Lab 8: Symbolic Execution

Software Testing 2023

2023/05/04

whoami

- Software Quality Lab @ EC547
- TA
  a. 蔡惠喬
     - hctsai.cs10@nycu.edu.tw
  b. 陳舜寧
     - xdev11.cs11@nycu.edu.tw
  c. 張書銘
     - ljp.cs10@nycu.edu.tw

GitHub Repo

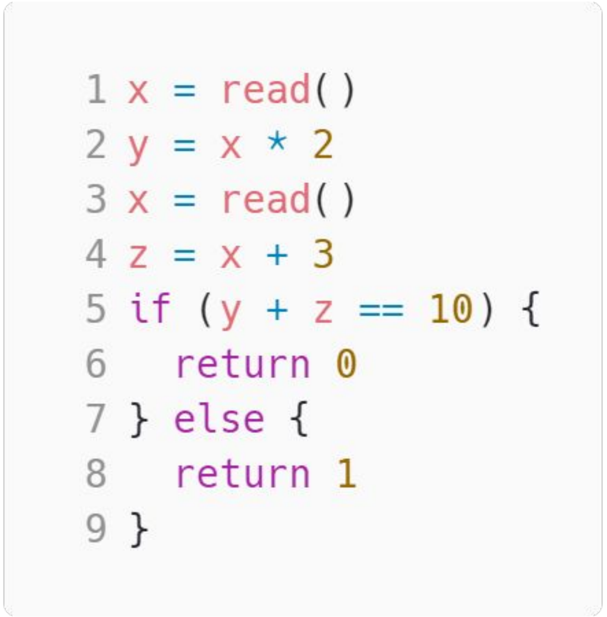
- <student_id>-ST-2023
- Add collaborators
    - XDEv11, chameleon10712, skhuang, LJP-TW

Symbolic Execution

Symbolic Execution

- The program's input was originally a **concrete value**, but this technique treats the input as an **abstract symbol**
- By tracking the operations performed on the input, we can determine **what conditions must be met for each basic block to be executed**
- By using a solver, we can calculate the input that satisfies those conditions, and obtain a solution that allows us to reach a specific basic block

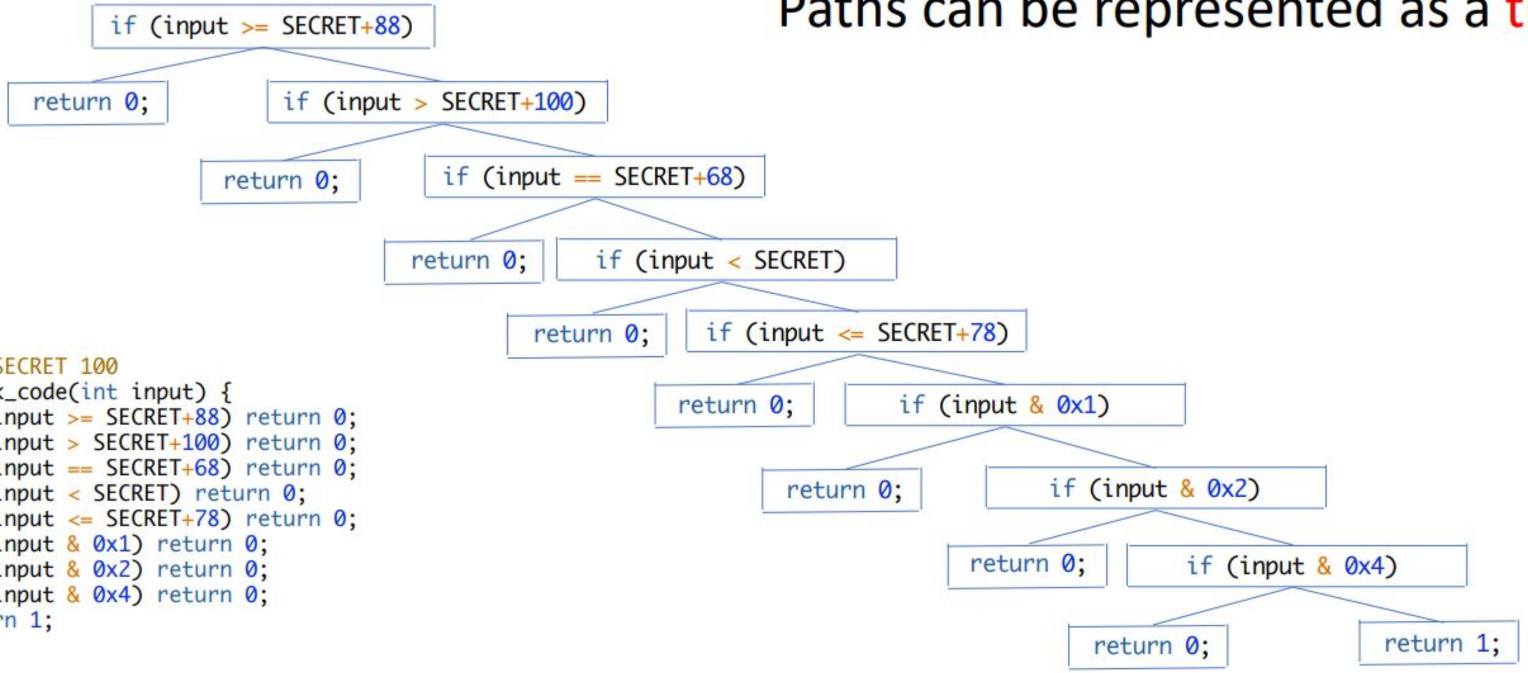
Symbolic Execution



```
1 x = read()
2 y = x * 2
3 x = read()
4 z = x + 3
5 if (y + z == 10) {
6   return 0
7 } else {
8   return 1
9 }
```

Line Number	States & Constraints
1	$x = S_1$
2	$x = S_1; y = S_1 * 2$
3	$x = S_2; y = S_1 * 2$
4	$x = S_2; y = S_1 * 2; z = S_2 + 3$
5	$x = S_2; y = S_1 * 2; z = S_2 + 3;$ $S_1 * 2 + S_2 + 3 == 10$
6	
7	$x = S_2; y = S_1 * 2; z = S_2 + 3;$ $S_1 * 2 + S_2 + 3 != 10$
8	

Symbolic Execution



Paths can be represented as a tree.

```
#define SECRET 100
int check_code(int input) {
    if (input >= SECRET+88) return 0;
    if (input > SECRET+100) return 0;
    if (input == SECRET+68) return 0;
    if (input < SECRET) return 0;
    if (input <= SECRET+78) return 0;
    if (input & 0x1) return 0;
    if (input & 0x2) return 0;
    if (input & 0x4) return 0;
    return 1;
}
```

At each branch operation, the SE engine will “fork” states

Symbolic Execution vs Fuzzing

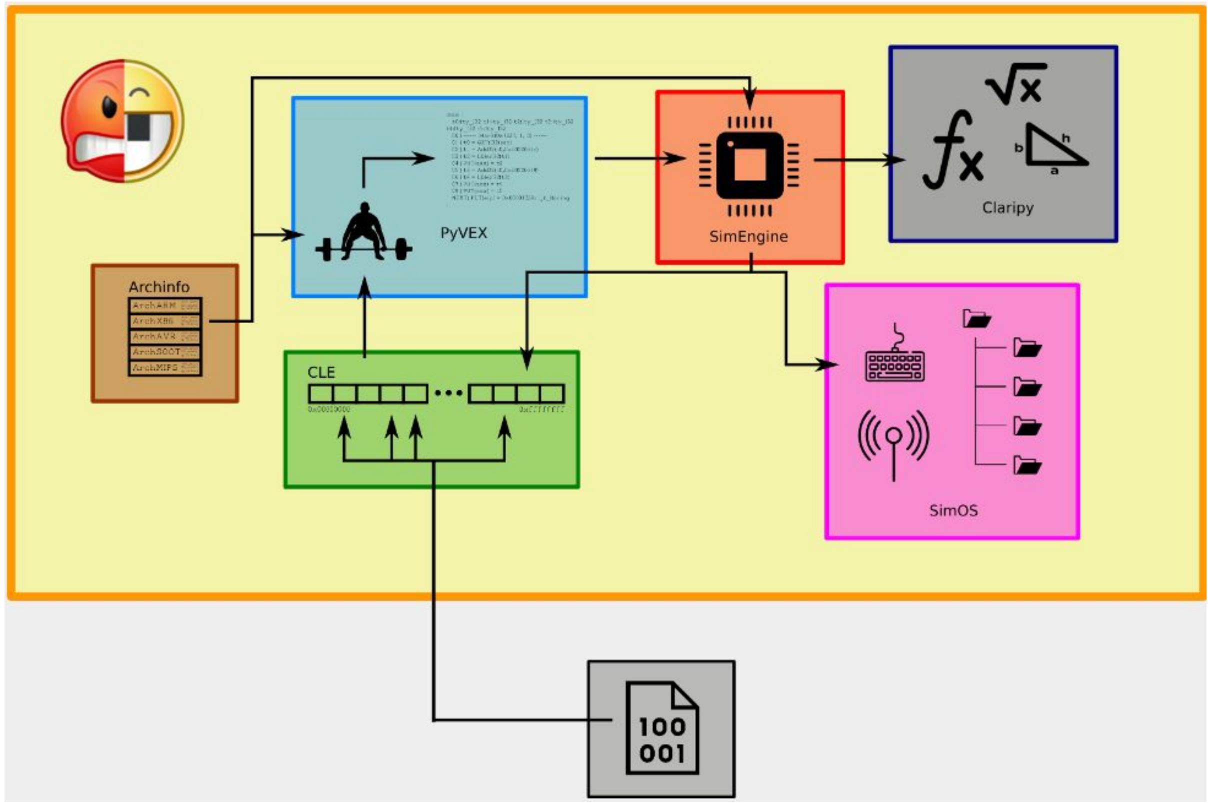
- Fuzzing
  - Pros: Fast
  - Cons: It is difficult to reach basic blocks with too complex conditions
- Symbolic Execution
  - Pros: The required input for reaching a certain basic block can be directly calculated
  - Cons: It is slow and there are other feasibility issues (e.g., path explosion)

Angr

Angr

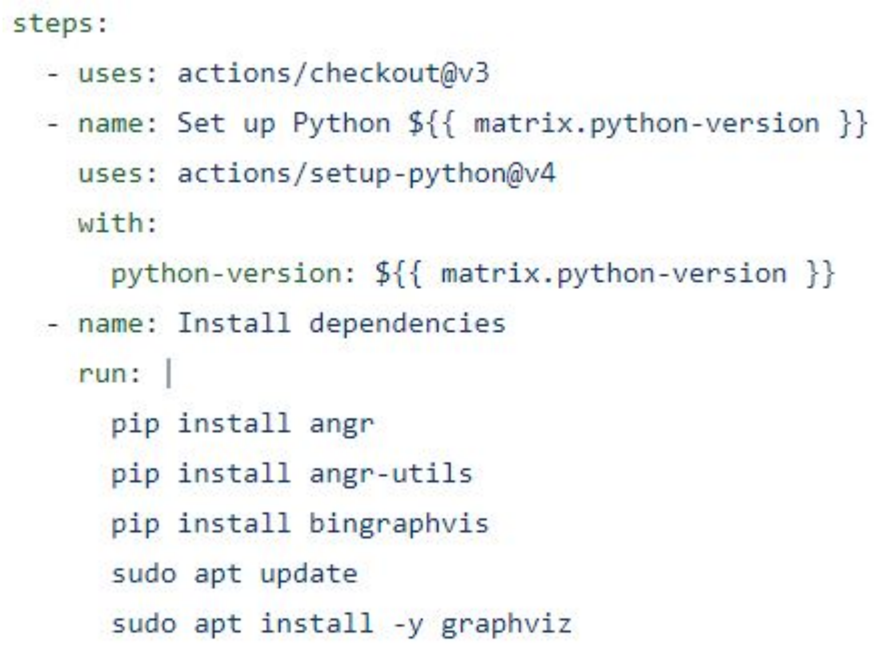
- Disassembly and intermediate-representation lifting
- Program instrumentation
- **Symbolic execution**
- Control-flow analysis
- Data-dependency analysis
- Value-set analysis (VSA)
- Decompilation

Angr Internal



Angr

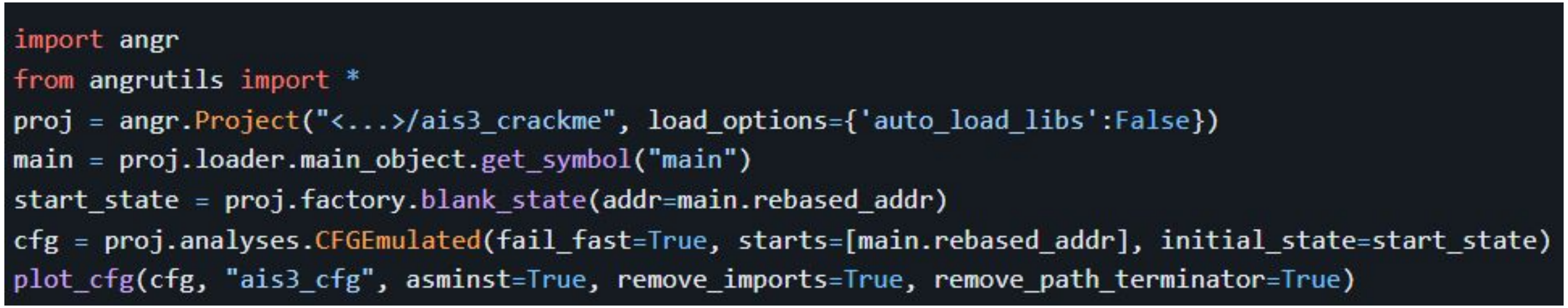
- Install
  - pip install angr
  - The other dependencies are only necessary for drawing CFG using angr



```
steps:
  - uses: actions/checkout@v3
  - name: Set up Python ${{ matrix.python-version }}
    uses: actions/setup-python@v4
    with:
      python-version: ${{ matrix.python-version }}
  - name: Install dependencies
    run: |
      pip install angr
      pip install angr-utils
      pip install bingraphvis
      sudo apt update
      sudo apt install -y graphviz
```

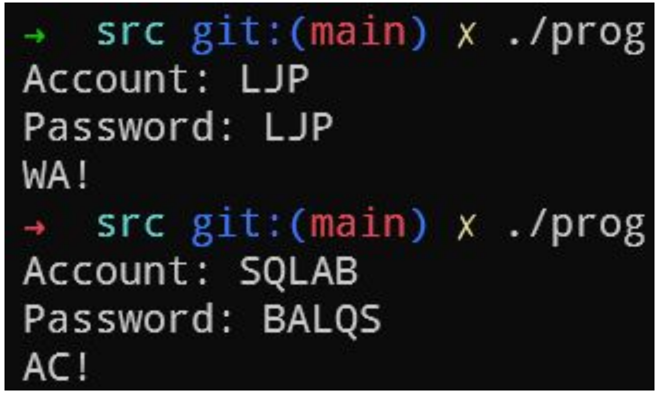
CFG

- Control-Flow Graph
- Use angr-utils to draw the CFG



```python
import angr
from angrutils import *
proj = angr.Project("<...>/ais3_crackme", load_options={'auto_load_libs':False})
main = proj.loader.main_object.get_symbol("main")
start_state = proj.factory.blank_state(addr=main.rebased_addr)
cfg = proj.analyses.CFGEmulated(fail_fast=True, starts=[main.rebased_addr], initial_state=start_state)
plot_cfg(cfg, "ais3_cfg", asminst=True, remove_imports=True, remove_path_terminator=True)
```

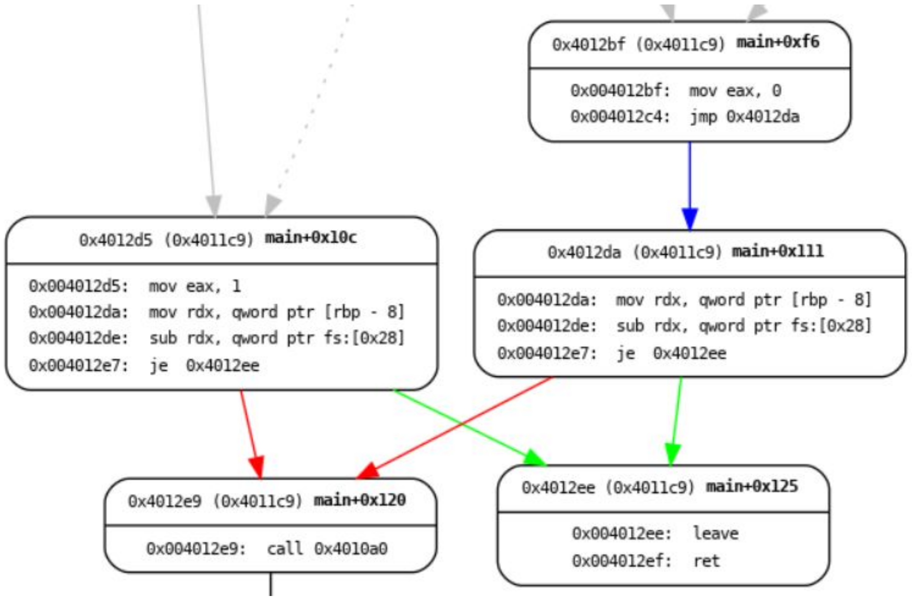
Example



```
➜  src git:(main) ✗ ./prog
Account: LJP
Password: LJP
WA!
➜  src git:(main) ✗ ./prog
Account: SQLAB
Password: BALQS
AC!
```

Lab08/example/src/prog

Example



```
0x4012bf (0x4011c9) main+0xf6

0x004012bf:  mov eax, 0
0x004012c4:  jmp 0x4012da
```

```
0x4012d5 (0x4011c9) main+0x10c

0x004012d5:  mov eax, 1
0x004012da:  mov rdx, qword ptr [rbp - 8]
0x004012de:  sub rdx, qword ptr fs:[0x28]
0x004012e7:  je  0x4012ee
```

```
0x4012da (0x4011c9) main+0x111

0x004012da:  mov rdx, qword ptr [rbp - 8]
0x004012de:  sub rdx, qword ptr fs:[0x28]
0x004012e7:  je  0x4012ee
```

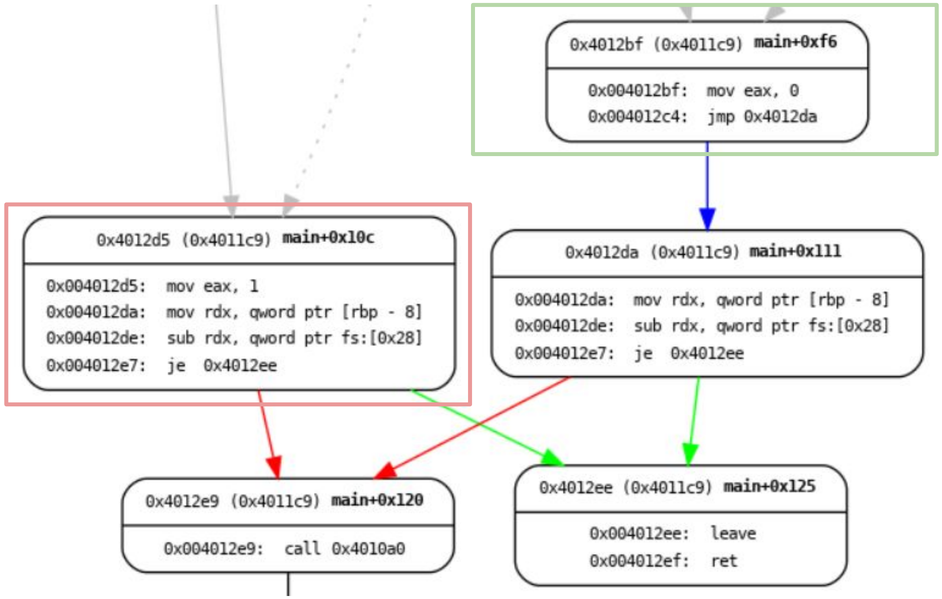
```
0x4012e9 (0x4011c9) main+0x120

0x004012e9:  call 0x4010a0
```

```
0x4012ee (0x4011c9) main+0x125

0x004012ee:  leave
0x004012ef:  ret
```

Lab08/example/my_cfg.png

CFG of main of Lab08/example/src/prog

Example



```
0x4012bf (0x4011c9) main+0xf6
0x004012bf:  mov eax, 0
0x004012c4:  jmp 0x4012da
```

```
0x4012d5 (0x4011c9) main+0x10c
0x004012d5:  mov eax, 1
0x004012da:  mov rdx, qword ptr [rbp - 8]
0x004012de:  sub rdx, qword ptr fs:[0x28]
0x004012e7:  je  0x4012ee
```

```
0x4012da (0x4011c9) main+0x111
0x004012da:  mov rdx, qword ptr [rbp - 8]
0x004012de:  sub rdx, qword ptr fs:[0x28]
0x004012e7:  je  0x4012ee
```

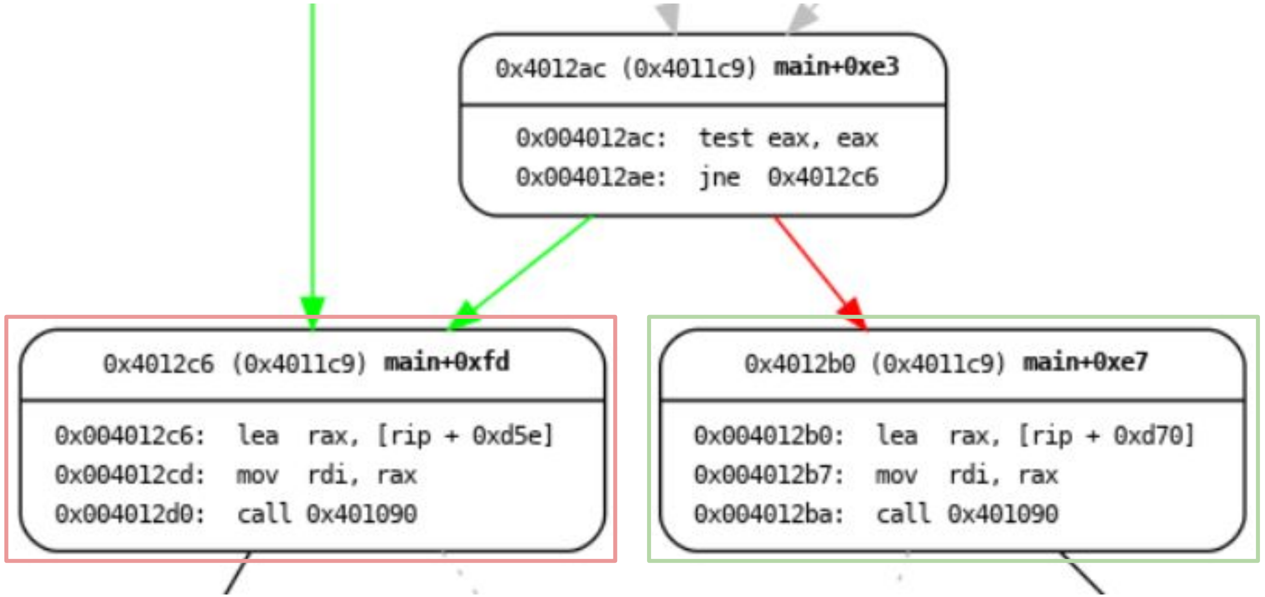
```
0x4012e9 (0x4011c9) main+0x120
0x004012e9:  call 0x4010a0
```

```
0x4012ee (0x4011c9) main+0x125
0x004012ee:  leave
0x004012ef:  ret
```

The return value is stored in the RAX register

If the return value is non-zero, it usually indicates an error

Example



```
0x4012ac (0x4011c9) main+0xe3
0x004012ac:   test eax, eax
0x004012ae:   jne  0x4012c6
```

```
0x4012c6 (0x4011c9) main+0xfd
0x004012c6:   lea  rax, [rip + 0xd5e]
0x004012cd:   mov  rdi, rax
0x004012d0:   call 0x401090
```

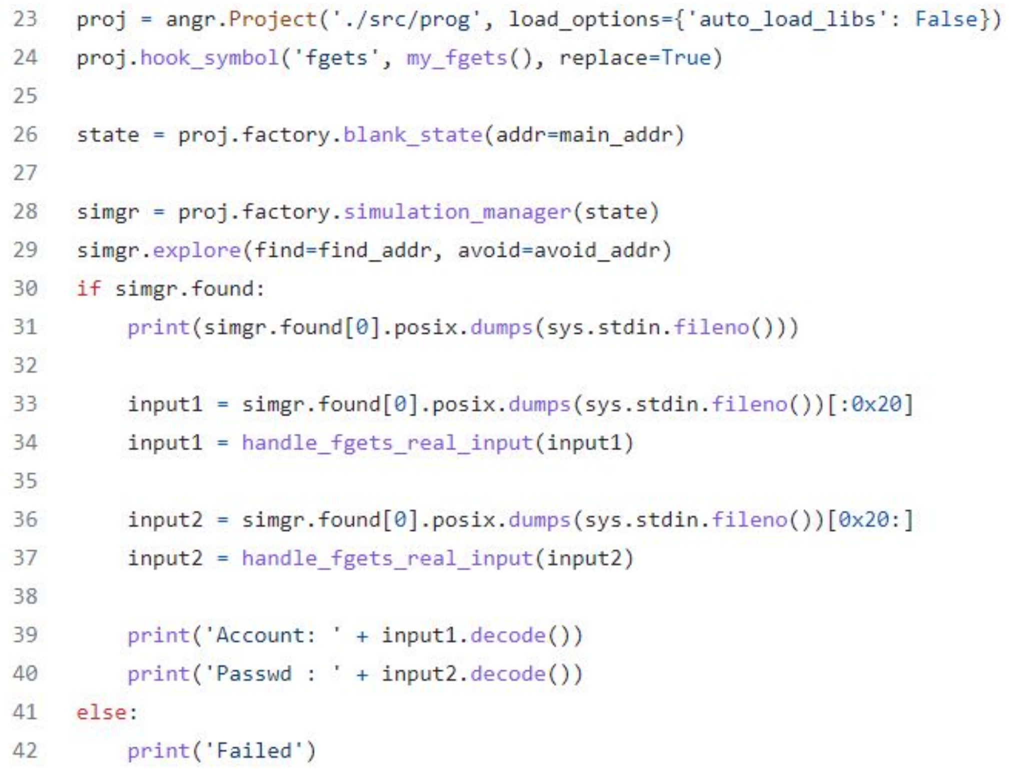
```
0x4012b0 (0x4011c9) main+0xe7
0x004012b0:   lea  rax, [rip + 0xd70]
0x004012b7:   mov  rdi, rax
0x004012ba:   call 0x401090
```

Upon tracing back, it can be found that if the program executes until 0x4012c6, the verification will fail

If it executes until 0x4012b0, the verification will succeed

Angr

1. Create Project
2. Create Init State
3. Create Simulation Manager
4. Explore State
5. Dump the stdin of the state

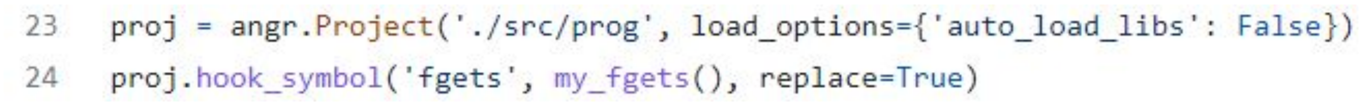


```python
proj = angr.Project('./src/prog', load_options={'auto_load_libs': False})
proj.hook_symbol('fgets', my_fgets(), replace=True)

state = proj.factory.blank_state(addr=main_addr)

simgr = proj.factory.simulation_manager(state)
simgr.explore(find=find_addr, avoid=avoid_addr)
if simgr.found:
    print(simgr.found[0].posix.dumps(sys.stdin.fileno()))

    input1 = simgr.found[0].posix.dumps(sys.stdin.fileno())[:0x20]
    input1 = handle_fgets_real_input(input1)

    input2 = simgr.found[0].posix.dumps(sys.stdin.fileno())[0x20:]
    input2 = handle_fgets_real_input(input2)

    print('Account:  ' + input1.decode())
    print('Passwd :  ' + input2.decode())
else:
    print('Failed')
```

Lab08/example/solve.py

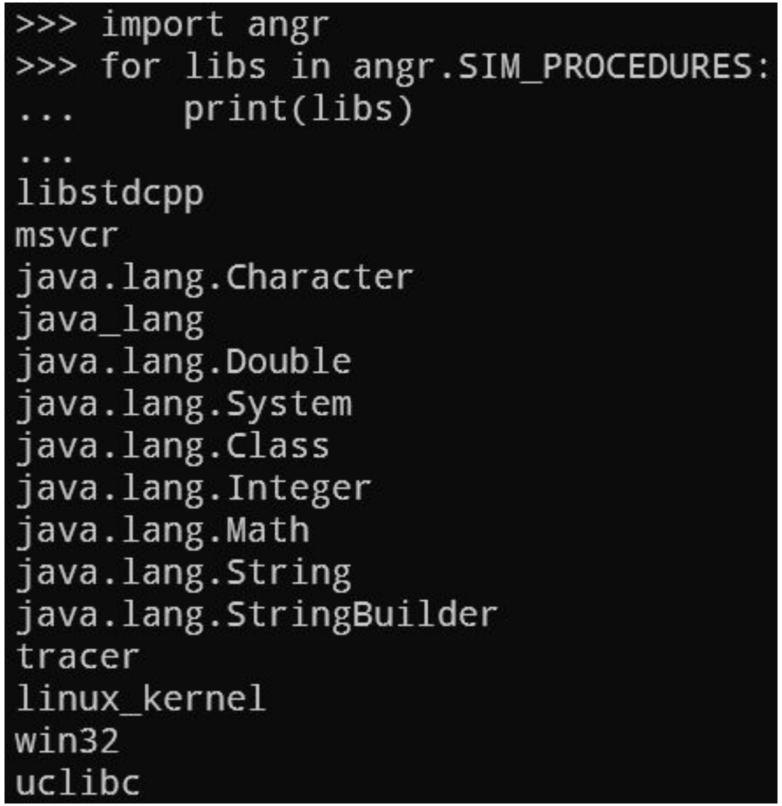
Angr Project

- if auto_load_libs is True (by default)
  - The real library function is executed
  - May cause a path explosion
- if auto_load_libs is False
  - Angr simulates the library function and refers to the simulated function as a SimProcedure
  - User can define custom SimProcedure

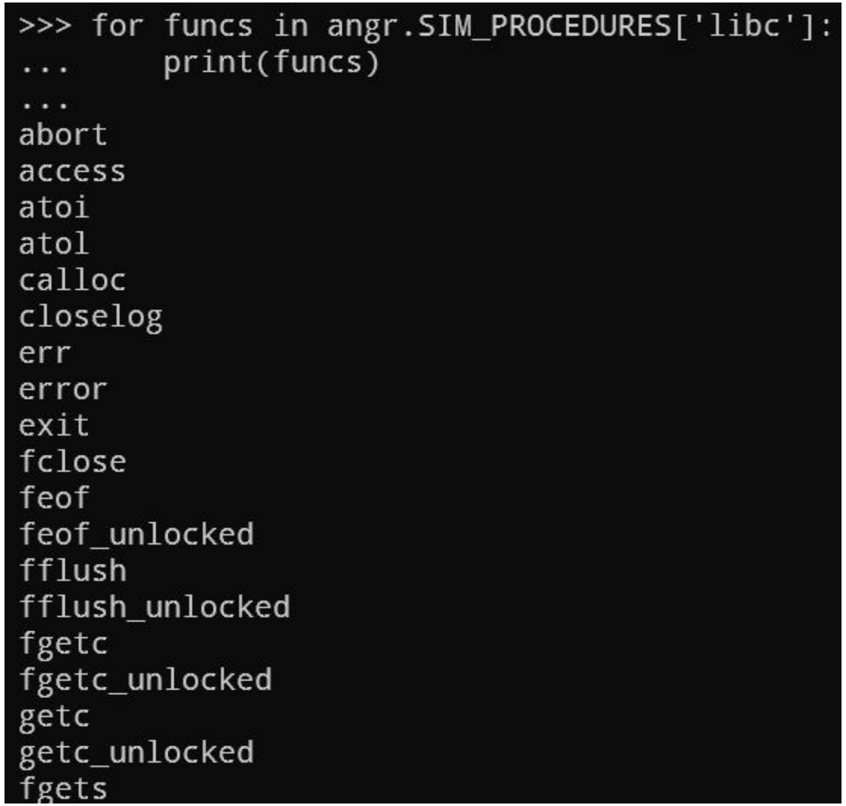


```
proj = angr.Project('./src/prog', load_options={'auto_load_libs': False})
proj.hook_symbol('fgets', my_fgets(), replace=True)
```

SimProcedure

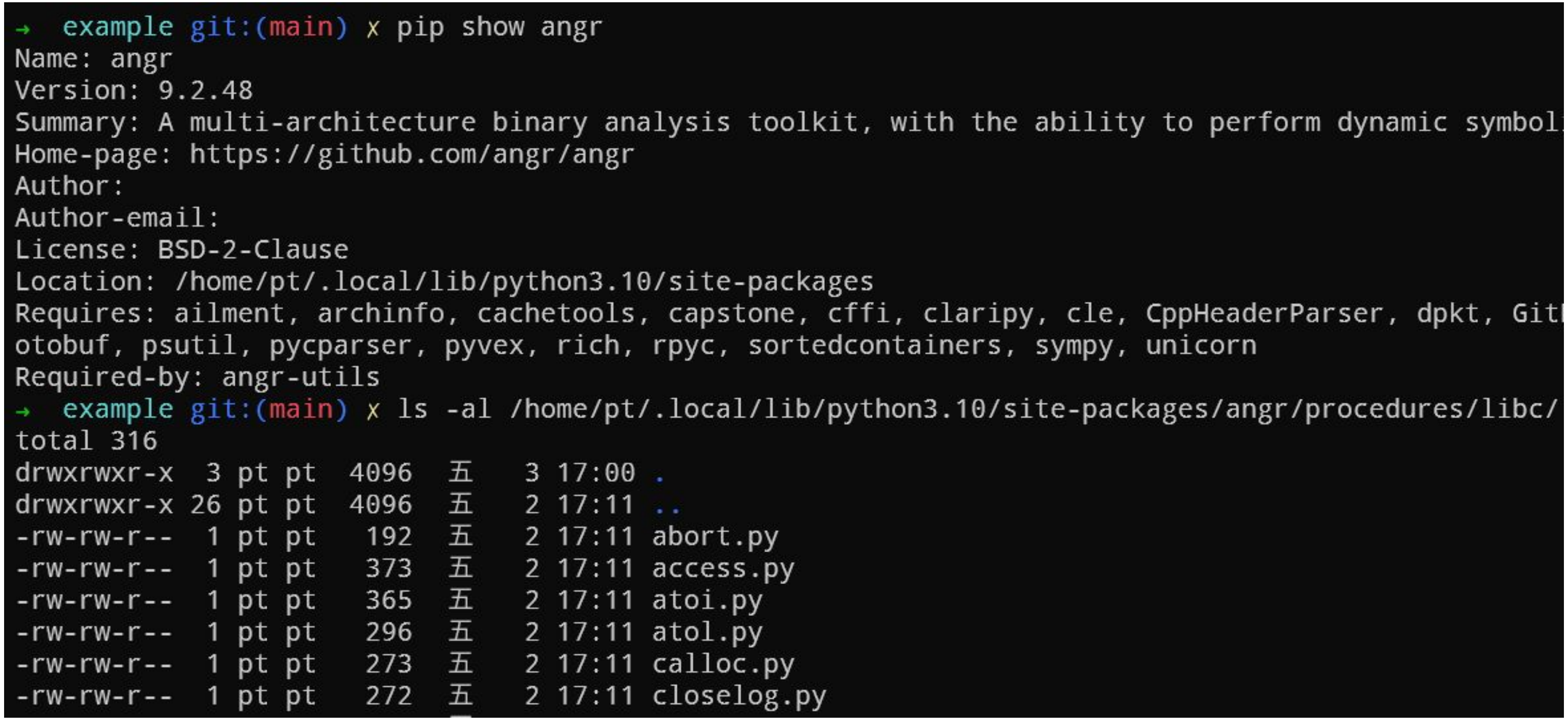


```
>>> import angr
>>> for libs in angr.SIM_PROCEDURES:
...     print(libs)
...
libstdcpp
msvcr
java.lang.Character
java_lang
java.lang.Double
java.lang.System
java.lang.Class
java.lang.Integer
java.lang.Math
java.lang.String
java.lang.StringBuilder
tracer
linux_kernel
win32
uclibc
```



```
>>> for funcs in angr.SIM_PROCEDURES['libc']:
...     print(funcs)
...
abort
access
atoi
atol
calloc
closelog
err
error
exit
fclose
feof
feof_unlocked
fflush
fflush_unlocked
fgetc
fgetc_unlocked
getc
getc_unlocked
fgets
```

SimProcedure



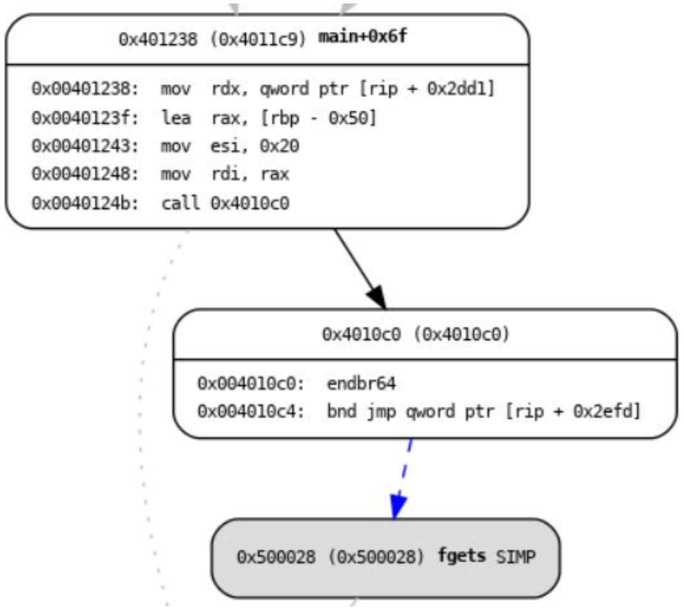
```
➜  example git:(main) ✗ pip show angr
Name: angr
Version: 9.2.48
Summary: A multi-architecture binary analysis toolkit, with the ability to perform dynamic symboli
Home-page: https://github.com/angr/angr
Author:
Author-email:
License: BSD-2-Clause
Location: /home/pt/.local/lib/python3.10/site-packages
Requires: ailment, archinfo, cachetools, capstone, cffi, claripy, cle, CppHeaderParser, dpkt, Git
otobuf, psutil, pycparser, pyvex, rich, rpyc, sortedcontainers, sympy, unicorn
Required-by: angr-utils
➜  example git:(main) ✗ ls -al /home/pt/.local/lib/python3.10/site-packages/angr/procedures/libc/
total 316
drwxrwxr-x  3 pt pt  4096  五   3 17:00 .
drwxrwxr-x 26 pt pt  4096  五   2 17:11 ..
-rw-rw-r--  1 pt pt   192  五   2 17:11 abort.py
-rw-rw-r--  1 pt pt   373  五   2 17:11 access.py
-rw-rw-r--  1 pt pt   365  五   2 17:11 atoi.py
-rw-rw-r--  1 pt pt   296  五   2 17:11 atol.py
-rw-rw-r--  1 pt pt   273  五   2 17:11 calloc.py
-rw-rw-r--  1 pt pt   272  五   2 17:11 closelog.py
```

Custom SimProcedure

- At times, the original SimProcedure may be overly complicated
  - Bad performance
  - Worse still, may result in the inability to explore a specific state

- In such situations, we can create a simplified SimProcedure

Custom SimProcedure


```
0x401238 (0x4011c9) main+0x6f

0x00401238:  mov   rdx, qword ptr [rip + 0x2dd1]
0x0040123f:  lea   rax, [rbp - 0x50]
0x00401243:  mov   esi, 0x20
0x00401248:  mov   rdi, rax
0x0040124b:  call  0x4010c0
```

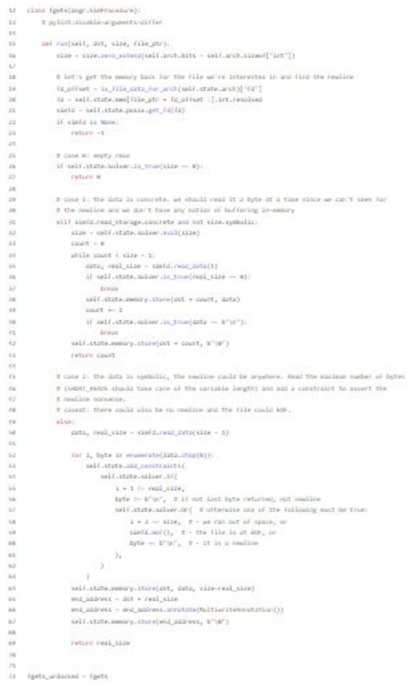
```
0x4010c0 (0x4010c0)

0x004010c0:  endbr64
0x004010c4:  bnd jmp qword ptr [rip + 0x2efd]
```

```
0x500028 (0x500028) fgets SIMP
```

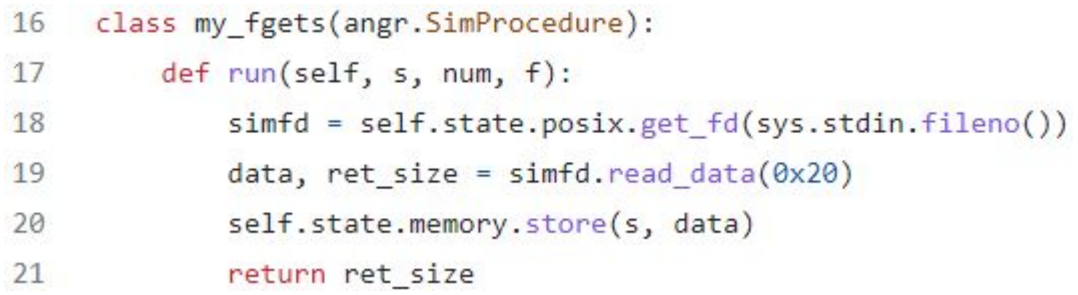
The parameters are stored in the following registers in order: RDI, RSI, RDX, RCX, R8, R9

fgets(rbp-0x50, 0x20, 0x40123f + 0x2dd1) → Read up to 0x20 bytes at most

Custom SimProcedure



```python
class fgets(angr.SimProcedure):
    # pylint:disable=arguments-differ

    def run(self, dst, size, file_ptr):
        size = size.zero_extend(self.arch.bits - self.arch.sizeof["int"])

        # let's get the memory back for the file we're interested in and find the newline
        fd_offset = io_file_data_for_arch(self.state.arch)["fd"]
        fd = self.state.mem[file_ptr + fd_offset :].int.resolved
        simfd = self.state.posix.get_fd(fd)
        if simfd is None:
            return -1

        # case 0: empty read
        if self.state.solver.is_true(size == 0):
            return 0

        # case 1: the data is concrete. we should read it a byte at a time since we can't seek for
        # the newline and we don't have any notion of buffering in-memory
        elif simfd.read_storage.concrete and not size.symbolic:
            size = self.state.solver.eval(size)
            count = 0
            while count < size - 1:
                data, real_size = simfd.read_data(1)
                if self.state.solver.is_true(real_size == 0):
                    break
                self.state.memory.store(dst + count, data)
                count += 1
                if self.state.solver.is_true(data == b"\n"):
                    break
            self.state.memory.store(dst + count, b"\0")
            return count

        # case 2: the data is symbolic, the newline could be anywhere. Read the maximum number of bytes
        # (SHORT_READS should take care of the variable length) and add a constraint to assert the
        # newline nonsense.
        # caveat: there could also be no newline and the file could EOF.
        else:
            data, real_size = simfd.read_data(size - 1)

            for i, byte in enumerate(data.chop(8)):
                self.state.add_constraints(
                    self.state.solver.If(
                        i + 1 != real_size,
                        byte != b"\n",  # if not last byte returned, not newline
                        self.state.solver.Or(  # otherwise one of the following must be true:
                            i + 2 == size,  # - we ran out of space, or
                            simfd.eof(),  # - the file is at EOF, or
                            byte == b"\n",  # - it is a newline
                        ),
                    )
                )
            self.state.memory.store(dst, data, size=real_size)
            end_address = dst + real_size
            end_address = end_address.annotate(MultiwriteAnnotation())
            self.state.memory.store(end_address, b"\0")

            return real_size


fgets_unlocked = fgets
```



```python
class my_fgets(angr.SimProcedure):
    def run(self, s, num, f):
        simfd = self.state.posix.get_fd(sys.stdin.fileno())
        data, ret_size = simfd.read_data(0x20)
        self.state.memory.store(s, data)
        return ret_size
```

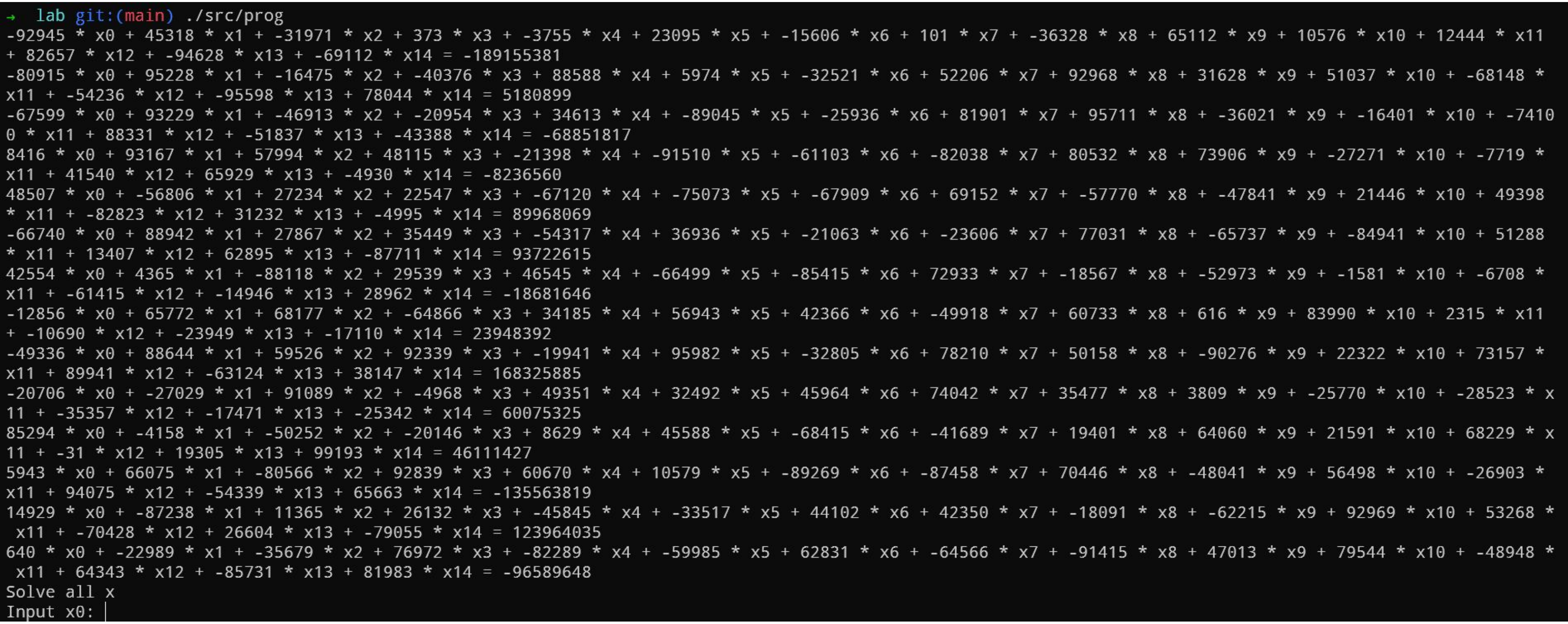
Write a custom SimProcedure to override the original one

Lab

Lab 8

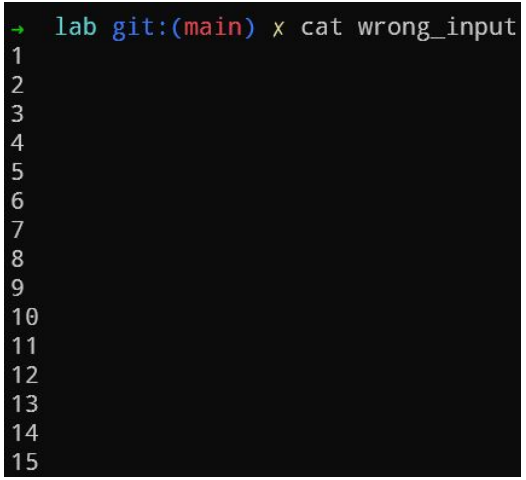
- Write a Python script using angr to obtain a solution that can solve the lab/src/prog
- Save the solution in lab/ with the filename "solve_input"
- The "solve_input" file will be used as the stdin to test lab/src/prog
- You can only modify the files listed below:
    - lab/solve.py

Lab 8

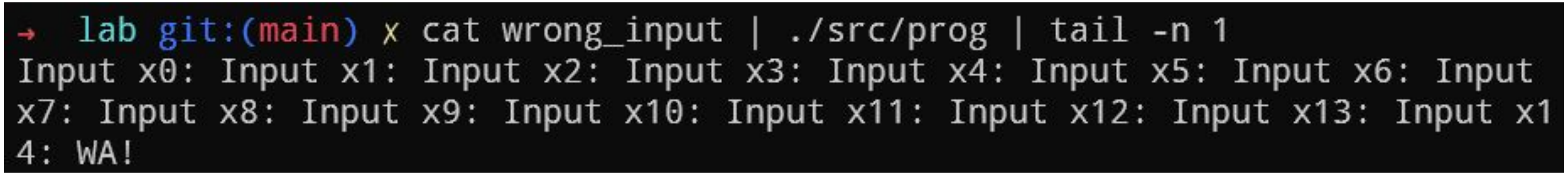


```
→  lab git:(main) ./src/prog
-92945 * x0 + 45318 * x1 + -31971 * x2 + 373 * x3 + -3755 * x4 + 23095 * x5 + -15606 * x6 + 101 * x7 + -36328 * x8 + 65112 * x9 + 10576 * x10 + 12444 * x11
+ 82657 * x12 + -94628 * x13 + -69112 * x14 = -189155381
-80915 * x0 + 95228 * x1 + -16475 * x2 + -40376 * x3 + 88588 * x4 + 5974 * x5 + -32521 * x6 + 52206 * x7 + 92968 * x8 + 31628 * x9 + 51037 * x10 + -68148 *
x11 + -54236 * x12 + -95598 * x13 + 78044 * x14 = 5180899
-67599 * x0 + 93229 * x1 + -46913 * x2 + -20954 * x3 + 34613 * x4 + -89045 * x5 + -25936 * x6 + 81901 * x7 + 95711 * x8 + -36021 * x9 + -16401 * x10 + -7410
0 * x11 + 88331 * x12 + -51837 * x13 + -43388 * x14 = -68851817
8416 * x0 + 93167 * x1 + 57994 * x2 + 48115 * x3 + -21398 * x4 + -91510 * x5 + -61103 * x6 + -82038 * x7 + 80532 * x8 + 73906 * x9 + -27271 * x10 + -7719 *
x11 + 41540 * x12 + 65929 * x13 + -4930 * x14 = -8236560
48507 * x0 + -56806 * x1 + 27234 * x2 + 22547 * x3 + -67120 * x4 + -75073 * x5 + -67909 * x6 + 69152 * x7 + -57770 * x8 + -47841 * x9 + 21446 * x10 + 49398
* x11 + -82823 * x12 + 31232 * x13 + -4995 * x14 = 89968069
-66740 * x0 + 88942 * x1 + 27867 * x2 + 35449 * x3 + -54317 * x4 + 36936 * x5 + -21063 * x6 + -23606 * x7 + 77031 * x8 + -65737 * x9 + -84941 * x10 + 51288
* x11 + 13407 * x12 + 62895 * x13 + -87711 * x14 = 93722615
42554 * x0 + 4365 * x1 + -88118 * x2 + 29539 * x3 + 46545 * x4 + -66499 * x5 + -85415 * x6 + 72933 * x7 + -18567 * x8 + -52973 * x9 + -1581 * x10 + -6708 *
x11 + -61415 * x12 + -14946 * x13 + 28962 * x14 = -18681646
-12856 * x0 + 65772 * x1 + 68177 * x2 + -64866 * x3 + 34185 * x4 + 56943 * x5 + 42366 * x6 + -49918 * x7 + 60733 * x8 + 616 * x9 + 83990 * x10 + 2315 * x11
+ -10690 * x12 + -23949 * x13 + -17110 * x14 = 23948392
-49336 * x0 + 88644 * x1 + 59526 * x2 + 92339 * x3 + -19941 * x4 + 95982 * x5 + -32805 * x6 + 78210 * x7 + 50158 * x8 + -90276 * x9 + 22322 * x10 + 73157 *
x11 + 89941 * x12 + -63124 * x13 + 38147 * x14 = 168325885
-20706 * x0 + -27029 * x1 + 91089 * x2 + -4968 * x3 + 49351 * x4 + 32492 * x5 + 45964 * x6 + 74042 * x7 + 35477 * x8 + 3809 * x9 + -25770 * x10 + -28523 * x
11 + -35357 * x12 + -17471 * x13 + -25342 * x14 = 60075325
85294 * x0 + -4158 * x1 + -50252 * x2 + -20146 * x3 + 8629 * x4 + 45588 * x5 + -68415 * x6 + -41689 * x7 + 19401 * x8 + 64060 * x9 + 21591 * x10 + 68229 * x
11 + -31 * x12 + 19305 * x13 + 99193 * x14 = 46111427
5943 * x0 + 66075 * x1 + -80566 * x2 + 92839 * x3 + 60670 * x4 + 10579 * x5 + -89269 * x6 + -87458 * x7 + 70446 * x8 + -48041 * x9 + 56498 * x10 + -26903 *
x11 + 94075 * x12 + -54339 * x13 + 65663 * x14 = -135563819
14929 * x0 + -87238 * x1 + 11365 * x2 + 26132 * x3 + -45845 * x4 + -33517 * x5 + 44102 * x6 + 42350 * x7 + -18091 * x8 + -62215 * x9 + 92969 * x10 + 53268 *
 x11 + -70428 * x12 + 26604 * x13 + -79055 * x14 = 123964035
640 * x0 + -22989 * x1 + -35679 * x2 + 76972 * x3 + -82289 * x4 + -59985 * x5 + 62831 * x6 + -64566 * x7 + -91415 * x8 + 47013 * x9 + 79544 * x10 + -48948 *
 x11 + 64343 * x12 + -85731 * x13 + 81983 * x14 = -96589648
Solve all x
Input x0: 
```

Lab 8

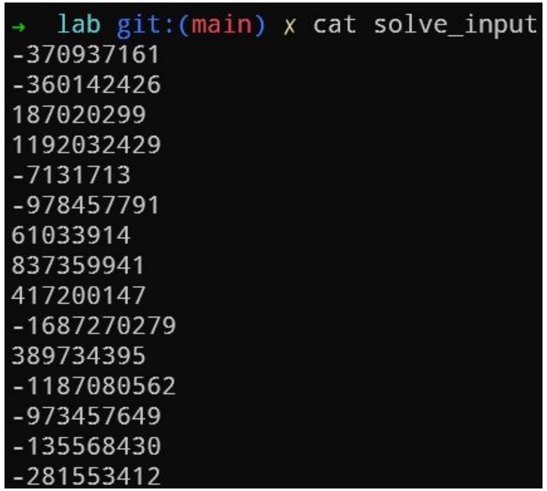


```
→  lab git:(main) ✗ cat wrong_input
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

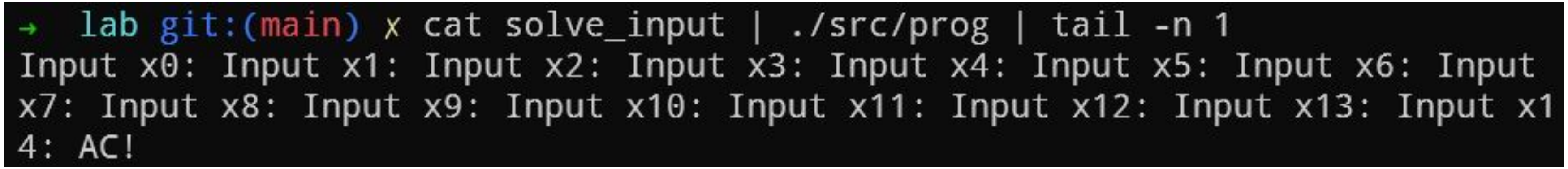


```
→  lab git:(main) ✗ cat wrong_input | ./src/prog | tail -n 1
Input x0: Input x1: Input x2: Input x3: Input x4: Input x5: Input x6: Input
x7: Input x8: Input x9: Input x10: Input x11: Input x12: Input x13: Input x1
4: WA!
```

Lab 8



```
→  lab git:(main) ✗ cat solve_input
-370937161
-360142426
187020299
1192032429
-7131713
-978457791
61033914
837359941
417200147
-1687270279
389734395
-1187080562
-973457649
-135568430
-281553412
```

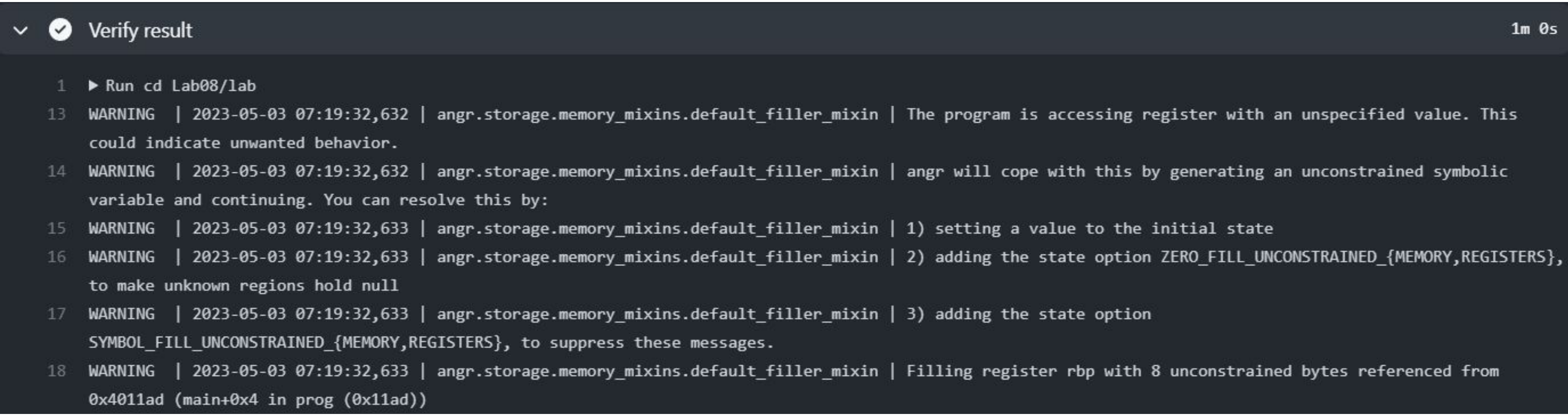


```
→  lab git:(main) ✗ cat solve_input | ./src/prog | tail -n 1
Input x0: Input x1: Input x2: Input x3: Input x4: Input x5: Input x6: Input
x7: Input x8: Input x9: Input x10: Input x11: Input x12: Input x13: Input x1
4: AC!
```

Lab Environment

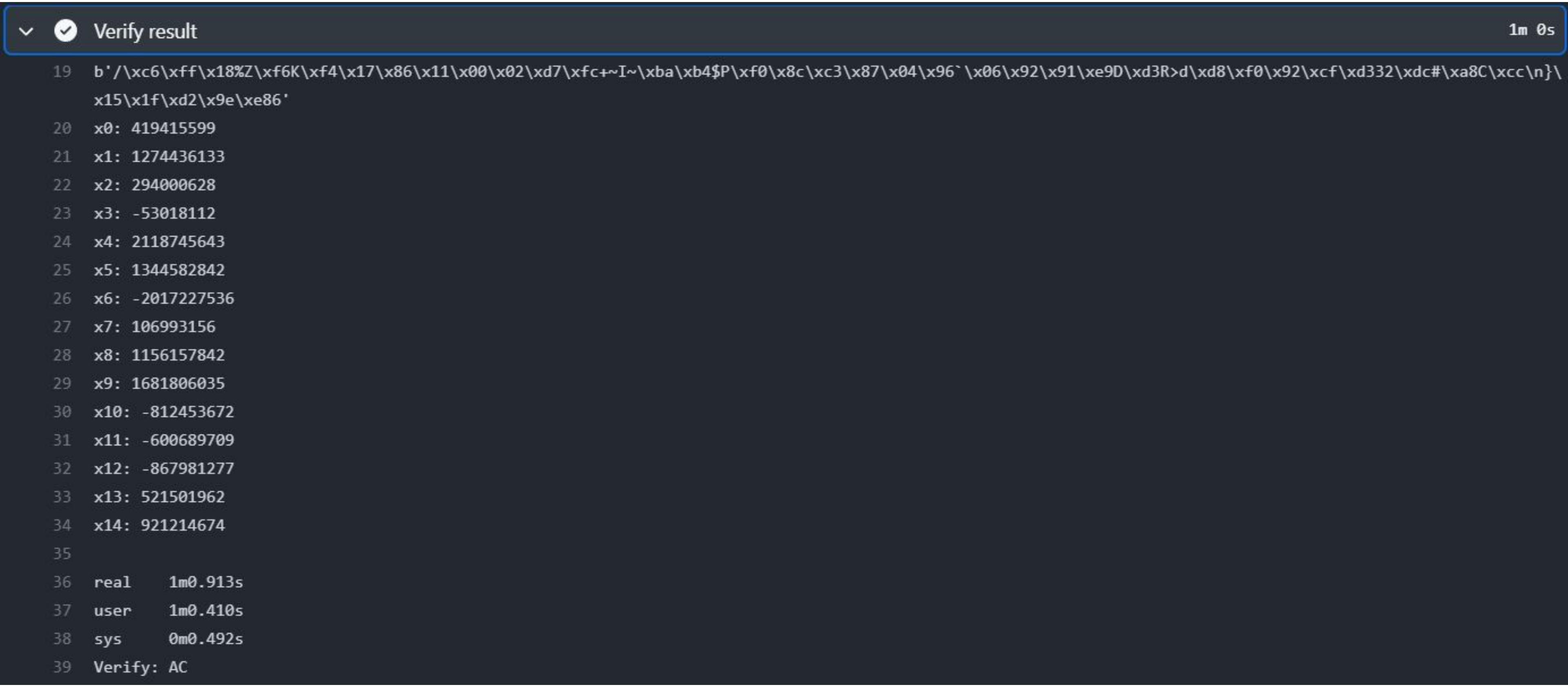
- Ubuntu 22.04
- Python 3.10

Example Output



Verify result 1m 0s

```
▶ Run cd Lab08/lab
WARNING  | 2023-05-03 07:19:32,632 | angr.storage.memory_mixins.default_filler_mixin | The program is accessing register with an unspecified value. This could indicate unwanted behavior.
WARNING  | 2023-05-03 07:19:32,632 | 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  | 2023-05-03 07:19:32,633 | angr.storage.memory_mixins.default_filler_mixin | 1) setting a value to the initial state
WARNING  | 2023-05-03 07:19:32,633 | angr.storage.memory_mixins.default_filler_mixin | 2) adding the state option ZERO_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to make unknown regions hold null
WARNING  | 2023-05-03 07:19:32,633 | angr.storage.memory_mixins.default_filler_mixin | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to suppress these messages.
WARNING  | 2023-05-03 07:19:32,633 | angr.storage.memory_mixins.default_filler_mixin | Filling register rbp with 8 unconstrained bytes referenced from 0x4011ad (main+0x4 in prog (0x11ad))
```



Verify result 1m 0s

```
b'/\xc6\xff\x18%Z\xf6K\xf4\x17\x86\x11\x00\x02\xd7\xfc+~I~\xba\xb4$P\xf0\x8c\xc3\x87\x04\x96`\x06\x92\x91\xe9D\xd3R>d\xd8\xf0\x92\xcf\xd332\xdc#\xa8C\xcc\n}\
x15\x1f\xd2\x9e\xe86'
x0: 419415599
x1: 1274436133
x2: 294000628
x3: -53018112
x4: 2118745643
x5: 1344582842
x6: -2017227536
x7: 106993156
x8: 1156157842
x9: 1681806035
x10: -812453672
x11: -600689709
x12: -867981277
x13: 521501962
x14: 921214674

real    1m0.913s
user    1m0.410s
sys     0m0.492s
Verify: AC
```

Submission

Submission

- Add Lab08 status badge in your README file
- Please submit your Github repo `<student_id>-ST-2023`, and upload these to E3:
    - commit URL
        - refer to Lab 1 submission
    - github action job URL
        - refer to Lab 3 submission

Reference

- Understanding Symbolic Execution
- throwing a tantrum, part 1: angr internals
- angr documentation
- angr-doc/CHEATSHEET.md at master
- Binary 自動分析的那些事
- angr入门小试
- GitHub - axt/angr-utils: Handy utilities for the angr binary analysis framework, most notably CFG visualization