《软件安全》实验报告

姓名：曹瑜　　学号：2212794　　班级：密码科学与技术

实验名称：

Angr 应用示例

实验要求：

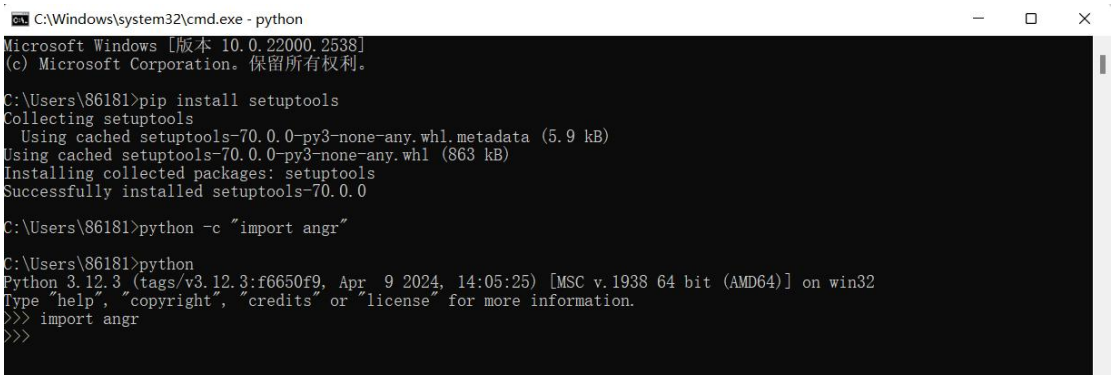
根据课本 8.4.3 章节，复现 sym-write 示例的两种 angr 求解方法，并就如何使用 angr 以及怎么解决一些实际问题做一些探讨。

实验过程：

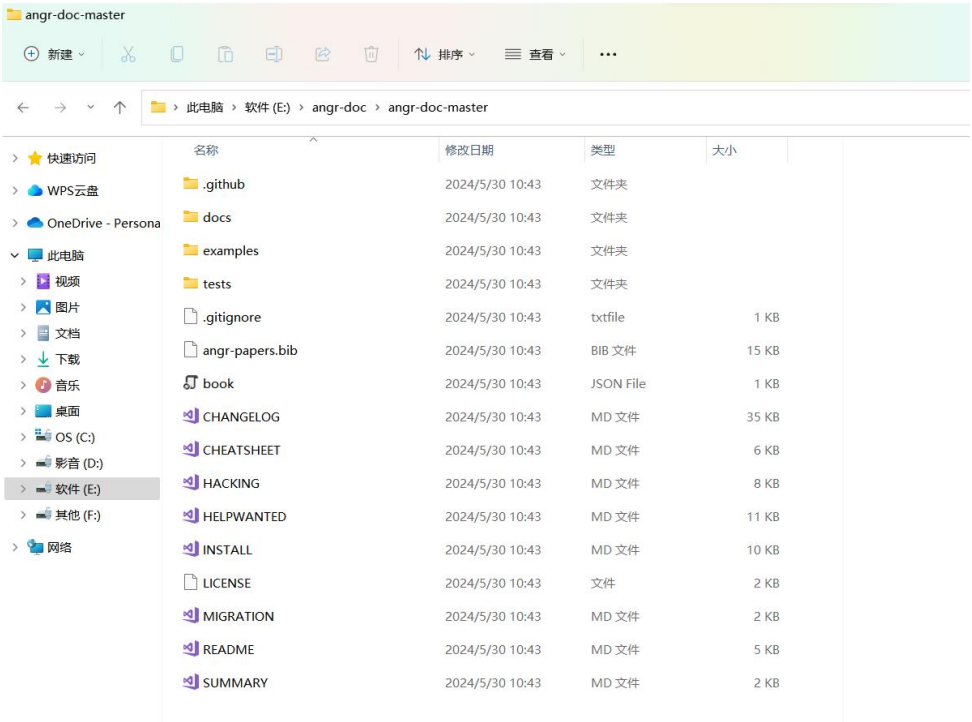
1、Angr 下载

下载 python3.12.3 后在控制台输入 pip install angr 安装 angr

由于 python3.12.3 中删除了 distutils 模块，故手动添加，之后 import angr 成功



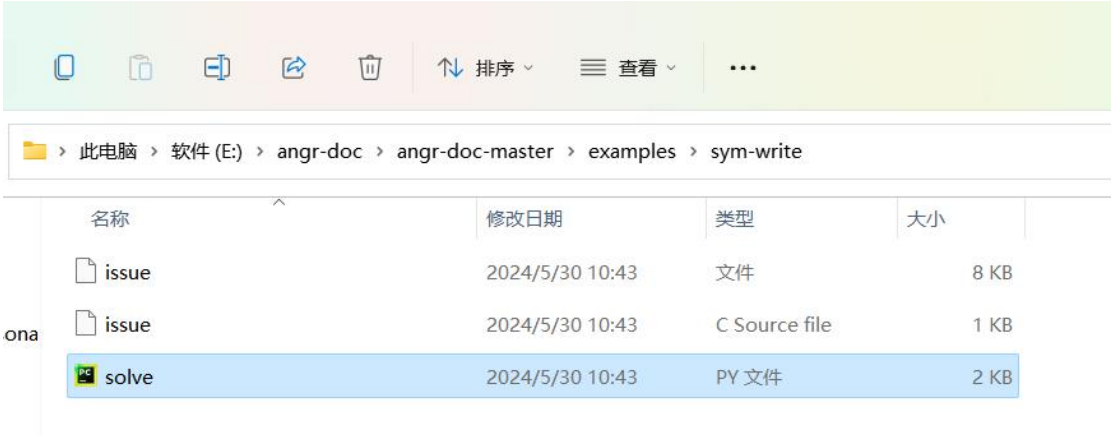
2、下载 angr-doc：



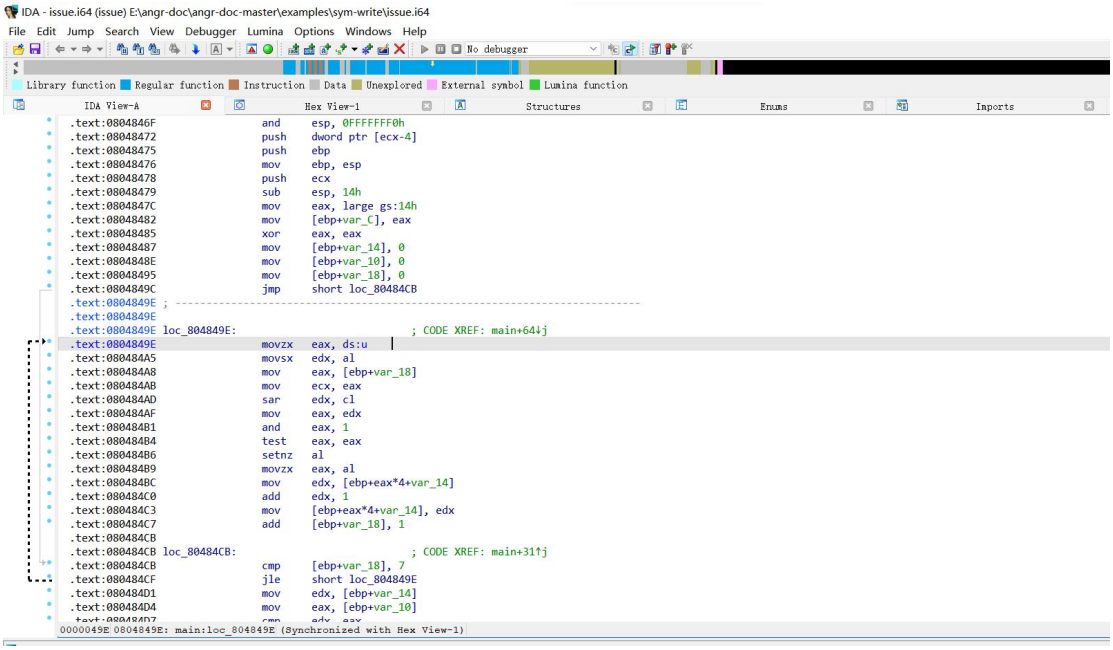
3、复现 sym-write：

解法一：

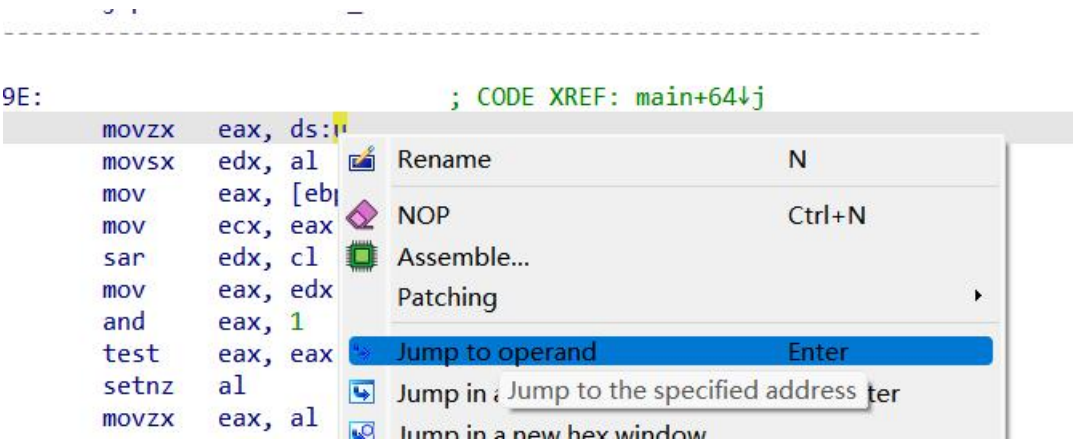
可见已生成了 solve.py



用 IDA pro 打开 isssue 文件进入文本视图：



跳转到 u



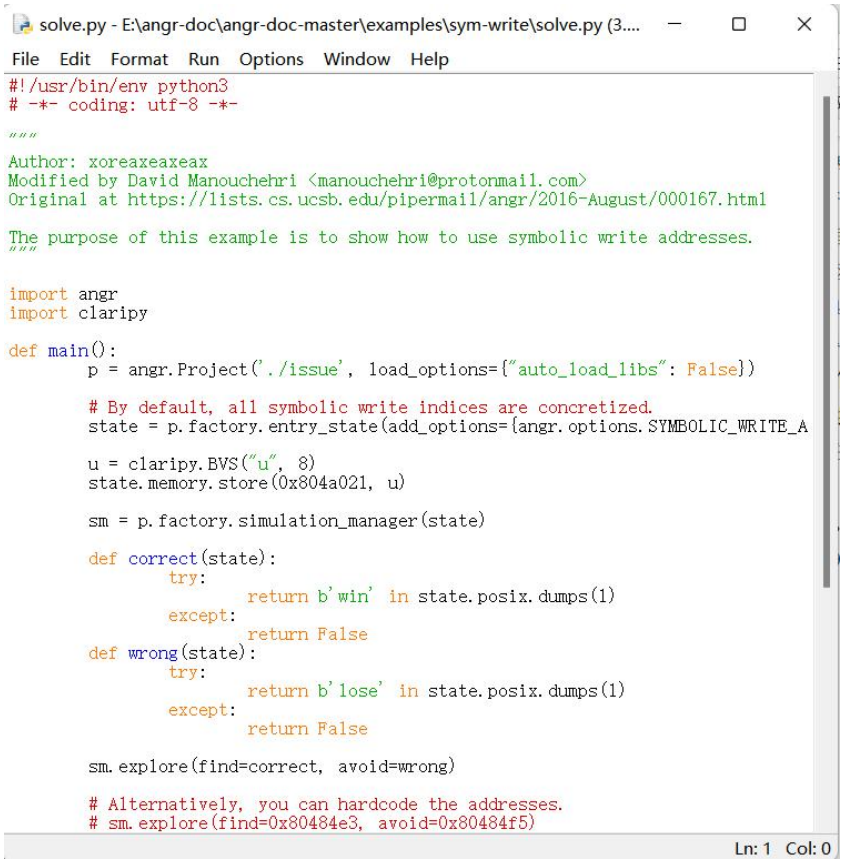
可看到 u 的地址：与 solve 中相同


```
# 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)
```

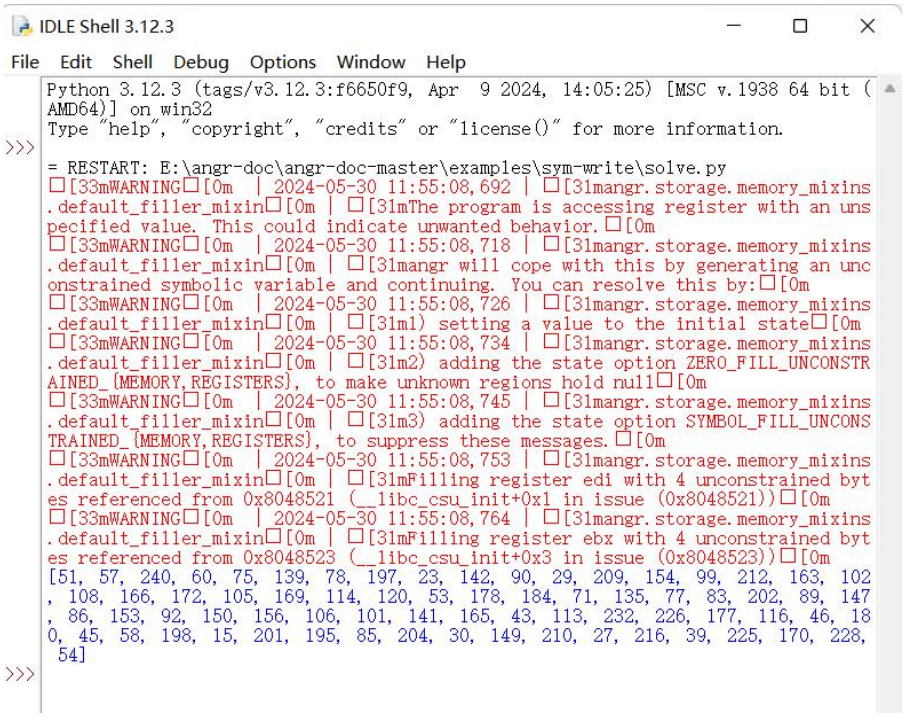
Sm.explore 会搜寻满足目标条件的状态，

一种是进行符号执行来遍历，另一种是通过地址进行定位

使用 IDLE 打开 solve.py



run model 后可看到 u 求解结果：



解法二：

使用 hook，进行符号执行得到想要的状态，打印出一个结果

solve1.py - E:/angr-doc/angr-doc-master/examples/sym-write/solve1.py (3.12.3)

File Edit Format Run Options Window Help

```
#!/usr/bin/env python3
# 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)))
```

IDLE Shell 3.12.3

File Edit Shell Debug Options Window Help

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>

= RESTART: E:/angr-doc/angr-doc-master/examples/sym-write/solve1.py
□[33mWARNING□[0m  | 2024-05-30 12:09:27,284 | □[31mangr.storage.memory_mixins
.default_filler_mixin□[0m | □[31mThe program is accessing memory with an unspe
cified value.  This could indicate unwanted behavior.□[0m
□[33mWARNING□[0m  | 2024-05-30 12:09:27,301 | □[31mangr.storage.memory_mixins
.default_filler_mixin□[0m | □[31mangr will cope with this by generating an unc
onstrained symbolic variable and continuing.  You can resolve this by:□[0m
□[33mWARNING□[0m  | 2024-05-30 12:09:27,309 | □[31mangr.storage.memory_mixins
.default_filler_mixin□[0m | □[31m1) setting a value to the initial state□[0m
□[33mWARNING□[0m  | 2024-05-30 12:09:27,314 | □[31mangr.storage.memory_mixins
.default_filler_mixin□[0m | □[31m2) adding the state option ZERO_FILL_UNCONSTR
AINED_{MEMORY,REGISTERS}, to make unknown regions hold null□[0m
□[33mWARNING□[0m  | 2024-05-30 12:09:27,323 | □[31mangr.storage.memory_mixins
.default_filler_mixin□[0m | □[31m3) adding the state option SYMBOL_FILL_UNCONS
TRAINED_{MEMORY,REGISTERS}, to suppress these messages.□[0m
□[33mWARNING□[0m  | 2024-05-30 12:09:27,330 | □[31mangr.storage.memory_mixins
.default_filler_mixin□[0m | □[31mFilling memory at 0x7fff0000 with 4 unconstra
ined bytes referenced from 0x8048472 (main+0x7 in issue (0x8048472))□[0m
□[33mWARNING□[0m  | 2024-05-30 12:09:27,340 | □[31mangr.storage.memory_mixins
.default_filler_mixin□[0m | □[31mFilling register ebp with 4 unconstrained byt
es referenced from 0x8048475 (main+0xa in issue (0x8048475))□[0m
83
>>>
```

心得体会：

通过本次实验，成功在 windows 上安装了 angr，并成功复现了 sym-write 示例的两种 angr 求解方法，在安装 angr 时可能遇到依赖问题，如缺少模块或版本不兼容，要确保使用最新的 pip 和 setuptools 版本，符号执行过程中遇到外部函数或系统调用时，可能导致执行中断或状态不准确，可以使用 angr 的钩子功能来模拟外部函数的行为，通过实验和实际应用，不仅掌握了 angr 的基本使用方法，还学会了如何应对符号执行过程中常见的问题。