 *cd /system/bin && ls -l* 和 *stat /root* 的结果来自adb shell，大家根据自己的测试机情况填入合适的结果。

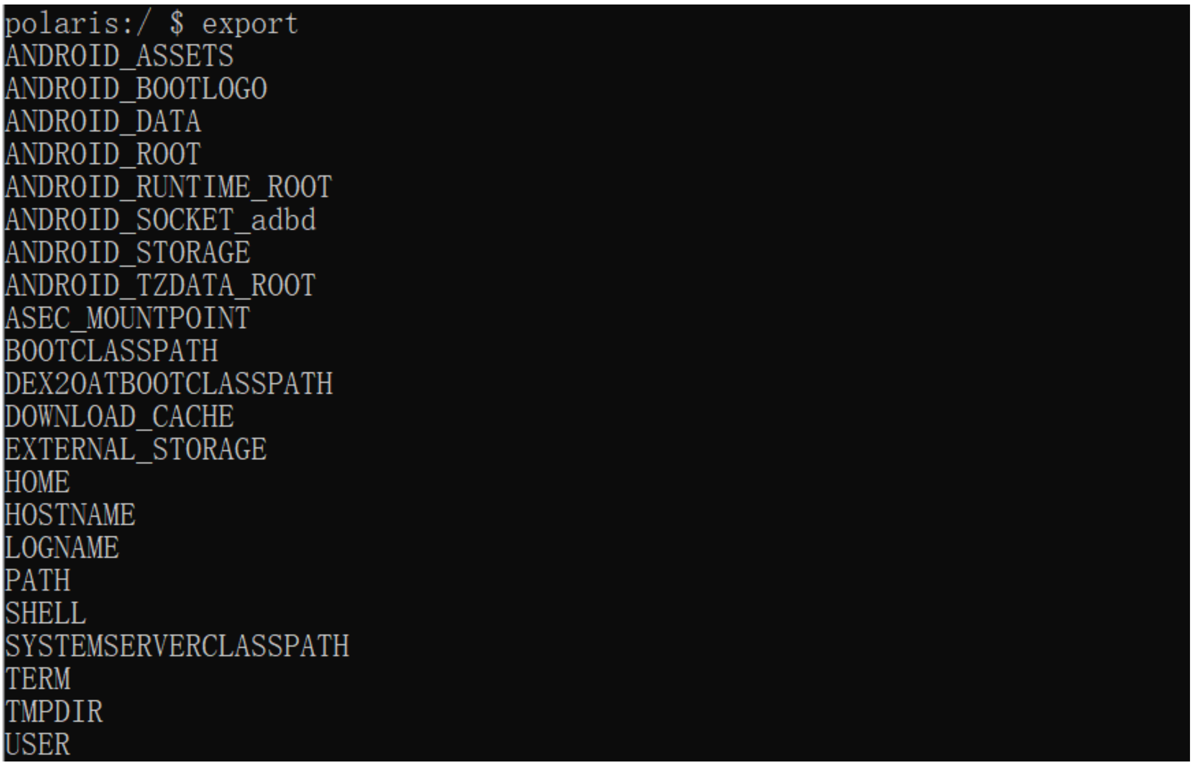
```
package com.lession5;

import com.github.unidbg.Emulator;
import com.github.unidbg.arm.context.EditableArm32RegisterContext;
import com.github.unidbg.linux.ARM32SyscallHandler;
import com.github.unidbg.linux.file.ByteArrayFileIO;
import com.github.unidbg.linux.file.DumpFileIO;
import com.github.unidbg.memory.SvcMemory;
import com.sun.jna.Pointer;

import java.util.concurrent.ThreadLocalRandom;

class yodaSyscallHandler extends ARM32SyscallHandler {

    public yodaSyscallHandler(SvcMemory svcMemory) {
        super(svcMemory);
    }


    @Override
    protected boolean handleUnknownSyscall(Emulator emulator, int NR) {
        switch (NR) {
            case 190:
                vfork(emulator);
                return true;
            case 114:
                wait4(emulator);
                return true;
        }
        return super.handleUnknownSyscall(emulator, NR);
    }

    private void vfork(Emulator<?> emulator) {
        EditableArm32RegisterContext context = (EditableArm32RegisterContext)
emulator.getContext();
        int childPid = emulator.getPid() +
ThreadLocalRandom.current().nextInt(256);
        int r0 = childPid;
        System.out.println("vfork pid=" + r0);
        context.setR0(r0);
    }

    private void wait4(Emulator emulator) {
        EditableArm32RegisterContext context = (EditableArm32RegisterContext)
emulator.getContext();
        int pid = context.getR0Int();
        Pointer wstatus = context.getR1Pointer();
        int options = context.getR2Int();
        Pointer rusage = context.getR3Pointer();
        System.out.println("wait4 pid=" + pid + ", wstatus=" + wstatus + ",
options=0x" + Integer.toHexString(options) + ", rusage=" + rusage);
    }

    protected int pipe2(Emulator<?> emulator) {
        EditableArm32RegisterContext context = (EditableArm32RegisterContext)
emulator.getContext();
        Pointer pipefd = context.getPointerArg(0);
        int flags = context.getIntArg(1);
        int write = getMinFd();
        this.fdMap.put(write, new DumpFileIO(write));
        int read = getMinFd();
        String stdout = "\n";
        // stdout中写入popen command 应该返回的结果
        String command = emulator.get("command");
        switch (command){
            case "uname -a":{
                stdout = "Linux localhost 4.9.186-perf-gd3d6708 #1 SMP PREEMPT
Wed Nov 4 01:05:59 CST 2020 aarch64\n";
            }
            break;
            case "cd /system/bin && ls -l":{
                stdout = "total 25152\n" +
                        "-rwxr-xr-x 1 root   shell     128688 2009-01-01 08:00
abb\n" +
                        "lrwxr-xr-x 1 root   shell          6 2009-01-01 08:00
acpi -> toybox\n" +
                        "-rwxr-xr-x 1 root   shell      30240 2009-01-01 08:00
adbd\n" +
                        "-rwxr-xr-x 1 root   shell        207 2009-01-01 08:00
am\n" +
                        "-rwxr-xr-x 1 root   shell     456104 2009-01-01 08:00
apexd\n" +
                        "lrwxr-xr-x 1 root   shell         13 2009-01-01 08:00
app_process -> app_process64\n" +
                        "-rwxr-xr-x 1 root   shell      25212 2009-01-01 08:00
app_process32\n"
```

```
            }
            break;
            case "stat /root":{
                stdout = "stat: '/root': No such file or directory\n";
            }
            break;
            default:
                System.out.println("command do not match!");
        }

        this.fdMap.put(read, new ByteArrayFileIO(0, "pipe2_read_side",
stdout.getBytes()));
        pipefd.setInt(0, read);
        pipefd.setInt(4, write);
        System.out.println("pipe2 pipefd=" + pipefd + ", flags=0x" + flags + ",
read=" + read + ", write=" + write + ", stdout=" + stdout);
        context.setR0(0);
        return 0;
    }
}
```

四、getenv的处理

getenv的出现频率也挺高，但因为这个样本没有用，我就没有讲 = =，我们单独立出这一节，通过demo来讲getenv。

首先我们看一下当前测试机有哪些环境变量



```
polaris:/ $ export
ANDROID_ASSETS
ANDROID_BOOTLOGO
ANDROID_DATA
ANDROID_ROOT
ANDROID_RUNTIME_ROOT
ANDROID_SOCKET_adbd
ANDROID_STORAGE
ANDROID_TZDATA_ROOT
ASEC_MOUNTPOINT
BOOTCLASSPATH
DEX2OATBOOTCLASSPATH
DOWNLOAD_CACHE
EXTERNAL_STORAGE
HOME
HOSTNAME
LOGNAME
PATH
SHELL
SYSTEMSERVERCLASSPATH
TERM
TMPDIR
USER
```

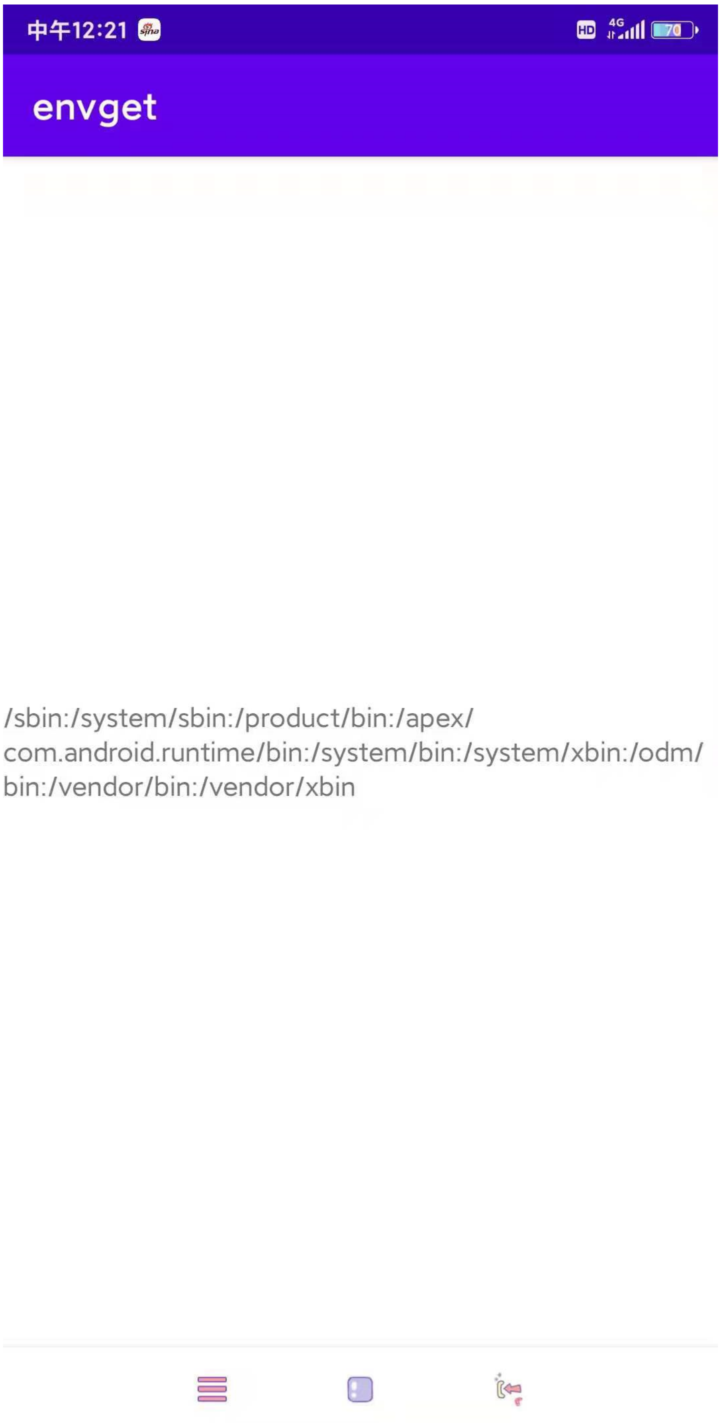
看一下PATH的内容

```
polaris:/ $ echo $PATH
/sbin:/system/sbin:/product/bin:/apex/com.android.runtime/bin:/system/bin:/syste
m/xbin:/odm/bin:/vendor/bin:/vendor/xbin
```

接下来写demo，在native中获取它

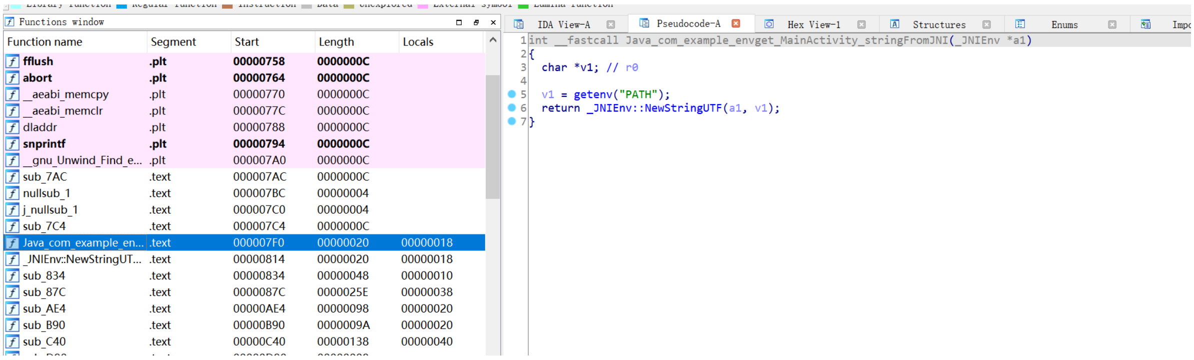
```
extern "C" JNIEXPORT jstring JNICALL
Java_com_example_envget_MainActivity_stringFromJNI(
        JNIEnv* env,
        jobject /* this */) {
    char *p;
    p = getenv("PATH");
    return env->NewStringUTF(p);
}
```

这是native中的代码，运行测试一下



```
/sbin:/system/sbin:/product/bin:/apex/com.android.runtime/bin:/system/bin:/system/xbin:/odm/bin:/vendor/bin:/vendor/xbin
```

接下来我们ida中看一下SO



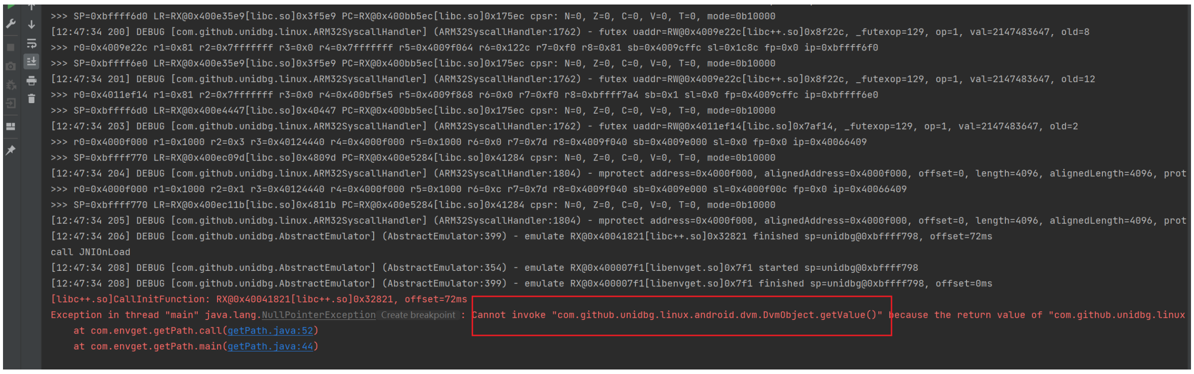
Function name	Segment	Start	Length	Locals
fflush	.plt	00000758	0000000C	
abort	.plt	00000764	0000000C	
__aeabi_memcpy	.plt	00000770	0000000C	
__aeabi_memclr	.plt	0000077C	0000000C	
dladdr	.plt	00000788	0000000C	
snprintf	.plt	00000794	0000000C	
__gnu_Unwind_Find_e...	.plt	000007A0	0000000C	
sub_7AC	.text	000007AC	0000000C	
nullsub_1	.text	000007BC	00000004	
j_nullsub_1	.text	000007C0	00000004	
sub_7C4	.text	000007C4	0000000C	
Java_com_example_en...	.text	000007F0	00000020	00000018
_JNIEnv::NewStringUT...	.text	00000814	00000020	00000018
sub_834	.text	00000834	00000048	00000010
sub_87C	.text	0000087C	0000025E	00000038
sub_AE4	.text	00000AE4	00000098	00000020
sub_B90	.text	00000B90	0000009A	00000020
sub_C40	.text	00000C40	00000138	00000040

```
int __fastcall Java_com_example_envget_MainActivity_stringFromJNI(_JNIEnv *a1)
{
  char *v1; // r0

  v1 = getenv("PATH");
  return _JNIEnv::NewStringUTF(a1, v1);
}
```

干净清爽，接下来进入Unidbg。

```
package com.envget;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Module;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class getPath extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;


    getPath(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机
        vm = emulator.createDalvikVM();
        // 设置是否打印相关调用细节
        vm.setVerbose(true);
        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/envget/libenvget.so"), true);
        module = dm.getModule();
        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }

    public static void main(String[] args) {
        getPath demo = new getPath();
        demo.call();
    }

    public void call() {
        List<Object> list = new ArrayList<>(10);
        list.add(vm.getJNIEnv());
        list.add(0);
        Number number = module.callFunction(emulator, 0x7f1, list.toArray())[0];

 System.out.println("result:"+vm.getObject(number.intValue()).getValue().toStrin
g());
    }
}
```

直接运行

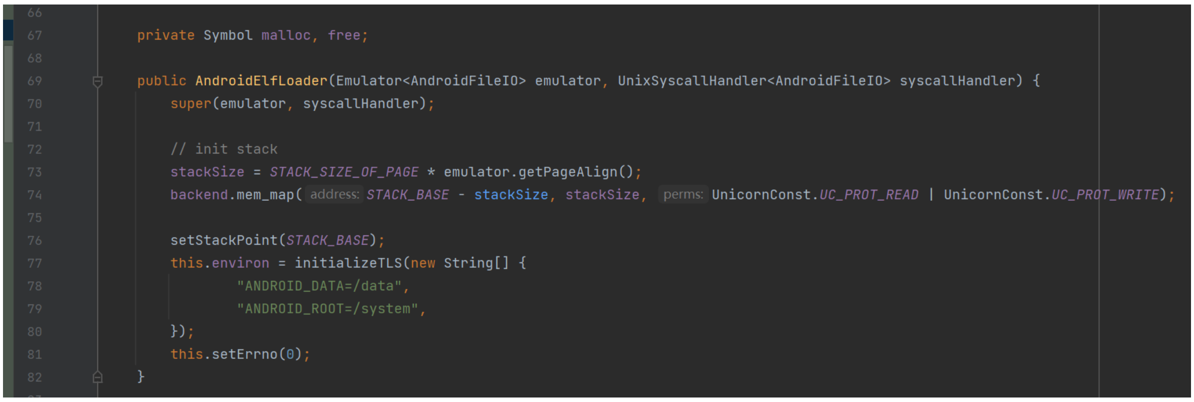


```
>>> SP=0xbffff6d0 LR=RX@0x400e35e9[libc.so]0x3f5e9 PC=RX@0x400bb5ec[libc.so]0x175ec cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
[12:47:34 200] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1762) - futex uaddr=RW@0x4009e22c[libc++.so]0x8f22c, _futexop=129, op=1, val=2147483647, old=8
>>> r0=0x4009e22c r1=0x81 r2=0x7fffffff r3=0x0 r4=0x7fffffff r5=0x4009f064 r6=0x122c r7=0xf0 r8=0x81 sb=0x4009cffc sl=0x1c8c fp=0x0 ip=0xbffff6f0
>>> SP=0xbffff6e0 LR=RX@0x400e35e9[libc.so]0x3f5e9 PC=RX@0x400bb5ec[libc.so]0x175ec cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
[12:47:34 201] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1762) - futex uaddr=RW@0x4009e22c[libc++.so]0x8f22c, _futexop=129, op=1, val=2147483647, old=12
>>> r0=0x4011ef14 r1=0x81 r2=0x7fffffff r3=0x0 r4=0x400bf5e5 r5=0x4009f868 r6=0x0 r7=0xf0 r8=0xbffff7a4 sb=0x1 sl=0x0 fp=0x4009cffc ip=0xbffff6e0
>>> SP=0xbffff6d0 LR=RX@0x400e4447[libc.so]0x40447 PC=RX@0x400bb5ec[libc.so]0x175ec cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
[12:47:34 203] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1762) - futex uaddr=RW@0x4011ef14[libc.so]0x7af14, _futexop=129, op=1, val=2147483647, old=2
>>> r0=0x4000f000 r1=0x1000 r2=0x3 r3=0x40124440 r4=0x4000f000 r5=0x1000 r6=0x0 r7=0x7d r8=0x4009f040 sb=0x4009e000 sl=0x0 fp=0x0 ip=0x40066409
>>> SP=0xbffff770 LR=RX@0x400ec09d[libc.so]0x4809d PC=RX@0x400e5284[libc.so]0x41284 cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
[12:47:34 204] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1804) - mprotect address=0x4000f000, alignedAddress=0x4000f000, offset=0, length=4096, alignedLength=4096, prot
>>> r0=0x4000f000 r1=0x1000 r2=0x1 r3=0x40124440 r4=0x4000f000 r5=0x1000 r6=0xc r7=0x7d r8=0x4009f040 sb=0x4009e000 sl=0x4000f00c fp=0x0 ip=0x40066409
>>> SP=0xbffff770 LR=RX@0x400ec11b[libc.so]0x4811b PC=RX@0x400e5284[libc.so]0x41284 cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
[12:47:34 205] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1804) - mprotect address=0x4000f000, alignedAddress=0x4000f000, offset=0, length=4096, alignedLength=4096, prot
[12:47:34 206] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x40041821[libc++.so]0x32821 finished sp=unidbg@0xbffff798, offset=72ms
call JNIOnLoad
[12:47:34 208] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:354) - emulate RX@0x400007f1[libenvget.so]0x7f1 started sp=unidbg@0xbffff798
[12:47:34 208] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x400007f1[libenvget.so]0x7f1 finished sp=unidbg@0xbffff798, offset=0ms
[libc++.so]CallInitFunction: RX@0x40041821[libc++.so]0x32821, offset=72ms
Exception in thread "main" java.lang.NullPointerException : Cannot invoke "com.github.unidbg.linux.android.dvm.DvmObject.getValue()" because the return value of "com.github.unidbg.linux
    at com.envget.getPath.call(getPath.java:52)
    at com.envget.getPath.main(getPath.java:44)
```

getValue取不到结果，原因就是getenv没有返回值，那么该怎么办呢？哈哈，这里轮到我炫技啦。茴香豆的“茴”有几种写法呢？这里给env返回正确的值有几种办法呢？

方法一

Unidbg提供了对环境变量的初始化，它在src/main/java/com/github/unidbg/linux/AndroidElfLoader.java中。

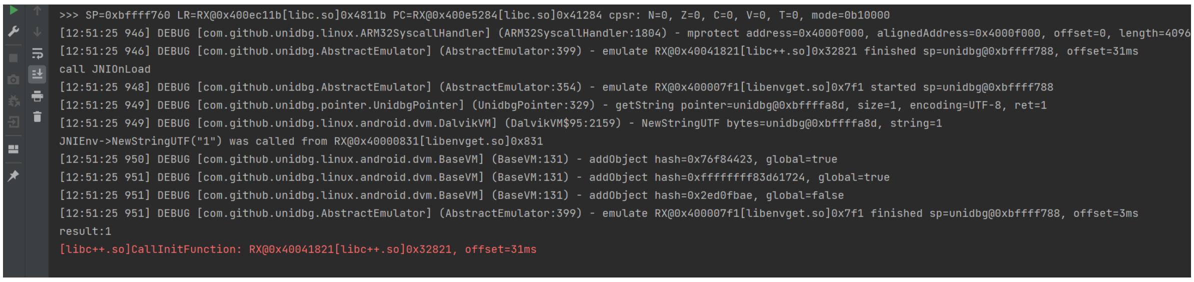


```

    private Symbol malloc, free;

    public AndroidElfLoader(Emulator<AndroidFileIO> emulator, UnixSyscallHandler<AndroidFileIO> syscallHandler) {
        super(emulator, syscallHandler);

        // init stack
        stackSize = STACK_SIZE_OF_PAGE * emulator.getPageAlign();
        backend.mem_map(STACK_BASE - stackSize, stackSize, UnicornConst.UC_PROT_READ | UnicornConst.UC_PROT_WRITE);

        setStackPoint(STACK_BASE);
        this.environ = initializeTLS(new String[] {
                "ANDROID_DATA=/data",
                "ANDROID_ROOT=/system",
        });
        this.setErrno(0);
    }
```

我们填上这一个就行，为了辨别不同方法是否生效，我们这里返回1

```
this.environ = initializeTLS(new String[] {
        "ANDROID_DATA=/data",
        "ANDROID_ROOT=/system",
        "PATH=1",
});
```

运行试试，一切正常。



```
>>> SP=0xbffff760 LR=RX@0x400ec11b[libc.so]0x4811b PC=RX@0x400e5284[libc.so]0x41284 cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
[12:51:25 946] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1804) - mprotect address=0x4000f000, alignedAddress=0x4000f000, offset=0, length=4096
[12:51:25 946] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x40041821[libc++.so]0x32821 finished sp=unidbg@0xbffff788, offset=31ms
call JNIOnLoad
[12:51:25 948] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:354) - emulate RX@0x400007f1[libenvget.so]0x7f1 started sp=unidbg@0xbffff788
[12:51:25 949] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=unidbg@0xbffffa8d, size=1, encoding=UTF-8, ret=1
[12:51:25 949] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$95:2159) - NewStringUTF bytes=unidbg@0xbffffa8d, string=1
JNIEnv->NewStringUTF("1") was called from RX@0x40000831[libenvget.so]0x831
[12:51:25 950] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x76f84423, global=true
[12:51:25 951] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0xffffffff83d61724, global=true
[12:51:25 951] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x2ed0fbae, global=false
[12:51:25 951] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x400007f1[libenvget.so]0x7f1 finished sp=unidbg@0xbffff788, offset=3ms
result:1
[libc++.so]CallInitFunction: RX@0x40041821[libc++.so]0x32821, offset=31ms
```

接下来**把PATH注释掉**，我们试第二种方法。

方法二

libc 提供了setenv方法，可以设置环境变量。

```java
package com.envget;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Module;
import com.github.unidbg.Symbol;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class getPath extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;


    getPath(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机
        vm = emulator.createDalvikVM();
        // 设置是否打印相关调用细节
        vm.setVerbose(true);
        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/envget/libenvget.so"), true);
        module = dm.getModule();
        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }
    public static void main(String[] args) {
        getPath demo = new getPath();
        demo.setEnv();
        demo.call();
    }

    public void call() {
        List<Object> list = new ArrayList<>(10);
        list.add(vm.getJNIEnv());
        list.add(0);
        Number number = module.callFunction(emulator, 0x7f1, list.toArray())[0];

 System.out.println("result:"+vm.getObject(number.intValue()).getValue().toStrin
g());
    }

    // setenv设置环境变量
    public void setEnv(){
        Symbol setenv = module.findSymbolByName("setenv", true);
        setenv.call(emulator, "PATH", "2", 0);
    };

}
```

再次运行返回结果即2。

方法三

我们也可以通过HookZz hook函数，替换结果

```
package com.envget;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
import com.github.unidbg.Symbol;
import com.github.unidbg.arm.context.EditableArm32RegisterContext;
import com.github.unidbg.hook.hookzz.HookEntryInfo;
import com.github.unidbg.hook.hookzz.HookZz;
import com.github.unidbg.hook.hookzz.IHookZz;
import com.github.unidbg.hook.hookzz.WrapCallback;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;
import com.github.unidbg.memory.MemoryBlock;
import com.github.unidbg.pointer.UnidbgPointer;

import java.io.File;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
```

```
public class getPath extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;


    getPath(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机
        vm = emulator.createDalvikVM();
        // 设置是否打印相关调用细节
        vm.setVerbose(true);
        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/envget/libenvget.so"), true);
        module = dm.getModule();
        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }

    public static void main(String[] args) {
        getPath demo = new getPath();
        demo.hookgetEnvByHookZz();
        demo.call();
    }

    public void call() {
        List<Object> list = new ArrayList<>(10);
        list.add(vm.getJNIEnv());
        list.add(0);
        Number number = module.callFunction(emulator, 0x7f1, list.toArray())[0];

 System.out.println("result:"+vm.getObject(number.intValue()).getValue().toStrin
g());
    }

    public void hookgetEnvByHookZz(){
        IHookZz hookZz = HookZz.getInstance(emulator);

        hookZz.wrap(module.findSymbolByName("getenv"), new
WrapCallback<EditableArm32RegisterContext>() {
            String name;
            @Override
            public void preCall(Emulator<?> emulator,
EditableArm32RegisterContext ctx, HookEntryInfo info) {
                name = ctx.getPointerArg(0).getString(0);
            }
            @Override
            public void postCall(Emulator<?> emulator,
EditableArm32RegisterContext ctx, HookEntryInfo info) {
```

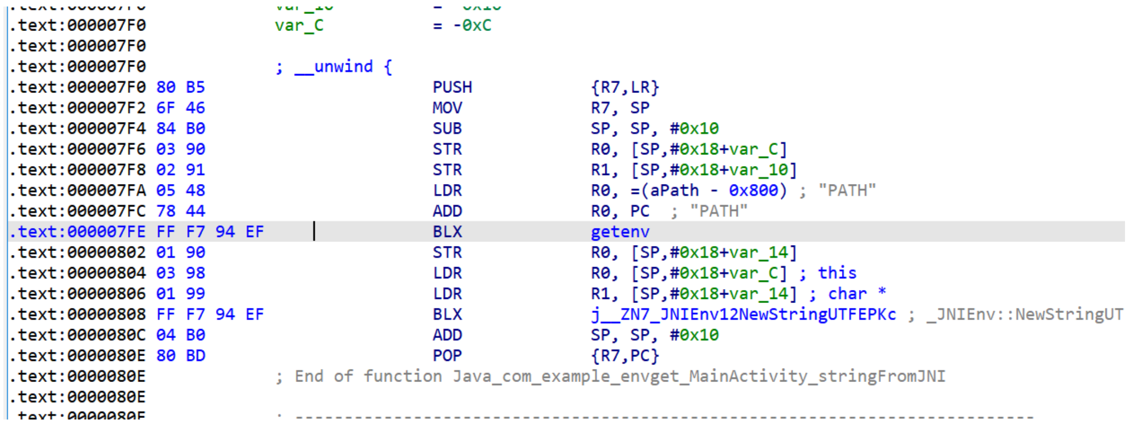
```
                switch (name){
                    case "PATH":{
                        MemoryBlock replaceBlock =
emulator.getMemory().malloc(0x100, true);
                        UnidbgPointer replacePtr = replaceBlock.getPointer();
                        String pathValue = "3";
                        replacePtr.write(0,
pathValue.getBytes(StandardCharsets.UTF_8), 0, pathValue.length());
                        ctx.setR0(replacePtr.toIntPeer());
                    }
                }

            }
        });
    };

}
```

方法四

我们也可以通过断点的方式hook

```
package com.envget;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
import com.github.unidbg.arm.context.EditableArm32RegisterContext;
import com.github.unidbg.debugger.BreakPointCallback;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;
import unicorn.ArmConst;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class getPath extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;


    getPath(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机
        vm = emulator.createDalvikVM();
        // 设置是否打印相关调用细节
        vm.setVerbose(true);
        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/envget/libenvget.so"), true);
        module = dm.getModule();
        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }

    public static void main(String[] args) {
        getPath demo = new getPath();
        demo.hookgetEnvByBreakPointer();
        demo.call();
    }

    public void call() {
        List<Object> list = new ArrayList<>(10);
        list.add(vm.getJNIEnv());
        list.add(0);
        Number number = module.callFunction(emulator, 0x7f1, list.toArray())[0];

 System.out.println("result:"+vm.getObject(number.intValue()).getValue().toStrin
g());
    }

    public void hookgetEnvByBreakPointer(){
        emulator.attach().addBreakPoint(module.base + 0x7FE, new
BreakPointCallback() {
            @Override
            public boolean onHit(Emulator<?> emulator, long address) {
                EditableArm32RegisterContext registerContext =
emulator.getContext();
                registerContext.getPointerArg(0).setString(0, "4");
                emulator.getBackend().reg_write(ArmConst.UC_ARM_REG_PC,
(address)+5);
                return true;
            }
        });
    }


}
```

我们直接让R0指针指向正确的值，并操纵PC寄存器跳过这条指令



```
.text:000007F0                 var_10          = -0x10
.text:000007F0                 var_C           = -0xC
.text:000007F0
.text:000007F0                 ; __unwind {
.text:000007F0 80 B5                           PUSH            {R7,LR}
.text:000007F2 6F 46                           MOV             R7, SP
.text:000007F4 84 B0                           SUB             SP, SP, #0x10
.text:000007F6 03 90                           STR             R0, [SP,#0x18+var_C]
.text:000007F8 02 91                           STR             R1, [SP,#0x18+var_10]
.text:000007FA 05 48                           LDR             R0, =(aPath - 0x800) ; "PATH"
.text:000007FC 78 44                           ADD             R0, PC  ; "PATH"
.text:000007FE FF F7 94 EF                     BLX             getenv
.text:00000802 01 90                           STR             R0, [SP,#0x18+var_14]
.text:00000804 03 98                           LDR             R0, [SP,#0x18+var_C] ; this
.text:00000806 01 99                           LDR             R1, [SP,#0x18+var_14] ; char *
.text:00000808 FF F7 94 EF                     BLX             j__ZN7_JNIEnv12NewStringUTFEPKc ; _JNIEnv::NewStringUT
.text:0000080C 04 B0                           ADD             SP, SP, #0x10
.text:0000080E 80 BD                           POP             {R7,PC}
.text:0000080E                 ; End of function Java_com_example_envget_MainActivity_stringFromJNI
.text:0000080E
.text:0000080E                 ; ---------------------------------------------------------------------------
```

这条指令四个字节长度，又因为thumb模式+1，所以address+5。

方法五

仿照SystemPropertyHook写一下，代码如下

getPath.java文件

```java
package com.envget;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Module;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
import com.github.unidbg.linux.android.dvm.AbstractJni;
import com.github.unidbg.linux.android.dvm.DalvikModule;
import com.github.unidbg.linux.android.dvm.VM;
import com.github.unidbg.memory.Memory;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class getPath extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;


    getPath(){
        // 创建模拟器实例，要模拟32位或者64位，在这里区分
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        // 模拟器的内存操作接口
        final Memory memory = emulator.getMemory();
        // 设置系统类库解析
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机
        vm = emulator.createDalvikVM();
        // 设置是否打印相关调用细节
        vm.setVerbose(true);
        // 加载so到虚拟内存，加载成功以后会默认调用init_array等函数

        EnvHook envHook = new EnvHook(emulator);
        memory.addHookListener(envHook);
        DalvikModule dm = vm.loadLibrary(new File("unidbg-android/src/test/resources/envget/libenvget.so"), true);
        module = dm.getModule();
        // 设置JNI
        vm.setJni(this);
        System.out.println("call JNIOnLoad");
        dm.callJNI_OnLoad(emulator);
    }

    public static void main(String[] args) {
        getPath demo = new getPath();
        demo.call();
    }

    public void call() {
        List<Object> list = new ArrayList<>(10);
        list.add(vm.getJNIEnv());
        list.add(0);
        Number number = module.callFunction(emulator, 0x7f1, list.toArray())[0];

 System.out.println("result:"+vm.getObject(number.intValue()).getValue().toString());
    }
}
```

EnvHook.java

```
package com.envget;

import com.github.unidbg.Emulator;
import com.github.unidbg.arm.ArmHook;
import com.github.unidbg.arm.HookStatus;
import com.github.unidbg.arm.context.RegisterContext;
import com.github.unidbg.hook.HookListener;
import com.github.unidbg.memory.SvcMemory;
import com.github.unidbg.pointer.UnidbgPointer;

public class EnvHook implements HookListener {

    private final Emulator<?> emulator;

    public EnvHook(Emulator<?> emulator) {
        this.emulator = emulator;
    }

    @Override
    public long hook(SvcMemory svcMemory, String libraryName, String symbolName, final long old) {
        if ("libc.so".equals(libraryName) && "getenv".equals(symbolName)) {
            if (emulator.is32Bit()) {
                return svcMemory.registerSvc(new ArmHook() {
                    @Override
                    protected HookStatus hook(Emulator<?> emulator) {
                        return getenv(old);
                    }
                }).peer;
            }
        }
        return 0;
    }

    private HookStatus getenv(long old) {
        RegisterContext context = emulator.getContext();
        UnidbgPointer pointer = context.getPointerArg(0);
        String key = pointer.getString(0);
        switch (key){
            case "PATH":{
                pointer.setString(0, "5");
                return HookStatus.LR(emulator, pointer.peer);
            }
        }
        return HookStatus.RET(emulator, old);
    }
}
```

五种方法全部演示完毕，在此处，显然是直接设置环境最方便，但在其他场景上，你可能需要求助另外四款。