一、前言

*methodID*问题是*Unidbg* 补*JAVA*环境中最糟糕的问题，毫不夸张的说，有效利用它可以挡住*50%*的*Unidbg*使用者。

二、描述和实现

以 *CallObjectMethod* 为例，可用于调用一个返回*object*的实例方法，最后一个参数是可变参数，对应于所调用方法的参数。

```
NativeType Call<type>Method(JNIEnv *env, jobject obj, jmethodID methodID, ...);
```

需要实例+方法ID+参数列表，我写一个简单的demo表达这个逻辑

MainActivity.java

```
package com.example.getmethodid;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

import com.example.getmethodid.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

    // Used to load the 'getmethodid' library on application startup.
    static {
        System.loadLibrary("getmethodid");
    }

    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        // Example of a call to a native method
        TextView tv = binding.sampleText;
        tv.setText(stringFromJNI());
    }

    /**
     * A native method that is implemented by the 'getmethodid' native library,
     * which is packaged with this application.
     */
    public native String stringFromJNI();
```

```
    public String getName() {
        return "lilac";
    }
}
```

native-lib.cpp

```
#include <jni.h>
#include <string>

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

    jclass mainactivity_clazz = env->GetObjectClass(mainactivity);
    jmethodID methodID = env->GetMethodID(mainactivity_clazz, "getName", "
()Ljava/lang/String;");
    auto j_result = (jstring)env->CallObjectMethod(mainactivity, methodID);
    const char* c_result = env->GetStringUTFChars(j_result, nullptr);
    return env->NewStringUTF(c_result);
}
```

接下来看Unidbg是如何处理这个问题的

```
package com.antiUnidbg;

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.*;
import com.github.unidbg.memory.Memory;

import java.io.File;

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

    showMethod() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/methodproblem/app-debug.apk"));
        vm.setJni(this);
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/methodproblem/libgetmethodid.so"), true);
        module = dm.getModule();
        vm.setVerbose(true); // 打印日志
```

```
        // 添加类到《不存在的类列表》
        vm.addNotFoundClass("my/fake/class");
    };

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

    public void call(){
        DvmClass dvmClass =
vm.resolveClass("com/example/getmethodid/MainActivity");
        String methodSign = "stringFromJNI()Ljava/lang/String;";
        DvmObject<?> dvmObject = dvmClass.newObject(null);

        StringObject obj = dvmObject.callJniMethodObject(emulator, methodSign);
        System.out.println(obj);
    }
}
```

运行

```
Run: showMethod
C:\Users\13352\.jdks\openjdk-16.0.2\bin\java.exe ...
Find native function Java_com_example_getmethodid_MainActivity_stringFromJNI => RX@0x40000939[libgetmethodid.so]0x939
JNIEnv->GetMethodID(com/example/getmethodid/MainActivity.getName()Ljava/lang/String;) => 0x15d46143 was called from RX@0x400009d7[libgetmethodid.so]0x9d7
[09:16:53 722]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=-1073744020, svcNumber=0x11f, PC=unidbg@0xfffe0284, LR=RX@0x40000a2b[
java.lang.UnsupportedOperationException Create breakpoint : com/example/getmethodid/MainActivity->getName()Ljava/lang/String;
    at com.github.unidbg.linux.android.dvm.AbstractJni.callObjectMethodV(AbstractJni.java:368)
    at com.github.unidbg.linux.android.dvm.AbstractJni.callObjectMethodV(AbstractJni.java:236)
    at com.github.unidbg.linux.android.dvm.DvmMethod.callObjectMethodV(DvmMethod.java:89)
    at com.github.unidbg.linux.android.dvm.DalvikVM$32.handle(DalvikVM.java:527)
    at com.github.unidbg.linux.ARM32SyscallHandler.hook(ARM32SyscallHandler.java:103)
    at com.github.unidbg.arm.backend.UnicornBackend$6.hook(UnicornBackend.java:305)
    at unicorn.Unicorn$NewHook.onInterrupt(Unicorn.java:128)
    at unicorn.Unicorn.emu_start(Native Method)
    at com.github.unidbg.arm.backend.UnicornBackend.emu_start(UnicornBackend.java:331)
    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.android.dvm.DvmObject.callJniMethod(DvmObject.java:133)
    at com.github.unidbg.linux.android.dvm.DvmObject.callJniMethodObject(DvmObject.java:93)
    at com.antiUnidbg.showMethod.call(showMethod.java:43)
    at com.antiUnidbg.showMethod.main(showMethod.java:35)
[09:16:53 725]  WARN [com.github.unidbg.AbstractEmulator] (AbstractEmulator:389) - emulate RX@0x40000939[libgetmethodid.so]0x939 exception sp=unidbg@0xbffff740, msg=com/example/getmethodid/Mai
```

按照提示补上对应方法

```
@Override
public DvmObject<?> callObjectMethodV(BaseVM vm, DvmObject<?> dvmObject, String
signature, VaList vaList) {
    switch (signature){
        case "com/example/getmethodid/MainActivity-
>getName()Ljava/lang/String;":{
            return new StringObject(vm, "lilac");
        }
    }
    return super.callObjectMethodV(vm, dvmObject, signature, vaList);
}
```

```
Run: showMethod
C:\Users\13352\.jdks\openjdk-16.0.2\bin\java.exe ...
Find native function Java_com_example_getmethodid_MainActivity_stringFromJNI => RX@0x40000939[libgetmethodid.so]0x939
JNIEnv->GetMethodID(com/example/getmethodid/MainActivity.getName()Ljava/lang/String;) => 0x15d46143 was called from RX@0x400009d7[libgetmethodid.so]0x9d7
JNIEnv->CallObjectMethodV(com.example.getmethodid.MainActivity@69379752, getName() => "lilac") was called from RX@0x40000a2b[libgetmethodid.so]0xa2b
JNIEnv->GetStringUtfChars("lilac") was called from RX@0x40000a73[libgetmethodid.so]0xa73
JNIEnv->NewStringUTF("lilac") was called from RX@0x40000a93[libgetmethodid.so]0xa93
"lilac"
```

一切都很顺利，没有问题就要制造问题？下面我们可能会显得跑题，但一切都围绕着一个主旨——**Unidbg对进程的JAVA世界一无所知。**

看一下 *GetMethodID* 的官方文档

> jmethodID GetMethodID(JNIEnv *env, jclass clazz,const char *name, const char *sig);
>
> Returns the method ID for an instance (nonstatic) method of a class or interface. The method may be defined in one of the `clazz`'s superclasses and inherited by `clazz`. The method is determined by its name and signature.
>
> 返回类或接口的实例（非静态）方法的方法ID。该方法可以在clazz的一个父类中定义，并由clazz继承。方法由其名称和签名决定。
>
> `GetMethodID()` causes an uninitialized class to be initialized.
>
> GetMethodID()会给未初始化的类做初始化。

一共两段话，可以构造许多甜蜜陷阱了，这篇文章里，我们关注第一段，第二段我们放到下篇里。首先我们考虑，子类调用父类方法的情况

MainActivity.java

```
package com.example.getmethodidexample0;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import com.example.getmethodidexample0.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

    // Used to load the 'getmethodidexample0' library on application startup.
    static {
        System.loadLibrary("getmethodidexample0");
    }

    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        TextView tv = binding.sampleText;
        tv.setText(stringFromJNI());
    }

    public native String stringFromJNI();
}
```

Phone.java

```java
package com.example.getmethodidexample0;

public class Phone {
    public String getPrice(){
        return "价格不清楚";
    }

    public String type(){
        return "手机";
    }
}
```

XiaoMi.java

```java
package com.example.getmethodidexample0;

public class XiaoMi extends Phone{

    @Override
    public String getPrice(){
        return "1999";
    }

}
```

接下来我们在Native中初始化一个XiaoMi实例，调用type方法。

```cpp
#include <jni.h>
#include <string>

extern "C" JNIEXPORT jstring JNICALL
Java_com_example_getmethodidexample0_MainActivity_stringFromJNI(
        JNIEnv* env,
        jobject /* this */) {

    jclass xiaomi_clz = env-
>FindClass("com/example/getmethodidexample0/XiaoMi");

    jmethodID init = env->GetMethodID(xiaomi_clz, "<init>", "()V");
    jobject xiaomiObject = env->NewObject(xiaomi_clz, init);

    jmethodID type = env->GetMethodID(xiaomi_clz, "type", "
()Ljava/lang/String;");
    auto j_type = (jstring)env->CallObjectMethod(xiaomiObject, type);
    const char* c_type = env->GetStringUTFChars(j_type, nullptr);

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

测试机运行

接下来Unidbg模拟执行

getmethodidexample1.java

```
package com.antiUnidbg;

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.*;
import com.github.unidbg.memory.Memory;

import java.io.File;

public class getmethodidexample1 extends AbstractJni {
```

```java
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    getmethodidexample1() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/getmethodidexample1/app-debug.apk"));
        vm.setJni(this);
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/getmethodidexample1/libgetmethodidexample0.so"),
true);
        module = dm.getModule();
        vm.setVerbose(true); // 打印日志

    };

    public static void main(String[] args) {
        getmethodidexample1 demo = new getmethodidexample1();
        demo.call();
    }

    public void call(){
        DvmClass dvmClass =
vm.resolveClass("com/example/getmethodidexample0/MainActivity");
        String methodSign = "stringFromJNI()Ljava/lang/String;";
        DvmObject<?> dvmObject = dvmClass.newObject(null);

        StringObject obj = dvmObject.callJniMethodObject(emulator, methodSign);
        System.out.println(obj);
    }

}
```

运行

```
C:\Users\13352\.jdks\openjdk-16.0.2\bin\java.exe ...
Find native function Java_com_example_getmethodidexample0_MainActivity_stringFromJNI => RX@0x40000995[libgetmethodidexample0.so]0x995
JNIEnv->FindClass(com/example/getmethodidexample0/XiaoMi) was called from RX@0x40000a2f[libgetmethodidexample0.so]0xa2f
JNIEnv->GetMethodID(com/example/getmethodidexample0/XiaoMi.<init>()V) => 0xe1209b2d was called from RX@0x40000a5f[libgetmethodidexample0.so]0xa5f
[12:20:09 855]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=-1073744028, svcNumber=0x119, PC=unidbg@0xfffe0224, LR=RX@
java.lang.UnsupportedOperationException Create breakpoint : com/example/getmethodidexample0/XiaoMi-><init>()V
    at com.github.unidbg.linux.android.dvm.AbstractJni.newObjectV(AbstractJni.java:683)
    at com.github.unidbg.linux.android.dvm.AbstractJni.newObjectV(AbstractJni.java:654)
    at com.github.unidbg.linux.android.dvm.DvmMethod.newObjectV(DvmMethod.java:213)
    at com.github.unidbg.linux.android.dvm.DalvikVM$26.handle(DalvikVM.java:407)
    at com.github.unidbg.linux.ARM32SyscallHandler.hook(ARM32SyscallHandler.java:103)
    at com.github.unidbg.arm.backend.UnicornBackend$6.hook(UnicornBackend.java:305)
    at unicorn.Unicorn$NewHook.onInterrupt(Unicorn.java:128)
    at unicorn.Unicorn.emu_start(Native Method)
    at com.github.unidbg.arm.backend.UnicornBackend.emu_start(UnicornBackend.java:331)
    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.android.dvm.DvmObject.callJniMethod(DvmObject.java:133)
    at com.github.unidbg.linux.android.dvm.DvmObject.callJniMethodObject(DvmObject.java:93)
    at com.antiUnidbg.getmethodidexample1.call(getmethodidexample1.java:41)
    at com.antiUnidbg.getmethodidexample1.main(getmethodidexample1.java:33)
[12:20:09 858]  WARN [com.github.unidbg.AbstractEmulator] (AbstractEmulator:389) - emulate RX@0x40000995[libgetmethodidexample0.so]0x995 exception sp=unidbg@0xbffff738, msg=com/exam
null
```

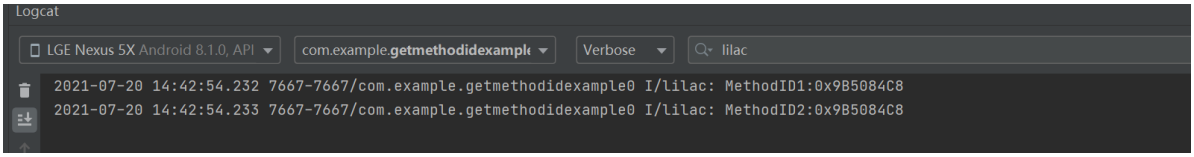
看起来一切顺利，补 *com/example/getmethodidexample0/XiaoMi-><init>()V* 实现即可，但实际上已经出了大问题。

> Unidbg对进程的JAVA世界一无所知

它并不知道这个方法继承自父类，我们重新改一下*native-lib.cpp*

```
#include <jni.h>
#include <string>
#include <android/log.h>
#define TAG "lilac"
// 定义info信息
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO,TAG,__VA_ARGS__)

extern "C" JNIEXPORT jstring JNICALL
Java_com_example_getmethodidexample0_MainActivity_stringFromJNI(
        JNIEnv* env,
        jobject /* this */) {

    jclass xiaomi_clz = env-
>FindClass("com/example/getmethodidexample0/XiaoMi");
    jmethodID methodID1 = env->GetMethodID(xiaomi_clz, "type", "
()Ljava/lang/String;");
    jclass phone_clz = env->FindClass("com/example/getmethodidexample0/Phone");
    jmethodID methodID2 = env->GetMethodID(phone_clz, "type", "
()Ljava/lang/String;");

    LOGI("MethodID1:0x%08X", methodID1);
    LOGI("MethodID2:0x%08X", methodID2);
    return env->NewStringUTF("");
}
```

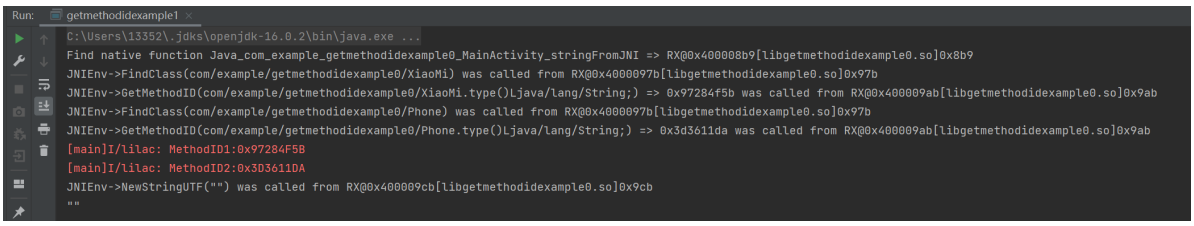
分别通过xiaomi与phone类获取type方法ID，并打印。因为xiaomi的type方法本就继承自phone，所以ID一致（因为xiaomi获取的就是phone的type方法嘛）。



在Unidbg中呢，我想结果应该不言而喻了，Unidbg对进程的JAVA世界一无所知，它不知道XiaoMi的type方法继承自父类Phone，而以为是两个不同类的不同方法，那么方法ID自然不同，在Unidbg中验证一下

```
package com.antiUnidbg;

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.*;
import com.github.unidbg.memory.Memory;

import java.io.File;

public class getmethodidexample1 extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;
```

```
    getmethodidexample1() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/getmethodidexample2/app-debug.apk"));
        vm.setJni(this);
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/getmethodidexample2/libgetmethodidexample0.so"),
true);
        module = dm.getModule();
        vm.setVerbose(true); // 打印日志

    };

    public static void main(String[] args) {
        getmethodidexample1 demo = new getmethodidexample1();
        demo.call();
    }

    public void call(){
        DvmClass dvmClass =
vm.resolveClass("com/example/getmethodidexample0/MainActivity");
        String methodSign = "stringFromJNI()Ljava/lang/String;";
        DvmObject<?> dvmObject = dvmClass.newObject(null);

        StringObject obj = dvmObject.callJniMethodObject(emulator, methodSign);
        System.out.println(obj);
    }

}
```



方法ID 确实不同，这是第一个可供检测的点，但我们不能止步于此，可以顺着methodID这个问题继续发散，因为XiaoMi的type方法本就来自Phone，所以我们可以不绕弯子，直接通过父类Phone类去获取ID

```
#include <jni.h>
#include <string>

extern "C" JNIEXPORT jstring JNICALL
Java_com_example_getmethodidexample0_MainActivity_stringFromJNI(
        JNIEnv* env,
        jobject /* this */) {

    jclass xiaomi_clz = env-
>FindClass("com/example/getmethodidexample0/XiaoMi");
    jclass phone_clz = env->FindClass("com/example/getmethodidexample0/Phone");
```

```
    jmethodID init = env->GetMethodID(xiaomi_clz, "<init>", "()V");
    jobject xiaomiObject = env->NewObject(xiaomi_clz, init);

    jmethodID type = env->GetMethodID(phone_clz, "type", "
()Ljava/lang/String;");
    auto j_type = (jstring)env->CallObjectMethod(xiaomiObject, type);
    const char* c_type = env->GetStringUTFChars(j_type, nullptr);

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

即xiaomi对象调用从phone类 取得的type方法，下面测试在Unidbg中模拟执行

```
package com.antiUnidbg;

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.*;
import com.github.unidbg.memory.Memory;

import java.io.File;

public class getmethodidexample1 extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    getmethodidexample1() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/getmethodidexample3/app-debug.apk"));
        vm.setJni(this);
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/getmethodidexample3/libgetmethodidexample0.so"),
true);
        module = dm.getModule();
        vm.setVerbose(true); // 打印日志

    };

    public static void main(String[] args) {
        getmethodidexample1 demo = new getmethodidexample1();
        demo.call();
    }

    public void call(){
        DvmClass dvmClass =
vm.resolveClass("com/example/getmethodidexample0/MainActivity");
        String methodSign = "stringFromJNI()Ljava/lang/String;";
        DvmObject<?> dvmObject = dvmClass.newObject(null);
```

```
        StringObject obj = dvmObject.callJniMethodObject(emulator, methodSign);
        System.out.println(obj);
    }

}
```

运行

```
Run: getmethodidexample1
C:\Users\13352\.jdks\openjdk-16.0.2\bin\java.exe ...
Find native function Java_com_example_getmethodidexample0_MainActivity_stringFromJNI => RX@0x40000995[libgetmethodidexample0.so]0x995
JNIEnv->FindClass(com/example/getmethodidexample0/XiaoMi) was called from RX@0x40000a3f[libgetmethodidexample0.so]0xa3f
JNIEnv->FindClass(com/example/getmethodidexample0/Phone) was called from RX@0x40000a3f[libgetmethodidexample0.so]0xa3f
JNIEnv->GetMethodID(com/example/getmethodidexample0/XiaoMi.<init>()V) => 0xe1209b2d was called from RX@0x40000a6f[libgetmethodidexample0.so]0xa6f
[13:33:34 484]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=-1073744036, svcNumber=0x119, PC=un
java.lang.UnsupportedOperationException Create breakpoint : com/example/getmethodidexample0/XiaoMi-><init>()V
    at com.github.unidbg.linux.android.dvm.AbstractJni.newObjectV(AbstractJni.java:683)
    at com.github.unidbg.linux.android.dvm.AbstractJni.newObjectV(AbstractJni.java:654)
    at com.github.unidbg.linux.android.dvm.DvmMethod.newObjectV(DvmMethod.java:213)
    at com.github.unidbg.linux.android.dvm.DalvikVM$26.handle(DalvikVM.java:407)
    at com.github.unidbg.linux.ARM32SyscallHandler.hook(ARM32SyscallHandler.java:103)
    at com.github.unidbg.arm.backend.UnicornBackend$6.hook(UnicornBackend.java:305)
    at unicorn.Unicorn$NewHook.onInterrupt(Unicorn.java:128)
    at unicorn.Unicorn.emu_start(Native Method)
    at com.github.unidbg.arm.backend.UnicornBackend.emu_start(UnicornBackend.java:331)
    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)
```

正常补一下

```
@Override
public DvmObject<?> newObjectV(BaseVM vm, DvmClass dvmClass, String signature,
VaList vaList) {
    switch (signature){
        case "com/example/getmethodidexample0/XiaoMi-><init>()V":{
            return
vm.resolveClass("com/example/getmethodidexample0/XiaoMi").newObject(null);
        }
    }
    return super.newObjectV(vm, dvmClass, signature, vaList);
}
```

继续运行

```
Run: getmethodidexample1
C:\Users\13352\.jdks\openjdk-16.0.2\bin\java.exe ...
Find native function Java_com_example_getmethodidexample0_MainActivity_stringFromJNI => RX@0x40000995[libgetmethodidexample0.so]0x995
JNIEnv->FindClass(com/example/getmethodidexample0/XiaoMi) was called from RX@0x40000a3f[libgetmethodidexample0.so]0xa3f
JNIEnv->FindClass(com/example/getmethodidexample0/Phone) was called from RX@0x40000a3f[libgetmethodidexample0.so]0xa3f
JNIEnv->GetMethodID(com/example/getmethodidexample0/XiaoMi.<init>()V) => 0xe1209b2d was called from RX@0x40000a6f[libgetmethodidexample0.so]0xa6f
JNIEnv->NewObjectV(class com/example/getmethodidexample0/XiaoMi, <init>() => com.example.getmethodidexample0.XiaoMi@4f51b3e0) was called from RX@0x40000ac1[libgetmethodidexample0.so]0xac1
JNIEnv->GetMethodID(com/example/getmethodidexample0/Phone.type()Ljava/lang/String;) => 0x3d3611da was called from RX@0x40000a6f[libgetmethodidexample0.so]0xa6f
[13:35:36 298]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=-1073744036, svcNumber=0x11f, PC=unidbg@0xfffe0284, LR=RX@0x40000b37[libge
com.github.unidbg.arm.backend.BackendException Create breakpoint : dvmObject=com.example.getmethodidexample0.XiaoMi@4f51b3e0, dvmClass=class com/example/getmethodidexample0/XiaoMi, jmethodID=unidbg@0x3
    at com.github.unidbg.linux.android.dvm.DalvikVM$32.handle(DalvikVM.java:524)
    at com.github.unidbg.linux.ARM32SyscallHandler.hook(ARM32SyscallHandler.java:103)
    at com.github.unidbg.arm.backend.UnicornBackend$6.hook(UnicornBackend.java:305)
    at unicorn.Unicorn$NewHook.onInterrupt(Unicorn.java:128)
    at unicorn.Unicorn.emu_start(Native Method)
    at com.github.unidbg.arm.backend.UnicornBackend.emu_start(UnicornBackend.java:331)
    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.android.dvm.DvmObject.callJniMethod(DvmObject.java:133)
    at com.github.unidbg.linux.android.dvm.DvmObject.callJniMethodObject(DvmObject.java:93)
    at com.antiUnidbg.getmethodidexample1.call(getmethodidexample1.java:41)
    at com.antiUnidbg.getmethodidexample1.main(getmethodidexample1.java:33)
[13:35:36 301]  WARN [com.github.unidbg.AbstractEmulator] (AbstractEmulator:389) - emulate RX@0x40000995[libgetmethodidexample0.so]0x995 exception sp=unidbg@0xbffff730, msg=dvmObject=com.example.ge
null
```

产生了报错，这个报错点进去你会发现，是Unidbg找不到方法，这是为什么？

```
jmethodID GetMethodID(JNIEnv *env, jclass clazz,const char *name, const char
*sig);
```

我们看一下Unidbg的实现

```
Pointer _GetMethodID = svcMemory.registerSvc(new ArmSvc() {
```

```
    @Override
    public long handle(Emulator<?> emulator) {
        RegisterContext context = emulator.getContext();
        UnidbgPointer clazz = context.getPointerArg(1);
        Pointer methodName = context.getPointerArg(2);
        Pointer argsPointer = context.getPointerArg(3);
        String name = methodName.getString(0);
        String args = argsPointer.getString(0);
        if (log.isDebugEnabled()) {
            log.debug("GetMethodID class=" + clazz + ", methodName=" + name + ",
args=" + args + ", LR=" + context.getLRPointer());
        }
        DvmClass dvmClass = classMap.get(clazz.toIntPeer());
        if (dvmClass == null) {
            throw new BackendException();
        } else {
            int hash = dvmClass.getMethodID(name, args);
            if (verbose && hash != 0) {
                System.out.printf("JNIEnv->GetMethodID(%s.%s%s) => 0x%x was
called from %s%n", dvmClass.getClassName(), name, args, hash & 0xffffffffL,
context.getLRPointer());
            }
            return hash;
        }
    }
});

int getMethodID(String methodName, String args) {
    String signature = getClassName() + "->" + methodName + args;
    int hash = signature.hashCode();
    if (log.isDebugEnabled()) {
        log.debug("getMethodID signature=" + signature + ", hash=0x" +
Long.toHexString(hash));
    }
    if (vm.jni == null || vm.jni.acceptMethod(this, signature, false)) {
        if (!methodMap.containsKey(hash)) {
            methodMap.put(hash, new DvmMethod(this, methodName, args, false));
        }
        return hash;
    } else {
        return 0;
    }
}
```

即每个类有一个methodMap，getMethodID时，Unidbg将查找的方法添加到Map里。结合上面我们的代码，产生了一个问题——Xiaomi在执行方法时，找不到方法ID，方法ID在Phone的methodMap里。

Unidbg考虑过这个情况，我们需要提前声明XiaoMI和Phone类的继承关系，这样的话，如果方法ID在当前类的methodMap里找不到，Unidbg就去超类的methodMap里找，修改Unidbg代码如下

```
package com.antiUnidbg;

import com.github.unidbg.AndroidEmulator;
import com.github.unidbg.Module;
import com.github.unidbg.linux.android.AndroidEmulatorBuilder;
import com.github.unidbg.linux.android.AndroidResolver;
```

```java
import com.github.unidbg.linux.android.dvm.*;
import com.github.unidbg.memory.Memory;

import java.io.File;

public class getmethodidexample1 extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    getmethodidexample1() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/getmethodidexample3/app-debug.apk"));
        vm.setJni(this);
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/getmethodidexample3/libgetmethodidexample0.so"),
true);
        module = dm.getModule();
        vm.setVerbose(true); // 打印日志

        // 声明XiaoMi继承自Phone
        vm.resolveClass("com/example/getmethodidexample0/XiaoMi",
vm.resolveClass("com/example/getmethodidexample0/Phone"));

    };

    public static void main(String[] args) {
        getmethodidexample1 demo = new getmethodidexample1();
        demo.call();
    }

    public void call(){
        DvmClass dvmClass =
vm.resolveClass("com/example/getmethodidexample0/MainActivity");
        String methodSign = "stringFromJNI()Ljava/lang/String;";
        DvmObject<?> dvmObject = dvmClass.newObject(null);

        StringObject obj = dvmObject.callJniMethodObject(emulator, methodSign);
        System.out.println(obj);
    }

    @Override
    public DvmObject<?> newObjectV(BaseVM vm, DvmClass dvmClass, String
signature, VaList vaList) {
        switch (signature){
            case "com/example/getmethodidexample0/XiaoMi-><init>()V":{
                return
vm.resolveClass("com/example/getmethodidexample0/XiaoMi").newObject(null);
            }
        }
        return super.newObjectV(vm, dvmClass, signature, vaList);
    }
}
```

即可回归正常补代码逻辑

```
Run: getmethodidexample1
JNIEnv->FindClass(com/example/getmethodidexample0/Phone) was called from RX@0x40000a3f[libgetmethodidexample0.so]0xa3f
JNIEnv->GetMethodID(com/example/getmethodidexample0/XiaoMi.<init>()V) => 0xe1209b2d was called from RX@0x40000a6f[libgetmethodidexample0.so]0xa6f
JNIEnv->NewObjectV(class com/example/getmethodidexample0/XiaoMi, <init>() => com.example.getmethodidexample0.XiaoMi@4f51b3e0) was called from RX@0x40000ac1[libgetmethodidexample0.so]0xac1
JNIEnv->GetMethodID(com/example/getmethodidexample0/Phone.type()Ljava/lang/String;) => 0x3d3611da was called from RX@0x40000a6f[libgetmethodidexample0.so]0xa6f
[13:53:22 520]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno=2, NR=-1073744036, svcNumber=0x11f, PC=unidbg@0xfffe0284, LR=RX@0x40000b37[libg
java.lang.UnsupportedOperationException Create breakpoint : com/example/getmethodidexample0/Phone->type()Ljava/lang/String;
    at com.github.unidbg.linux.android.dvm.AbstractJni.callObjectMethodV(AbstractJni.java:368)
    at com.github.unidbg.linux.android.dvm.AbstractJni.callObjectMethodV(AbstractJni.java:235)
    at com.github.unidbg.linux.android.dvm.DvmMethod.callObjectMethodV(DvmMethod.java:89)
    at com.github.unidbg.linux.android.dvm.DalvikVM$32.handle(DalvikVM.java:527)
    at com.github.unidbg.linux.ARM32SyscallHandler.hook(ARM32SyscallHandler.java:103)
    at com.github.unidbg.arm.backend.UnicornBackend$6.hook(UnicornBackend.java:305)
    at unicorn.Unicorn$NewHook.onInterrupt(Unicorn.java:128)
    at unicorn.Unicorn.emu_start(Native Method)
    at com.github.unidbg.arm.backend.UnicornBackend.emu_start(UnicornBackend.java:331)
    at com.github.unidbg.AbstractEmulator.emulate(AbstractEmulator.java:378)
    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.android.dvm.DvmObject.callJniMethod(DvmObject.java:133)
    at com.github.unidbg.linux.android.dvm.DvmObject.callJniMethodObject(DvmObject.java:93)
    at com.antiUnidbg.getmethodidexample1.call(getmethodidexample1.java:44)
    at com.antiUnidbg.getmethodidexample1.main(getmethodidexample1.java:36)
[13:53:22 527]  WARN [com.github.unidbg.AbstractEmulator] (AbstractEmulator:389) - emulate RX@0x40000995[libgetmethodidexample0.so]0x995 exception sp=unidbg@0xbffff730, msg=com/example/getmethodid
null
```

Unidbg初学者并没有能力搞清楚上述过程，因此我认为算是一种Anti-Unidbg 的手段，就好比Frida server改端口就可以过掉一部分的Anti-Frida，但新手常常意识不到这一点。

接下来，我们再深入一些，把陷阱埋得更深一些。如果子类重写了父类的方法，但我们通过父类获取methodID，让子类Call 这个MethodID，子类会调用哪个方法呢？当然是自己重写的方法。

```
#include <jni.h>
#include <string>

extern "C" JNIEXPORT jstring JNICALL
Java_com_example_getmethodidexample0_MainActivity_stringFromJNI(
        JNIEnv* env,
        jobject /* this */) {

    jclass xiaomi_clz = env-
>FindClass("com/example/getmethodidexample0/XiaoMi");
    jclass phone_clz = env->FindClass("com/example/getmethodidexample0/Phone");

    jmethodID init = env->GetMethodID(xiaomi_clz, "<init>", "()V");
    jobject xiaomiObject = env->NewObject(xiaomi_clz, init);

    jmethodID price = env->GetMethodID(phone_clz, "getPrice", "
()Ljava/lang/String;");
    auto j_price = (jstring)env->CallObjectMethod(xiaomiObject, price);
    const char* c_price = env->GetStringUTFChars(j_price, nullptr);

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

最好先回顾一下我们写了什么JAVA代码

Phone.java

```java
package com.example.getmethodidexample0;

public class Phone {
    public String getPrice(){
        return "价格不清楚";
    }

    public String type(){
        return "手机";
    }
}
```

XiaoMi.java

```java
package com.example.getmethodidexample0;

public class XiaoMi extends Phone{

    @Override
    public String getPrice(){
        return "1999";
    }

}
```

运行结果是1999，接下来Unidbg模拟执行

```java
package com.antiUnidbg;

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.*;
import com.github.unidbg.memory.Memory;

import java.io.File;

public class getmethodidexample1 extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    getmethodidexample1() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/getmethodidexample4/app-debug.apk"));
        vm.setJni(this);
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/getmethodidexample4/libgetmethodidexample0.so"),
true);
        module = dm.getModule();
```

```
        vm.setVerbose(true); // 打印日志


        vm.resolveClass("com/example/getmethodidexample0/XiaoMi",
vm.resolveClass("com/example/getmethodidexample0/Phone"));

    };

    public static void main(String[] args) {
        getmethodidexample1 demo = new getmethodidexample1();
        demo.call();
    }

    public void call(){
        DvmClass dvmClass =
vm.resolveClass("com/example/getmethodidexample0/MainActivity");
        String methodSign = "stringFromJNI()Ljava/lang/String;";
        DvmObject<?> dvmObject = dvmClass.newObject(null);

        StringObject obj = dvmObject.callJniMethodObject(emulator, methodSign);
        System.out.println(obj);
    }

    @Override
    public DvmObject<?> newObjectV(BaseVM vm, DvmClass dvmClass, String
signature, VaList vaList) {
        switch (signature){
            case "com/example/getmethodidexample0/XiaoMi-><init>()V":{
                return
vm.resolveClass("com/example/getmethodidexample0/XiaoMi").newObject(null);
            }
        }
        return super.newObjectV(vm, dvmClass, signature, vaList);
    }
}
```

运行Unidbg代码

我们要补 *com/example/getmethodidexample0/Phone->getPrice()Ljava/lang/String;*，尽管我们知道，真实逻辑中，执行的是 *com/example/getmethodidexample0/XiaoMi->getPrice()Ljava/lang/String;*，这意味着，如果根据报错去用Frida call getPrice获取结果，这个结果可能是错的。

```
@Override
public DvmObject<?> callObjectMethodV(BaseVM vm, DvmObject<?> dvmObject, String
signature, VaList vaList) {
    switch (signature){
        case "com/example/getmethodidexample0/Phone-
>getPrice()Ljava/lang/String;":{
            return new StringObject(vm, "1999");
        }
    }
    return super.callObjectMethodV(vm, dvmObject, signature, vaList);
}
```

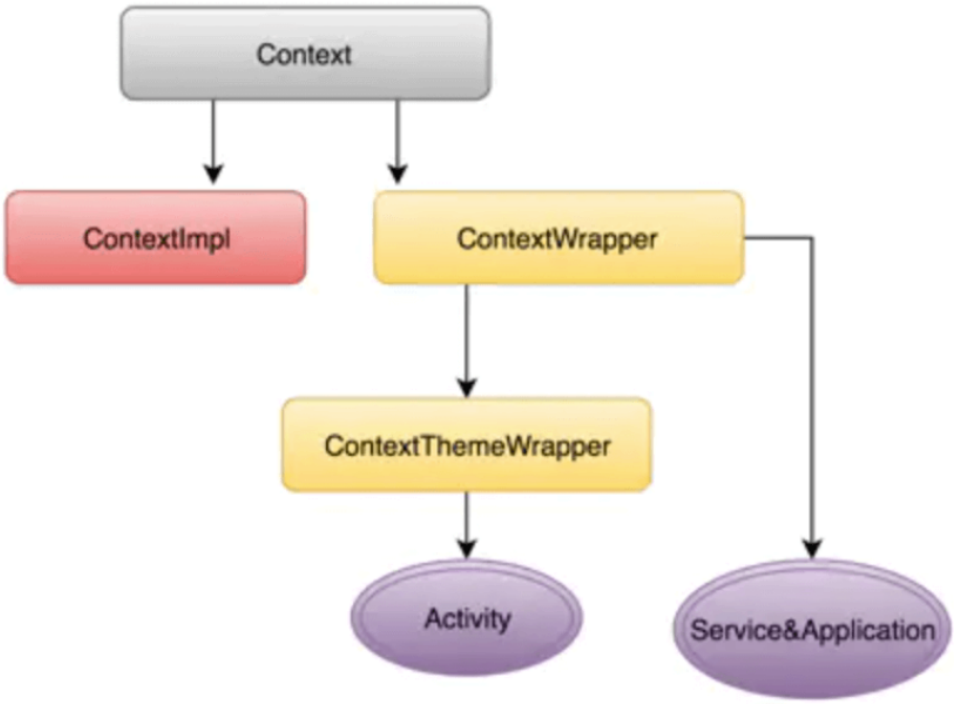
你应该也发现，在Unidbg的代码逻辑中，我们通过一种错误的妥协来实现目标，Phone类的getPrice方法并不应该返回1999，而是“价格不清楚”。如果样本再通过Phone对象调用getPrice，那它应该返回“价格不清楚”而不是“1999”，这里面显然产生了冲突。

根据这一点我们又可以设置陷阱，大家可以思考一下。除此之外，我们可以弄个双层陷阱。来看一下代码怎么写吧。

在JAVA层中，我们可以通过this.getPackAgeName获取包名。

```
activity_main.xml  MainActivity.java  ContextWrapper.class  Context.class  native-lib.cpp  Phone.java  XiaoMi.java
4   import android.os.Bundle;
5   import android.widget.TextView;
6
7   import com.example.getmethodidexample0.databinding.ActivityMainBinding;
8
9   public class MainActivity extends AppCompatActivity {
10
11      // Used to load the 'getmethodidexample0' library on application startup.
12      static {
13          System.loadLibrary( libname: "getmethodidexample0");
14      }
15
16      private ActivityMainBinding binding;
17
18      @Override
19      protected void onCreate(Bundle savedInstanceState) {
20          super.onCreate(savedInstanceState);
21
22          binding = ActivityMainBinding.inflate(getLayoutInflater());
23          setContentView(binding.getRoot());
24
25          TextView tv = binding.sampleText;
26          this.getPackageName();
27          tv.setText(stringFromJNI());
28      }
29
30      public native String stringFromJNI();
31  }
```

getPackageName由Context类中定义，ContextWrapper类中实现。MainActivity是Activity的子孙类，所以方法就这么继承下来了。



所以我们可以通过Context类获取getPackageName 方法ID，然后由一个MainActivity对象调用，这里的陷阱在于——在MainActivity中重写getPackageName 方法。

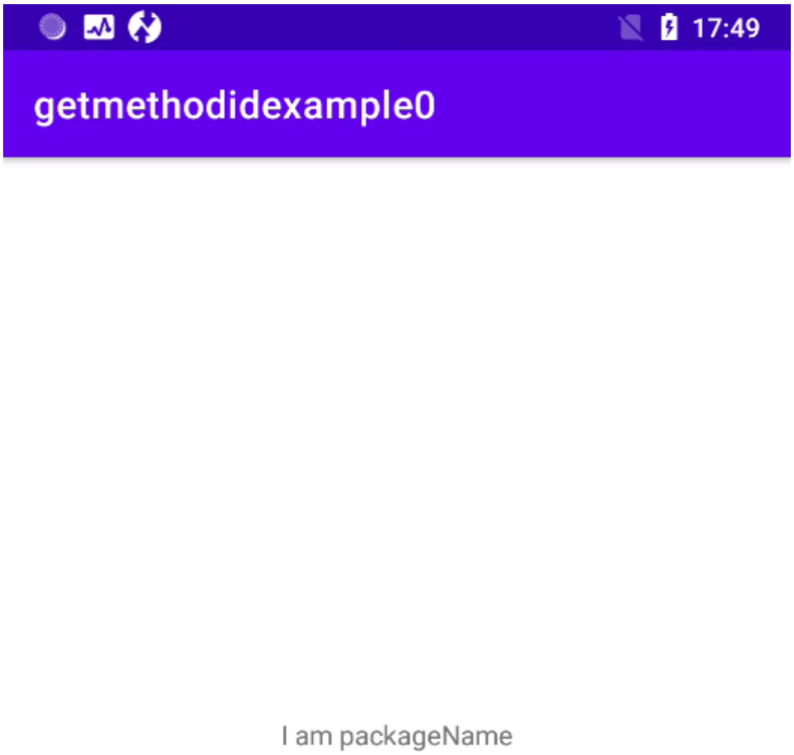
```
package com.example.getmethodidexample0;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

import com.example.getmethodidexample0.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

    // Used to load the 'getmethodidexample0' library on application startup.
    static {
        System.loadLibrary("getmethodidexample0");
    }

    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        TextView tv = binding.sampleText;

        tv.setText(stringFromJNI());
    }
```

```
    public native String stringFromJNI();

    @Override
    public String getPackageName() {
        return "I am packageName";
    }
}
```

接着看看*native-lib.cpp*

```
#include <jni.h>
#include <string>

extern "C" JNIEXPORT jstring JNICALL
Java_com_example_getmethodidexample0_MainActivity_stringFromJNI(
        JNIEnv* env,
        jobject mainactivity /* this */) {

    jclass context_clz = env->FindClass("android/content/Context");
    jmethodID methodID_getPackageName = env->GetMethodID(context_clz,
"getPackageName", "()Ljava/lang/String;");

    auto packageName = (jstring)env->CallObjectMethod(mainactivity,
methodID_getPackageName);

    const char* c_package_name = env->GetStringUTFChars(packageName, nullptr);
    return env->NewStringUTF(c_package_name);
}
```

测试机运行



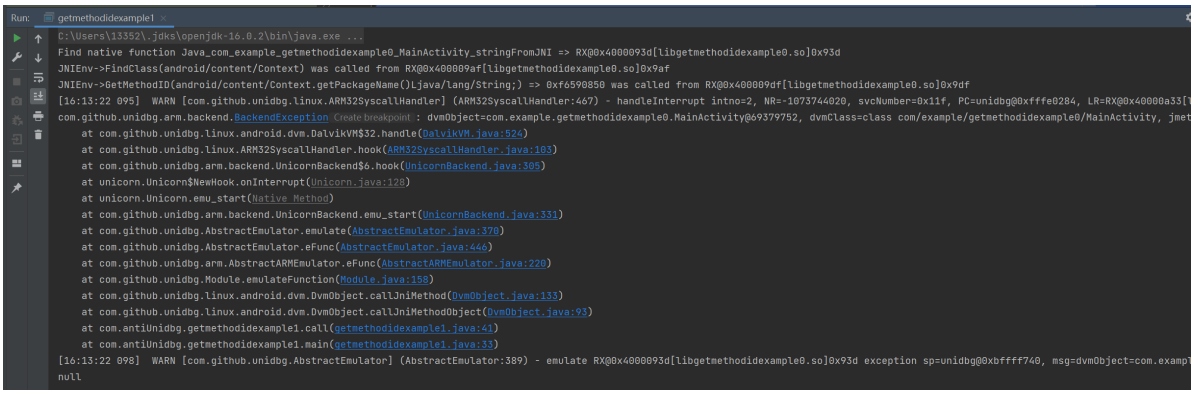
让我们捋一下做了什么——本类中重写的getPackageName方法，Native中通过Context获取getPackageName的方法ID。

那么在Unidbg 模拟执行中会遇到什么问题呢?

```
package com.antiUnidbg;

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.*;
import com.github.unidbg.memory.Memory;

import java.io.File;

public class getmethodidexample1 extends AbstractJni {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;
```

```
    getmethodidexample1() {
        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/getmethodidexample5/app-debug.apk"));
        vm.setJni(this);
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/getmethodidexample5/libgetmethodidexample0.so"),
true);
        module = dm.getModule();
        vm.setVerbose(true); // 打印日志

    };

    public static void main(String[] args) {
        getmethodidexample1 demo = new getmethodidexample1();
        demo.call();
    }

    public void call(){
        DvmClass dvmClass =
vm.resolveClass("com/example/getmethodidexample0/MainActivity");
        String methodSign = "stringFromJNI()Ljava/lang/String;";
        DvmObject<?> dvmObject = dvmClass.newObject(null);

        StringObject obj = dvmObject.callJniMethodObject(emulator, methodSign);
        System.out.println(obj);
    }

}
```

运行



首先遇到了找不到方法的问题，这可以难住一小部分人，而其余使用者会分析后，明确类的继承关系，让MainActivity继承自Context，

```
unidbg-master > unidbg-android > src > test > java > com > antiUnidbg > getmethodidexample1 > getmethodidexample1
findclass.java | DalvikVM.java | DvmClass.java | showMethod.java | AbstractJni.java | DexHelper.java | getmethodidexample1.java | BaseVM.java | trap.java | EnvTrap.java

        emulator = AndroidEmulatorBuilder
                .for32Bit()
                .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver( sdk: 23));
        vm = emulator.createDalvikVM(new File( pathname: "unidbg-android/src/test/resources/getmethodidexample5/app-debug.apk"));
        vm.setJni(this);
        DalvikModule dm = vm.loadLibrary(new File( pathname: "unidbg-android/src/test/resources/getmethodidexample5/libgetmethodidexample0.so"), forceCallInit: true);
        module = dm.getModule();
        vm.setVerbose(true); // 打印日志

        vm.resolveClass( className: "com/example/getmethodidexample0/MainActivity", vm.resolveClass( className: "android/content/Context"));
    };

    public static void main(String[] args) {
        getmethodidexample1 demo = new getmethodidexample1();
        demo.call();
    }

    public void call(){
        DvmClass dvmClass = vm.resolveClass( className: "com/example/getmethodidexample0/MainActivity");

Run: getmethodidexample1
C:\Users\13352\.jdks\openjdk-16.0.2\bin\java.exe ...
Find native function Java_com_example_getmethodidexample0_MainActivity_stringFromJNI => RX@0x4000093d[libgetmethodidexample0.so]0x93d
JNIEnv->FindClass(android/content/Context) was called from RX@0x400009af[libgetmethodidexample0.so]0x9af
JNIEnv->GetMethodID(android/content/Context.getPackageName()Ljava/lang/String;) => 0xf6590850 was called from RX@0x400009df[libgetmethodidexample0.so]0x9df
JNIEnv->CallObjectMethodV(com.example.getmethodidexample0.MainActivity@69379752, getPackageName() => "com.example.getmethodidexample0") was called from RX@0x40000a33[libgetmetho
JNIEnv->GetStringUtfChars("com.example.getmethodidexample0") was called from RX@0x40000a7b[libgetmethodidexample0.so]0xa7b
JNIEnv->NewStringUTF("com.example.getmethodidexample0") was called from RX@0x40000a9b[libgetmethodidexample0.so]0xa9b
"com.example.getmethodidexample0"

Process finished with exit code 0
```

这么做之后。。。直接出了结果， 但这个结果是错误的，因为Context getPackageName 在Unidbg中做了封装，直接返回”正确的包名“。下图是*AbstractJNI.java*

```
    case "android/app/Activity->getApplicationInfo()Landroid/content/pm/ApplicationInfo;":
        return new ApplicationInfo(vm);
    case "android/content/Context->getPackageName()Ljava/lang/String;":
    case "android/app/Activity->getPackageName()Ljava/lang/String;": {
        String packageName = vm.getPackageName();
        if (packageName != null) {
            return new StringObject(vm, packageName);
        }
        break;
    }
```

这个双重陷阱的设计十分巧妙。

三、尾声

Unidbg对进程的JAVA世界一无所知，基于这一点，我们可以埋下许多的坑，但这些技巧需要App本身有比较复杂的JAVA代码，混淆的Native代码，才能获得较好的效果，如果使用者能清晰方便的分析样本代码，那这些技巧就没用咯。