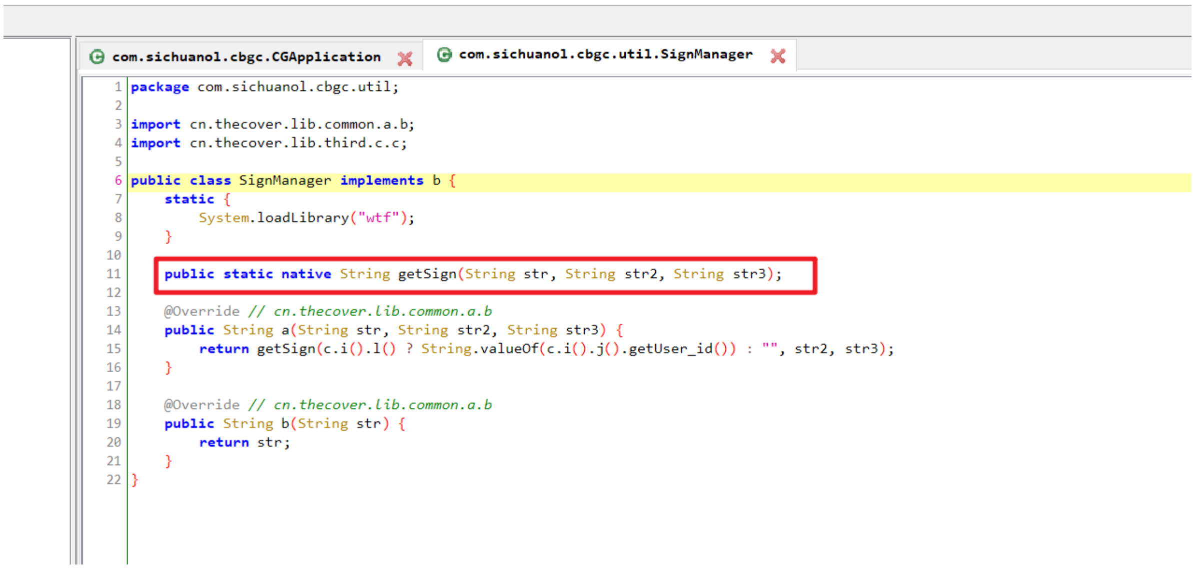
我和球友，以及群友每天都有很多交流，最常遇到的问题就是找不到Unidbg报错跑不下去的原因，打算以这个主题写一个系列，这是第一篇。

样本链接：https://pan.baidu.com/s/1uAqrcG22kDU7WdlLx7y8EQ
提取码：khqg

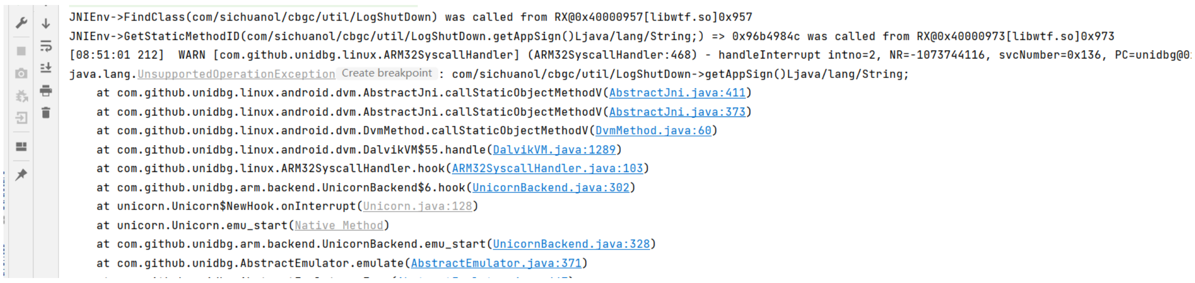

com.sichuanol.cbgc.CGApplication
com.sichuanol.cbgc.util.SignManager

```java
package com.sichuanol.cbgc.util;

import cn.thecover.lib.common.a.b;
import cn.thecover.lib.third.c.c;

public class SignManager implements b {
    static {
        System.loadLibrary("wtf");
    }

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

    @Override // cn.thecover.lib.common.a.b
    public String a(String str, String str2, String str3) {
        return getSign(c.i().l() ? String.valueOf(c.i().j().getUser_id()) : "", str2, str3);
    }

    @Override // cn.thecover.lib.common.a.b
    public String b(String str) {
        return str;
    }
}
```

这是目标函数，参数12填入空字符串，参数3字符串内容是毫秒级时间戳

```java
package com.cbgc;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
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.StringObject;
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 WTF extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    public WTF() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .setProcessName("com.cbgc")
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机,传入APK，Unidbg可以替我们处理许多问题
        File apkFile = new File("unidbg-
android/src/test/resources/CBGC/cbgc.apk");
        vm = emulator.createDalvikVM(apkFile);
        // 加载目标SO
        File file = new File("unidbg-
android/src/test/resources/CBGC/libwtf.so");
        DalvikModule dm = vm.loadLibrary(file, true);

        module = dm.getModule();
        vm.setJni(this); // 设置JNI
        vm.setVerbose(true); // 打印日志

        dm.callJNI_OnLoad(emulator);// 调用JNI OnLoad
    }


    public static void main(String[] args) {
        WTF wtf = new WTF();
        System.out.println(wtf.getSign());
    }


    public String getSign() {
        List<Object> list = new ArrayList<>(10);
        // arg1 env
        list.add(vm.getJNIEnv());
        // arg2 jobject/jclazz 一般用不到，直接填0
        list.add(0);

        list.add(vm.addLocalObject(new StringObject(vm, "")));
        list.add(vm.addLocalObject(new StringObject(vm, "")));
        list.add(vm.addLocalObject(new StringObject(vm, "1628093856262")));
        // 参数准备完成
        // call function
        Number number = module.callFunction(emulator, 0x931, list.toArray())[0];
        String result = vm.getObject(number.intValue()).getValue().toString();
        return result;
    }


}
```

运行代码



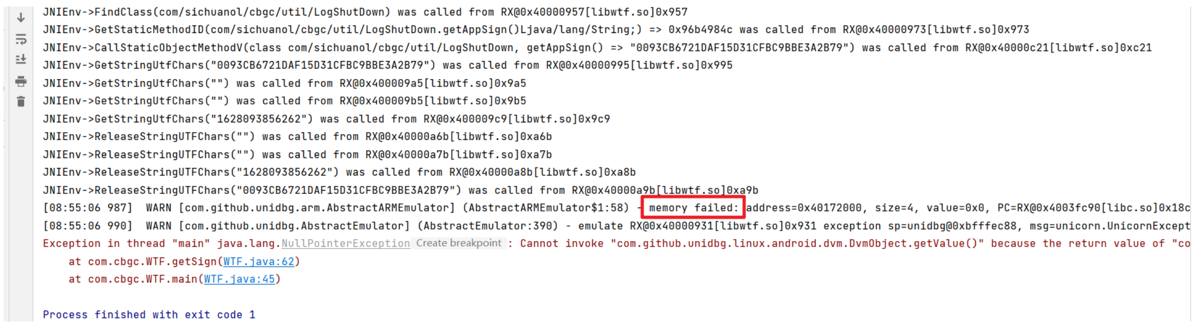
```
JNIEnv->FindClass(com/sichuanol/cbgc/util/LogShutDown) was called from RX@0x40000957[libwtf.so]0x957
JNIEnv->GetStaticMethodID(com/sichuanol/cbgc/util/LogShutDown.getAppSign()Ljava/lang/String;) => 0x96b4984c was called from RX@0x40000973[libwtf.so]0x973
[08:51:01 212]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:468) - handleInterrupt intno=2, NR=-1073744116, svcNumber=0x136, PC=unidbg@0
java.lang.UnsupportedOperationException Create breakpoint : com/sichuanol/cbgc/util/LogShutDown->getAppSign()Ljava/lang/String;
    at com.github.unidbg.linux.android.dvm.AbstractJni.callStaticObjectMethodV(AbstractJni.java:411)
    at com.github.unidbg.linux.android.dvm.AbstractJni.callStaticObjectMethodV(AbstractJni.java:373)
    at com.github.unidbg.linux.android.dvm.DvmMethod.callStaticObjectMethodV(DvmMethod.java:60)
    at com.github.unidbg.linux.android.dvm.DalvikVM$55.handle(DalvikVM.java:1289)
    at com.github.unidbg.linux.ARM32SyscallHandler.hook(ARM32SyscallHandler.java:103)
    at com.github.unidbg.arm.backend.UnicornBackend$6.hook(UnicornBackend.java:302)
    at unicorn.Unicorn$NewHook.onInterrupt(Unicorn.java:128)
    at unicorn.Unicorn.emu_start(Native Method)
    at com.github.unidbg.arm.backend.UnicornBackend.emu_start(UnicornBackend.java:328)
    at com.github.unidbg.AbstractEmulator.emulate(AbstractEmulator.java:371)
```

Frida call 一下，看看是啥返回值，补一下

```java
package com.cbgc;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
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.*;
import com.github.unidbg.memory.Memory;

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

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

    public WTF() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .setProcessName("com.cbgc")
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机,传入APK，Unidbg可以替我们处理许多问题
        File apkFile = new File("unidbg-
android/src/test/resources/CBGC/cbgc.apk");
        vm = emulator.createDalvikVM(apkFile);
        // 加载目标SO
        File file = new File("unidbg-
android/src/test/resources/CBGC/libwtf.so");
        DalvikModule dm = vm.loadLibrary(file, true);

        module = dm.getModule();
        vm.setJni(this); // 设置JNI
        vm.setVerbose(true); // 打印日志

        dm.callJNI_OnLoad(emulator);// 调用JNI OnLoad
    }


    public static void main(String[] args) {
        WTF wtf = new WTF();
        System.out.println(wtf.getSign());
    }


    public String getSign() {
        List<Object> list = new ArrayList<>(10);
        // arg1 env
        list.add(vm.getJNIEnv());
        // arg2 jobject/jclazz 一般用不到，直接填0
        list.add(0);

        list.add(vm.addLocalObject(new StringObject(vm, "")));
        list.add(vm.addLocalObject(new StringObject(vm, "")));
        list.add(vm.addLocalObject(new StringObject(vm, "1628093856262")));
        // 参数准备完成
```

```java
        // call function
        Number number = module.callFunction(emulator, 0x931, list.toArray())[0];
        String result = vm.getObject(number.intValue()).getValue().toString();
        return result;
    }


    @Override
    public DvmObject<?> callStaticObjectMethodV(BaseVM vm, DvmClass dvmClass,
String signature, VaList vaList) {
        if (signature.equals("com/sichuanol/cbgc/util/LogShutDown-
>getAppSign()Ljava/lang/String;")){
            return new StringObject(vm, "0093CB6721DAF15D31CFBC9BBE3A2B79");
        }
        return super.callStaticObjectMethodV(vm, dvmClass, signature, vaList);
    }


}
```

再次运行

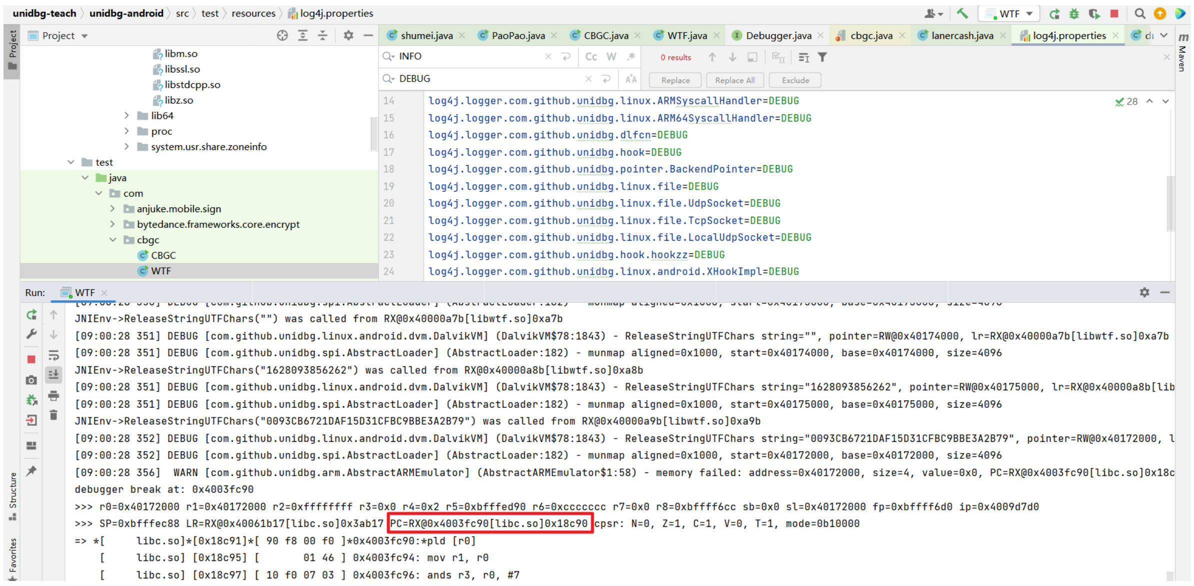


```
JNIEnv->FindClass(com/sichuanol/cbgc/util/LogShutDown) was called from RX@0x40000957[libwtf.so]0x957
JNIEnv->GetStaticMethodID(com/sichuanol/cbgc/util/LogShutDown.getAppSign()Ljava/lang/String;) => 0x96b4984c was called from RX@0x40000973[libwtf.so]0x973
JNIEnv->CallStaticObjectMethodV(class com/sichuanol/cbgc/util/LogShutDown, getAppSign() => "0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000c21[libwtf.so]0xc21
JNIEnv->GetStringUtfChars("0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000995[libwtf.so]0x995
JNIEnv->GetStringUtfChars("") was called from RX@0x400009a5[libwtf.so]0x9a5
JNIEnv->GetStringUtfChars("") was called from RX@0x400009b5[libwtf.so]0x9b5
JNIEnv->GetStringUtfChars("1628093856262") was called from RX@0x400009c9[libwtf.so]0x9c9
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a6b[libwtf.so]0xa6b
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a7b[libwtf.so]0xa7b
JNIEnv->ReleaseStringUTFChars("1628093856262") was called from RX@0x40000a8b[libwtf.so]0xa8b
JNIEnv->ReleaseStringUTFChars("0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000a9b[libwtf.so]0xa9b
[08:55:06 987]  WARN [com.github.unidbg.arm.AbstractARMEmulator] (AbstractARMEmulator$1:58) - memory failed: address=0x40172000, size=4, value=0x0, PC=RX@0x4003fc90[libc.so]0x18c
[08:55:06 990]  WARN [com.github.unidbg.AbstractEmulator] (AbstractEmulator:390) - emulate RX@0x40000931[libwtf.so]0x931 exception sp=unidbg@0xbfffec88, msg=unicorn.UnicornExcept
Exception in thread "main" java.lang.NullPointerException Create breakpoint : Cannot invoke "com.github.unidbg.linux.android.dvm.DvmObject.getValue()" because the return value of "co
    at com.cbgc.WTF.getSign(WTF.java:62)
    at com.cbgc.WTF.main(WTF.java:45)

Process finished with exit code 1
```

内存错误是Unidbg中最糟糕的错误之一，就好比感冒，可能是季节性流感带来的，也可能是扁桃体发炎，或者是劳累……人需要做一堆检查才能确认原因，Unidbg中的内存错误也是。

我们先进行第一种尝试，把Unidbg的日志全开，看是不是我们漏过了什么要点，导致这个问题

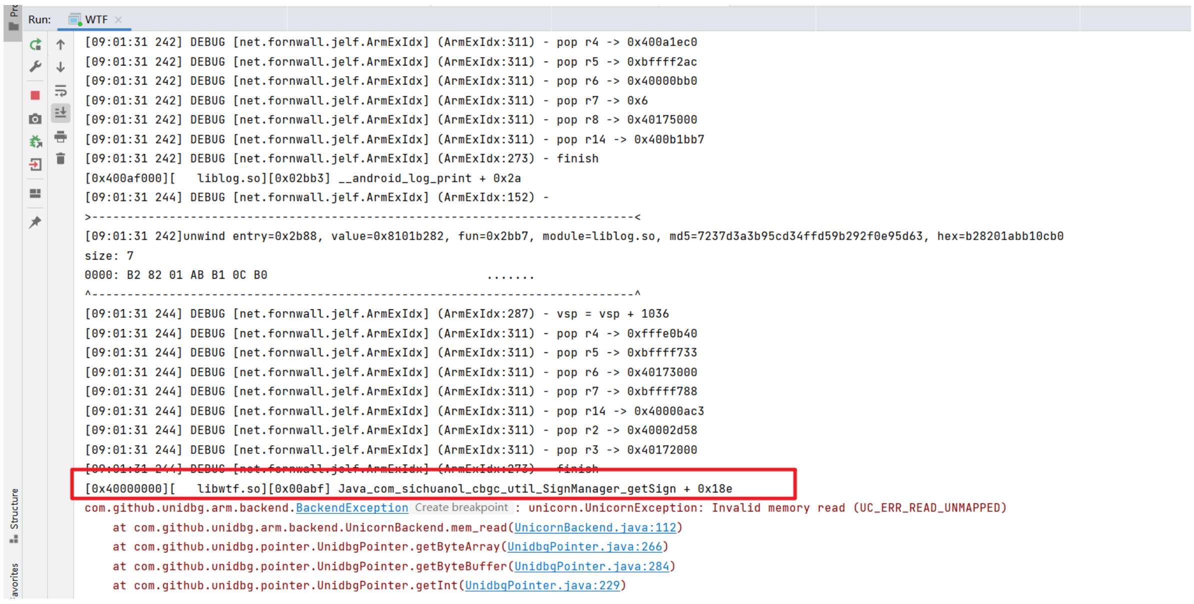
src/test/resources/log4j.properties中**INFO**全配置成**DEBUG**



```
log4j.logger.com.github.unidbg.linux.ARMSyscallHandler=DEBUG
log4j.logger.com.github.unidbg.linux.ARM64SyscallHandler=DEBUG
log4j.logger.com.github.unidbg.dlfcn=DEBUG
log4j.logger.com.github.unidbg.hook=DEBUG
log4j.logger.com.github.unidbg.pointer.BackendPointer=DEBUG
log4j.logger.com.github.unidbg.linux.file=DEBUG
log4j.logger.com.github.unidbg.linux.file.UdpSocket=DEBUG
log4j.logger.com.github.unidbg.linux.file.TcpSocket=DEBUG
log4j.logger.com.github.unidbg.linux.file.LocalUdpSocket=DEBUG
log4j.logger.com.github.unidbg.hook.hookzz=DEBUG
log4j.logger.com.github.unidbg.linux.android.XHookImpl=DEBUG
```

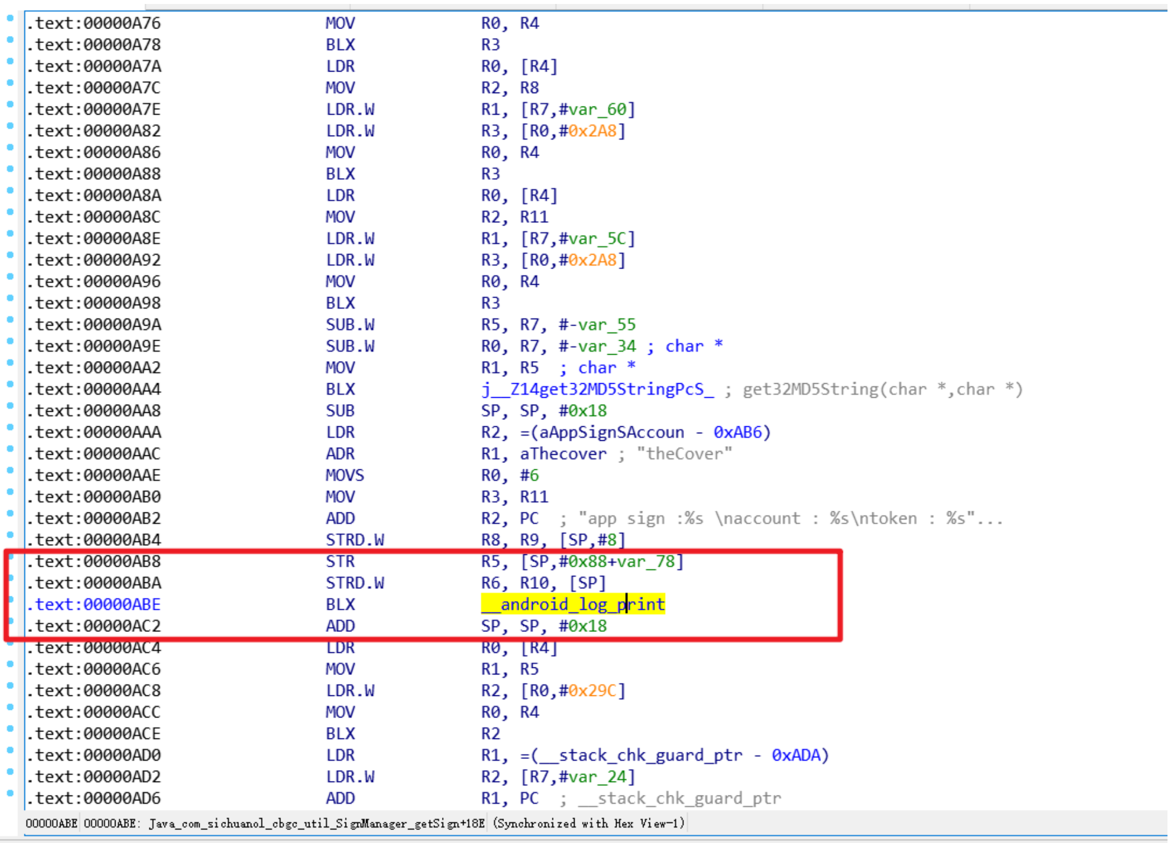
```
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a7b[libwtf.so]0xa7b
[09:00:28 351] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$78:1843) - ReleaseStringUTFChars string="", pointer=RW@0x40174000, lr=RX@0x40000a7b[libwtf.so]0xa7b
[09:00:28 351] DEBUG [com.github.unidbg.spi.AbstractLoader] (AbstractLoader:182) - munmap aligned=0x1000, start=0x40174000, base=0x40174000, size=4096
JNIEnv->ReleaseStringUTFChars("1628093856262") was called from RX@0x40000a8b[libwtf.so]0xa8b
[09:00:28 351] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$78:1843) - ReleaseStringUTFChars string="1628093856262", pointer=RW@0x40175000, lr=RX@0x40000a8b[lib
[09:00:28 351] DEBUG [com.github.unidbg.spi.AbstractLoader] (AbstractLoader:182) - munmap aligned=0x1000, start=0x40175000, base=0x40175000, size=4096
JNIEnv->ReleaseStringUTFChars("0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000a9b[libwtf.so]0xa9b
[09:00:28 352] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$78:1843) - ReleaseStringUTFChars string="0093CB6721DAF15D31CFBC9BBE3A2B79", pointer=RW@0x40172000, l
[09:00:28 352] DEBUG [com.github.unidbg.spi.AbstractLoader] (AbstractLoader:182) - munmap aligned=0x1000, start=0x40172000, base=0x40172000, size=4096
[09:00:28 356]  WARN [com.github.unidbg.arm.AbstractARMEmulator] (AbstractARMEmulator$1:58) - memory failed: address=0x40172000, size=4, value=0x0, PC=RX@0x4003fc90[libc.so]0x18c
debugger break at: 0x4003fc90
>>> r0=0x40172000 r1=0x40172000 r2=0xffffffff r3=0x0 r4=0x2 r5=0xbfffed90 r6=0xcccccccc r7=0x0 r8=0xbffff6cc sb=0x0 sl=0x40172000 fp=0xbffff6d0 ip=0x4009d7d0
>>> SP=0xbfffec88 LR=RX@0x40061b17[libc.so]0x3ab17 PC=RX@0x4003fc90[libc.so]0x18c90 cpsr: N=0, Z=1, C=1, V=0, T=1, mode=0b10000
=> *[     libc.so]*[0x18c91]*[ 90 f8 00 f0 ]*0x4003fc90:*pld [r0]
    [     libc.so] [0x18c95] [       01 46 ] 0x4003fc94: mov r1, r0
    [     libc.so] [0x18c97] [ 10 f0 07 03 ] 0x4003fc96: ands r3, r0, #7
```

问题出在libc里？日志全开后，Unidbg在最后发生错误的地方断了下来。输入bt指令查看调用栈



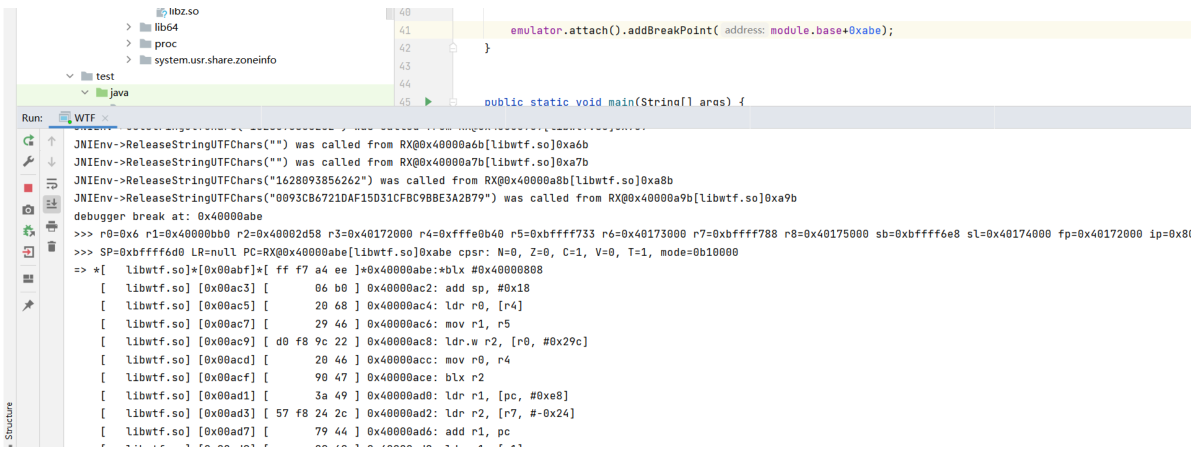
```
Run:  WTF
[09:01:31 242] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r4 -> 0x400a1ec0
[09:01:31 242] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r5 -> 0xbffff2ac
[09:01:31 242] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r6 -> 0x40000bb0
[09:01:31 242] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r7 -> 0x6
[09:01:31 242] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r8 -> 0x40175000
[09:01:31 242] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r14 -> 0x400b1bb7
[09:01:31 242] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:273) - finish
[0x400af000][   liblog.so][0x02bb3] __android_log_print + 0x2a
[09:01:31 244] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:152) -
>-----------------------------------------------------------------------------<
[09:01:31 242]unwind entry=0x2b88, value=0x8101b282, fun=0x2bb7, module=liblog.so, md5=7237d3a3b95cd34ffd59b292f0e95d63, hex=b28201abb10cb0
size: 7
0000: B2 82 01 AB B1 0C B0                                .......
^-----------------------------------------------------------------------------^
[09:01:31 244] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:287) - vsp = vsp + 1036
[09:01:31 244] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r4 -> 0xfffe0b40
[09:01:31 244] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r5 -> 0xbffff733
[09:01:31 244] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r6 -> 0x40173000
[09:01:31 244] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r7 -> 0xbffff788
[09:01:31 244] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r14 -> 0x40000ac3
[09:01:31 244] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r2 -> 0x40002d58
[09:01:31 244] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:311) - pop r3 -> 0x40172000
[09:01:31 244] DEBUG [net.fornwall.jelf.ArmExIdx] (ArmExIdx:273) - finish
[0x40000000][   libwtf.so][0x00abf] Java_com_sichuanol_cbgc_util_SignManager_getSign + 0x18e
com.github.unidbg.arm.backend.BackendException Create breakpoint : unicorn.UnicornException: Invalid memory read (UC_ERR_READ_UNMAPPED)
    at com.github.unidbg.arm.backend.UnicornBackend.mem_read(UnicornBackend.java:112)
    at com.github.unidbg.pointer.UnidbgPointer.getByteArray(UnidbgPointer.java:266)
    at com.github.unidbg.pointer.UnidbgPointer.getByteBuffer(UnidbgPointer.java:284)
    at com.github.unidbg.pointer.UnidbgPointer.getInt(UnidbgPointer.java:229)
```

Unidbg提供了大量详细的调用回溯信息，直接拉到底部，看是从目标样本哪里调用的。IDA中跳到0xabf地址



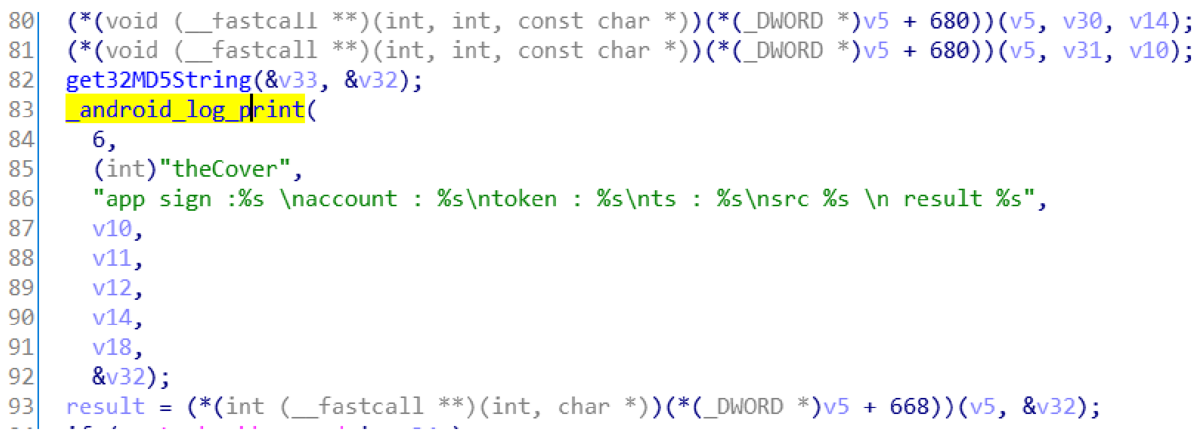
```
.text:00000A76                 MOV             R0, R4
.text:00000A78                 BLX             R3
.text:00000A7A                 LDR             R0, [R4]
.text:00000A7C                 MOV             R2, R8
.text:00000A7E                 LDR.W           R1, [R7,#var_60]
.text:00000A82                 LDR.W           R3, [R0,#0x2A8]
.text:00000A86                 MOV             R0, R4
.text:00000A88                 BLX             R3
.text:00000A8A                 LDR             R0, [R4]
.text:00000A8C                 MOV             R2, R11
.text:00000A8E                 LDR.W           R1, [R7,#var_5C]
.text:00000A92                 LDR.W           R3, [R0,#0x2A8]
.text:00000A96                 MOV             R0, R4
.text:00000A98                 BLX             R3
.text:00000A9A                 SUB.W           R5, R7, #-var_55
.text:00000A9E                 SUB.W           R0, R7, #-var_34 ; char *
.text:00000AA2                 MOV             R1, R5  ; char *
.text:00000AA4                 BLX             j__Z14get32MD5StringPcS_ ; get32MD5String(char *,char *)
.text:00000AA8                 SUB             SP, SP, #0x18
.text:00000AAA                 LDR             R2, =(aAppSignSAccoun - 0xAB6)
.text:00000AAC                 ADR             R1, aThecover ; "theCover"
.text:00000AAE                 MOVS            R0, #6
.text:00000AB0                 MOV             R3, R11
.text:00000AB2                 ADD             R2, PC  ; "app sign :%s \naccount : %s\ntoken : %s"...
.text:00000AB4                 STRD.W          R8, R9, [SP,#8]
.text:00000AB8                 STR             R5, [SP,#0x88+var_78]
.text:00000ABA                 STRD.W          R6, R10, [SP]
.text:00000ABE                 BLX             __android_log_print
.text:00000AC2                 ADD             SP, SP, #0x18
.text:00000AC4                 LDR             R0, [R4]
.text:00000AC6                 MOV             R1, R5
.text:00000AC8                 LDR.W           R2, [R0,#0x29C]
.text:00000ACC                 MOV             R0, R4
.text:00000ACE                 BLX             R2
.text:00000AD0                 LDR             R1, =(__stack_chk_guard_ptr - 0xADA)
.text:00000AD2                 LDR.W           R2, [R7,#var_24]
.text:00000AD6                 ADD             R1, PC  ; __stack_chk_guard_ptr
00000ABE 00000ABE: Java_com_sichuanol_cbgc_util_SignManager_getSign+18E (Synchronized with Hex View-1)
```

真是奇了个怪，打个日志怎么还报错了？把日志等级再改成**INFO**，我们在此处下断点清清爽爽做个分析。



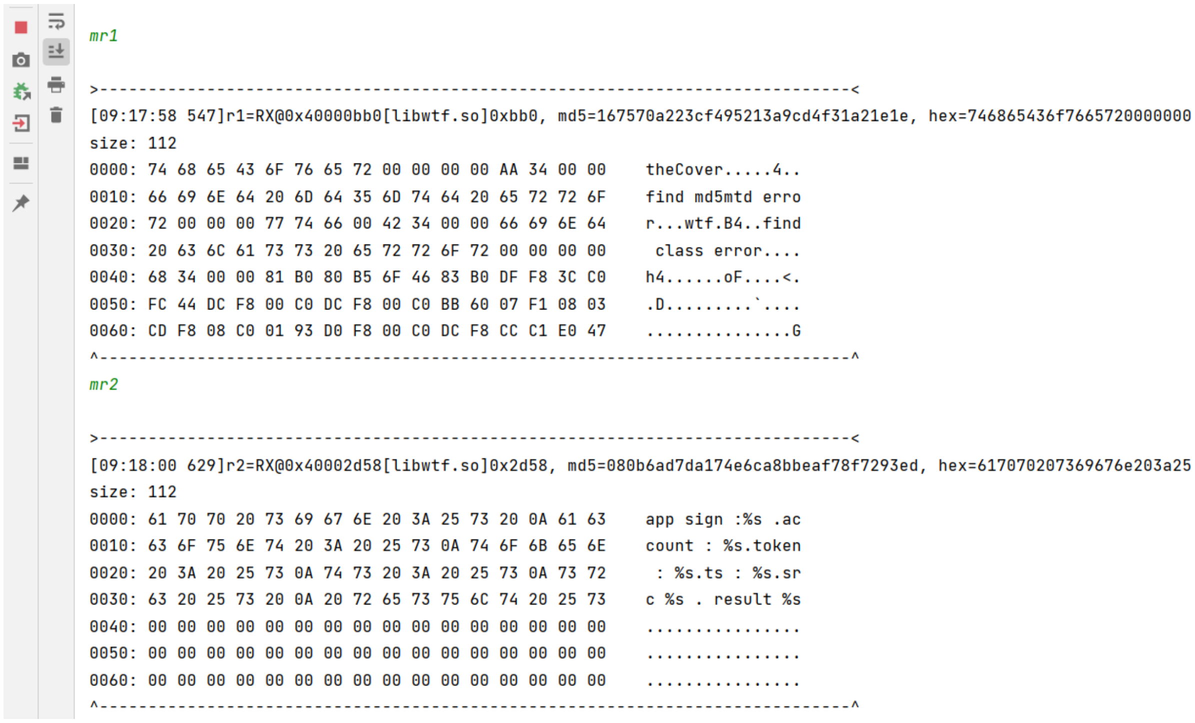
```
    emulator.attach().addBreakPoint( address: module.base+0xabe);
}

public static void main(String[] args) {
```

```
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a6b[libwtf.so]0xa6b
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a7b[libwtf.so]0xa7b
JNIEnv->ReleaseStringUTFChars("1628093856262") was called from RX@0x40000a8b[libwtf.so]0xa8b
JNIEnv->ReleaseStringUTFChars("0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000a9b[libwtf.so]0xa9b
debugger break at: 0x40000abe
>>> r0=0x6 r1=0x40000bb0 r2=0x40002d58 r3=0x40172000 r4=0xfffe0b40 r5=0xbffff733 r6=0x40173000 r7=0xbffff788 r8=0x40175000 sb=0xbffff6e8 sl=0x40174000 fp=0x40172000 ip=0x8
>>> SP=0xbffff6d0 LR=null PC=RX@0x40000abe[libwtf.so]0xabe cpsr: N=0, Z=0, C=1, V=0, T=1, mode=0b10000
=> *[   libwtf.so]*[0x00abf]*[ ff f7 a4 ee ]*0x40000abe:*blx #0x40000808
    [   libwtf.so] [0x00ac3] [       06 b0 ] 0x40000ac2: add sp, #0x18
    [   libwtf.so] [0x00ac5] [       20 68 ] 0x40000ac4: ldr r0, [r4]
    [   libwtf.so] [0x00ac7] [       29 46 ] 0x40000ac6: mov r1, r5
    [   libwtf.so] [0x00ac9] [ d0 f8 9c 22 ] 0x40000ac8: ldr.w r2, [r0, #0x29c]
    [   libwtf.so] [0x00acd] [       20 46 ] 0x40000acc: mov r0, r4
    [   libwtf.so] [0x00acf] [       90 47 ] 0x40000ace: blx r2
    [   libwtf.so] [0x00ad1] [       3a 49 ] 0x40000ad0: ldr r1, [pc, #0xe8]
    [   libwtf.so] [0x00ad3] [ 57 f8 24 2c ] 0x40000ad2: ldr r2, [r7, #-0x24]
    [   libwtf.so] [0x00ad7] [       79 44 ] 0x40000ad6: add r1, pc
```



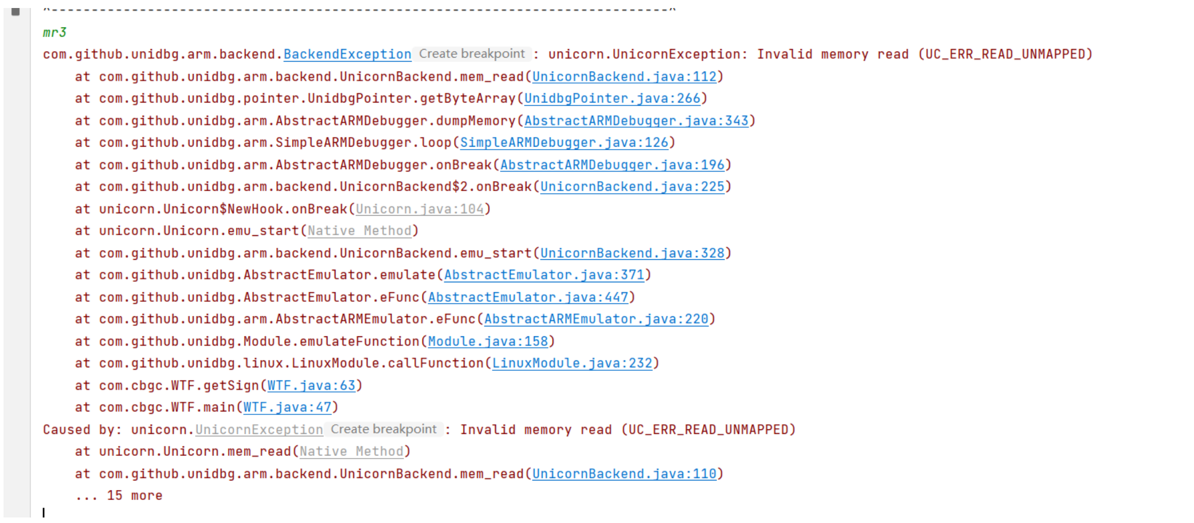
```
(*(void (__fastcall **)(int, int, const char *))(*(_DWORD *)v5 + 680))(v5, v30, v14);
(*(void (__fastcall **)(int, int, const char *))(*(_DWORD *)v5 + 680))(v5, v31, v10);
get32MD5String(&v33, &v32);
_android_log_print(
  6,
  (int)"theCover",
  "app sign :%s \naccount : %s\ntoken : %s\nts : %s\nsrc %s \n result %s",
  v10,
  v11,
  v12,
  v14,
  v18,
  &v32);
result = (*(int (__fastcall **)(int, char *))(*(_DWORD *)v5 + 668))(v5, &v32);
```

android_log_print是常用函数，参数1是此日志的优先级，参数2是tag，参数3是格式化字符串，之后的都是内容。Console Debugger中验证一下



```
mr1

>-----------------------------------------------------------------------------<
[09:17:58 547]r1=RX@0x40000bb0[libwtf.so]0xbb0, md5=167570a223cf495213a9cd4f31a21e1e, hex=746865436f7665720000000
size: 112
0000: 74 68 65 43 6F 76 65 72 00 00 00 00 AA 34 00 00    theCover.....4..
0010: 66 69 6E 64 20 6D 64 35 6D 74 64 20 65 72 72 6F    find md5mtd erro
0020: 72 00 00 00 77 74 66 00 42 34 00 00 66 69 6E 64    r...wtf.B4..find
0030: 20 63 6C 61 73 73 20 65 72 72 6F 72 00 00 00 00     class error....
0040: 68 34 00 00 81 B0 80 B5 6F 46 83 B0 DF F8 3C C0    h4......oF....<.
0050: FC 44 DC F8 00 C0 DC F8 00 C0 BB 60 07 F1 08 03    .D.........`....
0060: CD F8 08 C0 01 93 D0 F8 00 C0 DC F8 CC C1 E0 47    ...............G
^-----------------------------------------------------------------------------^
mr2

>-----------------------------------------------------------------------------<
[09:18:00 629]r2=RX@0x40002d58[libwtf.so]0x2d58, md5=080b6ad7da174e6ca8bbeaf78f7293ed, hex=617070207369676e203a25
size: 112
0000: 61 70 70 20 73 69 67 6E 20 3A 25 73 20 0A 61 63    app sign :%s .ac
0010: 63 6F 75 6E 74 20 3A 20 25 73 0A 74 6F 6B 65 6E    count : %s.token
0020: 20 3A 20 25 73 0A 74 73 20 3A 20 25 73 0A 73 72     : %s.ts : %s.sr
0030: 63 20 25 73 20 0A 20 72 65 73 75 6C 74 20 25 73    c %s . result %s
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
^-----------------------------------------------------------------------------^
```

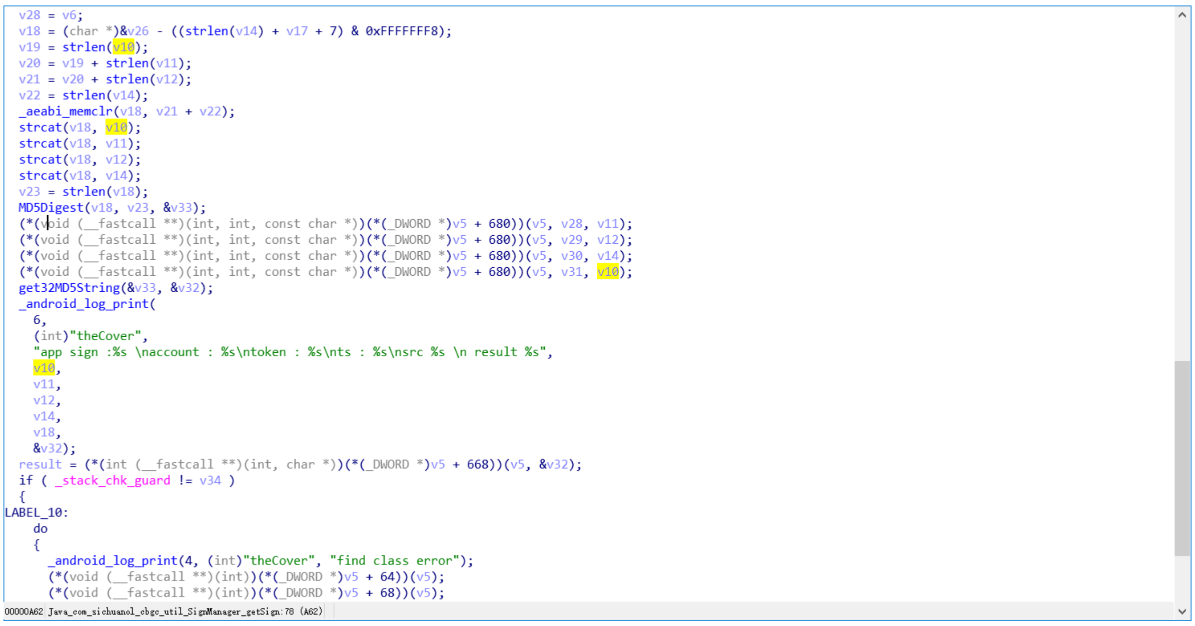
参数1是6，可以直接看，通过mrx查看其余参数，参数23也都正常，那参数4呢



```
mr3
com.github.unidbg.arm.backend.BackendException Create breakpoint : unicorn.UnicornException: Invalid memory read (UC_ERR_READ_UNMAPPED)
	at com.github.unidbg.arm.backend.UnicornBackend.mem_read(UnicornBackend.java:112)
	at com.github.unidbg.pointer.UnidbgPointer.getByteArray(UnidbgPointer.java:266)
	at com.github.unidbg.arm.AbstractARMDebugger.dumpMemory(AbstractARMDebugger.java:343)
	at com.github.unidbg.arm.SimpleARMDebugger.loop(SimpleARMDebugger.java:126)
	at com.github.unidbg.arm.AbstractARMDebugger.onBreak(AbstractARMDebugger.java:196)
	at com.github.unidbg.arm.backend.UnicornBackend$2.onBreak(UnicornBackend.java:225)
	at unicorn.Unicorn$NewHook.onBreak(Unicorn.java:104)
	at unicorn.Unicorn.emu_start(Native Method)
	at com.github.unidbg.arm.backend.UnicornBackend.emu_start(UnicornBackend.java:328)
	at com.github.unidbg.AbstractEmulator.emulate(AbstractEmulator.java:371)
	at com.github.unidbg.AbstractEmulator.eFunc(AbstractEmulator.java:447)
	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.cbgc.WTF.getSign(WTF.java:63)
	at com.cbgc.WTF.main(WTF.java:47)
Caused by: unicorn.UnicornException Create breakpoint : Invalid memory read (UC_ERR_READ_UNMAPPED)
	at unicorn.Unicorn.mem_read(Native Method)
	at com.github.unidbg.arm.backend.UnicornBackend.mem_read(UnicornBackend.java:110)
	... 15 more
```

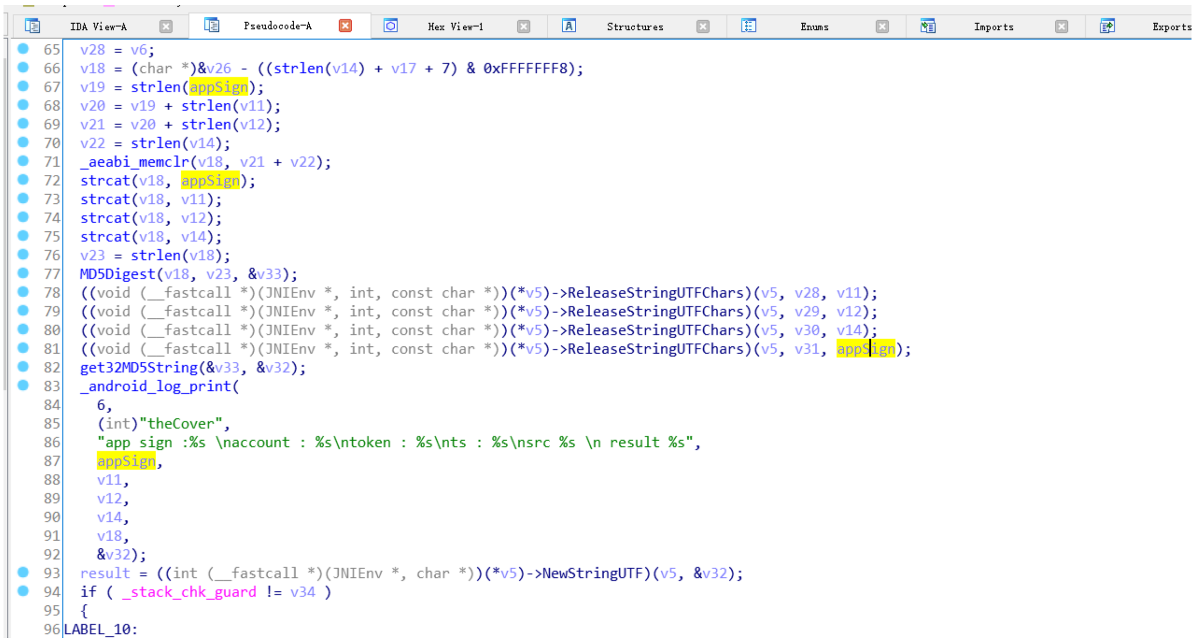
好家伙，内存异常了，看来问题出在这里，回IDA看看。

从格式化字符串上看，参数4应该就是“app sign”



```
v28 = v6;
v18 = (char *)&v26 - ((strlen(v14) + v17 + 7) & 0xFFFFFFF8);
v19 = strlen(v10);
v20 = v19 + strlen(v11);
v21 = v20 + strlen(v12);
v22 = strlen(v14);
_aeabi_memclr(v18, v21 + v22);
strcat(v18, v10);
strcat(v18, v11);
strcat(v18, v12);
strcat(v18, v14);
v23 = strlen(v18);
MD5Digest(v18, v23, &v33);
(*(void (__fastcall **)(int, int, const char *))(*(_DWORD *)v5 + 680))(v5, v28, v11);
(*(void (__fastcall **)(int, int, const char *))(*(_DWORD *)v5 + 680))(v5, v29, v12);
(*(void (__fastcall **)(int, int, const char *))(*(_DWORD *)v5 + 680))(v5, v30, v14);
(*(void (__fastcall **)(int, int, const char *))(*(_DWORD *)v5 + 680))(v5, v31, v10);
get32MD5String(&v33, &v32);
_android_log_print(
  6,
  (int)"theCover",
  "app sign :%s \naccount : %s\ntoken : %s\nts : %s\nsrc %s \n result %s",
  v10,
  v11,
  v12,
  v14,
  v18,
  &v32);
result = (*(int (__fastcall **)(int, char *))(*(_DWORD *)v5 + 668))(v5, &v32);
if ( _stack_chk_guard != v34 )
{
LABEL_10:
  do
  {
    _android_log_print(4, (int)"theCover", "find class error");
    (*(void (__fastcall **)(int))(*(_DWORD *)v5 + 64))(v5);
    (*(void (__fastcall **)(int))(*(_DWORD *)v5 + 68))(v5);
00000A62 Java_com_sichuanol_cbgc_util_SignManager_getSign:78 (A62)
```

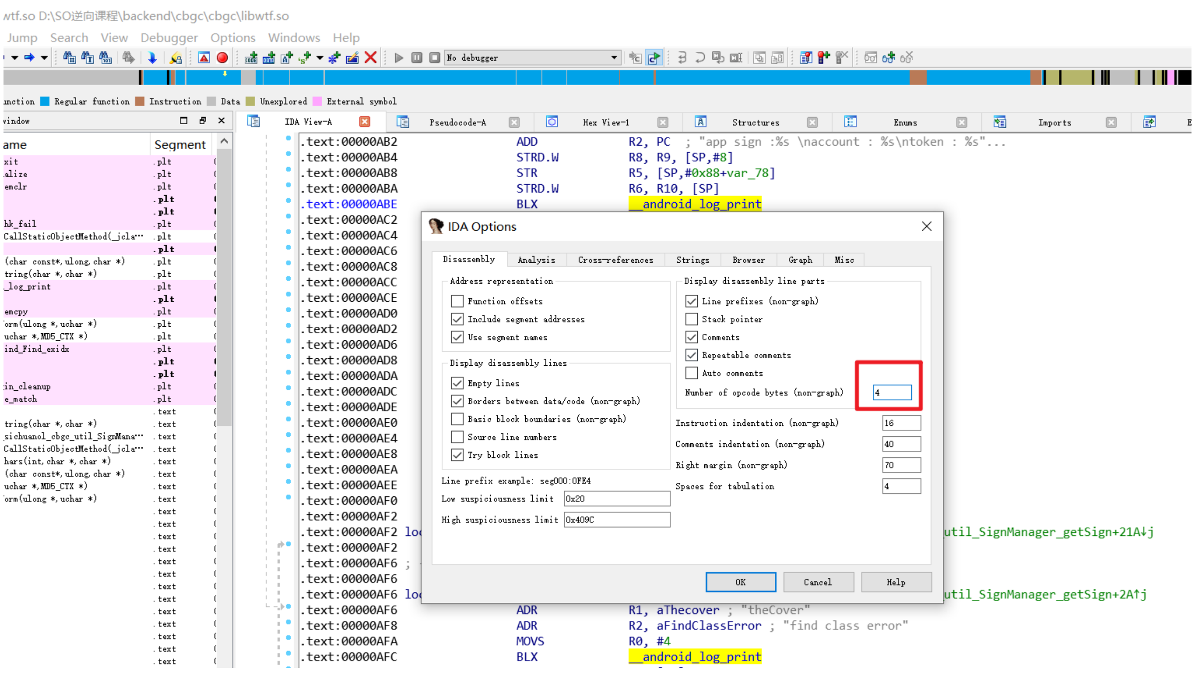
把JNIEnv声明一下，再改一下函数名，不然怪难受的



```
v28 = v6;
v18 = (char *)&v26 - ((strlen(v14) + v17 + 7) & 0xFFFFFFF8);
v19 = strlen(appSign);
v20 = v19 + strlen(v11);
v21 = v20 + strlen(v12);
v22 = strlen(v14);
_aeabi_memclr(v18, v21 + v22);
strcat(v18, appSign);
strcat(v18, v11);
strcat(v18, v12);
strcat(v18, v14);
v23 = strlen(v18);
MD5Digest(v18, v23, &v33);
((void (__fastcall *)(JNIEnv *, int, const char *))(*v5)->ReleaseStringUTFChars)(v5, v28, v11);
((void (__fastcall *)(JNIEnv *, int, const char *))(*v5)->ReleaseStringUTFChars)(v5, v29, v12);
((void (__fastcall *)(JNIEnv *, int, const char *))(*v5)->ReleaseStringUTFChars)(v5, v30, v14);
((void (__fastcall *)(JNIEnv *, int, const char *))(*v5)->ReleaseStringUTFChars)(v5, v31, appSign);
get32MD5String(&v33, &v32);
_android_log_print(
  6,
  (int)"theCover",
  "app sign :%s \naccount : %s\ntoken : %s\nts : %s\nsrc %s \n result %s",
  appSign,
  v11,
  v12,
  v14,
  v18,
  &v32);
result = ((int (__fastcall *)(JNIEnv *, char *))(*v5)->NewStringUTF)(v5, &v32);
if ( _stack_chk_guard != v34 )
{
LABEL_10:
```

啊这，appSign的字符串已经释放了，下面竟然还在打印它。。。这应该是开发时的BUG。我们需要这个调用不发生。

介绍几种常见方法吧


wtf.so D:\SO逆向课程\backend\cbgc\cbgc\libwtf.so

```
.text:00000AB2                 ADD             R2, PC  ; "app sign :%s \naccount : %s\ntoken : %s"...
.text:00000AB4                 STRD.W          R8, R9, [SP,#8]
.text:00000AB8                 STR             R5, [SP,#0x88+var_78]
.text:00000ABA                 STRD.W          R6, R10, [SP]
.text:00000ABE                 BLX             __android_log_print
...
.text:00000AF6                 ADR             R1, aThecover ; "theCover"
.text:00000AF8                 ADR             R2, aFindClassError ; "find class error"
.text:00000AFA                 MOVS            R0, #4
.text:00000AFC                 BLX             __android_log_print
```

IDA Options — Disassembly

Address representation
- [ ] Function offsets
- [x] Include segment addresses
- [x] Use segment names

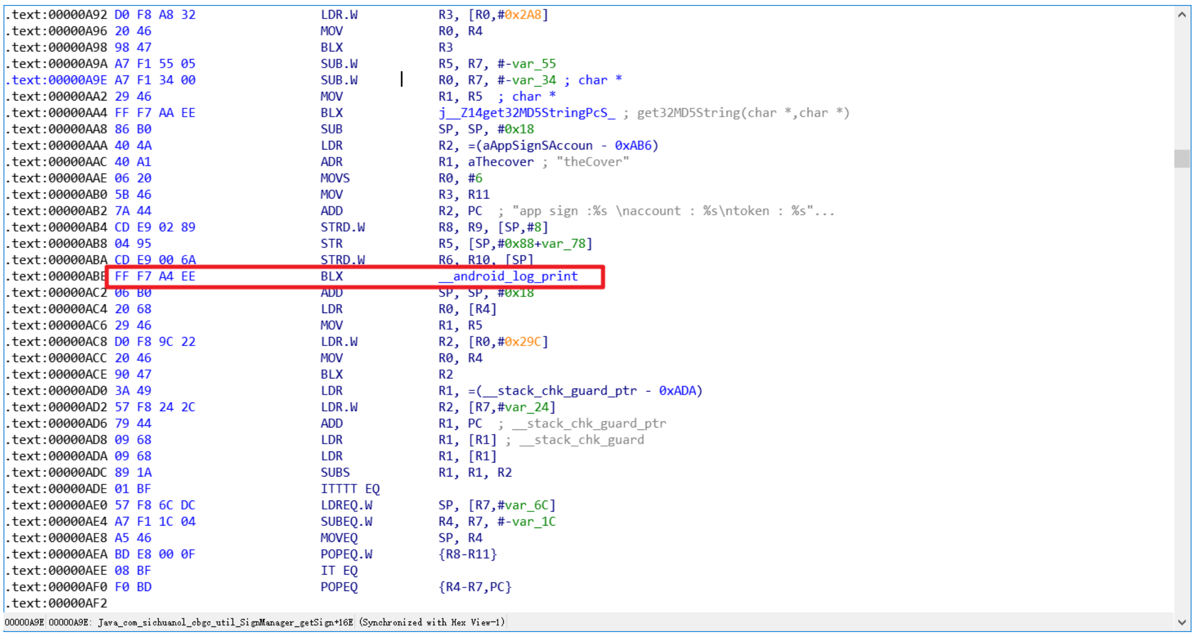
Display disassembly lines
- [x] Empty lines
- [x] Borders between data/code (non-graph)
- [ ] Basic block boundaries (non-graph)
- [ ] Source line numbers
- [x] Try block lines

Line prefix example: seg000:0FE4
Low suspiciousness limit: 0x20
High suspiciousness limit: 0x409C

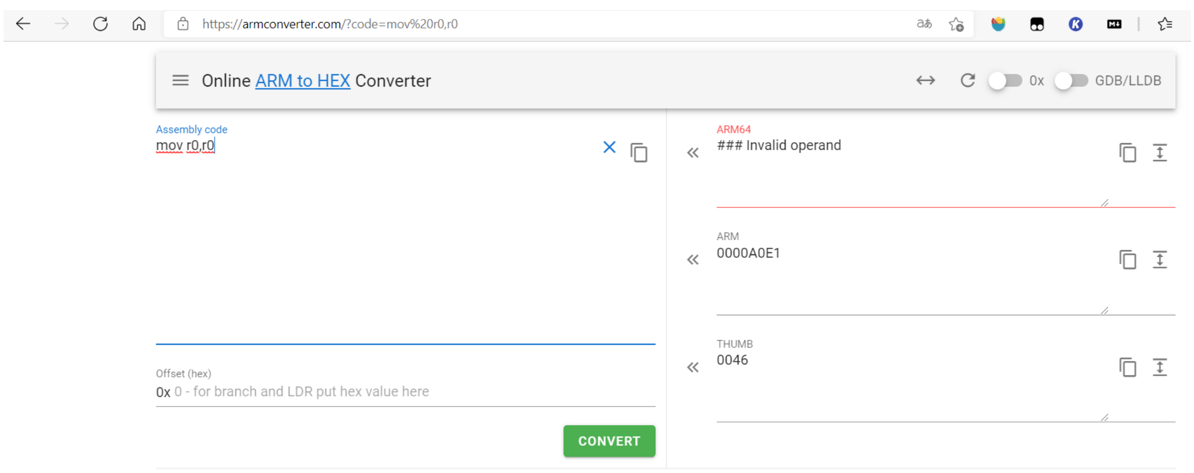
Display disassembly line parts
- [x] Line prefixes (non-graph)
- [ ] Stack pointer
- [x] Comments
- [x] Repeatable comments
- [ ] Auto comments

Number of opcode bytes (non-graph): 4
Instruction indentation (non-graph): 16
Comments indentation (non-graph): 40
Right margin (non-graph): 70
Spaces for tabulation: 4

Options-General打开字节码


```
.text:00000A92 D0 F8 A8 32                 LDR.W           R3, [R0,#0x2A8]
.text:00000A96 20 46                       MOV             R0, R4
.text:00000A98 98 47                       BLX             R3
.text:00000A9A A7 F1 55 05                 SUB.W           R5, R7, #-var_55
.text:00000A9E A7 F1 34 00                 SUB.W           R0, R7, #-var_34 ; char *
.text:00000AA2 29 46                       MOV             R1, R5  ; char *
.text:00000AA4 FF F7 AA EE                 BLX             j__Z14get32MD5StringPcS_ ; get32MD5String(char *,char *)
.text:00000AA8 86 B0                       SUB             SP, SP, #0x18
.text:00000AAA 40 4A                       LDR             R2, =(aAppSignSAccoun - 0xAB6)
.text:00000AAC 40 A1                       ADR             R1, aThecover ; "theCover"
.text:00000AAE 06 20                       MOVS            R0, #6
.text:00000AB0 5B 46                       MOV             R3, R11
.text:00000AB2 7A 44                       ADD             R2, PC  ; "app sign :%s \naccount : %s\ntoken : %s"...
.text:00000AB4 CD E9 02 89                 STRD.W          R8, R9, [SP,#8]
.text:00000AB8 04 95                       STR             R5, [SP,#0x88+var_78]
.text:00000ABA CD E9 00 6A                 STRD.W          R6, R10, [SP]
.text:00000ABE FF F7 A4 EE                 BLX             __android_log_print
.text:00000AC2 06 B0                       ADD             SP, SP, #0x18
.text:00000AC4 20 68                       LDR             R0, [R4]
.text:00000AC6 29 46                       MOV             R1, R5
.text:00000AC8 D0 F8 9C 22                 LDR.W           R2, [R0,#0x29C]
.text:00000ACC 20 46                       MOV             R0, R4
.text:00000ACE 90 47                       BLX             R2
.text:00000AD0 3A 49                       LDR             R1, =(__stack_chk_guard_ptr - 0xADA)
.text:00000AD2 57 F8 24 2C                 LDR.W           R2, [R7,#var_24]
.text:00000AD6 79 44                       ADD             R1, PC  ; __stack_chk_guard_ptr
.text:00000AD8 09 68                       LDR             R1, [R1] ; __stack_chk_guard
.text:00000ADA 09 68                       LDR             R1, [R1]
.text:00000ADC 89 1A                       SUBS            R1, R1, R2
.text:00000ADE 01 BF                       ITTTT EQ
.text:00000AE0 57 F8 6C DC                 LDREQ.W         SP, [R7,#var_6C]
.text:00000AE4 A7 F1 1C 04                 SUBEQ.W         R4, R7, #-var_1C
.text:00000AE8 A5 46                       MOVEQ           SP, R4
.text:00000AEA BD E8 00 0F                 POPEQ.W         {R8-R11}
.text:00000AEE 08 BF                       IT EQ
.text:00000AF0 F0 BD                       POPEQ           {R4-R7,PC}
.text:00000AF2
00000A9E 00000A9E: Java_com_sichuanol_cbgc_util_SignManager_getSign+16E (Synchronized with Hex View-1)
```

我们需要修改这四个字节


https://armconverter.com/?code=mov%20r0,r0

Online ARM to HEX Converter

Assembly code: mov r0,r0

ARM64: ### Invalid operand

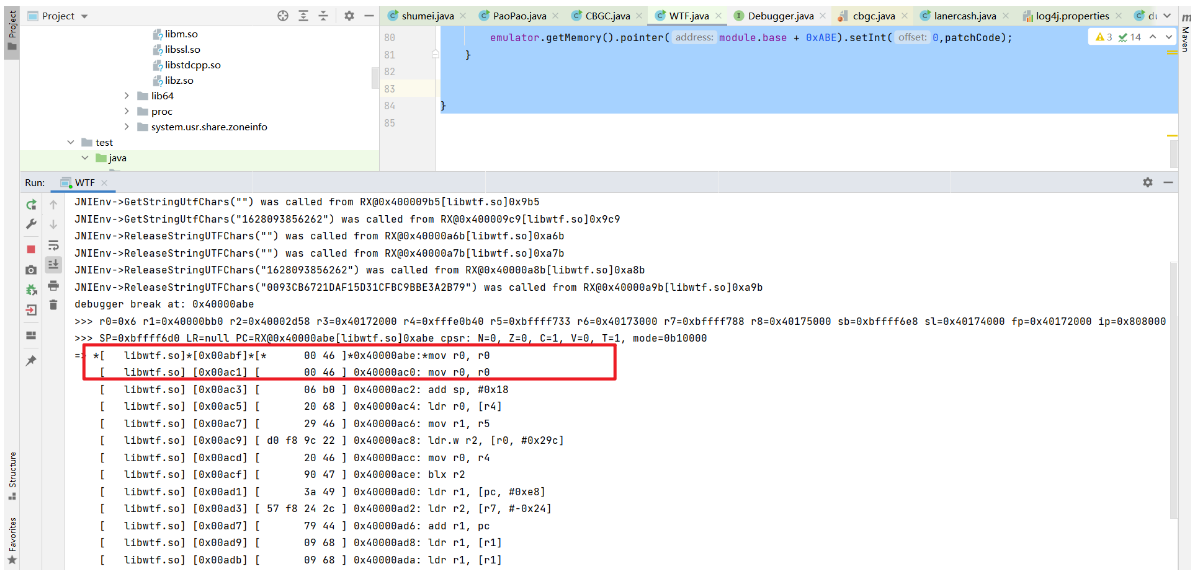
ARM: 0000A0E1

THUMB: 0046

Offset (hex): 0x 0 - for branch and LDR put hex value here

不如写两条无效指令吧，我首先写入0046，debug发现不对，调整了一下大小端序，再DEBUG似乎OK了

```java
package com.cbgc;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
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.*;
import com.github.unidbg.memory.Memory;

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

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

    public WTF() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .setProcessName("com.cbgc")
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机,传入APK，Unidbg可以替我们处理许多问题
        File apkFile = new File("unidbg-
android/src/test/resources/CBGC/cbgc.apk");
        vm = emulator.createDalvikVM(apkFile);
        // 加载目标SO
        File file = new File("unidbg-
android/src/test/resources/CBGC/libwtf.so");
        DalvikModule dm = vm.loadLibrary(file, true);

        module = dm.getModule();
        vm.setJni(this); // 设置JNI
        vm.setVerbose(true); // 打印日志

        dm.callJNI_OnLoad(emulator);// 调用JNI OnLoad

        emulator.attach().addBreakPoint(module.base+0xabe);
    }


    public static void main(String[] args) {
        WTF wtf = new WTF();
        wtf.patchLog();
        System.out.println(wtf.getSign());
    }


    public String getSign() {
        List<Object> list = new ArrayList<>(10);
        // arg1 env
        list.add(vm.getJNIEnv());
        // arg2 jobject/jclazz 一般用不到，直接填0
        list.add(0);

        list.add(vm.addLocalObject(new StringObject(vm, "")));
        list.add(vm.addLocalObject(new StringObject(vm, "")));
        list.add(vm.addLocalObject(new StringObject(vm, "1628093856262")));
        // 参数准备完成
        // call function
        Number number = module.callFunction(emulator, 0x931, list.toArray())[0];
        String result = vm.getObject(number.intValue()).getValue().toString();
        return result;
    }


    @Override
    public DvmObject<?> callStaticObjectMethodV(BaseVM vm, DvmClass dvmClass,
String signature, VaList vaList) {
        if (signature.equals("com/sichuanol/cbgc/util/LogShutDown-
>getAppSign()Ljava/lang/String;")){
            return new StringObject(vm, "0093CB6721DAF15D31CFBC9BBE3A2B79");
        }
        return super.callStaticObjectMethodV(vm, dvmClass, signature, vaList);
    }

    public void patchLog(){
        int patchCode = 0x46004600;
        emulator.getMemory().pointer(module.base + 0xABE).setInt(0,patchCode);
    }


}
```

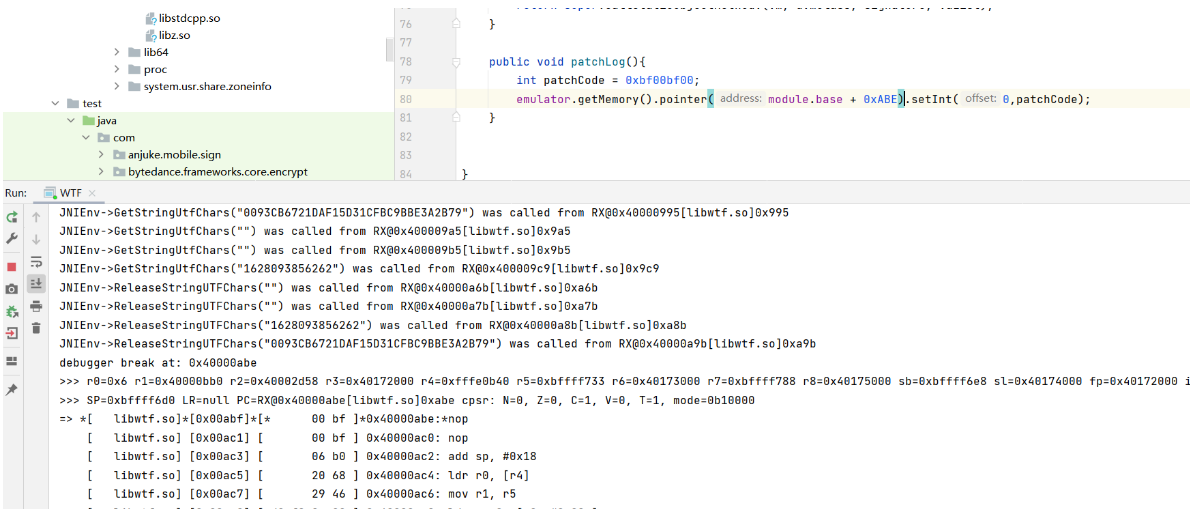


```
            emulator.getMemory().pointer( address: module.base + 0xABE).setInt( offset: 0,patchCode);
        }


}
```

```
JNIEnv->GetStringUtfChars("") was called from RX@0x400009b5[libwtf.so]0x9b5
JNIEnv->GetStringUtfChars("1628093856262") was called from RX@0x400009c9[libwtf.so]0x9c9
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a6b[libwtf.so]0xa6b
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a7b[libwtf.so]0xa7b
JNIEnv->ReleaseStringUTFChars("1628093856262") was called from RX@0x40000a8b[libwtf.so]0xa8b
JNIEnv->ReleaseStringUTFChars("0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000a9b[libwtf.so]0xa9b
debugger break at: 0x40000abe
>>> r0=0x6 r1=0x40000bb0 r2=0x40002d58 r3=0x40172000 r4=0xfffe0b40 r5=0xbffff733 r6=0x40173000 r7=0xbffff788 r8=0x40175000 sb=0xbffff6e8 sl=0x40174000 fp=0x40172000 ip=0x808000
>>> SP=0xbffff6d0 LR=null PC=RX@0x40000abe[libwtf.so]0xabe cpsr: N=0, Z=0, C=1, V=0, T=1, mode=0b10000
=> *[   libwtf.so]*[0x00abf]*[*      00 46 ]*0x40000abe:*mov r0, r0
    [   libwtf.so] [0x00ac1] [       00 46 ] 0x40000ac0: mov r0, r0
    [   libwtf.so] [0x00ac3] [       06 b0 ] 0x40000ac2: add sp, #0x18
    [   libwtf.so] [0x00ac5] [       20 68 ] 0x40000ac4: ldr r0, [r4]
    [   libwtf.so] [0x00ac7] [       29 46 ] 0x40000ac6: mov r1, r5
    [   libwtf.so] [0x00ac9] [ d0 f8 9c 22 ] 0x40000ac8: ldr.w r2, [r0, #0x29c]
    [   libwtf.so] [0x00acd] [       20 46 ] 0x40000acc: mov r0, r4
    [   libwtf.so] [0x00acf] [       90 47 ] 0x40000ace: blx r2
    [   libwtf.so] [0x00ad1] [       3a 49 ] 0x40000ad0: ldr r1, [pc, #0xe8]
    [   libwtf.so] [0x00ad3] [ 57 f8 24 2c ] 0x40000ad2: ldr r2, [r7, #-0x24]
    [   libwtf.so] [0x00ad7] [       79 44 ] 0x40000ad6: add r1, pc
    [   libwtf.so] [0x00ad9] [       09 68 ] 0x40000ad8: ldr r1, [r1]
    [   libwtf.so] [0x00adb] [       09 68 ] 0x40000ada: ldr r1, [r1]
```

或者填入两个nop也行

[Online ARM to HEX Converter (armconverter.com)]



```java
    }

    public void patchLog(){
        int patchCode = 0xbf00bf00;
        emulator.getMemory().pointer( address: module.base + 0xABE).setInt( offset: 0,patchCode);
    }

}
```

```
JNIEnv->GetStringUtfChars("0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000995[libwtf.so]0x995
JNIEnv->GetStringUtfChars("") was called from RX@0x400009a5[libwtf.so]0x9a5
JNIEnv->GetStringUtfChars("") was called from RX@0x400009b5[libwtf.so]0x9b5
JNIEnv->GetStringUtfChars("1628093856262") was called from RX@0x400009c9[libwtf.so]0x9c9
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a6b[libwtf.so]0xa6b
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a7b[libwtf.so]0xa7b
JNIEnv->ReleaseStringUTFChars("1628093856262") was called from RX@0x40000a8b[libwtf.so]0xa8b
JNIEnv->ReleaseStringUTFChars("0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000a9b[libwtf.so]0xa9b
debugger break at: 0x40000abe
>>> r0=0x6 r1=0x40000bb0 r2=0x40002d58 r3=0x40172000 r4=0xfffe0b40 r5=0xbffff733 r6=0x40173000 r7=0xbffff788 r8=0x40175000 sb=0xbffff6e8 sl=0x40174000 fp=0x40172000 i
>>> SP=0xbffff6d0 LR=null PC=RX@0x40000abe[libwtf.so]0xabe cpsr: N=0, Z=0, C=1, V=0, T=1, mode=0b10000
=> *[   libwtf.so]*[0x00abf]*[*      00 bf ]*0x40000abe:*nop
    [   libwtf.so] [0x00ac1] [       00 bf ] 0x40000ac0: nop
    [   libwtf.so] [0x00ac3] [       06 b0 ] 0x40000ac2: add sp, #0x18
    [   libwtf.so] [0x00ac5] [       20 68 ] 0x40000ac4: ldr r0, [r4]
    [   libwtf.so] [0x00ac7] [       29 46 ] 0x40000ac6: mov r1, r5
```

再看第二种办法，由Unibdg封装，汇编功能由KeyStone提供，也一切正常。

```java
package com.cbgc;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Emulator;
import com.github.unidbg.Module;
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.*;
import com.github.unidbg.memory.Memory;
import com.github.unidbg.pointer.UnidbgPointer;
import com.sun.jna.Pointer;
import keystone.Keystone;
import keystone.KeystoneArchitecture;
import keystone.KeystoneEncoded;
import keystone.KeystoneMode;

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

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

    public WTF() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .setProcessName("com.cbgc")
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        // 创建Android虚拟机,传入APK，Unidbg可以替我们处理许多问题
        File apkFile = new File("unidbg-
android/src/test/resources/CBGC/cbgc.apk");
        vm = emulator.createDalvikVM(apkFile);
        // 加载目标SO
        File file = new File("unidbg-
android/src/test/resources/CBGC/libwtf.so");
        DalvikModule dm = vm.loadLibrary(file, true);

        module = dm.getModule();
        vm.setJni(this); // 设置JNI
        vm.setVerbose(true); // 打印日志

        dm.callJNI_OnLoad(emulator);// 调用JNI OnLoad

        emulator.attach().addBreakPoint(module.base+0xabe);
    }


    public static void main(String[] args) {
        WTF wtf = new WTF();
//          wtf.patchLog();
        wtf.patchLog1();
        System.out.println(wtf.getSign());
    }


    public String getSign() {
        List<Object> list = new ArrayList<>(10);
        // arg1 env
        list.add(vm.getJNIEnv());
        // arg2 jobject/jclazz 一般用不到，直接填0
        list.add(0);

        list.add(vm.addLocalObject(new StringObject(vm, "")));
        list.add(vm.addLocalObject(new StringObject(vm, "")));
        list.add(vm.addLocalObject(new StringObject(vm, "1628093856262")));
        // 参数准备完成
        // call function
        Number number = module.callFunction(emulator, 0x931, list.toArray())[0];
        String result = vm.getObject(number.intValue()).getValue().toString();
        return result;
    }


    @Override
    public DvmObject<?> callStaticObjectMethodV(BaseVM vm, DvmClass dvmClass,
String signature, VaList vaList) {
        if (signature.equals("com/sichuanol/cbgc/util/LogShutDown-
>getAppSign()Ljava/lang/String;")){
            return new StringObject(vm, "0093CB6721DAF15D31CFBC9BBE3A2B79");
        }
        return super.callStaticObjectMethodV(vm, dvmClass, signature, vaList);
    }

    public void patchLog(){
        int patchCode = 0xbf00bf00;
        emulator.getMemory().pointer(module.base + 0xABE).setInt(0,patchCode);
    }

    public void patchLog1(){
        Pointer pointer = UnidbgPointer.pointer(emulator, module.base + 0xABE);
        assert pointer != null;
        try (Keystone keystone = new Keystone(KeystoneArchitecture.Arm,
KeystoneMode.ArmThumb)) {
            KeystoneEncoded encoded = keystone.assemble("nop");
            byte[] patch = encoded.getMachineCode();
            pointer.write(0, patch, 0, patch.length);
            pointer.write(2, patch, 0, patch.length);
        }
    };


}
```

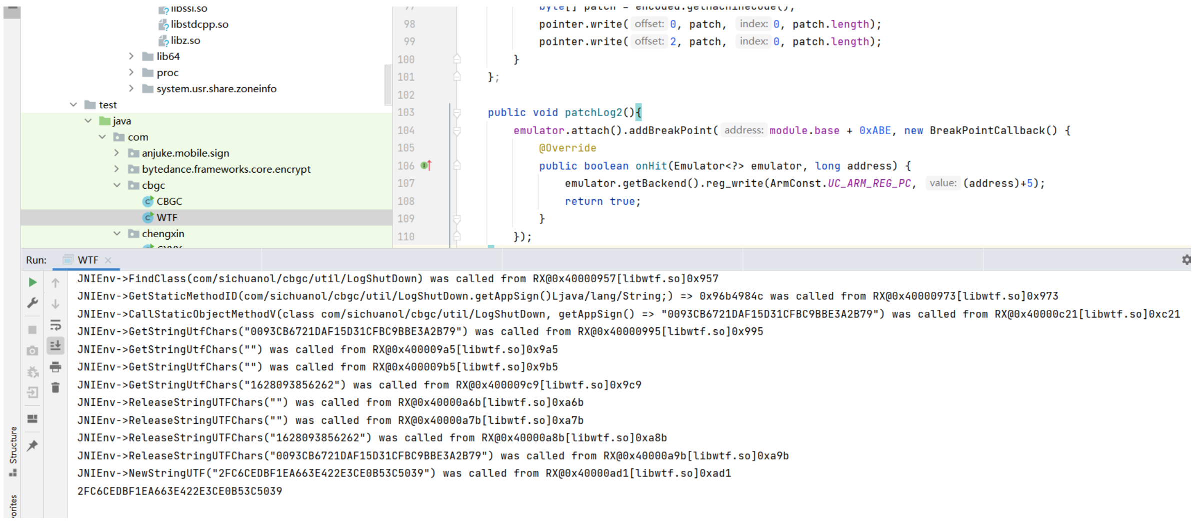
第三种办法呢？前两种办法都是静态打patch，和你用IDA的KeyPatch直接打patch没啥区别。有没有办法玩个花样呢？

```
public void patchLog2(){
    emulator.attach().addBreakPoint(module.base + 0xABE, new
BreakPointCallback() {
        @Override
        public boolean onHit(Emulator<?> emulator, long address) {
            emulator.getBackend().reg_write(ArmConst.UC_ARM_REG_PC,
(address)+5);
            return true;
        }
    });
}
```

解释一下做了什么，我们在log的调用处下了一个断点，inline hook 都是在调用前断下来了，所以在执行前断了下来，然后加了BreakPointCallback回调，在这个时机做处理，我们将PC指针直接往前加了5，PC指向哪里，函数就执行哪里，本来PC指向“blx log”这个地址，程序即将去执行log函数。但我们直接将PC加了5，为什么加5？

我们知道这里的log是个坑，它长四个字节，我们要越过这个坑，但加4不够，我们是thumb模式，再+1，所以就是+5。

除此之外，OnHit命中断点时返回true，true表示不用断下来变成调试模式，继续往下走。

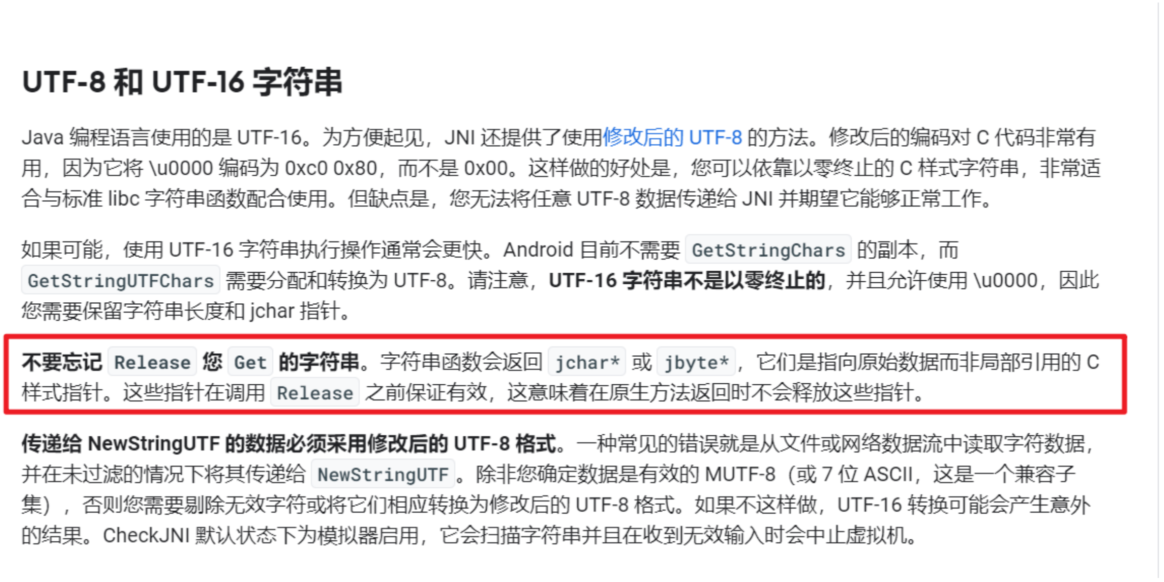


```
                pointer.write( offset: 0, patch,  index: 0, patch.length);
                pointer.write( offset: 2, patch,  index: 0, patch.length);
            }
        };

    public void patchLog2(){
        emulator.attach().addBreakPoint( address: module.base + 0xABE, new BreakPointCallback() {
            @Override
            public boolean onHit(Emulator<?> emulator, long address) {
                emulator.getBackend().reg_write(ArmConst.UC_ARM_REG_PC,  value: (address)+5);
                return true;
            }
        });
```

```
JNIEnv->FindClass(com/sichuanol/cbgc/util/LogShutDown) was called from RX@0x40000957[libwtf.so]0x957
JNIEnv->GetStaticMethodID(com/sichuanol/cbgc/util/LogShutDown.getAppSign()Ljava/lang/String;) => 0x96b4984c was called from RX@0x40000973[libwtf.so]0x973
JNIEnv->CallStaticObjectMethodV(class com/sichuanol/cbgc/util/LogShutDown, getAppSign() => "0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000c21[libwtf.so]0xc21
JNIEnv->GetStringUtfChars("0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000995[libwtf.so]0x995
JNIEnv->GetStringUtfChars("") was called from RX@0x400009a5[libwtf.so]0x9a5
JNIEnv->GetStringUtfChars("") was called from RX@0x400009b5[libwtf.so]0x9b5
JNIEnv->GetStringUtfChars("1628093856262") was called from RX@0x400009c9[libwtf.so]0x9c9
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a6b[libwtf.so]0xa6b
JNIEnv->ReleaseStringUTFChars("") was called from RX@0x40000a7b[libwtf.so]0xa7b
JNIEnv->ReleaseStringUTFChars("1628093856262") was called from RX@0x40000a8b[libwtf.so]0xa8b
JNIEnv->ReleaseStringUTFChars("0093CB6721DAF15D31CFBC9BBE3A2B79") was called from RX@0x40000a9b[libwtf.so]0xa9b
JNIEnv->NewStringUTF("2FC6CEDBF1EA663E422E3CE0B53C5039") was called from RX@0x40000ad1[libwtf.so]0xad1
2FC6CEDBF1EA663E422E3CE0B53C5039
```

同样是可以的。

问题就这么解决了，事实上，由于开发人员代码编写不严谨，而导致Unidbg跑不通并不是个例，比如按照开发规范，Get的字符串应该用对应的Release系列API进行释放，但有的开发人员会用free函数去释放这片内存，这是违反开发规范的操作，也会导致报错，遇到这种情况需要Hook 对应的"对free函数调用"。



## UTF-8 和 UTF-16 字符串

Java 编程语言使用的是 UTF-16。为方便起见，JNI 还提供了使用修改后的 UTF-8 的方法。修改后的编码对 C 代码非常有用，因为它将 \u0000 编码为 0xc0 0x80，而不是 0x00。这样做的好处是，您可以依靠以零终止的 C 样式字符串，非常适合与标准 libc 字符串函数配合使用。但缺点是，您无法将任意 UTF-8 数据传递给 JNI 并期望它能够正常工作。

如果可能，使用 UTF-16 字符串执行操作通常会更快。Android 目前不需要 `GetStringChars` 的副本，而 `GetStringUTFChars` 需要分配和转换为 UTF-8。请注意，**UTF-16 字符串不是以零终止的**，并且允许使用 \u0000，因此您需要保留字符串长度和 jchar 指针。

**不要忘记 `Release` 您 `Get` 的字符串**。字符串函数会返回 `jchar*` 或 `jbyte*`，它们是指向原始数据而非局部引用的 C 样式指针。这些指针在调用 `Release` 之前保证有效，这意味着在原生方法返回时不会释放这些指针。

**传递给 NewStringUTF 的数据必须采用修改后的 UTF-8 格式**。一种常见的错误就是从文件或网络数据流中读取字符数据，并在未过滤的情况下将其传递给 `NewStringUTF`。除非您确定数据是有效的 MUTF-8（或 7 位 ASCII，这是一个兼容子集），否则您需要剔除无效字符或将它们相应转换为修改后的 UTF-8 格式。如果不这样做，UTF-16 转换可能会产生意外的结果。CheckJNI 默认状态下为模拟器启用，它会扫描字符串并且在收到无效输入时会中止虚拟机。

Unidbg的issue区就有这样一个案例

[GetStringUTFChars获得的char* 无法用free释放, Invalid address xxxx passed to free: value not allocated · Issue #210 · zhkl0228/unidbg (github.com)]