Manticore Documentation

Release 0.1.0

Trail of Bits

Sep 06, 2017

Contents:

Manticore is a prototyping tool for dynamic binary analysis, with support for symbolic execution, taint analysis, and binary instrumentation.

CHAPTER 1

API

This API is under active development, and should be considered unstable.

Helpers

manticore.**issymbolic**(*value*)

Helper to determine whether an object is symbolic (e.g checking if data read from memory is symbolic)

Parameters **value** (*object*) – object to check

Returns whether *value* is symbolic

Return type bool

Manticore

class manticore.**Manticore**(*binary_path*, *args=None*)

The central analysis object.

Parameters

- **binary_path** (*str*) – Path to binary to analyze
- **args** (*list[str]*) – Arguments to provide to binary

add_hook(*pc*, *callback*)

Add a callback to be invoked on executing a program counter. Pass *None* for pc to invoke callback on every instruction. *callback* should be a callable that takes one `State` argument.

Parameters

- **pc** (*int or None*) – Address of instruction to hook
- **callback** (*callable*) – Hook function

hook(*pc*)
A decorator used to register a hook function for a given instruction address. Equivalent to calling `add_hook()`.

Parameters **pc** (*int or None*) – Address of instruction to hook

run(*timeout=0*)
Runs analysis.

terminate()
Gracefully terminate the currently-executing run. Typically called from within a `hook()`.

verbosity
Convenience interface for setting logging verbosity to one of several predefined logging presets. Valid values: 0-5

State

class manticore.core.state.**State**(*constraints*, *model*)
Representation of a unique program state/path.

Parameters

- **constraints** (*ConstraintSet*) – Initial constraints on state
- **model** (*Decree or Linux or Windows*) – Initial constraints on state

abandon()
Abandon the currently-active state.

Note: This must be called from the Executor loop, or a *hook()*.

constrain(*constraint*)
Constrain state.

Parameters **constraint** (*manticore.core.smtlib.Bool*) – Constraint to add

new_symbolic_buffer(*nbytes*, ***options*)
Create and return a symbolic buffer of length *nbytes*. The buffer is not written into State's memory; write it to the state's memory to introduce it into the program state.

Parameters

- **nbytes** (*int*) – Length of the new buffer
- **name** (*str*) – (keyword arg only) The name to assign to the buffer
- **cstring** (*bool*) – (keyword arg only) Whether or not to enforce that the buffer is a cstring (i.e. no bytes, except for the last byte). (bool)

Returns `Expression` representing the buffer.

new_symbolic_value(*nbits*, *label='val'*, *taint=frozenset([])*)
Create and return a symbolic value that is *nbits* bits wide. Assign the value to a register or write it into the address space to introduce it into the program state.

Parameters

- **nbits** (*int*) – The bitwidth of the value returned
- **label** (*str*) – The label to assign to the value
- **taint** (*tuple or frozenset*) – Taint identifier of this value

Returns `Expression` representing the value

solve_n(*expr*, *nsolves=1*, *policy='minmax'*)
Concretize a symbolic `Expression` into *nsolves* solutions.

Parameters **expr** (*manticore.core.smtlib.Expression*) – Symbolic value to concretize

Returns Concrete value

Return type list[int]

solve_one(*expr*)
Concretize a symbolic `Expression` into one solution.

Parameters **expr** (*manticore.core.smtlib.Expression*) – Symbolic value to concretize

Returns Concrete value

Return type int

symbolicate_buffer(*data*, *label='INPUT'*, *wildcard='+'*, *string=False*)
Mark parts of a buffer as symbolic (demarked by the wildcard byte)

Parameters

- **data** (*str*) – The string to symbolicate. If no wildcard bytes are provided, this is the identity function on the first argument.
- **label** (*str*) – The label to assign to the value
- **wildcard** (*str*) – The byte that is considered a wildcard
- **string** (*bool*) – Ensure bytes returned can not be

Returns If data does not contain any wildcard bytes, data itself. Otherwise, a list of values derived from data. Non-wildcard bytes are kept as is, wildcard bytes are replaced by Expression objects.

Cpu

class manticore.core.cpu.abstractcpu.**Cpu**(*regfile*, *memory*)
Base class for all Cpu architectures. Functionality common to all architectures (and expected from users of a Cpu) should be here. Commonly used by models and py:class:manticore.core.Executor

The following attributes need to be defined in any derived class

•arch

•mode

•max_instr_width

•address_bit_size

•pc_alias

•stack_alias

all_registers
Returns all register names for this CPU. Any register returned can be accessed via a *cpu.REG* convenience interface (e.g. *cpu.EAX*) for both reading and writing.

Returns valid register names

Return type tuple[str]

read_bytes(*where*, *size*)
Read from memory.

Parameters

- **where** (*int*) – address to read data from
- **size** (*int*) – number of bytes

Returns data

Return type list[int or Expression]

read_int(*where*, *size=None*)
Reads int from memory

Parameters

- **where** (*int*) – address to read from
- **size** – number of bits to read

Returns the value read

Return type int or BitVec

read_register(*register*)
Dynamic interface for reading cpu registers

Parameters **register** (*str*) – register name (as listed in *self.all_registers*)

Returns register value

:rtype int or long or Expression

write_bytes(*where*, *data*)
Write a concrete or symbolic (or mixed) buffer to memory

Parameters

- **where** (*int*) – address to write to
- **data** (*str or list*) – data to write

write_int(*where*, *expr*, *size=None*)
Writes int to memory

Parameters

- **where** (*int*) – address to write to
- **expr** (*int or BitVec*) – value to write
- **size** – bit size of *expr*

write_register(*register*, *value*)
Dynamic interface for writing cpu registers

Parameters

- **register** (*str*) – register name (as listed in *self.all_registers*)
- **value** (*int or long or Expression*) – register value

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index