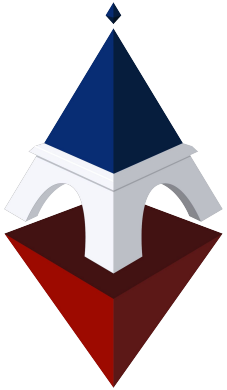


Reversing Ewasm contract 101

EthCC 2020 - Workshop

Whoami

Patrick Ventuzelo / @Pat_Ventuzelo

- Twitter / LinkedIn / Github / Blog

Independent Security Researcher
⇒ Trainings/Consulting

Previously:

- QuoScient GmbH
- P1 Security
- French DoD
- Airbus Defense & Space

⇒ Vulnerability research, Fuzzing
⇒ WebAssembly, Smart contracts
⇒ Blockchain security
⇒ Malware Analysis

Security trainings

WebAssembly Security
“From Reversing to Vulnerability Research”
(4-5 days)

- Introduction
- WebAssembly reversing
- Static analysis
- Dynamic analysis (DBI)
- Debugging
- (De-)Obfuscation
- Wasm games hacking
- Finding wasm Module vulnerabilities
- Emscripten & NodeJS exploit
- Wasm Module fuzzing
- Fuzzing Web-Browsers (V8, Webkit, ...)
- Fuzzing WebAssembly VMs (C/C++,Rust, ...)

https://webassembly-security.com/trainings

Rustlang security
“For Hackers and Developers”
(2-3 days)

- Introduction
- Rustlang vulnerabilities
- Panicking macros
- Unsafe codes
- Auditing tools
- Debugging (lldg, gdb, ...)

- Fuzzing (afl, hongfuzz, libfuzzer, ...)
- Triaging / Bugs analysis
- Code coverage
- Sanitizers (ASAN, MSAN, ...)
- Symbolic execution

https://webassembly-security.com/rust-security-training/

Today’s summary

- Introduction to WebAssembly
- What is Ewasm?
  - Ewasm execution engine / public testnet
  - Solidity/YUL to Ewasm
  - Ethereum Contract Interface (ECI)
- Reversing Ewasm bytecode
  - Why?
  - Disassembling
  - CFG & Call graph
- Examples
  - Deployment bytecode
  - Storage contract
  - ERC-20
- Conclusion

Introduction to WebAssembly

What is WebAssembly?

“Binary instruction format for a stack-based virtual machine”

- Low-level bytecode
- Compilation target for C/C++/Rust/Go/…

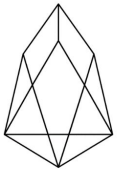
- Generic evolution of NaCl & Asm.js
- W3C standard
- MVP 1.0 (March 2017)
- Natively supported in all major browsers

- WebAssembly goals:
  - Be fast, efficient, and portable (**near-native speed**)
  - Easily **readable and debuggable** (wat/wast)
  - **Keep secure** (safe, sandboxed execution environment)
  - Don't break the web

WebAssembly for Blockchain

- **Ethereum (Ewasm)**
  - The Next Generation Ethereum Virtual Machine / Ewasm VM - link
- Ethereum (Parity)
  - pwasm: Parity Wasm contract - link
  - Sub0.1: Wasm and Substrate - video
- EOS
  - EOS-VM: A High-Performance Blockchain WebAssembly Interpreter - link
- DFINITY
  - Dfinity is a blockchain-based cloud computing project - source
- Spacemesh
  - Spacemesh Virtual Machine based on wasmer - blogpost, github
- Golem
  - gWASM task in Golem - link
  - sp-wasm - SpiderMonkey-based WebAssembly Sandbox

- Further reading:
  - **Wasm on the blockchain workshop Berlin 2019 - blogpost, videos**



spacemesh

golem

Compilation to WebAssembly

C/C++

```c
int fib(int n)
{
  if (n == 0 || n == 1)
    return n;
  else
    return (fib(n-1) + fib(n-2));
}
```

Rust

```rust
fn fib(n: u32) -> u32 {
    match n {
        0 => 1,
        1 => 1,
        _ => fib(n - 1) + fib(n - 2),
    }
}
```

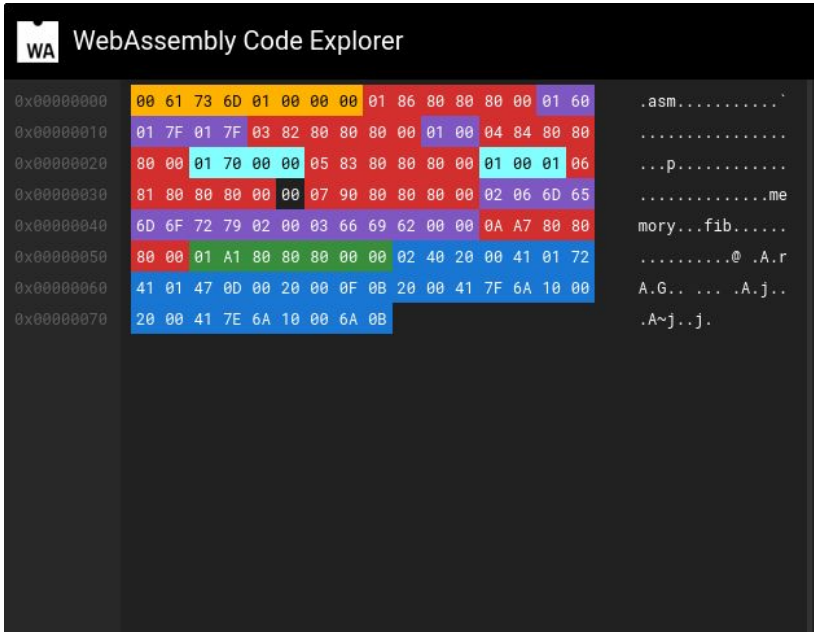
binary file (.wasm)

```
0061 736d 0100 0000
0186 8080 8000 0160
017f 017f 0382 8080
8000 0100 0484 8080
8000 0170 0000 0583
8080 8000 0100 0106
8180 8080 0000 0790
8080 8000 0206 6d65
6d6f 7279 0200 0366
6962 0000 0aa7 8080
8000 01a1 8080 8000
0002 4020 0041 0172
4101 470d 0020 000f
0b20 0041 7f6a 1000
2000 417e 6a10 006a
0b
```

WebAssembly Binary Format (wasm) - overview

- Binary format
- Magic number: **\x00asm**
- Module structure
  - Header
  - **11 Sections** + custom sections

Section Name	Code	Description
Type	1	Function signature declarations
Import	2	Import declarations
Function	3	Function declarations
Table	4	Indirect function table and other tables
Memory	5	Memory attributes
Global	6	Global declarations
Export	7	Exports
Start	8	Start function declaration
Element	9	Elements section
Code	10	Function bodies (code)
Data	11	Data segments



https://wasdk.github.io/wasmcodeexplorer/

Compilation to WebAssembly

C/C++

```c
int fib(int n)
{
  if (n == 0 || n == 1)
    return n;
  else
    return (fib(n-1) + fib(n-2));
}
```

Rust

```rust
fn fib(n: u32) -> u32 {
    match n {
        0 => 1,
        1 => 1,
        _ => fib(n - 1) + fib(n - 2),
    }
}
```

binary file (.wasm)

```
0061 736d 0100 0000
0186 8080 8000 0160
017f 017f 0382 8080
8000 0100 0484 8080
8000 0170 0000 0583
8080 8000 0100 0106
8180 8080 0000 0790
8080 8000 0206 6d65
6d6f 7279 0200 0366
6962 0000 0aa7 8080
8000 01a1 8080 8000
0002 4020 0041 0172
4101 470d 0020 000f
0b20 0041 7f6a 1000
2000 417e 6a10 006a
0b
```

wasm text format

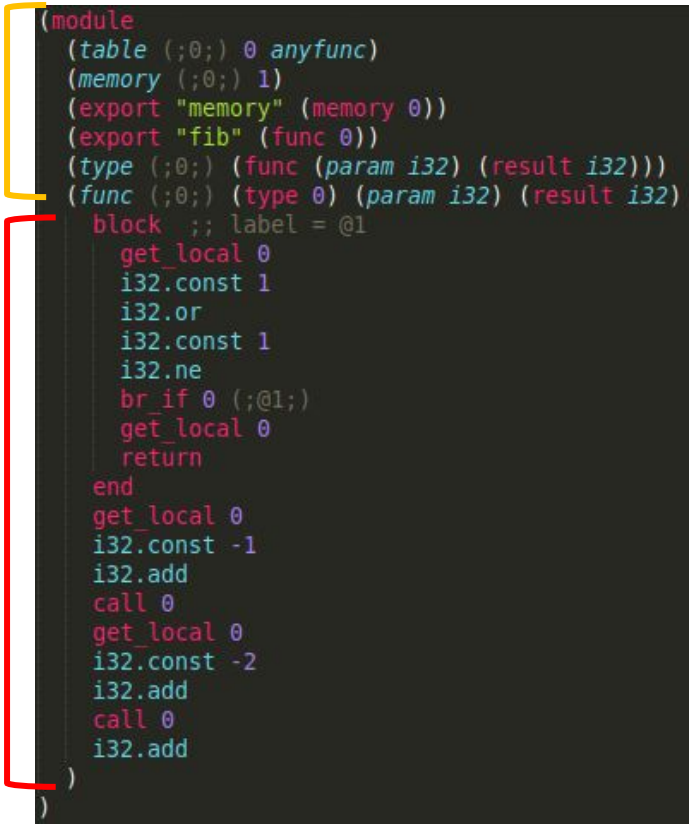
```
(module
 (table 0 anyfunc)
 (memory $0 1)
 (export "memory" (memory $0))
 (export "fib" (func $fib))
 (func $fib (; 0 ;) (param $0 i32) (result i32)
  (block $label$0
   (br_if $label$0
    (i32.ne
     (i32.or
      (get_local $0)
      (i32.const 1)
     )
     (i32.const 1)
    )
   )
   (return
    (get_local $0)
   )
  )
  (i32.add
   (call $fib
    (i32.add
     (get_local $0)
     (i32.const -1)
    )
   )
   (call $fib
    (i32.add
     (get_local $0)
     (i32.const -2)
    )
   )
  )
 )
)
```

WebAssembly Text Format

- Standardized text format
  - File extensions:
    - .wat or .wast
  - **S-expressions** (like LISP)
    - modules and section definitions
  - Functions body
    - **Linear representation**
    - Low-level instructions or S-expressions

- wasm2wat
  - Translate from the binary format back to the text format
  - **wasm ⇒ wat**

- wat2wasm
  - Translate from text format to the WebAssembly binary format
  - **wat/wast ⇒ wasm**
  - Online demo



WebAssembly Text Format

- Small **Turing-complete instruction set**
  - 172 instructions
  - Data types: **`i32, i64, f32, f64, (v)`**

- **Control-Flow** operators
  - Label: `block loop if else end`
  - Branch: `br br_if br_table`
  - Function call: `call call_indirect return`
- **Memory** operators
  - `load, store, etc.`
- **Variables** operators
  - `local, global,`
- **Arithmetic** operators (for `int` & `float`)
  - `+ - * / % && || ^ << >> etc.`
  - `sqrt ceil floor etc.`
- **Constant** operators (`const`)
- **Conversion** operators
  - `wrap trunc convert reinterpret etc.`

```
(module
  (table (;0;) 0 anyfunc)
  (memory (;0;) 1)
  (export "memory" (memory 0))
  (export "fib" (func 0))
  (type (;0;) (func (param i32) (result i32)))
  (func (;0;) (type 0) (param i32) (result i32)
    block  ;; label = @1
      get_local 0
      i32.const 1
      i32.or
      i32.const 1
      i32.ne
      br_if 0 (;@1;)
      get_local 0
      return
    end
    get_local 0
    i32.const -1
    i32.add
    call 0
    get_local 0
    i32.const -2
    i32.add
    call 0
    i32.add
  )
)
```

WebAssembly Instructions Set (ISA)

i32.add	i64.add	f32.add	f64.add	i32.wrap/i64	i32.load8_s	i32.store8
i32.sub	i64.sub	f32.sub	f64.sub	i32.trunc_s/f32	i32.load8_u	i32.store16
i32.mul	i64.mul	f32.mul	f64.mul	i32.trunc_s/f64	i32.load16_s	i32.store
i32.div_s	i64.div_s	f32.div	f64.div	i32.trunc_u/f32	i32.load16_u	i64.store8
i32.div_u	i64.div_u	f32.abs	f64.abs	i32.trunc_u/f64	i32.load	i64.store16
i32.rem_s	i64.rem_s	f32.neg	f64.neg	i32.reinterpret/f32	i64.load8_s	i64.store32
i32.rem_u	i64.rem_u	f32.copysign	f64.copysign	i64.extend_s/i32	i64.load8_u	i64.store
i32.and	i64.and	f32.ceil	f64.ceil	i64.extend_u/i32	i64.load16_s	f32.store
i32.or	i64.or	f32.floor	f64.floor	i64.trunc_s/f32	i64.load16_u	f64.store
i32.xor	i64.xor	f32.trunc	f64.trunc	i64.trunc_s/f64	i64.load32_s	
i32.shl	i64.shl	f32.nearest	f64.nearest	i64.trunc_u/f32	i64.load32_u	
i32.shr_u	i64.shr_u			i64.trunc_u/f64	i64.load	call
i32.shr_s	i64.shr_s	f32.sqrt	f64.sqrt	i64.reinterpret/f64	f32.load	call_indirect
i32.rotl	i64.rotl	f32.min	f64.min		f64.load	
i32.rotr	i64.rotr	f32.max	f64.max			
i32.clz	i64.clz				nop	grow_memory
i32.ctz	i64.ctz				block	current_memory
i32.popcnt	i64.popcnt			f32.demote/f64	loop	
i32.eqz	i64.eqz			f32.convert_s/i32	if	get_local
i32.eq	i64.eq			f32.convert_s/i64	else	set_local
i32.ne	i64.ne			f32.convert_u/i32	br	tee_local
i32.lt_s	i64.lt_s			f32.convert_u/i64	br_if	
i32.le_s	i64.le_s			f32.reinterpret/i32	br_table	get_global
i32.lt_u	i64.lt_u	f32.eq	f64.eq	f64.promote/f32	return	set_global
i32.le_u	i64.le_u	f32.ne	f64.ne	f64.convert_s/i32	end	
i32.gt_s	i64.gt_s	f32.lt	f64.lt	f64.convert_s/i64		i32.const
i32.ge_s	i64.ge_s	f32.le	f64.le	f64.convert_u/i32	drop	i64.const
i32.gt_u	i64.gt_u	f32.gt	f64.gt	f64.convert_u/i64	select	f32.const
i32.ge_u	i64.ge_u	f32.ge	f64.ge	f64.reinterpret/i64	unreachable	f64.const

What is Ewasm?

What is Ewasm?

- Ewasm ⇒ Ethereum-flavored WebAssembly (current rev4)
  - Primary candidate to replace EVM
    - Part of the Ethereum 2.0 "Serenity" roadmap
  - Deterministic smart contract execution engine
    - based on WebAssembly
  - Ethereum precompiled contracts in Rust - link

- Restricted subset of WebAssembly to be used for contracts in Ethereum.
  - Custom VM semantics
  - Semantics Contract Interface (ECI) Specification
    - Import symbols
    - Exported symbols
  - Defined Ethereum Environment Interface (EEI)
    - Set of methods available to Ewasm contracts
    - Data types & APIs
  - Metering specification
    - Measuring execution gas costs in a deterministic way
  - Reduce non-determinism in WebAssembly

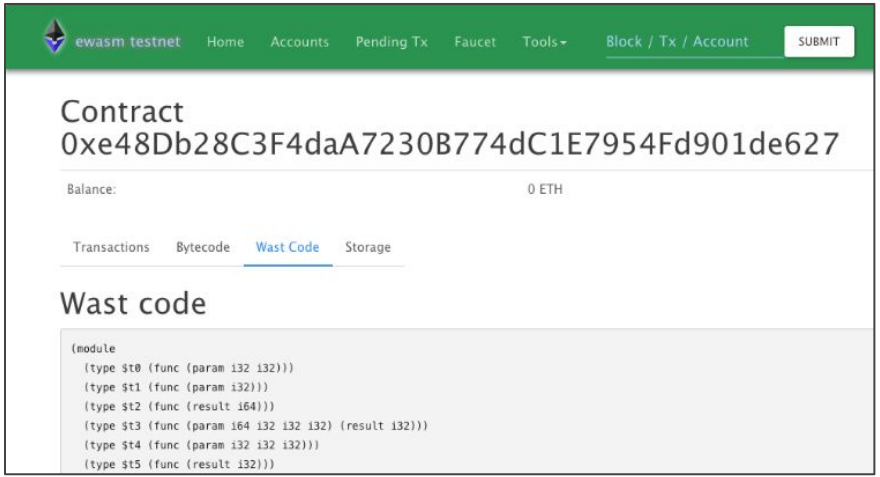
Ewasm execution engine

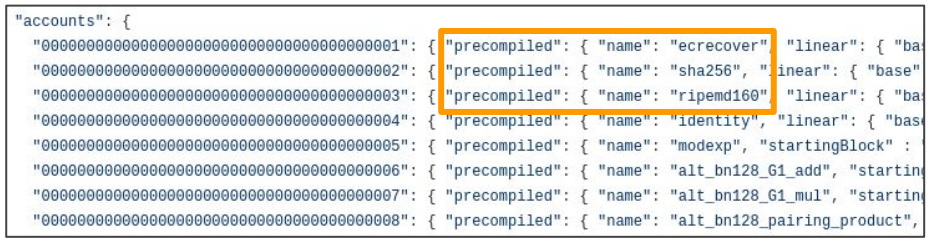
- Hera: Ewasm virtual machine
  - Implemented in C++ conforming to EVMC ABIv6.
  - Designed to leverage **various Wasm backends**, both **interpreters and AOT/JITs.**
  - Complete support of:
    - wabt - WebAssembly Binary Toolkit
    - Binaryen - Compiler infrastructure and toolchain library for WebAssembly
    - WAVM - WebAssembly Virtual Machine (use LLVM)

- Further Ewasm readings:
  - Devcon5
    - Ewasm: Past, Present, Future - video
    - Ewasm 2.0: State Execution in Eth 2.0 - video
    - Yul, Ewasm, Solidity: Progress and Future Plans - video
    - The Road to ETH 2.0: How to Build Ewasm DApps with Embark v5 - video
  - WASM on the blockchain - Berlin 2019 workshop
    - Ewasm overview and the precompile problem videos - part 1 / part 2
  - Devcon4:
    - Ewasm: Ethereum-flavored WebAssembly and Ethereum 2.0 videos - part 1 / part 2
  - WebAssembly / eWasm – What, and Why? - link

Ewasm public testnet

- **!!! CURRENTLY DOWN !!!**
  - guide / explorer / studio / faucet
- Supports executing of both
  - EVM 1.0 (Byzantium) bytecode
  - **Ewasm bytecode.**
    - using Hera Ewasm VM
- Action available:
  - deploying smart contracts
  - Interact with contracts
  - Etc.
- Precompiled contracts:
  - **`ecrecover, sha256, ripemd160`**
  - Metering
  - Complete list - link





Solidity/YUL compiled to Ewasm

- SOLL
  - WebAssembly bytecode **compiler** for the **Solidity and YUL languages**.
  - **Generating Ewasm files** from Solidity and Yul source code.
  - Developed by Second State

- Advantages
  - Easy to setup using docker
  - Support both Solidity and YUL
  - **Compiling and deploying an ERC20 contract onto the Ewasm testnet - blogpost / video**
  - SOLL getting started - link

- Compilation generate 2 contracts
  - `contract.deploy.wasm`
    - Input data of contract creation Tx
    - **loader + runtime** bytecode
  - `contract.wasm`
    - only **runtime** bytecode

```
root@681fca426a52:~/contract# file *
contract.bc:          LLVM IR bitcode
contract.ctor.ll:     C source, ASCII text
contract.deploy.bc:   LLVM IR bitcode
contract.deploy.ll:   ASCII text, with very long lines
contract.deploy.o:    WebAssembly (wasm) binary module version 0x1 (MVP)
contract.deploy.wasm: WebAssembly (wasm) binary module version 0x1 (MVP)
contract.ll:          ASCII text
contract.main.ll:     C source, ASCII text
contract.o:           WebAssembly (wasm) binary module version 0x1 (MVP)
contract.sol:         ASCII text
contract.wasm:        WebAssembly (wasm) binary module version 0x1 (MVP)
```

Ethereum Contract Interface (ECI)

- Ewasm restricted opcodes/instructions
  - Floats opcodes are disallowed.
  - Metering
    - Fixed gas cost/operators - link
    - Specific Memory metering

- Imported symbols (available APIs)
  - Only import symbols specified in the Ethereum Environment Interface (EEI).
    - `getBlockHash, getBlockDifficulty`
    - `call, callDataCopy callDelegate`
    - `etc.`

- Should only export 2 symbols
  - **`memory`**:
    - the shared memory space available for the EEI.
  - **`main`**
    - **contract entry point**
    - a function with no parameters and no result value.

32-bit Integer operators

Opcode	Cycle	IA-32 eqv.	Gas
i32.add	1	ADD	0.0045
i32.sub	1	SUB	0.0045
i32.mul	3	MUL	0.0135
i32.div_s	80	DIV	0.36
i32.div_u	80	DIV	0.36

getAddress

Gets address of currently executing account and stores it in memory at the given offset.

Parameters

- `resultOffset` **i32ptr** the memory offset at which the address is to be stored (`address`)

Returns

nothing

Trap conditions

- store to memory at `resultOffset` results in out of bounds access.

Storage contract - wast representation

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

```
(module
  (type (;0;) (func (param i32 i32)))
  (type (;1;) (func (result i32)))
  (type (;2;) (func (param i32 i32 i32)))
  (type (;3;) (func (param i32)))
  (type (;4;) (func))
  (type (;5;) (func (param i32 i64 i64 i64 i64)))
  (type (;6;) (func (param i64 i64 i64 i64)))
  (import "ethereum" "finish" (func (;0;) (type 0)))
  (import "ethereum" "getCallDataSize" (func (;1;) (type 1)))
  (import "ethereum" "callDataCopy" (func (;2;) (type 2)))
  (import "ethereum" "revert" (func (;3;) (type 0)))
  (import "ethereum" "getCallValue" (func (;4;) (type 3)))
  (import "ethereum" "storageStore" (func (;5;) (type 0)))
  (import "ethereum" "storageLoad" (func (;6;) (type 0)))
  (memory (;0;) 2)
  (global (;0;) (mut i32) (i32.const 66592))
  (global (;1;) i32 (i32.const 66592))
  (global (;2;) i32 (i32.const 1047))
  (export "memory" (memory 0))
  (export "main" (func 11))
  (data (;0;) (i32.const 1024) "Function is not payable")
  (func (;7;) (type 4))
  (func (;8;) (type 5) (param i32 i64 i64 i64 i64) ...
  (func (;9;) (type 6) (param i64 i64 i64 i64) ...
  (func (;10;) (type 3) (param i32) ...
  (func (;11;) (type 4) ...
```

Storage contract - wast representation

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

Imported symbols

Exported symbols

```
(module
  (type (;0;) (func (param i32 i32)))
  (type (;1;) (func (result i32)))
  (type (;2;) (func (param i32 i32 i32)))
  (type (;3;) (func (param i32)))
  (type (;4;) (func))
  (type (;5;) (func (param i32 i64 i64 i64 i64)))
  (type (;6;) (func (param i64 i64 i64 i64)))
  (import "ethereum" "finish" (func (;0;) (type 0)))
  (import "ethereum" "getCallDataSize" (func (;1;) (type 1)))
  (import "ethereum" "callDataCopy" (func (;2;) (type 2)))
  (import "ethereum" "revert" (func (;3;) (type 0)))
  (import "ethereum" "getCallValue" (func (;4;) (type 3)))
  (import "ethereum" "storageStore" (func (;5;) (type 0)))
  (import "ethereum" "storageLoad" (func (;6;) (type 0)))
  (memory (;0;) 2)
  (global (;0;) (mut i32) (i32.const 66592))
  (global (;1;) i32 (i32.const 66592))
  (global (;2;) i32 (i32.const 1047))
  (export "memory" (memory 0))
  (export "main" (func 11))
  (data (;0;) (i32.const 1024) "Function is not payable")
  (func (;7;) (type 4))
  (func (;8;) (type 5) (param i32 i64 i64 i64 i64) ...
  (func (;9;) (type 6) (param i64 i64 i64 i64) ...
  (func (;10;) (type 3) (param i32) ...
  (func (;11;) (type 4) ...
```

Reversing Ewasm bytecode

Reversing smart contract, but WHY?

- Analysis 3rd party closed-source contract
  - Due diligence
  - Bug bounty / **Vulnerability research**
  - Understand internal logics
    - Interaction with other contracts/accounts
    - Honeypot / Scam / Fake contract ?
  - CTF challenge

- Post compilation analysis
  - **Optimization**
    - Simplify arithmetic operations
    - **Reduce contract size**
    - Reduce gas cost
  - Proving contract correctness
  - Symbolic execution

- Etc.

```
0061 736d 0100 0000
0186 8080 8000 0160
017f 017f 0382 8080
8000 0100 0484 8080
8000 0170 0000 0583
8080 8000 0100 0106
8180 8080 0000 0790
8080 8000 0206 6d65
6d6f 7279 0200 0366
6962 0000 0aa7 8080
8000 01a1 8080 8000
0002 4020 0041 0172
4101 470d 0020 000f
0b20 0041 7f6a 1000
2000 417e 6a10 006a
0b
```

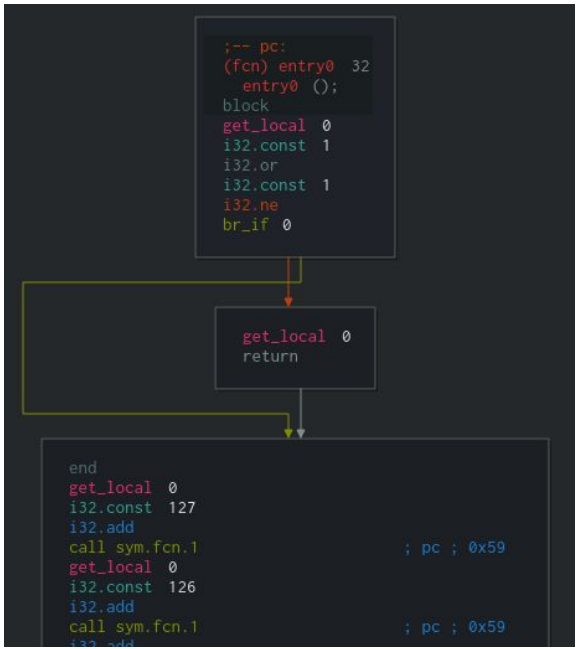
WebAssembly disassembling

- WebAssembly Text format
  - Good start but not enough...
  - Not easy to understand control flow
    - branches, blocks, etc.

- Disassembler
  - Translates machine language into assembly
  - Control Flow Graph (CFG)

- Radare2
  - Reverse engineering framework
  - Support WebAssembly
  - Cutter is the Qt GUI

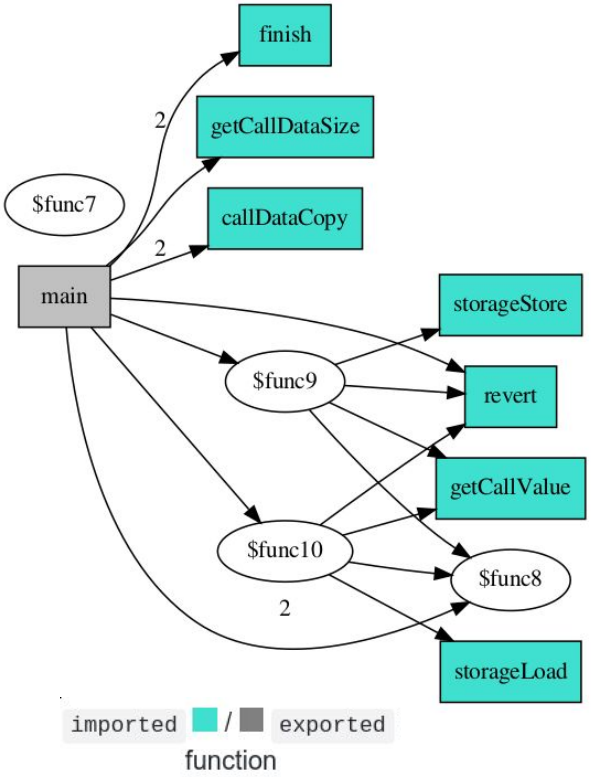
- Octopus
  - Security analysis framework
  - CFG & Call Graph generation



Call Flow Graph

- **Make function's interaction easy to understand**
  - Node is a function
  - Edge a direct call
  - Specify imported/exported functions

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```



- Octopus
  - `./octopus wasm.py -c -f storage.bytecode.wasm`
  - `-c, --call`
    - generate the call flow graph
  - `-f, --file`
    - given binary file (.wasm)

Other advanced techniques

- Symbolic execution
  - Manticore [0.3.0 release]
    - By *Trail Of Bits*
    - Recent support of wasm
  - [Blogpost] / [Documentation]

```python
from manticore.wasm import ManticoreWASM
from manticore.core.plugin import Plugin


def getchar(state):
    """ Symbolic `getchar` implementation. Returns an arbitrary single byte """
    res = state.new_symbolic_value(32, "getchar_res")
    state.constrain(0 < res)
    state.constrain(res < 256)
    return [res]


class PrintRetPlugin(Plugin):
    """ A plugin that looks for states that returned zero and solves for their inputs """

    def will_terminate_state_callback(self, state, *args):
        retval = state.stack.peek()
        if retval == 0:
            print("Solution found!")
            for sym in state.input_symbols:
                solved = state.solve_one(sym)
                print(f"{sym.name}: {chr(solved)} --> Return {retval}")


# Pass our symbolic implementation of the `getchar` function into the WASM environment
# as an import.
m = ManticoreWASM("if_check.wasm", env={"getchar": getchar})

# Register our state termination callback
m.register_plugin(PrintRetPlugin())

# Run the main function, which will call getchar
m.main()

# Save a copy of the inputs to the disk
m.finalize()
```

- Decompilation
  - Usually wasm to C code
    - None for wasm to Solidity
  - [wasm2c] / [wasmdec]

```c
209 static u32 fib(u32 p0) {
210   FUNC_PROLOGUE;
211   u32 i0, i1, i2;
212   i0 = p0;
213   i1 = 1u;
214   i0 |= i1;
215   i1 = 1u;
216   i0 = i0 != i1;
217   if (i0) {goto B0;}
218   i0 = p0;
219   goto Bfunc;
220   B0:;
221   i0 = p0;
222   i1 = 4294967295u;
223   i0 += i1;
224   i0 = fib(i0);
225   i1 = p0;
226   i2 = 4294967294u;
227   i1 += i2;
228   i1 = fib(i1);
229   i0 += i1;
230   Bfunc:;
231   FUNC_EPILOGUE;
232   return i0;
233 }
```

Workshop tools & codes

If you want reproduce at home ;)

- Ewasm contracts + TIPS
  - https://github.com/pventuzelo/reversing_ewasm_contract_101

Reversing Ewasm contract 101

Workshop given at EthCC 2020

Tool installation

Install octopus locally

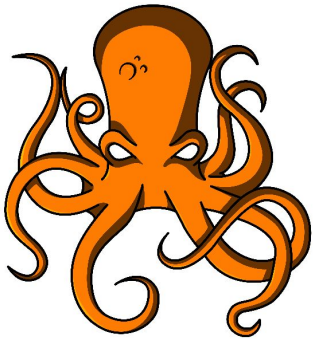
```
# Security Analysis tool for WebAssembly module and Blockchain Smart Contracts
git clone https://github.com/quoscient/octopus
```

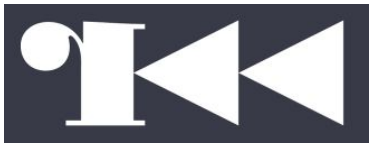
Follow the installation guide here

Install radare2 & Cutter

```
# Install Radare2
git clone https://github.com/radareorg/radare2
cd radare2
./sys/install.sh
```

Download Cutter here



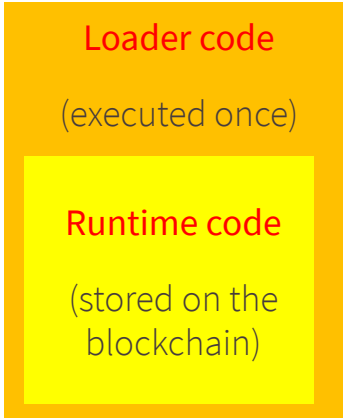


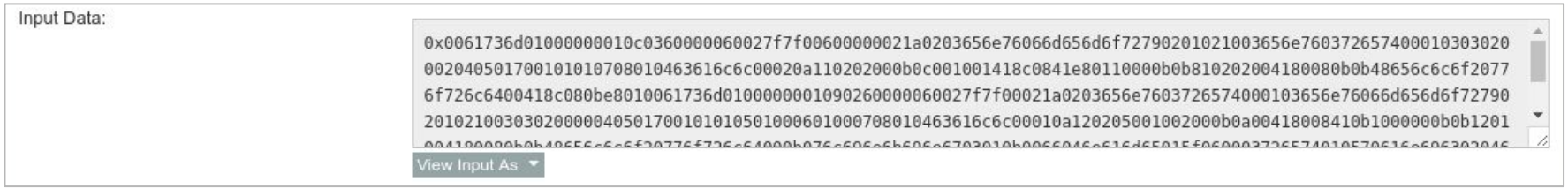
Example #1
Deployment bytecode

Deployment bytecode - Purpose

- What is the deployment bytecode ?
  - **Input data** of the **transaction that create the smart contract**

- Composed in 2 parts
  - **Loader** bytecode + embedded **runtime code**
  - Loader code
    - store the runtime code on the blockchain
  - Runtime code
    - Contract logic
    - Executed each time there is a Tx with this contract
  - Exactly the same way than with EVM

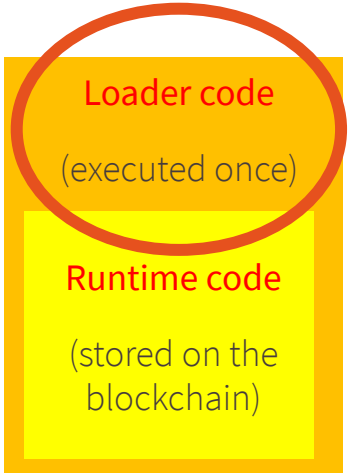




Deployment bytecode - Purpose

- What is the deployment bytecode ?
  - **Input data** of the **transaction that create the smart contract**

- Composed in 2 parts
  - **Loader** bytecode + embedded **runtime code**
  - Loader code
    - store the runtime code on the blockchain
  - Runtime code
    - Contract logic
    - Executed each time there is a Tx with this contract
  - Exactly the same way than with EVM



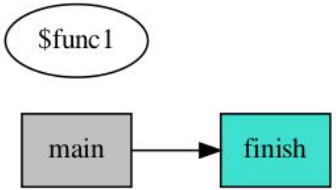
Input Data:

0x0061736d01000000010c0360000060027f7f00600000021a0203656e76066d656d6f7279020102100365e760372657400010303020
0020405017001010107080104636 16c6c00020a110202000b0c001001418c0841e80110000b0b810202004180080b0b48656c6c6f2077
6f726c6400418c080be8010061736d0100000001090260000060027f7f00021a0203656e760372657400010365e76066d656d6f72790
2010210030302000004050170010101050100060100070801046361 6c6c00010a120205001002000b0a00418008410b1000000b0b1201

View Input As

Deployment contract - Behavior analysis

- Simple Ewasm module
  - 2 internal functions
    - One is exported (**`main`**)
  - 2 exported symbols
    - `main & memory`
  - 1 imported symbols
    - `ethereum.finish`

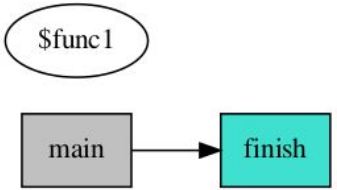



```
(module
  (type (;0;) (func (param i32 i32)))
  (type (;1;) (func))
  (import "ethereum" "finish" (func (;0;) (type 0)))
  (func (;1;) (type 1))
  (func (;2;) (type 1)
    i32.const 1024
    i32.const 1513
    call 0)
  (memory (;0;) 2)
  (global (;0;) (mut i32) (i32.const 68080))
  (global (;1;) i32 (i32.const 68080))
  (global (;2;) i32 (i32.const 2537))
  (export "memory" (memory 0))
  (export "main" (func 2))
  (data (;0;) (i32.const 1024) "\00asm\01\00\00\00\01&\07`
    \02\7f\7f\00`\00\01\7f`\03\7f\7f\7f\00`\01\7f\00`\00
    \00`\05\7f~~~~\00`\04~~~~\00\02\9f\01\07\08ethereum\06
    finish\00\00\08ethereum\0fgetCallDataSize\00\01\08
    ethereum\0ccallDataCopy\00\02\08ethereum\06revert\00
    \00\08ethereum\0cgetCallValue\00\03\08ethereum\0c
    storageStore\00\00\08ethereum\0bstorageLoad\00\00\03
    \06\05\04\05\06\03\04\04\05\01p\01\01\01\05\03\01\00
    \02\06\15\03\7f\01A\a0\88\04\0b\7f\00A\a0\88\04\0b\7f
    \00A\97\08\0b\07\11\02\06memory\02\00\04main\00\0b\0a
    \b6\09\05\02\00\0b\8d\03\00 \00 \04B8\88 \04B8\86\84
    \04B(\86B\80\80\80\80\80\80\c0\ff\00\83\84 \04B\18\86B
    \80\80\80\80\80\e0?\83\84 \04B\08\86B\80\80\80\80\f0
    \1f\83\84 \04B\08\88B\80\80\80\f8\0f\83\84 \04B\18\88B
    \80\80\fc\07\83\84 \04B(\88B\80\fe\03\83\847\03\00 \00
    A\18j \01B8\86 \01B(\86B\80\80\80\80\80\80\c0\ff\00\83
    \84 \01B\18\86B\80\80\80\80\80\e0?\83\84 \01B\08\86B
    \80\80\80\80\f0\1f\83\84 \01B\08\88B\80\80\80\f8\0f\83
```

[...]

Deployment contract - Behavior analysis

- Simple Ewasm module
  - 2 internal functions
    - One is exported (**`main`**)
  - 2 exported symbols
    - `main & memory`
  - 1 imported symbols
    - `ethereum.finish`



- Data section
  - **contains runtime Ewasm module**
  - TIP: start with wasm magic number: `'\00asm'`

- Main function
  - call to **`ethereum.finish(i32, i32)`**
    - 1st arg: **pointer** to runtime Ewasm module
    - 2nd arg: **size** of runtime Ewasm module

- **Why this Ewasm module is not optimized?**

Runtime module

```
(module
  (type (;0;) (func (param i32 i32)))
  (type (;1;) (func))
  (import "ethereum" "finish" (func (;0;) (type 0)))
  (func (;1;) (type 1))
  (func (;2;) (type 1)
    i32.const 1024
    i32.const 1513
    call 0)
  (memory (;0;) 2)
  (global (;0;) (mut i32) (i32.const 68080))
  (global (;1;) i32 (i32.const 68080))
  (global (;2;) i32 (i32.const 2537))
  (export "memory" (memory 0))
  (export "main" (func 2))
  (data (;0;) (i32.const 1024) "\00asm\01\00\00\00\01&\07`
    \02\7f\7f\00`\00\01\7f`\03\7f\7f\7f\00`\01\7f\00`\00
    \00`\05\7f~~~~\00`\04~~~~\00\02\9f\01\07\08ethereum\06
    finish\00\00\08ethereum\0fgetCallDataSize\00\01\08
    ethereum\0ccallDataCopy\00\02\08ethereum\06revert\00
    \00\08ethereum\0cgetCallValue\00\03\08ethereum\0c
    storageStore\00\00\08ethereum\0bstorageLoad\00\00\03
    \06\05\04\05\06\03\04\04\05\01p\01\01\01\05\03\01\00
    \02\06\15\03\7f\01A\a0\88\04\0b\7f\00A\a0\88\04\0b\7f
    \00A\97\08\0b\07\11\02\06memory\02\00\04main\00\0b\0a
    \b6\09\05\02\00\0b\8d\03\00 \00 \04B8\88 \04B8\86\84
    \04B(\86B\80\80\80\80\80\80\c0\ff\00\83\84 \04B\18\86B
    \80\80\80\80\80\e0?\83\84 \04B\08\86B\80\80\80\80\f0
    \1f\83\84 \04B\08\88B\80\80\80\f8\0f\83\84 \04B\18\88B
    \80\80\fc\07\83\84 \04B(\88B\80\fe\03\83\847\03\00 \00
    A\18j \01B8\86 \01B(\86B\80\80\80\80\80\80\c0\ff\00\83
    \84 \01B\18\86B\80\80\80\80\80\e0?\83\84 \01B\08\86B
    \80\80\80\80\f0\1f\83\84 \01B\08\88B\80\80\80\f8\0f\83
```

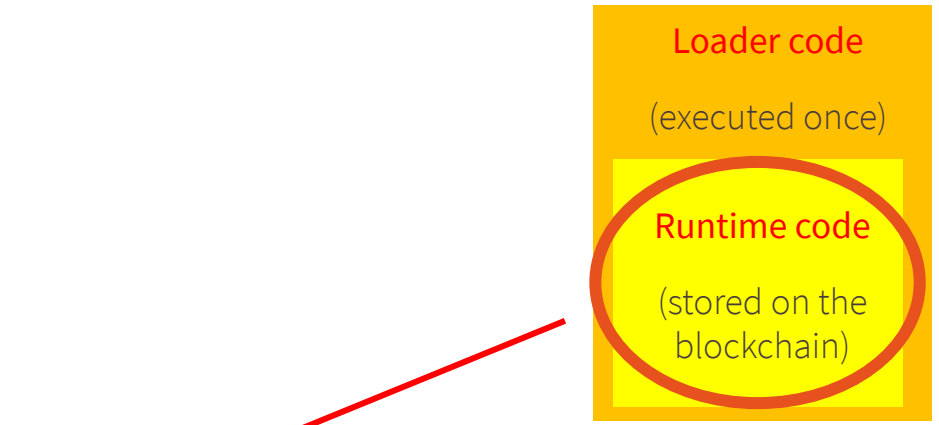
[...]

Example #2

Storage Ewasm contract

Storage Ewasm contract - Analysis

```
(module
  (type (;0;) (func (param i32 i32)))
  (type (;1;) (func))
  (import "ethereum" "finish" (func (;0;) (type 0)))
  (func (;1;) (type 1))
  (func (;2;) (type 1)
    i32.const 1024
    i32.const 1513
    call 0)
  (memory (;0;) 2)
  (global (;0;) (mut i32) (i32.const 68080))
  (global (;1;) i32 (i32.const 68080))
  (global (;2;) i32 (i32.const 2537))
  (export "memory" (memory 0))
  (export "main" (func 2))
  (data (;0;) (i32.const 1024) "\00asm\01\00\00\00\01&\07`
    \02\7f\7f\00`\00\01\7f`\03\7f\7f\7f\00`\01\7f\00`\00
    \00`\05\7f~~~~\00`\04~~~~\00\02\9f\01\07\08ethereum\06
    finish\00\00\08ethereum\0fgetCallDataSize\00\01\08
    ethereum\0ccallDataCopy\00\02\08ethereum\06revert\00
    \00\08ethereum\0cgetCallValue\00\03\08ethereum\0c
    storageStore\00\00\08ethereum\0bstorageLoad\00\00\03
    \06\05\04\05\06\03\04\04\05\01p\01\01\01\05\03\01\00
    \02\06\15\03\7f\01A\a0\88\04\0b\7f\00A\a0\88\04\0b\7f
    \00A\97\08\0b\07\11\02\06memory\02\00\04main\00\0b\0a
    \b6\09\05\02\00\0b\8d\03\00 \00 \04B8\88 \04B8\86\84
    \04B(\86B\80\80\80\80\80\80\c0\ff\00\83\84 \04B\18\86B
    \80\80\80\80\80\e0?\83\84 \04B\08\86B\80\80\80\80\f0
    \1f\83\84 \04B\08\88B\80\80\80\f8\0f\83\84 \04B\18\88B
    \80\80\fc\07\83\84 \04B(\88B\80\fe\03\83\847\03\00 \00
    A\18j \01B8\86 \01B(\86B\80\80\80\80\80\c0\ff\00\83
    \84 \01B\18\86B\80\80\80\80\80\e0?\83\84 \01B\08\86B
    \80\80\80\80\f0\1f\83\84 \01B\08\88B\80\80\80\f8\0f\83
```



Storage Ewasm contract - Quick analysis

- 7 imported symbols
  - `finish, revert`
  - `storageStore, storageLoad`
  - `etc.`
- 2 exported symbols
  - `main & memory`

```
(module
  (type (;0;) (func (param i32 i32)))
  (type (;1;) (func (result i32)))
  (type (;2;) (func (param i32 i32 i32)))
  (type (;3;) (func (param i32)))
  (type (;4;) (func))
  (type (;5;) (func (param i32 i64 i64 i64 i64)))
  (type (;6;) (func (param i64 i64 i64 i64)))
  (import "ethereum" "finish" (func (;0;) (type 0)))
  (import "ethereum" "getCallDataSize" (func (;1;) (type 1)))
  (import "ethereum" "callDataCopy" (func (;2;) (type 2)))
  (import "ethereum" "revert" (func (;3;) (type 0)))
  (import "ethereum" "getCallValue" (func (;4;) (type 3)))
  (import "ethereum" "storageStore" (func (;5;) (type 0)))
  (import "ethereum" "storageLoad" (func (;6;) (type 0)))
  (memory (;0;) 2)
  (global (;0;) (mut i32) (i32.const 66592))
  (global (;1;) i32 (i32.const 66592))
  (global (;2;) i32 (i32.const 1047))
  (export "memory" (memory 0))
  (export "main" (func 11))
  (data (;0;) (i32.const 1024) "Function is not payable")
  (func (;7;) (type 4))
  (func (;8;) (type 5) (param i32 i64 i64 i64 i64) ...
  (func (;9;) (type 6) (param i64 i64 i64 i64) ...
  (func (;10;) (type 3) (param i32) ...
  (func (;11;) (type 4) ...
```

Storage Ewasm contract - Quick analysis

- 7 imported symbols
  - `finish, revert`
  - `storageStore, storageLoad`
  - `etc.`
- 2 exported symbols
  - `main & memory`

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

- Where is **`set()`** and **`get()`** functions?

```
(module
  (type (;0;) (func (param i32 i32)))
  (type (;1;) (func (result i32)))
  (type (;2;) (func (param i32 i32 i32)))
  (type (;3;) (func (param i32)))
  (type (;4;) (func))
  (type (;5;) (func (param i32 i64 i64 i64 i64)))
  (type (;6;) (func (param i64 i64 i64 i64)))
  (import "ethereum" "finish" (func (;0;) (type 0)))
  (import "ethereum" "getCallDataSize" (func (;1;) (type 1)))
  (import "ethereum" "callDataCopy" (func (;2;) (type 2)))
  (import "ethereum" "revert" (func (;3;) (type 0)))
  (import "ethereum" "getCallValue" (func (;4;) (type 3)))
  (import "ethereum" "storageStore" (func (;5;) (type 0)))
  (import "ethereum" "storageLoad" (func (;6;) (type 0)))
  (memory (;0;) 2)
  (global (;0;) (mut i32) (i32.const 66592))
  (global (;1;) i32 (i32.const 66592))
  (global (;2;) i32 (i32.const 1047))
  (export "memory" (memory 0))
  (export "main" (func 11))
  (data (;0;) (i32.const 1024) "Function is not payable")
  (func (;7;) (type 4))
  (func (;8;) (type 5) (param i32 i64 i64 i64 i64) ...
  (func (;9;) (type 6) (param i64 i64 i64 i64) ...
  (func (;10;) (type 3) (param i32) ...
  (func (;11;) (type 4) ...
```

Storage Ewasm contract - Quick analysis

- 7 imported symbols
  - `finish, revert`
  - `storageStore, storageLoad`
  - `etc.`
- 2 exported symbols
  - `main & memory`

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

- Where is **`set()`** and **`get()`** functions?
  - **dispatcher function**

```
(module
  (type (;0;) (func (param i32 i32)))
  (type (;1;) (func (result i32)))
  (type (;2;) (func (param i32 i32 i32)))
  (type (;3;) (func (param i32)))
  (type (;4;) (func))
  (type (;5;) (func (param i32 i64 i64 i64 i64)))
  (type (;6;) (func (param i64 i64 i64 i64)))
  (import "ethereum" "finish" (func (;0;) (type 0)))
  (import "ethereum" "getCallDataSize" (func (;1;) (type 1)))
  (import "ethereum" "callDataCopy" (func (;2;) (type 2)))
  (import "ethereum" "revert" (func (;3;) (type 0)))
  (import "ethereum" "getCallValue" (func (;4;) (type 3)))
  (import "ethereum" "storageStore" (func (;5;) (type 0)))
  (import "ethereum" "storageLoad" (func (;6;) (type 0)))
  (memory (;0;) 2)
  (global (;0;) (mut i32) (i32.const 66592))
  (global (;1;) i32 (i32.const 66592))
  (global (;2;) i32 (i32.const 1047))
  (export "memory" (memory 0))
  (export "main" (func 11))
  (data (;0;) (i32.const 1024) "Function is not payable")
  (func (;7;) (type 4))
  (func (;8;) (type 5) (param i32 i64 i64 i64 i64) ...
  (func (;9;) (type 6) (param i64 i64 i64 i64) ...
  (func (;10;) (type 3) (param i32) ...
  (func (;11;) (type 4) ...
```

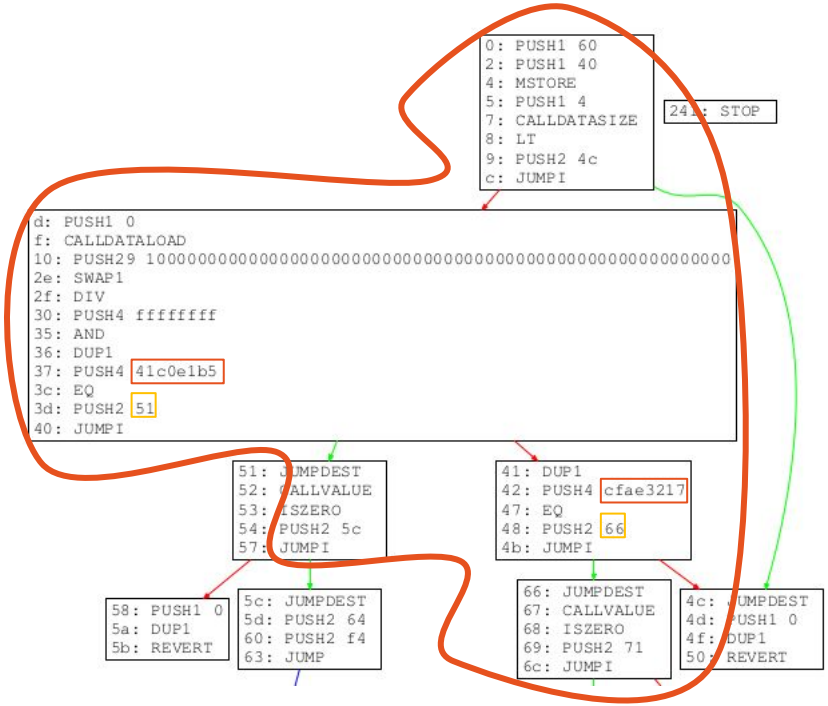
Quick reminder: EVM contract dispatcher function

- Runtime code entry point is a **Dispatcher function**
  - Switch on the first 4 bytes of the transaction payload
  - execute the associated code of the given function signature.

- Two functions signatures here:
  - **`41c0e1b5`**
  - **`cfae3217`**

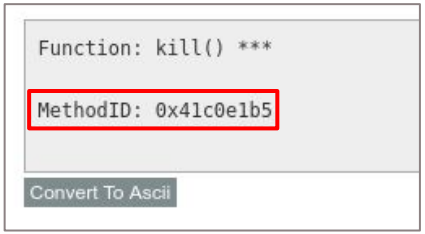
```
41: DUP1
42: PUSH4   FUNC_HASH
47: EQ
48: PUSH2   FUNC_OFFSET
4b: JUMPI
```

- See my talk at Devcon 4:
  - Reversing Ethereum Smart Contracts to find out what's behind EVM bytecode - slides / video



Quick reminder: Function's signature reverse lookup

- When you interact with a contract:
  - **You send the function signature (MethodID) followed by the arguments**
  - Signature, Argument #1, Argument #2 (256-bits words)



Search Signatures 0x70a08231 Search

ID	text signature	bytes signature
31808	distributeTokens(address[],uint256[])	0x4bd09c2a
31807	distributeTokens(address[],uint256)	0x256fa241
31806	finishMinting(address)	0x76192200
31805	salvageTokens(address,uint256)	0xaf303a11
31804	finishSalvage(address)	0xe63b029d
31803	setSalvageable(address,bool)	0xc9206ddf
31802	freezeAccounts(address[],bool)	0xc341b9f6
31801	lockAccounts(address[],uint256)	0xe5ac7291
31800	isUnlockedBoth(address)	0x5789baa5
31799	isUnlocked(address)	0x2bbf532a

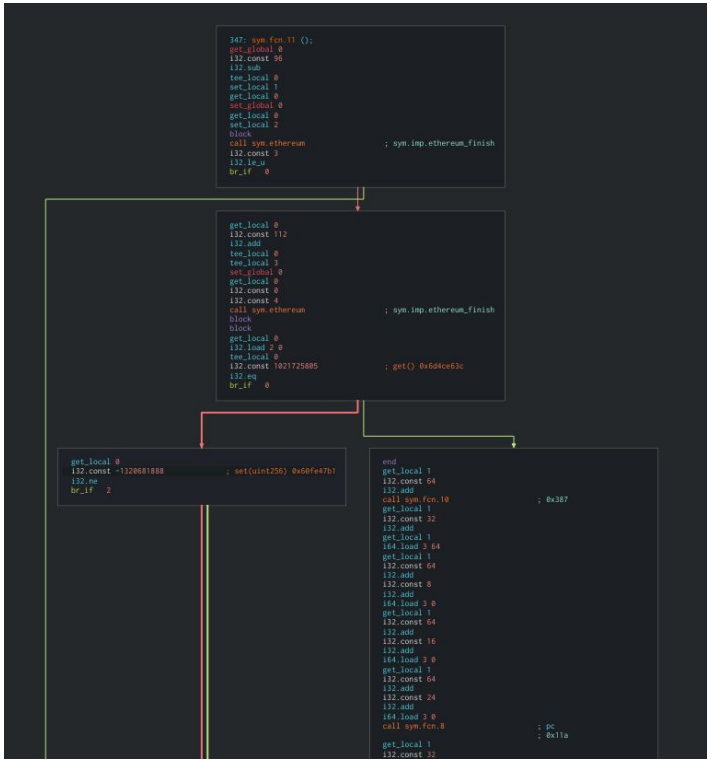
https://www.4byte.directory/signatures/

Storage Ewasm contract - Dispatcher function

- **`main`** function:
  - **dispatcher function**
  - 7 basic blocks
  - 175 instructions

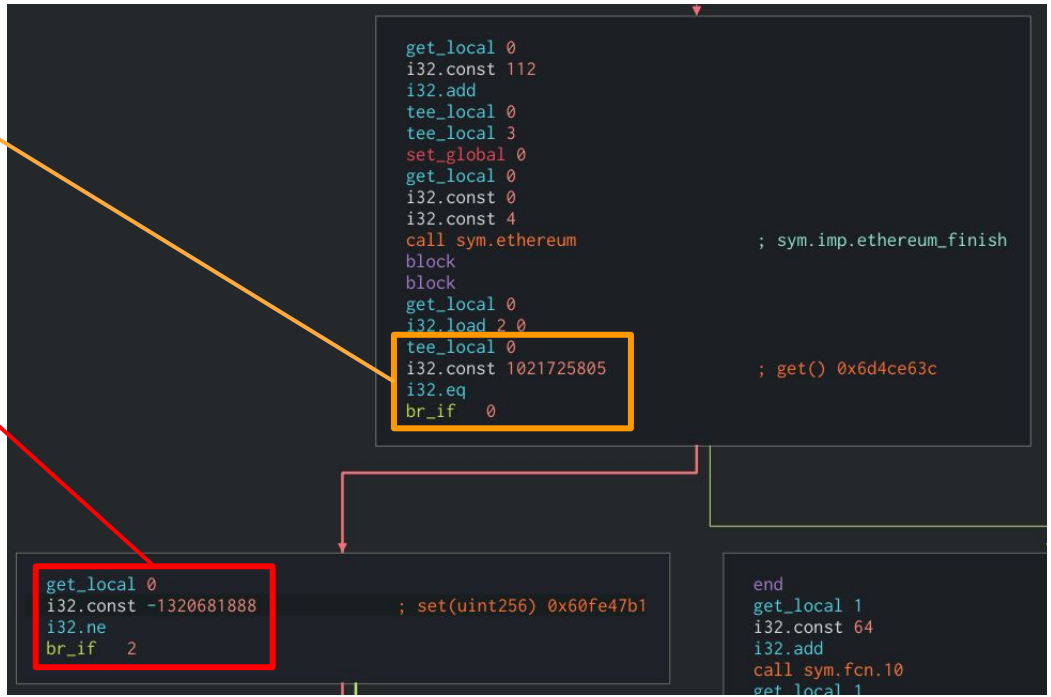
- We should find 2 public functions
  - **`get()`**
  - **`set(uint)`**

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```



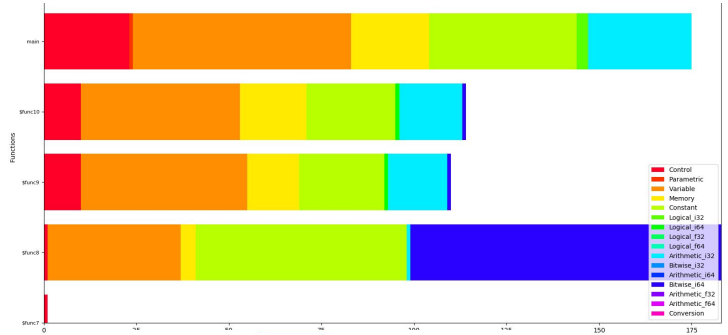
Storage Ewasm contract - Function's signature

- `i32.const 1021725805`
  - sig: 0x6d4ce63c
  - **`get()`**
- `i32.const -1320681888`
  - sig: 0x60fe47b1
  - **`set(uint256)`**
- i32 to hex signature
  - byteorder = little-endian
  - Online converter here

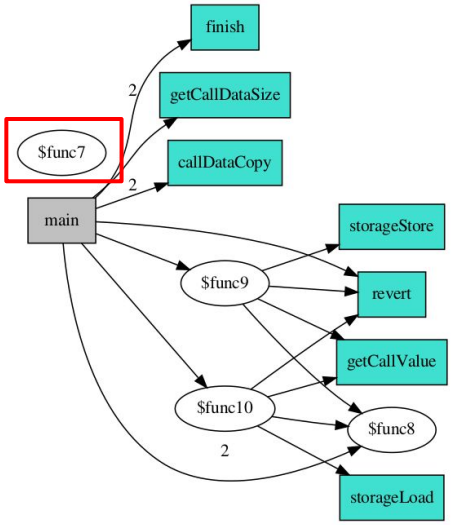


Storage Ewasm contract - Optimization

- Instructions analytics using [Octopus]
  - **Visual analytics about types of instructions per functions** inside the WebAssembly module
  - `./octopus_wasm.py -y -f contract.wasm`
  - Give you quick information about:
    - **Number** of functions
    - **Size** of the functions (number of instructions)
    - **Type** of instructions per functions



- Call graph
  - `./octopus_wasm.py -c -f contract.wasm`
  - can help us to find directly not optimized code
  - In this contract:
    - **`$func7` is not reachable**
    - worst is a completely useless function



Example #3

ERC-20 Ewasm contract

ERC-20 Ewasm contract

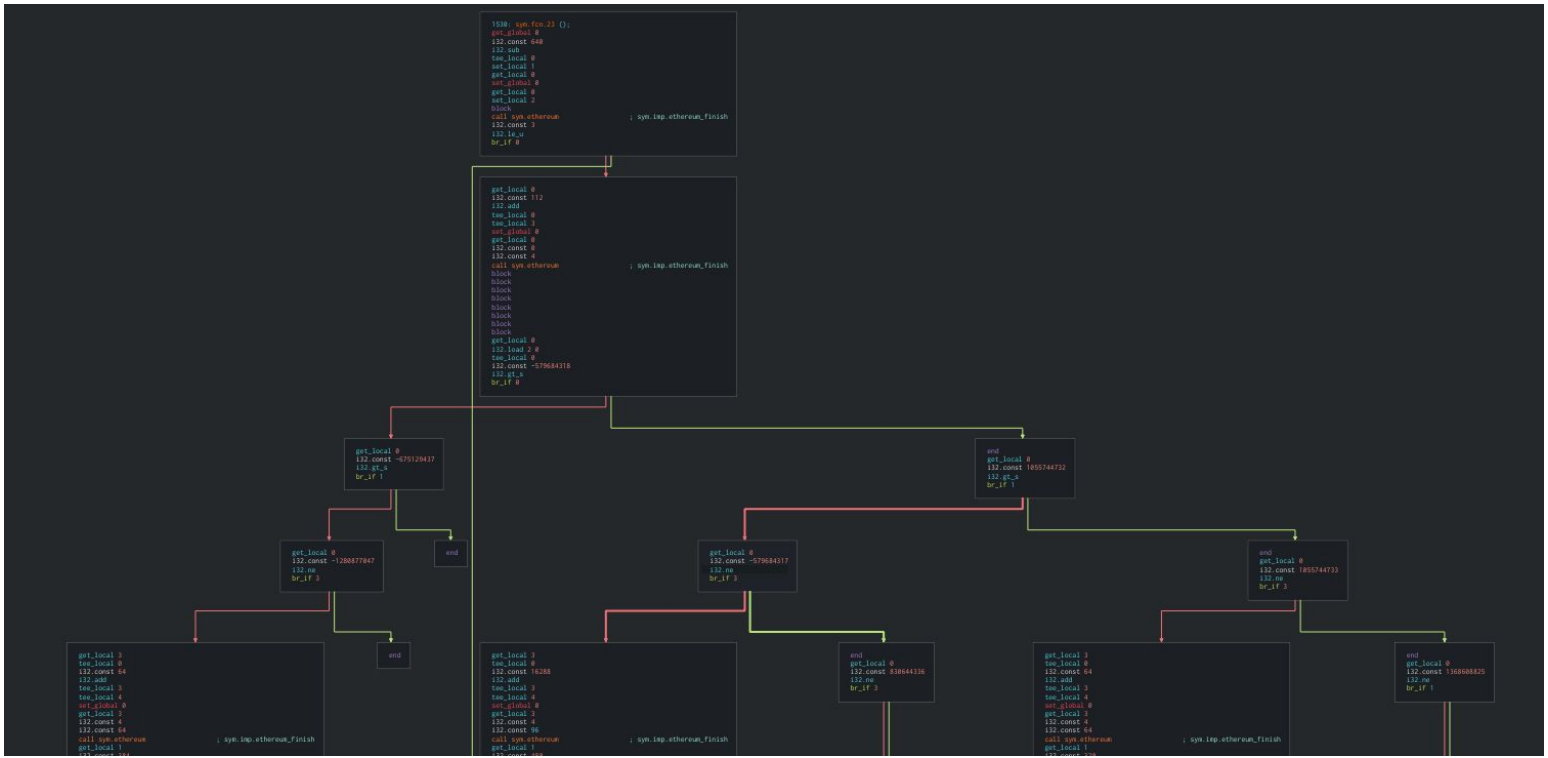
- 11 imported symbols
  - `finish, revert`
  - `storageStore, storageLoad`
  - `getCaller, getCallValue`
- 2 exported symbols
  - `main & memory`

- Data section
  - **contains interesting strings**
  - `ERC20: approve/transfert …`
  - `SafeMath: … overflow`

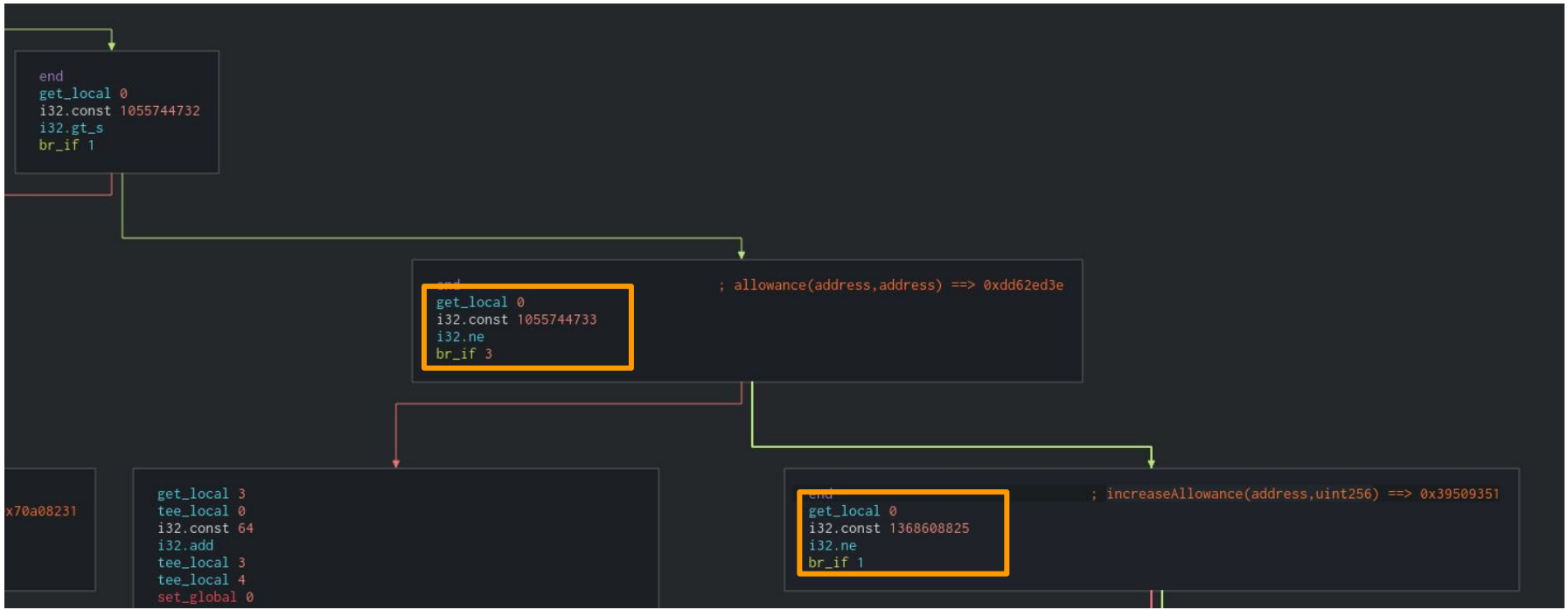
- More complexe than Storage contract
  - More & Bigger functions
  - More arguments (i64)
  - Call graph more complexe

```
(module
  (type (;0;) (func (param i32 i32)))
  (type (;1;) (func (result i32)))
  (type (;2;) (func (param i32 i32 i32)))
  (type (;3;) (func (param i32)))
  (type (;4;) (func (result i64)))
  (type (;5;) (func (param i64 i32 i32 i32) (result i32)))
  (type (;6;) (func))
  (type (;7;) (func (param i32 i64 i64 i64 i64)))
  (type (;8;) (func (param i64 i64 i64 i64 i64 i64 i64)))
  (type (;9;) (func (param i64 i64 i64 i64 i64 i64 i64 i64 i64 i64)))
  (type (;10;) (func (param i32 i64 i64 i64 i64 i64 i64)))
  (type (;11;) (func (param i32 i64 i64 i64)))
  (import "ethereum" "finish" (func (;0;) (type 0)))
  (import "ethereum" "getCallDataSize" (func (;1;) (type 1)))
  (import "ethereum" "callDataCopy" (func (;2;) (type 2)))
  (import "ethereum" "revert" (func (;3;) (type 0)))
  (import "ethereum" "getCallValue" (func (;4;) (type 3)))
  (import "ethereum" "storageLoad" (func (;5;) (type 0)))
  (import "ethereum" "getGasLeft" (func (;6;) (type 4)))
  (import "ethereum" "callStatic" (func (;7;) (type 5)))
  (import "ethereum" "returnDataCopy" (func (;8;) (type 2)))
  (import "ethereum" "getCaller" (func (;9;) (type 3)))
  (import "ethereum" "storageStore" (func (;10;) (type 0)))
  (memory (;0;) 2)
  (global (;0;) (mut i32) (i32.const 66784))
  (global (;1;) i32 (i32.const 66784))
  (global (;2;) i32 (i32.const 1246))
  (export "memory" (memory 0))
  (export "main" (func 23))
  (data (;0;) (i32.const 1024) "ERC20: approve from the zero
    addressERC20: approve to the zero addressSafeMath: subtraction
    overflowSafeMath: addition overflowERC20: transfer from the zero
    addressERC20: transfer to the zero addressFunction is not payable")
  (func (;11;) (type 6))
  (func (;12;) (type 7) (param i32 i64 i64 i64 i64) ...
 216 (func (;13;) (type 8) (param i64 i64 i64 i64 i64 i64 i64) ...
 306 (func (;14;) (type 9) (param i64 i64 i64 i64 i64 i64 i64 i64 i64 i64) ...
 975 (func (;15;) (type 8) (param i64 i64 i64 i64 i64 i64 i64) ...
1625 (func (;16;) (type 10) (param i32 i64 i64 i64 i64 i64 i64) ...
2087 (func (;17;) (type 8) (param i64 i64 i64 i64 i64 i64 i64) ...
2177 (func (;18;) (type 11) (param i32 i64 i64 i64) ...
2474 (func (;19;) (type 3) (param i32) ...
2589 (func (;20;) (type 8) (param i64 i64 i64 i64 i64 i64 i64) ...
3243 (func (;21;) (type 9) (param i64 i64 i64 i64 i64 i64 i64 i64 i64 i64) ...
4421 (func (;22;) (type 9) (param i64 i64 i64 i64 i64 i64 i64 i64 i64 i64) ...
4907 (func (;23;) (type 6) ...
5942 (table (;0;) 1 1 funcref)
5943 )
```

ERC-20 Ewasm contract - dispatcher function



ERC-20 Ewasm contract - Function's signature



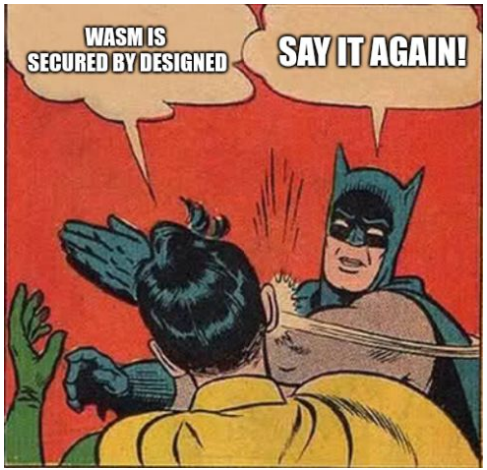
Conclusion

Conclusion

- Ewasm contracts in my opinion:
  - are easier to analyze than EVM contract
  - allow us to leverage on existing tools for wasm
  - can bring new developers/adepts in Ethereum

- Ewasm Gas metering
  - can have a huge impact on performance
  - **Gas metering optimizer tool should be created**
  - “standard library” like safemath should be:
    - optimized and maybe precompiled

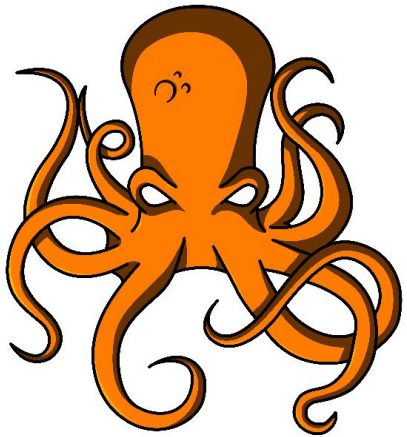
- Caution !!! Old vulnerabilities can still exists
  - Integer overflow/underflows
  - Time Of Check to Time Of Use (TOCTOU)
  - Reentrancy
  - etc.

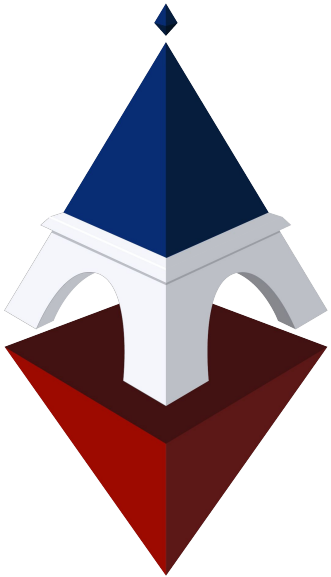


Thanks for your attention

Patrick Ventuzelo / @Pat_Ventuzelo / ventuzelo.patrick@gmail.com

- Twitter / LinkedIn / Github / Blog
- Octopus:
  - https://github.com/pventuzelo/octopus





Security trainings

WebAssembly Security
“From Reversing to Vulnerability Research”
(4-5 days)

- Introduction
- WebAssembly reversing
- Static analysis
- Dynamic analysis (DBI)
- Debugging
- (De-)Obfuscation
- Wasm games hacking
- Finding wasm Module vulnerabilities
- Emscripten & NodeJS exploit
- Wasm Module fuzzing
- Fuzzing Web-Browsers (V8, Webkit, ...)
- Fuzzing WebAssembly VMs (C/C++,Rust, ...)

https://webassembly-security.com/trainings

Rustlang security
“For Hackers and Developers”
(2-3 days)

- Introduction
- Rustlang vulnerabilities
- Panicking macros
- Unsafe codes
- Auditing tools
- Debugging (lldg, gdb, ...)

- Fuzzing (afl, hongfuzz, libfuzzer, ...)
- Triaging / Bugs analysis
- Code coverage
- Sanitizers (ASAN, MSAN, ...)
- Symbolic execution

https://webassembly-security.com/rust-security-training/