Qiling Framework: HITB 2021 AMS

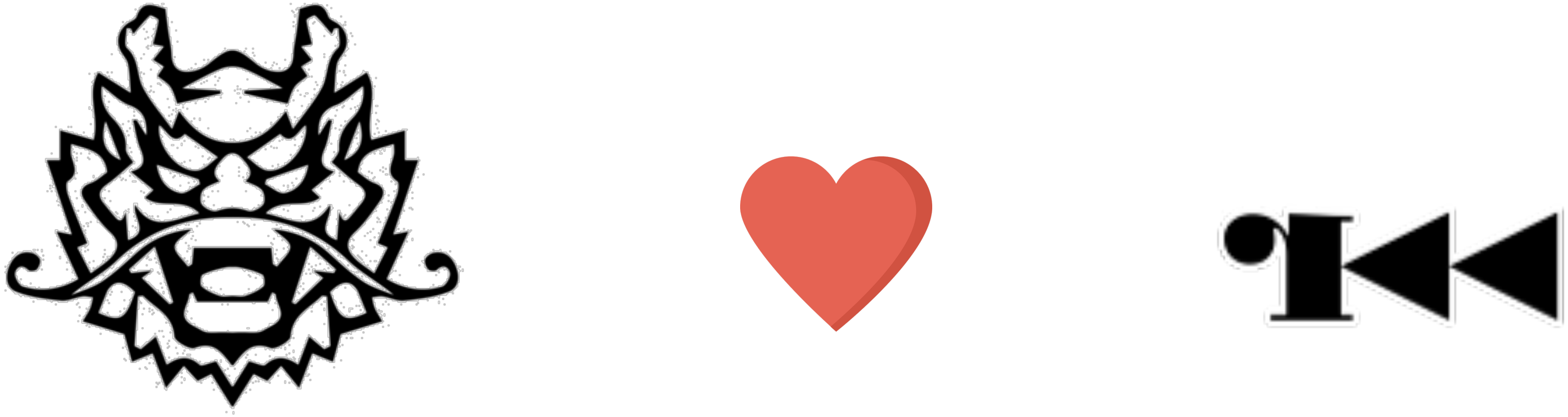
May 2021

Ziqiao Kong (@Lazymio)

twitter: @lazymio https://twitter.com/pwnedmio

Story

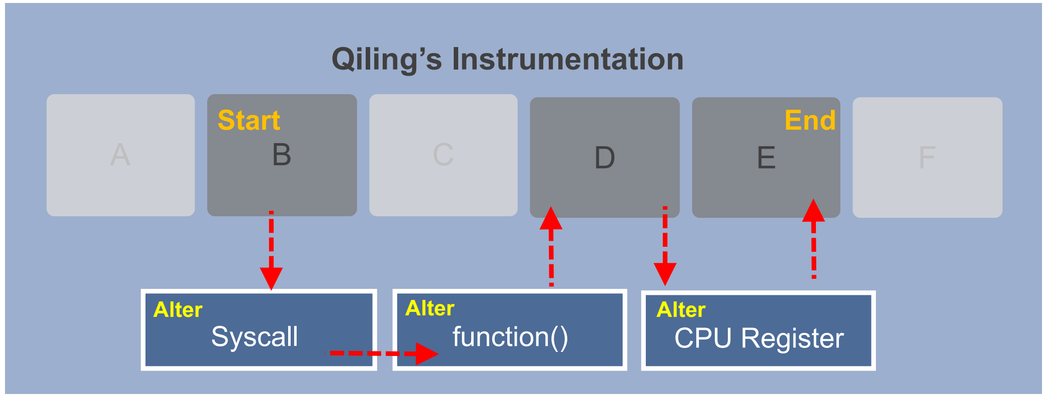
Story



When Qiling Meets Radare2

Why Radare2?

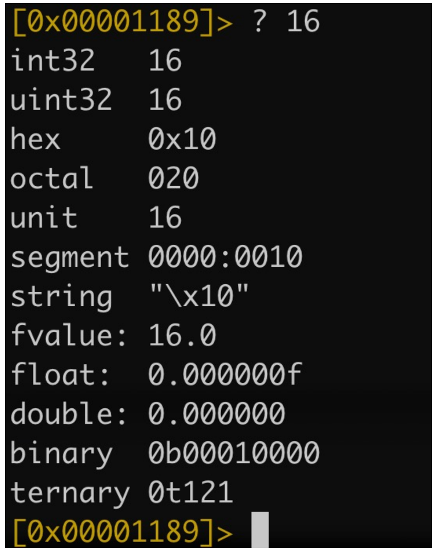
- Qiling provides almost the best dynamic instrumentation experience
  - The system emulation.
  - Flexible hooks && snapshots.
  - Full control of the sandbox.
- What's Next?
  - Static analysis.
  - Symbolic execution.
- Stand on the shoulders of giants.
  - But the FREE ones. (Yes, I hate idapython)
  - Radare2 is the best alternative.



Qiling's Instrumentation

Why Radare2? Cont.

- 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]>
```

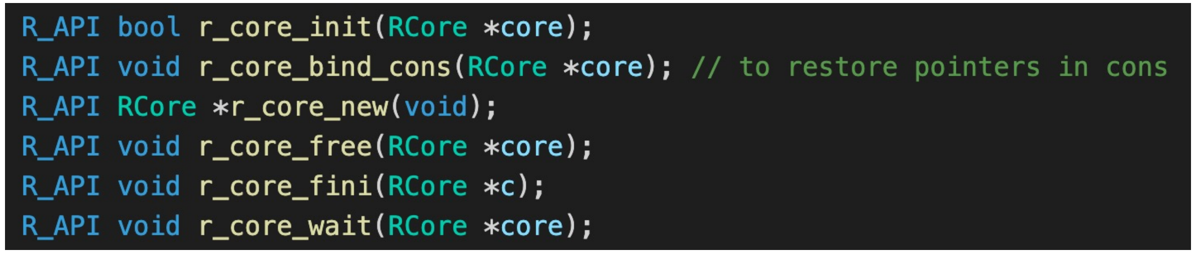
Integration

r2pipe vs rlang

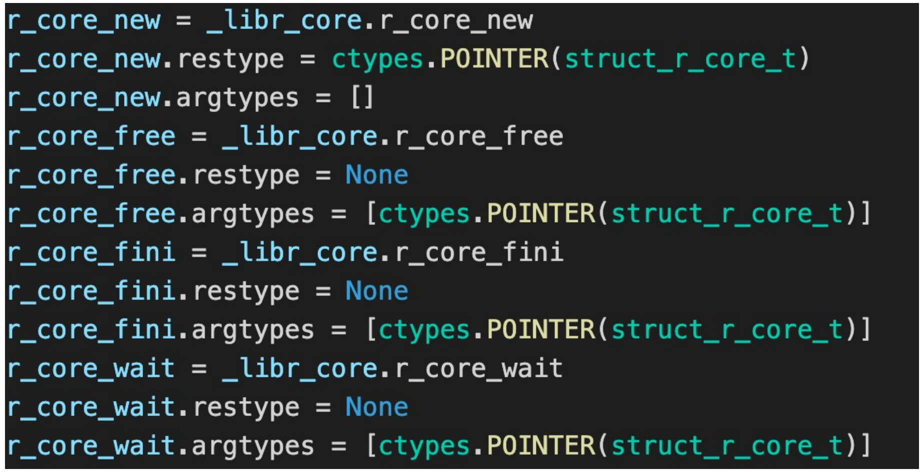
- r2pipe was the only available python bindings at that time.
    - It requires radare2 being installed system-wide.
    - We hope to minimize the Qiling installation to `pip install`.
    - Sometimes we would like to call the low-level API.
- rlang is the other way, running a python interpreter in R2.
    - 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.
- Have a try
  - pip install r2libr
  - That’s all, no need for any extra installation.
- Demo.
  - Execute “???????” by r2libr.
- Looks very verbose but we get low-level API.
  - Can be used to implement an r2pipe in minutes.



```
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)]
```

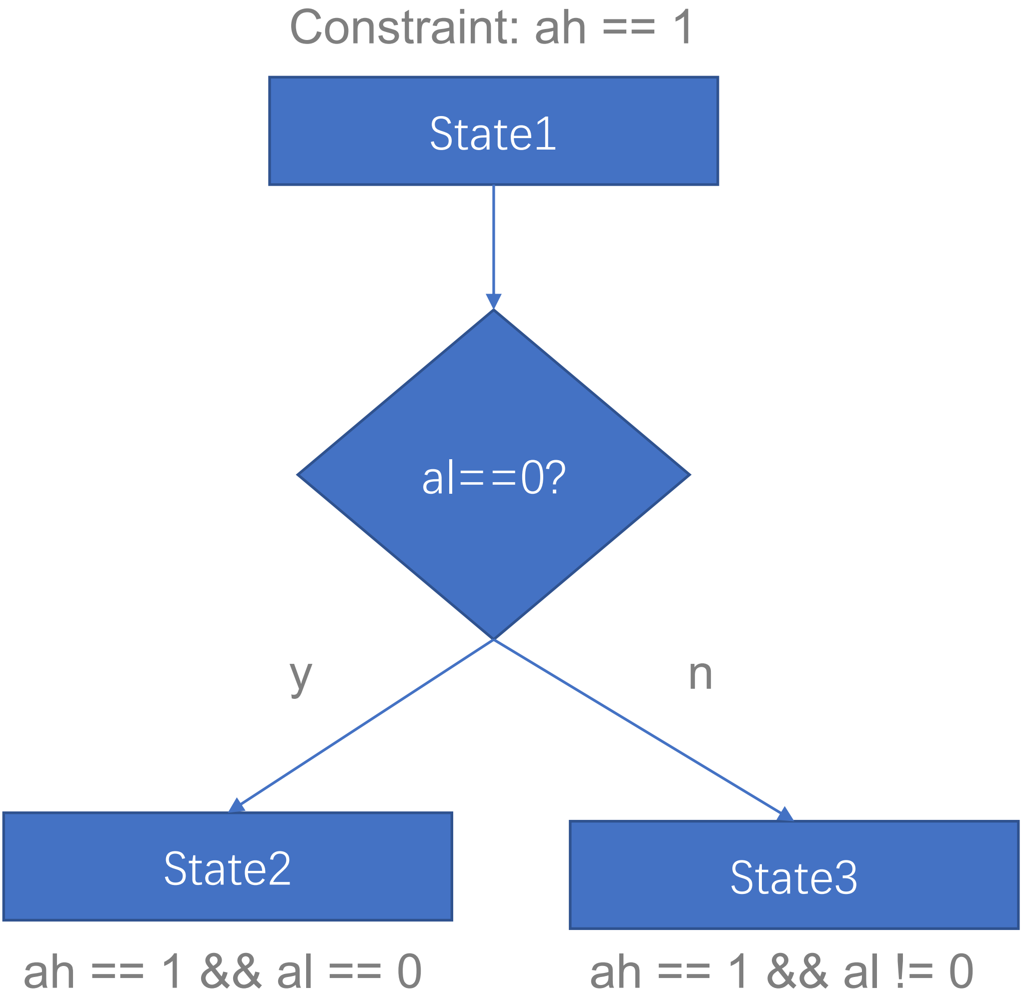
The Story of Symex Starts.

Symbolic Execution: Introduction

- Introduce symbolic execution in two lines.
  - You know x = 1, then x + 1 = 2.
  - You know x + 1 = 2, then x = 1, where x is our symbolic value.
- Essential of an intermediate language.
  - Reduce the large instruction set to micro-operations.
  - Easy to implement and instrument.
  - Cross-architectures.
- ESIL is a good choice.
  - Again, FREE.
  - Evaluable, and easy to interpret.
  - Short Demo by “ae”

Symbolic Execution: Details

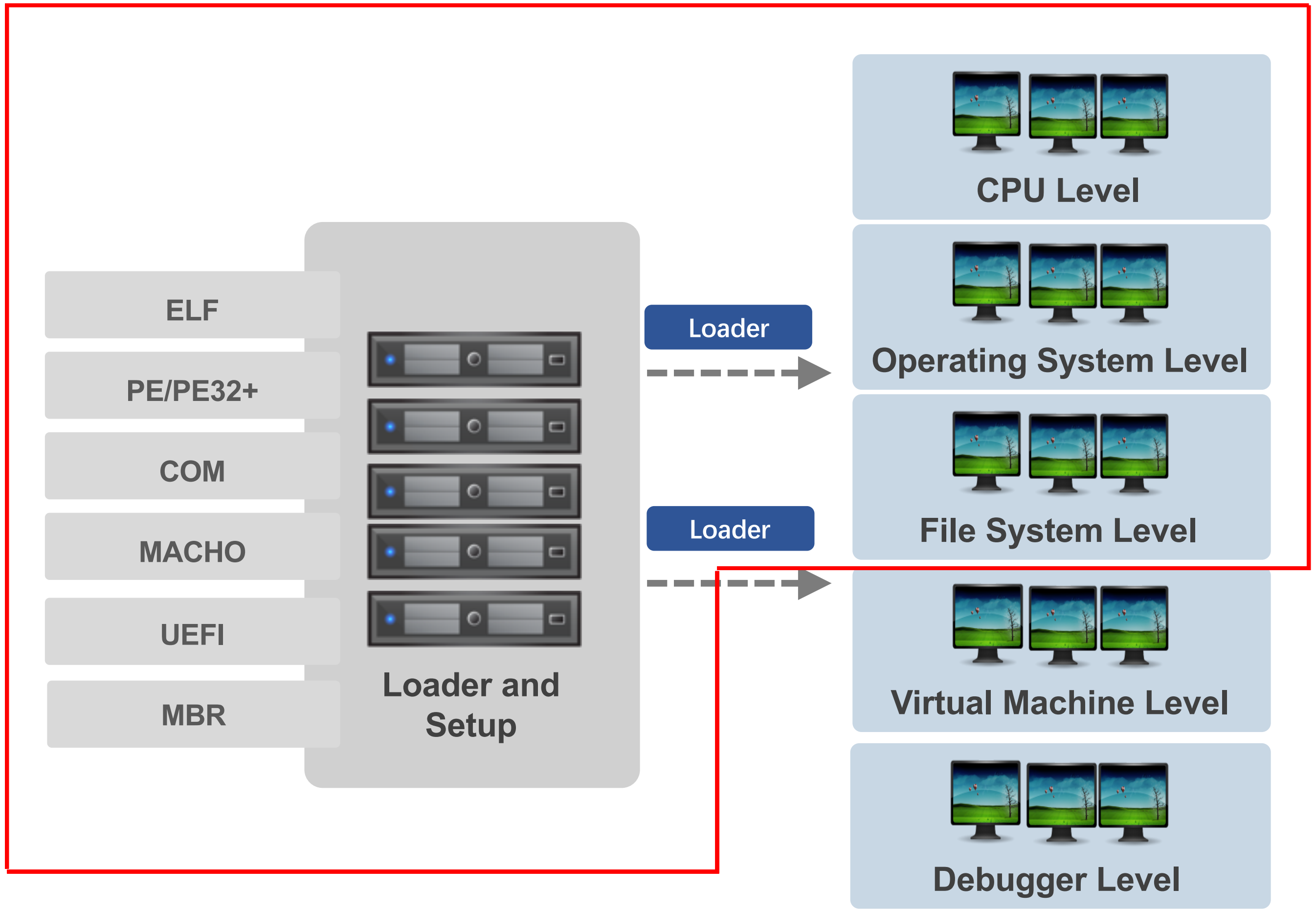
- The core concept: state.
  - Includes the full memory and registers at a specific time.
  - Includes the constraints to reach such state.
  - Should be immutable.
- General steps:
  - Input sym values.
  - Execute and gather constraints.
  - Reach some point and solve the constraints.
  - Evaluate the sym values.
- Example on the right fugure.
  - The state1 is forked to state2 and state3 after if statement.
  - State1 != State2 != State3
  - When the engine reaches the state we would like, say state3, we use an SAT solver (z3) to eval the value of ah or al.
- For large and complex algorithm, symex saves lots of time.



Note: ah and al may be symbolic values.

Symbolic Execution: Qiling

- Components reuse.
  - System emulation.
  - Binary load and memory setup.
- Our Goal: User can switch the underlying engine while keeping the high-level API unchanged.



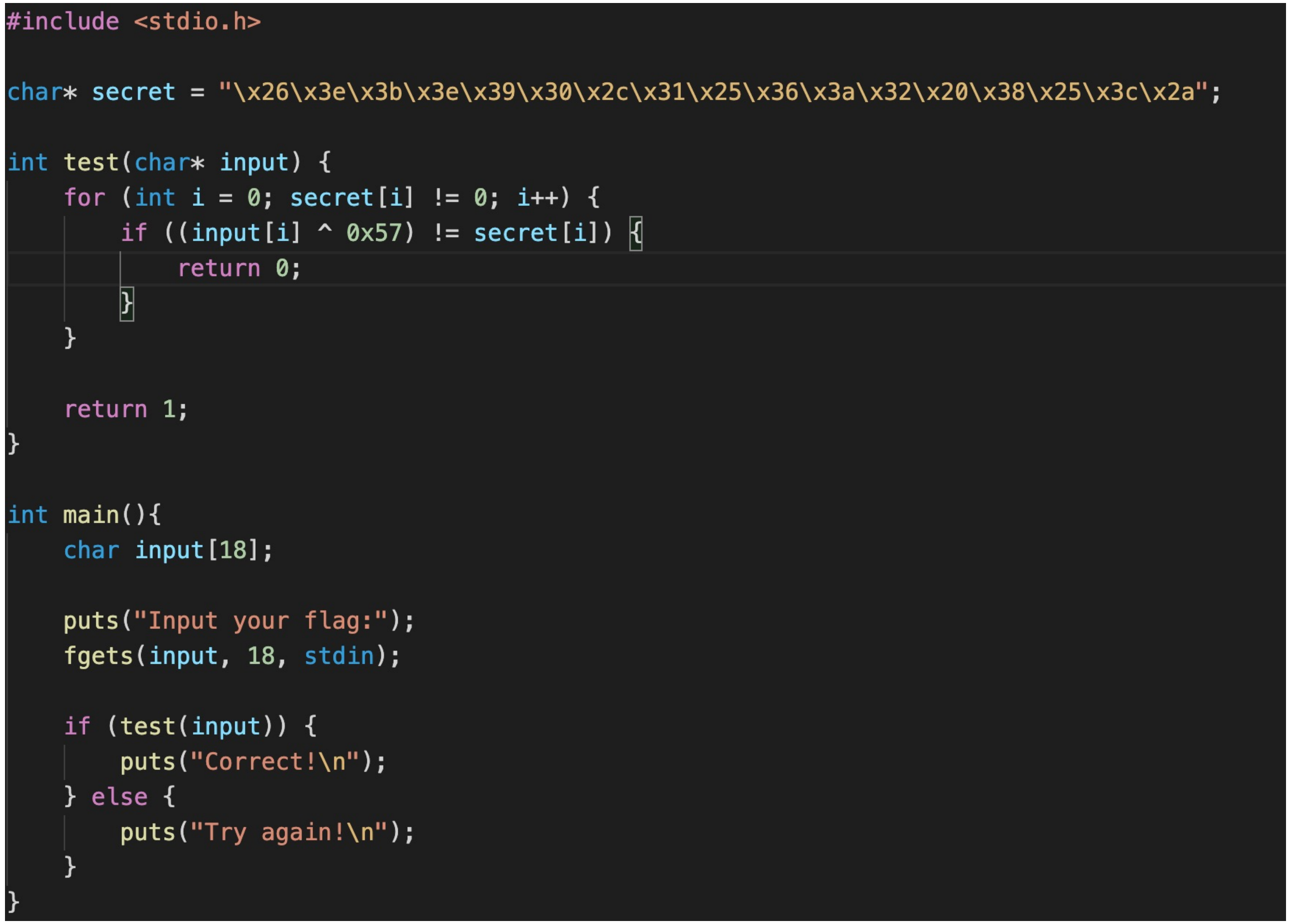
Symbolic Execution: Difficulties

- 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.

Demo: A simple crackme.

Source

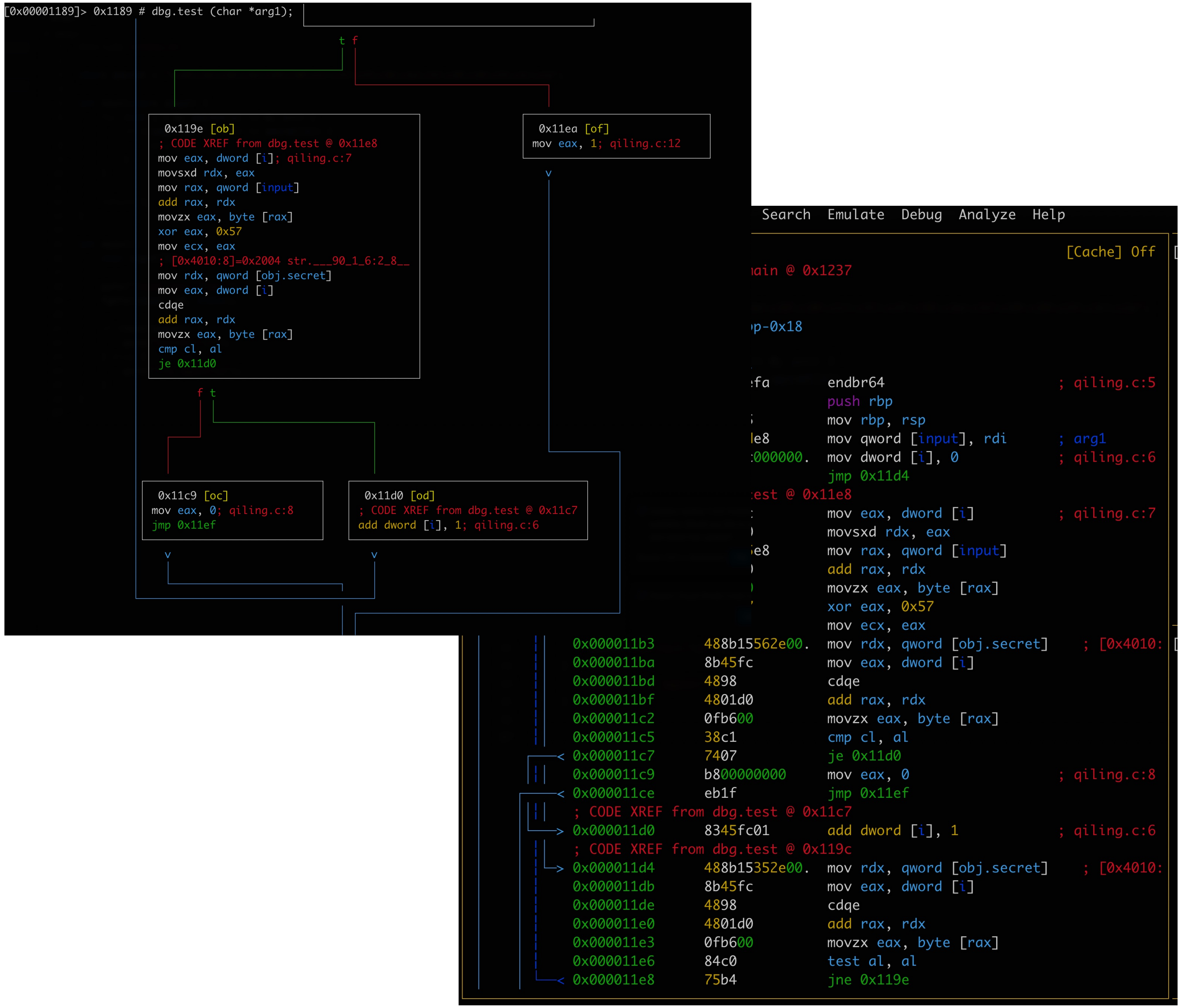
- An extreme simple crackme.
  - Input is xor-ed with 0x57.
  - A verification function.



```c
#include <stdio.h>

char* secret = "\x26\x3e\x3b\x3e\x39\x30\x2c\x31\x25\x36\x3a\x32\x20\x38\x25\x3c\x2a";

int test(char* input) {
    for (int i = 0; secret[i] != 0; i++) {
        if ((input[i] ^ 0x57) != secret[i]) {
            return 0;
        }
    }

    return 1;
}

int main(){
    char input[18];

    puts("Input your flag:");
    fgets(input, 18, stdin);

    if (test(input)) {
        puts("Correct!\n");
    } else {
        puts("Try again!\n");
    }
}
```

Analysis

- R2 Visual Mode

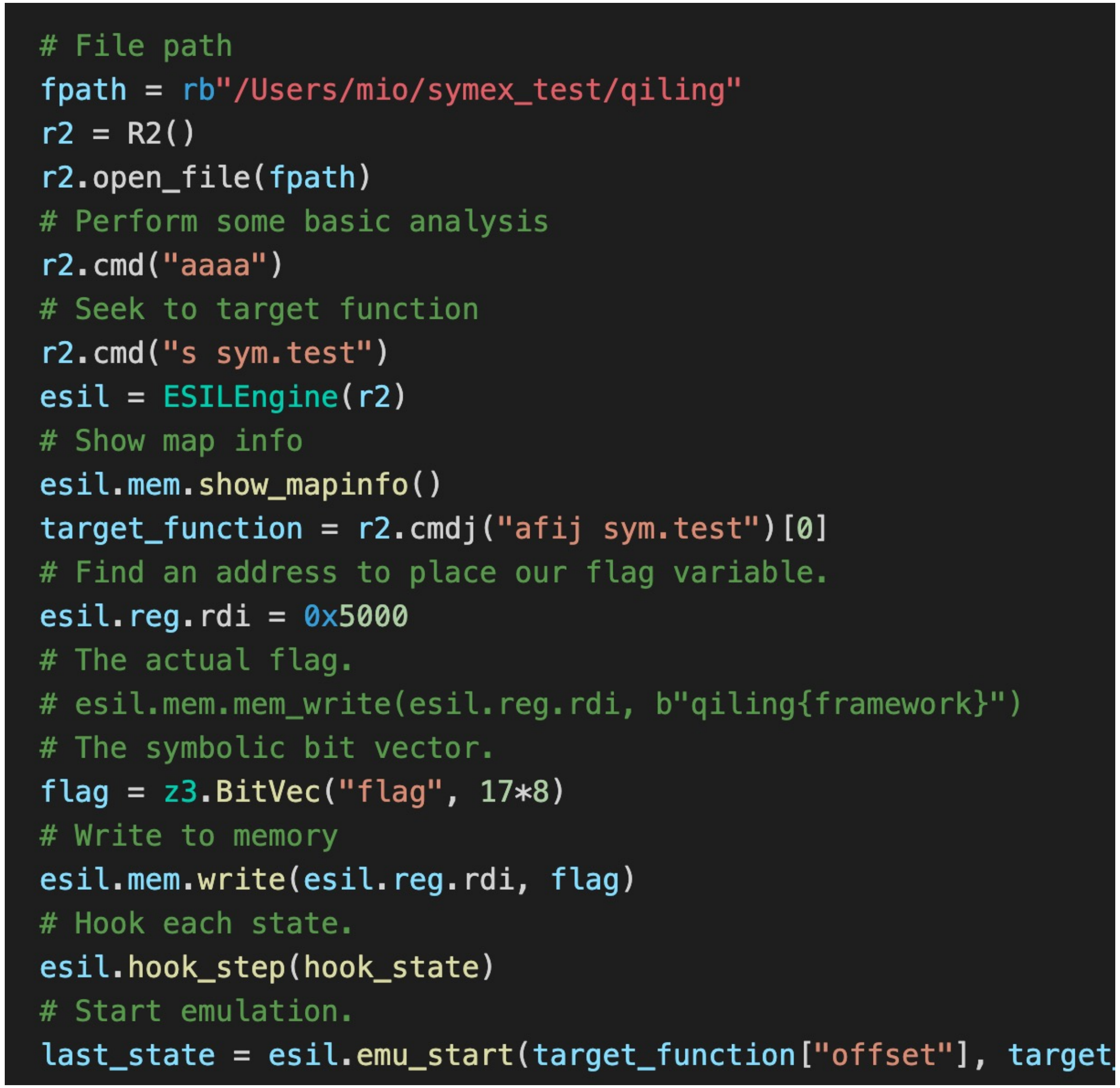


```
[0x00001189]> 0x1189 # dbg.test (char *arg1);

                         t f

 0x119e [ob]                                          0x11ea [of]
; CODE XREF from dbg.test @ 0x11e8                    mov eax, 1; qiling.c:12
mov eax, dword [i]; qiling.c:7
movsxd rdx, eax
mov rax, qword [input]
add rax, rdx
movzx eax, byte [rax]
xor eax, 0x57
mov ecx, eax
; [0x4010:8]=0x2004 str.___90_1_6:2_8__
mov rdx, qword [obj.secret]
mov eax, dword [i]
cdqe
add rax, rdx
movzx eax, byte [rax]
cmp cl, al
je 0x11d0

      f t

 0x11c9 [oc]                      0x11d0 [od]
mov eax, 0; qiling.c:8            ; CODE XREF from dbg.test @ 0x11c7
jmp 0x11ef                        add dword [i], 1; qiling.c:6
```

```
                                                      [Cache] Off
ain @ 0x1237

p-0x18

:fa             endbr64                          ; qiling.c:5
                push rbp
;               mov rbp, rsp
le8             mov qword [input], rdi           ; arg1
:000000.        mov dword [i], 0                 ; qiling.c:6
                jmp 0x11d4
est @ 0x11e8
:               mov eax, dword [i]               ; qiling.c:7
)               movsxd rdx, eax
e8              mov rax, qword [input]
)               add rax, rdx
)               movzx eax, byte [rax]
)               xor eax, 0x57
                mov ecx, eax
0x000011b3      488b15562e00.   mov rdx, qword [obj.secret]    ; [0x4010:
0x000011ba      8b45fc          mov eax, dword [i]
0x000011bd      4898            cdqe
0x000011bf      4801d0          add rax, rdx
0x000011c2      0fb600          movzx eax, byte [rax]
0x000011c5      38c1            cmp cl, al
0x000011c7      7407            je 0x11d0
0x000011c9      b800000000      mov eax, 0                     ; qiling.c:8
0x000011ce      eb1f            jmp 0x11ef
; CODE XREF from dbg.test @ 0x11c7
0x000011d0      8345fc01        add dword [i], 1               ; qiling.c:6
; CODE XREF from dbg.test @ 0x119c
0x000011d4      488b15352e00.   mov rdx, qword [obj.secret]    ; [0x4010:
0x000011db      8b45fc          mov eax, dword [i]
0x000011de      4898            cdqe
0x000011e0      4801d0          add rax, rdx
0x000011e3      0fb600          movzx eax, byte [rax]
0x000011e6      84c0            test al, al
0x000011e8      75b4            jne 0x119e
```

Solve

- Familiar API design like current Qiling API.
  - esil.mem.read/write
  - esil.reg.rax = 1
  - esil.hook_state
  - esil.mem.show_mapinfo
- They would be put under ql namespace after integration like:
  - ql.mem.read/write
  - ql.reg.rax = 1
  - ql.hook_state
  - ql.mem.show_mapinfo



```
# File path
fpath = rb"/Users/mio/symex_test/qiling"
r2 = R2()
r2.open_file(fpath)
# Perform some basic analysis
r2.cmd("aaaa")
# Seek to target function
r2.cmd("s sym.test")
esil = ESILEngine(r2)
# Show map info
esil.mem.show_mapinfo()
target_function = r2.cmdj("afij sym.test")[0]
# Find an address to place our flag variable.
esil.reg.rdi = 0x5000
# The actual flag.
# esil.mem.mem_write(esil.reg.rdi, b"qiling{framework}")
# The symbolic bit vector.
flag = z3.BitVec("flag", 17*8)
# Write to memory
esil.mem.write(esil.reg.rdi, flag)
# Hook each state.
esil.hook_step(hook_state)
# Start emulation.
last_state = esil.emu_start(target_function["offset"], target
```

Future

Future

- Lots of extra code and testcases for corner cases need to be done, especially the memory and registers implementation.
- Rearrange the code to integrate the symex engine to Qiling codebase better.
  - The code will be released after some iteration and refactor.
- Speed up the symex by optimizing memory copy.
- Make contributions to ctypeslib, radare2 during our test and usage.
- Integrate Qiling and R2 in the other way, by running Qiling inside R2.

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