angr / angr-doc

Branch: master angr-doc / docs / courses / step0-basic_symbol_execution.md Find file Copy path

dukebarman Fix link to symbolic execution page 10bd3c7 on Mar 27

4 contributors

80 lines (62 sloc) 2.73 KB

angr courses - Step 0 - Basic symbolic execution

The first thing you are going to do with angr is executing symbolicaly your program. As a reminder, you can check what symbolic execution is here.

The binary and source code for this course can be found here.

```
>>> import angr

# We load the binary in angr
>>> project = angr.Project('docs/courses/src/step0.bin')

# Let's make things more readable
>>> addr_main = 0x4004a6
>>> first_jmp = 0x4004b9
>>> endpoint = 0x4004d6
>>> first_branch_left = 0x4004bb
>>> first_branch_right = 0x4004c2
>>> second_branch_left = 0x4004ca
>>> second_branch_right = 0x4004d1


# We create a state so that angr starts at the beginning of the main function
>>> main_state = project.factory.blank_state(addr=addr_main)
>>> sm = project.factory.simgr(main_state)
>>> assert sm.active[0].addr == addr_main


# Our simulation manager hasn't done anything yet, so it only has one active state
# which address is main
# Let's step
# The simgr.step functions accepts different arguments to regulate
# the stepping. Here, let's try to step until we reach the first comparison
>>> sm.step(until=lambda pg: pg.active[0].addr >= first_jmp)


# We now have two active states. Each of them took a branch from the
# comparison and will progress independently from the other one
>>> print(sm)
>>> for i, s in enumerate(sm.active):
```

```
...     print 'Active state %d: %s' % (i, hex(s.addr))
>>> assert len(sm.active) == 2
>>> assert sm.active[0].addr == first_branch_left
>>> assert sm.active[1].addr == first_branch_right


# If we make the first step, it will continue until reaching the endpoint
# The other one, however, will reach another comparison and should
# split again
>>> sm.step()
>>> print(sm)
>>> for i, s in enumerate(sm.active):
...     print 'Active state %d: %s' % (i, hex(s.addr))
>>> assert len(sm.active) == 3
>>> assert sm.active[0].addr == endpoint
>>> assert sm.active[1].addr == second_branch_left
>>> assert sm.active[2].addr == second_branch_right


# Good, we now have three states
# - The two first states reached the endpoint, and became unconstrained, since
# we started executing directly at main function. We would have seen these 2 states
# if we had enabled save_unconstrained option of our SimulationManager.
# - The other one will have the same history thus stop stepping at the endpoint
>>> sm.step()
>>> print(sm)
>>> for i, s in enumerate(sm.active):
...     print 'Active state %d: %s' % (i, hex(s.addr))
>>> assert len(sm.active) == 1
>>> assert sm.active[0].addr == endpoint


# The same effect can be done by using simgr.explore()
# The explorer will step every state until no more states are active
>>> sm = project.factory.simgr(main_state)
>>> sm.explore()
>>> assert len(sm.active) == 0
```