《漏洞利用及渗透测试基础》实验报告

姓名：管昀玫 学号：2013750 班级：计科一班

**实验名称：**

Angr应用示例实验

**实验要求：**

根据课本8.4.3章节，复现sym-write示例的两种angr求解方法，并就如何使用angr以及怎么解决一些实际问题做一些探讨。

**实验过程：**

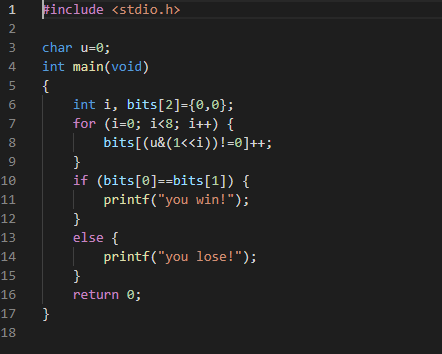
1. 实验准备

pip install angr安装angr

去angr的官网下载angr-doc，获取说明文档和测试用例

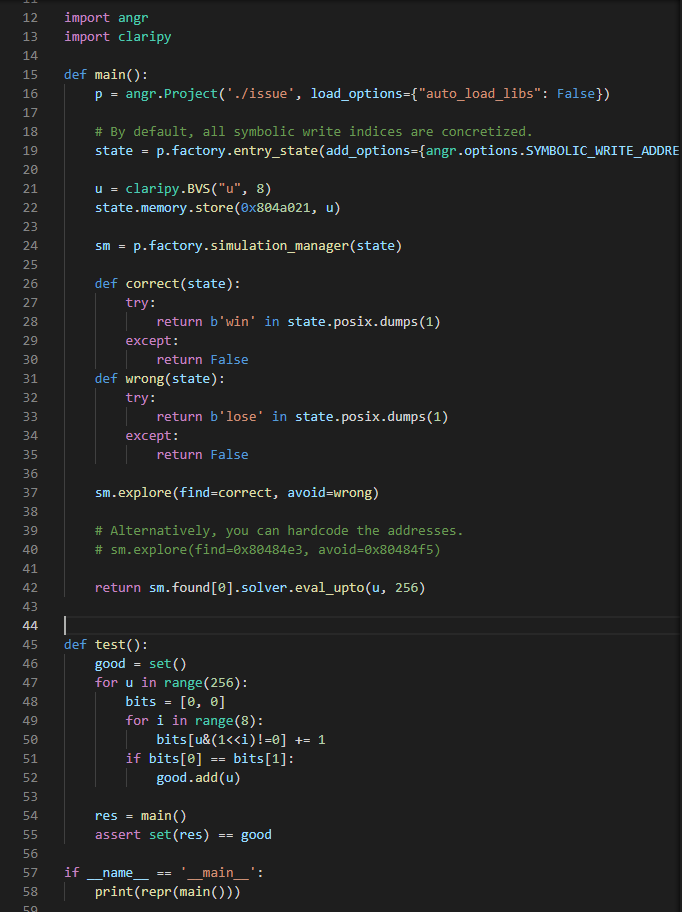
1. 实验过程
   1. 目标分析

观察源代码



可以看出我们需要达到的目标应当是在屏幕上输出you win，而达到这个分支的条件是让bits[0]==bits[1]，根据上文的计算可以看出，bits的取值是和u的取值有关的，因此需要破解的也就是u的取值

源码solve.py：



* 1. angr分析

源码solve.py有以下作用：

整个python程序将执行print(repr(main()))语句，进而，将main函数的返回值打印出来，repr()函数将object对象转化为string类型。

在上述Angr示例中，几个关键步骤如下：

（1）新建一个Angr工程，并且载入二进制文件。auto\_load\_libs设置为false，将不会自动载入依赖的库，默认情况下设置为false。如果设置为true，转入库函数执行，有可能给符号执行带来不必要的麻烦。

（2）初始化一个模拟程序状态的SimState对象state（使用函数entry\_state()），该对象包含了程序的内存、寄存器、文件系统数据、符号信息等等模拟运行时动态变化的数据。此外，也可以使用函数blank\_state()初始化模拟程序状态的对象state，在该函数里可通过给定参数addr的值指定程序起始运行地址。

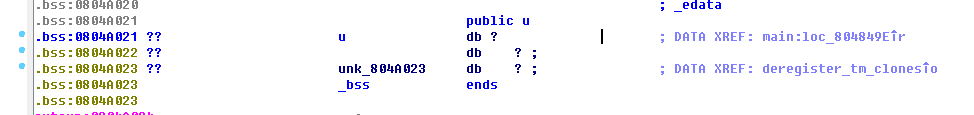
（3）将要求解的变量符号化，注意这里符号化后的变量存在二进制文件的存储区。

（4）创建模拟管理器（Simulation Managers）进行程序执行管理。初始化的state可以经过模拟执行得到一系列的states，模拟管理器sm的作用就是对这些states进行管理。

（5）进行符号执行得到想要的状态，得到想要的状态。上述程序所表达的状态就是，符号执行后，源程序里打印出的字符串里包含win字符串，而没有包含lose字符串。在这里，状态被定义为两个函数，通过符号执行得到的输出state.posix.dumps(1)中是否包含win或者lose的字符串来完成定义。

（6）获得到state之后，通过solver求解器，求解u的值

在state.memory.store(0x804a021, u)这句中，是把符号变量保存在指定地址中，这个地址是就是二进制文件中.bss段u的地址。可以通过IDA\_PRO去寻找在源代码中u对应的位置，如下图



在bss段中可以找到public u，0884A021，也就是u这个全局变量的地址

在这之后，我们可以对整个文件进行符号执行，以找到符号变量u的位置。

* 1. 运行结果

定义了两个搜寻结果函数，以检查输出字符串是win还是lose：

    def correct(state):

        try:

            return b'win' in state.posix.dumps(1)

        except:

            return False

    def wrong(state):

        try:

            return b'lose' in state.posix.dumps(1)

        except:

            return False

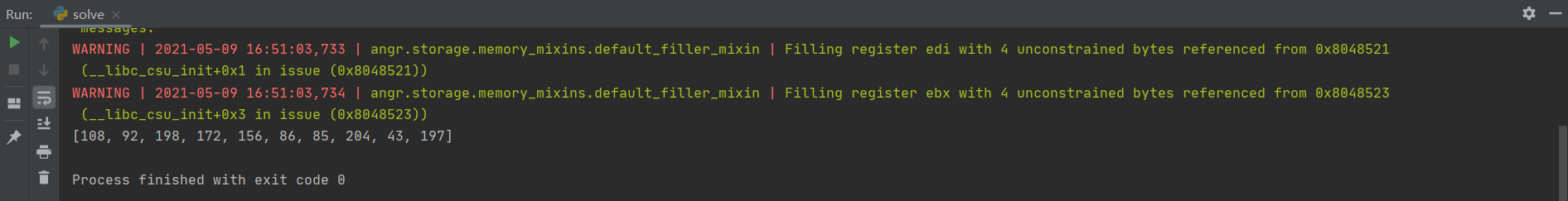
使用sm.explore函数来搜索状态想要的状态，即得到满足correct条件且不满足wrong条件的state；也可以写成通过地址进行定位。

    sm.explore(find=correct, avoid=wrong)

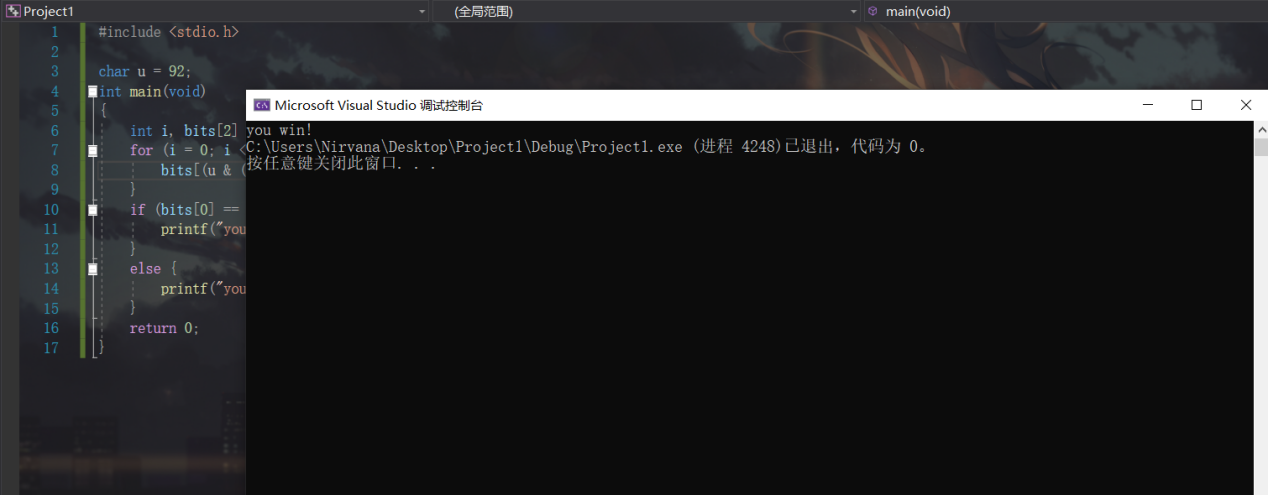
    # Alternatively, you can hardcode the addresses.

    # sm.explore(find=0x80484e3, avoid=0x80484f5)

运行结果如下：



得到了10个求出来的u的值，将其带回到源程序去验证



成功输出了you win ，实验成功。

另一种解法：

#another\_solution

# coding=utf-8

import angr

import claripy

def hook\_demo(state):

    state.regs.eax = 0

p = angr.Project("./issue", load\_options={"auto\_load\_libs": False})

# hook函数：addr为待hook的地址

# hook为hook的处理函数，在执行到addr时，会执行这个函数，同时把当前的state对象作为参数传递过去

# length 为待hook指令的长度，在执行完 hook 函数以后，angr 需要根据 length 来跳过这条指令，执行下一条指令

# hook 0x08048485处的指令（xor eax,eax），等价于将eax设置为0

# hook并不会改变函数逻辑，只是更换实现方式，提升符号执行速度

p.hook(addr=0x08048485, hook=hook\_demo, length=2)

state = p.factory.blank\_state(addr=0x0804846B, add\_options={"SYMBOLIC\_WRITE\_ADDRESSES"})

u = claripy.BVS("u", 8)

state.memory.store(0x0804A021, u)

sm = p.factory.simulation\_manager(state)

sm.explore(find=0x080484DB)

st = sm.found[0]

print(repr(st.solver.eval(u)))

在其中，

p.hook(addr=0x08048485, hook=hook\_demo, length=2)

这句中的08048485为xor eax,eax，等价于将eax设置为0。

上述代码与前面的解法有三处区别：

* 采用了hook函数，将0x08048485处的长度为2的指令通过自定义的hook\_demo进行替代，功能是一致的，原始xor eax, eax和state.regs.eax = 0是相同的作用，这里只是演示，可以将一些复杂的系统函数调用，比如printf等，可以进行hook，提升符号执行的性能。
* 进行符号执行得到想要的状态，有变化，变更为find=0x080484DB。因为源程序win和lose是互斥的，所以，只需要给定一个find条件即可。
* 最后，eval(u)替代了原来的eval\_upto，将打印一个结果出来。

**心得体会：**

**关于本实验：**

在第一个solve.py文件中，explore函数也可以用find=0x80484e3, avoid=0x80484f5来代替，即通过符号执行是否到达特定代码区的地址。使用IDA反汇编可知0x80484e3是printf("you win!")对应的汇编语句；0x80484f5则是printf("you lose!")对应的汇编语句。

而对于sm.found[0].solver.eval\_upto(u, 256)，这里有多个函数可以使用，eval\_upto(e, n, cast\_to=None, \*\*kwargs) 求解一个表达式多个可能的求解方案，e-表达式，n-所需解决方案的数量；eval(e, \*\*kwargs) 评估一个表达式以获得任何可能的解决方案；eval\_one(e, \*\*kwargs)求解表达式以获得唯一可能的解决方案。

**一些心得：**

之前有在假期的时候自己学过一些angr，但是当时看英文文档就没怎么看懂，而且当时看的时候很多的名词像什么hook之类的都不太清楚，看起来很费劲，结果就是看完一遍也用了几个里面的测试用例，最后还是什么也不会，也忘完了。现在再学的时候加上以前的理解，对angr了解的比以前要好一些了，同时对于符号执行和符号执行的求解器也是更加敬畏了，能想出来这些还有设计出来解法的人是真的太强了，这个的功能也是太厉害了。

记得之前看文档时，里面主要提及的就是利用hook和符号执行等技术，实现这类的计算求解问题，但是也可以用于其他的方面：比如爆破、漏洞挖掘、画出程序的CFG等。