Qiling Framework: Symex and further

Sept 2021 R2CON

Ziqiao Kong (@Lazymio)

twitter: @pwnedmio https://github.com/wtdcode

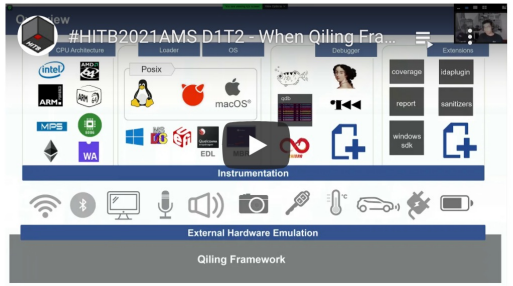
Fast Recap

Fast Recap

HITBSecConf

HOME LIVESTREAM TRAINING CONFERENCE COMMSEC A

When Qiling Framework Meets Symbolic Execution

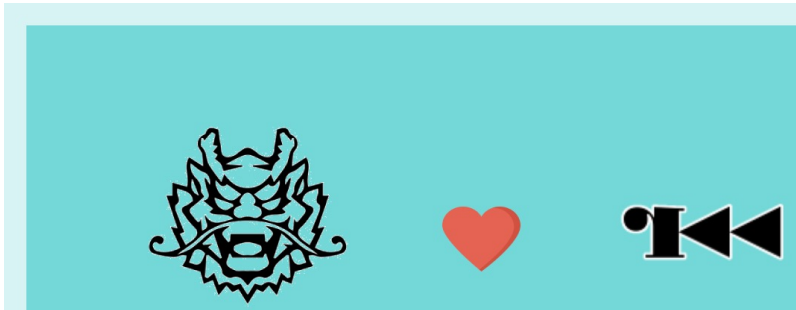


LOCATION: **Track 2**
DATE: **May 27, 2021**
TIME: **2:30 pm - 3:30 pm**


KAIJERN LAU


KONG ZIQIAO


WU CHENXU

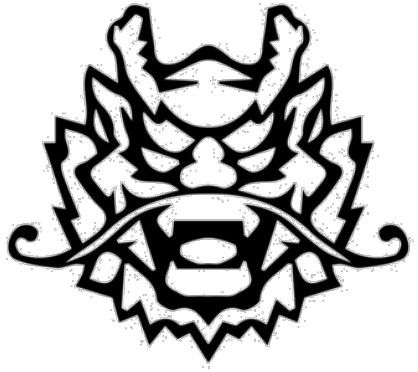


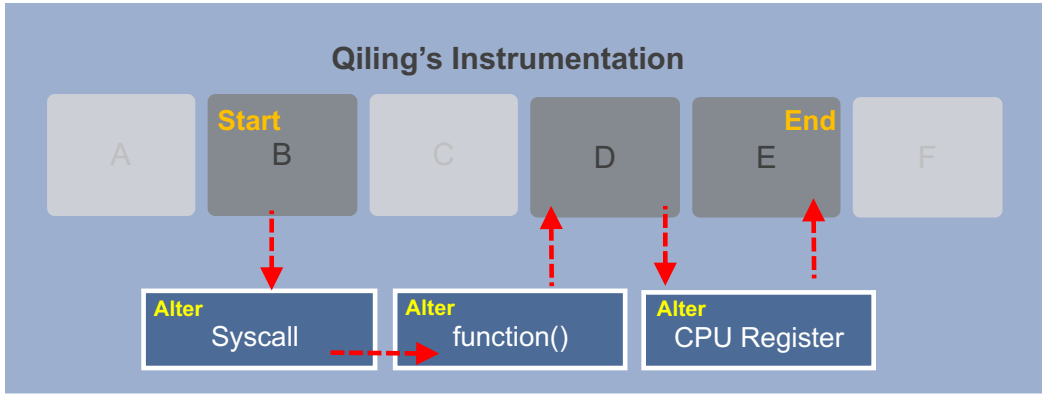
HITB AMS2021

Today: Some interesting details

Recap: Qiling

- Qiling provides almost the best dynamic instrumentation experience
  - The system emulation.
  - Flexible hooks && snapshots.
  - Full control of the sandbox.
- What's Next?
  - Symbolic execution
- Stand on the shoulders of giants.
  - But the FREE ones. (No idapython)
  - Radare2 is the best alternative.





Recap: R2

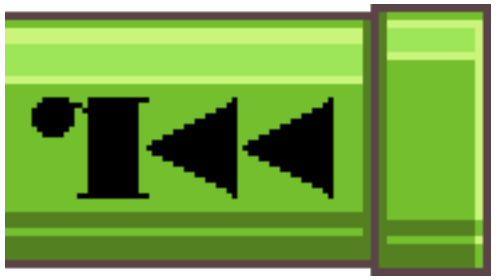
- Swiss-knife of the reverse engineering.
  - With almost the steepest learning curve. ;)
  - Follow the UNIX philosophy.
  - Source is your best friend.
- Find almost everything you need for security analysis.
  - Disassembly.
  - Control flow graph.
  - Debugging.
  - Tons of utilities, ? <int> is my favorite.
- What we focus on: ESIL.
  - Evaluable Strings Intermediate Language.
  - Reverse polish notation.
  - Designed for interpretation and suitable for symbolic execution.

```
[0x00001189]> ? 16
int32   16
uint32  16
hex     0x10
octal   020
unit    16
segment 0000:0010
string  "\x10"
fvalue: 16.0
float:  0.000000f
double: 0.000000
binary  0b00010000
ternary 0t121
[0x00001189]>
```

R2libr(former pyr2)

r2pipe vs rlang

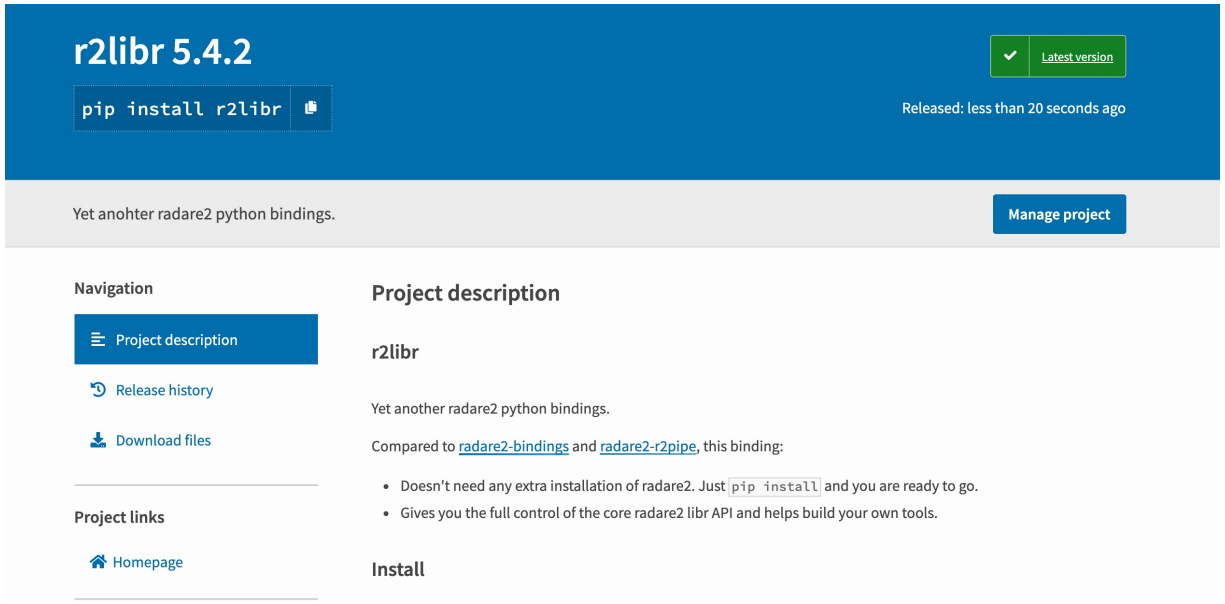
- r2pipe was the only available python bindings at that time.
  - It requires radare2 being installed system-wide.
  - We hope to minimize the Qiling installation to `pip install`.
  - Sometimes we would like to call the low-level API.
- rlang is the other way, running a python interpreter in R2.
  - Seems good but we expect to run standalone.
  - Still system-wide R2 installation is required.
- Let’s invent the wheel!



r2libr

- So, I wrote a brand-new python bindings for R2: r2libr.
- How it works?
  - R2 headers is clean enough to do auto-generation.
  - Bindings are generated automatically with ctypeslib and Github CI.

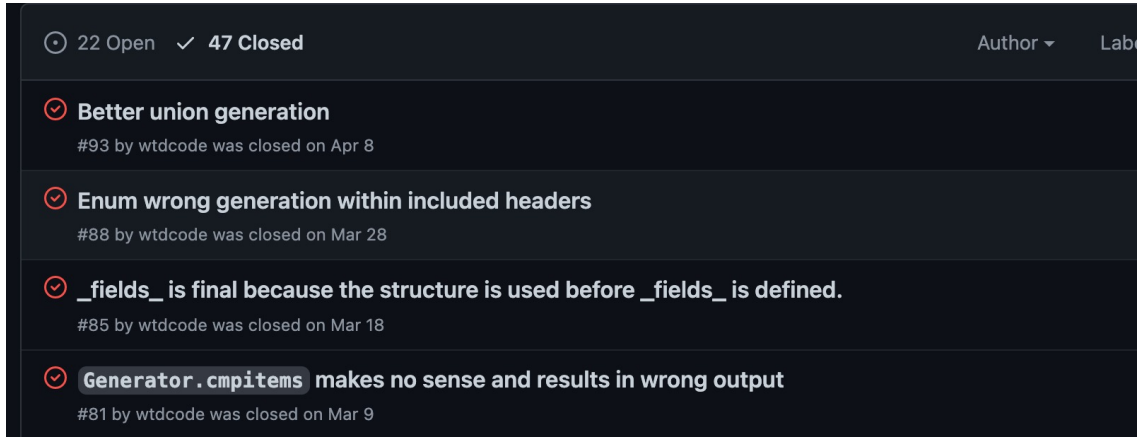
```
R_API bool r_core_init(RCore *core);
R_API void r_core_bind_cons(RCore *core); // to restore pointers in cons
R_API RCore *r_core_new(void);
R_API void r_core_free(RCore *core);
R_API void r_core_fini(RCore *c);
R_API void r_core_wait(RCore *core);
```




```
r_core_new = _libr_core.r_core_new
r_core_new.restype = ctypes.POINTER(struct_r_core_t)
r_core_new.argtypes = []
r_core_free = _libr_core.r_core_free
r_core_free.restype = None
r_core_free.argtypes = [ctypes.POINTER(struct_r_core_t)]
r_core_fini = _libr_core.r_core_fini
r_core_fini.restype = None
r_core_fini.argtypes = [ctypes.POINTER(struct_r_core_t)]
r_core_wait = _libr_core.r_core_wait
r_core_wait.restype = None
r_core_wait.argtypes = [ctypes.POINTER(struct_r_core_t)]
```

r2libr – How it works

- We have: C headers
- We expect: Python ctypes Bindings
- The tools: ctypeslib2
  - Backend by libclang
  - Not actively maintained, many bugs
- That's all?
  - No

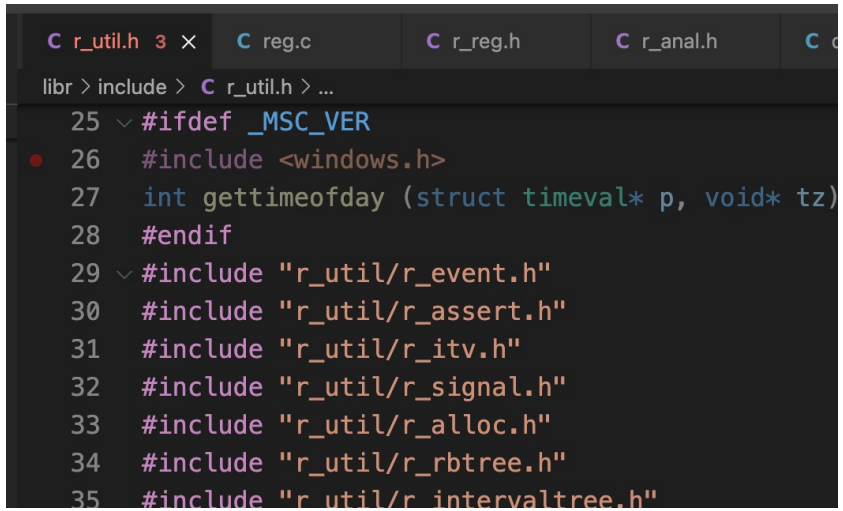



```c
struct s{
    int i;
    union {
        long l;
    };
};
```

```python
class struct_s(Structure):
    pass

class union_s_0(Union):
    _pack_ = 1 # source:False
    _fields_ = [
    ('l', ctypes.c_int64),
     ]

struct_s._pack_ = 1 # source:False
struct_s._anonymous_ = ( "s_0", )
struct_s._fields_ = [
    ('i', ctypes.c_int32),
    ('PADDING_0', ctypes.c_ubyte * 4),
    ('s_0', union_s_0),
]
```

r2libr – Black Magic

- r_util
  - Managed by multiple headers, no R_API
  - ctypeslib won't generated symbols in included headers
- Solution
  - Do expansion manually, like standard C preprocessor
  - Apply ctypeslib on the expanded headers




```python
# We have to expand r_util manually.
# Note that we don't need to expand headers deeper since we only focus on R_API.      You, 6 months ago • Moved from https://
# FIXME: Any better approach?
def expand_util(pargs):
    r_util_path = Path(pargs.build) / "include" / "libr" / "r_util.h"
    r_util_gen_path = Path(pargs.build) / "include" / "libr" / "r_util_gen.h"
    with open(r_util_path, "r+") as f:
        content = f.read()
    sub_util_headers = re.findall(r'\n#include "(r_util/r_.*.h)"', content)
    sub_util_headers.extend(re.findall(r"#include <(r_.*h)>", content))
    output_util = ""
    generated_headers = set()
    for ln in content.splitlines(keepends=True):
        headers = re.findall(r'^#include "(r_util/r_.*.h)"', ln)
        if len(headers) == 0:
            headers = re.findall(r"^#include <(r_.*h)>", ln)
        if len(headers) == 0 and "r_util/r_print.h" in ln:
            all_utils = set([ f"r_util/{util}" for util in os.listdir(Path(pargs.build) / "include" / "libr" / "r_util")])
            headers = list(all_utils.difference(generated_headers))
            print("Going to generate the following utils which are not included in r_util.h")
            print("\n".join(headers))
        if len(headers) == 0:
            output_util += ln
        else:
            for header in headers:
                with open(Path(pargs.build) / "include" / "libr" / header) as f:
                    output_util += f.read()
                    output_util += "\n"
                generated_headers.add(header)
    with open(r_util_gen_path, "w+") as f:
        f.write(output_util)

def handle_lib(lib, pargs):
    if lib == "util":
        expand_util(pargs)
        fpath = str(Path(pargs.build) / "include" / "libr" / f"r_util_gen.h")
```

r2libr – Still Black Magic

- Put everything together
  - Unfortunately, ctypeslib is designed to output a single script
  - But we would like a library, so we have to put everything together
- Solution
  - Mangle the output by regex, dirty but it works.
  - Gives each r_* module a separate namespace
  - Remove the local path generated
  - Write a wrapper to load all r_* libs

```python
def post_handle(binding_content, lib_name):
    # Convert the lib reference to imported r2lib.
    # e.g.
    # _libraries['libr_core.so.5.2.0-git'] => _libr_core
    for _lib in libs:
        binding_content = binding_content.replace(f"_libraries['{libs_path[_lib].name}']", f"_libr_{_lib}")
    # Import all r2libs.
    binding_content = binding_content.replace("import ctypes", "import ctypes\n" + "\n".join([f"from .r_libs import r_{_lib} as _libr_{_lib}" for _lib in libs]))
    # Remove the redundant assignment
    # e.g.
    # _libr_core = ctypes.CDLL('/path/to/libr_core.so.5.2.0-git')
    for _lib in libs:
        binding_content = re.sub(rf".*ctypes.CDLL.*{libs_path[_lib].name}.*\n", "", binding_content)
    # Remove clang2py args in comments.
    # e.g.
    # TARGET arch is: ['arg1', 'arg2']
    binding_content = re.sub(rf".*TARGET arch is.*\n", "", binding_content)

    return binding_content
```

r2libr – Last Black Magic

- Meson
  - We would like cross-platform so meson seems good
  - But it has critical bug on macOS, see https://github.com/mesonbuild/meson/issues/2121
  - Basically, you can't use relative path to refer dylibs on meson built libraries
- Solution
  - Black magic: Parse the libraries and rewrite dyld_path

```
# Meson seems not to play well with macOS and radare2
# assumes that users will install the whole project to system.
# So we have to rewite all lib load path for our bindings.
# Refs:
# - https://github.com/mesonbuild/meson/issues/2121
def rewrite_dyld_path(dylib: Path):
    def _read_until_zero(fp):
        cur = fp.tell()
        s = b""
        ch = fp.read(1)
        while ch != b'\x00' and ch != b'':
            s += ch
            ch = fp.read(1)
        fp.seek(cur, 0)
        return s.decode("utf-8")
    def _parse_libr_name(path):
        result = re.findall(r"(libr_.*\.dylib$)", path)
        if len(result) == 0:
            return None
        else:
            return result[0]
    def _verbose_call(*args, **kwargs):
        print(f"Calling: {args[0]}")
        return subprocess.call(*args, **kwargs)
    with open(dylib, "rb+") as f:
        magic = f.read(4)
        if magic != b'\xcf\xfa\xed\xfe':
            return
        _, _, _, load_num = struct.unpack("<IIII", f.read(16))

        # Skip file header
        f.seek(0x20, 0)
```

r2libr – Finally

```
In [16]: import libr

In [17]: r_core = libr.r_core.r_core_new()

In [18]: import ctypes

In [19]: print(ctypes.string_at(libr.r_core.r_core_cmd_str(r_core, ctypes.create_string_buffer("??????".encode("utf-8")))).d
    ...: ecode("utf-8"))
  _
 0 0  <  What are you doing?
 |_|

In [20]:
```

Some details of Symex

Symbolic Execution: Difficulties

- Implementation is much more complex than expected
  - R2 itself doesn’t have memory R/W implemented, so we have to do it own.
  - Also we can’t use R2 registers implementation due to symbolic values.
  - As a result, we did a full re-implementation of ESIL.
  - May become another emulation engine to replace Unicorn.
- Symex is never a silver bullet.
  - Really slow since we have to keep each state immutable.
  - State explosion for complex function and make it unacceptable slow.

Symbolic Execution: Memory Implementation

- Memory
  - Could hold symbolic values
  - Support full clone since it's immutable
  - Support mmap and unmmap
  - Managed by segments
  - Really complex to emulate a memory

```
# Insert
def __insert_seg(self, seg1: ESILMemorySegment):
    while True:
        # Iterate seg1.
        idx, seg2 = self.__search_seg(seg1.end)
        if seg2 is None:
            # In the front of the all segs.
            # Note: Adjacent segs list [seg1][seg2] goes to the next if.
            self._segs.insert(0, seg1)
            break
        elif seg2.end >= seg1.end: # Note we are searching seg1.end instead of seg1.end - 1
            if seg2.start >= seg1.start:
                # Overlapped
                #      [s    e    g    1]
                #                   [s   e   g   2]
                # Result
                #      [s        e         g       1]
                content = seg1.content + seg2.content[(seg1.end - seg2.start):]
                new_seg = ESILMemorySegment(seg1.start, content)
                seg1 = new_seg
                del self._segs[idx]
            else:
                # Overlapped
                #      [s    e    g    1]
                #    [s       e       g       2]
                # Result
                #    [s        e         g       1]
                content = seg2.content[:(seg1.start - seg2.start)] + seg1.content + seg2.content[(seg1.end - seg2.start):]
                new_seg = ESILMemorySegment(seg2.start, content)
                self._segs[idx] = new_seg
                break
        elif seg2.end >= seg1.start:
            if seg2.start >= seg1.start:
```

Symbolic Execution: Register

- Register
  - Again, could hold symbolic values
  - Support full clone since it’s immutable
  - Much simpler than memory

```
# Insert
def __insert_seg(self, seg1: ESILMemorySegment):
    while True:
        # Iterate seg1.
        idx, seg2 = self.__search_seg(seg1.end)
        if seg2 is None:
            # In the front of the all segs.
            # Note: Adjacent segs list [seg1][seg2] goes to the next if.
            self._segs.insert(0, seg1)
            break
        elif seg2.end >= seg1.end: # Note we are searching seg1.end instead of seg1.end - 1
            if seg2.start >= seg1.start:
                # Overlapped
                #      [s    e    g    1]
                #                   [s   e   g   2]
                # Result
                #      [s        e         g       1]
                content = seg1.content + seg2.content[(seg1.end - seg2.start):]
                new_seg = ESILMemorySegment(seg1.start, content)
                seg1 = new_seg
                del self._segs[idx]
            else:
                # Overlapped
                #      [s    e    g    1]
                #    [s       e       g       2]
                # Result
                #    [s        e         g       1]
                content = seg2.content[:(seg1.start - seg2.start)] + seg1.content + seg2.content[(seg1.end - seg2.start):]
                new_seg = ESILMemorySegment(seg2.start, content)
                self._segs[idx] = new_seg
                break
        elif seg2.end >= seg1.start:
            if seg2.start >= seg1.start:
```

Symbolic Execution: ESIL

- ESIL
  - With our own memory and register implementation, we have to implement the ESIL interpreter by ourselves
  - Careful to handle symbolic values in branches
  - Search all possible states after a branch
  - ESILSolve is really a nice reference

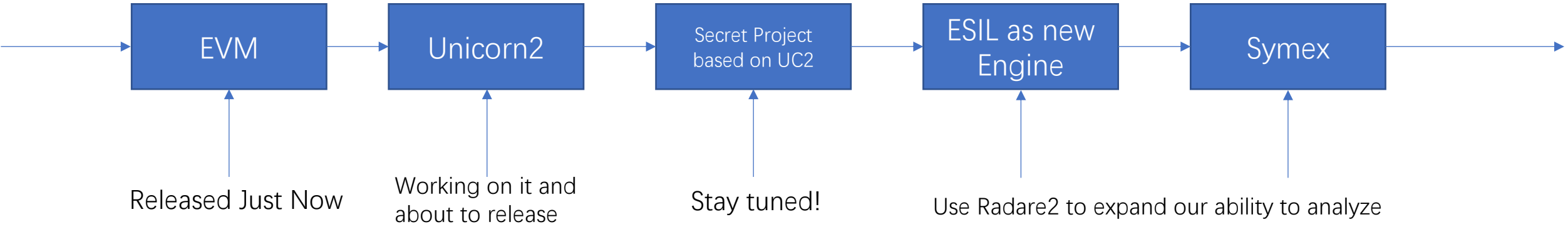
```
self._ops = {
    "$" : self.__esil_interrupt,
    "$z" : self.__esil_zf,
    "$c" : self.__esil_cf,
    "$b" : self.__esil_bf,
    "$p" : self.__esil_pf,
    "$s" : self.__esil_sf,
    "$o" : self.__esil_of,
    "$ds" : self.__esil_ds,
    "$jt" : self.__esil_jt,
    "$js" : self.__esil_js,
    "$r" : self.__esil_rs,
    "$$" : self.__esil_address,
    "~" : self.__esil_signext,
    "==" : self.__esil_cmp,
    "<" : self.__esil_smaller,
    ">" : self.__esil_bigger,
    "<=" : self.__esil_smaller_equal,
    ">=" : self.__esil_bigger_equal,
    "?{" : self.__esil_if,
    "<<" : self.__esil_lsl,
    "<<=" : self.__esil_lsleq,
    ">>" : self.__esil_lsr,
    ">>=" : self.__esil_lsreq,
    ">>>>" : self.__esil_asr,
    ">>>>=" : self.__esil_asreq,
    ">>>" : self.__esil_ror,
    "<<<" : self.__esil_rol,
    "&" : self.__esil_and,
    "&=" : self.__esil_andeq,
    "}" : self.__esil_nop,
    "}{" : self.__esil_nop,
```

```
def __esil_if(self):
    if self._skip > 0:
        self._skip += 1
        return
    p_src = self._stack.pop()
    src = self.__get_param(p_src)

    if z3.is_bv_value(src):
        # Normal cases without any symbolic input.
        src = src.as_long()

        if src == 0:
            self._skip += 1

        # if self._conditions is None:
        #     self._conditions = src != 0
        # elif type(self._conditions) is bool:
        #     self._conditions = (src != 0 and self._conditions)
        # else:
        #     self._conditions = z3.And(z3.BoolVal(src != 0), self._conditions)
    else:
        # Firstly, we assume we force to execute the True branch.
        if self._conditions is None:
            self._conditions = (src != z3.BitVecVal(0, src.size()))
        else:
            self._conditions = z3.And(src != z3.BitVecVal(0, src.size()), self._conditions)
```

```
            for new_pc in pcs:
                new_state = self.clone()

                new_state.solver.add(pc == new_pc)
                # Set to the concrete PC
                new_state.reg.PC = new_pc
                new_states.append(new_state)
```

Future

Future Steps



Future steps

Credits

- Radare2 for the nice project. https://github.com/radareorg/radare2
- ctypeslib for r2libr implementation. https://github.com/trolldbois/ctypeslib
- ESILSolve for the implementation reference. https://github.com/radareorg/esilsolve
- angr for the design reference. https://github.com/angr/angr
- z3 for the excellent solver. https://github.com/Z3Prover/z3
- @pancake for the timely help. https://twitter.com/trufae