《软件安全》实验报告

姓名：梁婧涵 学号：2112155 班级：1120

实验名称：Angr应用示例

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：

新建一个Angr工程，并且载入二进制文件

创建程序入口点的state

将要求解的变量符号化

创建SimulationManager进行程序执行管理

搜索满足我们目标的state

求解程序执行到state时，符号化变量所需要的约束条件

解出约束条件，获得目标值

1. 安装angr库，并输入解决代码，输出

#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;

}

2.Angr分析源码及API说明

（1）

import angr

import claripy

def main():

#新建一个工程，导入二进制文件，example就是文件的名字，后面的选项是选择不自动加载依赖项

proj = angr.Project('example',load\_options={"auto\_load\_libs":False})

#创建一个SimState对象，这也就是我们进行符号执行的核心对象，包括了符号信息、内存信息、寄存器信息等

state = proj.factory.entry\_state(add\_options={"SYMBPLIC\_WRITE\_ADDRESS"})

#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，这个地址是和二进制文件相关联的，使用IDA打开，此地址对应的.bss段全局变量u的地址

state.memory.store(0x601041,u) #我对代码进行过重新编译，因此和github提供的二进制文件的地址不相同

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

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

#step开始执行，直到路径超过一条的时候停止，也就是我们的if else那里

sm.step(until = lambda lpg:len(lpg.active) > 1)

#对于每个可能的state都进行检查，

for I in range(len(sm.active)):

print "possible %d : " %I , sm.active[I].state.se.any\_int(u)

return sm.active[0].state.se.any\_int(u)

if \_\_name\_\_ == "\_\_main\_\_":

print repr(main())

1. 在Angr中，我们使用state表示程序（二进制文件）的运行状态，这个状态机就包括内存、寄存器等。在Angr的API文档中，对entry\_state函数的定义如下：

entry\_state(\*\*kwargs)

返回一个代表着程序入口点的state对象。所有的参数都是可选的。

参数： .addr-state开始的程序地址（非入口点）

.initial\_prefix-如果提供了这个参数，所有的符号化寄存器以及符号变量的名字前面都带有此字符串

.fs-一个字典，字典的键为文件名，值为SimFile对象

.concrete\_fs-boolean；指明了在打开文件时，主文件系统是否应该被询问

.chroot- 一个伪造的root路径，和真实chroot指令等效。只有在concrete\_fs为True时生效

.argc- 程序所需要的argc。可以是int或者bitvector。如果不提供，默认为args的长度

.args- 程序所需要的参数列表。可以是string或者bitvector

.env- 字典；用于指明程序运行环境，键和值都可以是strings或者bitvector

返回值：初始状态

返回类型：SiMState 使用add\_options可以增加选项

（3）

使用.explore方法的find参数，程序会执行到满足find条件的地方停止，这个find条件可以是一个指令、一个地址、或者是一个满足某些约束的函数。当有任意一个state符合find条件时，这个state将被置入found区域，并且停止执行。接下来搜索found区域，就可以忽略此state继续执行，或者对次装填进行操作。当然可以制定avoid参数规避满足某些条件的state。

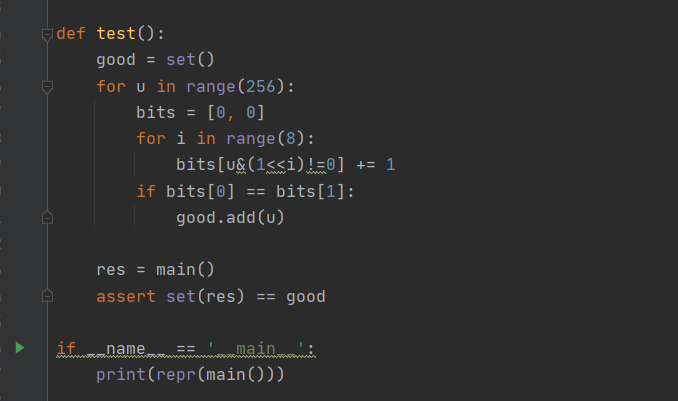
sm.explore(find=correct, avoid=wrong)的含义是，寻找到满足correct条件且不满足wrong条件的state，即我们最终所要的校验成功的state。获得到state之后，运行state.solver.eval(u)求解u的值。



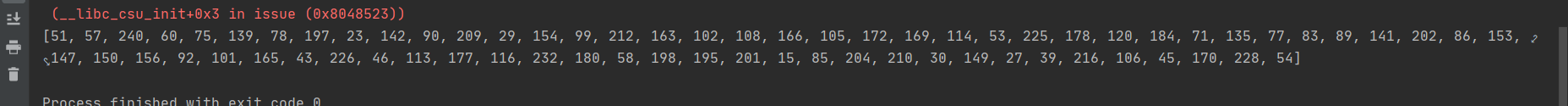
Wrong和correct函数是根据返回的结果中有没有lose和win返回bool值。

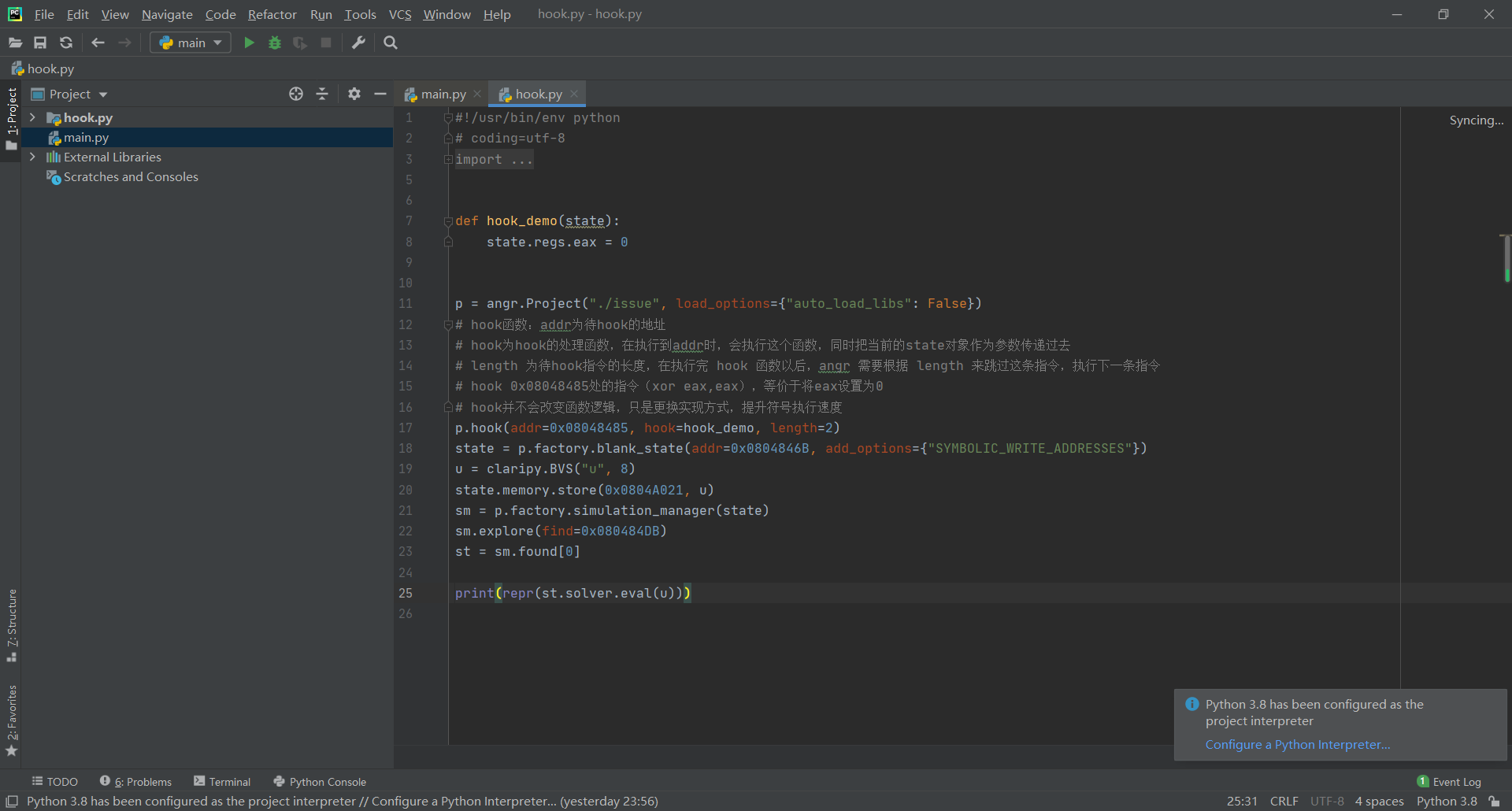
然后用explore函数进行状态搜寻，检查输出字符串是win还是lose

最后用eval\_upto获取多个返回结果



结果：







心得体会：

通过完成Angr应用示例实验，我对符号执行技术及其在二进制程序分析和漏洞挖掘中的应用有了更深刻的认识。以下是我在实验中的一些心得体会：

首先，Angr作为一个功能强大的符号执行框架，为二进制程序分析和漏洞挖掘提供了一个高效、灵活的工具。通过使用Angr，我学会了如何在符号执行环境中对二进制程序进行探索和分析，包括对程序的路径和状态进行建模，以及如何使用Angr提供的丰富的API来执行各种符号执行操作。

其次，Angr在解决复杂的二进制程序分析问题上展现出了出色的能力。在实验中，我尝试了使用Angr来解决不同类型的问题，如寻找程序的路径约束、发现程序中的漏洞、生成测试用例等。通过这些实践，我深刻认识到了符号执行技术在应对复杂程序分析问题时的独特优势，例如可以自动化地发现程序中的隐藏漏洞，对复杂的程序行为进行全面分析等。