《漏洞利用及渗透测试基础》实验报告

姓名： 艾明旭 学号：2111033 班级：信息安全一班

**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

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。

**实验过程：**

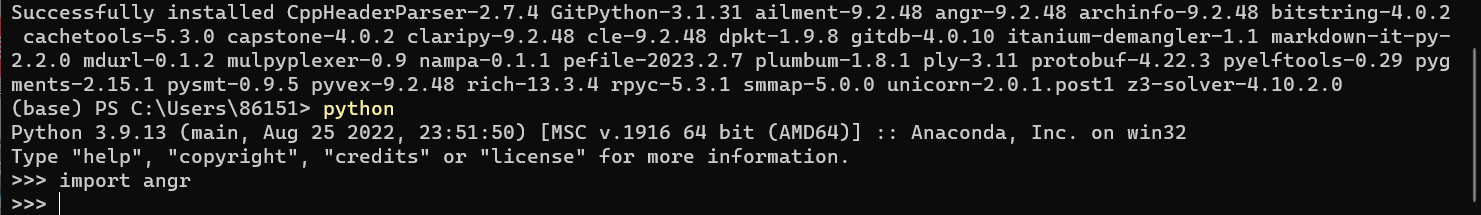
1. **安装Angr**

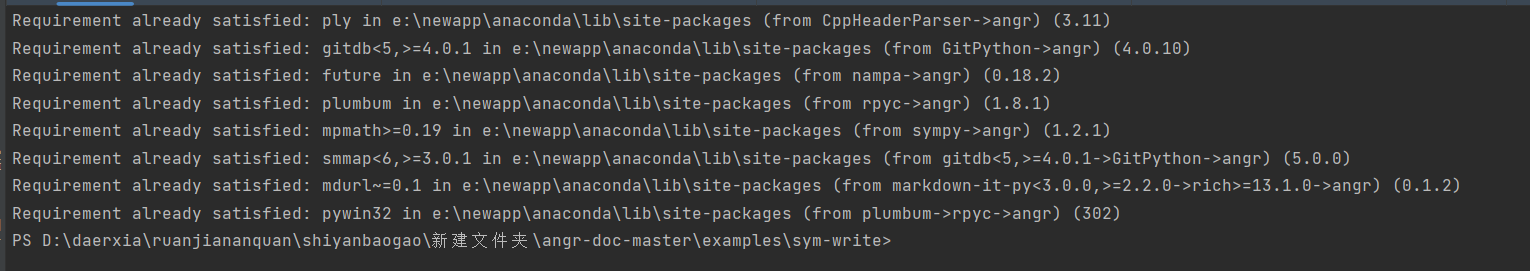
首先安装python3，再选择将 python 增加到 path 中，然后，打开命令控制台，使用 PIP 命令安装

angr：pip install angr

**测试安装。**输入命令 python，进入 python 界面，然后输入 import angr，成功，

说明安装没有问题





将 https://github.com/angr/angr-doc 里的所有文档以 zip 方式下载到本地。

在 angr-doc 里有各类 Example，展示了 Angr 的用法，比如 cmu\_binary\_bomb、simple\_heap\_overflow 等二进制爆破、堆溢出等漏洞挖掘、软件分析的典型案例。下面，我们以 sym-write 为例子，来说明 angr 的用法。

**Angr 示例** 以 sym-write 为例子，来说明 angr 的用法

源码 issue.c如下：

#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;

}

源码solve.py如下，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

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})//创建工程 指定启动项

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，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})//默认从入口点开始执行

# 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)//通过函数调用完成

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. 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)//搜索满足状态的特定结果（包含win的 ）

# 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

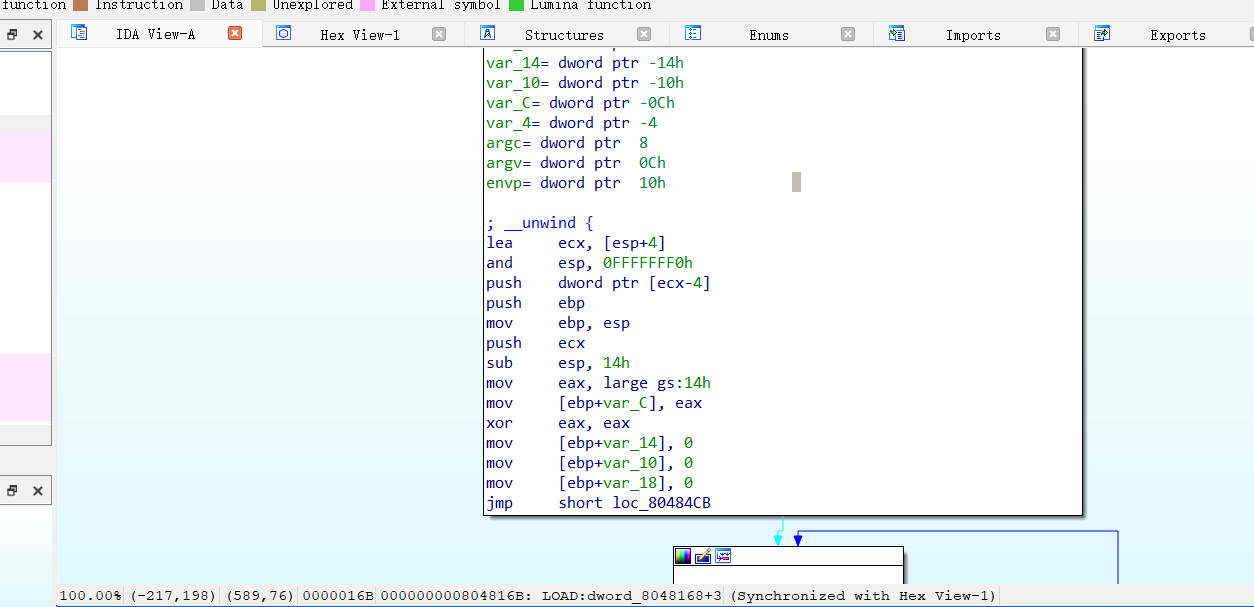
return sm.found[0].solver.eval\_upto(u, 256)//256为求解结果最大数量

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

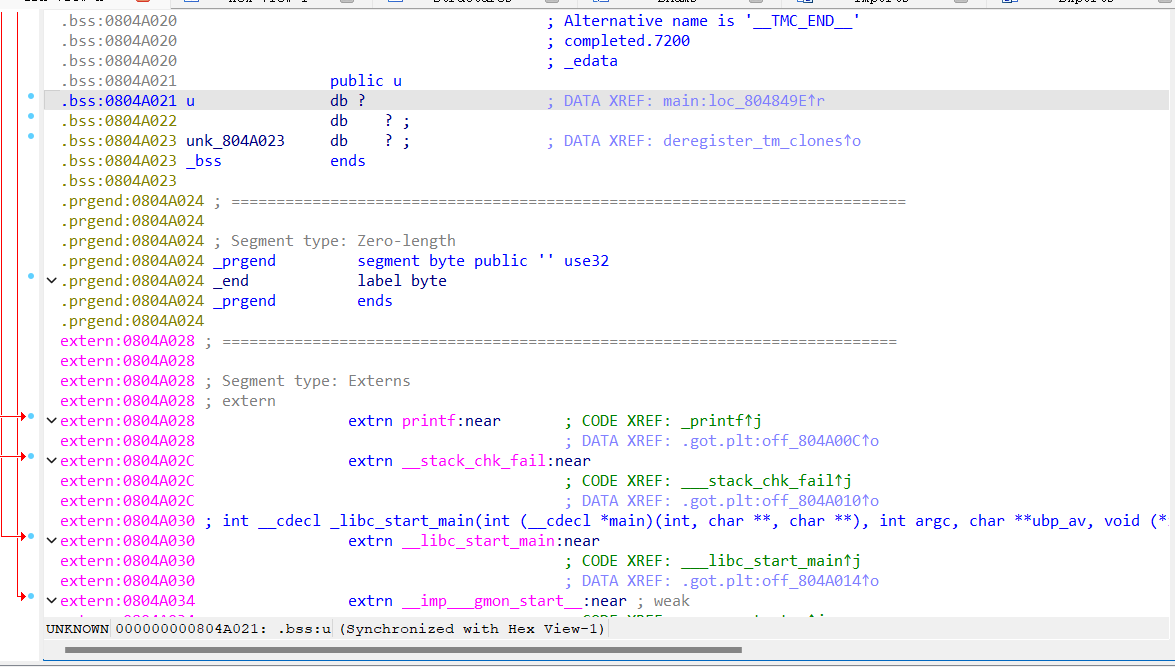
# repr()函数将object对象转化为string类型

print(repr(main()))

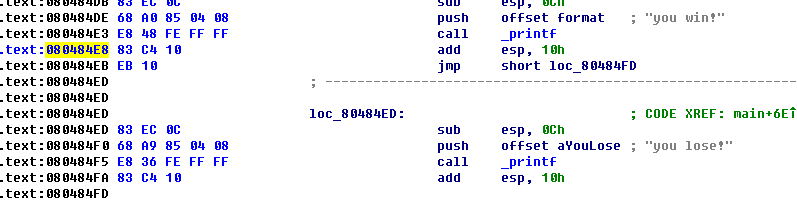
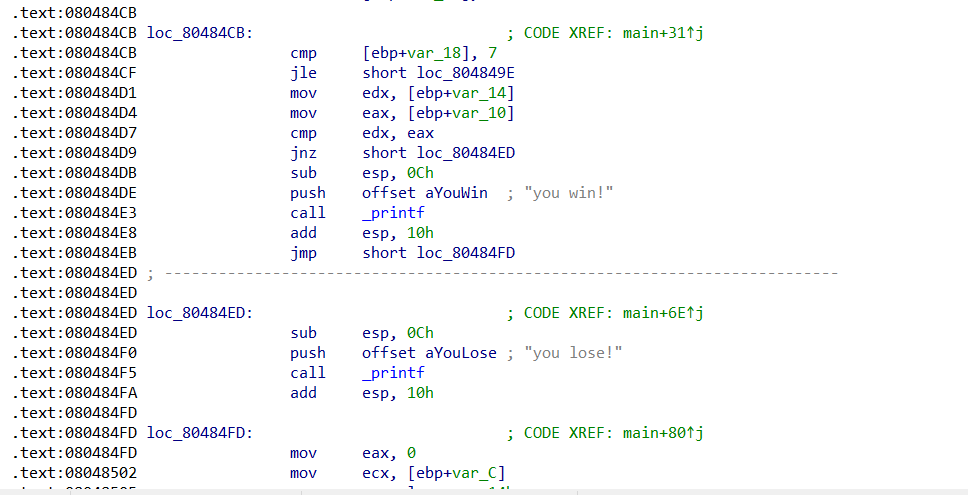
用idaFree打开issue文件



可以看到u的地址是0x804A021

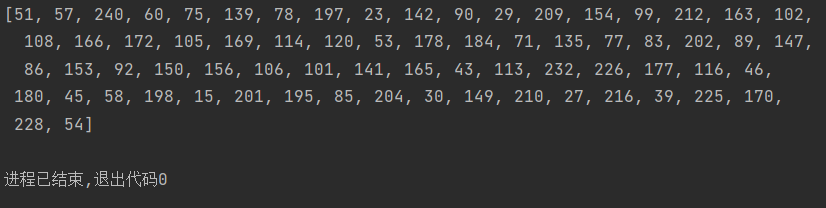


除了通过函数sm.explore(find=correct, avoid=wrong)， 也可以通过直接定位地址得到想要的状态0x80484DE对应输出you win，0x80484F0对应 you lose



**运行solve.py**

所求u的结果如下



**其他解法。**对于上述程序，也可以采用下面的代码来进行求解：

|  |
| --- |
| #!/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)//原指令不运行，运行hook\_demo，指令长度为2    这样便于进行复杂处理  state = p.factory.blank\_state(addr=0x0804846B, add\_options={"SYMBOLIC\_WRITE\_ADDRESSES"})//指定开始位置  u = claripy.BVS("u", 8)//定义符号变量  state.memory.store(0x0804A021, u)  sm = p.factory.simulation\_manager(state)  sm.explore(find=0x080484DB)//因为有if else分支，不用avoid    st = sm.found[0]  print(repr(st.solver.eval(u))) |

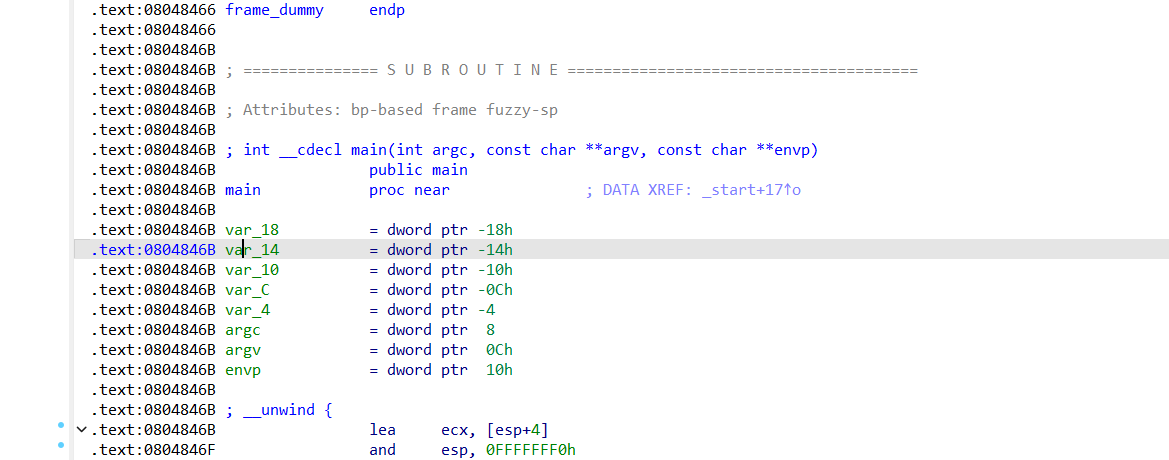
上述代码与前面的解法有三处区别：

（1）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。

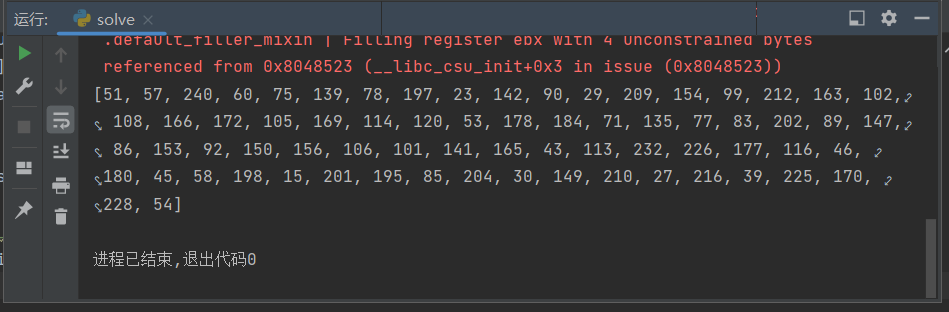
（2）进行符号执行得到想要的状态，有变化，变更为find=0x080484DB。因为源程序win和lose是互斥的，所以，只需要给定一个find条件即可。

（3）最后，eval(u)替代了原来的eval\_upto，将打印一个结果出来。

0x0804846B，整个程序的入口点



运行结果如下图：



总结：

如何使用angr:angr也叫符号执行暴力破解。打开python，导入anger :import angr，然后进行初始化，载入文件，创建对象，传参进行操作等

应用场合：逆向分析，漏洞挖掘等

**心得体会：**

初步了解angr如何使用，知道了所给angr示例的暴力破解的逻辑、blank\_state()等函数的作用

Angr作为一个二进制分析框架的基本特性和优势，学习了Angr的建模能力、符号执行技术和状态管理等核心内容。

Angr具备了很强的建模能力，可以将二进制程序抽象成一个符号状态，通过符号变量和符号表达式来描述程序执行状态。这种建模方式使得Angr可以处理复杂的程序逻辑，例如分支、循环、条件判断等，具有很强的灵活性和可扩展性。

Angr采用了符号执行技术，可以自动化地寻找程序的执行路径、判断程序输入、输出和状态等，可以有效地优化程序分析过程，减少人工干预，提高分析效率。Angr还提供了丰富的分析工具和API，可以对程序进行静态和动态分析，生成符号执行图、寻找缺陷、反汇编代码等。

总之，在本次实验中，我理解了Angr框架的核心思想和工作方式。我认为Angr可以帮助我们更好地理解和分析程序行为，并且可以提高分析效率和精度。Angr作为一个二进制分析框架，具有重要的实用和研究价值。