

## 一、前言

这是Anti-Unidbg的第一篇，实际上本篇讲的算不上什么Anti-Unidbg，最多算个小技巧，知道了就没用，不知道就能难倒一些人。

## 二、描述

Android是基于Linux内核的操作系统，存在系统环境变量这一概念，如下通过adb shell 查看系统环境变量的key和value。

```
C:\Users\13352>adb shell
polaris:/ $ export
ANDROID_ASSETS
ANDROID_BOOTLOGO
ANDROID_DATA
ANDROID_ROOT
ANDROID_RUNTIME_ROOT
ANDROID_SOCKET_adbd
ANDROID_STORAGE
ANDROID_TZDATA_ROOT
ASEC_MOUNTPOINT
BOOTCLASSPATH
DEX2OATBOOTCLASSPATH
DOWNLOAD_CACHE
EXTERNAL_STORAGE
HOME
HOSTNAME
LOGNAME
PATH
SHELL
SYSTEMSERVERCLASSPATH
TERM
TMPDIR
USER
polaris:/ $ echo $ANDROID_DATA
/data
polaris:/ $
```

每个Android进程会继承系统环境变量，除此之外，也可以通过如下API增删环境变量

*Java*

```
// OS.java
public static void setenv(String name, String value, boolean overwrite);
public static String getenv(String name);
```

*Native*

```
// 无需额外头文件
int setenv(const char *name, const char *value, int overwrite);
char *getenv(const char *name);
```

我们可以设置进程环境变量，在目标函数中检测或使用此环境变量，如果不存在，说明目标函数的执行环境就存在问题，有可能由Unidbg执行或者重打包SO 单独Call。

### 三、实践

*MainActivity.java* 添加一个自定义环境变量

```
package com.example.getenv;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.system.ErrnoException;
import android.system.Os;
import android.widget.TextView;

import com.example.getenv.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

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

    private ActivityMainBinding binding;

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

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

        try {
            // 添加一个环境变量，键name，值lilac。
            Os.setenv("name", "lilac", true);
        } catch (ErrnoException e) {
            e.printStackTrace();
        }

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

    public native String stringFromJNI();
}
```

*native-lib.cpp* 读取这个环境变量

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

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

    const char *value = getenv("name");
    if(value == nullptr){
        value = "fake call!";
    }
}
```

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

如果正常运行，应该返回lilac。

getenv

lilac



## 四、Unidbg模拟执行

*EnvTrap.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 EnvTrap {
    private final AndroidEmulator emulator;
    private final VM vm;
    private final Module module;

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

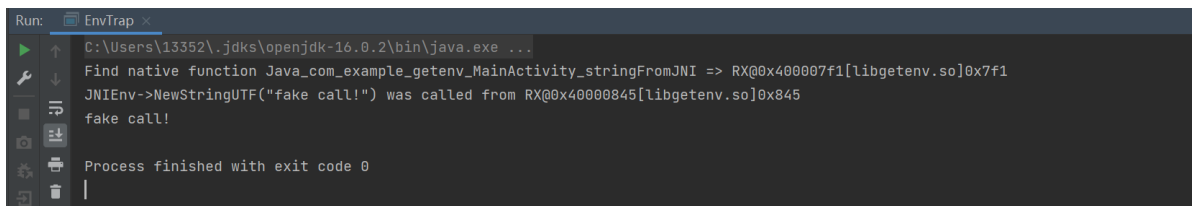
    public static void main(String[] args) {
        EnvTrap demo = new EnvTrap();
        demo.call();
    }

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

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

```

运行测试，可以发现返回了错误的值。



```

Run: EnvTrap x
C:\Users\13352\.jdk\openjdk-16.0.2\bin\java.exe ...
Find native function Java_com_example_getenv_MainActivity_stringFromJNI => RX@0x400007f1[libgetenv.so]0x7f1
JNIEnv->NewStringUTF("fake call!") was called from RX@0x40000845[libgetenv.so]0x845
fake call!
Process finished with exit code 0

```

打开Unidbg全部调试开关，查看是否有相关信息输出，比如是否会提示getenv没取到值？无感检测才是好检测。

*unidbg-android/src/test/resources/log4j.properties*

```

4 log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
5 log4j.appender.stdout.layout.ConversionPattern=[%d{HH:mm:ss SSS}] %p [%c] (%C{1}:%L) - %m%n
6
7 log4j.logger.com.github.unidbg=DEBUG
8 log4j.logger.com.github.unidbg.AbstractEmulator=DEBUG
9 log4j.logger.com.github.unidbg.linux.libc++.so=DEBUG
10 log4j.logger.com.github.unidbg.linux.android.dvm=DEBUG
11 log4j.logger.com.github.unidbg.linux.android.ArmlD64=DEBUG
12 log4j.logger.com.github.unidbg.linux.AndroidElfLoader=DEBUG
13 log4j.logger.com.github.unidbg.unix.UnixSyscallHandler=DEBUG
14 log4j.logger.com.github.unidbg.linux.ARMSyscallHandler=DEBUG
15 log4j.logger.com.github.unidbg.linux.ARM64SyscallHandler=DEBUG
16 log4j.logger.com.github.unidbg.dlfcn=DEBUG
17 log4j.logger.com.github.unidbg.hook=DEBUG
18 log4j.logger.com.github.unidbg.pointer.BackendPointer=DEBUG
19 log4j.logger.com.github.unidbg.linux.file=DEBUG
20 log4j.logger.com.github.unidbg.linux.file.UdpSocket=DEBUG
21 log4j.logger.com.github.unidbg.linux.file.TcpSocket=DEBUG
22 log4j.logger.com.github.unidbg.linux.file.LocalUdpSocket=DEBUG
23 log4j.logger.com.github.unidbg.hook.hookzz=DEBUG
24 log4j.logger.com.github.unidbg.linux.android.XHookImpl=DEBUG
25 log4j.logger.net.fornwall.jelf.AndroidRelocationIterator=DEBUG
26 log4j.logger.com.github.unidbg.AndroidRelocationTest=DEBUG
27 log4j.logger.com.github.unidbg.unix.file=DEBUG
28 log4j.logger.com.github.unidbg.arm.backend.hypervisor.HypervisorBackend64=DEBUG

```

## 再次运行

```

Run: EnvTrap
>>> SP=0xbffff770 LR=RX@0x400ec9d[libc.so]0x4809d PC=RX@0x400e5284[libc.so]0x41284 cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
>>> d0=0x0(0.0) d1=0x3935312032203120(3.696225012140986E-33) d2=0x322032034203736(3.0022298612178987E-67) d3=0x3436333832203235(3.536676186840298E-57) d4=0x2030203020302030(1
>>> 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)
[19:05:52 562] DEBUG [com.github.unidbg.linux.ARMSyscallHandler] (ARMSyscallHandler:1803) - mprotect address=0x4000f000, alignedAddress=0x4000f000, offset=0, length=4096,
>>> r0=0x4000f000 r1=0x1000 r2=0x1 r3=0x40124440 r4=0x4009f000 r5=0x1000 r6=0xc r7=0x7d r8=0x4009f040 sb=0x4009f000 sl=0x4009f00c fp=0x0 ip=0x40066409
>>> SP=0xbffff770 LR=RX@0x400ec11b[libc.so]0x4811b PC=RX@0x400e5284[libc.so]0x41284 cpsr: N=0, Z=0, C=0, V=0, T=0, mode=0b10000
>>> d0=0x0(0.0) d1=0x3935312032203120(3.696225012140986E-33) d2=0x322032034203736(3.0022298612178987E-67) d3=0x3436333832203235(3.536676186840298E-57) d4=0x2030203020302030(1
>>> 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)
[19:05:52 564] DEBUG [com.github.unidbg.linux.ARMSyscallHandler] (ARMSyscallHandler:1803) - mprotect address=0x4000f000, alignedAddress=0x4000f000, offset=0, length=4096,
[19:05:52 565] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x40041821[libc++.so]0x32821 finished sp=unidbg@0xbffff798, offset=68ms
[19:05:52 568] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x76f84423, global=true
[19:05:52 568] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x603c71ac, global=true
Find native function Java_com_example_getenv_MainActivity_stringFromJNI => RX@0x400007f1[libgetenv.so]0x7f1
[19:05:52 569] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x23529fee, global=false
[19:05:52 576] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:354) - emulate RX@0x400007f1[libgetenv.so]0x7f1 started sp=unidbg@0xbffff798
[libc++.so]CallInitFunction: RX@0x40041821[libc++.so]0x32821, offset=68ms
[19:05:52 578] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=RX@0x40001fc9[libgetenv.so]0x1fc9, size=10, encoding=UTF-8, ret=fake call
[19:05:52 578] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM:187:2908) - NewStringUTF bytes=RX@0x40001fc9[libgetenv.so]0x1fc9, string=fake call!
JNIEnv->NewStringUTF("fake call!") was called from RX@0x40000845[libgetenv.so]0x845
[19:05:52 578] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0xffffffff83d61724, global=true
[19:05:52 578] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x2fd6b6c7, global=false
[19:05:52 578] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x400007f1[libgetenv.so]0x7f1 finished sp=unidbg@0xbffff798, offset=1ms
fake call!

```

风平浪静，Unidbg并没有给出提示。如何修复这个问题？首先，我们需要知道如何在Unidbg中添加环境变量？

在 `src/main/java/com/github/unidbg/linux/AndroidElfLoader.java` 中增加系统环境变量是最简单的办法

```

unidbg-master unidbg-android src main java com github unidbg linux AndroidElfLoader AndroidElfLoader
AbstractElfLoader.java log4j.properties DvmMethod.java DalvikVM.java ARMSyscallHandler.java DexHelper.java trap.java StringObject.java AndroidElfLoader.java EnvTrap.java
69 public AndroidElfLoader(Emulator<AndroidFileIO> emulator, UnixSyscallHandler<AndroidFileIO> syscallHandler) {
70     super(emulator, syscallHandler);
71
72     // init stack
73     stackSize = STACK_SIZE_OF_PAGE * emulator.getPageAlign();
74     backend.mem_map(address: STACK_BASE - stackSize, stackSize, perms: UnicornConst.UC_PROT_READ | UnicornConst.UC_PROT_WRITE);
75
76     setStackPoint(STACK_BASE);
77     this.envIRON = initializeTLS(new String[] {
78         "ANDROID_DATA=/data",
79         "ANDROID_ROOT=/system",
80         "name=lilac"
81     });
82     this.setErrno(0);
83 }
84
85 @Override
86 public void setLibraryResolver(LibraryResolver libraryResolver) {
87     syscallHandler.addIOResolver((AndroidResolver) libraryResolver);
88     super.setLibraryResolver(libraryResolver);
89 }
90
91 Run: EnvTrap
[19:10:02 888] DEBUG [com.github.unidbg.linux.ARMSyscallHandler] (ARMSyscallHandler:1803) - mprotect address=0x4000f000, alignedAddress=0x4000f000, offset=0, length=4096, alignedLength
[19:10:02 889] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x40041821[libc++.so]0x32821 finished sp=unidbg@0xbffff788, offset=70ms
[19:10:02 892] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x76f84423, global=true
[19:10:02 893] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x603c71ac, global=true
Find native function Java_com_example_getenv_MainActivity_stringFromJNI => RX@0x400007f1[libgetenv.so]0x7f1
[19:10:02 893] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x23529fee, global=false
[19:10:02 893] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:354) - emulate RX@0x400007f1[libgetenv.so]0x7f1 started sp=unidbg@0xbffff788
[19:10:02 896] DEBUG [com.github.unidbg.pointer.UnidbgPointer] (UnidbgPointer:329) - getString pointer=unidbg@0xbffffa80, size=5, encoding=UTF-8, ret=lilac
[19:10:02 896] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM:187:2908) - NewStringUTF bytes=unidbg@0xbffffa80, string=lilac
JNIEnv->NewStringUTF("lilac") was called from RX@0x40000845[libgetenv.so]0x845
[19:10:02 896] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0xffffffff83d61724, global=true
[19:10:02 896] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x2fd6b6c7, global=false
[19:10:02 897] DEBUG [com.github.unidbg.AbstractEmulator] (AbstractEmulator:399) - emulate RX@0x400007f1[libgetenv.so]0x7f1 finished sp=unidbg@0xbffff788, offset=2ms
lilac
[libc++.so]CallInitFunction: RX@0x40041821[libc++.so]0x32821, offset=77ms

```

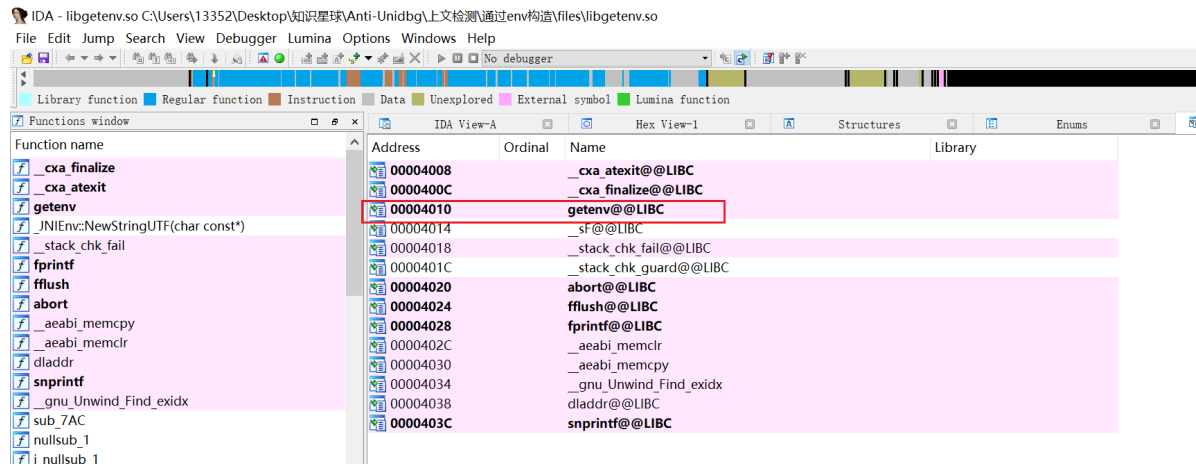
除此之外，如下两个方式也是可以的，这里就不赘述了

- call setenv 添加环境变量

- hook getenv 返回正确环境变量

## 五、Anti Anti-Unidbg

Unidbg并没给我们任何提示或者报错，如果样本JAVA层存在加固且业务逻辑复杂，Native层存在混淆，分析者很难自上而下阅读代码时，很难感知我们设计的这个简单陷阱。



这就涉及到Unidbg的一个问题，调试全开的情况下，显示SO加载、Dlopen调用、系统调用、JNI调用、文件读写这些，但对SO导入函数的调用并没有相关显示。如果分析者使用Unidbg或者Frida等工具，Hook SO的所有导入函数，就会注意到如getenv这类敏感API的调用。