

COMP4901O Competitive Programming in Cybersecurity II

SOMP1010: angr Management

Crack binaries with righteous indignation!

TrebledJ

May 6, 2022



>FIREBIRD CTF
TEAM



Preamble

Code of Ethics

- ▶ The exercises for the course should be attempted ONLY INSIDE THE SECLUDED LAB ENVIRONMENT documented or provided. Please note that most of the attacks described in the slides would be ILLEGAL if attempted on machines that you do not have explicit permission to test and attack. The university, course lecturer, lab instructors, teaching assistants and the Firebird CTF team assume no responsibility for any actions performed outside the secluded lab.
- ▶ Do not intentionally disrupt other students who are working on the challenges or disclose private information you found on the challenge server (e.g. IP address of other students).



Preamble

Code of Ethics

- ▶ The exercises for the course should be attempted ONLY INSIDE THE SECLUDED LAB ENVIRONMENT documented or provided. Please note that most of the attacks described in the slides would be ILLEGAL if attempted on machines that you do not have explicit permission to test and attack. The university, course lecturer, lab instructors, teaching assistants and the Firebird CTF team assume no responsibility for any actions performed outside the secluded lab.
- ▶ Do not intentionally disrupt other students who are working on the challenges or disclose private information you found on the challenge server (e.g. IP address of other students).
- ▶ When solving CTF challenges or hunting angr documentation for some obscure class, it is imperative that you remain calm. Do not destroy the challenge server or make stupid decisions out of frustration or angr!



Preamble

Today's Talk

- ▶ Reverse!





Preamble

Today's Talk

- ▶ Reverse!
- ▶ Less focus on angr's API
- ▶ More focus on the big picture
- ▶ More memes





Preamble

Today's Talk

- ▶ Reverse!
- ▶ Less focus on angr's API
- ▶ More focus on the big picture
- ▶ More memes
- ▶ Unrelated to the [GUI tool](#)





How to Manage Your angr

① Introduction

② Back to the Basics

Early Stages of Reverse-Engineering Grief and Trauma

③ Training Your angr

Wild PRIMEAPE used RAGE!

④ Analysing with angr – CFGs

Investigating the Behavioural Economics of Machine Code

⑤ Debugging angr Programs

Towards Enhanced Emotional Resilience and Cognizance



Introduction

Introduction



Introduction

What is this angr?

Highly modularised and customisable framework for...

- ▶ Symbolic execution
- ▶ Binary analysis (control flow, value set, dependencies)
- ▶ Reversing binaries with righteous indignation





Introduction

What is this angr?

Highly modularised and customisable framework for...

- ▶ Symbolic execution
- ▶ Binary analysis (control flow, value set, dependencies)
- ▶ Reversing binaries with righteous indignation

Repo: <https://github.com/angr/angr>

Docs (Book-like): <https://docs.angr.io/>

API: <https://api.angr.io/>

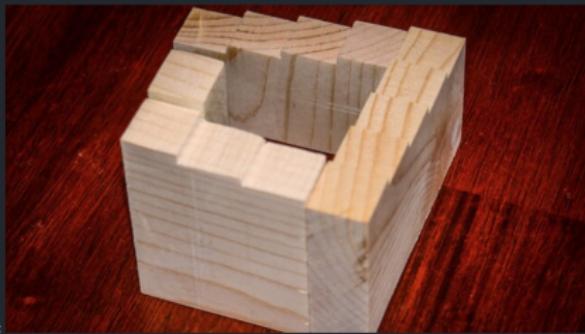




Motivation

逐步拾回自己 / Step by Step

Summary



- Thumbnail:
- Song: <https://www.youtube.com/watch?v=-DZd0CF13nI>
- Author: cire_meat_pop
- Categories: Reverse, Misc, ★☆☆☆☆
- Points: 100
- Solves: 68/234 (Secondary: 19/103, Tertiary: 25/65, Open: 21/7)

Description

Retrieve the flag step by step.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	M	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31	A32	A33	A34	A35	A36	A37	A38	A39	A40	A41	A42	A43	A44	A45	A46	A47	A48	A49	A50	A51	A52	A53	A54	A55	A56	A57	A58	A59	A60	A61	A62	A63	A64	A65	A66	A67	A68	A69	A70	A71	A72	A73	A74	A75	A76	A77	A78	A79	A80	A81	A82	A83	A84	A85	A86	A87	A88	A89	A89	A90	A91	A92	A93	A94	A95	A96	A97	A98	A99	A100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	69	70	71	72	73	74	75	76	77	78	79	79	80	81	82	83	84	85	86	87	88	89	89	90	91	92	93	94	95	96	97	98	99	99	100																	



Motivation

18:06 **Dickson** I am quite wondered how people solve step by step 😂

step-by-step 18 Messages >

There are no recent messages in this thread.

Dickson we solved it by human lol 😂

Dickson (wasted like 19 hours on it)

→ @ERASER Click to see attachment

19:49 **Sierrrrr** Oh I was an idiot. I click it cell by cell

18:26 **bread** solved . . . by hand (edited)

18:12 **ERASER** human and logic lol

used 2 hours



[Back to the Basics](#)

Back to the Basics

Early Stages of Reverse-Engineering Grief and Trauma



Symbolic Execution – General Idea

Code

```
void hello(uint8_t c) {  
    uint8_t win;  
    uint32_t a = c;  
    a = a*2 + 0xdeadbeef;  
    if (a >= 0 && a < 0xbaad)  
        win = 1;  
    else  
        win = 0;  
}
```

Concrete

```
c = 0xf0  
a = 0x000000f0  
a = 0xdeadc0cf  
win = 0
```



Symbolic Execution – General Idea

Code

```
void hello(uint8_t c) {  
    uint8_t win;  
    uint32_t a = c;  
    a = a*2 + 0xdeadbeef;  
    if (a >= 0 && a < 0xbaad)  
        win = 1;  
    else  
        win = 0;  
}
```

Concrete

```
c = 0xf0  
a = 0x000000f0  
a = 0xdeadc0cf  
win = 0
```

Symbolic

```
c is an 8-bit input var  
a_0 = ZeroExt(24, c)  
a = a_0 * 2 + 0xdeadbeef  
win = If(And(UGE(a, 0),  
            ULT(a, 0xbaad)),  
            1,  
            0)
```



angr Level 1 – Constrain your angr!

```
import angr
from claripy import *

p = angr.Project('./level1')
flag_bvs = [BVS(f'f_{i}', 8) for i in range(32)]
state = p.factory.entry_state(
    stdin=Concat(*flag_bvs, BVV('\n', 8)))

for bv, ch in zip(flag_bvs, b'flag{'):
    state.solver.add(bv == BVV(ch, 8))

for bv in flag_bvs:
    state.solver.add(bv >= BVV(0x20, 8))
    state.solver.add(bv <= BVV(0x7f, 8))

simgr = p.factory.simgr(state)
simgr.explore(find=0x40170f)
print(simgr.found[0].posix.dumps(0))
```



angr Level 1 – Constrain your angr!

```
import angr
from claripy import *

p = angr.Project('./level1')
flag_bvs = [BVS(f'f_{i}', 8) for i in range(32)]
state = p.factory.entry_state(
    stdin=Concat(*flag_bvs, BVV('\n', 8)))

for bv, ch in zip(flag_bvs, b'flag{'):
    state.solver.add(bv == BVV(ch, 8))

for bv in flag_bvs:
    state.solver.add(bv >= BVV(0x20, 8))
    state.solver.add(bv <= BVV(0x7f, 8))

simgr = p.factory.simgr(state)
simgr.explore(find=0x40170f)
print(simgr.found[0].posix.dumps(0))
```



angr Level 1 – Constrain your angr!

```
import angr
from claripy import *

p = angr.Project('./level1')
flag_bvs = [BVS(f'f_{i}', 8) for i in range(32)]
state = p.factory.entry_state(
    stdin=Concat(*flag_bvs, BVV('\n', 8)))

for bv, ch in zip(flag_bvs, b'flag{'):
    state.solver.add(bv == BVV(ch, 8))

for bv in flag_bvs:
    state.solver.add(bv >= BVV(0x20, 8))
    state.solver.add(bv <= BVV(0x7f, 8))

simgr = p.factory.simgr(state)
simgr.explore(find=0x40170f)
print(simgr.found[0].posix.dumps(0))
```



angr Level 1 – Constrain your angr!

```
import angr
from claripy import *

p = angr.Project('./level1')
flag_bvs = [BVS(f'f_{i}', 8) for i in range(32)]
state = p.factory.entry_state(
    stdin=Concat(*flag_bvs, BVV('\n', 8)))

for bv, ch in zip(flag_bvs, b'flag{'):
    state.solver.add(bv == BVV(ch, 8))

for bv in flag_bvs:
    state.solver.add(bv >= BVV(0x20, 8))
    state.solver.add(bv <= BVV(0x7f, 8))

simgr = p.factory.simgr(state)
simgr.explore(find=0x40170f)
print(simgr.found[0].posix.dumps(0))
```



angr Level 1 – Constrain your angr!

```
import angr
from claripy import *

p = angr.Project('./level1')
flag_bvs = [BVS(f'f_{i}', 8) for i in range(32)]
state = p.factory.entry_state(
    stdin=Concat(*flag_bvs, BVV('\n', 8)))

for bv, ch in zip(flag_bvs, b'flag{'):
    state.solver.add(bv == BVV(ch, 8))

for bv in flag_bvs:
    state.solver.add(bv >= BVV(0x20, 8))
    state.solver.add(bv <= BVV(0x7f, 8))

simgr = p.factory.simgr(state)
simgr.explore(find=0x40170f)
print(simgr.found[0].posix.dumps(0))
```



angr Level 1 – Constrain your angr!

```
import angr
from claripy import *

p = angr.Project('./level1')
flag_bvs = [BVS(f'f_{i}', 8) for i in range(32)]
state = p.factory.entry_state(
    stdin=Concat(*flag_bvs, BVV('\n', 8)))

for bv, ch in zip(flag_bvs, b'flag{'):
    state.solver.add(bv == BVV(ch, 8))

for bv in flag_bvs:
    state.solver.add(bv >= BVV(0x20, 8))
    state.solver.add(bv <= BVV(0x7f, 8))

simgr = p.factory.simgr(state)
simgr.explore(find=0x40170f)
print(simgr.found[0].posix.dumps(0))
```



angr Level 1 – Constrain your angr!

```
import angr
from claripy import *

p = angr.Project('./level1')
flag_bvs = [BVS(f'f_{i}', 8) for i in range(32)]
state = p.factory.entry_state(
    stdin=Concat(*flag_bvs, BVV('\n', 8)))

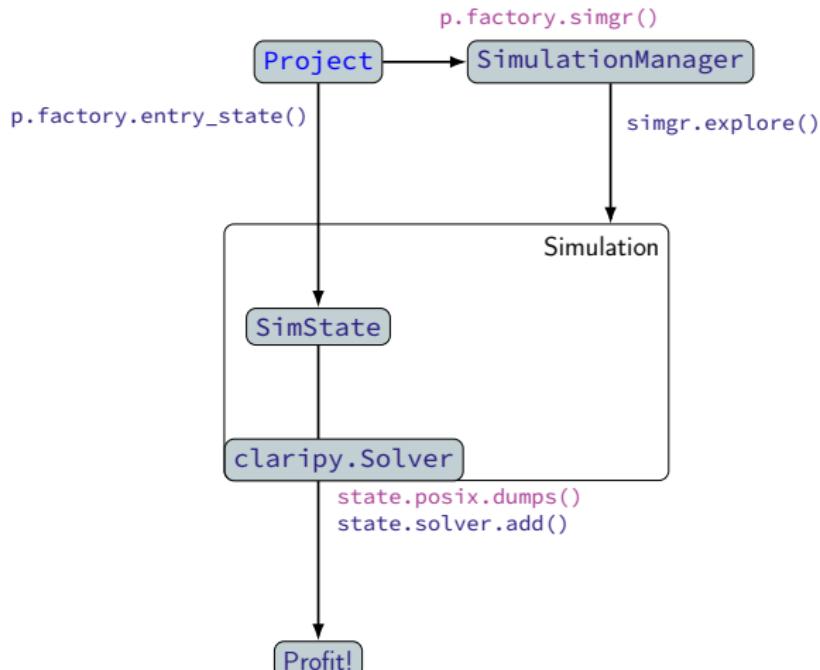
for bv, ch in zip(flag_bvs, b'flag{'):
    state.solver.add(bv == BVV(ch, 8))

for bv in flag_bvs:
    state.solver.add(bv >= BVV(0x20, 8))
    state.solver.add(bv <= BVV(0x7f, 8))

simgr = p.factory.simgr(state)
simgr.explore(find=0x40170f)
print(simgr.found[0].posix.dumps(0))
```

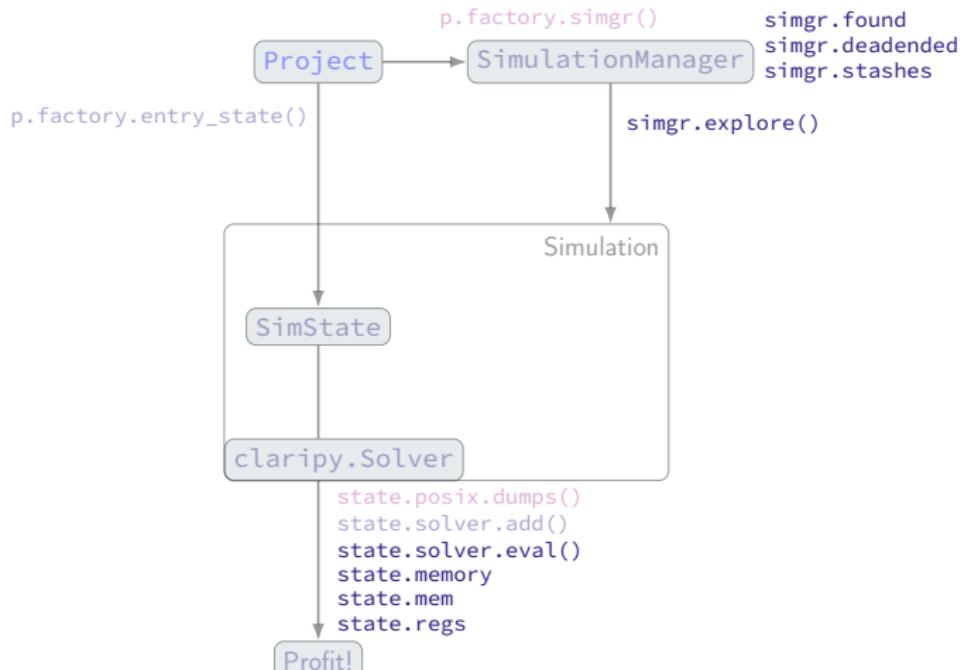


angr Level 1 – Constrain your angr!





What's next?





Training Your angr

Training Your angr

Wild PRIMEAPE used RAGE!





Some Terminology



- ▶ bitvector (BV): a sequence of bits
- ▶ symbol: unknown data (Schrödinger's cat)
- ▶ concrete: known data (Schrödinger's cat outside the box)
- ▶ concretise: process of [symbol → concrete] w.r.t. constraints
(opening Schrödinger's box)



Some Terminology

- ▶ bitvector (BV): a sequence of bits
- ▶ symbol: unknown data (Schrödinger's cat)
- ▶ concrete: known data (Schrödinger's cat outside the box)
- ▶ concretise: process of [symbol → concrete] w.r.t. constraints
(opening Schrödinger's box)
- ▶ mixin: a design pattern promoting customisations (plugins)



Training Your ang

Concepts – SimState

imgflip.com



SimState Concretisation

`state.solver.eval(e, cast_to=None)`

Evaluates (concretises) a bitvector and optionally casts to the type specified by `cast_to` (`int` or `bytes`).



`state.posix.dumps(fd)`

Concretises and dumps stdin (0), stdout (1), or stderr (2) as bytes.



A Tale of Two SimState Memory Lanes



DefaultMemory (Mixin Galore)

```
state.memory.store(addr: int, data: int | bytes | BV)
state.memory.load(addr: int, size: int) → BV
```

SimMemView

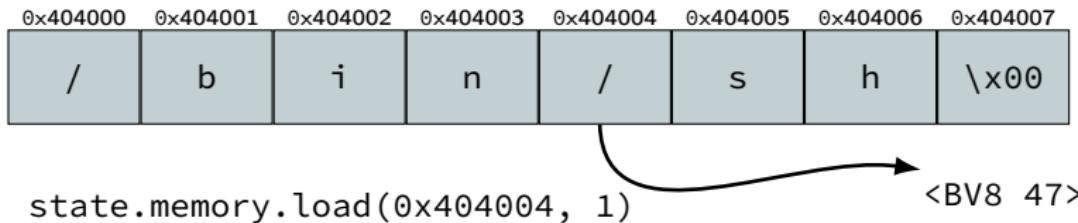
```
state.mem[<addr>].<type>.<resolved | concrete>
```



state.memory

```
state.memory.store(0x404000, b'/bin/sh\x00')
```

```
bv = state.memory.load(0x404004, 4)
state.solver.eval(bv, cast_to=bytes)
→ b'/sh\x00'
```



Byte order depends on endianness! Check `state.arch`!

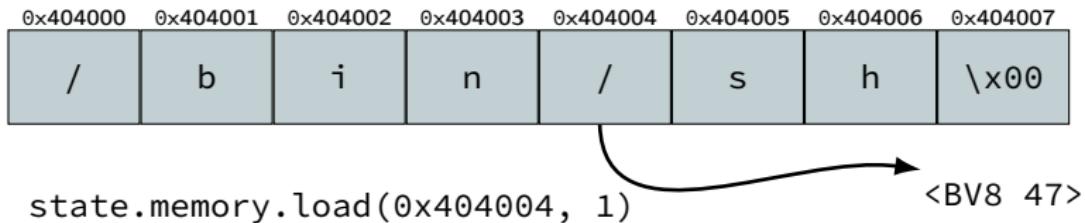
More: <https://docs.angr.io/core-concepts/states#low-level-interface-for-memory>.



state.memory

```
state.memory.store(0x404000, b'/bin/sh\x00')
```

```
bv = state.memory.load(0x404004, 4)
state.solver.eval(bv, cast_to=bytes)
→ b'/sh\x00'
```



Byte order depends on endianness! Check `state.arch`!

More: <https://docs.angr.io/core-concepts/states#low-level-interface-for-memory>.



state.mem



```
state.mem[0x404000].uint32_t = 0x41424344
```

```
state.mem[0x404000].uint16_t.resolved  
→ <BV16 0x4344>
```

```
state.mem[0x404000].uint16_t.concrete  
→ 0x4344
```

0x404000	0x404001	0x404002	0x404003
0x44	0x43	0x42	0x41

(Little Endian)



state.mem



```
state.mem[0x404000].uint32_t = 0x41424344
```

```
state.mem[0x404000].uint16_t.resolved  
→ <BV16 0x4344>
```

```
state.mem[0x404000].uint16_t.concrete  
→ 0x4344
```

0x404000	0x404001	0x404002	0x404003
0x44	0x43	0x42	0x41

(Little Endian)



state.mem

```
state.mem[0x404000].uint32_t = 0x41424344
```

```
state.mem[0x404000].uint16_t.resolved  
→ <BV16 0x4344>
```

```
state.mem[0x404000].uint16_t.concrete  
→ 0x4344
```

0x404000	0x404001	0x404002	0x404003
0x44	0x43	0x42	0x41

(Little Endian)



state.mem

```
>>> state.mem[0x404000]
<<untyped> <unresolvable> at 0x404000>
>>> _.
_.CharT           _.init_state(      _.resolved          _.types
_.STRONGREF_STATE  _.int           _.set_state(       _.uint16_t
_.array(          _.int16_t        _.set_strongref_state(  _.uint32_t
_.basic_string     _.int32_t        _.short            _.uint64_t
_.bool             _.int64_t        _.signed           _.uint8_t
_.byte             _.int8_t         _.size_t          _.uintptr_t
_.char             _.long          _.ssize            _.unsigned
_.concrete         _.memo(         _.state            _.va_list
_.copy(           _.merge(        _.store(          _.void
_.deref            _.ptrdiff_t     _.string           _.widen(
_.double           _.qword         _.struct          _.word
_.dword            _.register_default(  _.time_t
_.float            _.resolvable
```



state.regs

state.regs.pc

→ <BV64 0x4011c8>

```
>>> state.regs.  
Display all 222 possibilities? (y or n)  
state.regs.STRONGREF_STATE      state.regs.fc3210          state.regs.r13w          state.regs.tag2  
state.regs.ac                  state.regs.flags          state.regs.r14             state.regs.tag3  
state.regs.acflag              state.regs.fpreg          state.regs.r14b            state.regs.tag4  
state.regs.ah                  state.regs.fpround        state.regs.r14d            state.regs.tag5  
state.regs.al                  state.regs.fpntag         state.regs.r14w            state.regs.tag6  
state.regs.ax                  state.regs.fpu_regs       state.regs.r15             state.regs.tag7  
state.regs.bh                  state.regs.fpu_tags       state.regs.r15b            state.regs.widen(  
state.regs.bl                  state.regs.fs              state.regs.r15d            state.regs.xmm0  
state.regs.bp                  state.regs.fs_const       state.regs.r15w            state.regs.xmm0hq  
state.regs.bph                 state.regs.ftop           state.regs.r8              state.regs.xmm0lq  
state.regs.bpl                 state.regs.get(          state.regs.r8b             state.regs.xmm1  
state.regs.bx                  state.regs.gs              state.regs.r8d             state.regs.xmm10  
state.regs.cc_dep1              state.regs.gs_const       state.regs.r8w             state.regs.xmm10hq  
state.regs.cc_dep2              state.regs.id              state.regs.r9              state.regs.xmm10lq  
state.regs.cc_ndep              state.regs.idflag         state.regs.r9b             state.regs.xmm11  
state.regs.cc_op                state.regs.init_state(   state.regs.r9d             state.regs.xmm11hq  
state.regs.ch                  state.regs.ip              state.regs.r9w             state.regs.xmm11lq  
state.regs.cl                  state.regs.ip_at_syscall state.regs.rax             state.regs.xmm12  
state.regs.cmlen                state.regs.memo(          state.regs.rbp             state.regs.xmm12hq  
state.regs.cmstart              state.regs.merge(         state.regs.rbx             state.regs.xmm12lq  
state.regs.copy(                state.regs.mm0             state.regs.rcx             state.regs.xmm13  
state.regs.cr0                 state.regs.mm1             state.regs.rdi             state.regs.xmm13hq  
state.regs.cr2                 state.regs.mm2             state.regs.rdx             state.regs.xmm13lq  
state.regs.cr3                 state.regs.mm3             state.regs.register_default( state.regs.xmm14
```



simgr.explore(find, avoid, n, num_find, ...)

Simulate until constraints are met (or until no more states could be stepped).



simgr.explore(find, avoid, n, num_find, ...)

Simulate until constraints are met (or until no more states could be stepped).

- ▶ **find:** int | [int] | (SimState → bool) = None
- ▶ **avoid:** int | [int] | (SimState → bool) = None
 - ▶ An address, a list of addresses, or a predicate which takes a state and returns whether it should be marked found/avoided (**return True**) or continue to be active (**return False**).



simgr.explore(find, avoid, n, num_find, ...)

Simulate until constraints are met (or until no more states could be stepped).

- ▶ **find:** `int | [int] | (SimState → bool) = None`
- ▶ **avoid:** `int | [int] | (SimState → bool) = None`
 - ▶ An address, a list of addresses, or a predicate which takes a state and returns whether it should be marked found/avoided (`return True`) or continue to be active (`return False`).
- ▶ **n:** `int = None`
 - ▶ Number of steps to advance. If `None`, run until other conditions are met.
- ▶ **num_find:** `int = 1`
 - ▶ Number of states to look for in the ‘`found`’ stash.



simgr.stashes[...]

A horizontal progress bar consisting of a teal segment followed by an orange segment.

Where are the simulated states?

Note: `simgr.stashes['active'] == simgr.active`.

More: <https://docs.angr.io/core-concepts/pathgroups#stash-types>.



simgr.stashes[...]

Where are the simulated states?

- ▶ ‘active’: States here are alive and well.
- ▶ ‘found’: States here match search criteria and constraints.

Note: `simgr.stashes['active'] == simgr.active`.

More: <https://docs.angr.io/core-concepts/pathgroups#stash-types>.



simgr.stashes[...]

Where are the simulated states?

- ▶ ‘**active**’: States here are alive and well.
- ▶ ‘**found**’: States here match search criteria and constraints.
- ▶ ‘**deadended**’: States finished executing or can’t continue.
 - ▶ Possible causes: **exit(n)**, no more sat states, program crashed (invalid instruction, memory access, etc.).
 - ▶ “Why did my state deadend?” See Slide 66 to see where to start debugging!

Note: `simgr.stashes['active'] == simgr.active`.

More: <https://docs.angr.io/core-concepts/pathgroups#stash-types>.



simgr.stashes[...]

Where are the simulated states?

- ▶ ‘**active**’: States here are alive and well.
- ▶ ‘**found**’: States here match search criteria and constraints.
- ▶ ‘**deadended**’: States finished executing or can’t continue.
 - ▶ Possible causes: `exit(n)`, no more sat states, program crashed (invalid instruction, memory access, etc.).
 - ▶ “Why did my state deadend?” See Slide 66 to see where to start debugging!
- ▶ ‘**errored**’: States that terminated due to a Python exception (simulation error, claripy error, etc.).

Note: `simgr.stashes['active'] == simgr.active`.

More: <https://docs.angr.io/core-concepts/pathgroups#stash-types>.



SOMP1010: angr Management – Midterm





Q1 – Address?

```
if (DAT_001041c9 != '\0') {
    /* Interesting loop... */
    lVar3 = 0;
    do {
        (&DAT_00104180)[lVar3] =
            (&DAT_00104180)[lVar3] ^ (byte)(0x750729cbe569d1dd >> (((byte)lVar3 & 7) << 3));
        lVar3 += 1;
    } while (lVar3 != 0x49);
    DAT_001041c9 = '\0';
}
```



Q1 – Address?

```
if (DAT_001041c9 != '\0') {
    /* Interesting loop... */
    lVar3 = 0;
    do {
        (&DAT_00104180)[lVar3] =
            (&DAT_00104180)[lVar3] ^ (byte)(0x750729cbe569d1dd >> (((byte)lVar3 & 7) << 3));
        lVar3 += 1;
    } while (lVar3 != 0x49);
    DAT_001041c9 = '\0';
}
```

What is the address of the secret?

- (A) 0x104180
- (B) 0x1041c9
- (C) 0x49
- (D) 0x750729cbe569d1dd



Q1 – Address?

```
if (DAT_001041c9 != '\0') {
    /* Interesting loop... */
    lVar3 = 0;
    do {
        (&DAT_00104180)[lVar3] =
            (&DAT_00104180)[lVar3] ^ (byte)(0x750729cbe569d1dd >> (((byte)lVar3 & 7) << 3));
        lVar3 += 1;
    } while (lVar3 != 0x49);
    DAT_001041c9 = '\0';
}
```

What is the address of the secret?

- (A) 0x104180
- (B) 0x1041c9
- (C) 0x49
- (D) 0x750729cbe569d1dd



Q2 – Length?

```
if (DAT_001041c9 != '\0') {
    /* Interesting loop... */
    lVar3 = 0;
    do {
        (&DAT_00104180)[lVar3] =
            (&DAT_00104180)[lVar3] ^ (byte)(0x750729cbe569d1dd >> (((byte)lVar3 & 7) << 3));
        lVar3 += 1;
    } while (lVar3 != 0x49);
    DAT_001041c9 = '\0';
}
```

What is the length of the secret?

- (B) 0x48



Q3 – How to print?

```
simgr = p.factory.simgr()
simgr.explore(find=0x401206)
s = simgr.found[0]
```

Which of the following correctly concretises and prints out the secret (assuming there are no null bytes in the 72 characters)?

- (A) s.mem[0x404180].string.concrete
- (B) s.solver.eval(s.memory.load(0x404180, 72), cast_to=bytes)
- (C) b''.join(s.mem[0x404180+i].char.concrete for i in range(72))

Note: tooling is a PIE binary. By default, ghidra and angr loads PIE binaries at a base address of 0x1000000 and 0x4000000 respectively; so by default, 0x101234 in ghidra == 0x401234 in angr.



Q3 – How to print?

```
simgr = p.factory.simgr()
simgr.explore(find=0x401206)
s = simgr.found[0]
```

Which of the following correctly concretises and prints out the secret (assuming there are no null bytes in the 72 characters)?

- (A) `s.mem[0x404180].string.concrete`
- (B) `s.solver.eval(s.memory.load(0x404180, 72), cast_to=bytes)`
- (C) `b''.join(s.mem[0x404180+i].char.concrete for i in range(72))`

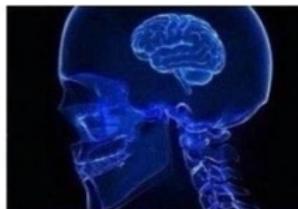
Note: tooling is a PIE binary. By default, ghidra and angr loads PIE binaries at a base address of 0x1000000 and 0x4000000 respectively; so by default, 0x101234 in ghidra == 0x401234 in angr.



Training Your angr

SOMP1010: angr Management – Midterm

```
s.mem[0x404180]
.string
.concrete
```



```
s.solver
.eval(
    s.memory.load(0x404180, 72),
    cast_to=bytes)
```



```
b''.join(
    s.mem[0x404180+i]
    .char
    .concrete
    for i in range(72))
```

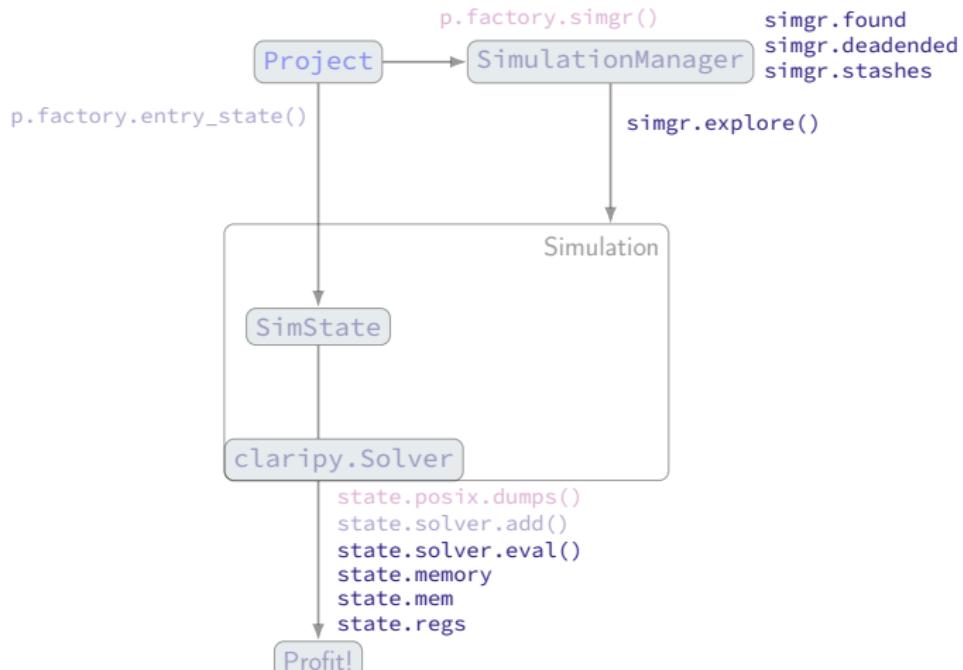


```
state.options |= sim_options.refs
...
b''.join(s.solver.eval(d.data,
cast_to=bytes) for d in [a for
a in s.history.actions if
'mem/write' in repr(a)][-73:-1])
```



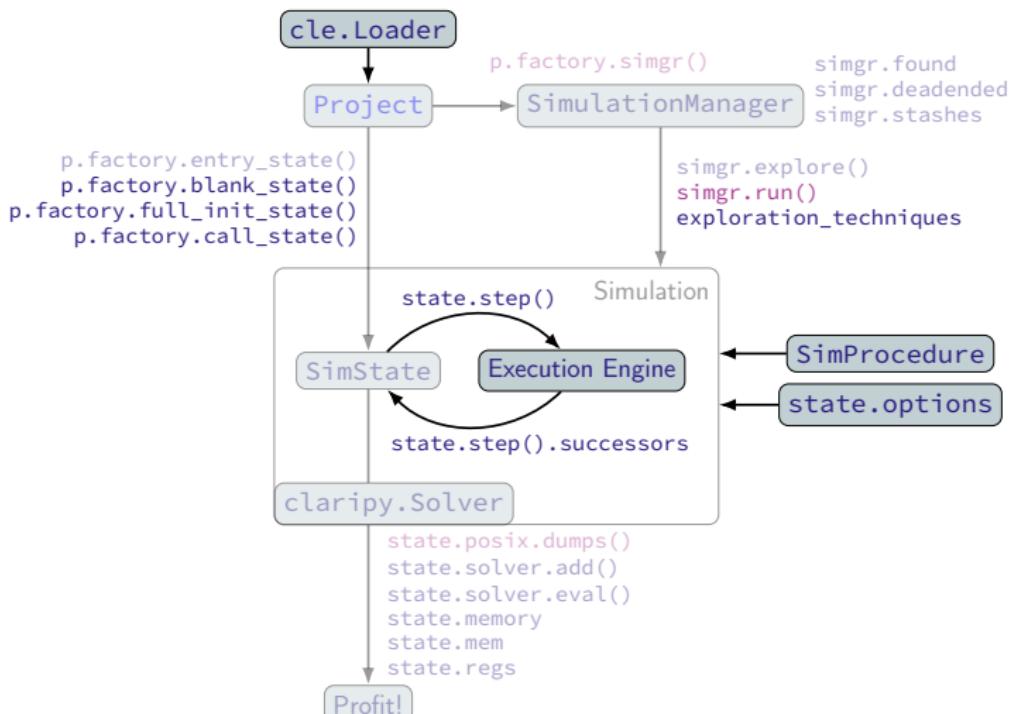


Summary





But wait— there's more!



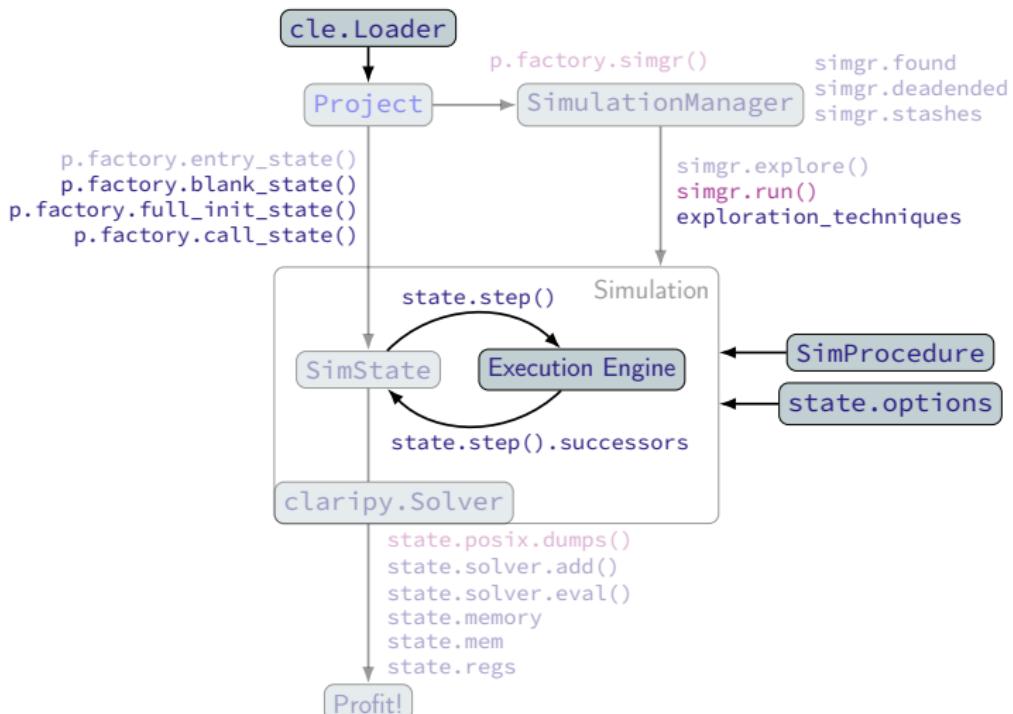


But wait— there's more!

- ▶ cle.Loader: options to load x86, AVR, MIPS, etc. instructions from binary, Intel hex, etc. into an unified interface.
- ▶ State Presets: start simulating from the beginning of an arbitrary function or point in code.
- ▶ exploration_techniques: different ways to explore paths, keep track of states, etc.
- ▶ Hooks and SimProcedures: attach your own emulation code to functions/callbacks.
- ▶ state.options: fine-tune the simulation and solver engines.
- ▶ Execution Engine: customise which mixins are used to process instructions.

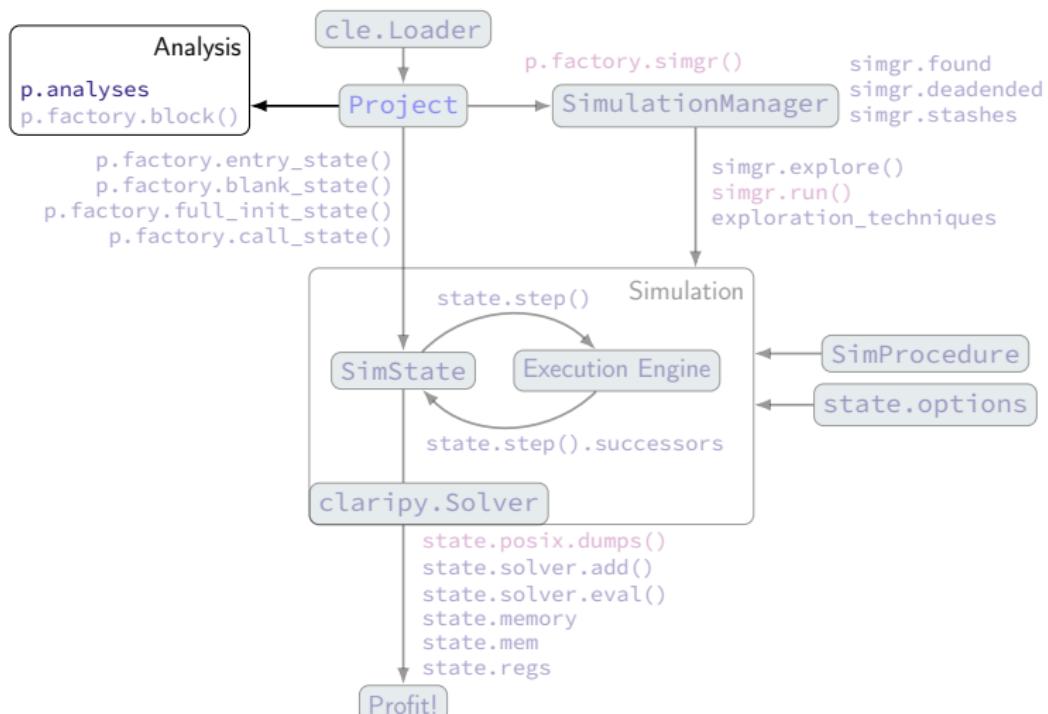


What's next?





What's next?





Training Your angr

Questions?

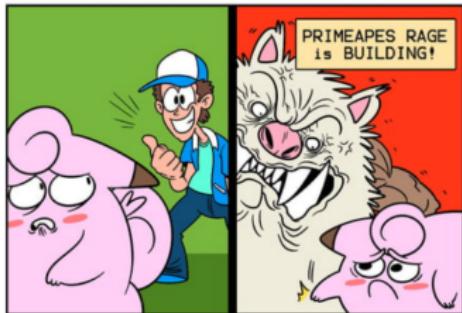




Analysis with angr – CFGs

Analysis with angr – CFGs

Investigating the Behavioural Economics of Machine Code





Analysis with angr – CFGs

An Appetiser

Challenge 12 Solves X

Labyrinth

477

Author: sky

To get the flag you'll need to get to the end of a maz- five randomly generated mazes within five minutes.

This is an automatic reversing challenge. You will be provided an ELF as a hex string. You should analyze it, construct an input to make it terminate with exit(0), and then respond with your input in the same format. You will need to solve five binaries within a five minute timeout.

Challenge is solvable in under 4096 bytes of input and larger inputs may be buffered strangely on the server.

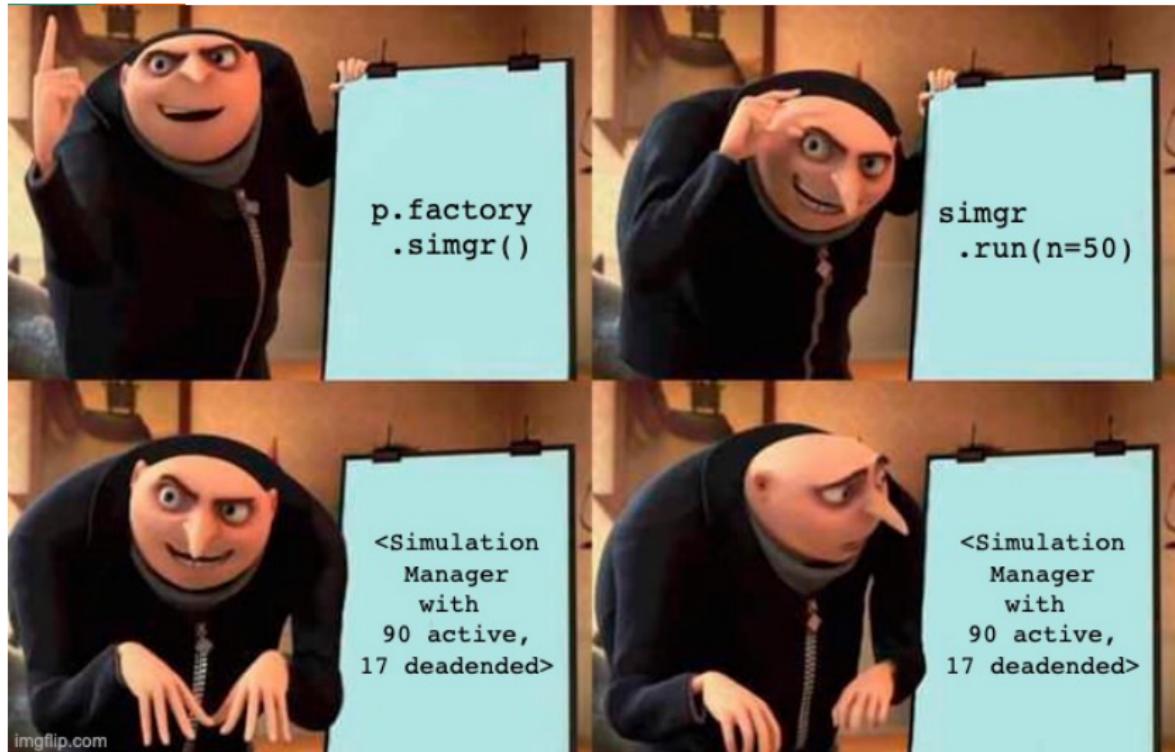
SNI: [labyrinth](#)

[labyrinth.zip](#)



Analysis with angr – CFGs

Gru's Plan





What went wrong?

The screenshot shows the radare2 decompiler interface with the title "Decompile: main - (labyrinth-elf)". The code editor displays the following assembly-like pseudocode:

```
1
2 undefined8 main(void)
3
4 {
5     setvbuf(stdout,NULL,2,0);
6     setvbuf(stdin,NULL,2,0);
7     setvbuf(stderr,NULL,2,0);
8     function_133();
9     return 1;
10}
11
```



What went wrong?

The screenshot shows the radare2 decompiler interface with the title "Decompile: function_133 - (labyrinth-elf)". The assembly code is as follows:

```
1
2 void function_133(void)
3 {
4     uint local_c;
5
6     local_c = 0;
7     __isoc99_scanf(&DAT_00119004,&local_c);
8     local_c ^= 0x5e371ffe;
9     if (local_c == 0x178) {
10         function_376();
11         return;
12     }
13     /* WARNING: Subroutine does not return */
14     exit(1);
15 }
```

The code decompiles a C function named `function_133`. It initializes a local variable `local_c` to 0, reads a value from memory into it using `__isoc99_scanf`, and then performs a bitwise XOR operation with `0x5e371ffe`. If `local_c` equals `0x178`, it calls `function_376` and returns. Otherwise, it exits with a status of 1.



What went wrong?

Decompile: function_376 - (labyrinth-elf)

```
1
2 void function_376(void)
3 {
4     uint local_c;
5
6     local_c = 0;
7     __isoc99_scanf(&DAT_00119004,&local_c);
8     local_c ^= 0xef2bbe35;
9     if (local_c == 0x85) {
10         function_133();
11     }
12     else {
13         if (local_c != 0x156) {
14             /* WARNING: Subroutine does not return */
15             exit(1);
16         }
17         function_342();
18     }
19 }
20 return;
21 }
```



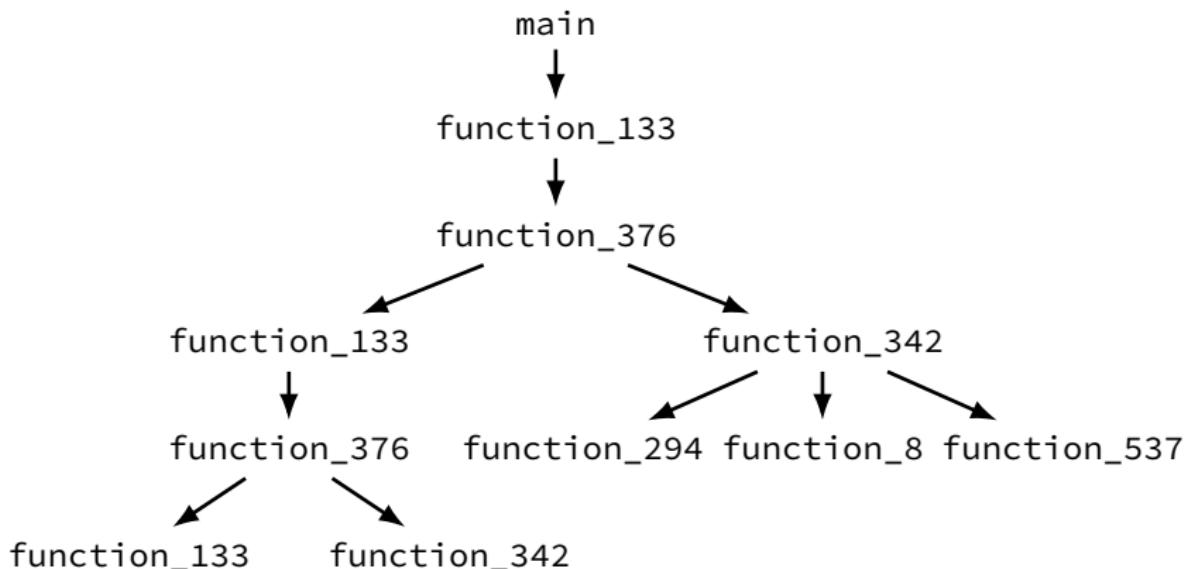
What went wrong?

Decompile: function_376 - (labyrinth-elf)

```
1
2 void function_376(void)
3 {
4     uint local_c;
5
6     local_c = 0;
7     __isoc99_scanf(&DAT_00119004,&local_c);
8     local_c ^= 0xef2bbe35;
9     if (local_c == 0x85) {
10         function_133(); ← Repeats!
11     }
12     else {
13         if (local_c != 0x156) {
14             /* WARNING: Subroutine does not return */
15             exit(1);
16         }
17         function_342();
18     }
19     return;
20 }
```



Path Explosion





Analysis with angr – CFGs What went wrong?

Path Explosion



imgflip.com



Path Explosion



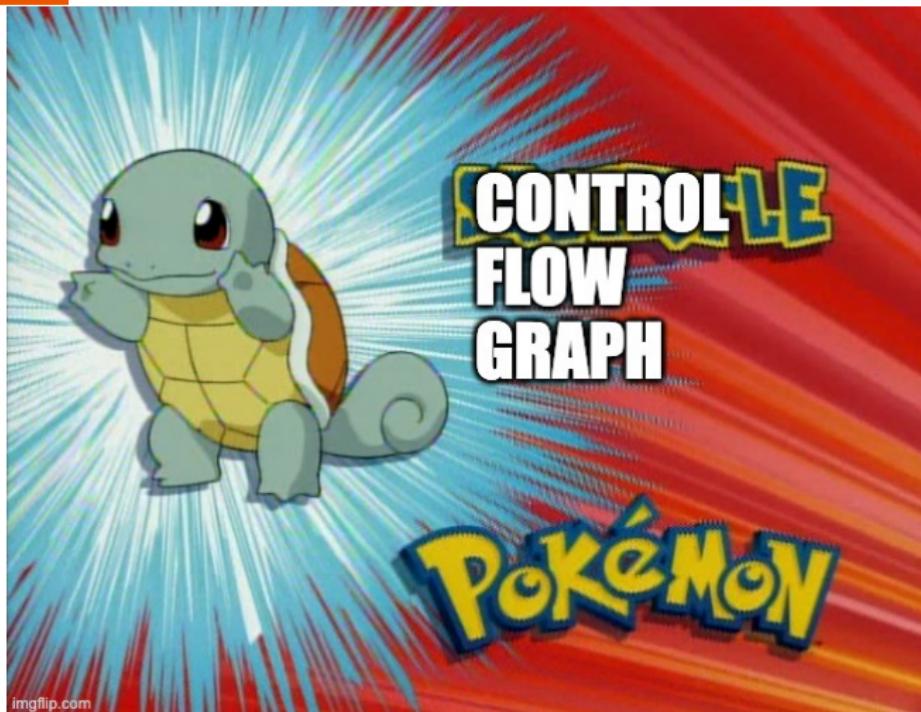


Path Explosion





Control Flow Graphs to the Rescue



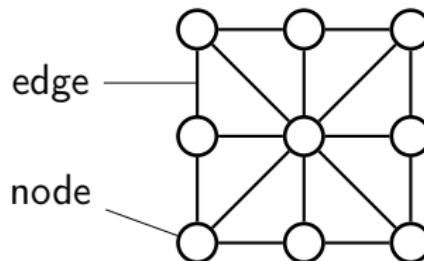


G-raph?



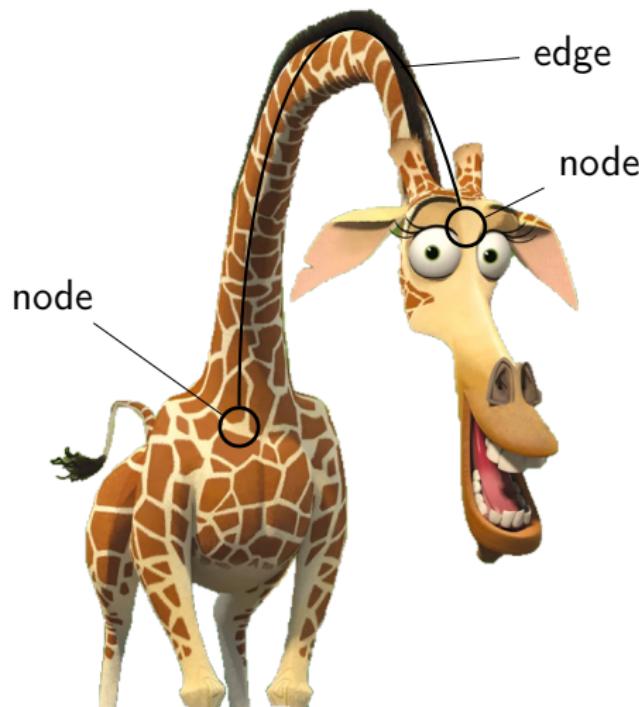


Graphs



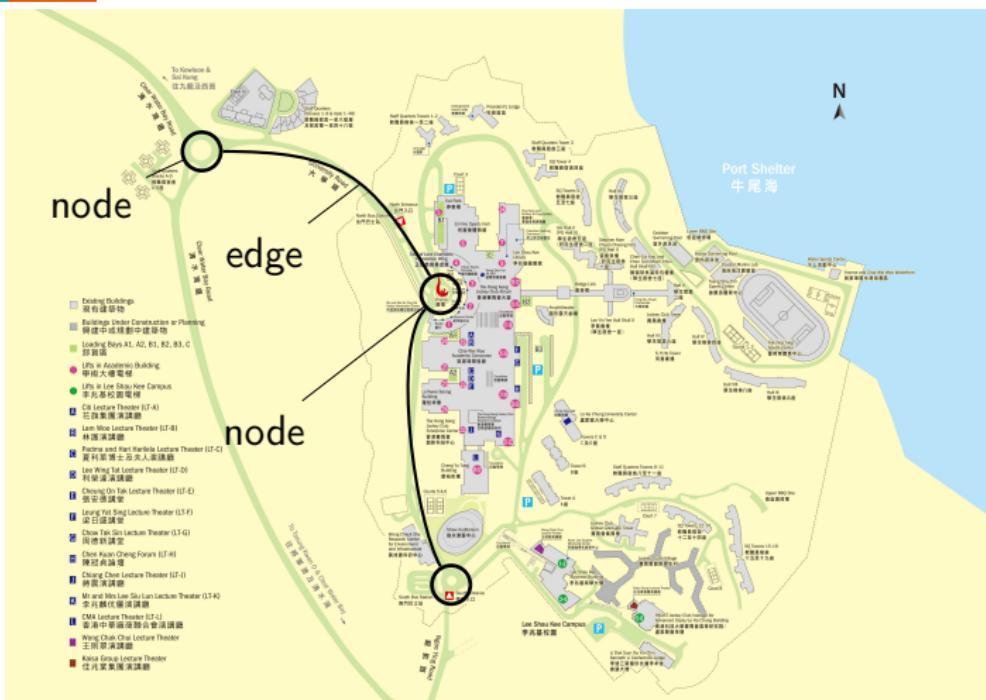


Graph!



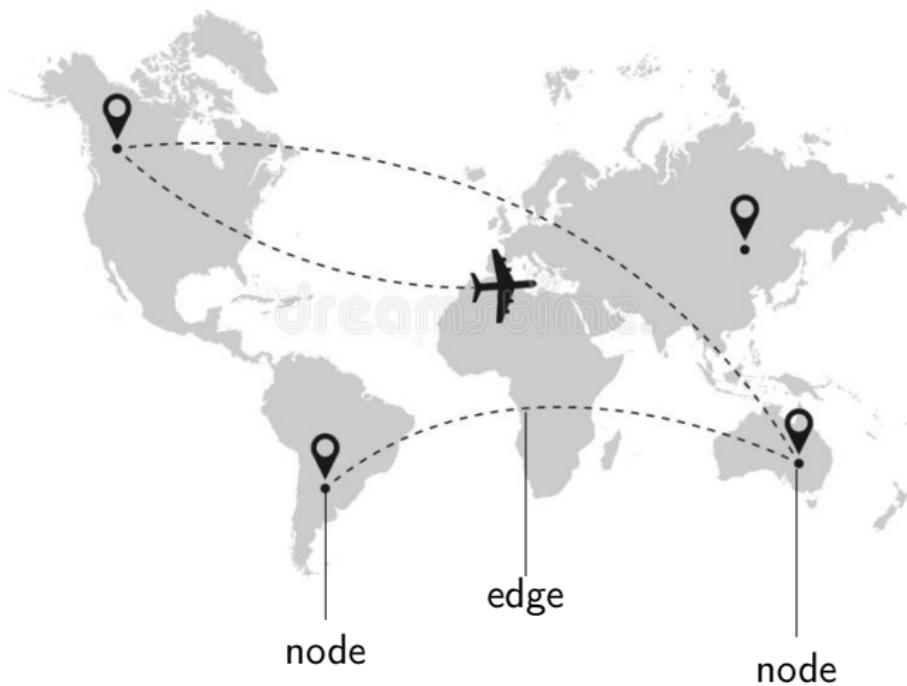


Graphs



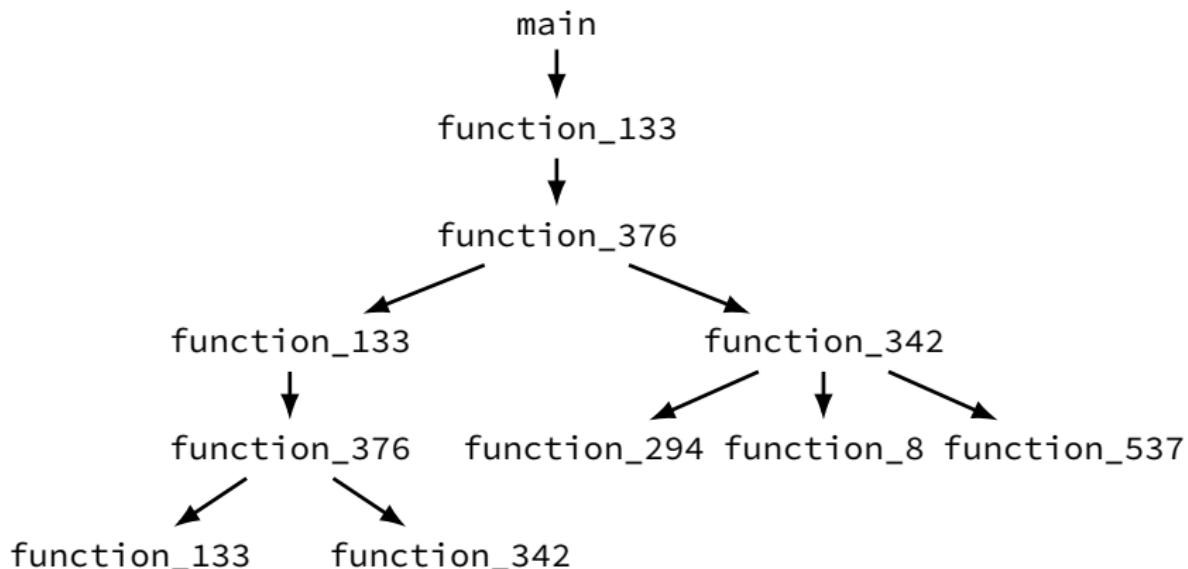


Graphs





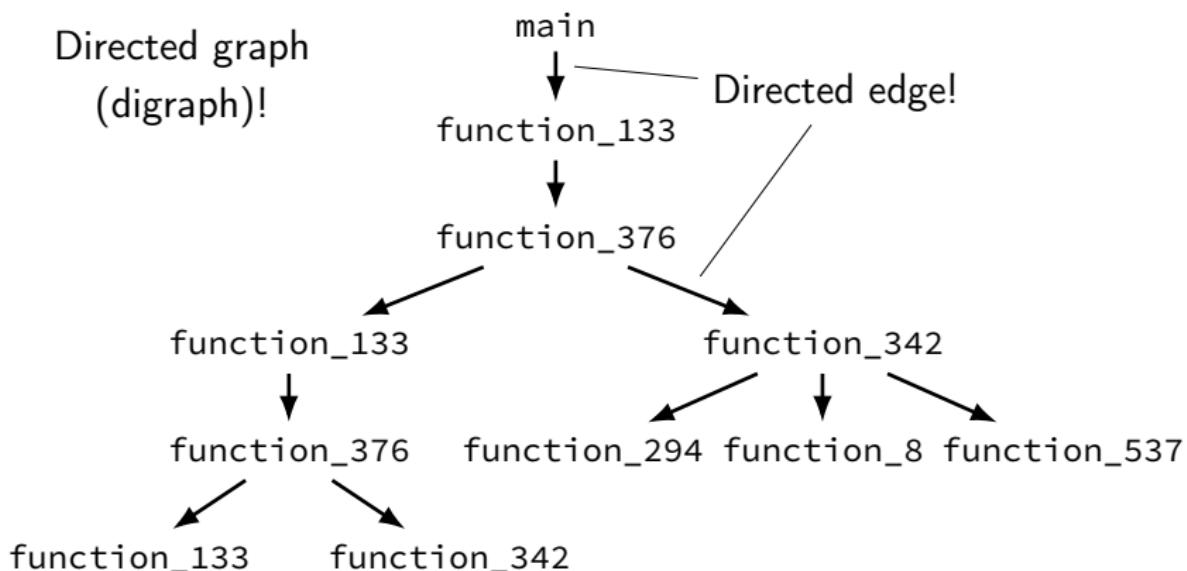
Programs as Graphs





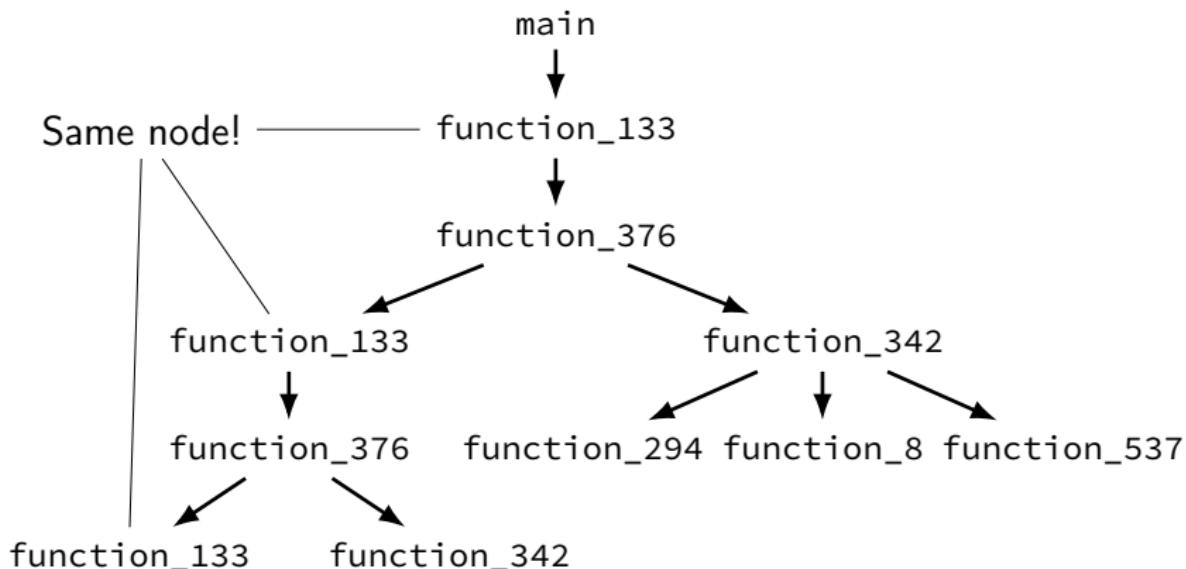
Programs as Graphs

Directed graph
(digraph)!



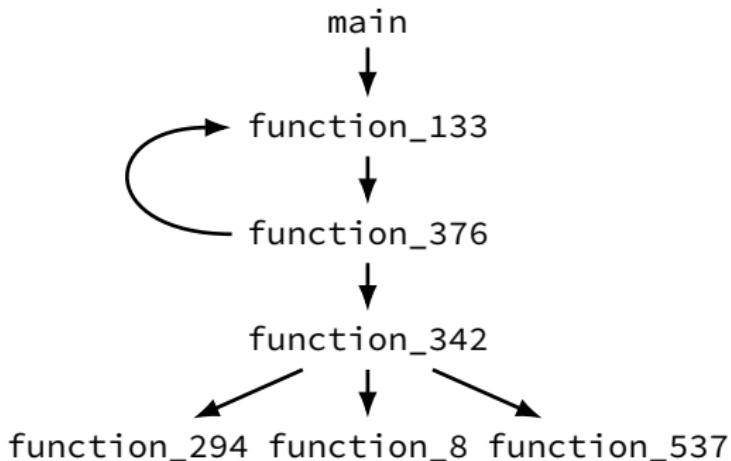


Programs as Graphs



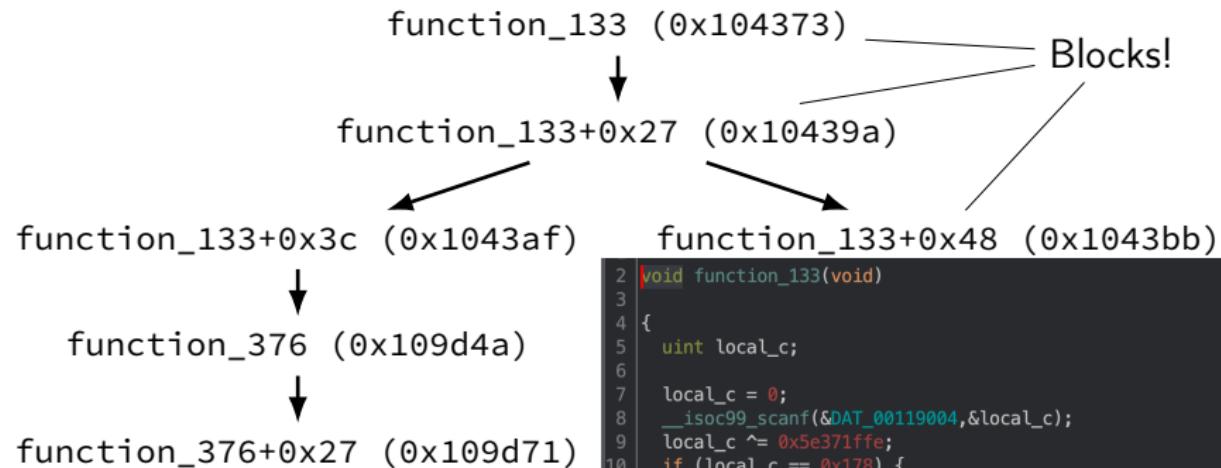


Programs as Graphs



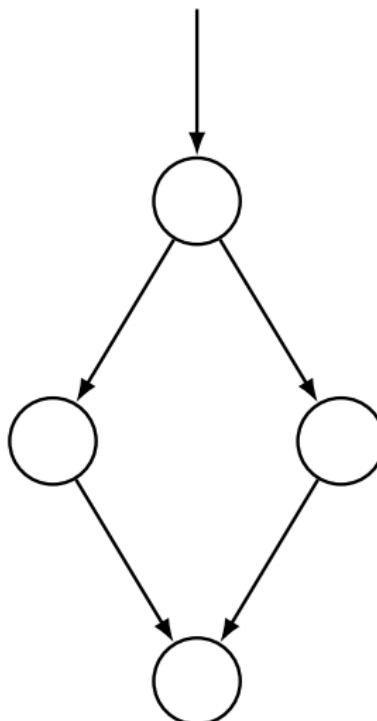


What about branches and loops?



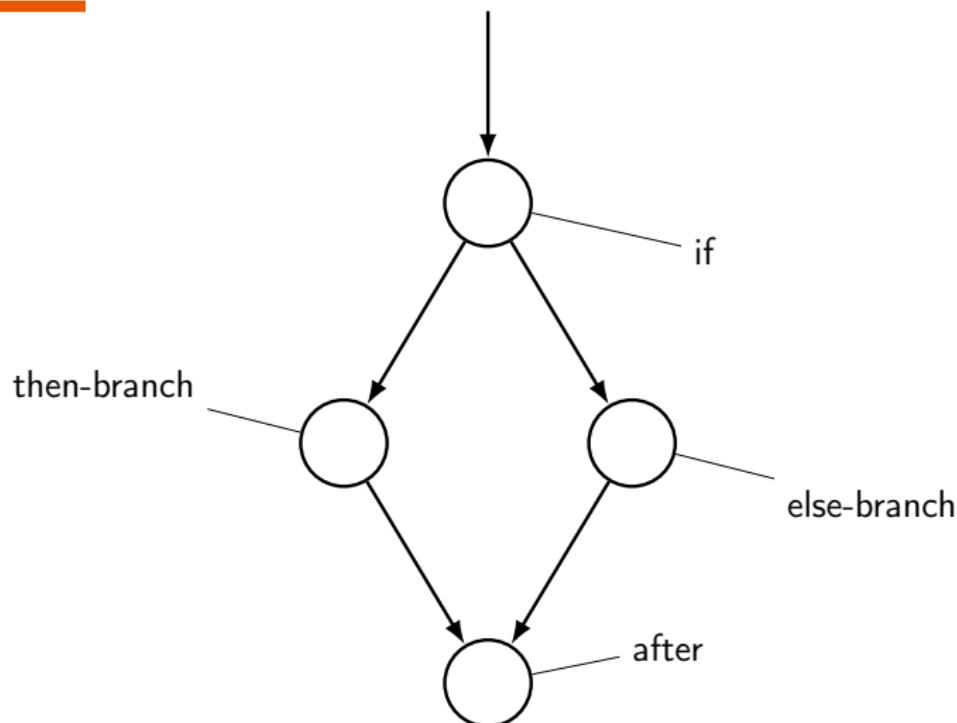


Do you recognise these CFGs?



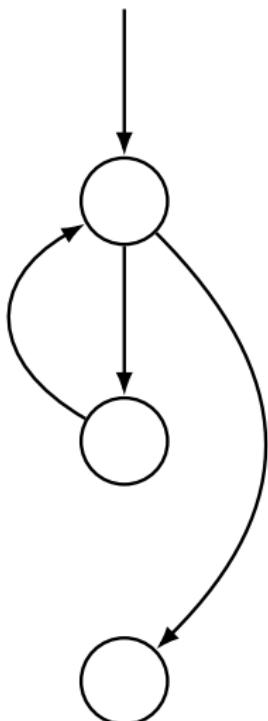


Do you recognise these CFGs?



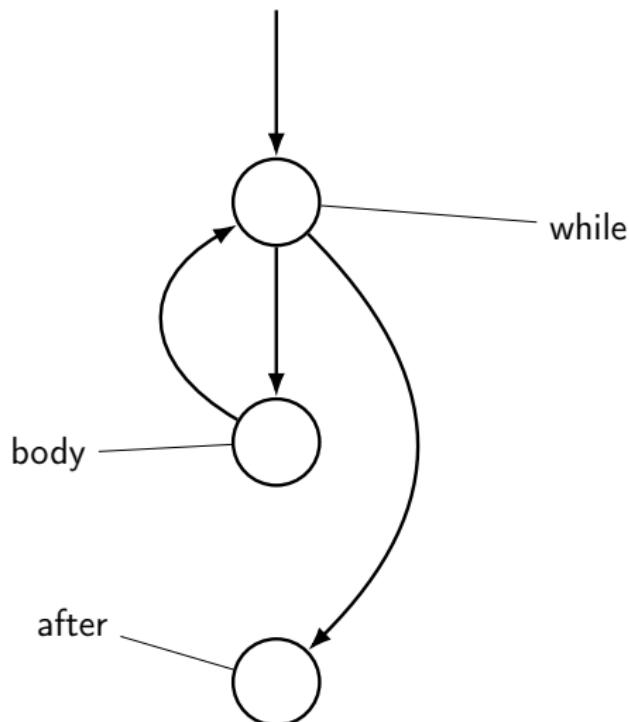


Do you recognise these CFGs?



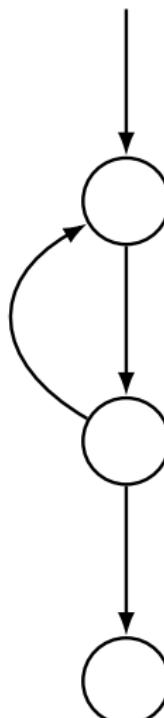


Do you recognise these CFGs?



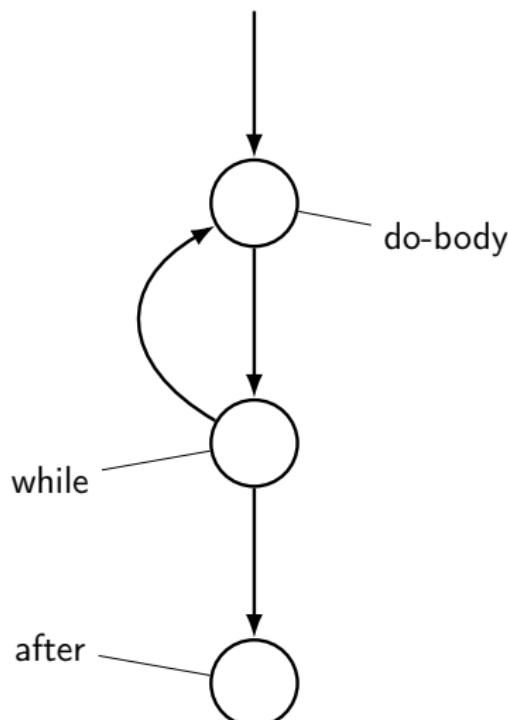


Do you recognise these CFGs?





Do you recognise these CFGs?





CFGs in angr

p.analyses

.CFGFast

p.analyses

.CFGEmulated



CFGs in angr

p.analyses

.CFGFast

- ▶ static

p.analyses

.CFGEmulated

- ▶ dynamic



CFGs in angr

p.analyses

.CFGFast

- ▶ static
- ▶ faster

p.analyses

.CFGEmulated

- ▶ dynamic
- ▶ slower



CFGs in angr

p.analyses

.CFGFast

- ▶ static
- ▶ faster
- ▶ can analyse basic jumps

p.analyses

.CFGEmulated

- ▶ dynamic
- ▶ slower
- ▶ can analyse more obscure/hidden jumps, with context



CFGs in angr

p.analyses

.CFGFast

- ▶ static
- ▶ faster
- ▶ can analyse basic jumps

p.analyses

.CFGEmulated

- ▶ dynamic
- ▶ slower
- ▶ can analyse more obscure/hidden jumps, with context

Class parameters? Check the [docs](#) + source code!



Demo – Labyrinth

Challenge 12 Solves X

Labyrinth

477

Author: sky

To get the flag you'll need to get to the end of a maz- five randomly generated mazes within five minutes.

This is an automatic reversing challenge. You will be provided an ELF as a hex string. You should analyze it, construct an input to make it terminate with exit(0), and then respond with your input in the same format. You will need to solve five binaries within a five minute timeout.

Challenge is solvable in under 4096 bytes of input and larger inputs may be buffered strangely on the server.

SNI: [labyrinth](#)

[labyrinth.zip](#)



Demo – Labyrinth

Challenge 12 Solves X

Labyrinth

477

Author: sky

To get the flag you'll need to get to the end of a maz- five randomly generated mazes within five minutes.

This is an automatic reversing challenge. You will be provided an ELF as a hex string. You should analyze it, construct an input to make it terminate with exit(0), and then respond with your input in the same format. You will need to solve five binaries within a five minute timeout.

Challenge is solvable in under 4096 bytes of input and larger inputs may be buffered strangely on the server.

SNI: **labyrinth**

[labyrinth.zip](#)



Summary – Labyrinth



1. Load project



Summary – Labyrinth



1. Load project
2. Construct CFG (`p.analyses.CFGFast`)



Summary – Labyrinth

1. Load project
2. Construct CFG (`p.analyses.CFGFast`)
3. Get 'from' and 'to' nodes (`main, exit(0)`)
4. Construct shortest path (`networkx.shortest_path`)
5. Follow shortest path



Summary – Labyrinth

1. Load project
2. Construct CFG (`p.analyses.CFGFast`)
3. Get 'from' and 'to' nodes (`main, exit(0)`)
4. Construct shortest path (`networkx.shortest_path`)
5. Follow shortest path
6. Take a dump
7. Profit!



But wait– there's moar!

Fun with CFGs

- ▶ Reachability
- ▶ Compiler Optimisations



But wait– there's moar!

Fun with CFGs

- ▶ Reachability
- ▶ Compiler Optimisations

Moar Analyses

- ▶ Value Flow Graph (VFG), Value Set Analysis (VSA)
 - ▶ What values can this variable hold?
 - ▶ CFGs are to instructions as VFGs are to data.



But wait– there's moar!

Fun with CFGs

- ▶ Reachability
- ▶ Compiler Optimisations

Moar Analyses

- ▶ Value Flow Graph (VFG), Value Set Analysis (VSA)
 - ▶ What values can this variable hold?
 - ▶ CFGs are to instructions as VFGs are to data.
- ▶ Data Dependency Graph (DDG)
 - ▶ What statements does this variable depend on?



But wait– there's moar!

Fun with CFGs

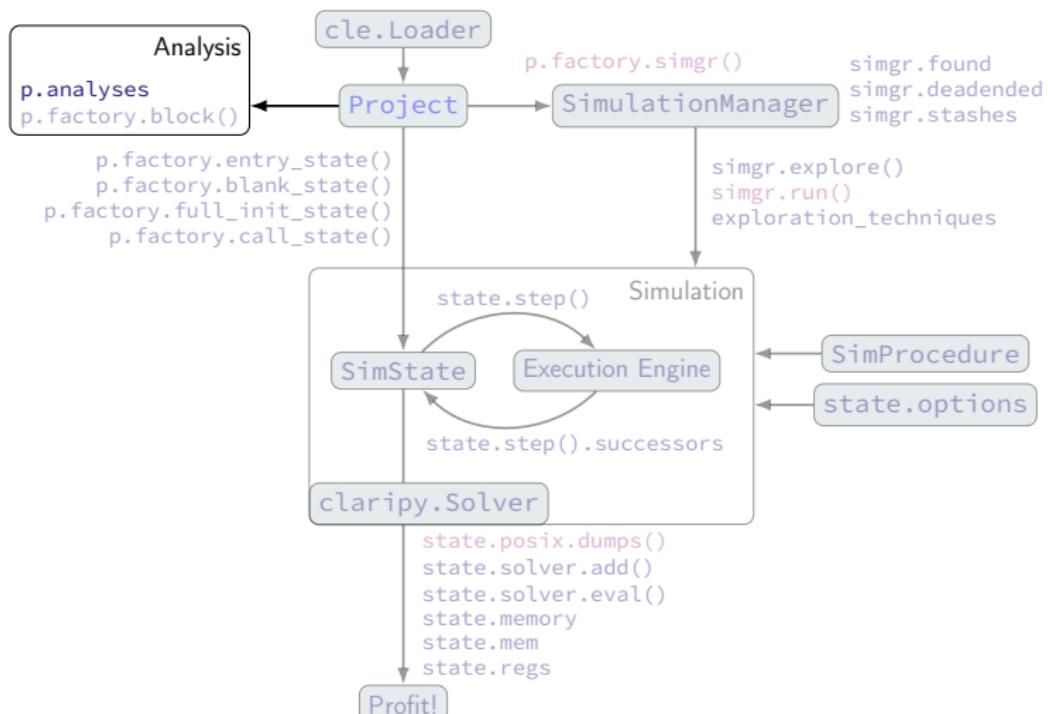
- ▶ Reachability
- ▶ Compiler Optimisations

Moar Analyses

- ▶ Value Flow Graph (VFG), Value Set Analysis (VSA)
 - ▶ What values can this variable hold?
 - ▶ CFGs are to instructions as VFGs are to data.
- ▶ Data Dependency Graph (DDG)
 - ▶ What statements does this variable depend on?
- ▶ Moar!

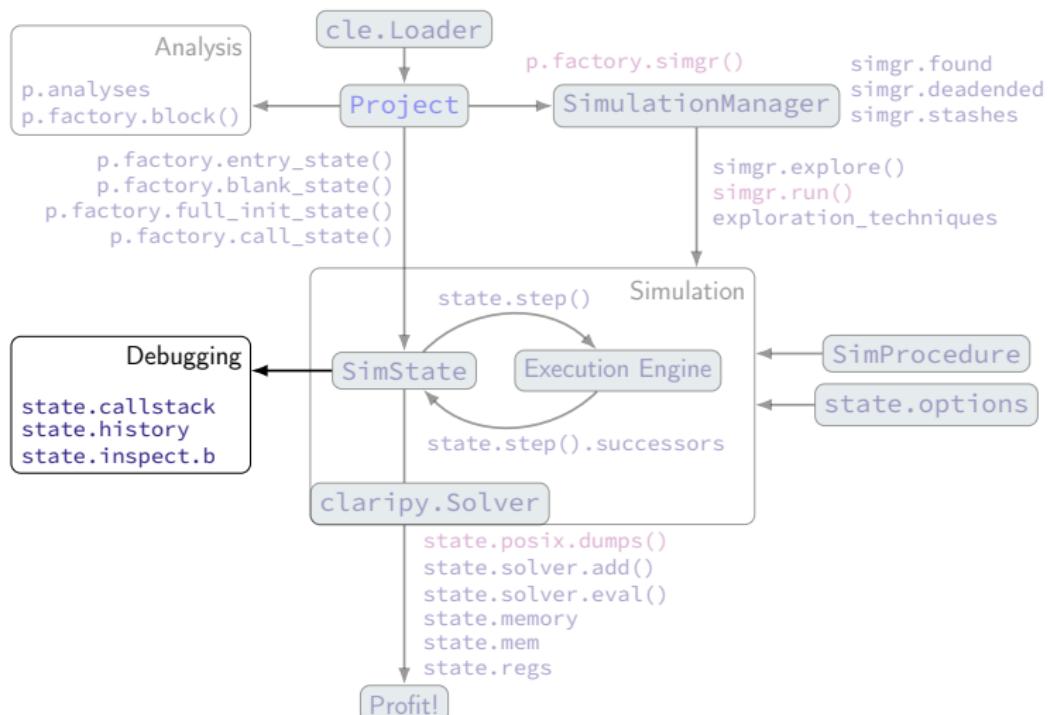


What's next?





What's next?





Questions?





Debugging angr Programs

Debugging angr Programs

Towards Enhanced Emotional Resilience and Cognizance





logging

```
import logging

logging.getLogger('angr')
    .setLevel(logging.INFO)

logging.getLogger('angr.analyses.cfg')
    .setLevel(logging.DEBUG)
```

Level
DEBUG
INFO
WARN
ERROR
FATAL



```
INFO | 2022-04-28 13:11:56,279 | angr.analyses.cfg_base | Loaded 3 indirect jump resolvers (2 timeless, 1 generic).
DEBUG | 2022-04-28 13:11:56,282 | angr.analyses.cfg.cfg_fast | CFG recovery covers 1 regions:
DEBUG | 2022-04-28 13:11:56,282 | angr.analyses.cfg.cfg_fast | ... 0x401155 - 0x418be0
INFO | 2022-04-28 13:11:56,294 | angr.analyses.cfg.cfg_fast | Loaded 0 exception handlings from 0 binaries.
INFO | 2022-04-28 13:11:56,311 | angr.analyses.cfg.cfg_fast | Found 1745 functions with prologue scanning.
DEBUG | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Returning a new recon address: 0x418bda
DEBUG | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Searching address 418bda
DEBUG | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Searching address 418bdb
DEBUG | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Force-scanning to 0x418bda
DEBUG | 2022-04-28 13:12:01,390 | angr.analyses.cfg.cfg_fast | Force-scan jumping failed
DEBUG | 2022-04-28 13:12:03,074 | angr.analyses.cfg_base | Function chunk 0x418bda is probably used as a function alias
INFO | 2022-04-28 13:12:03,178 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO | 2022-04-28 13:12:03,190 | angr.engines.engine | Ticked state: <IRSB from 0x4011b3: 1 sat>
INFO | 2022-04-28 13:12:03,190 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
WARNING | 2022-04-28 13:12:03,205 | angr.stubs.memory_mixins.default_fill_mixin | The program is accessing memory at 0x418bda
```



logging

```
import logging

logging.getLogger('angr')
    .setLevel(logging.INFO)
```

```
logging.getLogger('angr.analyses.cfg')
    .setLevel(logging.DEBUG)
```

Level
DEBUG
INFO
WARN
ERROR
FATAL



```
INFO  | 2022-04-28 13:11:56,279 | angr.analyses.cfg_base | Loaded 3 indirect jump resolvers (2 timeless, 1 generic).
DEBUG | 2022-04-28 13:11:56,282 | angr.analyses.cfg.cfg_fast | CFG recovery covers 1 regions:
DEBUG | 2022-04-28 13:11:56,282 | angr.analyses.cfg.cfg_fast | ... 0x401155 - 0x418be0
INFO  | 2022-04-28 13:11:56,294 | angr.analyses.cfg.cfg_fast | Loaded 0 exception handlings from 0 binaries.
INFO  | 2022-04-28 13:11:56,311 | angr.analyses.cfg.cfg_fast | Found 1745 functions with prologue scanning.
DEBUG | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Returning a new recon address: 0x418bda
DEBUG | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Searching address 418bda
DEBUG | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Searching address 418bdb
DEBUG | 2022-04-28 13:12:01,389 | angr.analyses.cfg.cfg_fast | Force-scanning to 0x418bda
DEBUG | 2022-04-28 13:12:01,390 | angr.analyses.cfg.cfg_fast | Force-scan jumping failed
DEBUG | 2022-04-28 13:12:03,074 | angr.analyses.cfg_base | Function chunk 0x418bda is probably used as a function alias
INFO  | 2022-04-28 13:12:03,178 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
INFO  | 2022-04-28 13:12:03,190 | angr.engines.engine | Ticked state: <IRSB from 0x4011b3: 1 sat>
INFO  | 2022-04-28 13:12:03,190 | angr.sim_manager | Stepping active of <SimulationManager with 1 active>
WARNING| 2022-04-28 13:12:03,205 | angr.stubs.memory_mixins.default_fill_mixin | The program is accessing memory at 0x418bda
```

Check `logging.root.manager.loggerDict` for existing loggers!



Debugging States

- ▶ `state.callstack`: addresses, flow.
- ▶ `state.history`: data, constraints, jumps.
- ▶ `state.inspect.b`: breakpoints!



Debugging States

- ▶ `state.callstack`: addresses, flow.
- ▶ `state.history`: data, constraints, jumps.
- ▶ `state.inspect.b`: breakpoints!

Actually these aren't just for debugging!



state.callstack

```
>>> print(state.callstack)
Backtrace:
Frame 0: 0x40a327 => 0x4011c4, sp = 0x7fffffffefefe78
Frame 1: 0x411863 => 0x40a2d8, sp = 0x7fffffffefefe98
Frame 2: 0x401f84 => 0x411809, sp = 0x7fffffffefefeb8
Frame 3: 0x406c14 => 0x401f41, sp = 0x7fffffffefefed8
Frame 4: 0x4069b2 => 0x406bbe, sp = 0x7fffffffefefef8
Frame 5: 0x402b53 => 0x406936, sp = 0x7fffffffefeff18
Frame 6: 0x401d4b => 0x402aed, sp = 0x7fffffffefeff38
Frame 7: 0x4104d4 => 0x401d08, sp = 0x7fffffffefeff58
Frame 8: 0x40b5aa => 0x410464, sp = 0x7fffffffefeff78
Frame 9: 0x413179 => 0x40b4f3, sp = 0x7fffffffefeff98
Frame 10: 0x4148df => 0x413123, sp = 0x7fffffffefffb8
Frame 11: 0x402523 => 0x41487d, sp = 0x7fffffffefffd8
Frame 12: 0x4011b3 => 0x4024c1, sp = 0x7fffffffefefff8
Frame 13: 0x0 => 0x0, sp = 0xffffffffffffffffffff
```



state.callstack

```
>>> print(state.callstack)
Backtrace:
Frame 0: 0x40a327 => 0x4011c4, sp = 0x7fffffffefefe78
Frame 1: 0x411863 => 0x40a2d8, sp = 0x7fffffffefefe98
Frame 2: 0x401f84 => 0x411809, sp = 0x7fffffffefefeb8
Frame 3: 0x406c14 => 0x401f41, sp = 0x7fffffffefefed8
Frame 4: 0x4069b2 => 0x406bbe, sp = 0x7fffffffefefef8
Frame 5: 0x402b53 => 0x406936, sp = 0x7fffffffefeff18
Frame 6: 0x401d4b => 0x402aed, sp = 0x7fffffffefeff38
Frame 7: 0x4104d4 => 0x401d08, sp = 0x7fffffffefeff58
Frame 8: 0x40b5aa => 0x410464, sp = 0x7fffffffefeff78
Frame 9: 0x413179 => 0x40b4f3, sp = 0x7fffffffefeff98
Frame 10: 0x4148df => 0x413123, sp = 0x7fffffffefffb8
Frame 11: 0x402523 => 0x41487d, sp = 0x7fffffffefffd8
Frame 12: 0x4011b3 => 0x4024c1, sp = 0x7fffffffeffff8
Frame 13: 0x0 => 0x0, sp = 0xffffffffffffffffffff
```

Useful Stuff

```
print(state.callstack)      # Display backtrace.
state.callstack.stack_ptr # Stack pointer (int).
state.callstack.func_addr # Address of current function.
state.callstack.ret_addr  # Return address.
```



state.history – Overview

A collection of different iterables...

- ▶ `.actions`: tracks data and constraints.



state.history – Overview

A collection of different iterables...

- ▶ `.actions`: tracks data and constraints.
- ▶ `.bbl_addrs`: blocks executed to reach this state.
- ▶ `.descriptions`: logs as strings.
- ▶ `.jump_guards`: branch constraints.
- ▶ `.events`: similar to `.actions` but moar.
 - ▶ Useful for checking why states deadended.



state.history – Viewing

Useful Stuff

- ▶ `state.history.actions.hardcopy`
 - ▶ Turn the actions iterable into a list.
- ▶ `print(*state.history.actions, sep='\n')`
 - ▶ Prints each action on its own line.



state.history.actions

Tracks data (register/memory read and writes) and constraints.

```
>>> print(*state.history.actions, sep='\n')
<SimActionData __isoc99_sccanf() reg/read: rdi ----> <BV64 0x419004>>
<SimActionData __isoc99_sccanf() mem/read: <BV64 0x419004> ----> <BV1032 0x25750a00011b033b741b0000bc81feff201c0000ca81feff3c1c00001f82feff5c1c0000da82feff7c1c00005583feff9c1c0000aa83feffb1c0000
<SimActionData __isoc99_sccanf() mem/read: <BV64 0x419004> ----> <BV24 0x25750a>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[7:0] == scanf_u_2_32 / 0x1 % 0>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[15:8] == scanf_u_2_32 / 0xa % 0>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[23:16] == scanf_u_2_32 / 0x64 % 0>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[31:24] == scanf_u_2_32 / 0x3e % 0>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[39:32] == scanf_u_2_32 / 0x27 % 0>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[47:40] == scanf_u_2_32 / 0x18 % 0>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[55:48] == scanf_u_2_32 / 0xf4 % 0>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[63:56] == scanf_u_2_32 / 0x98 % 0>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[71:64] == scanf_u_2_32 / 0x5f % 0>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[79:72] == scanf_u_2_32 / 0x3b % 0>>
<SimActionData __isoc99_sccanf() reg/read: rsi ----> <BV64 0x7fffffffffffffec>>
<SimActionData __isoc99_sccanf() mem/write: <BV64 0x7fffffffffffffec> <<---- <BV32 scanf_u_2_32>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_1_stdin_4_8 == 10>>
```



state.history.actions

Tracks data (register/memory read and writes) and constraints.

```
>>> print(*state.history.actions, sep='\n')
<SimActionData __isoc99_sccanf() reg/read: rdi ----> <BV64 0x419004>>
<SimActionData __isoc99_sccanf() mem/read: <BV64 0x419004> ----> <BV1032 0x25750a00011b033b741b0000bc81feff201c0000ca81feff3c1c00001f82feff5c1c0000da82feff7c1c00005583feff9c1c0000aa83feffb1c0000
<SimActionData __isoc99_sccanf() mem/read: <BV64 0x419004> ----> <BV24 0x25750a>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[7:0] == scanf_u_2_32 / 0x1 % 0>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[15:8] == scanf_u_2_32 / 0xa % 0>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[23:16] == scanf_u_2_32 / 0x64 % 0>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[31:24] == scanf_u_2_32 / 0x3e % 0>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[39:32] == scanf_u_2_32 / 0x27 % 0>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[47:40] == scanf_u_2_32 / 0x18 % 0>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[55:48] == scanf_u_2_32 / 0xf4 % 0>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[63:56] == scanf_u_2_32 / 0x98 % 0>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[71:64] == scanf_u_2_32 / 0x5f % 0>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_0_stdin_3_80[79:72] == scanf_u_2_32 / 0x3b % 0>
<SimActionData __isoc99_sccanf() reg/read: rsi ----> <BV64 0x7fffffffffffffec>>
<SimActionData __isoc99_sccanf() mem/write: <BV64 0x7fffffffffffffec> <<---- <BV32 scanf_u_2_32>>
<SimActionConstraint __isoc99_sccanf() <SAO <Bool packet_1_stdin_4_8 == 10>>
```

Useful Stuff

Add this before
constructing the simgr!

```
state.options |= sim_options.refs # Track EVERYTHING!
```



state.inspect.b(event_type, **kwargs)



event_type

- ▶ `mem_read`, `mem_write`: memory is being read/written.
- ▶ `reg_read`, `reg_write`: register is being read/written.
- ▶ `constraints`: new constraints are being added.
- ▶ `call`: a `call` instruction is hit.
- ▶ `ret`: a `ret` instruction is hit.
- ▶ `symbolic_variable`: a new symbolic variable is being created.
- ▶ `syscall`: a syscall is being executed.



state.inspect.b(event_type, **kwargs)

when=

- ▶ `BP_BEFORE`: trigger before the breakpoint.
- ▶ `BP_AFTER`: trigger after the breakpoint.

action=

- ▶ `func`: execute your own Python handler.
- ▶ `BP_IPDB`: enter an IPython debugger.
- ▶ `BP_IPYTHON`: enter an IPython shell.

**kwargs

- ▶ Different kwargs are available for each `event_type`! Check the link below.



Appendix

⑥ Appendix

Tips

angr Model

More Links



angr Tips

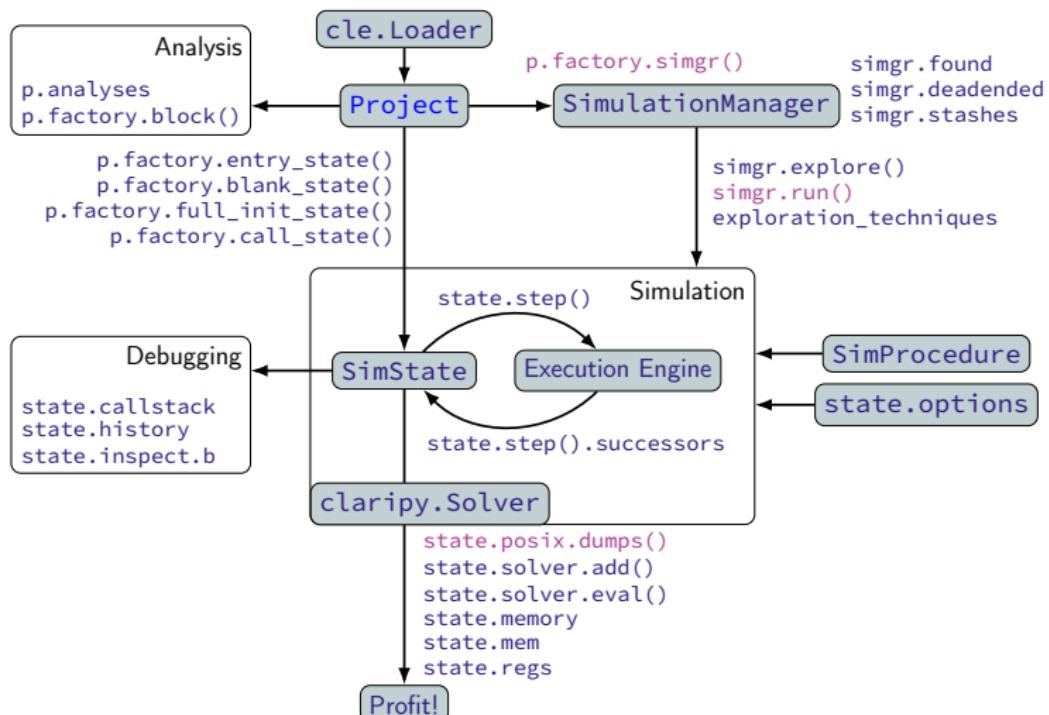


Kalm

- ▶ Breathe and relax.
- ▶ Always use angr in a virtual environment (e.g. [venv](#), [conda](#)).
- ▶ Use a Jupyter notebook/IPython, or run scripts interactively (`python3 -i script.py`) to easily play around with variables and methods.
- ▶ `import monkeyhex` to conveniently view numbers as hex.
- ▶ Tab completion is your friend.
- ▶ The most updated documentation is the code itself.



angr Model (Link Galore)





More Links

- ▶ Repo: <https://github.com/angr/angr>
- ▶ Docs (Book-like): <https://docs.angr.io/>
- ▶ API: <https://api.angr.io/>
- ▶ Examples: <https://docs.angr.io/examples>
- ▶ More Examples: <https://github.com/angr/angr-doc/blob/master/docs/more-examples.md>
- ▶ Attack Plan (Strategies):
<https://github.com/bordig-f/angr-strategies>
- ▶ CFG (Wikipedia):
https://en.wikipedia.org/wiki/Control-flow_graph
- ▶ CFG (1970s paper, a potentially interesting read, mathy/theoretical):
<https://dl.acm.org/doi/10.1145/390013.808479>



Appendix

Questions?

