Unicorn基础学习

原创

C4cke 于 2021-09-06 11:00:42 发布 168 收藏

分类专栏： python 文章标签： Unicorn

本文链接：https://blog.csdn.net/qq_35289660/article/details/120129658

Unicorn基础学习

跟着看雪无名侠的文章学的：Unicorn 在 Android 的应用

知识点在他给的印象笔记里，我也就学了快速入门。。。

基础入门是这篇：Unicorn快速入门

下面给了两份代码，都差不多，也是粘的大佬的，然后写了注释，方便理解。

模拟ARM执行

```
from unicorn import *
#导入ARM指令的指令集
from unicorn.arm_const import *

#我们要模拟执行的ARM指令
ARM_CODE    = b"\x37\x00\xa0\xe3\x03\x10\x42\xe0"
# mov r0, #0x37;
# sub r1, r2, r3
# Test ARM

# callback for tracing instructions 回调函数
def hook_code(uc, address, size, user_data):
    print(">>> Tracing instruction at 0x%x, instruction size = 0x%x" %(address, size))

def test_arm():
    print("Emulate ARM code")
    try:
        # Initialize emulator in ARM mode
        #使用Uc类初始化 Unicorn ,创建一个虚拟机对象。
        # 此类接受 2 个参数：硬件架构和硬件模式。在此示例中，我们要模拟 X86 架构的 32 位代码。作为回报，我们在mu 中有一个此类的变量。
        mu = Uc(UC_ARCH_ARM, UC_MODE_THUMB)



        # map 2MB memory for this emulation
        #在第9行声明的地址处使用mem_map方法为此模拟映射 2MB 内存。这个过程中所有的CPU操作都应该只访问这块内存。
        # 该内存使用默认权限 READ、WRITE 和 EXECUTE 进行映射。
        ADDRESS = 0x10000
```

```
        ADDRESS = 0x10000
        mu.mem_map(ADDRESS, 2 * 0x10000)

        #将要模拟的代码写入我们刚才映射的内存中。方法mem_write接受 2 个参数：要写入的地址和要写入内存的代码。
        mu.mem_write(ADDRESS, ARM_CODE)

        #使用reg_write方法设置R0、R1和R2寄存器的值
        mu.reg_write(UC_ARM_REG_R0, 0x1234)
        mu.reg_write(UC_ARM_REG_R2, 0x6789)
        mu.reg_write(UC_ARM_REG_R3, 0x3333)

        #hook机制，（类似 windows事件相应），满足Type类型，hook就被命中
        # UC_HOOK_CODE：hook指令，每指令一条指令就命中我们的hook。
        # hook_code：回调函数，不同类型的hook，对应的callback的参数也不相同
        # beging~end，代表执行hook的区间
        mu.hook_add(UC_HOOK_CODE, hook_code, begin=ADDRESS, end=ADDRESS+0x1000)

        # emulate machine code in infinite time
        #使用方法emu_start启动仿真。这个 API 有 4 个参数：仿真代码的地址、仿真停止的地址（紧跟在X86_CODE32的最后一个字节之后）、要仿真的时间和要仿真的指令数。
        # 如果我们像这个例子一样忽略最后两个参数，Unicorn 将在无限时间内和无限数量的指令中模拟代码，
        #emu_start 可以通过timeout参数设置最长执行时长，防止线程死在虚拟机里面。原型如下
        # 直到模拟执行的代码执行完毕。
        mu.emu_start(ADDRESS, ADDRESS + len(ARM_CODE))

        #emu_start执行完后
        # 取出模拟执行的结果
        r0 = mu.reg_read(UC_ARM_REG_R0)
        r1 = mu.reg_read(UC_ARM_REG_R1)

        print(">>> R0 = 0x%x" % r0)
        print(">>> R1 = 0x%x" % r1)

    except UcError as e:
        print("ERROR: %s" % e)

if __name__ == '__main__':
    test_arm()
```

模拟X86执行

```python
from __future__ import print_function
from unicorn import *
#导入X86指令的指令集
from unicorn.x86_const import *

# callback for tracing instructions 回调函数
def hook_code(uc, address, size, user_data):
    print(">>> Tracing instruction at 0x%x, instruction size = 0x%x" %(address, size))


# code to be emulated
X86_CODE32 = b"\x41\x4a"
# INC ecx; DEC edx
# ecx自增1，edx自减1

# memory address where emulation starts
ADDRESS = 0x1000000

print("Emulate i386 code")
try:
    # Initialize emulator in X86-32bit mode
    mu = Uc(UC_ARCH_X86, UC_MODE_32)

    # map 2MB memory for this emulation
    mu.mem_map(ADDRESS, 2 * 1024 * 1024)

    # write machine code to be emulated to memory
    mu.mem_write(ADDRESS, X86_CODE32)

    # initialize machine registers
    mu.reg_write(UC_X86_REG_ECX, 0x1234)
    mu.reg_write(UC_X86_REG_EDX, 0x7890)

    #hook机制，（类似 windows事件相应），满足Type类型，hook就被命中
    # UC_HOOK_CODE：hook指令，每指令一条指令就命中我们的hook。
    # hook_code：回调函数，不同类型的hook，对应的callback的参数也不相同
    # beging~end，代表执行hook的区间
    mu.hook_add(UC_HOOK_CODE, hook_code, begin=ADDRESS, end=ADDRESS+0x1000)

    # emulate code in infinite time & unlimited instructions
    mu.emu_start(ADDRESS, ADDRESS + len(X86_CODE32))

    # now print out some registers
    print("Emulation done. Below is the CPU context")

    r_ecx = mu.reg_read(UC_X86_REG_ECX)
    r_edx = mu.reg_read(UC_X86_REG_EDX)
    print(">>> ECX = 0x%x" %r_ecx)
    print(">>> EDX = 0x%x" %r_edx)

except UcError as e:
    print("ERROR: %s" % e)
```