Swivel: Hardening WebAssembly against Spectre

Shravan Narayan, Craig Disselkoen, Daniel Moghimi, Sunjay Cauligi,
Evan Johnson, Zhao Gang, Anjo Vahldiek-Oberwagner, Ravi Sahita,
Hovav Shacham, Dean Tullsen, Deian Stefan
What is WebAssembly (Wasm)?

Platform-independent bytecode

Runs C/C++/Rust in the browser

Designed for isolation
Wasm is used outside the browser

**Securing Firefox with WebAssembly**

By Nathan Froyd

Securing Firefox with WebAssembly

Protecting the security and privacy of individuals is a central tenet of Mozilla’s mission, and so we constantly endeavor to make our users safer online. With a...

So today, we’re adding a third approach to our arsenal: RLBox, a new sandboxing technology developed by researchers at the University of California San Diego, the University of Texas, Austin, and Stanford University, allows us to quickly and efficiently convert existing Firefox components to run inside a...

**WebAssembly on Cloudflare Workers**

By Kenton Verdts

Announcing Lucet: Fastly’s native WebAssembly compiler and runtime

Published March 28, 2019

Today, we’re thrilled to announce the open sourcing of Lucet, Fastly’s native WebAssembly compiler and runtime. WebAssembly is a technology created to enable web browsers to safely execute programs at near-native speeds. It has been shipping in the four major browsers since early 2017.

**Project Oak**

The goal of Project Oak is to create a specification and a reference implementation for the secure transfer, storage and processing of data.
Wasm on FaaS platforms

Faas host process

Wasm Client A

Wasm Client B

Wasm Client Z

Faas Runtime
How does Wasm enforce isolation?

- **Stack**
  - `local.get localidx`
  - `local.set localidx`

- **Linear Memory**
  - `load offset1 offset2`
  - `store offset1 offset2`

- **Globals**
  - `globals.get globalidx`
  - `globals.set globalidx`

- **Control Flow**
  - `load offset1 offset2`
  - `store offset1 offset2`

```c
void foo() {
  // Safe stack
  return;
}
```

```c
if (addr in heap)
    read(*addr);
else
    abort("OOB");
```

```c
if (CFI_valid(fn_ptr))
    fnptr();
else
    abort("CFI");
```
Problem: Spectre breaks Wasm isolation
Eg: Using Spectre-PHT to break isolation

```c
if (addr in heap) {
    x = read(*addr);
    y = read(x);
}
```

Expected: false
Predicted: true
Eg: Using Spectre-{BTB, RSB}

```
(*fnptr)();
return;
```

Expected: 0x1111111
Predicted: 0xbadc0de

Speculative JOP/ROP
Alternately: Poisoning → Self exfiltration
Solution: Add fences!
We tried this: it’s too slow!
Our solution: Swivel

Swivel is a Wasm compiler that prevents:
• Breakout and poisoning attacks via Spectre-{PHT, BTB, RSB}

Swivel has two backends:
• Swivel-SFI: safety using only software checks
• Swivel-CET: safety with existing hardware extensions, allows hyperthreading
Fundamental problem

Wasm safety checks: function granularity

Speculative control flow can start anywhere
• Can bypass security checks

We need a new abstraction when compiling Wasm

Wasm code

```
func_foo:
  ...
  mem_bounds_check <reg_mem>
  call bar
  ...
  load <reg_mem>
  jmp ...
```
Key abstraction: Linear blocks (LB)

Like basic blocks, except ...

- Instruction sequences that end in a jump or call instruction
- Must include safety checks within the block
- Checks are speculatively safe

Wasm code

```
func_foo:
  ...
  mem_bounds_check <reg_mem>
call bar
  ...
  load <reg_mem>
jmp ...
```

Wasm code with LB

```
linear_block_1:
  ...
  call bar

linear_block_2:
  ...
  safe_mem_bounds_mask <reg_mem>
  load <reg_mem>
jmp ...
```

1. Terminator is control flow / call inst
2. Checks are in same LB as instruction
3. Checks are speculatively safe masks
Swivel-SFI: Builds on Linear blocks (LB)

Spectre-PHT: LBs handles Spectre-PHT breakout attacks

Spectre-BTB: LBs ensure that BTB targets only predict LBs
  • Problem: BTB may not be empty when we enter the sandbox
  • Solution: Flush the BTB before entering the sandbox

Spectre-RSB: LBs ensure that RSB only predict LBs
  • Problem: RSB underflow → predict arbitrary target
  • Solution: Separate control stack + use jumps instead of returns
What about sandbox poisoning attacks?

But we can’t flush the conditional branch predictors (CBP)!
What about sandbox poisoning attacks?

**Swivel-SFI Deterministic**

- **Wasm Client A**
  - CBP to BTB
- **Wasm Client B**
  - CBP to BTB
- BTB flush
- FaaS Runtime

**Swivel-SFI ASLR**

- **Wasm Client A**
  - Randomize sandbox location
- **Wasm Client B**
  - Randomize sandbox location
- FaaS Runtime
Swivel’s security guarantees

<table>
<thead>
<tr>
<th>Attack variant</th>
<th>Swivel-SFI</th>
<th></th>
<th>Swivel-CET</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASLR</td>
<td>Det</td>
<td>ASLR</td>
<td>Det</td>
</tr>
<tr>
<td>Spectre-PHT</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>in-place</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>out-of-place</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>in-place</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>out-of-place</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>in-place</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>out-of-place</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>
Evaluation

Performance
• Standard benchmark suites – SPEC 2006
• Macro benchmark – mock FaaS platform with Swivel services
• Baseline: Stock (insecure) Wasm

Security
• Implemented POC’s for Spectre-{PHT, BTB, RSB}
SPEC 2006 benchmark

Swivel ASLR: < 10%
Swivel Det: 3% to 240%

Fences are too slow!
Std fence solutions: 6x to 19x
Min fence solutions: 2x to 5x
FaaS platform benchmark

<table>
<thead>
<tr>
<th>Swivel Protection</th>
<th>XML to JSON</th>
<th>Templatated HTML</th>
<th>Image classification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Throughput</td>
<td>Perf Loss</td>
<td>Throughput</td>
</tr>
<tr>
<td>Stock (unsafe)</td>
<td>531</td>
<td>-</td>
<td>4.81k</td>
</tr>
<tr>
<td>Swivel-SFI ASLR</td>
<td>459</td>
<td>13.6%</td>
<td>803</td>
</tr>
<tr>
<td>Swivel-SFI Det</td>
<td>350</td>
<td>34.1%</td>
<td>2.90k</td>
</tr>
<tr>
<td>Swivel-CET ASLR</td>
<td>498</td>
<td>6.2%</td>
<td>898</td>
</tr>
<tr>
<td>Swivel-CET Det</td>
<td>338</td>
<td>36.3%</td>
<td>3.50k</td>
</tr>
</tbody>
</table>
Summary

Swivel secures Wasm from Spectre attacks

Swivel-SFI: backward compatible approach
Swivel-CET: leverages hardware extensions, supports hyperthreading

Key abstraction: linear blocks

https://swivel.programming.systems
@ShrNarayan