Binary Analysis Craft - BinCraft

Author

Anciety <anciety@starcross.tech>

- DEFCON FINAL 2018-2019 r3kapig team member
- DEFCON FINAL 2020 r3kapig team leader
- WCTF 2019 3rd place
- Real World CTF 2018 final 5th place
- Binary Researcher@StarCrossTech (starcross.tech) PortalLab
- Reverse Tool Enthusiast

Author

Ios <ios@starcross.tech>

- DEFCON 2018-2020 r3kapig team member
- Ios system reseacher
- Binary Resercher @ StarCross PortalLab

Author

Cdxy

- Researcher@StarCrossTech (starcross.tech) PortalLab
- DEFCON 2018-2020 r3kapig team member
- Published topics in BlackHat, BlueHat, OWASP, Xcon, PHDays…
- XCTF Co-organizer
- Topics: Container Security, incident response, RE…

Content

What is Missing for Reversing Tools?

BinCraft, the binary craft

- SleighCraft
- QueryCraft
- Use cases

Current Reversing Tools

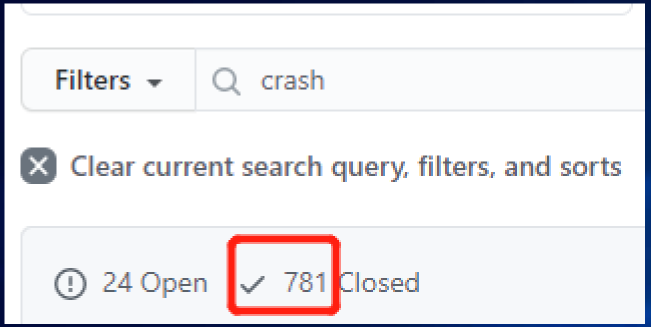
- I know, we have many reversing tools already.
- Ghidra
- IDA pro
- Radare2/Rizin
- Binary Ninja
- Capstone(&unicorn)
- Angr
- BAP
- Why do we need a new one?

Ghidra

- **Usable Project, the most competitive one (vs. IDA)**
- **Decompiler available**
- **But...**
  - JVM only
  - No library-like functionality

Radare2/Rizin

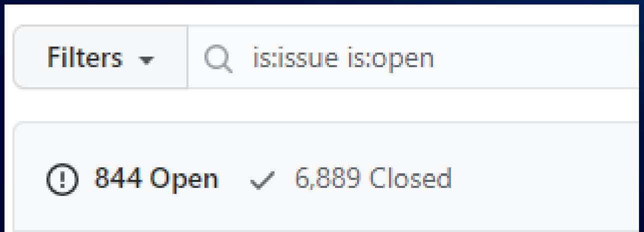
- **UNIX-like reversing**
- **A cool project, but…**
- **C is hard.**
- **Many needs to be fixed.**
  - I myself frequently encounter bugs that stop me from accomplish my job.



Filters ▾ | crash

☒ Clear current search query, filters, and sorts

24 Open ✓ 781 Closed

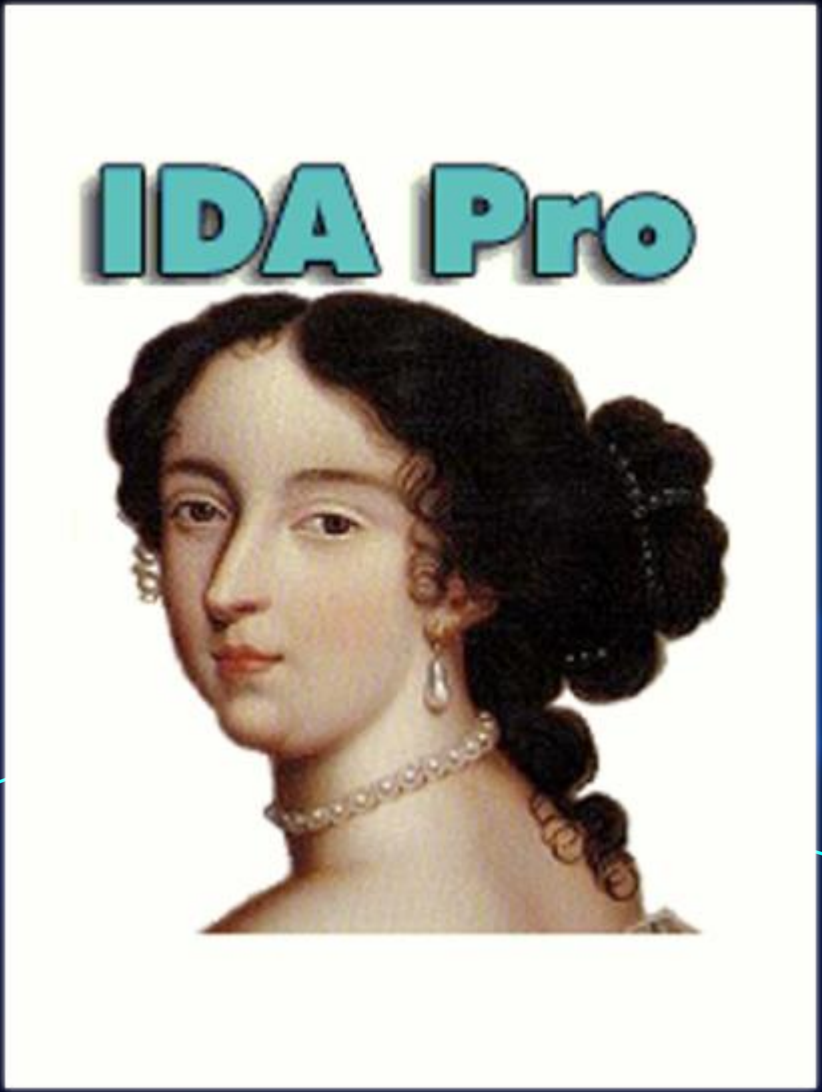


Filters ▾ | is:issue is:open

844 Open ✓ 6,889 Closed

IDA pro

- The most used.
- But…
- Proprietary Software
- Expensive.
- Slow API
- Hard to use headless



Binary Ninja

- Good to use when scripting
- Proprietary Product
- No library-like (second time developable)

Capstone

- Not a complete reversing tool, but a library
- No IR support
- No algorithms

Angr

- Known as a binary symbolic execution tool
- Also can be used for binary analysis
- IR available
- But…
- Python is needed. (So use Angr in C/C++/Rust or other language is a bad idea)

Bap

- Binary Analysis Platform
- Introduces a lot of interesting ideas in binary analysis
- But…
- Ocaml has its own idea..

What is missing for reversing?

- Actually, no.
- They are all good and working.

Then why do we need a new one?

A new one? But...

- Previous tools are working good.
- Although they have problems, you don’ t need to stick to only one.
- A new one costs a lot!!

So why?

- We just want new ideas in binary analysis!
- To explore the power of reusing ghidra basics
- Maybe, build a full-featured SQL-based binary analysis framework in the end
    - Like r2, but they stick to file, we stick to SQL

BinCraft – the binary analysis craft

Goal

- Stand on the shoulder of ghidra, but not totally depend on it (so, not an extension)
- Design a new reversing toolkit paradigm: SQL-based binary analysis
- Component-style, components should work by themselves.
  - They can also be used as a basic for other projects.
  - Or write automatic analyses

SleighCraft

SleighCraft – the basic craft of binary analysis

- Deals with basic disassembly, i.e, binary => disassembly
- Also, **binary => IR**(missing in capstone)
- No vm needed (no Python vm, JVM or any other vm)
- Can be used as a library (like capstone)
- Based on ghidra’ s sleigh engine

Pcode IR

- SleighCraft binds to ghidra’ s decompilation engine
- IR also uses its IR, P-code IRs

Pcode IR

Pcode model

- RAM => address space
- Register => varnode
- Instruction => operation

Address space

- A linear memory space
- Identified by addr space name + offset (addr)

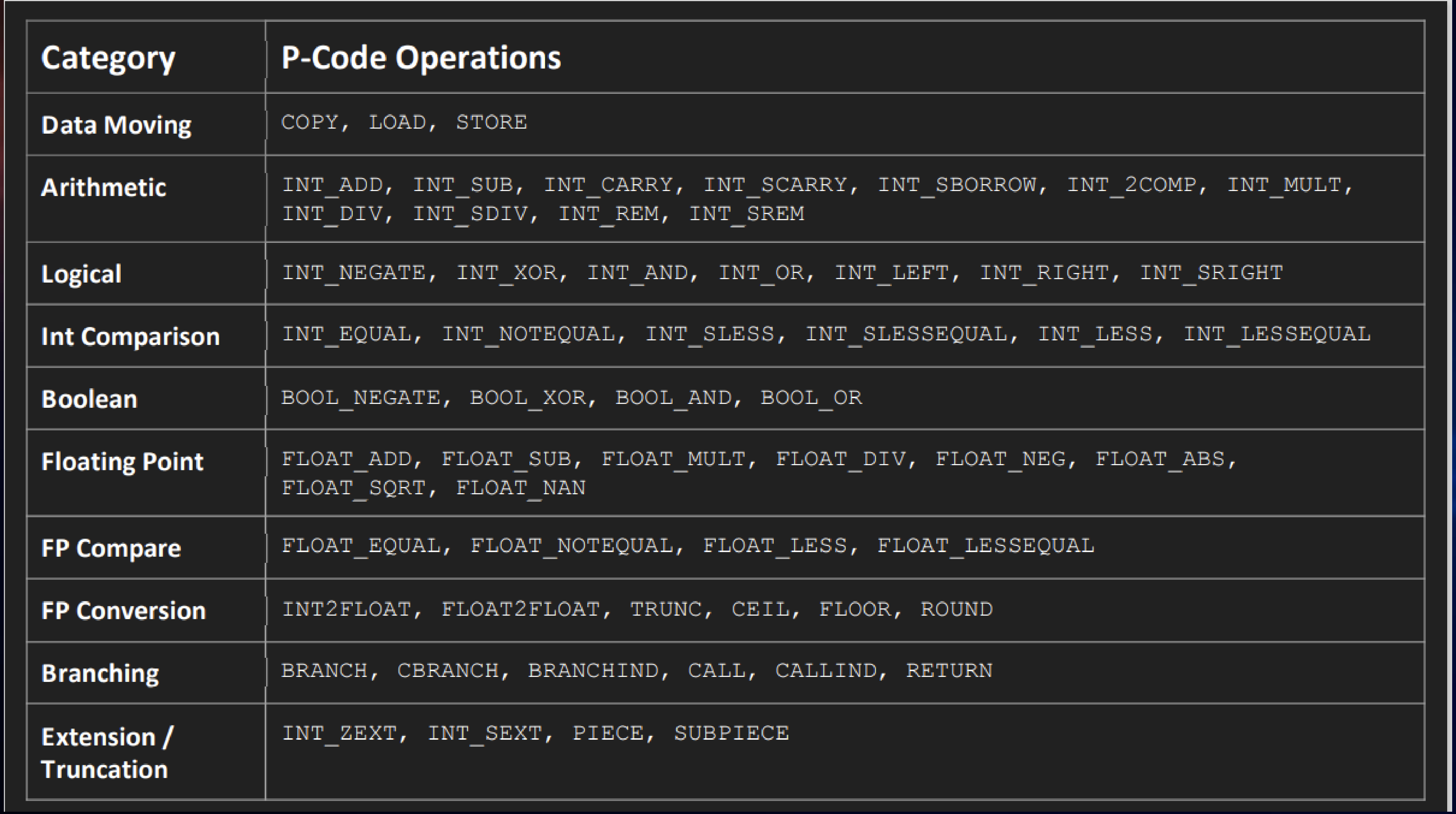
Varnode

- Several continuous bytes in the address space

Operation

- Add, Sub, Mov...

Pcode IR operations



Category	P-Code Operations
Data Moving	COPY, LOAD, STORE
Arithmetic	INT_ADD, INT_SUB, INT_CARRY, INT_SCARRY, INT_SBORROW, INT_2COMP, INT_MULT, INT_DIV, INT_SDIV, INT_REM, INT_SREM
Logical	INT_NEGATE, INT_XOR, INT_AND, INT_OR, INT_LEFT, INT_RIGHT, INT_SRIGHT
Int Comparison	INT_EQUAL, INT_NOTEQUAL, INT_SLESS, INT_SLESSEQUAL, INT_LESS, INT_LESSEQUAL
Boolean	BOOL_NEGATE, BOOL_XOR, BOOL_AND, BOOL_OR
Floating Point	FLOAT_ADD, FLOAT_SUB, FLOAT_MULT, FLOAT_DIV, FLOAT_NEG, FLOAT_ABS, FLOAT_SQRT, FLOAT_NAN
FP Compare	FLOAT_EQUAL, FLOAT_NOTEQUAL, FLOAT_LESS, FLOAT_LESSEQUAL
FP Conversion	INT2FLOAT, FLOAT2FLOAT, TRUNC, CEIL, FLOOR, ROUND
Branching	BRANCH, CBRANCH, BRANCHIND, CALL, CALLIND, RETURN
Extension / Truncation	INT_ZEXT, INT_SEXT, PIECE, SUBPIECE

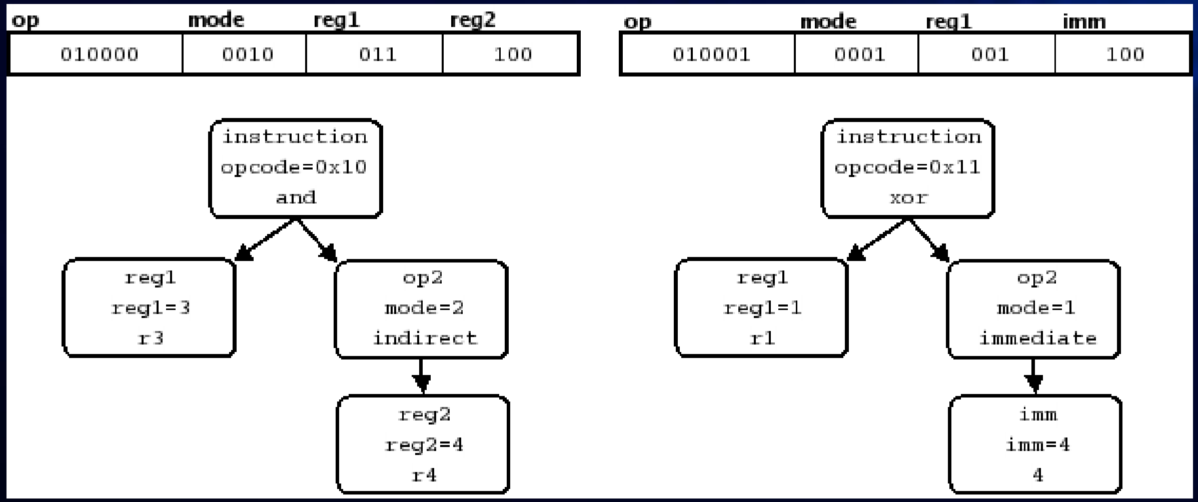
Sleigh

- Ghidra’ s instruction decoding engine
- DSL to decode instruction to...
  - Disassembly
  - Pcode IR
- The only industrial DSL decoding binary for now.

Meet Sleigh

Sleigh describes how to decode binary

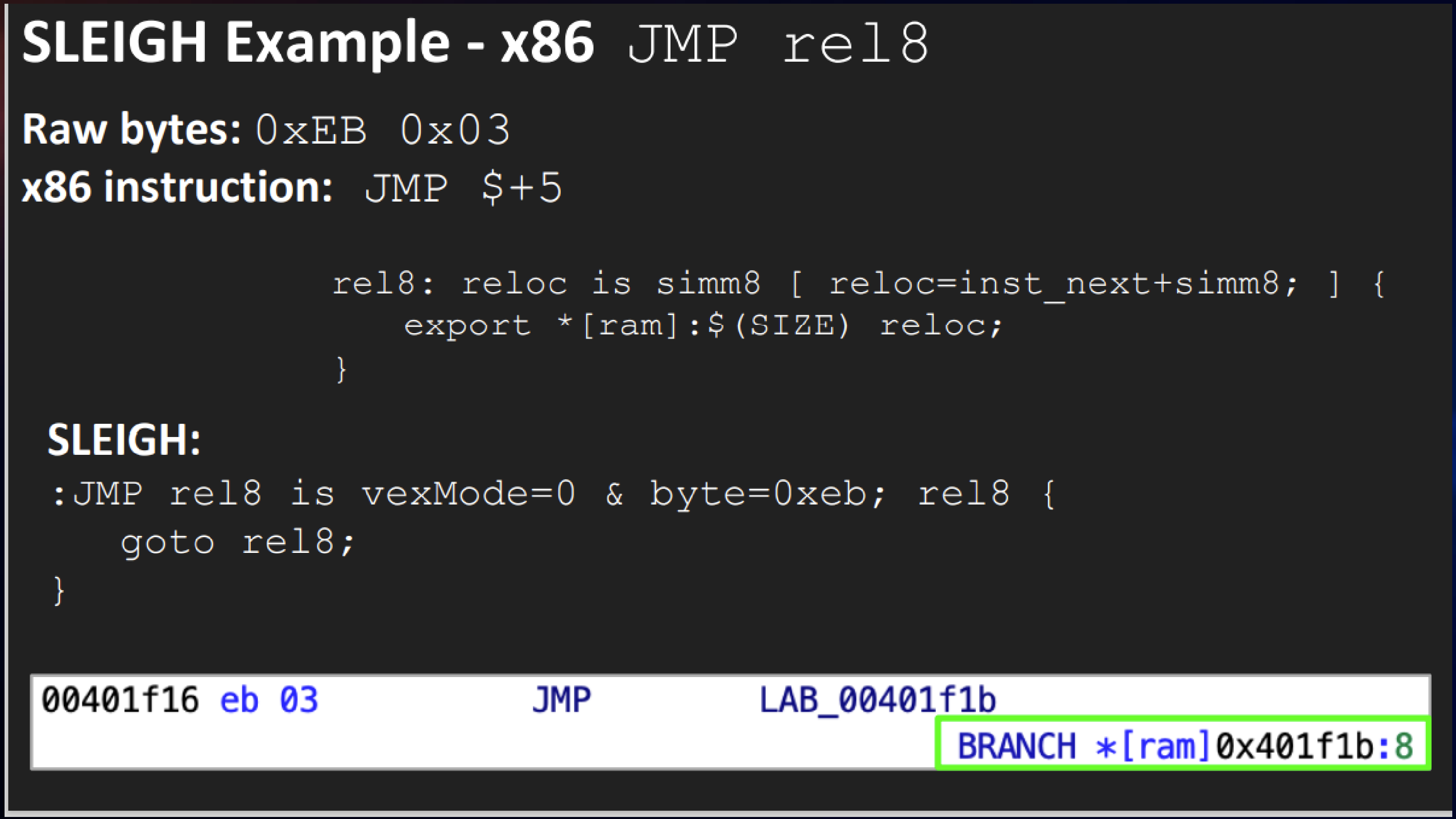
- It parses the binary into a tree
- The tree then describes how to display (disassemble)
- Also describes how to encode to Pcode



op	mode	reg1	reg2
010000	0010	011	100

op	mode	reg1	imm
010001	0001	001	100

Sleigh Example (from infiliate-ghidra)



SLEIGH Example - x86 `JMP rel8`

Raw bytes: `0xEB 0x03`

x86 instruction: `JMP $+5`

```
rel8: reloc is simm8 [ reloc=inst_next+simm8; ] {
    export *[ram]:$(SIZE) reloc;
}
```

SLEIGH:

```
:JMP rel8 is vexMode=0 & byte=0xeb; rel8 {
   goto rel8;
}
```

```
00401f16 eb 03          JMP          LAB_00401f1b
                                              BRANCH *[ram]0x401f1b:8
```

Why Sleigh

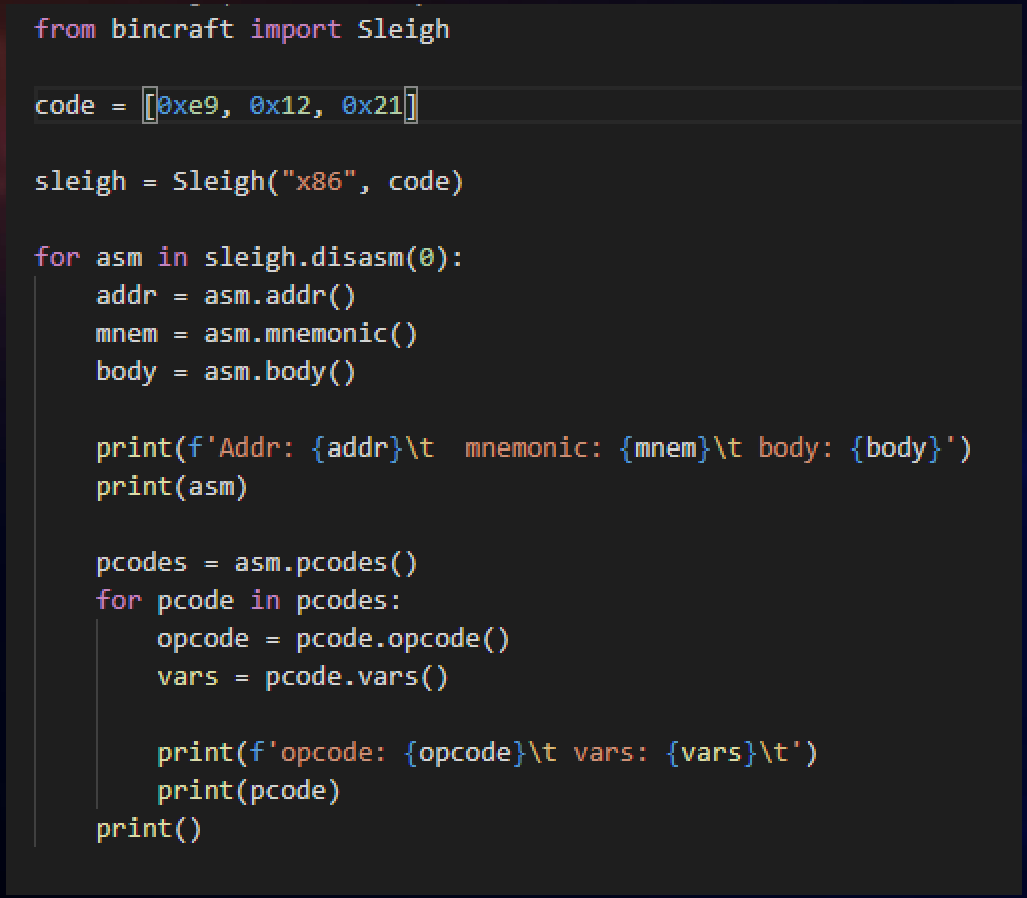
SleighCraft chose sleigh for...

- Extensibility: give the DSL, the library can now disassemble new archs
- IR: IR Is directly available
- Availability: ghidro provides sleigh engine out of the box

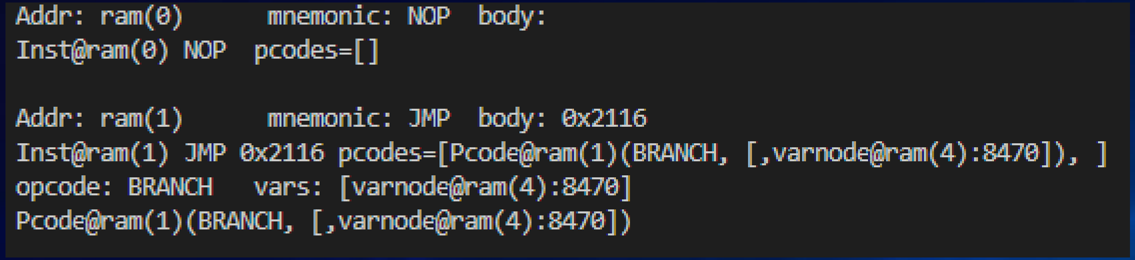
Then what is sleighcraft?

- Sleigh engine (cpp) to Rust binding
- Rust to other language bindings
  - Python
  - Js
  - …
- Reason: Write binding in Rust is way easier than in C/C++

SleighCraft Example (python)



```python
from bincraft import Sleigh

code = [0xe9, 0x12, 0x21]

sleigh = Sleigh("x86", code)

for asm in sleigh.disasm(0):
    addr = asm.addr()
    mnem = asm.mnemonic()
    body = asm.body()

    print(f'Addr: {addr}\t  mnemonic: {mnem}\t body: {body}')
    print(asm)

    pcodes = asm.pcodes()
    for pcode in pcodes:
        opcode = pcode.opcode()
        vars = pcode.vars()

        print(f'opcode: {opcode}\t vars: {vars}\t')
        print(pcode)
    print()
```



```
Addr: ram(0)      mnemonic: NOP  body:
Inst@ram(0) NOP  pcodes=[]

Addr: ram(1)      mnemonic: JMP  body: 0x2116
Inst@ram(1) JMP 0x2116 pcodes=[Pcode@ram(1)(BRANCH, [,varnode@ram(4):8470]), ]
opcode: BRANCH   vars: [varnode@ram(4):8470]
Pcode@ram(1)(BRANCH, [,varnode@ram(4):8470])
```

QueryCraft

SQL based binary analysis

What is SQL based?

- Radare2: commandline based
  - Use commandline to specify your target
  - Use commandline to modify current status (variable names, etc)
- Sql based
  - Use SQL to specify your target
  - Use SQL to modify current status (variable names, etc.)
- Implemented as a SQLite extension
  - So we get all language bindings free.
  - Just load the extension!

QueryCraft

- Load the extension
- > .load ./libQueryCraft.so
- Read the binary, returns an id
- > SELECT qc_read_bin("/bin/ls"); # returns id 1 (a number)
- Use the id, do the analyses!

QueryCraft Schema

- **QueryCraft functions accept table as input, read or transform the table content**
- **Table can be custom (inserted manually by user) or created by QueryCraft**
- **Input table is constraint by fields**
- **Example: schema “pcode”**
  - Requires fields:
    - “space”, “offset”: as Varnode
    - “op”: operation
    - “op1_space”, “op1_offset”, “op1_size”: and “op2”, “op3”, “out”, different varnodes argument

QueryCraft Analyses

Analyses can be:

- Transform: insert or deletes rows in a table
- Annotate: creates a new table (or reuse previous table). The table contains a new information relates input table.
- Onetime: side-effect free analyses.

QueryCraft Analyses

- **Example: transform analysis**
- **Dead Code Eliminate**
- **> SELECT qc_dead_code_elim(“input_table”);**
- **Input:**
  - Arg 1: “input_table” input table name, the table should follow the pcode schema
- **Transform**
  - The rows in the table can be deleted if is dead code

QueryCraft Analyses

- **Example: annotate analysis**
- **Control Flow Graph generate**
- **> SELECT qc_cfg(“input_table” , “out_table”)**
- **Input:**
  - Arg 1: input table name, also follows “pcode”
  - Arg 2: output table name
- **Annotate:**
  - Out table will contain the information about the variables identified

QueryCraft Analyses

- **Example: Onetime analysis**
- **Onetime Disassemble**
- **> SELECT * from qc_disasm(x’ 909090’ , 0);**
- **Input:**
  - Arg 1: the bytes to disasm
  - Arg 2: the address of disassmble
- **One time output:**
  - The onetime analysis works like a table
  - “qc_disasm” returns the disassembly of 0x909090 in a table

Why QueryCraft

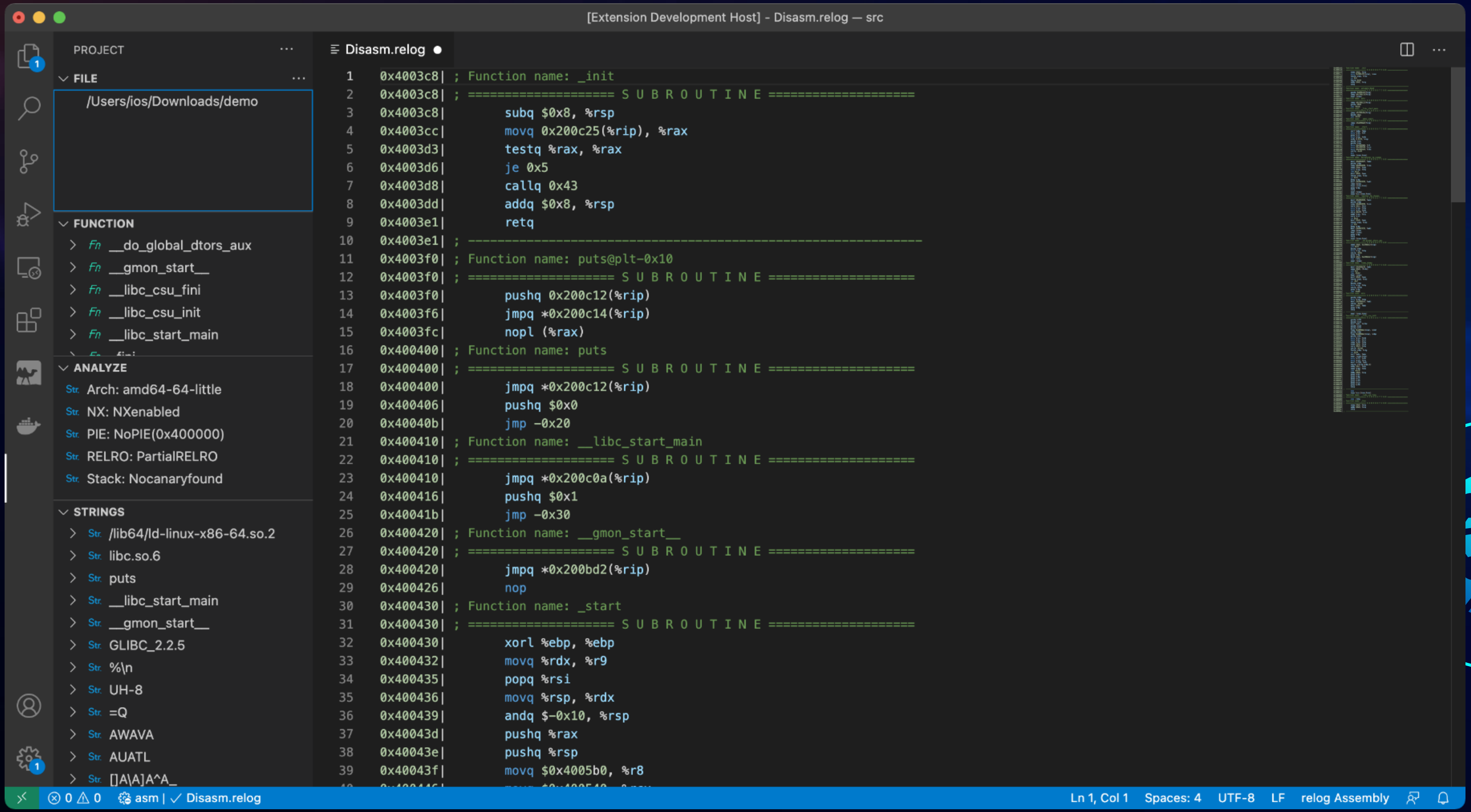
- **It is fun!**
- **SQL is good to extract certain information.**
- **Consider:**
  - Find all “mov-mov-jmp” patterns (may happen when deobfuscation)
  - In ghidra: write a script (in python, using api)
  - In IDA: write a script (in python, using api)
  - In SQL: a single select

Future of BinCraft

- **Initially, BinCraft may be just a toy showing ideas.**
- **Finally, the ideas may combine.**
- **And we may get to a full-featured reversing tool**
  - Also with new ideas!

BinCraft Usecases

Interactive malware/binary analysis



[Extension Development Host] - Disasm.relog — src

PROJECT

FILE

/Users/ios/Downloads/demo

FUNCTION

- Fn __do_global_dtors_aux
- Fn __gmon_start__
- Fn __libc_csu_fini
- Fn __libc_csu_init
- Fn __libc_start_main

ANALYZE

- Str Arch: amd64-64-little
- Str NX: NXenabled
- Str PIE: NoPIE(0x400000)
- Str RELRO: PartialRELRO
- Str Stack: Nocanaryfound

STRINGS

- Str /lib64/ld-linux-x86-64.so.2
- Str libc.so.6
- Str puts
- Str __libc_start_main
- Str __gmon_start__
- Str GLIBC_2.2.5
- Str %\n
- Str UH-8
- Str =Q
- Str AWAVA
- Str AUATL
- Str []A\A]A^A_

Disasm.relog ●

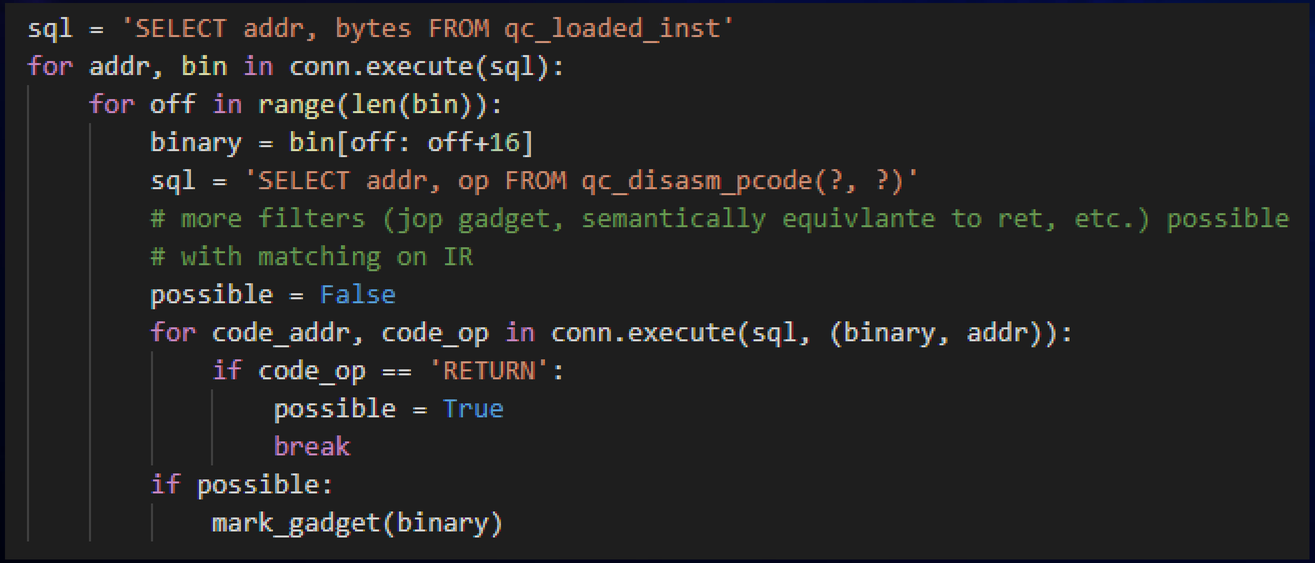
```
0x4003c8| ; Function name: _init
0x4003c8| ; ==================== S U B R O U T I N E ====================
0x4003c8|       subq $0x8, %rsp
0x4003cc|       movq 0x200c25(%rip), %rax
0x4003d3|       testq %rax, %rax
0x4003d6|       je 0x5
0x4003d8|       callq 0x43
0x4003dd|       addq $0x8, %rsp
0x4003e1|       retq
0x4003e1| ; --------------------------------------------------------------
0x4003f0| ; Function name: puts@plt-0x10
0x4003f0| ; ==================== S U B R O U T I N E ====================
0x4003f0|       pushq 0x200c12(%rip)
0x4003f6|       jmpq *0x200c14(%rip)
0x4003fc|       nopl (%rax)
0x400400| ; Function name: puts
0x400400| ; ==================== S U B R O U T I N E ====================
0x400400|       jmpq *0x200c12(%rip)
0x400406|       pushq $0x0
0x40040b|       jmp -0x20
0x400410| ; Function name: __libc_start_main
0x400410| ; ==================== S U B R O U T I N E ====================
0x400410|       jmpq *0x200c0a(%rip)
0x400416|       pushq $0x1
0x40041b|       jmp -0x30
0x400420| ; Function name: __gmon_start__
0x400420| ; ==================== S U B R O U T I N E ====================
0x400420|       jmpq *0x200bd2(%rip)
0x400426|       nop
0x400430| ; Function name: _start
0x400430| ; ==================== S U B R O U T I N E ====================
0x400430|       xorl %ebp, %ebp
0x400432|       movq %rdx, %r9
0x400435|       popq %rsi
0x400436|       movq %rsp, %rdx
0x400439|       andq $-0x10, %rsp
0x40043d|       pushq %rax
0x40043e|       pushq %rsp
0x40043f|       movq $0x4005b0, %r8
```

⊗ 0 ⚠ 0 asm | ✓ Disasm.relog Ln 1, Col 1 Spaces: 4 UTF-8 LF relog Assembly

BinCraft Usecases

Cross-arch rop gadget search

- Example of automatic static analysis
- Works in any language with sqlite3 binding



```python
sql = 'SELECT addr, bytes FROM qc_loaded_inst'
for addr, bin in conn.execute(sql):
    for off in range(len(bin)):
        binary = bin[off: off+16]
        sql = 'SELECT addr, op FROM qc_disasm_pcode(?, ?)'
        # more filters (jop gadget, semantically equivlante to ret, etc.) possible
        # with matching on IR
        possible = False
        for code_addr, code_op in conn.execute(sql, (binary, addr)):
            if code_op == 'RETURN':
                possible = True
                break
        if possible:
            mark_gadget(binary)
```

BinCraft Usecases

CTF solving – DEFCON 2020 cross arch shellcode

- Bruteforce bytes
- Disassemble
- Inspect IR with SleighCraft python API

DEMO

@Starcross portallab

Github.com/starcrossportal/bincraft

@Starcross portallab