

《漏洞利用及渗透测试基础》实验报告

姓名：王晶 学号：2310420 班级：信息安全法学

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) 安装过程

首先激活我们的 python 环境 pytorch_py39，在该环境下安装 angr 包，使用命令 `pip install pytorch_py39`

```
D:\anoinstall\envs\pytorch_py39>conda activate pytorch_py39

(pytorch_py39) D:\anoinstall\envs\pytorch_py39>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)
  Preparing metadata (setup.py) ... done
Collecting GitPython (from angr)
  Downloading GitPython-3.1.44-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-5.5.2-py3-none-any.whl.metadata (5.4 kB)
Collecting capstone==5.0.0.post1 (from angr)
  Downloading capstone-5.0.0.post1-py3-none-win_amd64.whl.metadata (3.5 kB)
Collecting cffi>=1.14.0 (from angr)
  Downloading cffi-1.17.1-cp39-cp39-win_amd64.whl.metadata (1.6 kB)
Collecting claripy==9.2.102 (from angr)
```

(2) 下载 angr 应用文档

anqr / anqr-doc 公共存档

















<> 代码 议题 拉取请求 操作 项目 安全 洞察

master 20 分支 109 标签 转到文件 <> 代码

twizmwazin Add archival notice bf38070 · 2 年之前 1,314 次提交

.github	Fix copy-paste issue in question template	3 年之前
docs	Update pysoot install directions (#428)	3 年之前
examples	Update example to remove references of progressbar (#462)	2 年之前
tests	Remove api docs (#461)	2 年之前
.gitignore	Add SimLibrary to API docs	7 年之前
CHANGELOG.md	Capitalize 'Python' (#405)	3 年之前
CHEATSHEET.md	cast to only support bytes and int (#409)	3 年之前
HACKING.md	black (#445)	2 年之前
HELPWANTED.md	oops there was a better place to put that	4 年之前
INSTALL.md	Capitalize 'Python' (#405)	3 年之前
LICENSE	Add license; closes #225	7 年之前

下载示例文档后如图所示

 .github	2025/5/8 9:59	文件夹
 docs	2025/5/8 9:59	文件夹
 examples	2025/5/8 9:59	文件夹
 tests	2025/5/8 9:59	文件夹
 .gitignore	2025/5/8 9:59	Git Ignore 源文件
 anqr-papers.bib	2025/5/8 9:59	BibTeX 源文件
 book.json	2025/5/8 9:59	JSON 文件
 CHANGELOG.md	2025/5/8 9:59	Markdown 源文件
 CHEATSHEET.md	2025/5/8 9:59	Markdown 源文件
 HACKING.md	2025/5/8 9:59	Markdown 源文件
 HELPWANTED.md	2025/5/8 9:59	Markdown 源文件
 INSTALL.md	2025/5/8 9:59	Markdown 源文件
 LICENSE	2025/5/8 9:59	文件
 MIGRATION.md	2025/5/8 9:59	Markdown 源文件
 README.md	2025/5/8 9:59	Markdown 源文件
 SUMMARY.md	2025/5/8 9:59	Markdown 源文件

其中示例文档里面 examples 文件夹下有我们本次模拟的所需的全部文件，如图所示

此电脑 > 学习 (E:) > angr-doc-master > angr-doc-master > examples > sym-write				
排序 查看 ...				
名称	修改日期	类型	大小	
issue	2025/5/8 9:59	文件	8 KB	
issue.c	2025/5/8 9:59	C 源文件	1 KB	
solve.py	2025/5/8 9:59	Python 源文件	2 KB	

(3) 复现方法一

solve.py 文件源码如下

```
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()))
```

issue.c 文件源码如下

```

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

```

我们运行 solve.py 文件，输出所有的结果，执行成功

```

PS E:\软件安全> & 'd:\anoinstall\envs\pytorch_py39\python.exe' 'c:\Users\jane2\.vscode\extensions\ms-python.debugpy-2025.6.0-win32-x64\bundled\libs\debugpy\launcher' '16072' '-' 'E:\软件安全\1\1.DV'
PS E:\软件安全> e:: cd 'e:\angr-doc-master\angr-doc-master\examples\sym-write'; & 'd:\anoinstall\envs\pytorch_py39\python.exe' 'c:\Users\jane2\.vscode\extensions\ms-python.debugpy-2025.6.0-win32-x64\bundled\libs\debugpy\launcher' '17630' '-' 'e:\angr-doc-master\angr-doc-master\examples\sym-write\solve.py'
WARNING | 2025-05-08 10:14:55,460 | angr.storage.memory_mixins.default_filler_mixin | The program is accessing register with an unspecified value. This could indicate unwanted behavior.
WARNING | 2025-05-08 10:14:55,460 | angr.storage.memory_mixins.default_filler_mixin | angr will cope with this by generating an unconstrained symbolic variable and continuing. You can resolve this by:
WARNING | 2025-05-08 10:14:55,462 | angr.storage.memory_mixins.default_filler_mixin | 1) setting a value to the initial state
WARNING | 2025-05-08 10:14:55,462 | angr.storage.memory_mixins.default_filler_mixin | 2) adding the state option ZERO_FILL_UNCONSTRAINED_MEMORY, REGISTERS, to make unknown regions hold null
WARNING | 2025-05-08 10:14:55,462 | angr.storage.memory_mixins.default_filler_mixin | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_MEMORY, REGISTERS, to suppress these messages.
WARNING | 2025-05-08 10:14:55,462 | angr.storage.memory_mixins.default_filler_mixin | Filling register edi with 4 unconstrained bytes referenced from 0x8048521 (__libc_csu_init+0x1 in issue (0x8048521))
WARNING | 2025-05-08 10:14:55,466 | angr.storage.memory_mixins.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, 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]
PS E:\angr-doc-master\angr-doc-master\examples\sym-write>

```

(4) 方法一代码分析

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 state 之后, 通过 solver 求解器, 求解 u 的值。
7. 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

(5) 复现方法二

solve2.py 文件源码如下

```

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  p = angr.Project("./issue", load_options={"auto_load_libs": False})
8  # hook 函数: addr 为待 hook 的地址
9  # 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
10 # 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
11 # hook 0x08048485 处的指令 (xor eax,eax), 等价于将 eax 设置为 0
12 # 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
13 p.hook(addr=0x08048485, hook=hook_demo, length=2)
14 state = p.factory.blank_state(addr=0x0804846B,
15 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)))

```

运行后结果如图所示, 只输出一个结果 83

```

PS E:\angr-doc-master\angr-doc-master\examples\sym-write> ^C
PS E:\angr-doc-master\angr-doc-master\examples\sym-write> e; cd 'e:\软件安全'; & 'd:\anoinstall\envs\pytorch_py39\python.exe' 'c:\Users\jane2\.vscode\extensions\ms-python.debugpy-2025.6.0-win32-x64\bundled\libs\debugpy\launcher' '17745' '-.' 'E:\软件安全\tr1.py'
PS E:\软件安全> e; cd 'e:\软件安全'; & 'd:\anoinstall\envs\pytorch_py39\python.exe' 'c:\Users\jane2\.vscode\extensions\ms-python.debugpy-2025.6.0-win32-x64\bundled\libs\debugpy\launcher' '17777' '0' '-.' 'E:\软件安全\tr1.py'
PS E:\软件安全> e; cd 'e:\angr-doc-master\angr-doc-master\examples\sym-write'; & 'd:\anoinstall\envs\pytorch_py39\python.exe' 'c:\Users\jane2\.vscode\extensions\ms-python.debugpy-2025.6.0-win32-x64\bundled\libs\debugpy\launcher' '17939' '-.' 'E:\angr-doc-master\angr-doc-master\examples\sym-write\tr1.py'
WARNING | 2025-05-08 10:20:24,909 | angr.storage.memory_mixins.default_filler_mixin | The program is accessing memory with an unspecified value. This could indicate unwanted behavior.
WARNING | 2025-05-08 10:20:24,909 | angr.storage.memory_mixins.default_filler_mixin | angr will cope with this by generating an unconstrained symbolic variable and continuing. You can resolve this by:
WARNING | 2025-05-08 10:20:24,909 | angr.storage.memory_mixins.default_filler_mixin | 1) setting a value to the initial state
WARNING | 2025-05-08 10:20:24,909 | angr.storage.memory_mixins.default_filler_mixin | 2) adding the state option ZERO_FILL_UNCONSTRAINED_MEMORY, REGISTERS, to make unknown regions hold null
WARNING | 2025-05-08 10:20:24,909 | angr.storage.memory_mixins.default_filler_mixin | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_MEMORY, REGISTERS, to suppress these messages.
WARNING | 2025-05-08 10:20:24,909 | angr.storage.memory_mixins.default_filler_mixin | Filling memory at 0x7ffff0000 with 4 unconstrained bytes referenced from 0x8048472 (main+0x7 in issue (0x8048472))
WARNING | 2025-05-08 10:20:24,909 | angr.storage.memory_mixins.default_filler_mixin | Filling register ebp with 4 unconstrained bytes referenced from 0x8048475 (main+0xa in issue (0x8048475))
83
PS E:\angr-doc-master\angr-doc-master\examples\sym-write>

```

(6) 分析

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

(1) angr 简介

angr 是一个支持多处理架构的用于二进制文件分析的工具包，它提供了动态符号执行的能力以及多种静态分析的能力。项目创建的初衷，是为了整合此前多种二进制分析方式的优点，并开发一个平台，以供二进制分析人员比较不同二进制分析方式的优劣，并根据自身需要开发新的二进制分析系统和方式。

也正是因为 angr 是一个二进制文件分析的工具包，因此它可以被使用者扩展，用于自动化逆向工程、漏洞挖掘等多个方面。

angr 官方文档 README - [angr Documentation](#)

angr 的 API 文档 [angr API documentation](#) — [angr 9.2.26 documentation](#)

angr_ctf 项目 GitHub - [jakespringer/angr_ctf](#)

(2) angr 的核心概念

1. 顶层接口

Project 类是 angr 的主类，也是 angr 的开始，通过初始化该类的对象，可以将你想要分析的二进制文件加载进来：

```
import angr
```

```
p = angr.Project('/bin/true')
```

Project 类中有许多方法和属性，例如加载的文件名、架构、程序入口点、大小端等等

2. 状态

状态包含了程序运行时的一切信息，寄存器、内存的值、文件系统以及符号变量等。Project 实际上只是将二进制文件加载进来了，要执行它，实际上是对 SimState 对象进行操作，它是程序的状态。用 docker 来比喻，Project 相当于开发环境，State 则是使用开发环境制作的镜像。要创建状态，需要使用 Project 对象中的 factory，它还可以用于创建模拟管理器和基本块（后面提

到), 如下:

```
init_state = p.factory.entry_state()
```

3. 模拟管理器

上述方式只是预设了程序开始分析时的状态, 我们要分析程序就必须要让它到达下一个状态, 这就需要模拟管理器的帮助 (简称 SM). 使用以下指令能创建一个 SM, 它需要传入一个 state 或者 state 的列表作为参数:

```
simgr = p.factory.simgr(state)
```

stash 是一个列表, 可以使用 python 支持的方式去遍历其中的元素, 也可以使用常见的列表操作。但 angr 提供了一种更高级的方式, 在 stash 名字前加上 one_, 可以得到 stash 中的第一个状态, 加上 mp_, 可以得到一个 multiplexed 版本的 stash

4. 符号执行

符号执行就是给程序传递一个符号而不是具体的值, 让这个符号伴随程序运行, 当碰见分支时, angr 会保存所有分支, 以及分支后的所有分支, 并且在分支时, 保存进入该分支时的判断条件, 通常这些判断条件是对符号的约束。

当 angr 运行到目标状态时, 就可以调用求解器对一路上收集到的约束进行求解, 最终得到某个符号能够到达当前状态的值。

angr 会沿着分支按照某种策略 (默认 DFS) 进行状态搜索, 当达到目标状态 (也就是 backdoor 能够执行的状态), 此时 angr 已经收集了两个约束 ($x > 0$ 以及 $x \leq 5$), 那么 angr 就通过这两个约束对 x 进行求解, 解出来的 x 值就是能够让程序执行 backdoor 的输入。

(3) angr 的使用步骤

准备环境

```
pip install angr
```

加载目标程序

```
import angr
proj = angr.Project('./target_binary', auto_load_libs=False)
```

定义起始地址

```
start_state = proj.factory.entry_state()
```

设置符号化输入

```
arg0 = angr.claripy.BVS('arg0', 8 * 20) # 20 字节的符号变量
start_state = proj.factory.full_init_state(stdin=arg0)
```

创建路径组进行符号执行

```
simgr = proj.factory.simulation_manager(start_state)
```

设置目标和避免条件

```
# 设定目标地址（如打印 "Correct!" 的位置）
target_addr = 0x401234
avoid_addr = 0x401111
simgr.explore(find=target_addr, avoid=avoid_addr)
```

获取结果

```
if simgr.found:
    found_state = simgr.found[0]
    # 输出输入值
    solution = found_state.posix.stdin.concretize()[0]
    print(solution)
```

(4) Angr 的实际使用:

1. 自动路径探索

探索特定条件下才会执行的代码路径，例如后门、崩溃点、逻辑炸弹。

基本步骤为：到达某地址（如触发漏洞的函数）；回溯能否到达某条件分支；构造特定条件输入

```
simgr.explore(find=0x401234, avoid=0x401000)
```

2. 漏洞利用辅助分析

判断是否可以从输入控制流触发漏洞点，如栈溢出函数或 `strcpy()`。

```
cfg = proj.analyses.CFGFast()
vuln_func = cfg.kb.functions.function(name='vulnerable')
```

结合 `simgr.explore(find=...)` 使用，可以进一步找到可控输入。

3. 控制流图分析

重建程序结构，获取所有函数、调用关系、跳转逻辑

```
cfg = proj.analyses.CFGFast()
for func in cfg.kb.functions.values():
    print(f"Function: {func.name} @ {hex(func.addr)}")
```

4. 模拟系统 IO 行为

对系统调用模拟输入输出行为，避免因为环境依赖卡住执行

```
proj = angr.Project('file', auto_load_libs=False)
state = proj.factory.entry_state()
```

5. 条件分支分析

确定某些分支条件成立时输入取值的范围

```
cond = state.solver.eval_upto(sym_var, 5)
print("Possible values:", cond)
```

6. HOOK

跳过复杂或不相关的函数，手动模拟返回值

```
class CustomHook(angr.SimProcedure):
    def run(self):
        return self.state.solver.BVV(1, 32) # 返回常数

proj.hook(0x401234, CustomHook())
```

7. 字符串约束逆推

用于解密类题目中，把某些 hash、加密函数逆推为对应字符串输入。

```
input_hash(input) == target_hash
```

8. 多状态并行模拟

保持多个状态，探索分支分化后的路径，并判断哪些路径终止。

```
simgr = proj.factory.simulation_manager(start_state)
while len(simgr.active) > 0:
    simgr.step()
    print(simgr)
```

9. angr 与 CTF

angr_ctf 则是一个专门针对 angr 的项目，里面有 17 个 angr 相关的题目。这些题目只有一个唯一的要求：你需要找出能够使程序输出 “Good Job” 的输入，这也是符号执行常见的应用场景。

00_angr_find	2022/11/24 17:14	文件夹
01_angr_avoid	2022/11/24 17:05	文件夹
02_angr_find_condition	2022/11/24 17:05	文件夹
03_angr_symbolic_registers	2022/11/24 17:05	文件夹
04_angr_symbolic_stack	2022/11/24 17:05	文件夹
05_angr_symbolic_memory	2022/11/24 17:05	文件夹
06_angr_symbolic_dynamic_memory	2022/11/24 17:05	文件夹
07_angr_symbolic_file	2022/11/24 17:05	文件夹
08_angr_constraints	2022/11/24 17:05	文件夹
09_angr_hooks	2022/11/24 17:05	文件夹
10_angr_simprocedures	2022/11/24 17:05	文件夹
11_angr_sim_scanf	2022/11/24 17:05	文件夹
12_angr_veritesting	2022/11/24 17:05	文件夹
13_angr_static_binary	2022/11/24 17:05	文件夹
14_angr_shared_library	2022/11/24 17:05	文件夹
15_angr_arbitrary_read	2022/11/24 17:05	文件夹
16_angr_arbitrary_write	2022/11/24 17:05	文件夹
17_angr_arbitrary_jump	2022/11/24 17:05	文件夹
dist	2022/11/25 17:38	文件夹
solutions	2022/8/12 3:06	文件夹

项目中序号开头的文件夹里面是题目的源码和题解。

dist 中保存了各个题目编译后的可执行文件，均是 ELF-32bit

solutions 中集合了所有题目的题解，也是所有序号开头文件夹的合集。