《软件安全》实验报告

姓名：冯思程 学号：2112213 班级： 计算机科学与技术二班

**实验名称：**

Angr应用示例

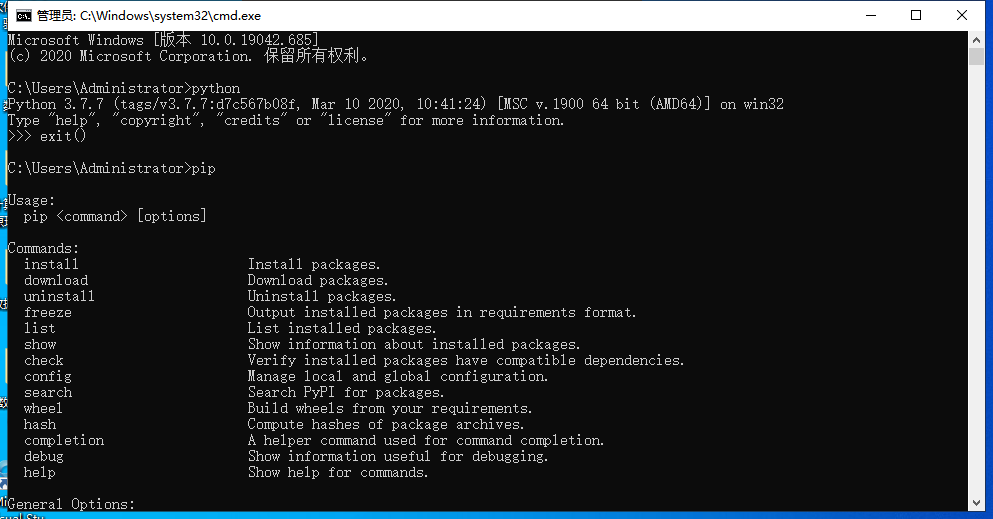
**实验要求：**

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。

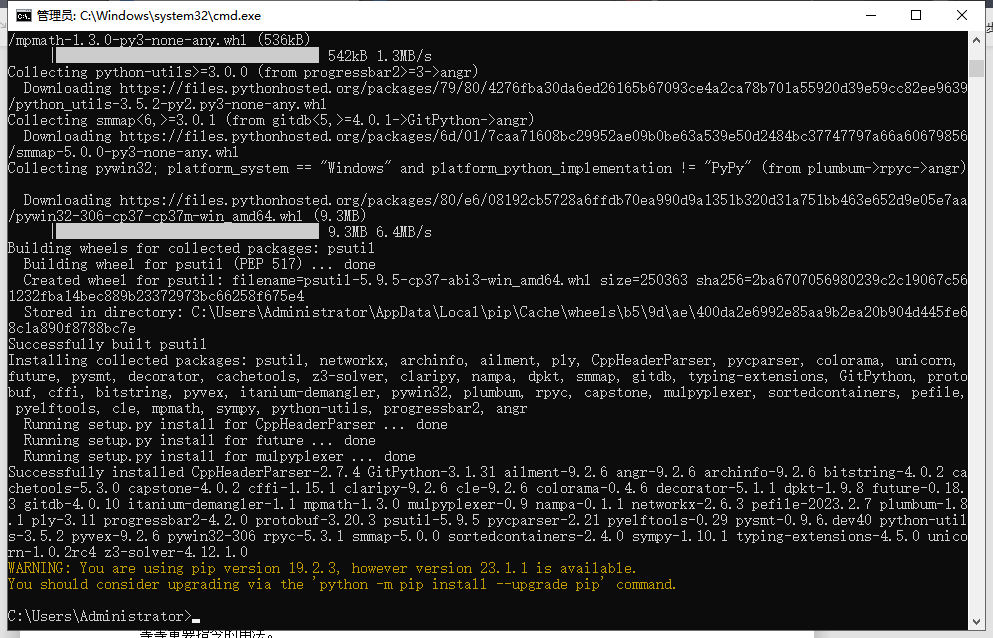
**实验过程：**

1. **首先进行Angr工具的安装：**

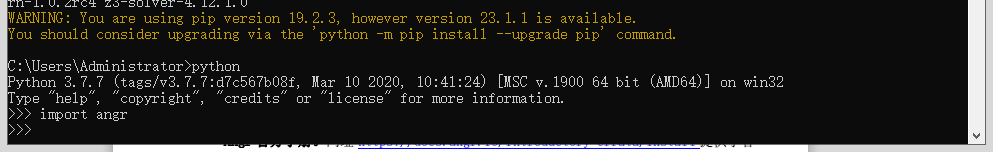
首先发现需要安装python环境，检查发现没有安装python环境，所以先进行python安装，进入python官网，选择好适合的版本下载，然后在命令行工具中检查，成功安装好了，如下图：



然后进行Angr工具安装，在命令行中使用命令：pip install angr进行安装，如下图：



验证一下Angr工具已经安装成功，如下图：



然后下载angr-doc文档，下载后解压，如下图：



1. **复现sym-write示例的两种angr 求解方法：**

程序源码如下：

#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;

}

在这个程序其中会根据对条件的判断输出win还是lose，然后不确定的u的取值会影响其条件判断，然后我们试图找出来u的值使得输出win。

**法一：**然后我们编写一个执行程序进行实验，其中我们主要定义了一个main函数，并将main函数的返回值转换成string类型打印出来。其中main函数的关键部分如下：

（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（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（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）进行符号执行得到想要的状态，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。 （6）获得到 state 之后，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-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；eval\_one(e, \*\*kwargs)求解表达式以获得唯一可能的解决方案。代码如下：

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

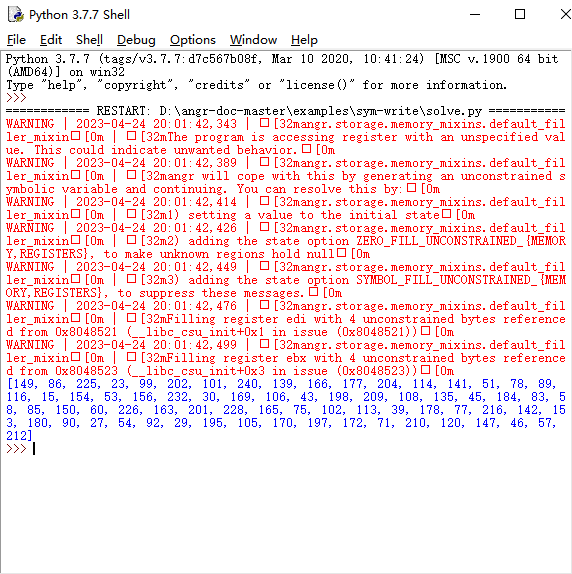
return sm.found[0].solver.eval\_upto(u, 256)

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

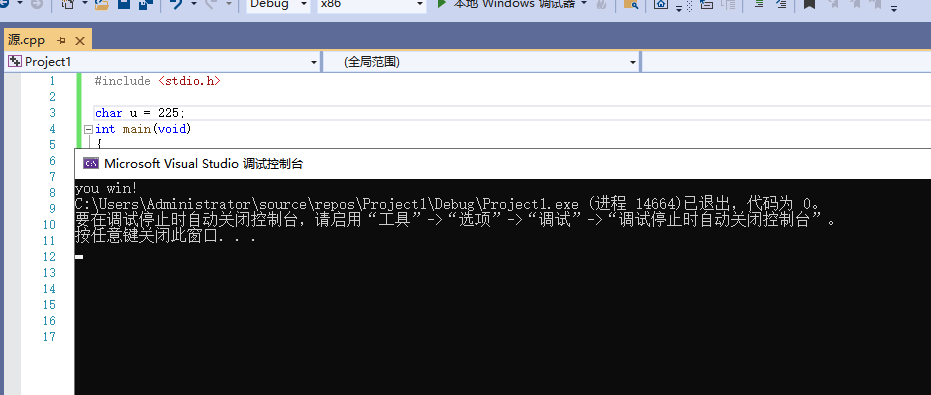
# repr()函数将 object 对象转化为 string 类型

print(repr(main()))

在windows 10环境下，选择填写的solve.py，点右键选择Edit with IDLE Edit with IDLE 3.7 (64 bit)，将弹出界面，选择 Run run model，结果如下：



蓝色部分就是最终的解，由于使用了eval\_upto函数，所以可以输出多个解，每个解都可以带回到源程序进行验证，将u带成225进行验证，结果如下：



**法二：**代码如下：

#!/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)))

上述代码与前面的解法有三处区别：

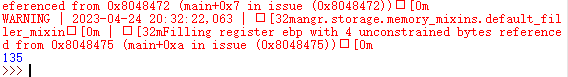
 采用了 hook 函数，将 0x08048485 处的长度为 2 的指令通过自定义的 hook\_demo 进行替代，功能是一致的，原始xor eax, eax和state.regs.eax = 0是相同的作用，

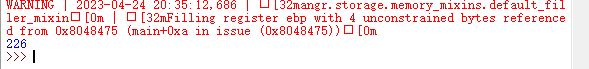
这里只是演示，可以将一些复杂的系统函数调用，比如printf等，可以进行hook，提升符号执行的性能。 

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。 

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

新建一个txt文件将上述代码存入（注意保证缩进和注释完整），在保存的时候将文件类型写成.py，命名为solve2，然后选择填写的solve2.py，点右键选择Edit with IDLE Edit with IDLE 3.7 (64 bit)，将弹出界面，选择 Run run model，结果如下：





第二种方法求解只会输出一个值，而且是第一种方法结果中的一个。使用hook跳过一定的步骤但是不改变算法逻辑，根据程序本身win和lose是互斥的，得出一个满足条件便退出了。

1. **如何使用angr以及怎么解决一些实际问题：**

使用 angr 的大概步骤:

1)创建 project

2)设置 state

3)建 符号量 : BVS (bitvector symbolic ) 或 BVV (bitvector value)

4)把符号量设置到内存或者其他地方

5)设置 Simulation Managers ， 进行路径探索的对象

6)运行，探索满足路径需要的值

7)约束求解，获取执行结果

Angr强大的二进制程序分析功能在近几年CTF中大放异彩，可以用来恢复控制流图，进行二进制分析与强化、获取程序控制流（CFG)、生成rop链、发现漏洞、加密程序、污点跟踪等等。

对一些应用进行具体的说明：

1）寻找程序中的漏洞：使用 Angr 可以模拟程序的执行过程，从而找出可能的漏洞，例如缓冲区溢出、格式化字符串漏洞等。

2）寻找程序中的密码：如果你需要找出程序中的硬编码密码，可以使用 Angr 对程序进行符号执行，然后在符号执行过程中检查密码的条件。

3）寻找程序中的特定代码路径：如果你需要找到程序中的特定代码路径，例如需要在特定的输入下触发某个特定的代码块，可以使用 Angr 进行符号执行来查找该代码路径。

4）反混淆：有些恶意软件会使用混淆技术来隐藏其真实代码，可以使用 Angr 对程序进行符号执行并分析其执行路径，从而还原混淆后的代码。

5）软件保护：如果你需要保护自己的软件，可以使用 Angr 对程序进行符号执行，并添加额外的约束条件，以防止程序被反汇编或破解。

**心得体会：**

当我完成 Angr 中 sym-write 示例的实验后，我深深感受到了符号执行和约束求解的强大和魅力。

符号执行的思想很有意思：Angr 中的符号执行思想非常有趣，它将程序状态和输入参数都视为符号变量，然后通过约束条件和执行路径来求解这些符号变量的具体取值。这种思想不仅能够实现各种有趣的功能，还能够让我们重新审视程序的执行过程和运行逻辑。

Angr 的 API 非常灵活：Angr 提供了非常灵活和强大的 API，包括符号变量的创建、约束条件的添加、状态的更新等。在实验中，我熟悉了和掌握了这些 API 的基本用法，并且发现它们可以让我们轻松实现符号执行和约束求解功能。

不同的求解方法各有优缺点：在实验中，我尝试了两种不同的求解方法，这些方法各有优缺点，我需要根据具体的应用场景和求解需求来选择合适的方法。例如，如果需要快速求解而且只需要有一个解，可以使用第二种方法；如果需要全部解，可以使用第一种方法。

Angr 的原理和机制很有意思：除了 API 和求解方法之外，我还深入了解了 Angr 的原理和机制，包括程序状态的模拟、路径的约束和搜索、路径的剪枝和优化等。通过深入理解这些机制，我更好地掌握了 Angr 的应用和技术，也能够更好地解决实际问题。

总之，Angr 是一个非常强大和有用的二进制分析工具，通过实验和实践，我更深入地理解了 Angr 的应用和技术，也能够更好地解决实际问题。我相信，在以后的学习和实践中，我会进一步掌握 Angr 的技术和工具，也会发现更多有趣的应用和领域。