《漏洞利用及渗透测试基础》实验报告

姓名：孙蕗 学号：2112060 班级： 信安1班

**实验名称：**

Angr应用示例

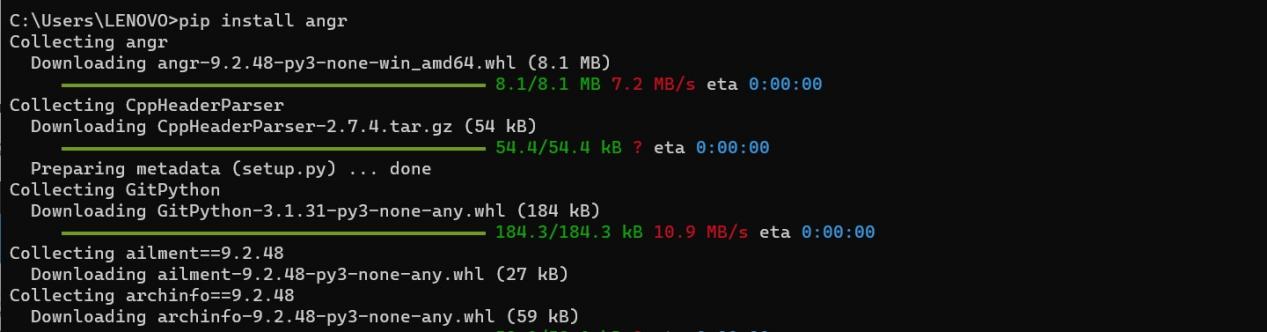
**实验要求：**

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：**

1. 下载安装包

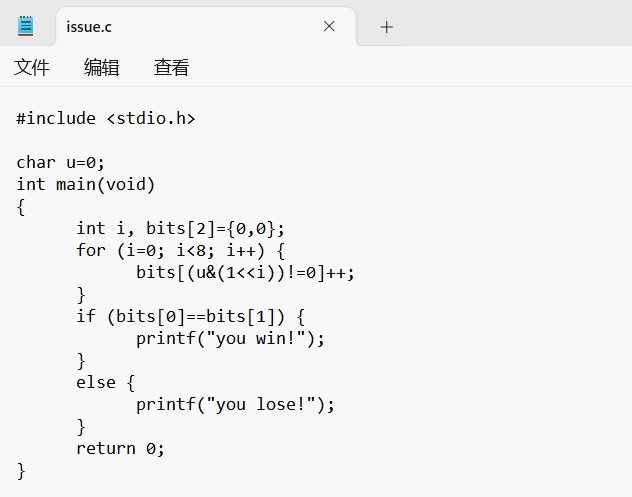
使用pip install angr命令安装



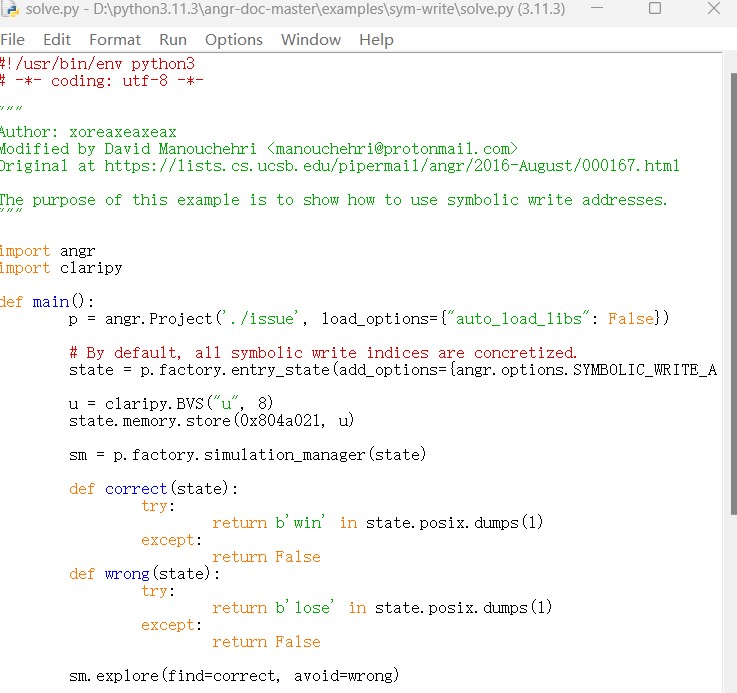
1. 测试安装

输入命令python，进入python界面，然后输入import angr，如果成功说明安装没问题

1. issue.c源代码



1. solve.py源代码



#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Author: xoreaxeaxeax

Modified by David Manouchehri <manouchehri@protonmail.com>

Original at https://lists.cs.ucsb.edu/pipermail/angr/2016-August/000167.html

The purpose of this example is to show how to use symbolic write addresses.

"""

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://doc.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

# By default, all symbolic write indices are concretized.

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close

# 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)

# Alternatively, you can hardcode the addresses.

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

# 获得state后，通过solver求解器，求解u的值

# eval\_upto(e,n,cast\_to=None,\*\*kwargs)求解一个表达式多个可能的求解方案，e为表达式，n为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为表达式

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

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

def test():

good = set()

for u in range(256):

bits = [0, 0]

for i in range(8):

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

if bits[0] == bits[1]:

good.add(u)

res = main()

assert set(res) == good

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

# repr()函数将object对象转化成string类型

print(repr(main()))

上述代码定义了一个main函数。

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

在上述Angr示例中，6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，有助于给符号执行带来麻烦。
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sm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。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。

注意：这里也可以用 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。

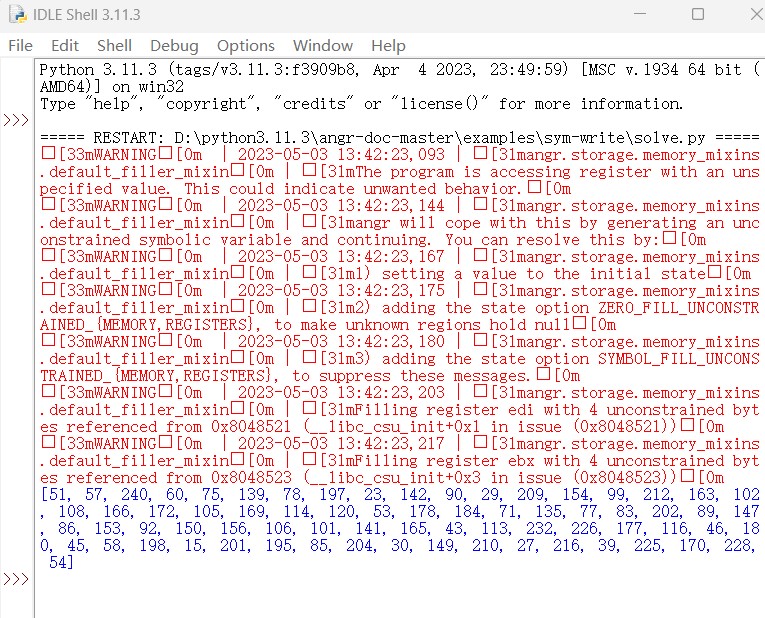
1. 获得state后，通过solver求解器，求解u的值。

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为表达式，n为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。

1. 实验验证

在Windows10环境下，选择填写的solve.py右击，选择Edit with IDLE->Edit with IDLE 3.11(64 bit)命令，在弹出的界面中选择Run->run model 命令。

蓝色部分即为输出的u的求解结果，因为采用了eval\_upto函数，给出了多个解，对每个解都可以代入源程序进行验证。



1. 其他解法

# ！/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的待处理函数，在执行到addr时，会执行这个函数，同时把当前的state对象作为参数传递出去

# length为待Hook指令长度，在执行完hook函数后，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)

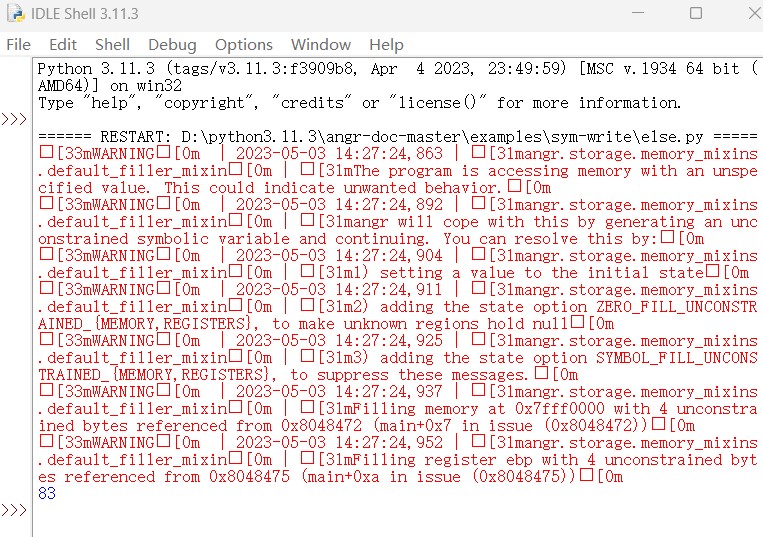
sm.explore(find=0x080484DB)

st = sm.found[0]

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

上述代码与前面的解法有3个区别。

1. 采用了hook函数，将0x080485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，如printf等，可以进行Hook，提升符号执行的性能。
2. 进行符号执行得到想要的状态，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。
3. 最后，eval（u）替代了原来的eval\_upto，将打印一个结果出来。
4. 其他解法运行结果



**心得体会：**

Angr的使用：首先安装angr,分析二进制文件，进行符号执行。然后利用angr进行漏洞发现和漏洞利用。angr的fuzzer工具可以用于自动发现程序中的漏洞，angr中的exploit工具可以用于自动生成利用代码，以利用程序中的漏洞。