

《软件安全》实验报告

姓名：王昱

学号：2212046

班级：信息安全班

一、实验名称：

Angr 应用示例

二、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。

三、实验原理：

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。

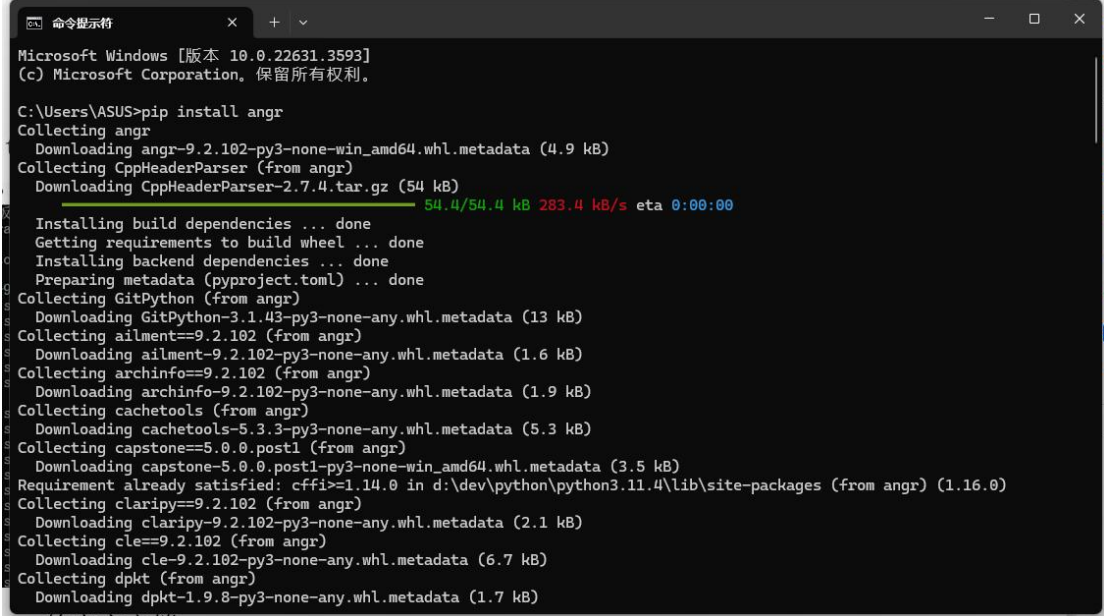
Angr 是一个基于 python 的二进制漏洞分析框架，它将以前多种分析技术集成进来，它能够进行动态的符号执行分析（如 KLEE 和 Mayhem），也能够进行多种静态分析。

四、实验过程：

（一）在 Windows 环境下安装 Angr

1. 安装 python3。

2. 下载好 python 后，使用 PIP 命令安装 angr: `pip install angr`



```
Microsoft Windows [版本 10.0.22631.3593]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\ASUS>pip install angr
Collecting angr
  Downloading angr-9.2.102-py3-none-win_amd64.whl.metadata (4.9 kB)
Collecting CppHeaderParser (from angr)
  Downloading CppHeaderParser-2.7.4.tar.gz (54 kB)
    54.4/54.4 kB 283.4 kB/s eta 0:00:00
Installing build dependencies ... done
Getting requirements to build wheel ... done
Installing backend dependencies ... done
Preparing metadata (pyproject.toml) ... done
Collecting GitPython (from angr)
  Downloading GitPython-3.1.43-py3-none-any.whl.metadata (13 kB)
Collecting ailment==9.2.102 (from angr)
  Downloading ailment-9.2.102-py3-none-any.whl.metadata (1.6 kB)
Collecting archinfo==9.2.102 (from angr)
  Downloading archinfo-9.2.102-py3-none-any.whl.metadata (1.9 kB)
Collecting cachetools (from angr)
  Downloading cachetools-5.3.3-py3-none-any.whl.metadata (5.3 kB)
Collecting capstone==5.0.0.post1 (from angr)
  Downloading capstone-5.0.0.post1-py3-none-win_amd64.whl.metadata (3.5 kB)
Requirement already satisfied: cffi>=1.14.0 in d:\dev\python\python3.11.4\lib\site-packages (from angr) (1.16.0)
Collecting claripy==9.2.102 (from angr)
  Downloading claripy-9.2.102-py3-none-any.whl.metadata (2.1 kB)
Collecting cle==9.2.102 (from angr)
  Downloading cle-9.2.102-py3-none-any.whl.metadata (6.7 kB)
Collecting dpkt (from angr)
  Downloading dpkt-1.9.8-py3-none-any.whl.metadata (1.7 kB)
```

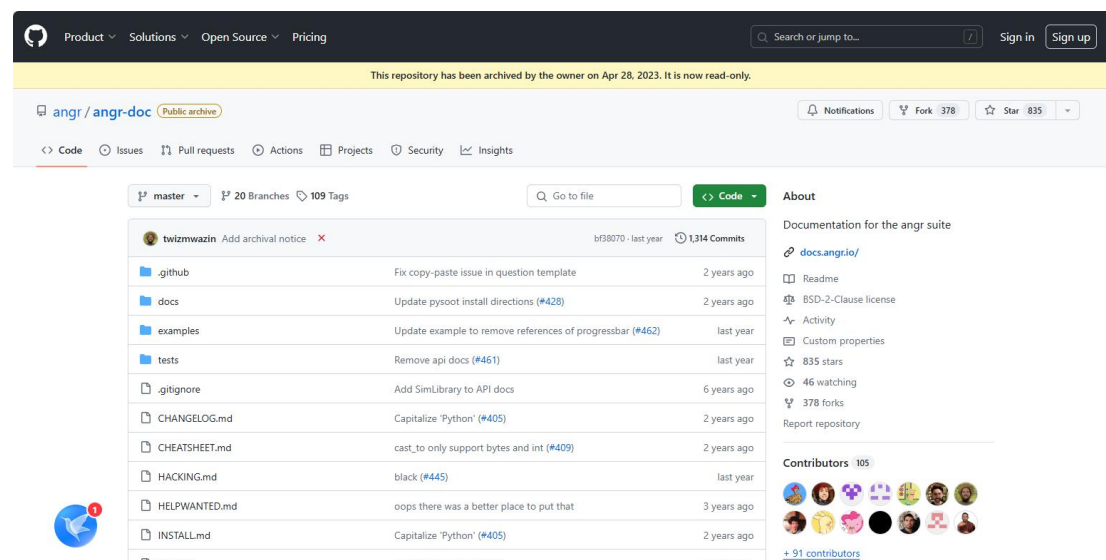
3. 验证是否安装成功

```
C:\Users\ASUS>python
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import angr
>>>
```

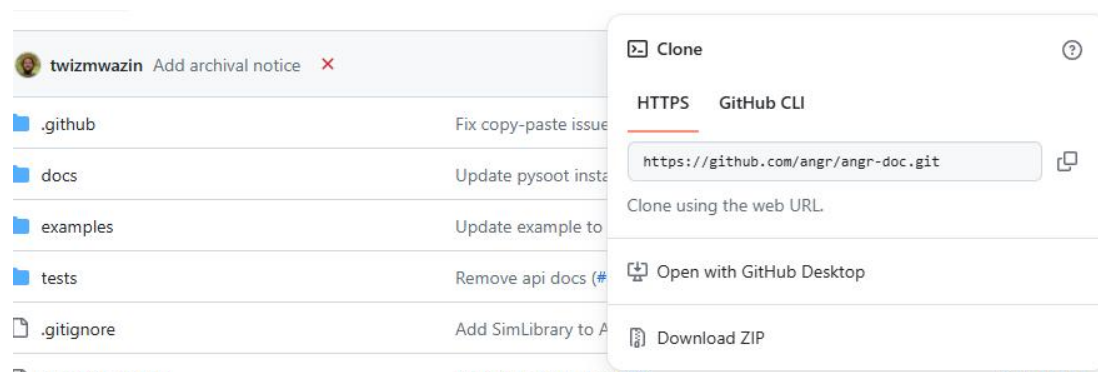
进入 python 界面后 import angr 没有报错，说明安装成功！

4. 下载 Angr 官方手册

进入 GitHub 上 angr 的开源项目 <https://github.com/angr>:

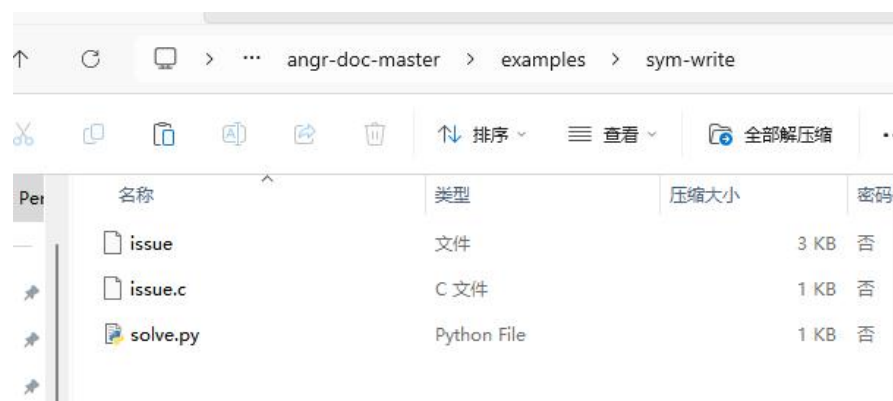


下载其中 docs 的所有文件



(二) Angr 求解方法示例一

1. 以 sym-write 为例子，来说明 angr 的用法。首先打开 angr 的开源文档下的 angr-doc-master\examples\sym-write



2.Issue.c 文件分析:

```
1. #include <stdio.h>
2. char u=0;
3. int main(void)
4. {
5.     int i, bits[2]={0,0};
6.     for (i=0; i<8; i++) {
7.         bits[(u&(1<i))!=0]++;
8.     }
9.     if (bits[0]==bits[1]) {
10.        printf("you win!");
11.    }
12.    else {
13.        printf("you lose!");
14.    }
15.    return 0;
16. }
```

我们需要实现的是, 利用 angr 找到能够输出 “you win!” 的 u 值

3.solve.py 分析:

```
1. import angr
2. import claripy
3.
4. def main():
5.     p = angr.Project('./issue', load_options={"auto_load_libs": False})
6.     u = claripy.BVS("u", 8)
7.     state.memory.store(0x804a021, u)
8.     sm = p.factory.simulation_manager(state)
9.     def correct(state):
10.         try:
11.             return b'win' in state.posix.dumps(1)
12.         except:
13.             return False
14.     def wrong(state):
15.         try:
16.             return b'lose' in state.posix.dumps(1)
17.         except:
18.             return False
19.
20.     sm.explore(find=correct, avoid=wrong)
21.     return sm.found[0].solver.eval_upto(u, 256)
22.
23. if __name__ == '__main__':
24.     # repr()函数将 object 对象转化为 string 类型
25.     print(repr(main()))
```

在上述 Angr 示例中，几个关键步骤如下：

(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 可以通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。

(6) 获得到 state 之后, 通过 solver 求解器, 求解 u 的值。

这里有多函数可以使用，`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。

4. 上述代码验证实验。选择 solve.py, 点右键选择 Edit with IDLE→Edit with IDLE 3.11 (64 bit), 将弹出界面, 选择 Run→run module, 界面如下:

The image shows two side-by-side Windows command prompt windows. The left window is titled 'solve.py - C:\Users\ASUS\Desktop\solve.py (3.11.5)' and contains a Python script. The script defines a function 'main()' that performs several steps: 1. Create a project and load a binary file 'state'. 2. Initialize a 'SimState' object and set initial values for 'blank_state', 'entry_state', and 'add_options'. 3. Create a symbol variable 'u' and set its value to '0'. 4. Create a 'SimulationManager' object. 5. Use 'exploit' to find a 'win' condition. 6. Use 'exploit' to find a 'lose' condition. 7. Use 'exploit' to find a 'correct' condition. The right window is titled 'IDLE Shell 3.11.5' and shows the output of the script. It includes warnings about memory mixins and a list of memory addresses that are 'unconstrained'.

```

solve.py - C:\Users\ASUS\Desktop\solve.py (3.11.5)
File Edit Format Run Options Window Help

import angrr
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
    p = angrr.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、
    # blank_state：可通过给定参数addr的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state中，很多选项说明见https
    # SYMBOLIC_WRITE_ADDRESSES：允许通过具体地址来动态处理符号地址的写操作
    state = p.factory.entry_state(add_options=[angrr.options.SYMBOLIC_WRITE_A

    # 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')

    # 把符号变量u保存到指定的地址中，这个地址就是二进制文件中.bss段的地址
    state.memory.store(0x804a021, u)

    # 4. 创建一个SimulationManager对象，这个对象和我们的状态有关系
    sm = p.factory.simulation_manager(state)

    # 5. 使用exploit函数进行状态探索，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s
    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)

```

```

Python 3.11.5 (tags/v3.11.5:ecce6b9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits()" or "license()" for more information.

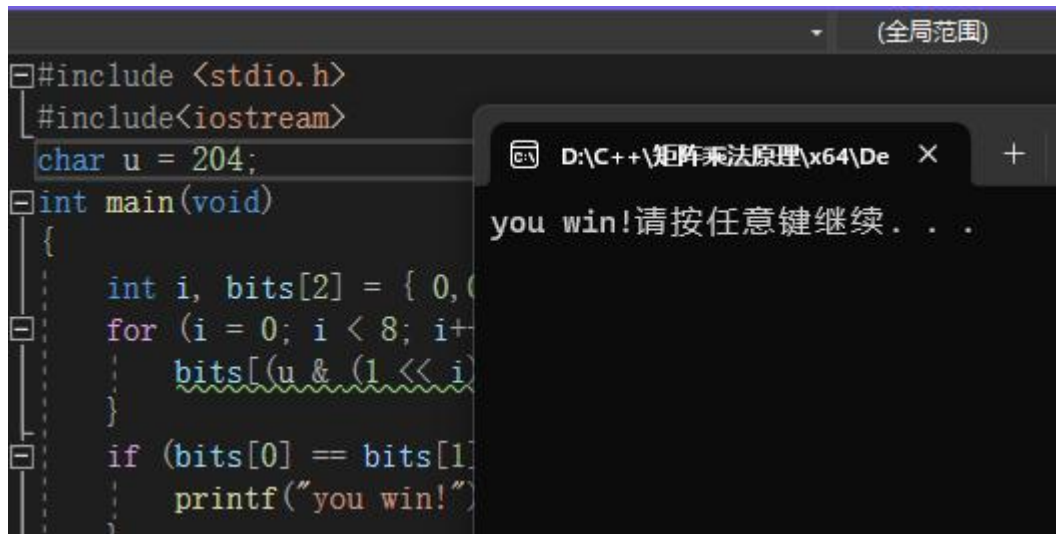
>>>
= RESTART: C:\Users\ASUS\Desktop\solve.py
[3mWARNING: [0m 2024-05-23 12:19:56.220 | [3m[angrr.storage.memory_mixins.d
default_filler_mixin [0m | [3m[13mthe program is accessing register with an unspeci
fied value. This could indicate unwanted behavior. [0m
[3mWARNING: [0m 2024-05-23 12:19:56.236 | [3m[angrr.storage.memory_mixins.d
default_filler_mixin [0m | [3m[13m[angrr will cope with this by generating an uncon
strained symbolic variable and continuing. You can resolve this by: [0m
[3mWARNING: [0m 2024-05-23 12:19:56.244 | [3m[angrr.storage.memory_mixins.d
default_filler_mixin [0m | [3m[13m[1] setting a value to the initial state [0m
[3mWARNING: [0m 2024-05-23 12:19:56.248 | [3m[angrr.storage.memory_mixins.d
default_filler_mixin [0m | [3m[13m[2] adding the state option ZERO_FILL_UNCONSTRAINE
D (MEMORY_REGISTERS), to make unknown regions hold null [0m
[3mWARNING: [0m 2024-05-23 12:19:56.256 | [3m[angrr.storage.memory_mixins.d
default_filler_mixin [0m | [3m[13m[3] adding the state option SYMBOL_FILL_UNCONSTRAI
NED (MEMORY_REGISTERS), to suppress these messages. [0m
[3mWARNING: [0m 2024-05-23 12:19:56.263 | [3m[angrr.storage.memory_mixins.d
default_filler_mixin [0m | [3m[13mfilling register edb with unconstrained bytes r
eferred from 0x80484e21 - libc.rsu_init+0x1 in issue (0x80485211) [0m
[3mWARNING: [0m 2024-05-23 12:19:56.271 | [3m[angrr.storage.memory_mixins.d
default_filler_mixin [0m | [3m[13mfilling register ebx with 4 unconstrained bytes r
eferred from 0x80484e23 - libc.rsu_init+0x3 in issue (0x80484e23) [0m
51, 57, 61, 240, 60, 75, 139, 78, 197, 23, 142, 90, 154, 99, 212, 163, 102,
108, 166, 172, 105, 169, 114, 120, 63, 178, 184, 71, 135, 77, 83, 202, 89, 147, 86
, 153, 92, 150, 156, 106, 101, 141, 165, 43, 113, 232, 226, 177, 116, 46, 180, 45,
58, 198, 15, 201, 195, 85, 204, 30, 149, 210, 27, 216, 39, 225, 170, 228, 54]
>>>

```

可以看到，程序正确分析出了我们想要的 `u` 值。

5.验证结果是否正确

输入 204，显示”you win! ”,即结果正确！



```
#include <stdio.h>
#include <iostream>
char u = 204;

int main(void)
{
    int i, bits[2] = { 0, 0 };
    for (i = 0; i < 8; i++)
        bits[u & (1 << i)] = 1;
    if (bits[0] == bits[1])
        printf("you win!");
}
```

（三）Angr 求解方法示例二

1.Solve2 代码分析：

```
1. #!/usr/bin/env python
2. # coding=utf-8
3. import angr
4. import claripy
5.
6. def hook_demo(state):
7.     state.regs.eax = 0
8.
9. p = angr.Project("./issue", load_options={"auto_load_libs": False})
10. p.hook(addr=0x08048485, hook=hook_demo, length=2)
11. state = p.factory.blank_state(addr=0x0804846B, add_options={"SYMBOLIC_WRITE_ADDRESSES"})
12. u = claripy.BVS("u", 8)
13. state.memory.store(0x0804A021, u)
14. sm = p.factory.simulation_manager(state)
15. sm.explore(find=0x080484DB)
16. st = sm.found[0]
17. print(repr(st.solver.eval(u)))
```

分析：

主要采用了 hook 函数：addr 为待 hook 的地址，将 0x08048485 处的长度为 2 的指令通过自定义的 hook_demo 进行替代。

```
1. def hook_demo(state):
2.     state.regs.eax = 0
```

- 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。

.text:0804847C	mov	eax, large gs:14h
.text:08048482	mov	[ebp+var_C], eax
.text:08048485	xor	eax, eax
.text:08048487	mov	[ebp+var_14], 0
.text:0804848E	mov	[ebp+var_10], 0
.text:08048495	mov	[ebp+var_18], 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。

2. 执行代码，得出一个值

```
#/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的回调函数, 在执行到addr时, 会执行这个函数, 同时把当前的st
# length 为待hook指令的长度, 在执行完 hook 函数以后, angr 需要根据 len
# 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",
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)))
```

```
Python 3.11.5 (tags/v3.11.5:ccc6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>>
RESTART: C:\Users\ASUS\Desktop\solve.py
[33mWARNING[0m | 2024-05-23 12:43:27.675 | [31mangr.storage.memory_mixins
.default_filler_mixi[0m | [31mThe program is accessing memory with an unspe
cified value. This could indicate unwanted behavior.[0m
[33mWARNING[0m | 2024-05-23 12:43:27.684 | [31mangr.storage.memory_mixins
.default_filler_mixi[0m | [31mangr will cope with this by generating an unc
onstrained symbolic variable and continuing. You can resolve this by [0m
[33mWARNING[0m | 2024-05-23 12:43:27.696 | [31mangr.storage.memory_mixins
.default_filler_mixi[0m | [31m2) adding the state option ZERO_FILL_UNCONSTRA
INED_NEW_VAR_REGISTERS, to make unknown regions hold null[0m
[33mWARNING[0m | 2024-05-23 12:43:27.705 | [31mangr.storage.memory_mixins
.default_filler_mixi[0m | [31m3) adding the state option SYMBOL_FILL_UNCONS
TRAINED_MEMORY_REGISTERS, to suppress these messages.[0m
[33mWARNING[0m | 2024-05-23 12:43:27.712 | [31mangr.storage.memory_mixins
.default_filler_mixi[0m | [31mFilling memory at 0x7ff00000 with 4 unconstra
ined bytes referenced from 0x08048472 (main+0x7 in issue (0x08048472))[0m
[33mWARNING[0m | 2024-05-23 12:43:27.722 | [31mangr.storage.memory_mixins
.default_filler_mixi[0m | [31mFilling register ebp with 4 unconstrained byt
es referenced from 0x08048475 (main+0xa in issue (0x08048475))[0m
[0m
83
>>>
```

3. 结果验证

如图，输入 83 后显示“you win!”，说明结果正确。

```
#include <stdio.h>
#include <iostream>
char u = 83;

int main(void)
{
    int i, bits[2] = { 0, 0 };
    for (i = 0; i < 8; i++) {
        bits[u & (1 << i)] ^= 1;
    }
    if (bits[0] == bits[1]) {
        printf("you win!");
    }
    else {
        printf("you lose!");
    }
    system("pause");
    return 0;
}
```

```
you win!请按任意键继续. . .
```

五、实验分析：

1 第二种方法相比第一种的区别：

(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。

2. 如何使用 angr

在使用 Angr 时，主要包含一下几个步骤：

- 新建一个工程，导入二进制文件，可以通过参数选项来设置是否自动加载依赖项。
- 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容，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()` 可以通过给定参数 `addr` 的值来指定程序起始运行地址

`entry_state()` 可以知名程序在初始运行时的状态

`add_option()` 可以获取一个独立的选项来添加到某个 state 中

- 创建一个符号变量，然后将这个符号变量保存到指定的地址中，这个地址就是二进制文件中 .bss 段中对应的地址。
- 创建一个 Simulation Manager 对象，这个对象和我们提供的状态有关。
- 使用 `explore` 函数对状态进行搜索，检查输出的字符串的结果，进行符号执行的时候，要制定想要得到的状态，级得到满足 `correct` 的条件并且不满足 `wrong` 条件的 state。
- 获取到 state 后，可以通过 solver 求解器，求解符号的值。

3. 如何解决实际问题

在实际应用中，`angr` 是一个强大的二进制分析框架，可帮助我们找到并利用二进制文件中的漏洞。通过组合动态符号执行、静态分析和约束求解等技术，我们可以更有效地分析复杂的二进制程序并解决实际问题。`Angr` 可以为我们提供一种很好的代码分析手段，能够辅助我们对于代码中复杂的分支跳转条件进行解析。在实际应用的过程中，可以结合静态逆向分析工具 `IDA pro` 等来确定想要到达的路径和需要规避的路径，

确定这些条件之后，再使用 **Angr** 进行符号执行，获取到达路径所需要的条件。另外对于一些复杂的逻辑推理的过程，也可尝试使用 **Angr** 来辅助求解。

六、心得体会：

1. **angr** 是一个功能全面的二进制分析框架，它集成了多种分析技术，包括符号执行、静态分析、控制流图分析和数据流分析等。这些功能的结合，使得 **angr** 可以深入地分析复杂的二进制程序，找到程序中的潜在漏洞和逻辑缺陷。
2. 符号执行的优势：符号执行是 **angr** 的核心功能之一。通过符号执行，可以模拟程序的执行过程，而不需要实际运行程序。这对于查找程序中的漏洞、理解复杂程序行为非常有用。符号执行可以覆盖到手工分析无法触及的路径，发现隐藏的安全问题。
3. 动静结合的分析方法：在使用 **angr** 进行分析时，结合静态分析工具（如 **IDA Pro**）可以更好地理解程序的结构和逻辑。静态分析工具可以帮助我们确定关键路径和代码块，而 **angr** 的动态符号执行可以验证这些路径并进一步分析其行为。这种动静结合的方法可以提高分析的准确性和效率。
4. 这次实践使我对二进制分析有了更深刻的理解，也为将来解决实际问题提供了宝贵的经验。