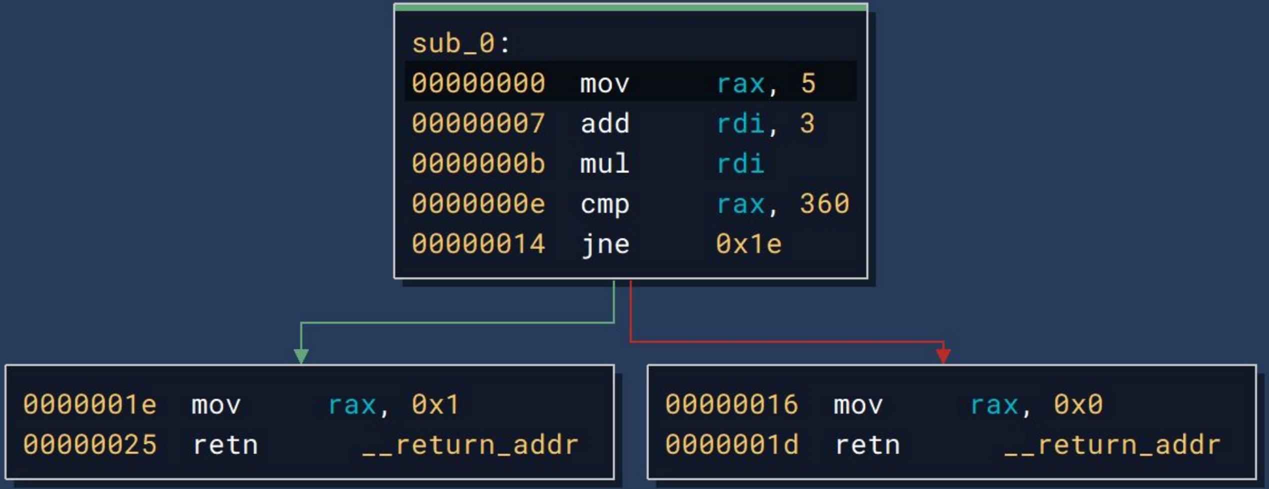
Symbolic Execution

Using the angr Framework

What is symbolic execution?

- Execution of assembly code with symbolic values
- Values can be expressions such as 5*x, instead of a concrete number
- This makes systematic solving of certain programs possible

Symbolic Execution


```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e
```

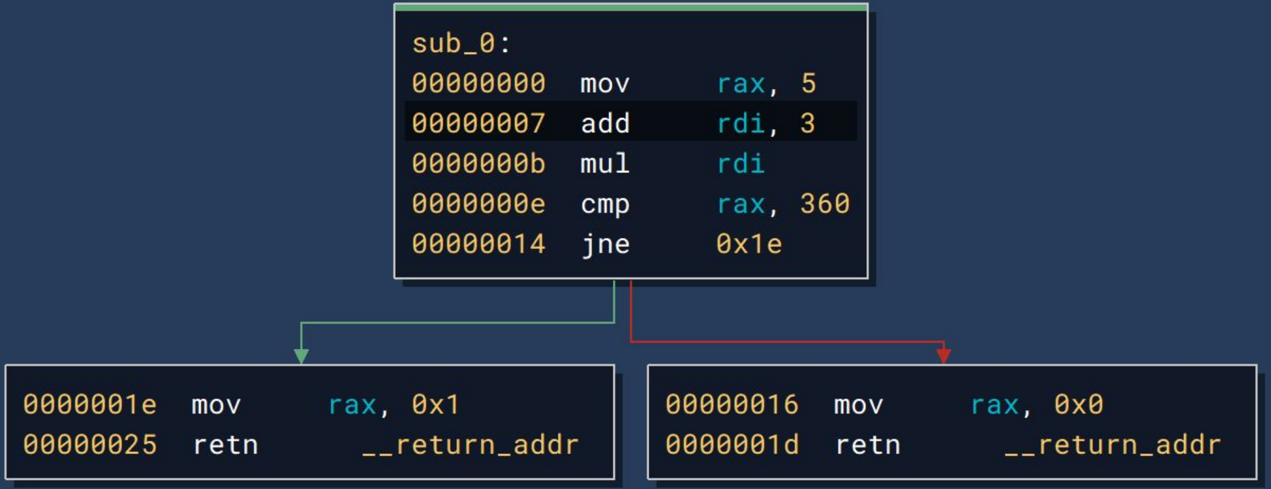
```
0000001e  mov     rax, 0x1
00000025  retn      __return_addr
```

```
00000016  mov     rax, 0x0
0000001d  retn      __return_addr
```

Variables

```
RAX = RAX
RDI = RDI
ZF  = ZF
```

Symbolic Execution



```
sub_0:
00000000  mov      rax, 5
00000007  add      rdi, 3
0000000b  mul      rdi
0000000e  cmp      rax, 360
00000014  jne      0x1e
```

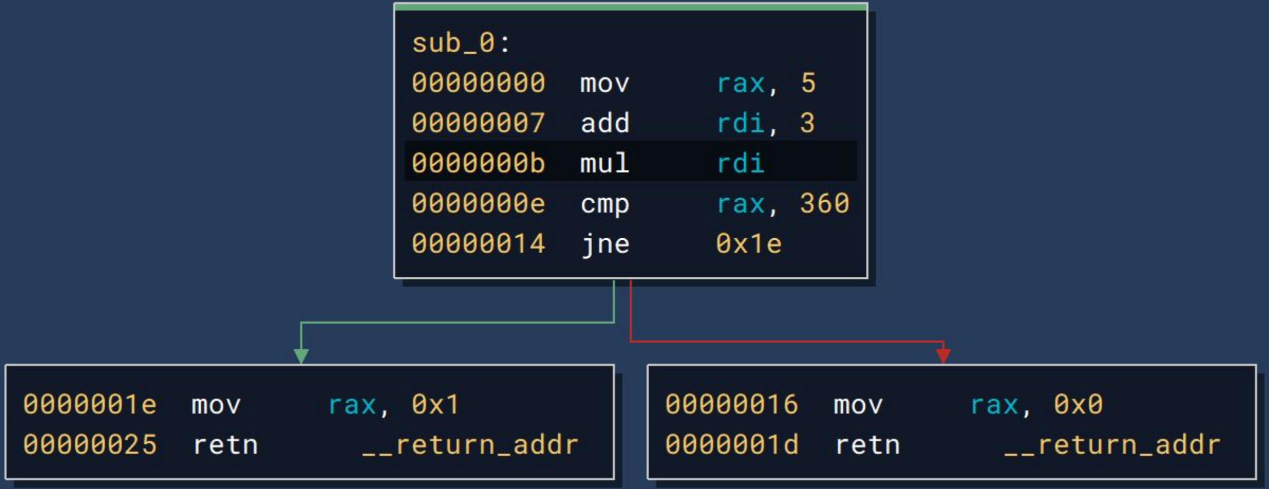
```
0000001e  mov      rax, 0x1
00000025  retn        __return_addr
```

```
00000016  mov      rax, 0x0
0000001d  retn        __return_addr
```

Variables

```
RAX = 5
RDI = RDI
ZF  = ZF
```

Symbolic Execution



```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e
```

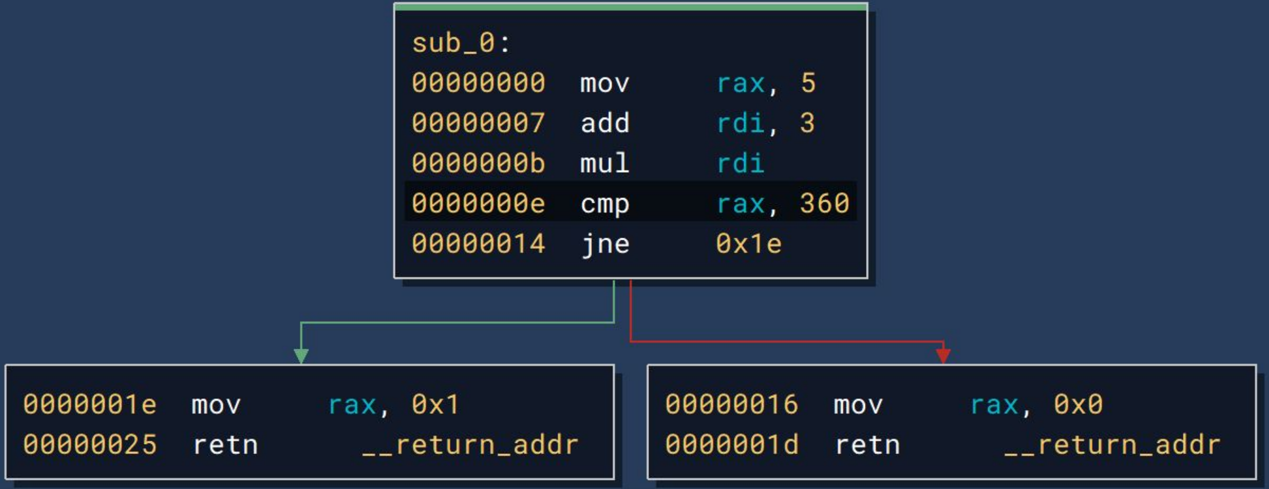
```
0000001e  mov     rax, 0x1
00000025  retn      __return_addr
```

```
00000016  mov     rax, 0x0
0000001d  retn      __return_addr
```

Variables

```
RAX = 5
RDI = RDI+3
ZF  = ZF
```

Symbolic Execution


```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e
```

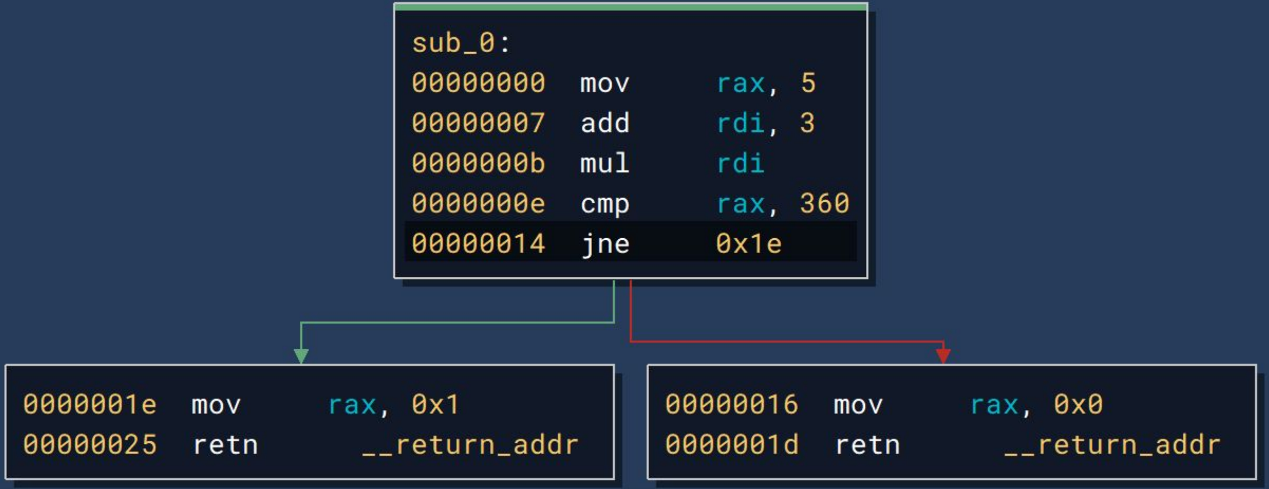
```
0000001e  mov     rax, 0x1
00000025  retn      __return_addr
```

```
00000016  mov     rax, 0x0
0000001d  retn      __return_addr
```

Variables

```
RAX = 5*(RDI+3)
RDI = RDI+3
ZF  = ZF
```

Symbolic Execution



```
sub_0:
00000000  mov    rax, 5
00000007  add    rdi, 3
0000000b  mul    rdi
0000000e  cmp    rax, 360
00000014  jne    0x1e
```

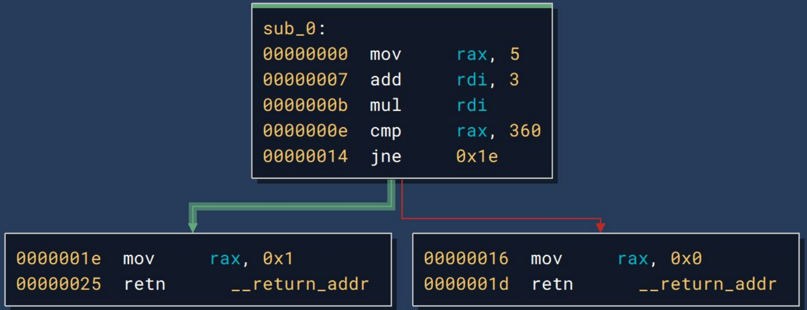
```
0000001e  mov    rax, 0x1
00000025  retn     __return_addr
```

```
00000016  mov    rax, 0x0
0000001d  retn     __return_addr
```

Variables

```
RAX = 5*(RDI+3)
RDI = RDI+3
ZF  = 5*(RDI+3)==360
```

Symbolic Execution



```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e
```

```
0000001e  mov     rax, 0x1
00000025  retn      __return_addr
```

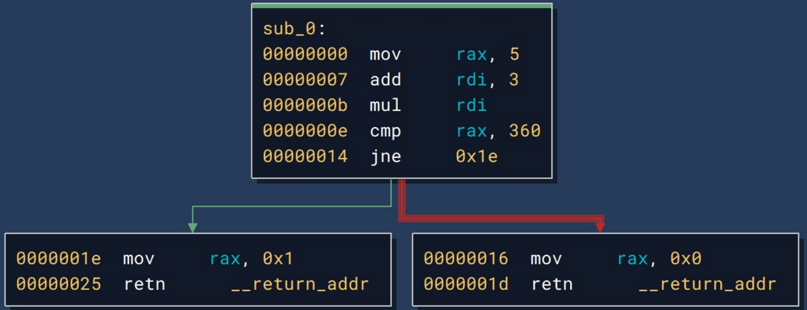
```
00000016  mov     rax, 0x0
0000001d  retn      __return_addr
```

Variables

```
RAX = 5*(RDI+3)
RDI = RDI+3
ZF  = 5*(RDI+3)==360
```

Assertions

```
5*(RDI+3)==360
```



```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e
```

```
0000001e  mov     rax, 0x1
00000025  retn      __return_addr
```

```
00000016  mov     rax, 0x0
0000001d  retn      __return_addr
```

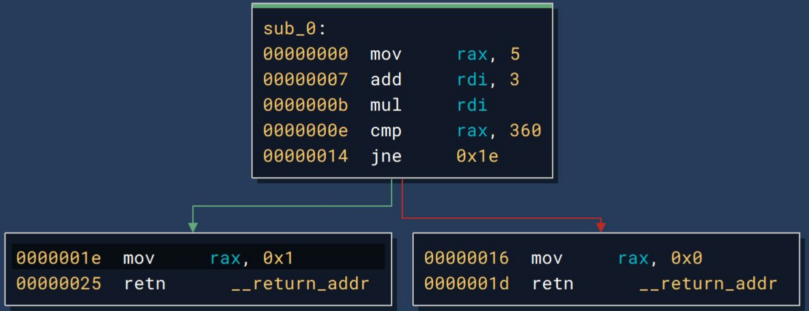
Variables

```
RAX = 5*(RDI+3)
RDI = RDI+3
ZF  = 5*(RDI+3)==360
```

Assertions

```
5*(RDI+3)!=360
```

Symbolic Execution



```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e
```

```
0000001e  mov     rax, 0x1
00000025  retn       __return_addr
```

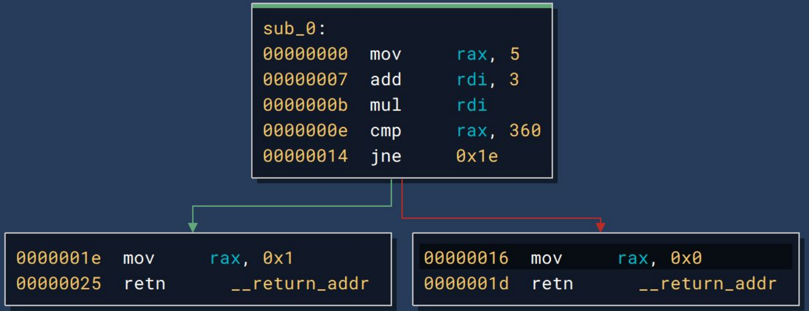
```
00000016  mov     rax, 0x0
0000001d  retn       __return_addr
```

Variables

```
RAX = 5*(RDI+3)
RDI = RDI+3
ZF  = 5*(RDI+3)==360
```

Assertions

```
5*(RDI+3)==360
```



```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e
```

```
0000001e  mov     rax, 0x1
00000025  retn       __return_addr
```

```
00000016  mov     rax, 0x0
0000001d  retn       __return_addr
```

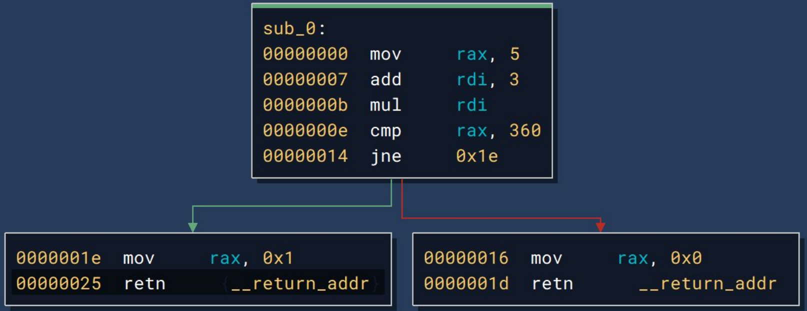
Variables

```
RAX = 5*(RDI+3)
RDI = RDI+3
ZF  = 5*(RDI+3)==360
```

Assertions

```
5*(RDI+3)!=360
```

Symbolic Execution



```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e
```

```
0000001e  mov     rax, 0x1
00000025  retn      __return_addr
```

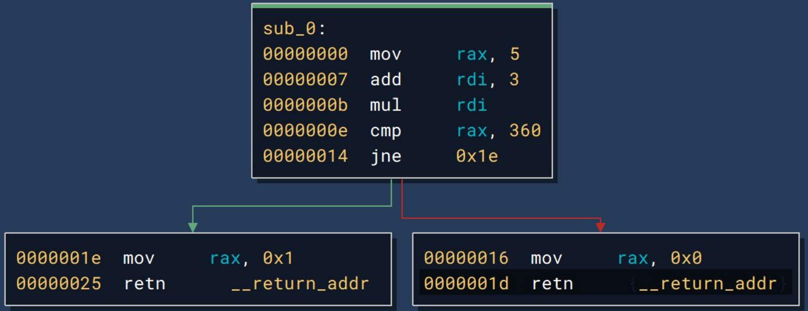
```
00000016  mov     rax, 0x0
0000001d  retn      __return_addr
```

Variables

```
RAX = 1
RDI = RDI+3
ZF  = 5*(RDI+3)==360
```

Assertions

```
5*(RDI+3)==360
```



```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e
```

```
0000001e  mov     rax, 0x1
00000025  retn      __return_addr
```

```
00000016  mov     rax, 0x0
0000001d  retn      __return_addr
```

Variables

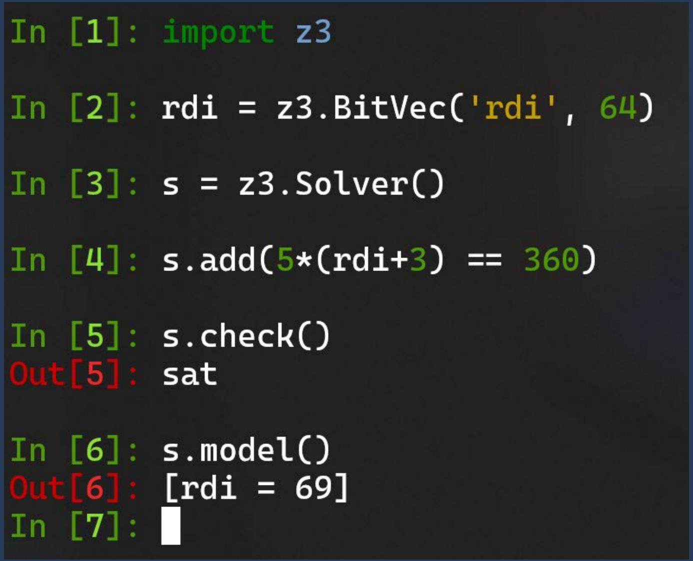
```
RAX = 0
RDI = RDI+3
ZF  = 5*(RDI+3)==360
```

Assertions

```
5*(RDI+3)!=360
```

Z3 to the rescue

- Z3 is a SMT solver developed by Microsoft
- Has very easy-to-use Python bindings

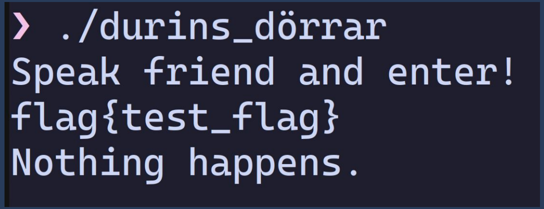


```
In [1]: import z3

In [2]: rdi = z3.BitVec('rdi', 64)

In [3]: s = z3.Solver()

In [4]: s.add(5*(rdi+3) == 360)

In [5]: s.check()
Out[5]: sat

In [6]: s.model()
Out[6]: [rdi = 69]
In [7]: 
```

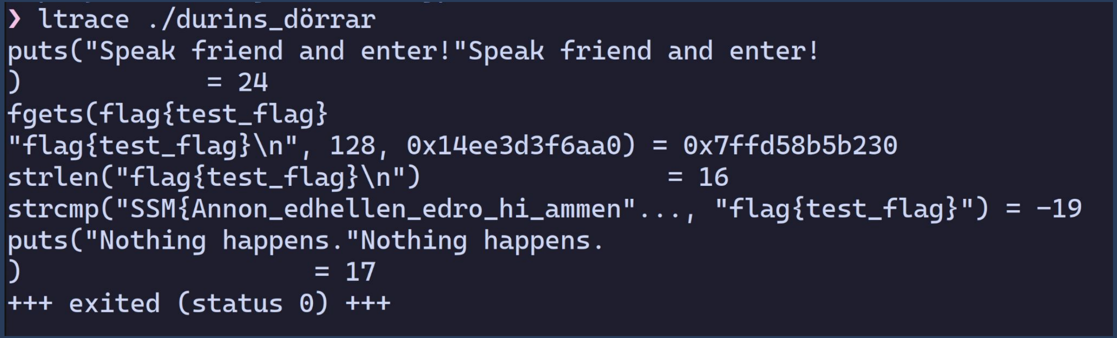
angr

- Binary Analysis tool kit made in Python
- Implements symbolic execution of various architectures
- Very easy to use and get going with

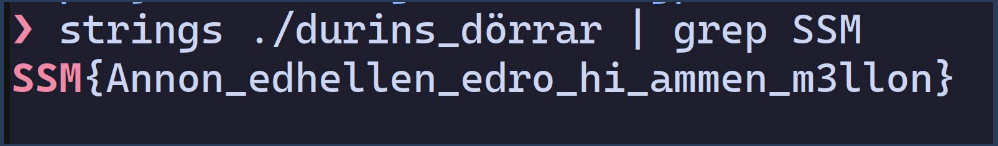
Example Challenge



```
❯ ./durins_dörrar
Speak friend and enter!
flag{test_flag}
Nothing happens.
```

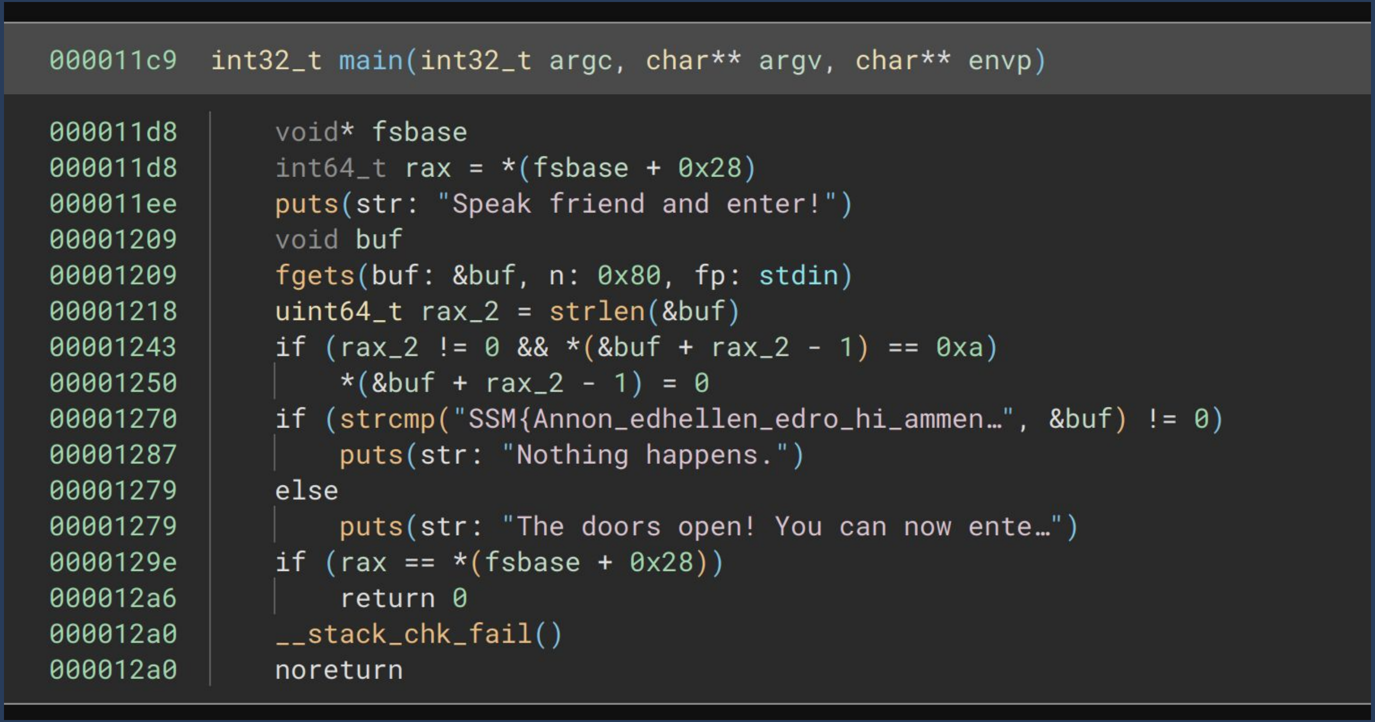


```
❯ ltrace ./durins_dörrar
puts("Speak friend and enter!"Speak friend and enter!
)            = 24
fgets(flag{test_flag}
"flag{test_flag}\n", 128, 0x14ee3d3f6aa0) = 0x7ffd58b5b230
strlen("flag{test_flag}\n")                = 16
strcmp("SSM{Annon_edhellen_edro_hi_ammen"..., "flag{test_flag}") = -19
puts("Nothing happens."Nothing happens.
)                  = 17
+++ exited (status 0) +++
```

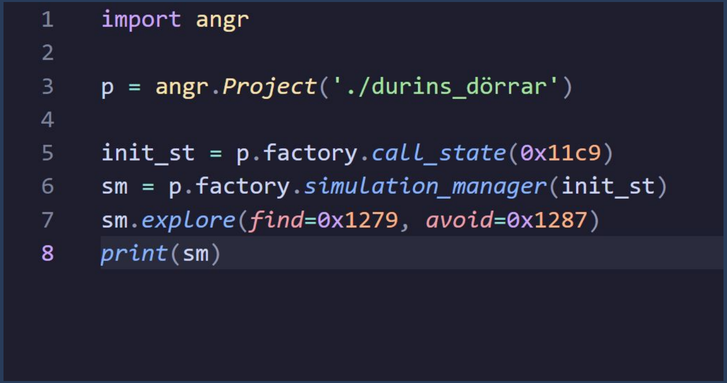


```
❯ strings ./durins_dörrar | grep SSM
SSM{Annon_edhellen_edro_hi_ammen_m3llon}
```

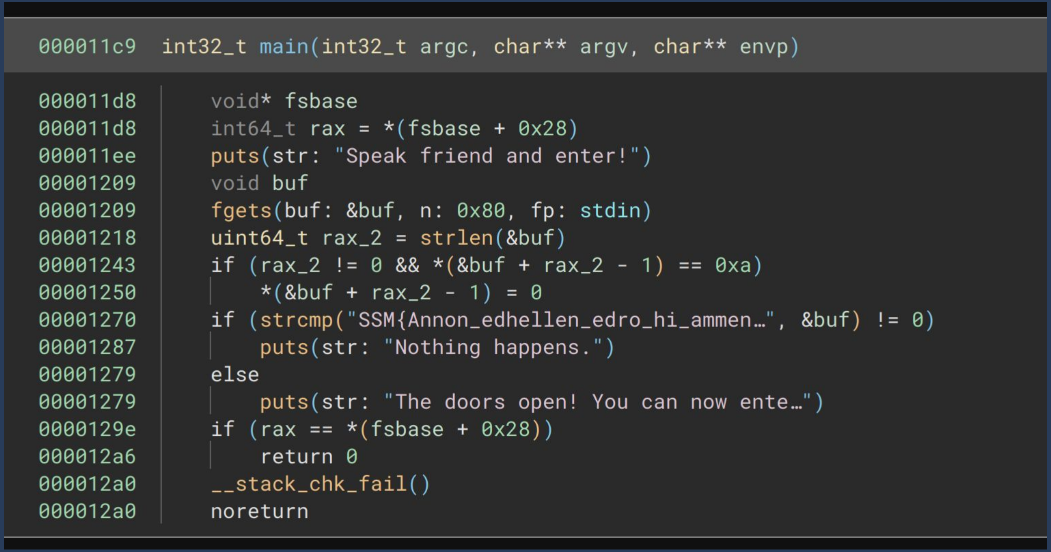
Example Challenge


```
000011c9  int32_t main(int32_t argc, char** argv, char** envp)

000011d8      void* fsbase
000011d8      int64_t rax = *(fsbase + 0x28)
000011ee      puts(str: "Speak friend and enter!")
00001209      void buf
00001209      fgets(buf: &buf, n: 0x80, fp: stdin)
00001218      uint64_t rax_2 = strlen(&buf)
00001243      if (rax_2 != 0 && *(&buf + rax_2 - 1) == 0xa)
00001250          *(&buf + rax_2 - 1) = 0
00001270      if (strcmp("SSM{Annon_edhellen_edro_hi_ammen…", &buf) != 0)
00001287          puts(str: "Nothing happens.")
00001279      else
00001279          puts(str: "The doors open! You can now ente…")
0000129e      if (rax == *(fsbase + 0x28))
000012a6          return 0
000012a0      __stack_chk_fail()
000012a0      noreturn
```

Example Challenge

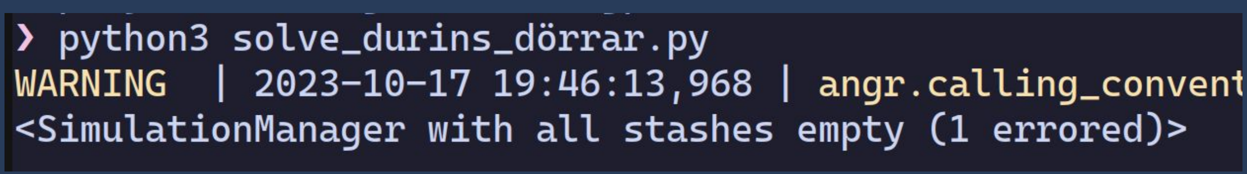


```python
import angr

p = angr.Project('./durins_dörrar')

init_st = p.factory.call_state(0x11c9)
sm = p.factory.simulation_manager(init_st)
sm.explore(find=0x1279, avoid=0x1287)
print(sm)
```



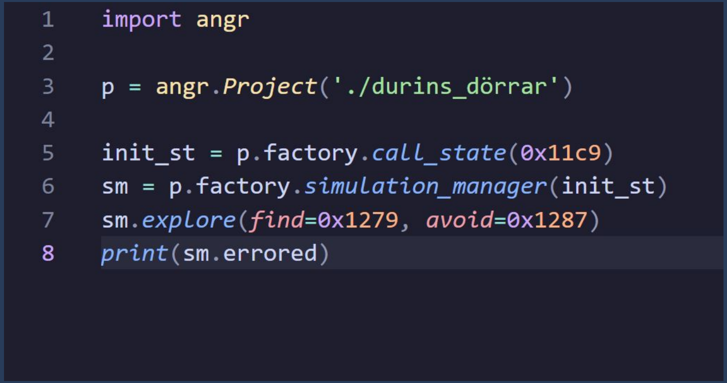
```
000011c9  int32_t main(int32_t argc, char** argv, char** envp)

000011d8      void* fsbase
000011d8      int64_t rax = *(fsbase + 0x28)
000011ee      puts(str: "Speak friend and enter!")
00001209      void buf
00001209      fgets(buf: &buf, n: 0x80, fp: stdin)
00001218      uint64_t rax_2 = strlen(&buf)
00001243      if (rax_2 != 0 && *(&buf + rax_2 - 1) == 0xa)
00001250          *(&buf + rax_2 - 1) = 0
00001270      if (strcmp("SSM{Annon_edhellen_edro_hi_ammen…", &buf) != 0)
00001287          puts(str: "Nothing happens.")
00001279      else
00001279          puts(str: "The doors open! You can now ente…")
0000129e      if (rax == *(fsbase + 0x28))
000012a6          return 0
000012a0      __stack_chk_fail()
000012a0      noreturn
```

Example Challenge

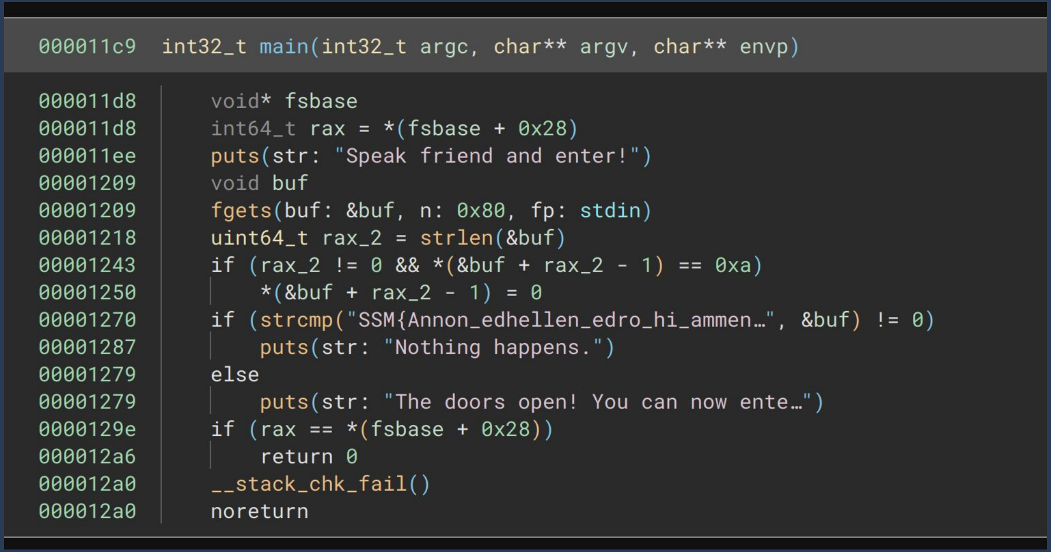


```
❯ python3 solve_durins_dörrar.py
WARNING  | 2023-10-17 19:46:13,968 | angr.calling_convent
<SimulationManager with all stashes empty (1 errored)>
```

Example Challenge

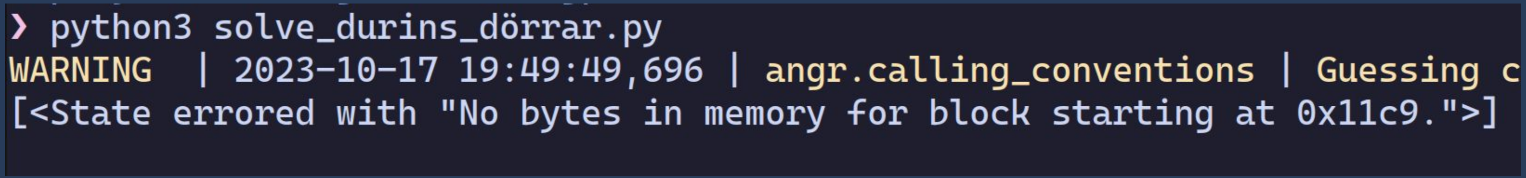


```python
import angr

p = angr.Project('./durins_dörrar')

init_st = p.factory.call_state(0x11c9)
sm = p.factory.simulation_manager(init_st)
sm.explore(find=0x1279, avoid=0x1287)
print(sm.errored)
```



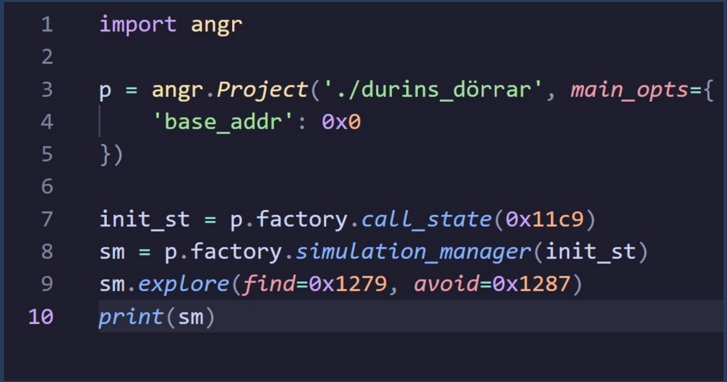
```
000011c9  int32_t main(int32_t argc, char** argv, char** envp)

000011d8      void* fsbase
000011d8      int64_t rax = *(fsbase + 0x28)
000011ee      puts(str: "Speak friend and enter!")
00001209      void buf
00001209      fgets(buf: &buf, n: 0x80, fp: stdin)
00001218      uint64_t rax_2 = strlen(&buf)
00001243      if (rax_2 != 0 && *(&buf + rax_2 - 1) == 0xa)
00001250          *(&buf + rax_2 - 1) = 0
00001270      if (strcmp("SSM{Annon_edhellen_edro_hi_ammen…", &buf) != 0)
00001287          puts(str: "Nothing happens.")
00001279      else
00001279          puts(str: "The doors open! You can now ente…")
0000129e      if (rax == *(fsbase + 0x28))
000012a6          return 0
000012a0      __stack_chk_fail()
000012a0      noreturn
```

Example Challenge

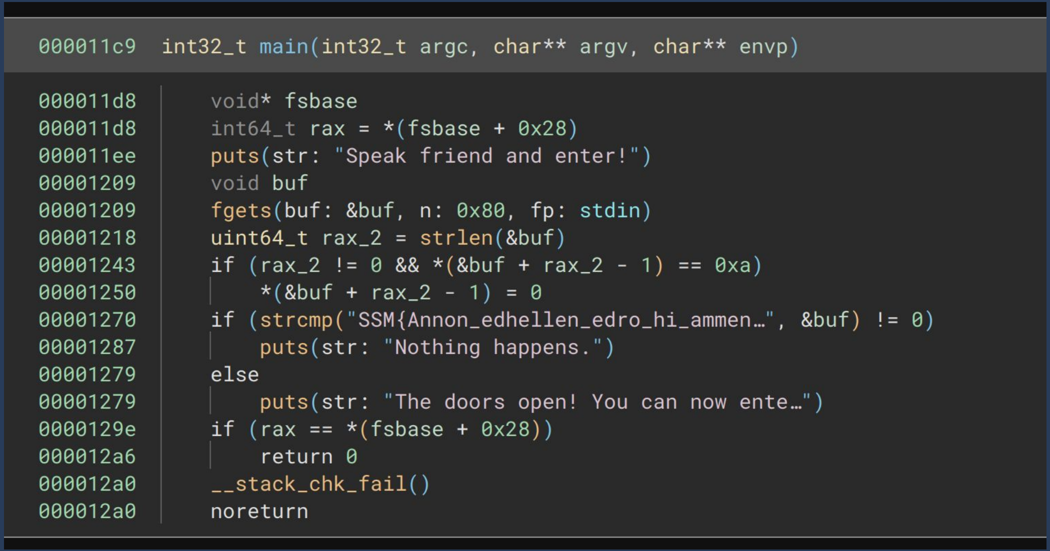


```
❯ python3 solve_durins_dörrar.py
WARNING  | 2023-10-17 19:49:49,696 | angr.calling_conventions | Guessing c
[<State errored with "No bytes in memory for block starting at 0x11c9.">]
```

Example Challenge

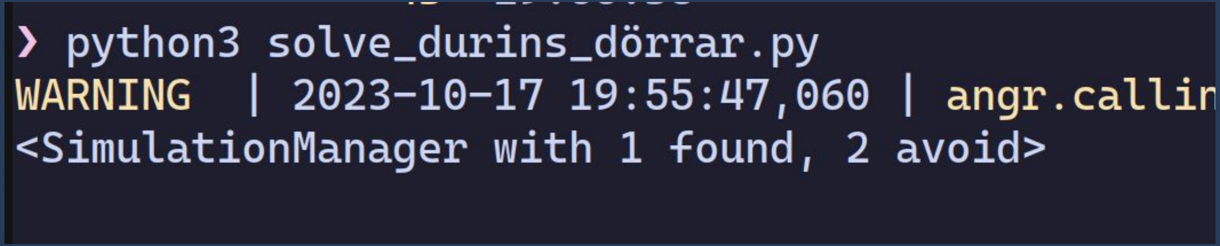


```python
import angr

p = angr.Project('./durins_dörrar', main_opts={
    'base_addr': 0x0
})

init_st = p.factory.call_state(0x11c9)
sm = p.factory.simulation_manager(init_st)
sm.explore(find=0x1279, avoid=0x1287)
print(sm)
```



```
000011c9  int32_t main(int32_t argc, char** argv, char** envp)

000011d8      void* fsbase
000011d8      int64_t rax = *(fsbase + 0x28)
000011ee      puts(str: "Speak friend and enter!")
00001209      void buf
00001209      fgets(buf: &buf, n: 0x80, fp: stdin)
00001218      uint64_t rax_2 = strlen(&buf)
00001243      if (rax_2 != 0 && *(&buf + rax_2 - 1) == 0xa)
00001250          *(&buf + rax_2 - 1) = 0
00001270      if (strcmp("SSM{Annon_edhellen_edro_hi_ammen…", &buf) != 0)
00001287          puts(str: "Nothing happens.")
00001279      else
00001279          puts(str: "The doors open! You can now ente…")
0000129e      if (rax == *(fsbase + 0x28))
000012a6          return 0
000012a0      __stack_chk_fail()
000012a0      noreturn
```

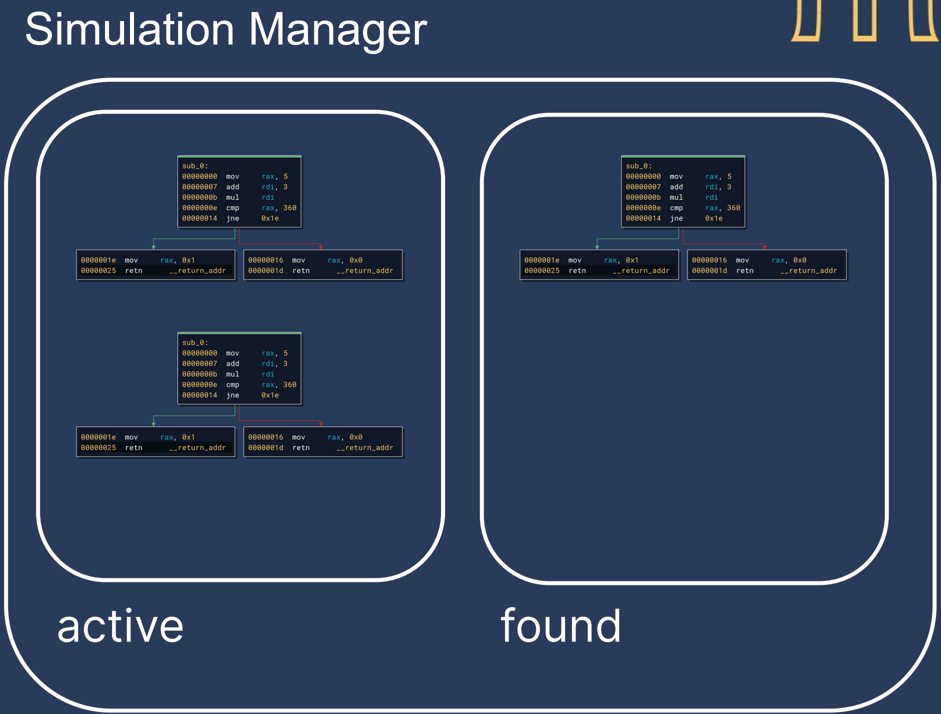
Example Challenge



```
❯ python3 solve_durins_dörrar.py
WARNING  | 2023-10-17 19:55:47,060 | angr.callin
<SimulationManager with 1 found, 2 avoid>
```

Simulation Structure

- Simulation manager contains multiple stashes
- Each stash contains multiple states
- A state is the executed program at a certain point in time

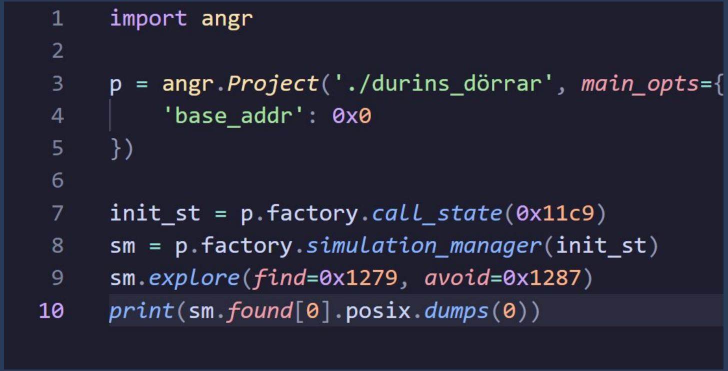


```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e

0000001e  mov     rax, 0x1
00000025  retn      __return_addr

00000016  mov     rax, 0x0
0000001d  retn      __return_addr
```

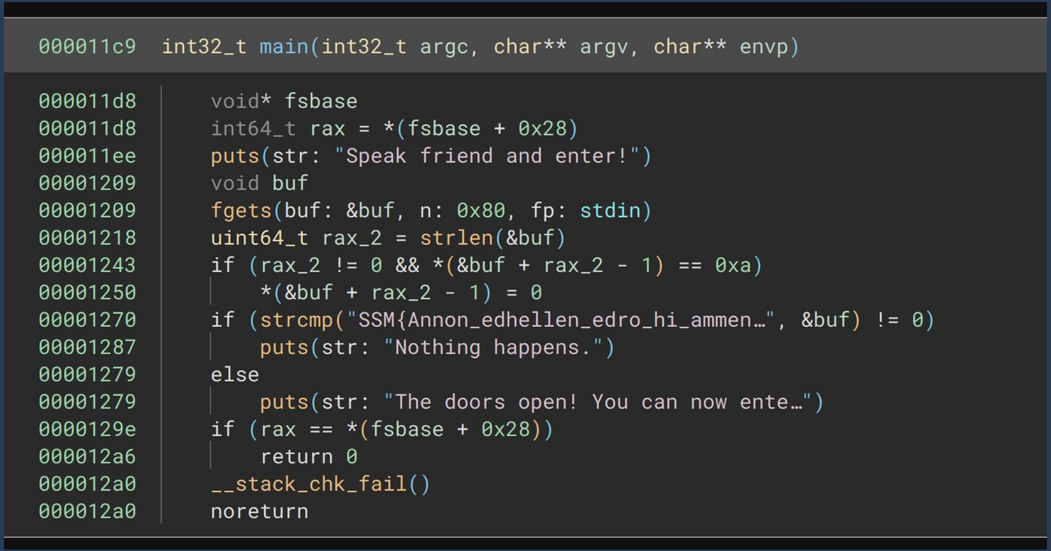
```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e

0000001e  mov     rax, 0x1
00000025  retn      __return_addr

00000016  mov     rax, 0x0
0000001d  retn      __return_addr
```

```
sub_0:
00000000  mov     rax, 5
00000007  add     rdi, 3
0000000b  mul     rdi
0000000e  cmp     rax, 360
00000014  jne     0x1e

0000001e  mov     rax, 0x1
00000025  retn      __return_addr

00000016  mov     rax, 0x0
0000001d  retn      __return_addr
```

Example Challenge

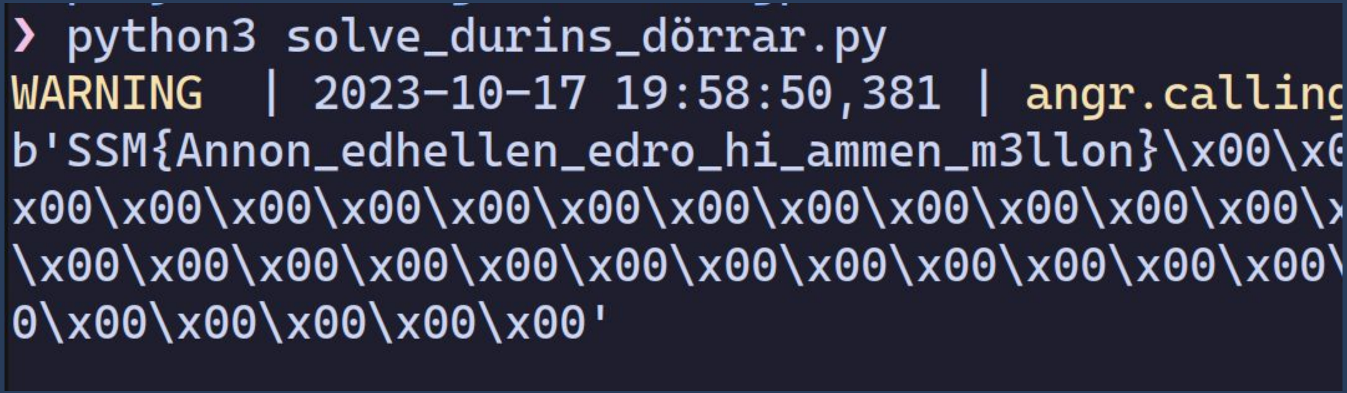


```python
import angr

p = angr.Project('./durins_dörrar', main_opts={
    'base_addr': 0x0
})

init_st = p.factory.call_state(0x11c9)
sm = p.factory.simulation_manager(init_st)
sm.explore(find=0x1279, avoid=0x1287)
print(sm.found[0].posix.dumps(0))
```



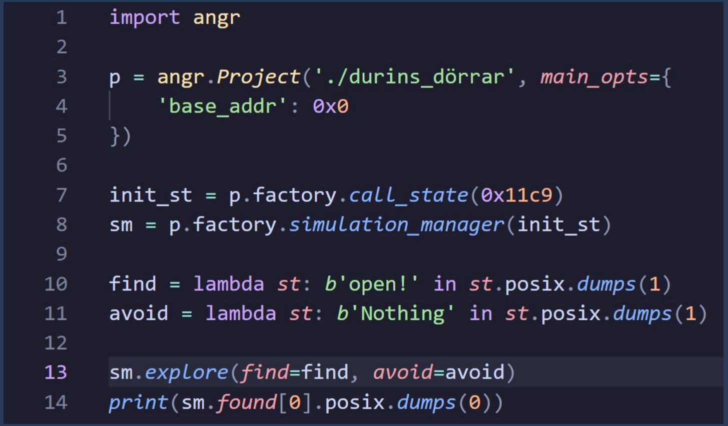
```
000011c9  int32_t main(int32_t argc, char** argv, char** envp)

000011d8      void* fsbase
000011d8      int64_t rax = *(fsbase + 0x28)
000011ee      puts(str: "Speak friend and enter!")
00001209      void buf
00001209      fgets(buf: &buf, n: 0x80, fp: stdin)
00001218      uint64_t rax_2 = strlen(&buf)
00001243      if (rax_2 != 0 && *(&buf + rax_2 - 1) == 0xa)
00001250          *(&buf + rax_2 - 1) = 0
00001270      if (strcmp("SSM{Annon_edhellen_edro_hi_ammen…", &buf) != 0)
00001287          puts(str: "Nothing happens.")
00001279      else
00001279          puts(str: "The doors open! You can now ente…")
0000129e      if (rax == *(fsbase + 0x28))
000012a6          return 0
000012a0      __stack_chk_fail()
000012a0      noreturn
```

Example Challenge

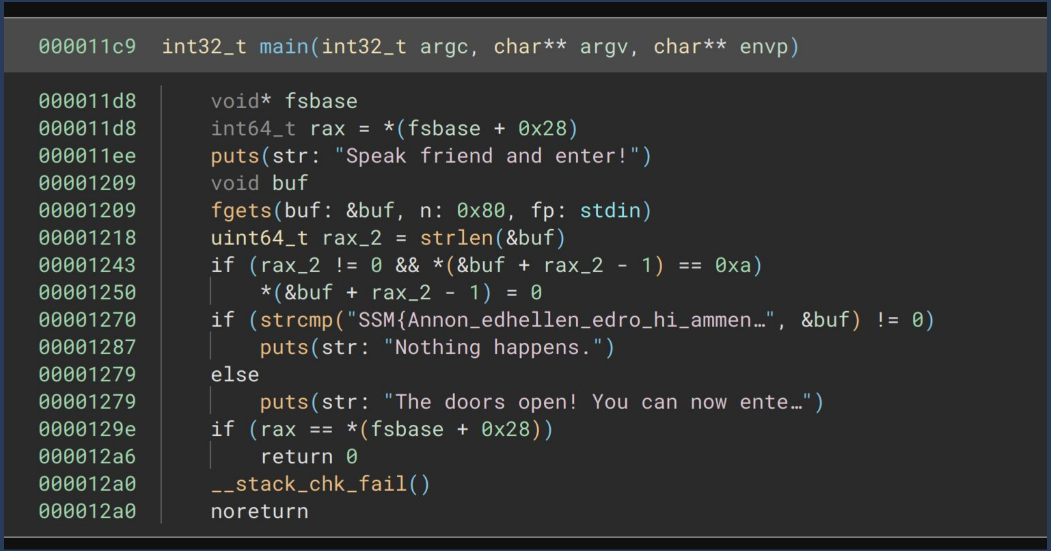


```
❯ python3 solve_durins_dörrar.py
WARNING  | 2023-10-17 19:58:50,381 | angr.callin
b'SSM{Annon_edhellen_edro_hi_ammen_m3llon}\x00\x0
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
0\x00\x00\x00\x00\x00'
```

Example Challenge



```python
import angr

p = angr.Project('./durins_dörrar', main_opts={
    'base_addr': 0x0
})

init_st = p.factory.call_state(0x11c9)
sm = p.factory.simulation_manager(init_st)

find = lambda st: b'open!' in st.posix.dumps(1)
avoid = lambda st: b'Nothing' in st.posix.dumps(1)

sm.explore(find=find, avoid=avoid)
print(sm.found[0].posix.dumps(0))
```



```
000011c9  int32_t main(int32_t argc, char** argv, char** envp)

000011d8      void* fsbase
000011d8      int64_t rax = *(fsbase + 0x28)
000011ee      puts(str: "Speak friend and enter!")
00001209      void buf
00001209      fgets(buf: &buf, n: 0x80, fp: stdin)
00001218      uint64_t rax_2 = strlen(&buf)
00001243      if (rax_2 != 0 && *(&buf + rax_2 - 1) == 0xa)
00001250          *(&buf + rax_2 - 1) = 0
00001270      if (strcmp("SSM{Annon_edhellen_edro_hi_ammen…", &buf) != 0)
00001287          puts(str: "Nothing happens.")
00001279      else
00001279          puts(str: "The doors open! You can now ente…")
0000129e      if (rax == *(fsbase + 0x28))
000012a6          return 0
000012a0      __stack_chk_fail()
000012a0      noreturn
```

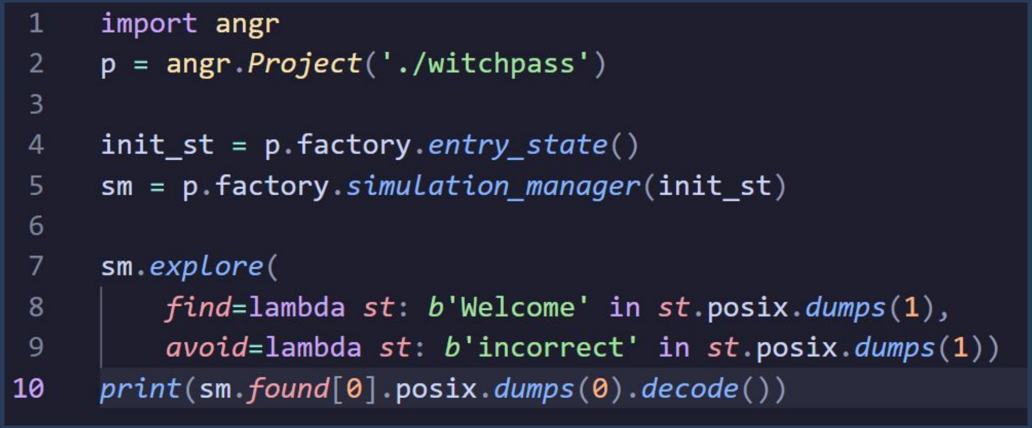
Practice!

Try to solve “witchpass” from the challenge handout

Note “scaffold.py”, which contains some of the boilerplate you might want to use

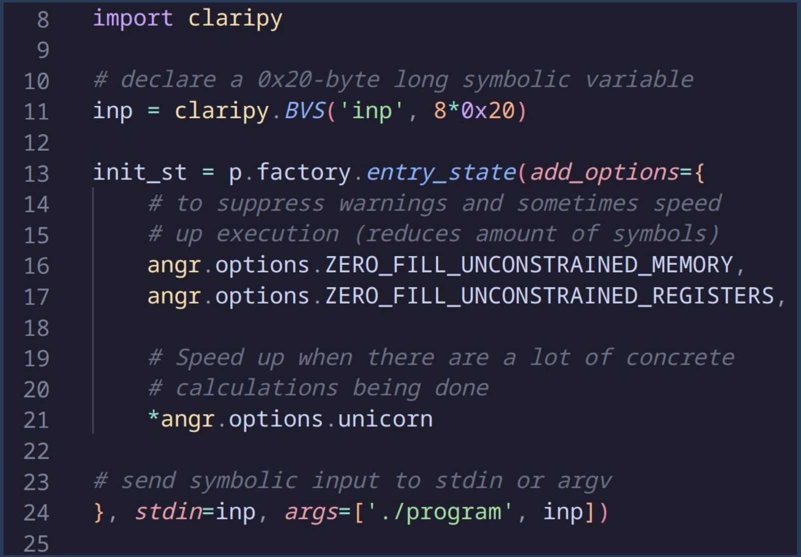
Install with `pip install angr`

“witchpass” solution

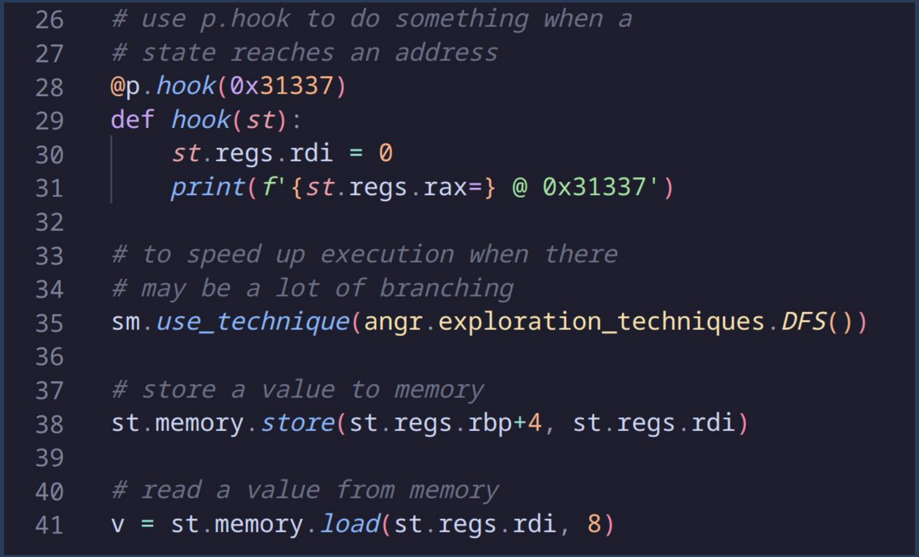


```python
import angr
p = angr.Project('./witchpass')

init_st = p.factory.entry_state()
sm = p.factory.simulation_manager(init_st)

sm.explore(
    find=lambda st: b'Welcome' in st.posix.dumps(1),
    avoid=lambda st: b'incorrect' in st.posix.dumps(1))
print(sm.found[0].posix.dumps(0).decode())
```

Tips & Tricks



```python
import claripy

# declare a 0x20-byte long symbolic variable
inp = claripy.BVS('inp', 8*0x20)

init_st = p.factory.entry_state(add_options={
    # to suppress warnings and sometimes speed
    # up execution (reduces amount of symbols)
    angr.options.ZERO_FILL_UNCONSTRAINED_MEMORY,
    angr.options.ZERO_FILL_UNCONSTRAINED_REGISTERS,

    # Speed up when there are a lot of concrete
    # calculations being done
    *angr.options.unicorn

# send symbolic input to stdin or argv
}, stdin=inp, args=['./program', inp])

```



```python
# use p.hook to do something when a
# state reaches an address
@p.hook(0x31337)
def hook(st):
    st.regs.rdi = 0
    print(f'{st.regs.rax=} @ 0x31337')

# to speed up execution when there
# may be a lot of branching
sm.use_technique(angr.exploration_techniques.DFS())

# store a value to memory
st.memory.store(st.regs.rbp+4, st.regs.rdi)

# read a value from memory
v = st.memory.load(st.regs.rdi, 8)
```