Lab 8: Symbolic Execution

Software Testing 2022

2022/04/28

Symbolic Execution

Symbolic Execution

- 符號執行
- 透過分析程式，讓輸入可以到達特定的 basic block。
- 通常會使用一個符號值 (λ) 作為輸入，而非具體值。
- 在程式執行時可以得到相應的 path constraint，然後通過 constraint solver 來取得可以到達目標的具體值。

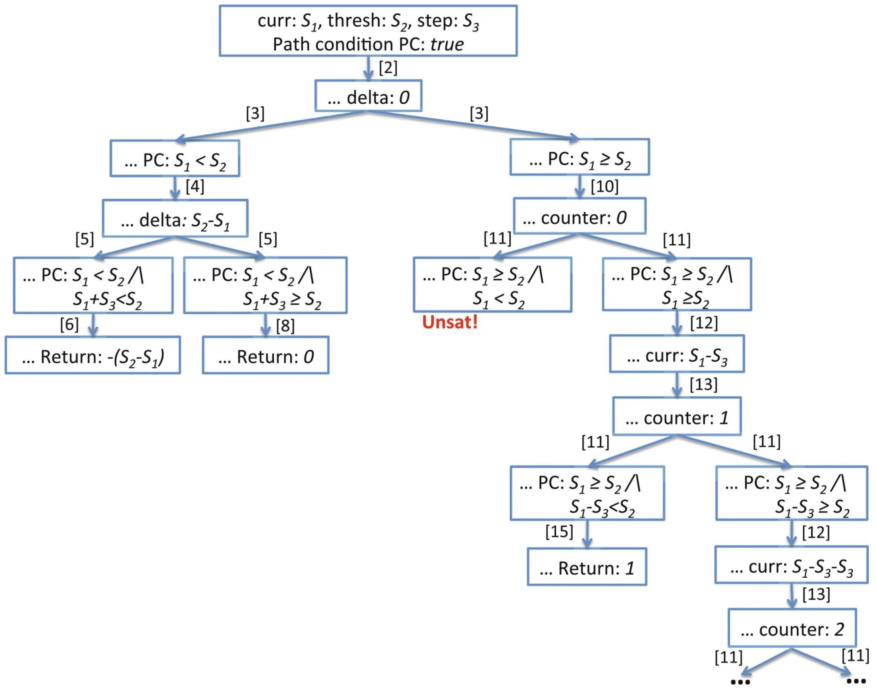
Example

```
1  int f() {
2    y = read();
3    z = y * 2;
4    if (z == 12) {
5      fail();
6    } else {
7      printf("OK");
8    }
9  }
```

2	y = read()	y = λ
3	z = y * 2	z = λ * 2
4	z == 12	λ * 2 == 12
5	fail()	
6	z != 12	λ * 2 != 12
7	printf()	

More Example

```
1  int compute(int curr, int thresh, int step){
2    int delta = 0;
3    if (curr < thresh){
4      delta = thresh - curr;
5      if ((curr + step) < thresh)
6        return -delta;
7      else
8        return 0;
9    } else {
10       int counter = 0;
11       while (curr >= thresh) {
12         curr = curr - step;
13         counter++;
14       }
15       return counter;
16   }
17 }
```



常見的工具

- KLEE
- S2E
- Angr

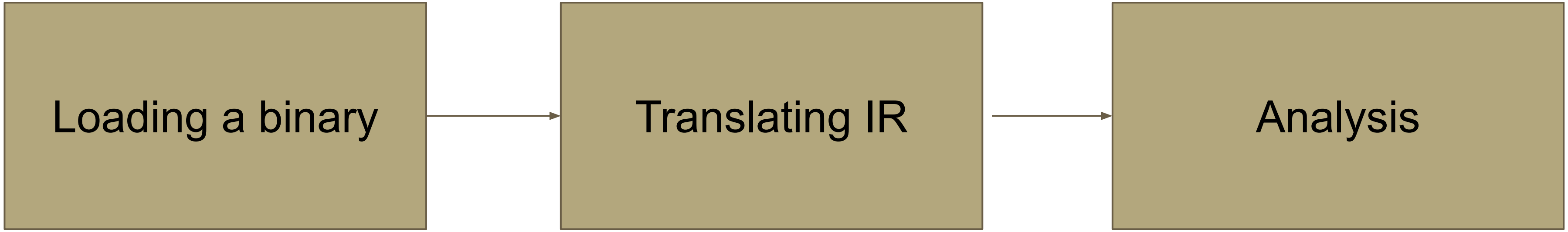
Angr

- Disassembly and intermediate-representation lifting
- Program instrumentation
- Symbolic execution
- Control-flow analysis
- Data-dependency analysis
- Value-set analysis (VSA)
- Decompilation
- Github: https://github.com/angr/angr

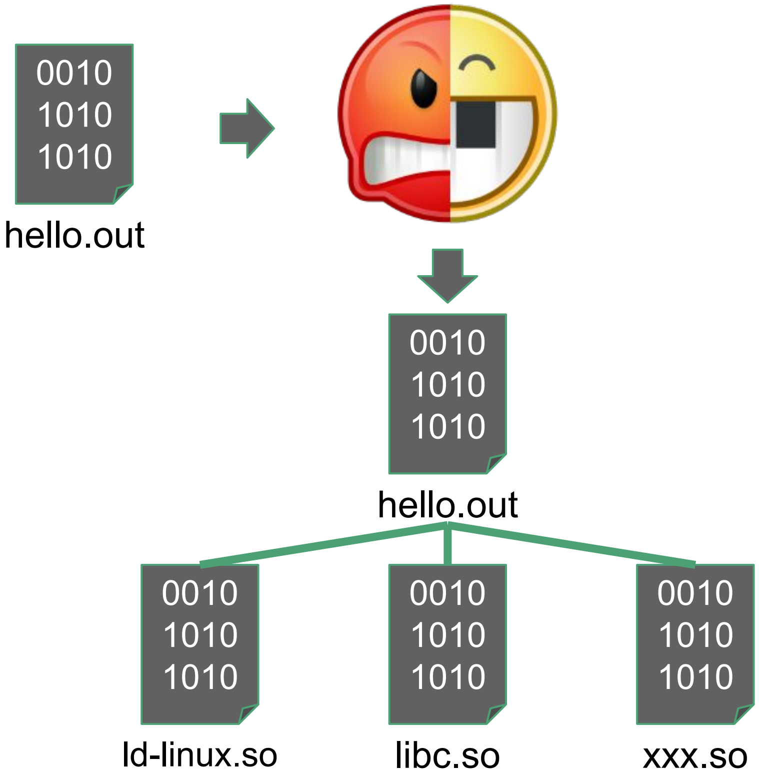
Angr install

- 基本只要
  - `$ pip3 install angr`
  - `$ pip3 install angr-utils`
  - `$ pip3 install bingraphvis`
  - `$ sudo apt install graphviz`
- angr 會使用一些特殊版本 libraries
  - 推薦使用 python 虛擬環境
  - `$ sudo apt-get install python3-dev libffi-dev build-essential virtualenvwrapper`
  - 改.bashrc (https://ithelp.ithome.com.tw/articles/10265702)
  - `$ mkvirtualenv --python=$(which python3) angr && pip install angr angr-utils bingraphvis`
- 以上為 ubuntu 如果是別的系統或安裝過程有問題, 可以參考:
  - https://docs.angr.io/introductory-errata/install

Angr 內部執行流程



The loader


0010
1010
1010
hello.out
0010
1010
1010
hello.out
0010
1010
1010
ld-linux.so
0010
1010
1010
libc.so
0010
1010
1010
xxx.so

Use

```
In [1]: import angr

In [2]: proj = angr.Project('./target')
WARNING | 2022-04-27 23:43:52,014 | cle.loader | The main binary is a position-i
ndependent executable. It is being loaded with a base address of 0x400000.
^[[A^[[A
In [3]: hex(proj.entry)
Out[3]: '0x4010c0'
```

The factory

- block
- states
- simulation managers

block

```
In [4]: block = proj.factory.block(proj.entry)

In [5]: block.pp()
0x4010c0:       endbr64
0x4010c4:       xor     ebp, ebp
0x4010c6:       mov     r9, rdx
0x4010c9:       pop     rsi
0x4010ca:       mov     rdx, rsp
0x4010cd:       and     rsp, 0xfffffffffffffff0
0x4010d1:       push    rax
0x4010d2:       push    rsp
0x4010d3:       lea     r8, [rip + 0x956]
0x4010da:       lea     rcx, [rip + 0x8df]
0x4010e1:       lea     rdi, [rip + 0x798]
0x4010e8:       call    qword ptr [rip + 0x2eea]
```

states

- state => a program at a given point in time
- 取得 program 初始狀態

```
In [6]: state = proj.factory.entry_state()

In [7]: state.regs.rip
Out[7]: <BV64 0x4010c0>
```

simulation manager

- primary interface in angr for performing execution
- contain several stashes of states
    - active is initialized with the state we passed in

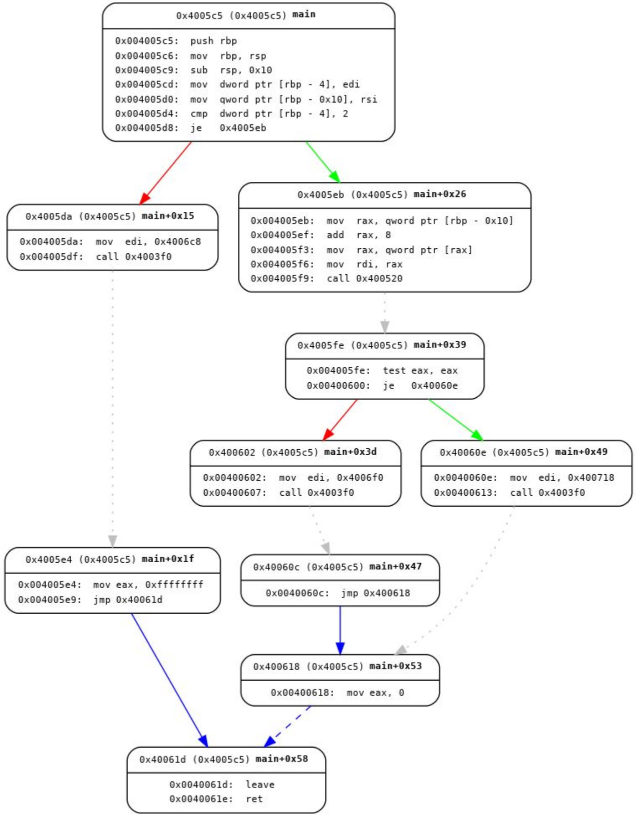
```
In [8]: simgr = proj.factory.simulation_manager(state)

In [9]: simgr.active
Out[9]: [<SimState @ 0x4010c0>]

In [10]: simgr.step()
Out[10]: <SimulationManager with 1 active>

In [11]: simgr.active
Out[11]: [<SimState @ 0x523fc0>]

In [12]: simgr.active[0].regs.rip
Out[12]: <BV64 0x523fc0>

In [13]: state.regs.rip
Out[13]: <BV64 0x4010c0>
```

CFG

- angr 可以產生 binary 的 cfg
- 使用方法 : https://github.com/axt/angr-utils
- 注意事項:
    - `proj = angr.Project("./a.out", load_options={'auto_load_libs':False})`
        - 如果 auto_load_libs 為 true, 那麼程序如果調用到庫函數的話就會直接調用 真正的庫函數 ，如果有的庫函數邏輯比較複雜，可能分析程序就出不來了~~。同時 angr 使用 python 實現了很多的庫函數（保存在 angr.SIM_PROCEDURES 裡面） ，默認情況下會使用列表內部的函數來替換實際的函數調用，如果不在列表內才會進入到真正的 library.如果 auto_load_libs 為 false ， 程序調用函數時，會直接返回一個 不受約束的符號值。

reference : https://kknews.cc/zh-tw/code/pkm5jne.html

CFG



0x4005c5 (0x4005c5) main
0x004005c5: push rbp
0x004005c6: mov rbp, rsp
0x004005c9: sub rsp, 0x10
0x004005cd: mov dword ptr [rbp - 4], edi
0x004005d0: mov qword ptr [rbp - 0x10], rsi
0x004005d4: cmp dword ptr [rbp - 4], 2
0x004005d8: je 0x4005eb
0x4005da (0x4005c5) main+0x15
0x004005da: mov edi, 0x4006c8
0x004005df: call 0x4003f0
0x4005eb (0x4005c5) main+0x26
0x004005eb: mov rax, qword ptr [rbp - 0x10]
0x004005ef: add rax, 8
0x004005f3: mov rax, qword ptr [rax]
0x004005f6: mov rdi, rax
0x004005f9: call 0x400520
0x4005fe (0x4005c5) main+0x39
0x004005fe: test eax, eax
0x00400600: je 0x40060e
0x400602 (0x4005c5) main+0x3d
0x00400602: mov edi, 0x4006f0
0x00400607: call 0x4003f0
0x40060e (0x4005c5) main+0x49
0x0040060e: mov edi, 0x400718
0x00400613: call 0x4003f0
0x4005e4 (0x4005c5) main+0x1f
0x004005e4: mov eax, 0xffffffff
0x004005e9: jmp 0x40061d
0x40060c (0x4005c5) main+0x47
0x0040060c: jmp 0x400618
0x400618 (0x4005c5) main+0x53
0x00400618: mov eax, 0
0x40061d (0x4005c5) main+0x58
0x0040061d: leave
0x0040061e: ret

Create symbolic object

- angr's solver engine is called Claripy
  - `import claripy`
- BV : bitvector
  - `claripy.BVS('x', 32)`
- FP : floating-point
  - `claripy.FPS('y', claripy.fp.FSORT_DOUBLE)`
- Bool : boolean
  - `claripy.BoolV(True)`

reference : https://docs.angr.io/advanced-topics/claripy

explore

- 在 simulation managers 執行 explore，條件匹配的狀態
  - find => 符合條件
  - avoid => 不符合
- `simgr.explore(find=find_addr, avoid=avoid_addr)`
  - avoid_addr = [0x400c06, 0x400bc7]
  - find_addr = 0x400c10d

Get solution

- 可以去查看 found 是否有產生能執行到的 constraint
  - `found = simgr.found[0]`
- 透過 solver.eval 去拿到符合該解的值
  - `found.solver.eval(sym_arg, cast_to=bytes)`

Sample code

Create symbolic object

```
sym_arg_size = 15 #Length in Bytes because we will multiply with 8 later
sym_arg = claripy.BVS('sym_arg', 8*sym_arg_size)
```

Create a state with a symbolic argument

```
argv = [proj.filename]
argv.append(sym_arg)
state = proj.factory.entry_state(args=argv)
```

Generate a simulation manager object

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

Symbolically execute until we find a state satisfying our `find=` and `avoid=` parameters

```
avoid_addr = [0x400c06, 0x400bc7]
find_addr = 0x400c10d
simgr.explore(find=find_addr, avoid=avoid_addr)
```

```
found = simgr.found[0] # A state that reached the find condition from explore
found.solver.eval(sym_arg, cast_to=bytes) # Return a concrete string value for the sym arg to reach this state
```

reference : https://github.com/angr/angr-doc/blob/master/CHEATSHEET.md

Lab

Lab 8

- 我們提供一個 linux binary，請透過 angr 產生出他的 cfg
- 請找到能夠讓該程式印出 correct\n 的輸入
- 繳交方式：學號.zip (zip內包含以下檔案)
    - target.cfg : 用來找位置的 cfg 圖片
    - solve.py : 執行 angr 的 python (please use python3)
    - flag.txt : 正確的參數輸入 (argv[1])
- Note :
    - 此 Lab 有開 pie
        - angr 會自動把 pie base 放到 0x400000
        - 照著投影片第16頁步驟所產出來的cfg內地址已經是rebase過的

Reference

Reference

- https://docs.angr.io/
- https://github.com/angr/angr
- https://github.com/angr/angr-doc
- https://github.com/angr/angr-doc/blob/master/CHEATSHEET.md
- https://github.com/axt/angr-utils