

# 《软件安全》实验报告

姓名：陆皓喆 学号：2211044 班级：信息安全

## 实验名称：

*Angr*应用实例

## 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 安装Python3和Angr

在我们的电脑中，本来就含有`python3`，我们只需要在终端输入 `pip install angr` 就可以安装`angr`了。安装完成后，我们就可以开始我们的实验了。

```
Terminal 本地 x + v
chertools, bitstring, archinfo, ailment, rpyc, pyformlang, nampa, gitdb, claripy, cffi, pyvex, GitPython, cle, angr
Successfully installed CppHeaderParser-2.7.4 GitPython-3.1.43 ailment-9.2.100 angr-9.2.100 archinfo-9.2.100 bitarray-2.9.2 bitstring-4.2.1 cacheto
ols-5.3.3 capstone-5.0.0.post1 cffi-1.16.0 claripy-9.2.100 cle-9.2.100 colorama-0.4.6 decorator-5.1.1 dpkt-1.9.8 future-1.0.0 gitdb-4.0.11 itanium
-demangler-1.1 mpmath-1.3.0 mulpyplexer-0.9 nampa-0.1.1 pefile-2023.2.7 plumbum-1.8.2 ply-3.11 psutil-5.9.8 pycparser-2.22 pydot-2.0.0 pyelftools-
0.31 pyformlang-1.0.10 pysmt-0.9.5 pyvex-9.2.100 pywin32-306 rpyc-6.0.0 smmap-5.0.1 sortedcontainers-2.4.0 sympy-1.12 unicorn-2.0.1.post1 unique-l
og-filter-0.1.0 z3-solver-4.10.2.0

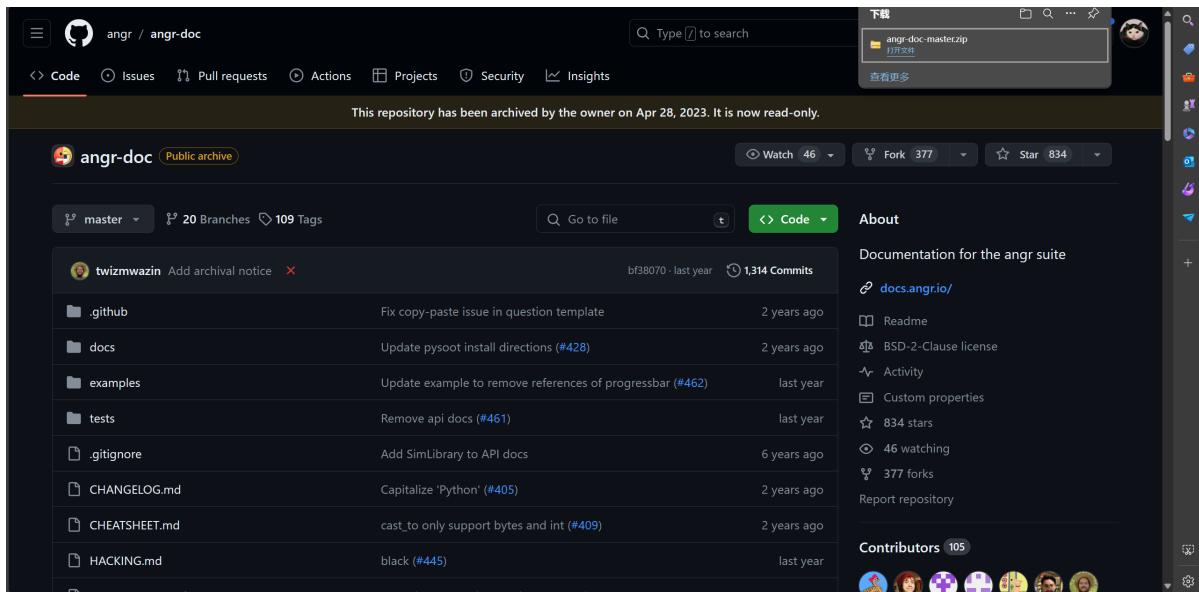
[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
(venv) PS D:\Python_learning>
```

为了确保我们成功的安装了`angr`，我们进行测试，在代码区输入 `import angr`，发现没有报错，说明我们已经成功地将库导入到了`python`文件中。

```
Project v
Python_learning D:\Python_learning
  venv library root
    lab9.py
  External Libraries

Run lab9 x
D:\Python_learning\venv\Scripts\python.exe D:\Python_learning\lab9.py
```

然后，我们下载`angr`的官方文档来获得实验所需要的样例。我们进入提供的网址<https://github.com/angr/angr-doc>



我们点击`download`，下载完压缩包再完成解压，获得以下文件：

名称	修改日期	类型	大小
今天			
.gitignore	2024/4/26 23:44	Git Ignore 源文件	1 KB
angr-papers	2024/4/26 23:44	BibTeX 源文件	15 KB
book	2024/4/26 23:44	JSON File	1 KB
CHANGELOG	2024/4/26 23:44	Markdown File	35 KB
CHEATSHEET	2024/4/26 23:44	Markdown File	6 KB
HACKING	2024/4/26 23:44	Markdown File	8 KB
HELPWANTED	2024/4/26 23:44	Markdown File	11 KB
INSTALL	2024/4/26 23:44	Markdown File	10 KB
LICENSE	2024/4/26 23:44	文件	2 KB
MIGRATION	2024/4/26 23:44	Markdown File	2 KB
README	2024/4/26 23:44	Markdown File	5 KB
SUMMARY	2024/4/26 23:44	Markdown File	2 KB
examples	2024/4/26 23:44	文件夹	
tests	2024/4/26 23:44	文件夹	
.github	2024/4/26 23:44	文件夹	
docs	2024/4/26 23:44	文件夹	

## 2 复现sym-write的两种方法

### 2.1 求解方法1

issue.c 源码：

```
#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;
}

```

solve.py 源码:

```

import angr
import claripy

def main():
    p = angr.Project('./issue', load_options={"auto_load_libs": False})

    # By default, all symbolic write indices are concretized.
    state = p.factory.entry_state(
        add_options={angr.options.SYMBOLIC_WRITE_ADDRESSES})

    u = claripy.BVS("u", 8)
    state.memory.store(0x804a021, u)

    sm = p.factory.simulation_manager(state)

    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

    sm.explore(find=correct, avoid=wrong)

    # Alternatively, you can hardcode the addresses.
    # sm.explore(find=0x80484e3, avoid=0x80484f5)

    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)

```

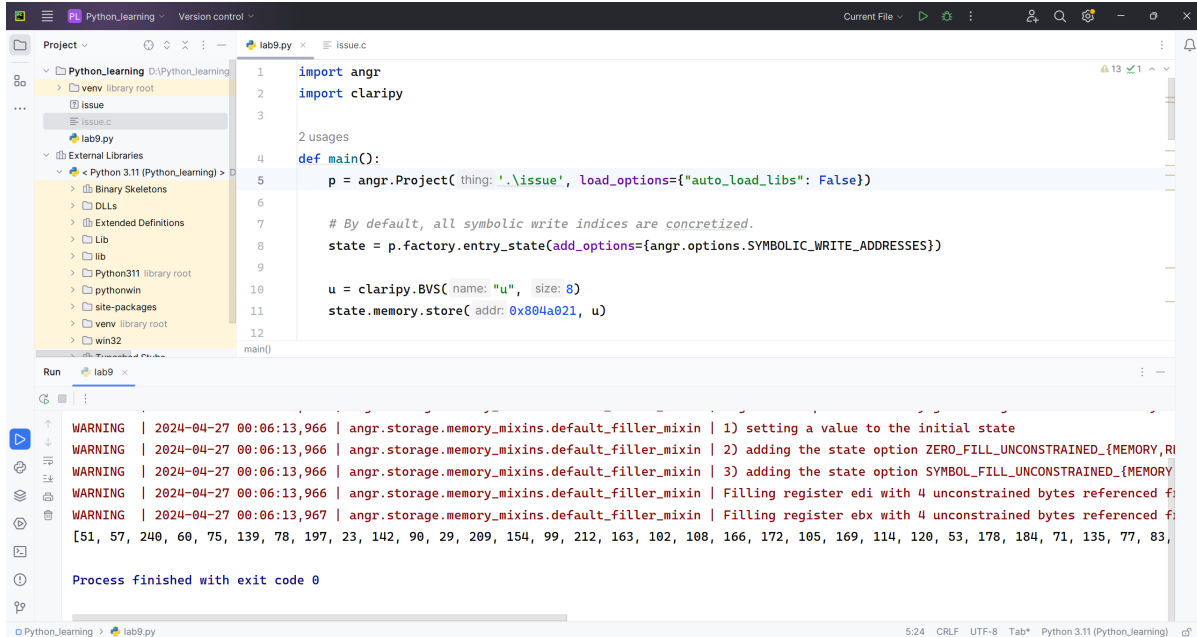
```

res = main()
assert set(res) == good

if __name__ == '__main__':
    print(repr(main()))

```

我们将其导入到`pycharm`中运行，得到以下的结果：



可以发现我们得到了`u`的所有结果：

```

[51, 57, 240, 60, 75, 139, 78, 197, 23, 142, 90, 29, 209, 154, 99, 212, 163, 102, 108, 166, 172, 105, 169, 114, 120, 53, 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]

```

以上的每一个解我们都可以带回到源程序中进行验证。

## 对于第一种解法的分析：

主要步骤是：

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（`SimulationManagers`）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。
6. 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。
7. 获得到 `state` 之后，通过 `solver` 求解器，求解 `u` 的值。
8. 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。

## 2.2 求解方法2

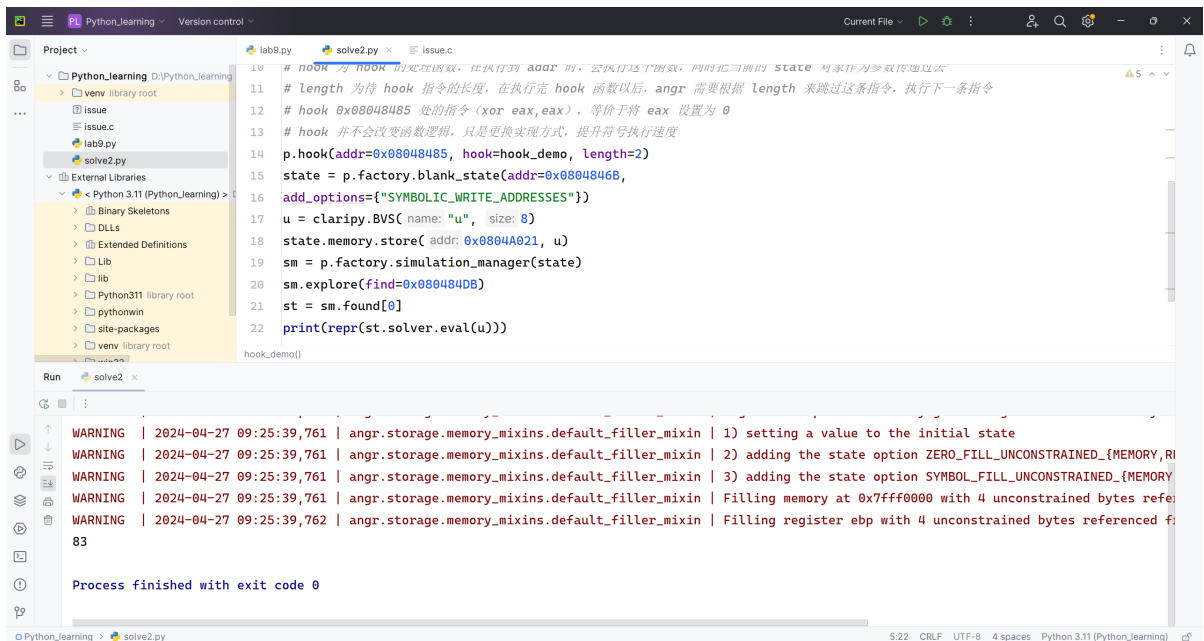
下面我们来实现第二种方法的求解。

`solve2.py` 源码如下：

```
#!/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)))
```

我们使用该代码进行运行，获得以下的结果：



我们由上面的结果可以得到：跟解法一不同，我们使用该解法只获得了一个解83，相比于解法一的很多个解略有差异。

### 对于第二种解法的分析：

第二种解法与前一种解法的区别在于以下三处。

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。所以上面的第二种解法中，只输出了一个结果，而不是像第一个解法那样输入一大堆的解。

## 3 Angr在实际问题上的应用

### 3.1 如何使用angr库

我们如何在实际问题中使用该 angr 库呢？

Angr 是一个用于符号执行的 *Python* 框架，它可以用于自动化地分析二进制文件并生成输入，以达到某些预期的目标。下面就简单介绍一下 angr 库的应用方式。

Angr 是一个强大的二进制分析工具，用于静态和动态分析。它可以用于诸如反向工程、漏洞发现、恶意代码分析等任务。以下是使用 Angr 的一般步骤：

1. 安装Angr：可以通过 pip 安装 Angr。在命令行中运行以下命令来安装：

```
pip install angr
```

2. 导入Angr：在 *Python* 脚本中导入 Angr 库：

```
import angr
```

3. 创建二进制文件的项目：使用 `angr.Project()` 函数来创建一个 Angr 项目，将二进制文件加载到该项目中：

```
project = angr.Project("/path/to/binary")
```

4. 设置分析参数：根据需要设置 Angr 分析的参数。例如，可以设置初始状态、约束条件、路径搜索策略等。
5. 执行分析：执行所需的分析任务。Angr 提供了各种分析功能，如符号执行、符号执行路径搜索、符号化执行等。以下是一些常见的分析任务：
6. 符号执行：通过 `project.factory.entry_state()` 创建初始状态，并使用 `project.factory.simulation_manager()` 创建模拟器进行符号执行。

```
entry_state = project.factory.entry_state()
simgr = project.factory.simulation_manager(entry_state)
simgr.run()
```

7. 路径搜索：在符号执行的基础上，使用路径搜索策略来探索程序的不同执行路径。

```
simgr.explore()
```

8. 约束求解：对于符号执行的结果，可以使用约束求解器来求解符号变量的具体取值。

```
for found in simgr.found:
    print(found.solver.eval(input_var))
```

9. 分析结果解释：根据分析的结果，解释和理解程序的行为、漏洞或其他关键信息。

## 3.2 用angr库解决实际问题

Angr 可以用于解决各种实际问题，包括但不限于以下几种情况：

1. 漏洞挖掘：Angr 可以用于在二进制程序中发现漏洞，如缓冲区溢出、整数溢出等。通过符号执行和路径搜索，Angr 能够探索程序的不同执行路径，并识别可能存在漏洞的代码路径。
2. 逆向工程：Angr 可用于逆向工程任务，如反编译、恶意代码分析等。通过分析程序的控制流和数据流，Angr 可以帮助理解程序的逻辑结构和算法。
3. 加密算法分析：Angr 可以用于分析加密算法的实现，识别潜在的弱点或漏洞。通过符号执行和约束求解，Angr 可以分析加密算法的安全性，并找到可能的攻击路径。
4. CTF竞赛：Angr 在 CTF 竞赛中被广泛应用，用于解决各种二进制逆向和加密相关的挑战。使用 Angr 可以自动化地分析和解决复杂的 CTF 问题，节省时间和精力。
5. 漏洞利用：Angr 可以用于漏洞利用和漏洞利用开发。通过符号执行和路径搜索，Angr 可以找到程序中存在的漏洞，并生成利用该漏洞的有效输入数据。
6. Fuzzing测试：Angr 可用于生成和分析模糊测试的输入。通过符号执行和路径搜索，Angr 可以帮助发现程序中的潜在漏洞和错误。
7. 固件分析：Angr 可以用于分析嵌入式设备的固件，识别其中的漏洞或后门。通过符号执行和路径搜索，Angr 可以帮助理解固件的工作原理和逻辑。

# 心得体会：

---

通过本次实验，我学习了一个强大的`python`库的使用方法，了解了其基本原理，也知道了 `angr` 库可以解决很多实际的问题，高效又准确。而且在 CTF 竞赛中，我也可以使用该库来进行 `pwn` 的逆向分析等，还可以用来进行漏洞利用与开发。通过本次的实验，我对`python`库的作用又有了进一步的深入了解，希望在后续的学习中，还能学到更多的库的知识。