WasmBounds: Eliminating Memory Bounds Checks in WebAssembly

Emma Sudo and Keith Winstein
Stanford University

In Wasm 1.0:

- Reserve 8 GiB of virtual address space and efficiently protect against out of bounds accesses using guard pages

With 64-bit Memory, Single-Byte Page Size, Embedded Systems:

- Guard pages method is impractical, so usually software bounds checks are used at a cost

```
if base + offset > length of memory:
  trap()
```

Observation:

- **Safe programming languages are already bounds checking**, but this information is less obvious at the Wasm binary level
- Use abstract interpretation to recognize safe accesses and eliminate the associated software bounds check

Dot Product in Rust with Built-In Array Access Checks:

```rust
const N: i32 = 1024;

static first: [i32; N] = [0; N];
static second: [i32; N] = [0; N];

fn dotproduct() -> i32 {
  let mut ret = 0;
  for i in 0..N {
    ret += first[i] * second[i];
  }
  ret
}
```

Dot Product in C with Safe Access Helpers:

```c
const int N = 1024;

int first[N];
int second[N];

int get_first(int n) {
    return first[n % N];
}

int get_second(int n) {
    return second[n % N];
}

int dotproduct(void) {
  int ret = 0;
  for (unsigned int i = 0; i < N; ++i) {
    ret += get_first(i) * get_second(i);
  }
  return ret;
}
```

Dot Product in C Compiled to Wasm

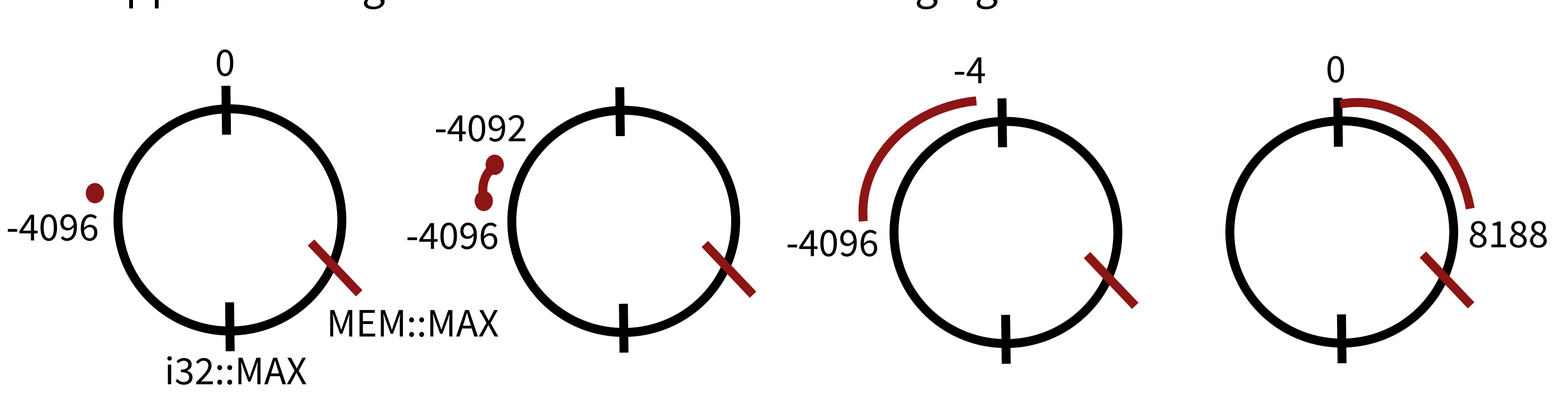
```
1        (memory 1)
2        (func $dotproduct (result i32)
3          (local $res i32)
4          (local $offset i32)
5          i32.const 0
6        local.set $res
7          i32.const -4096
8        local.set $offset
9            loop $loop_start
10             local.get $offset
11             i32.const 8192
12           i32.add
13           i32.load
14             local.get $offset
15             i32.const 4096
16           i32.add
17           i32.load
18         i32.mul
19           local.get $res
20         i32.add
21       local.set 0
22           local.get $offset
23           i32.const 4
24         i32.add
25         local.tee $offset
26       br_if $loop_start
27         end
28       local.get $res
29       )
```

Why are the loads on lines 13 and 17 safe?

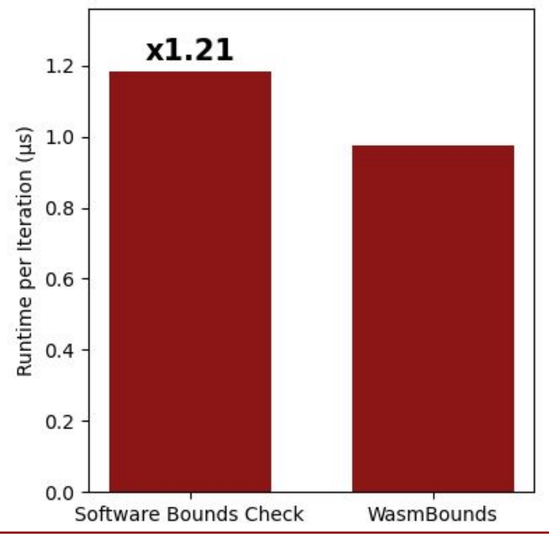
$offset starts as -4096. Each loop iteration increments $offset by 4, and the loop continues until $offset = -4. Thus, $offset ranges from [-4096, -4] at the loads and the address ranges from [0, 8188]. Since the memory is 64 KiB, both loads will always be in bounds.

WasmBounds uses Abstract Interpretation to Derive the Safety of the Two Loads:

WasmBounds operates over a strided, sign-agnostic interval domain. Wasm instructions are translated into abstract operations over these intervals, soundly over-approximating the addresses while converging in few iterations.



Dot Product Microbenchmark:



Conclusion:

- Abstract interpretation is a viable method for recovering safe accesses from a Wasm binary
- Eliminating redundant bounds checks can lead to substantial speedups on embedded platforms or with Wasm features that make virtual guard pages impractical