

《软件安全》实验报告

姓名：刘星宇 学号：2212824 班级：信息安全法学双学位班

实验名称：

Angr 应用示例实验

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. 准备环境，安装 angr

在 angr 官网下载 angr-doc， pip install angr 安装 angr

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

C:\Users\liuxi\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.11>pip install angr
Collecting angr
  Obtaining dependency information for angr from https://files.pythonhosted.org/packages/81/34/a6ca976f47a86e7b1db09248e19a6d8dc2991fe80d3bda05b38533cf97/angr-9.2.103-py3-none-win_amd64.whl.metadata
  Downloading angr-9.2.103-py3-none-win_amd64.whl.metadata (4.8 kB)
Collecting CppHeaderParser (from angr)
  Downloading CppHeaderParser-2.7.4.tar.gz (54 kB)
    54.4/54.4 kB 217.8 kB/s eta 0:00:00
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Collecting GitPython (from angr)
  Obtaining dependency information for GitPython from https://files.pythonhosted.org/packages/e9/bd/cc3a402a6439c15c3e94333e13042b915bbeab54edc457c723931fed3f/GitPython-3.1.43-py3-none-any.whl.metadata
  Downloading GitPython-3.1.43-py3-none-any.whl.metadata (13 kB)
Collecting ailment==9.2.103 (from angr)
  Obtaining dependency information for ailment==9.2.103 from https://files.pythonhosted.org/packages/02/54/522d801646
```

在 angr 的官网下载 angr-doc，获取说明文档和测试用例

2. 实验过程

- 1) 目标分析

观察源代码

Issue.c

```
1  #include <stdio.h>
2
3  char u=0;
4  int main(void)
5  {
6      int i, bits[2]={0,0};
7      for (i=0; i<8; i++) {
8          bits[(u&(1<<i))!=0]++;
9      }
10     if (bits[0]==bits[1]) {
11         printf("you win!");
12     }
13     else {
14         printf("you lose!");
15     }
16     return 0;
17 }
18
```

Solve.py

```
12  ~ import angr
13      import claripy
14
15      2 用法
16  ~ def main():
17      p = angr.Project('./issue', load_options={"auto_load_libs": False})
18
19      # By default, all symbolic write indices are concretized.
20      state = p.factory.entry_state(add_options={angr.options.SYMBOLIC_WRITE_ADDRESSES})
21
22      u = claripy.BVS("u", 8)
23      state.memory.store(0x804a021, u)
24
25      sm = p.factory.simulation_manager(state)
```

2) Angr 分析

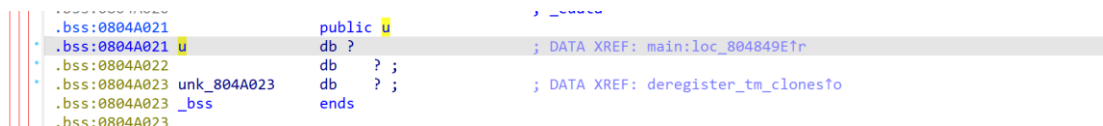
源码 solve.py 有以下作用：

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。

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。看
- 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。

在 `state.memory.store(0x804a021, u)` 这句中，是把符号变量保存在指定地址中，这个地址就是二进制文件中 .bss 段 `u` 的地址。可以通过 IDA_PRO 去寻找在源代码中 `u` 对应的位置，如下图



在 .bss 段中可以找到 `public u, 0804A021`，也就是 `u` 这个全局变量的地址在这之后，我们可以对整个文件进行符号执行，以找到符号变量 `u` 的位置。

3) 运行结果

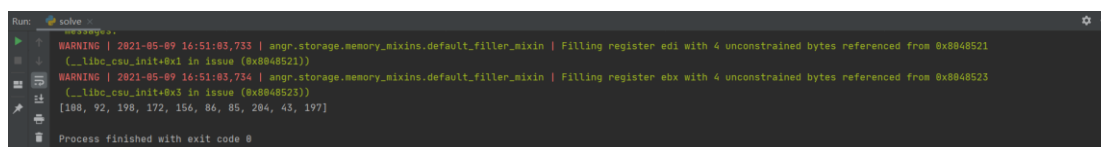
定义了两个搜寻结果函数，以检查输出字符串是 `win` 还是 `lose`：并使用 `sm.explore` 函数来搜索状态想要的状态，即得到满足 `correct` 条件且不满足 `wrong` 条件的 `state`；也可以写成通过地址进行定位。

```
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)
```

运行结果如下：



得到了 10 个求出来的 `u` 的值，将其带回到源程序去验证

 Microsoft Visual Studio 调试控制台

you win!

成功输出了 `you win`，实验成功。

另一种解法：

```
#another_solution
# 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)))

```

在其中,

```
p.hook(addr=0x08048485, hook=hook_demo, length=2)
```

这句中的 08048485 为 xor eax,eax, 等价于将 eax 设置为 0。

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。

心得体会:

在本次实验中, 我深入学习了 angr 这一强大的符号执行框架。起初, 由于英文文档的复杂性和对诸如 hook 等技术名词的不熟悉, 我面临了诸多挑战。然而, 随着学习的深入, 我逐渐克服了这些困难, 对 angr 有了更为全面的理解。

我认识到，**angr** 不仅仅是一个简单的工具，而是一个集成了多种先进技术的框架。它利用 **hook** 和符号执行等机制，为我们提供了一种全新的解决复杂计算问题的方式。这种技术让我深感震撼，同时也让我对能够提出并设计出这些解决方案的专家们充满了敬意。在实验中，我亲身体验了 **angr** 在自动化漏洞挖掘、软件安全分析以及生成测试用例等方面的应用。这些实际操作不仅加深了我对 **angr** 功能的理解，也让我看到了它在安全研究和软件开发领域的巨大潜力。

总的来说，这次 **angr** 的学习实验让我受益匪浅。我不仅掌握了 **angr** 的基本使用方法和技巧，还对其背后的原理和机制有了更深入的了解。我相信，在未来的学习和工作中，我会继续深入探索 **angr** 的更多功能和应用，为软件安全领域做出更大的贡献。