Wasabi:
A Framework for Dynamically Analyzing WebAssembly

http://wasabi.software-lab.org

Daniel Lehmann and Michael Pradel
TU Darmstadt, Germany
TL;DR

- **WebAssembly**: bytecode for the web
  - New and important platform
    - Need for tooling

- **Wasabi**: dynamic analysis framework for WebAssembly
  - Observe any operation
  - Analysis API in JavaScript
  - Binary instrumentation
  - Open source: http://wasabi.software-lab.org
WebAssembly in a Nutshell

• [Haas et al., PLDI 2017]

• **Fast**: within 1.5x – 1.9x of native
  • Binary format: compact, quick to parse
  • Instructions map closely to hardware
  • No GC, predictable performance

• **Safe**: static types, separated code and data, ...

• **Portable**: all major browsers, ARM/x86
WebAssembly in a Nutshell

• Designed as a **compilation target**
  • C/C++ via Emscripten
  • Rust, Go, ...

• Many **use cases:**
  • Alternative to JavaScript on the client
  • Audio/video processing, compression, machine learning
  • Games
  • ...

Unreal Engine 4: Zen Garden demo
Dynamic Analysis Frameworks

• New platform ➔ Need for **dynamic analysis** tools

  ![Checkmark] Correctness  ![Timer] Performance  ![Shield] Security

• **Frameworks** as a basis

<table>
<thead>
<tr>
<th></th>
<th>Pin</th>
<th>Valgrind</th>
<th>RoadRunner</th>
<th>Jalangi</th>
<th>Wasabi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>x86-64</td>
<td>JVM</td>
<td>JVM</td>
<td>JavaScript</td>
<td>WebAssembly</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>native binaries</td>
<td>byte code</td>
<td>source code</td>
<td></td>
<td>binary code</td>
</tr>
<tr>
<td>Analysis Language</td>
<td>C/C++</td>
<td>Java</td>
<td>JavaScript</td>
<td></td>
<td>JavaScript</td>
</tr>
</tbody>
</table>
Wasabi Overview

Instrumentation Phase

Input Program → Wasabi → Instrumented Program

Analysis Phase

Instrumented Program → Low-level Hooks → Meta Information → Client Analysis

Wasabi

Low-level Hooks

Meta Information

Client Analysis

Wasabi

Low-level Hooks

Meta Information

Client Analysis

Wasabi

Low-level Hooks

Meta Information

Client Analysis

Wasabi

Low-level Hooks

Meta Information

Client Analysis

Wasabi

Low-level Hooks

Meta Information

Client Analysis

Wasabi

Low-level Hooks

Meta Information

Client Analysis

Wasabi

Low-level Hooks

Meta Information

Client Analysis

Wasabi

Low-level Hooks

Meta Information

Client Analysis
Client Analysis Example

• Analysis in **JavaScript**
  • High-level language
  • Familiar to web developers

• E.g., **crypto miner detection**
  • [Wang et al., ESORICS ’18]
  • Gather instruction profile
  • 11 LOC
  • No manual instrumentation

```javascript
let inst = {};
Wasabi.binary = function(loc, op, args) {
    switch (op) {
        case "i32.add":
        case "i32.and":
        case "i32.shl":
        case "i32.shr_u":
        case "i32.xor":
            inst[op] = (inst[op] || 0)+1;
    }
};
```
Client Analysis API

- **Every instruction** can be observed
  - Location, inputs, outputs
- **Grouping** of instructions
  - Similar instructions have single hook
  - 23 hooks instead of >100
- **Statically computed** information
  - E.g., resolve relative branch targets
- **Type mapping:**
  WebAssembly → JavaScript

<table>
<thead>
<tr>
<th>Hook</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>loc, func, args...</td>
</tr>
<tr>
<td>binary</td>
<td>loc, op, arg1, arg2, result</td>
</tr>
<tr>
<td>return</td>
<td>loc, results...</td>
</tr>
<tr>
<td>br_if</td>
<td>loc, target, condition</td>
</tr>
</tbody>
</table>

```

f32 → number
f64
i32 → boolean
i64 → long.js object
```

8
Wasabi Overview

Input Program → Wasabi → Instrumented Program → Low-level Hooks → Meta Information → Client Analysis

Instrumentation Phase

Analysis Phase

Wasabi

Instrument

generate

extract

calls

WS

JC

JS
Static Binary Instrumentation

- **Why binary, why static?**
  - **Different producers** of WebAssembly
  - **Source code** not always available
  - Static instrumentation is **reliable**

<table>
<thead>
<tr>
<th>Original Code</th>
<th>Instrumented Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>f32.abs</code></td>
<td>copy argument to local_arg</td>
</tr>
<tr>
<td></td>
<td><code>f32.abs</code></td>
</tr>
<tr>
<td></td>
<td>duplicate stack top</td>
</tr>
<tr>
<td></td>
<td>copy local_arg to stack</td>
</tr>
<tr>
<td></td>
<td>call hook_f32.abs</td>
</tr>
</tbody>
</table>

Add WebAssembly function:

```plaintext
fun hook_f32.abs(
    arg: f32,
    res: f32)
{
    ...
}
```

Add local: `local_arg: f32`
Instrumentation Challenges

- **Polymorphic** instructions: drop, select, ...
  - Instrumentation must do type checking

- But functions are **monomorphic** (= fixed type)
  - Monomorphization: 1 hook per concrete type
  - Infinitely many type **combinations** for call, return
  - On-demand: only for types that appear in the binary

- Other challenges
  - Dynamic block nesting, resolving branch labels, handling i64s
Low-Level Hooks

- Bridge between WebAssembly and JavaScript

**Instrumented Program:**

```
hook_drop_f32
```

**Low-Level Hooks:**

```
fun hook_drop_f32(arg) {
  call Wasabi.drop(...)
}
```

```
fun hook_drop_i64(arg) {
  convert i64 arg
  call Wasabi.drop(...)
}
```

**(High-Level) Client Analysis:**

```
Wasabi.drop = (... => {
...
}
```
Evaluation

• Program Test Set
• Example Analyses
• Instrumentation Overhead
  • Code size
  • Runtime
Program Test Set

<table>
<thead>
<tr>
<th>Program</th>
<th># Instructions</th>
<th>Time to Instr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolyBench/C</td>
<td>(mean =) 23 772</td>
<td>23 ms</td>
</tr>
<tr>
<td>PSPDFKit</td>
<td>7 178 854</td>
<td>5.1 s</td>
</tr>
<tr>
<td>Unreal Engine 4</td>
<td>20 603 058</td>
<td>15.5 s</td>
</tr>
</tbody>
</table>

- Also tested on WebAssembly **spec test suite**
## Example Analyses

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Hooks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction coverage</td>
<td>all</td>
</tr>
<tr>
<td>Branch coverage</td>
<td>if, br_if, br_table, select</td>
</tr>
<tr>
<td>Call graph extraction</td>
<td>call_pre</td>
</tr>
<tr>
<td>Instruction mix</td>
<td>all</td>
</tr>
<tr>
<td>Basic block profiling</td>
<td>begin</td>
</tr>
<tr>
<td>Memory access tracing</td>
<td>load, store</td>
</tr>
<tr>
<td>Dynamic taint analysis</td>
<td>all</td>
</tr>
<tr>
<td>Crypto miner instruction profile</td>
<td>binary</td>
</tr>
</tbody>
</table>
Instrumentation Overhead

- Runtime
- Code size

Binary Size Increase

polybench (mean)
PSPDFKit
Unreal Engine 4

Relative Runtime

polybench
PSPDFKit
Unreal Engine 4
(all geomean)
Conclusion

• **WebAssembly**: bytecode for the web
  - New and important platform
  - Need for tooling

• **Wasabi**: dynamic analysis framework for WebAssembly
  - Observe any operation
  - Analysis API in JavaScript
  - Binary instrumentation
  - Open source: [http://wasabi.software-lab.org](http://wasabi.software-lab.org)