

一、前言

这是Anti-Unidbg的第二篇，初步讨论 Unidbg 的JNI 调用问题。

二、描述

Unicorn无法跑dex（至少目前没看到开源的方案），更别提跑通完整的Android Framework，那么当Native通过JNI这座桥梁和JAVA世界做交互时，Unidbg如何处理呢？

可以简单理解为Hook后打补丁，Unidbg预先补了一些基础的、对系统类和方法的访问，其余交由用户处理。实际过程复杂很多，详见 *unidbg-android/src/main/java/com/github/unidbg/linux/android/dvm/AbstractJni.java*，但不影响这样一个事实

Unidbg对进程的JAVA世界一无所知

Anti-Unidbg的一半武器，都由它贡献。（而另一半武器，则来自Unidbg对目标SO以外的Native世界缺少了解，这是后话）。比如文章一的问题，换个角度理解就是Unidbg不知道JAVA世界里添加了环境变量，所以让我们成功设置了一个陷阱。

本篇要讲什么呢——FindClass

```
jclass FindClass(JNIEnv *env, const char *name);
```

FindClass 用于查找类名为name的JAVA类，如果找到就返回jclass引用，没有就返回null并抛出异常。这个API有一个常见的作用——Anti 脱壳机。

比如开发者想检测YouPK（一款基于ART的主动调用的脱壳机），于是他阅读了YouPK代码，发现其中存在这么一个关键类



所以他认为，如果FindClass 查找到了这个类，就说明样本运行在YouPK脱壳机环境中。这合理吗？这很合理。

那么我们就得想怎么新瓶装旧酒，让这个API在Anti-Unidbg上也发挥出一些作用了。

那它怎么判断目标样本里是否存在FindClass所寻找的类呢？我想你应该也猜到了——默认有。

FindClass是为了后续使用这个类，所以凡是找某个类，它应该是存在的。否则干嘛？想报异常崩溃嘛？

这个逻辑大部分时候是对的，代价就是我们可以利用这一点来检测Unidbg。

来看下面的代码，FindClass 查找一些不可能存在的类，并做对应的异常处理防止代码崩溃影响业务本身逻辑。

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

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

    const char *result;
    jclass fake_clazz = env->FindClass("my/fake/class");
    bool exc = env->ExceptionCheck();
    if(exc){
        // 清除异常，不要让进程崩溃
        env->ExceptionClear();
    }
    if(fake_clazz == nullptr){
        result = "everything ok";
    } else{
        result = "unidbg detect";
    }
    return env->NewStringUTF(result);
}
```

代码做了三件事

- 找my/fake/class这个不存在的类
- 异常处理，让进程不崩溃，继续往下运行
- 如果FindClass 找到了类，走A逻辑，否则走B逻辑

看一下真机运行情况



everything ok



三、Unidbg模拟执行

findclass.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 com.github.unidbg.virtualmodule.android.AndroidModule;

import java.io.File;
```

```

public class findclass {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    findclass() {
        emulator = AndroidEmulatorBuilder
            .for32Bit()
            .build();

        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/findclass/app-debug.apk"));
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/findclass/libfindmyclass.so"), true);
        module = dm.getModule();
        vm.setVerbose(true); // 打印日志
    };

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

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

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

```

运行

```

Run: findclass
>>> 00=0x0(0.0) d1=0x34353512032203120(3.696225012140986E-33) d2=0x3220302034203(3.002229861217897E-67) d3=0x3435353832203203(3.536676186840298E-57) d4=0x203020302030
>>> d8=0x0(0.0) d9=0x0(0.0) d10=0x0(0.0) d11=0x0(0.0) d12=0x0(0.0) d13=0x0(0.0) d14=0x0(0.0) d15=0x0(0.0)
[22:33:12 068] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1803) - mprotect address=0x4000f000, alignedAddress=0x4000f000, offset=0, length=
>>> 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=RX00x400ec11b[libc.so]0x4811b PC=RX00x400e5284[libc.so]0x41284 cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
>>> d0=0x0(0.0) d1=0x3933312032203120(3.696225012140986E-33) d2=0x3220302034203736(3.002229861217897E-67) d3=0x3435353832203203(3.536676186840298E-57) d4=0x203020302030
>>> d8=0x0(0.0) d9=0x0(0.0) d10=0x0(0.0) d11=0x0(0.0) d12=0x0(0.0) d13=0x0(0.0) d14=0x0(0.0) d15=0x0(0.0)
[22:33:12 071] DEBUG [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:1803) - mprotect address=0x4000f000, alignedAddress=0x4000f000, offset=0, length=
[22:33:12 072] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX00x40041821[libc++.so]0x32821 finished sp=unidbg@0xbffff798, offset=85ms
[22:33:12 076] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x76f84423, global=true
[22:33:12 077] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x63658116, global=true
Find native function Java_com_example_findmyclass_MainActivity_stringFromJNI => RX00x40008ad[libfindmyclass.so]0x8ad
[22:33:12 077] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x23f7d05e, global=false
[22:33:12 079] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:354) - emulate RX00x40008ad[libfindmyclass.so]0x8ad started sp=unidbg@0xbffff798
[22:33:12 079] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX00x400020fc[libfindmyclass.so]0x20fc, size=13, encoding=UTF-8, r
JNIEnv->FindClass(my/fake/class) was called from RX00x4000933[libfindmyclass.so]0x933
[22:33:12 080] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0xffffffffa9bb72a0, global=true
[22:33:12 080] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$3:82) - FindClass env=unidbg@0xffff12a0, className=my/fake/class, hash=0xa9bb72a0
[22:33:12 080] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$225:3411) - ExceptionCheck throwable=null
[22:33:12 080] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX00x40002118[libfindmyclass.so]0x2118, size=13, encoding=UTF-8, r
[22:33:12 080] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$187:2988) - NewStringUTF bytes=RX00x40002118[libfindmyclass.so]0x2118, string=unidbg detect
JNIEnv->NewStringUTF("unidbg detect") was called from RX00x400097d[libfindmyclass.so]0x97d
[22:33:12 081] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0xffffffff83d61724, global=true
[22:33:12 081] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x60704c, global=false
[22:33:12 081] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX00x40008ad[libfindmyclass.so]0x8ad finished sp=unidbg@0xbffff798, offset=2m
unidbg detect
[libc++.so]CallInitFunction: RX00x40041821[libc++.so]0x32821, offset=85ms

```

可以发现，我们甜蜜的陷阱已经构造成功了，成功让Unidbg和真机结果产生了差异，而且几乎可以认定，执行环境就是Unidbg。

有一些办法可以让它更隐蔽

- FindClass 查找数十个类，绝大多数都正常，只有一个类是不存在的，浑水摸鱼。
- 并不是查找一个不存在的类，而是查找一个当前时机尚未加载的类，或者当前加载器无法加载的类，这样会更有迷惑性。

四、Anti Anti-Unidbg

最朴素的办法——对目标函数做JNItrace，同时打开Unidbg JNI相关日志，耐心仔细的逐条对比，[JNITrace](#)和[Jtrace](#)可以完成这个任务。在Unidbg中怎么表示出“目标类不存在”这样一种状态呢？从Unidbg代码中我们可以看出，设计者其实考虑过这个问题。

```
Pointer _FindClass = svcMemory.registerSvc(new Armsvc() {
    @Override
    public long handle(Emulator<?> emulator) {
        RegisterContext context = emulator.getContext();
        Pointer env = context.getPointerArg(0);
        Pointer className = context.getPointerArg(1);
        String name = className.getString(0);

        boolean notFound = notFoundClassSet.contains(name);
        if (verbose) {
            if (notFound) {
                System.out.printf("JNIEnv->FindNoClass(%s) was called from %s\n", name, context.getLRPointer());
            } else {
                System.out.printf("JNIEnv->FindClass(%s) was called from %s\n", name, context.getLRPointer());
            }
        }

        if (notFound) {
            throwable =
                resolveClass("java/lang/NoClassDefFoundError").newObject(name);
            return 0;
        }

        DvmClass dvmClass = resolveClass(name);
        long hash = dvmClass.hashCode() & 0xffffffffL;
        if (log.isDebugEnabled()) {
            log.debug("FindClass env=" + env + ", className=" + name + ", hash=0x" + Long.toHexString(hash));
        }
        return hash;
    }
});
```

使用者可以手动添加标注一个类为不存在

```
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 com.github.unidbg.virtualmodule.android.AndroidModule;

import java.io.File;

public class findclass {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

    findclass() {
        emulator = AndroidEmulatorBuilder
            .for32Bit()
            .build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver(23));
        vm = emulator.createDalvikVM(new File("unidbg-
android/src/test/resources/findclass/app-debug.apk"));
        DalvikModule dm = vm.loadLibrary(new File("unidbg-
android/src/test/resources/findclass/libfindmyclass.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");
    };

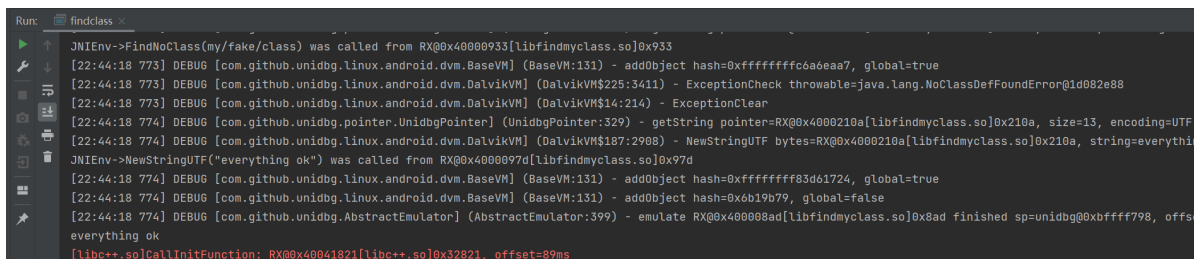
    public static void main(String[] args) {
        findclass demo = new findclass();
        demo.call();
    }

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

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

```

一切就顺利了



```

Run: findclass
[22:44:18 773] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0xffffffffc6a6eaa7, global=true
[22:44:18 773] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$225:3411) - ExceptionCheck throwable=java.lang.NoClassDefFoundError@1d082e88
[22:44:18 773] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$14:214) - ExceptionClear
[22:44:18 774] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x4000210a[libfindmyclass.so]0x210a, size=13, encoding=UTF
[22:44:18 774] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$187:2908) - NewStringUTF bytes=RX@0x4000210a[libfindmyclass.so]0x210a, string=everything
JNIEnv->NewStringUTF("everything ok") was called from RX@0x4000097d[libfindmyclass.so]0x97d
[22:44:18 774] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0xfffffffff83d61724, global=true
[22:44:18 774] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x6b19b79, global=false
[22:44:18 774] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x400088ad[libfindmyclass.so]0x8ad finished sp=unidbg@0xbffff798, offs
everything ok
[libc++.so]CallInitFunction: RX@0x40041821[libc++.so]0x32821, offset=89ms

```

五、尾声

除此之外，getMethodID，getStaticMethodID等JNI方法也可以制造同样陷阱，毕竟Unidbg也没办法知道某个类是否有某个方法，所以只能同样选择相信它**有**。