

软件安全实验报告

姓名：辛浩然 学号：2112514 班级：信息安全、法学

1 实验名称

Angr 应用示例

2 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。

3 实验过程

3.1 Angr 安装

Windows 下安装 Angr。首先安装 Python3。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，说明安装成功。

```
PS C:\Users\15990> python
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import angr
>>> |
```

3.2 sym-write 示例

下载 GitHub 上 angr 的开源项目。本次使用的例子是 sys_write。

源码 issue.c 如下：

```
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 }

```

该代码首先检查 `u` 的二进制数的第 `i` 位是否为 1。如果是，则将数组 `bits` 的第二个元素（即 1 的数量）加 1，否则将数组 `bits` 的第一个元素（即 0 的数量）加 1。如果 `u` 的二进制数中 0 的数量等于 1 的数量，则输出 “you win!”；否则输出 “you lose!”。

3.2.1 求解方法一复现

源码 `solve.py` 如下：

```

1  import angr
2  import claripy
3  def main():
4      # 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
5      p = angr.Project('./issue', load_options={"auto_load_libs": False})
6
7      # 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
8      # blank_state(): 可通过给定参数 addr 的值指定程序起始运行地址
9      # entry_state(): 指明程序在初始运行时的状态，默认从入口点执行
10     # add_options 获取一个独立的选项来添加到某个 state 中
11     # SYMBOLIC_WRITE_ADDRESSES: 允许通过具体化策略处理符号地址的写操作
12     state = p.factory.entry_state(add_options={angr.options.
        SYMBOLIC_WRITE_ADDRESSES})
13
14     # 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
15     u = claripy.BVS("u", 8)
16     # 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
17     state.memory.store(0x804a021, u)
18
19     # 4. 创建一个 Simulation Manager 对象，这个对象和我们的状态有关系
20     sm = p.factory.simulation_manager(state)
21
22     # 5. 使用 explore 函数进行状态搜寻，检查输出字符串是 win 还是 lose
23     # state.posix.dumps(1) 获得所有标准输出
24     # state.posix.dumps(0) 获得所有标准输入
25     def correct(state):

```

```

26     try:
27         return b'win' in state.posix.dumps(1)
28     except:
29         return False
30 def wrong(state):
31     try:
32         return b'lose' in state.posix.dumps(1)
33     except:
34         return False
35 # 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
36 sm.explore(find=correct, avoid=wrong)
37 # 获得到 state 之后，通过 solver 求解器，求解 u 的值
38 # 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
39 return sm.found[0].solver.eval_upto(u, 256)
40 if __name__ == '__main__':
41     # repr()函数将 object 对象转化为 string 类型
42     print(repr(main()))

```

代码分析

上述代码的关键步骤如下：

1. 新建一个 Angr 工程，并且载入二进制文件。auto_load_libs 设置为 false，不会自动载入依赖的库。
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。
3. 将要求解的变量符号化，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。
6. 获得到 state 之后，通过 solver 求解器，求解 u 的值。

实验验证

选择填写的 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 model，界面如下：

```
IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\nku\23spring\软件安全\实验\9-Angr应用示例\angr-doc-master\angr-doc-master\examples\sym-write\solve.py
[33mWARNING[0m | 2023-04-25 16:35:59.638 | [31mangr.storage.memory_mixins
.default_filler_mixin[0m | [31mThe program is accessing register with an uns
pecified value. This could indicate unwanted behavior.[0m
[33mWARNING[0m | 2023-04-25 16:35:59.687 | [31mangr.storage.memory_mixins
.default_filler_mixin[0m | [31mangr will cope with this by generating an unc
onstrained symbolic variable and continuing. You can resolve this by:[0m
[33mWARNING[0m | 2023-04-25 16:35:59.711 | [31mangr.storage.memory_mixins
.default_filler_mixin[0m | [31m1) setting a value to the initial state[0m
[33mWARNING[0m | 2023-04-25 16:35:59.732 | [31mangr.storage.memory_mixins
.default_filler_mixin[0m | [31m2) adding the state option ZERO_FILL_UNCONSTRA
INED (MEMORY.REGISTERS), to make unknown regions hold null[0m
[33mWARNING[0m | 2023-04-25 16:35:59.758 | [31mangr.storage.memory_mixins
.default_filler_mixin[0m | [31m3) adding the state option SYMBOL_FILL_UNCONST
RAINED (MEMORY.REGISTERS), to suppress these messages.[0m
[33mWARNING[0m | 2023-04-25 16:35:59.778 | [31mangr.storage.memory_mixins
.default_filler_mixin[0m | [31mFilling register edi with 4 unconstrained byt
es referenced from 0x8048521 (__libc_csu_init+0x1 in issue (0x8048521))[0m
[33mWARNING[0m | 2023-04-25 16:35:59.800 | [31mangr.storage.memory_mixins
.default_filler_mixin[0m | [31mFilling register ebx with 4 unconstrained byt
es referenced from 0x8048523 (__libc_csu_init+0x3 in issue (0x8048523))[0m
[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, 18
0, 45, 58, 198, 15, 201, 195, 85, 204, 30, 149, 210, 27, 216, 39, 225, 170, 228,
54]
>>>
```

蓝色部分，就是输出的 `u` 的求解的结果，因为我们采用了 `eval_upto` 函数，给出了多个解。对其中一个解，带入源程序进行验证，验证了其正确性：

```
PS D:\nku\23spring\软件安全\实验\9-Angr应用示例\angr-doc-master\angr-doc-master\examples\sym-write\solve.py
51
you win!
```

3.2.2 求解方法二复现

源码如下：

```
1 import angr
2 import claripy
3 def hook_demo(state):
4     state.regs.eax = 0
5
6 p = angr.Project("./issue", load_options={"auto_load_libs": False})
7 # hook 函数: addr 为待 hook 的地址
8 # 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 对象作参数传递过去
9 # 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
10 # hook 0x08048485 处的指令 (xor eax,eax)，等价于将 eax 设置为 0
11 # 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
12 p.hook(addr=0x08048485, hook=hook_demo, length=2)
13 state = p.factory.blank_state(addr=0x0804846B,
```

```

14     add_options={"SYMBOLIC_WRITE_ADDRESSES"})
15     u = claripy.BVS("u", 8)
16     state.memory.store(0x0804A021, u)
17     sm = p.factory.simulation_manager(state)
18     sm.explore(find=0x080484DB)
19     st = sm.found[0]
20     print(repr(st.solver.eval(u)))

```

代码分析

上述代码与方法一不同的地方在于：

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。
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. eval(u) 替代了原来的 eval_upto，将打印一个结果出来。

实验验证

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 model，界面如下：

蓝色部分，就是输出的 u 的求解的结果，给出了一个解。

对其中一个解，带入源程序进行验证，验证了其正确性：

```
PS D:\nku\23spring\软件安全\实验\实验\9-Angr应用示例\angr-doc-master>
}
83
you win!
```

3.3 总结

3.3.1 angr 的使用步骤

1. 导入 angr 模块，并使用 `angr.Project()` 函数加载目标二进制文件。
2. 定义程序的入口点。可以使用 `proj.entry` 属性指定程序入口点的地址，或者使用 `proj.factory.entry_state()` 函数创建一个初始状态来作为入口点。
3. 创建一个路径探索器对象 (path explorer) 来探索程序的不同执行路径。可以使用 `proj.factory.path_group()` 函数来创建一个路径探索器。
4. 设置路径探索器的探索策略和约束条件。可以使用 `path_group.explore()` 方法来指定探索策略和约束条件，如深度优先或广度优先搜索，或者添加符号约束或限制条件等。
5. 对探索器进行迭代，探索不同的执行路径。可以使用 `path_group.step()` 方法来迭代探索器，直到探索完所有的路径或者达到某个停止条件。
6. 根据探索结果，分析程序的行为和漏洞。可以使用 angr 提供的一系列分析工具和函数，如路径约束求解、符号执行、污点分析、模拟执行等，来分析程序的行为和漏洞。
7. 输出分析结果。

3.3.2 实际问题解决

可以利用 angr 解决以下实际问题：

1. 二进制文件逆向工程：使用 Angr 可以对二进制文件进行逆向工程，查找其中的漏洞、函数、数据等。
2. 自动化测试：使用 Angr 可以自动化测试软件，找出其中的缺陷和漏洞。可以生成随机输入数据，然后对软件进行符号执行，找出导致崩溃或异常的输入。
3. 符号执行：使用 Angr 可以执行符号执行，即通过符号表达式来执行代码，从而找出所有可能的路径和输入。可以使用 Angr 来检测缓冲区溢出、代码注入等漏洞。

4 心得体会

通过本次实验，复现了 sym-write 示例的两种 angr 求解方法，深入实践、理解了如何利用 angr 以及如何解决一些实际问题。