

《漏洞利用及渗透测试基础》实验报告

姓名：齐明杰 学号：2113997 班级：信安2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：

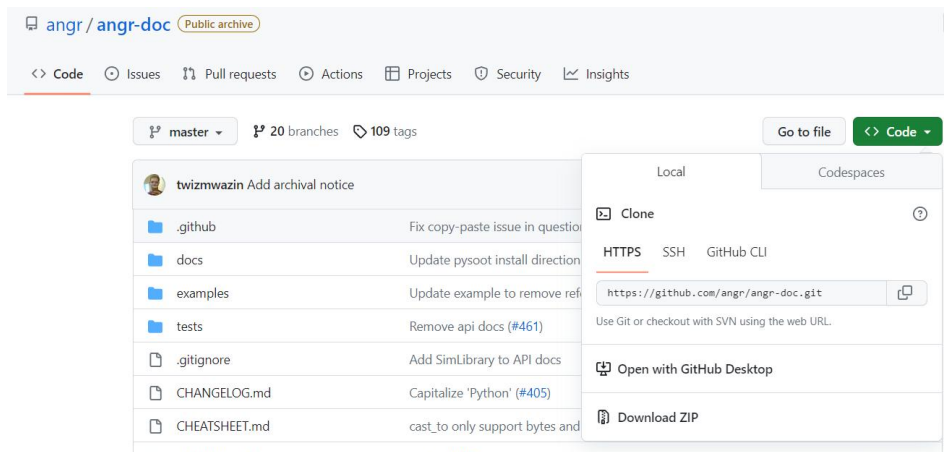
一、安装 Angr

使用命令：pip install angr，即可安装 angr 包(已经安装 python3 环境)

```
管理员: 命令提示符 - pip install angr
Microsoft Windows [版本 10.0.19045.2006]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\Administrator>pip install angr
Collecting angr
  Using cached angr-9.2.48-py3-none-win_amd64.whl (8.1 MB)
Requirement already satisfied: CppHeaderParser in d:\programdata\anaconda3\lib\site-packages (from angr) (2.7.4)
Requirement already satisfied: GitPython in d:\programdata\anaconda3\lib\site-packages (from angr) (3.1.31)
Requirement already satisfied: ailment==9.2.48 in d:\programdata\anaconda3\lib\site-packages (from angr) (9.2.48)
Requirement already satisfied: archinfo==9.2.48 in d:\programdata\anaconda3\lib\site-packages (from angr) (9.2.48)
Requirement already satisfied: cachetools in d:\programdata\anaconda3\lib\site-packages (from angr) (5.3.0)
Requirement already satisfied: capstone==5.0.0rc2, >=3.0.5rc2 in d:\programdata\anaconda3\lib\site-packages (from angr) (5.0.0rc2)
Requirement already satisfied: cffi==1.14.0 in d:\programdata\anaconda3\lib\site-packages (from angr) (1.15.1)
Requirement already satisfied: claripy==9.2.48 in d:\programdata\anaconda3\lib\site-packages (from angr) (9.2.48)
Requirement already satisfied: cle==9.2.48 in d:\programdata\anaconda3\lib\site-packages (from angr) (9.2.48)
Requirement already satisfied: dpkt in d:\programdata\anaconda3\lib\site-packages (from angr) (1.9.8)
Requirement already satisfied: itanium-demangler in d:\programdata\anaconda3\lib\site-packages (from angr) (1.1)
Requirement already satisfied: multiplexer in d:\programdata\anaconda3\lib\site-packages (from angr) (0.9)
Requirement already satisfied: nmap in d:\programdata\anaconda3\lib\site-packages (from angr) (0.1.1)
Requirement already satisfied: networkx==2.8.1, >=2.0 in d:\programdata\anaconda3\lib\site-packages (from angr) (2.8.4)
Requirement already satisfied: protobuf==3.19.0 in d:\programdata\anaconda3\lib\site-packages (from angr) (4.22.3)
Requirement already satisfied: psutil in d:\programdata\anaconda3\lib\site-packages (from angr) (6.9.0)
Requirement already satisfied: pycparser==2.18 in d:\programdata\anaconda3\lib\site-packages (from angr) (2.21)
Requirement already satisfied: pyvex==9.2.48 in d:\programdata\anaconda3\lib\site-packages (from angr) (9.2.48)
```

在 github 上下载 angr 的官方文档：



二、Angr 示例

在 angr-doc 里有各类 Example，展示了 Angr 的用法。我们以 sym-write 为例子，源码 issue.c 如下图所示：

```
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
```

源码 solve.py 如下:

```
1. #!/usr/bin/env python3
2. # -*- coding: utf-8 -*-
3.
4. """
5. Author: xoreaxeaxeax
6. Modified by David Manouchehri <manouchehri@protonmail.com>
7. Original at https://lists.cs.ucsb.edu/pipermail/angr/2016-August/000167.html
8.
9. The purpose of this example is to show how to use symbolic write addresses.
10. """
11.
12. import angr
13. import claripy
14.
15. def main():
16.     p = angr.Project('./issue', load_options={"auto_load_libs": False})
17.
18.     # By default, all symbolic write indices are concretized.
19.     state = p.factory.entry_state(add_options={angr.options.SYMBOLIC_WRITE_ADDRESSES})
20.
21.     u = claripy.BVS("u", 8)
22.     state.memory.store(0x804a021, u)
23.
24.     sm = p.factory.simulation_manager(state)
25.
26.     def correct(state):
27.         try:
28.             return b'win' in state.posix.dumps(1)
29.         except:
```

```

30.         return False
31.     def wrong(state):
32.         try:
33.             return b'lose' in state.posix.dumps(1)
34.         except:
35.             return False
36.
37.     sm.explore(find=correct, avoid=wrong)
38.
39.     # Alternatively, you can hardcode the addresses.
40.     # sm.explore(find=0x80484e3, avoid=0x80484f5)
41.
42.     return sm.found[0].solver.eval_upto(u, 256)
43.
44.
45. def test():
46.     good = set()
47.     for u in range(256):
48.         bits = [0, 0]
49.         for i in range(8):
50.             bits[u&(1<<i)!=0] += 1
51.         if bits[0] == bits[1]:
52.             good.add(u)
53.
54.     res = main()
55.     assert set(res) == good
56.
57. if __name__ == '__main__':
58.     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。

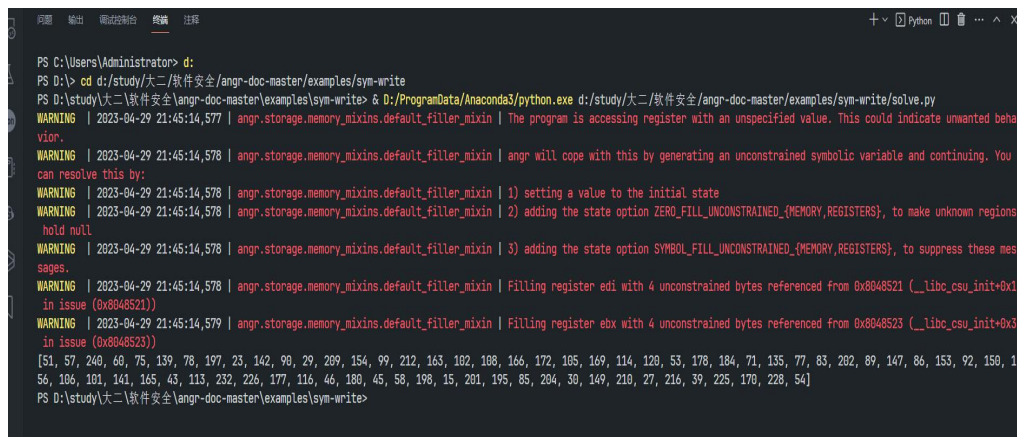
（4）创建模拟管理器（Simulation Managers）进行程序执行管理。初始化的 state 可以经过模拟执行得到一系列的 states，模拟管理器 sm 的作用就是对这些 states 进行管理。

（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。

运行上述代码，得到如下结果：

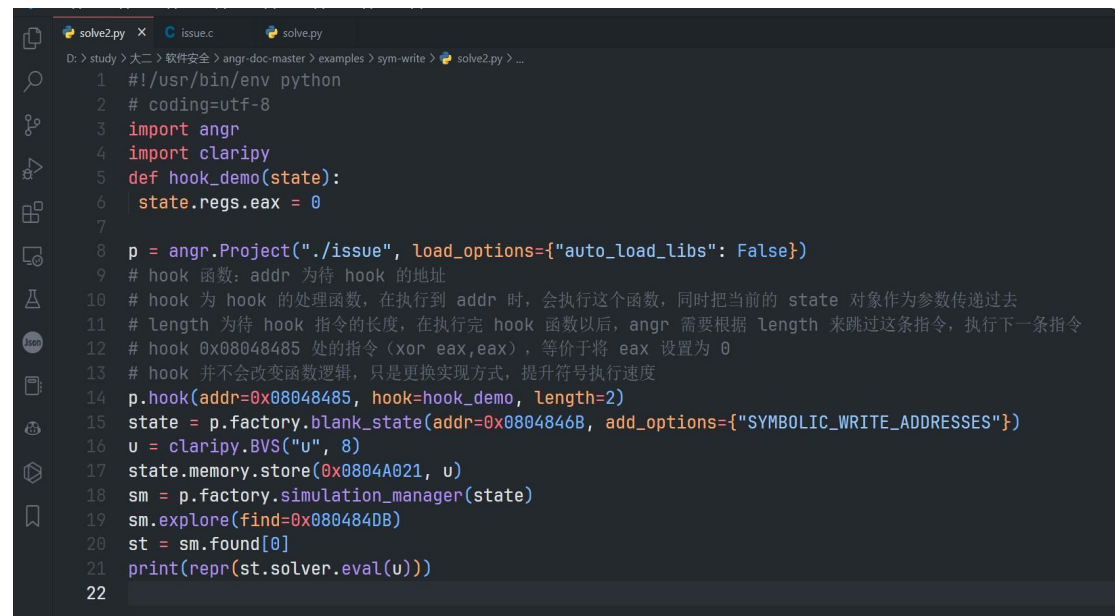


```
PS C:\Users\Administrator> d:
PS D:\> cd d:/study/大二/软件安全/angr-doc-master/examples/sym-write
PS D:\study\大二\软件安全\angr-doc-master\examples\sym-write> & D:/ProgramData/Anaconda3/python.exe d:/study/大二/软件安全/angr-doc-master/examples/sym-write/solve.py
WARNING | 2023-04-29 21:45:14,577 | angr.storage.memory_mixin.default_filler_mixin | The program is accessing register with an unspecified value. This could indicate unwanted behavior.
WARNING | 2023-04-29 21:45:14,578 | angr.storage.memory_mixin.default_filler_mixin | angr will cope with this by generating an unconstrained symbolic variable and continuing. You can resolve this by:
WARNING | 2023-04-29 21:45:14,578 | angr.storage.memory_mixin.default_filler_mixin | 1) setting a value to the initial state
WARNING | 2023-04-29 21:45:14,578 | angr.storage.memory_mixin.default_filler_mixin | 2) adding the state option ZERO_FILL_UNCONSTRAINED_MEMORY_REGISTERS, to make unknown regions hold null
WARNING | 2023-04-29 21:45:14,578 | angr.storage.memory_mixin.default_filler_mixin | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_MEMORY_REGISTERS, to suppress these messages.
WARNING | 2023-04-29 21:45:14,579 | angr.storage.memory_mixin.default_filler_mixin | Filling register edi with 4 unconstrained bytes referenced from 0x8048521 (__libc_csu_init+0x1 in issue (0x8048521))
WARNING | 2023-04-29 21:45:14,579 | angr.storage.memory_mixin.default_filler_mixin | Filling register ebx with 4 unconstrained bytes referenced from 0x8048523 (__libc_csu_init+0x3 in issue (0x8048523))
[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, 69, 147, 86, 153, 92, 150, 156, 186, 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, 178, 228, 54]
PS D:\study\大二\软件安全\angr-doc-master\examples\sym-write>
```

可见，底下列表部分，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。

三、Angr 第二种解法

使用如下代码(`solve2.py`):



```
1 #!/usr/bin/env python
2 # coding=utf-8
3 import angr
4 import claripy
5 def hook_demo(state):
6     state.regs.eax = 0
7
8 p = angr.Project("./issue", load_options={"auto_load_libs": False})
9 # hook 函数: addr 为 hook 的地址
10 # 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
11 # length 为 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
12 # hook 0x08048485 处的指令 (xor eax,eax)，等价于将 eax 设置为 0
13 # 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
14 p.hook(addr=0x08048485, hook=hook_demo, length=2)
15 state = p.factory.blank_state(addr=0x0804846B, add_options={"SYMBOLIC_WRITE_ADDRESSES"})
16 u = claripy.BVS("u", 8)
17 state.memory.store(0x0804A021, u)
18 sm = p.factory.simulation_manager(state)
19 sm.explore(find=0x080484DB)
20 st = sm.found[0]
21 print(repr(st.solver.eval(u)))
22
```

其中，`0x0804A021` 在 IDA 反汇编视图中如下：

如何使用 angr 以及解决实际问题：

- 1、安装 angr：在开始之前，我们需要确保已经在系统中安装了 angr。
- 2、加载二进制文件：angr 的第一步是加载目标二进制文件。我们需要创建一个 Project 对象，让 angr 分析二进制文件并收集有关该文件的重要信息。
- 3、符号执行：angr 的主要功能之一是符号执行。符号执行允许我们在不实际执行程序的情况下，模拟程序执行的过程。这对于查找程序中的漏洞和理解程序行为非常有用。在 angr 中，我们可以使用 SimulationManager 对象进行符号执行。
- 4、约束求解：angr 使用内置的约束求解器（名为 claripy）来处理符号变量。这使得我们可以生成满足特定条件的输入数据。例如，通过为符号变量添加约束，我们可以找到导致程序崩溃或触发特定漏洞的输入。
- 5、检测漏洞：angr 可以帮助我们自动检测二进制文件中的常见漏洞，如缓冲区溢出、格式化字符串攻击等。我们可以编写自定义的查找函数来识别程序中的漏洞，然后使用 angr 的 SimulationManager 对象在程序执行过程中查找这些漏洞。
- 6、利用漏洞：一旦找到漏洞，我们可以利用 angr 来生成利用该漏洞的输入。这对于测试系统的安全性或验证修复方案的有效性非常有用。通过分析满足特定条件的程序状态，我们可以获取触发漏洞的输入数据。
- 7、静态分析：除了动态符号执行外，angr 也支持多种静态分析方法，如控制流图（CFG）分析、数据流分析等。这些方法可以帮助我们更好地理解程序的结构和行为。
- 8、组合分析方法：angr 的一个优势是它可以将不同的分析方法结合在一起，以实现更高效的二进制分析。例如，我们可以先使用静态分析找到感兴趣的函数或代码块，然后使用符号执行进一步分析这些部分。

总之，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。

心得体会：

通过本次实验，我学会了如何在 angr 分析框架中处理符号化的内存写地址，并更加了解了符号执行在二进制分析中的应用。此外，我认识到了 angr 的强大功能，以及如何结合 Claripy 进行约束求解。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。