《漏洞利用及渗透测试基础》实验报告

姓名：沙璇 学号：1911562 班级：信息安全

**实验名称：**

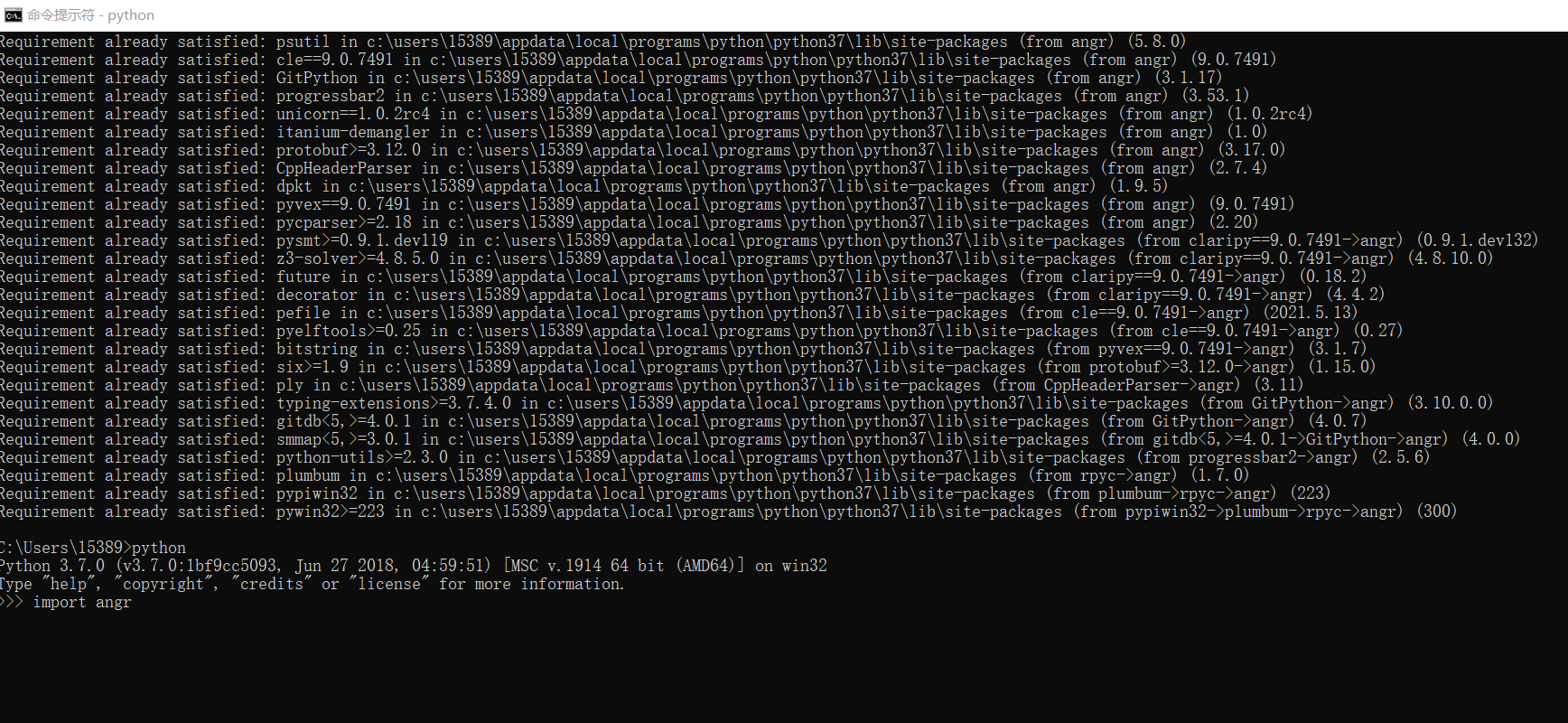
Angr应用

**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首先打开命令控制台，输入pip install angr。输入命令python，进入python界面，然后输入import angr。



②

angr-doc-master\examples\sym-write\issue.c：

#include <stdio.h>

char u=0;

int main(void)

{

int i, bits[2]={0,0};

for (i=0; i<8; i++) {

bits[(u&(1<<i))!=0]++;

}

if (bits[0]==bits[1]) {

printf("you win!");

}

else {

printf("you lose!");

}

return 0;}

*Solve.py*：

import angr

import claripy

def main():

# 1. 新建一个工程，导入二进制文件，后面的选项是选择不自动加载依赖项，不会自动载入依赖的库

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

# 2. 初始化一个模拟程序状态的SimState对象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

# blank\_state():可通过给定参数addr的值指定程序起始运行地址

# entry\_state():指明程序在初始运行时的状态，默认从入口点执行

# add\_options获取一个独立的选项来添加到某个state中，更多选项说明见https://docs.angr.io/appendix/options

# SYMBOLIC\_WRITE\_ADDRESSES：允许通过具体化策略处理符号地址的写操作

state = p.factory.entry\_state(add\_options={angr.options.SYMBOLIC\_WRITE\_ADDRESSES})

# 3. 创建一个符号变量，这个符号变量以8位bitvector形式存在，名称为u

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

# 把符号变量保存到指定的地址中，这个地址是就是二进制文件中.bss段u的地址

state.memory.store(0x804a021, u)

# 4. 创建一个Simulation Manager对象，这个对象和我们的状态有关系

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

# 5. 使用explore函数进行状态搜寻，检查输出字符串是win还是lose

# state.posix.dumps(1)获得所有标准输出

# state.posix.dumps(0)获得所有标准输入

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

# 进行符号执行得到想要的状态，即得到满足correct条件且不满足wrong条件的state

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

# 也可以写成下面的形式，直接通过地址进行定位

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

# 获得到state之后，通过solver求解器，求解u的值

# eval\_upto(e, n, cast\_to=None, \*\*kwargs) 求解一个表达式指定个数个可能的求解方案 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。 e - 表达式

# eval\_one(e, \*\*kwargs) 求解表达式以获得唯一可能的解决方案。 e - 表达式

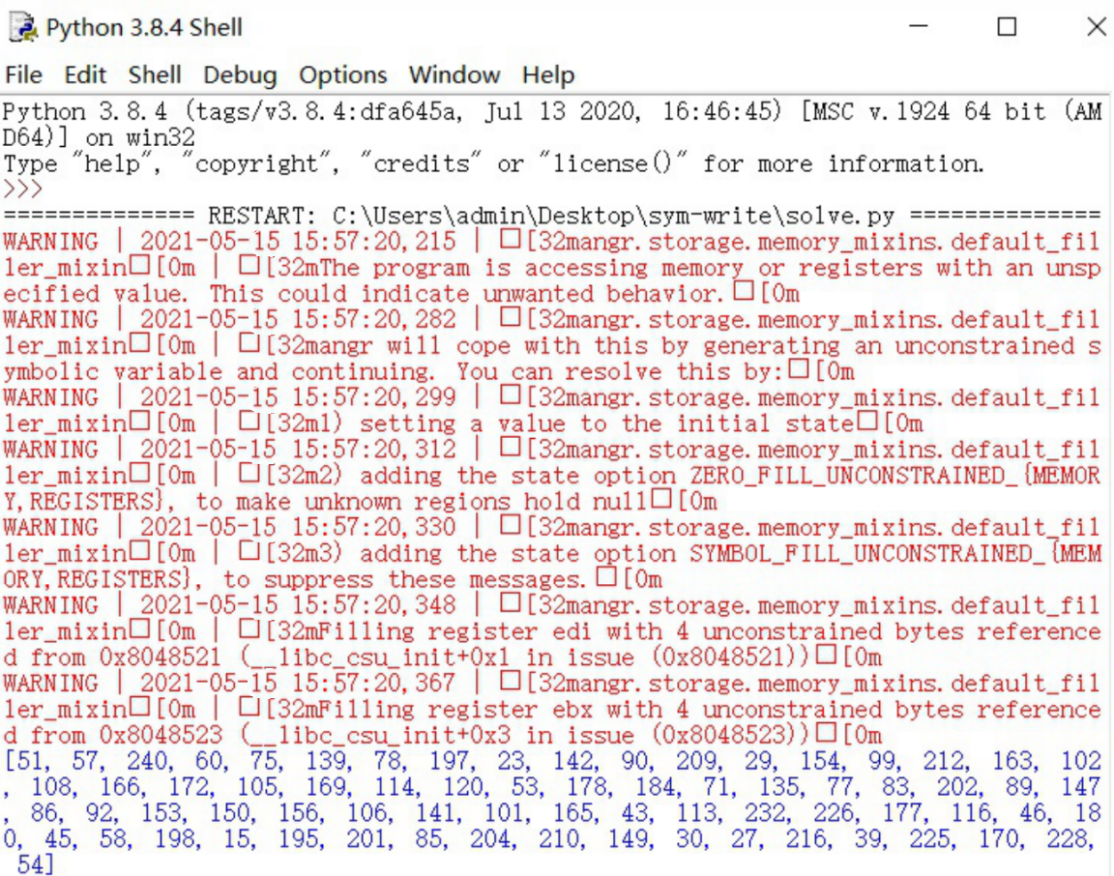
return sm.found[0].solver.eval\_upto(u, 256)

if \_\_name\_\_ == '\_\_main\_\_':

# repr()函数将object对象转化为string类型

print(repr(main()))

③实验验证：选择solve.py，右键选择Edit with IDLEEdit with IDLE 3.9 (64 bit)，将弹出界面，选择Run model。

****

下面蓝色的部分就是u求解的结果，因为我们采用了eval\_upto函数，所以给出了多个解。

解法2：

解法2采用以下代码进行求解。

#!/usr/bin/env python

# 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)

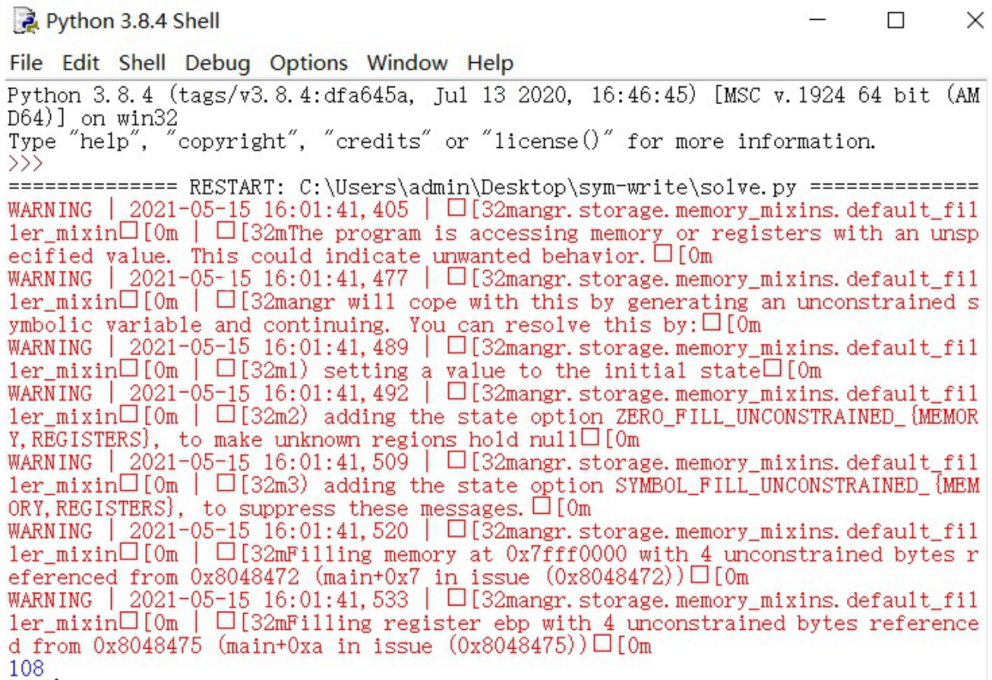
st = sm.found[0]

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

2解法与1解法的区别是：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，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。

eval(u)替代了eval\_upto，只打印一个结果。

运行结果如下，可以看到结果不同。



**心得体会**：Angr 是一个二进制代码分析工具，能够自动化完成二进制文件的分析，并找出漏洞。在 二进制代码中寻找并且利用漏洞是一项非常具有挑战性的工作，它的挑战性主要在于人工很 难直观的看出二进制代码中的数据结构、控制流信息等。 Angr 是一个基于 python 的二进制漏洞分析框架，它将以前多种分析技术集成进来，它 能够进行动态的符号执行分析（如 KLEE 和 Mayhem），也能够进行多种静态分析。