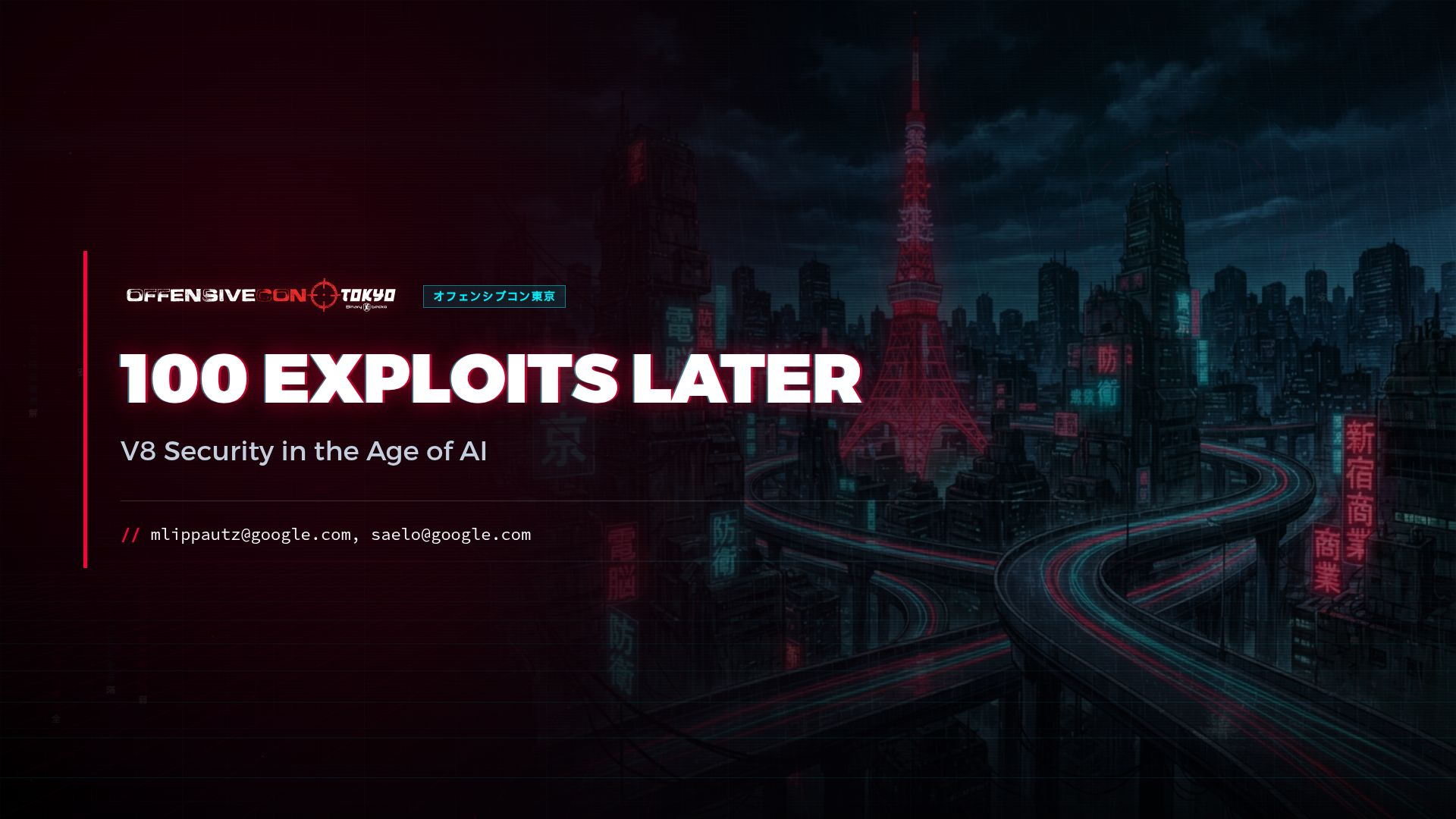
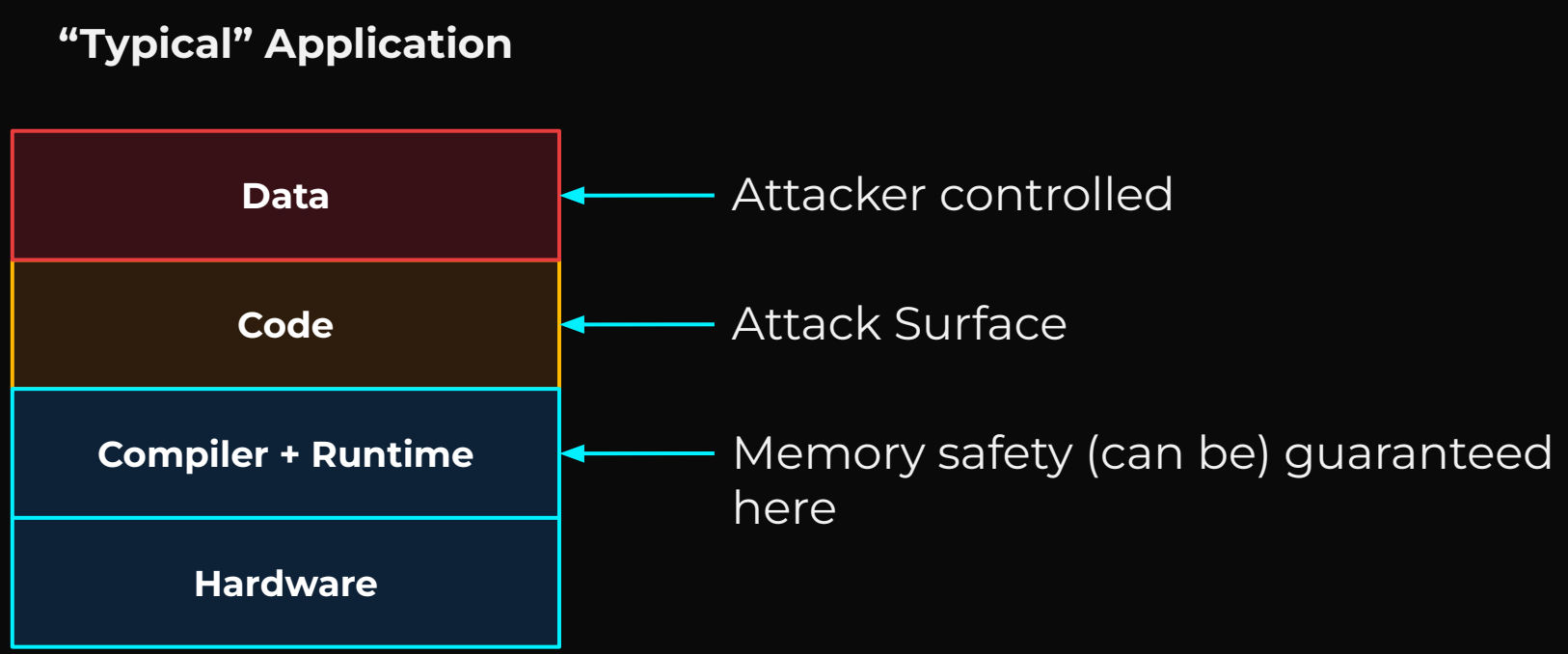
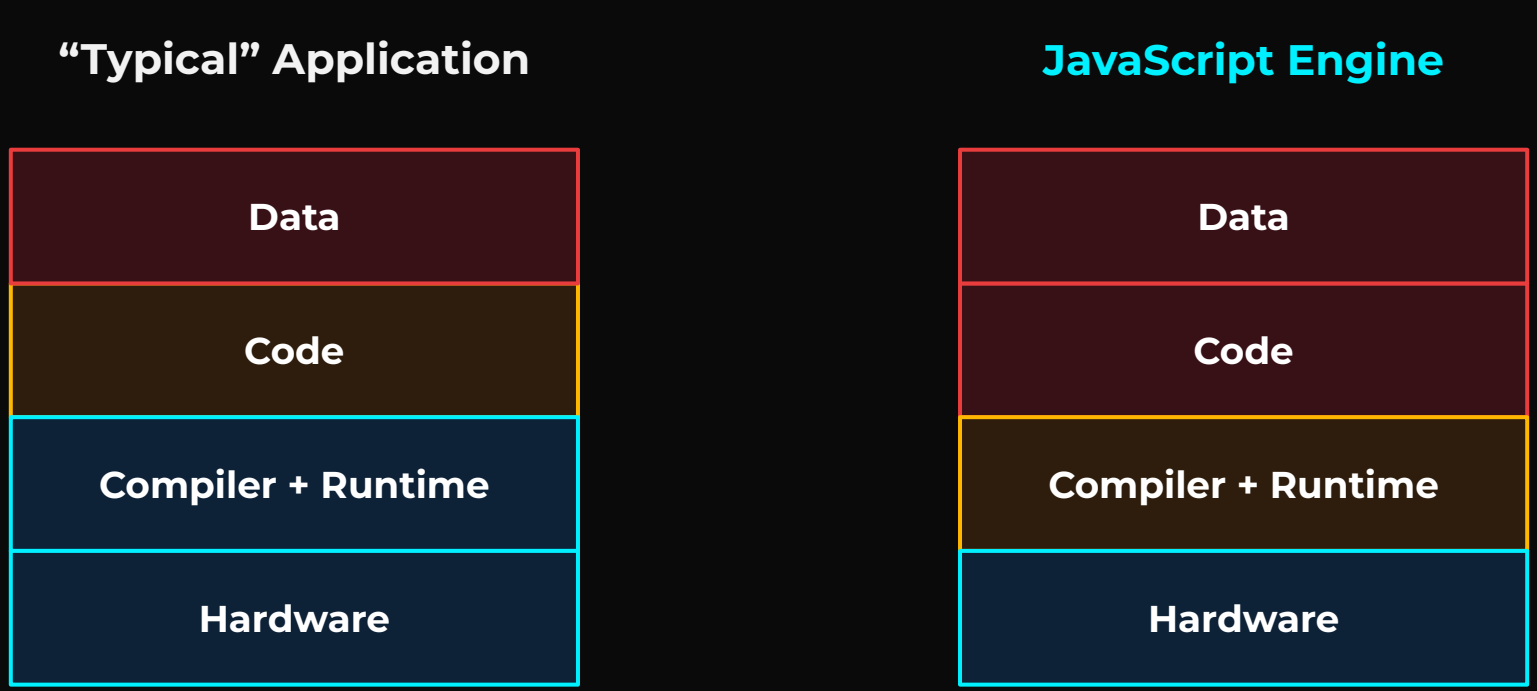

OFFENSIVECON TOKYO
オフェンシブコン東京
100 EXPLOITS LATER
V8 Security in the Age of AI
// mlippautz@google.com, saelo@google.com

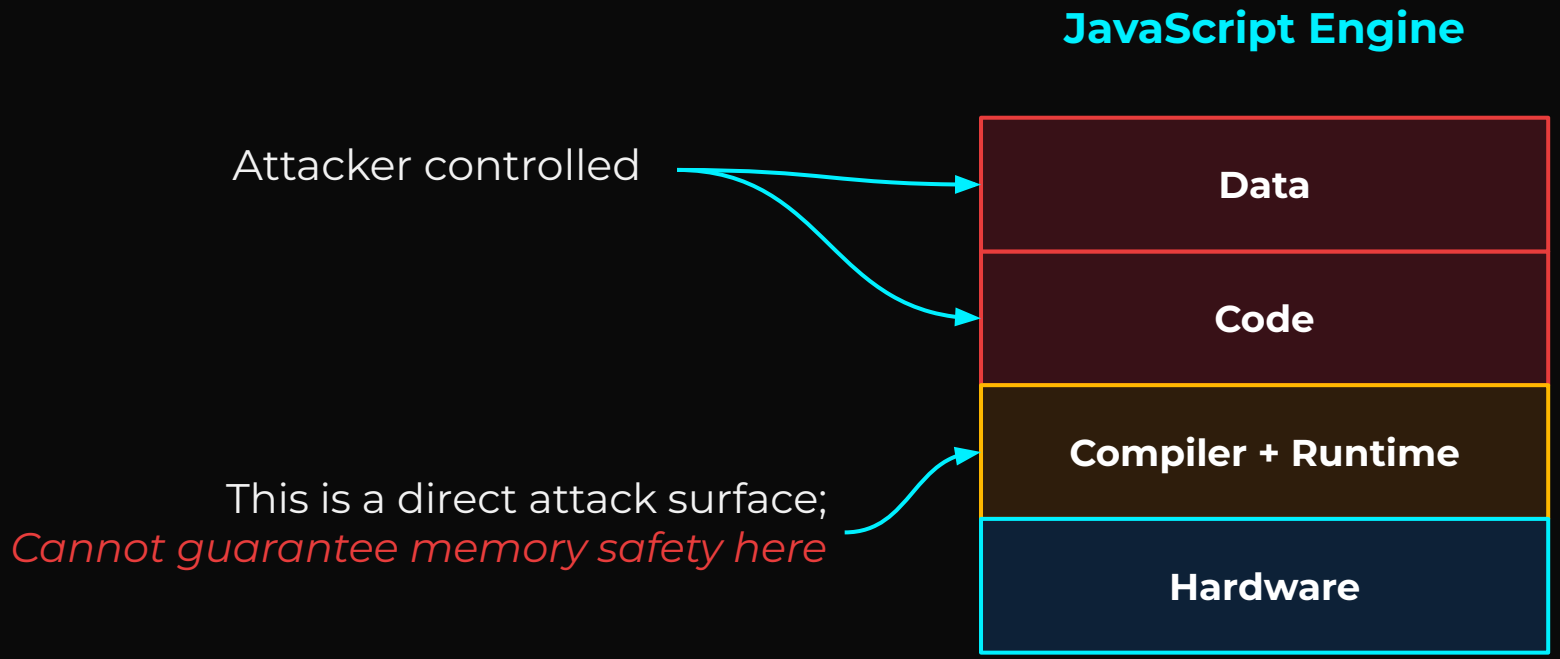
Why JavaScript Engine Security is hard



Why JavaScript Engine Security is hard



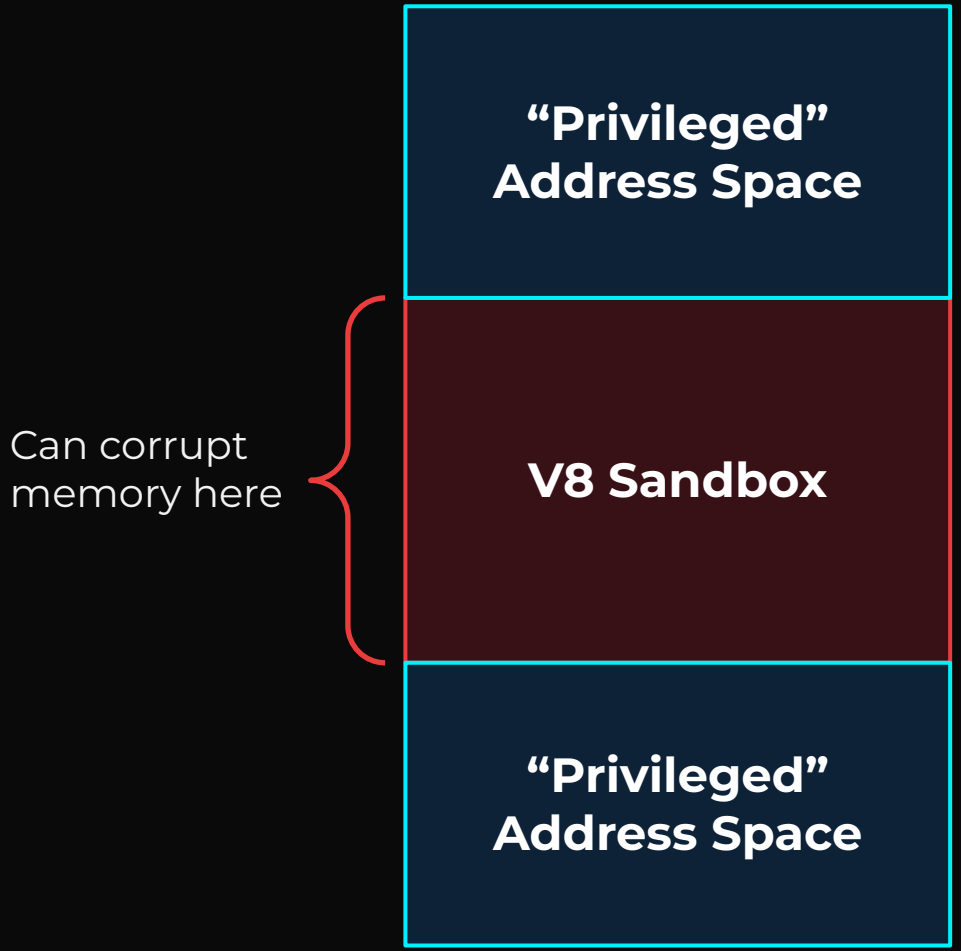
Why JavaScript Engine Security is hard

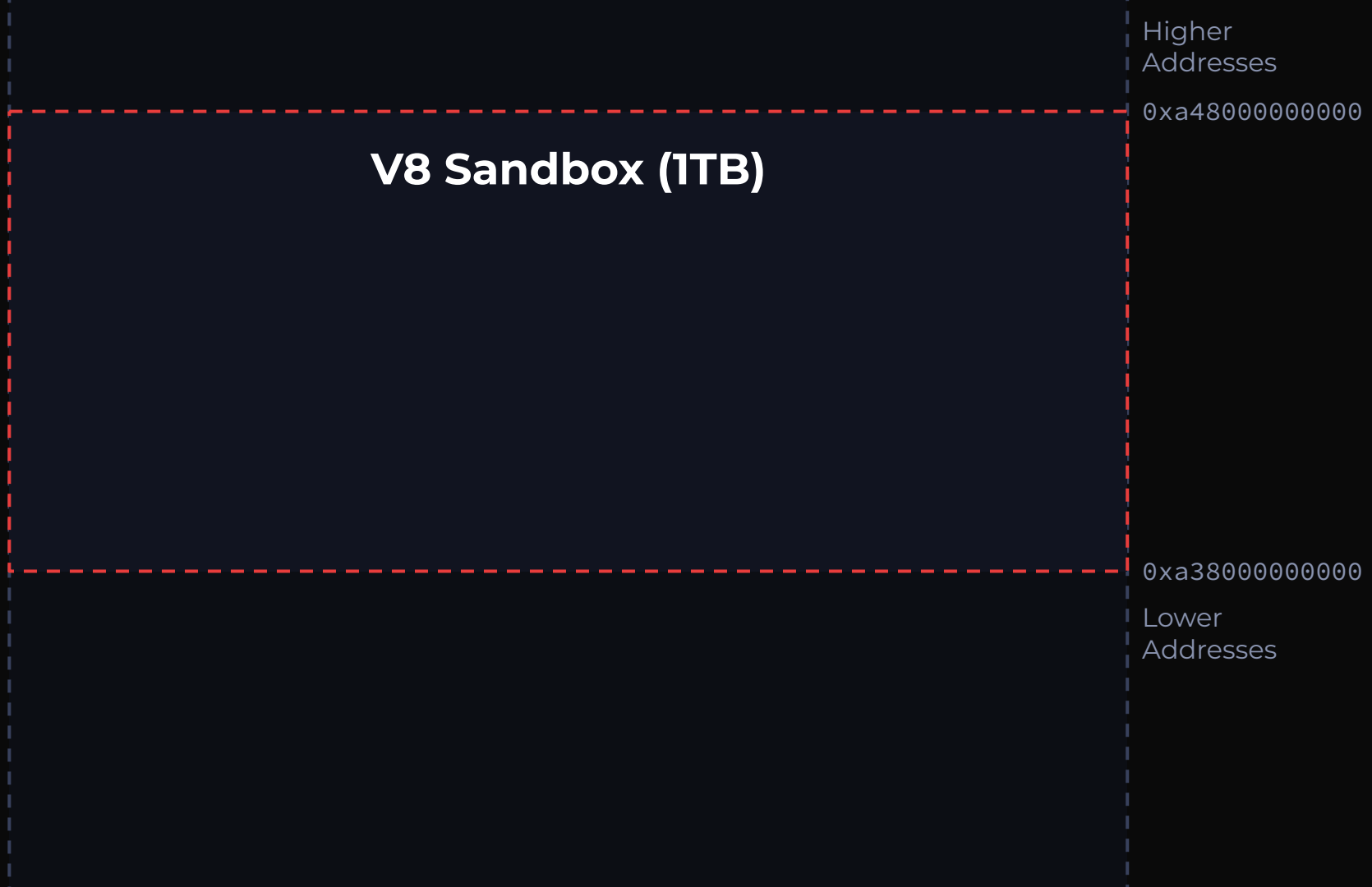


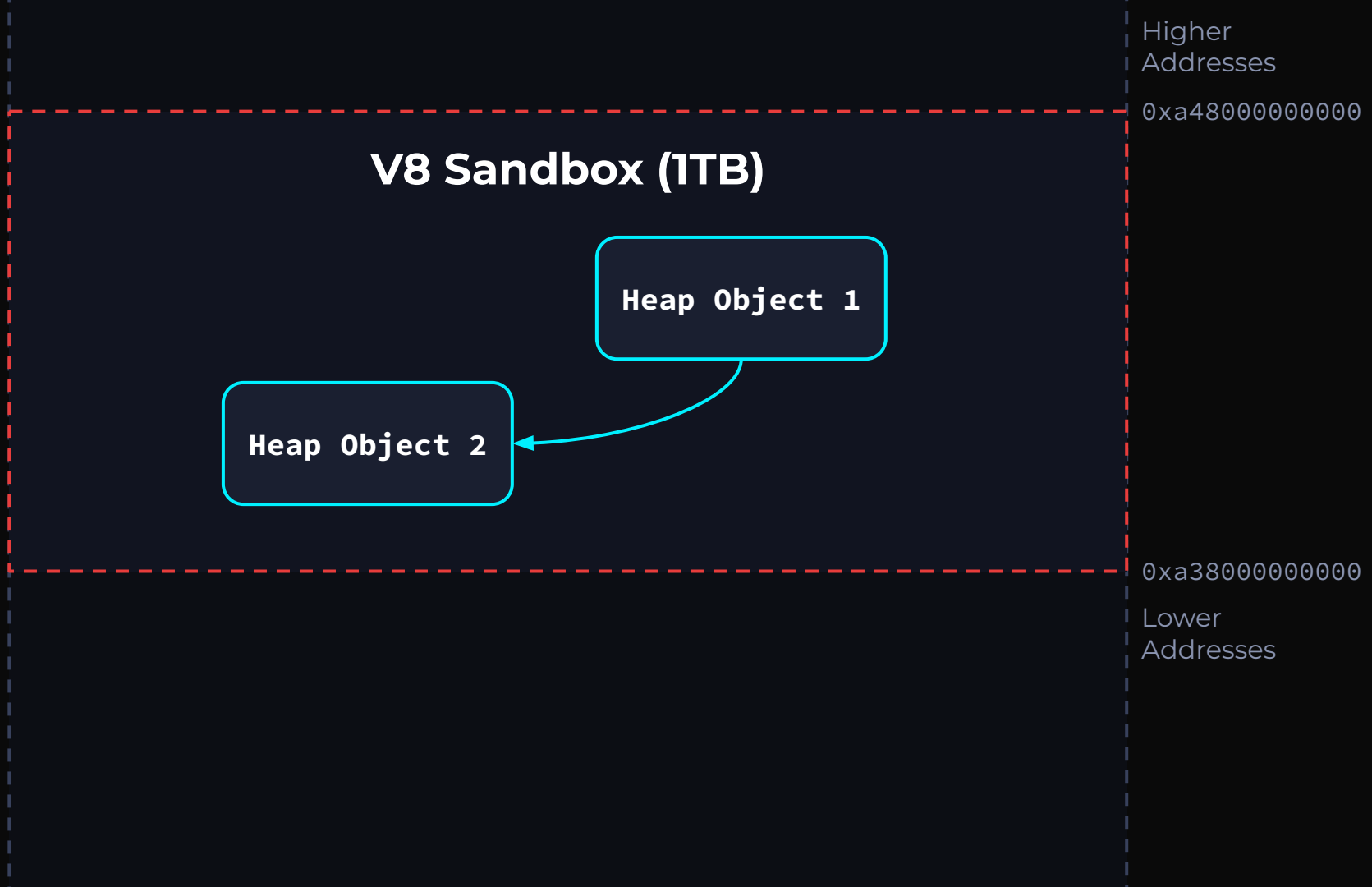
Sandbox basics

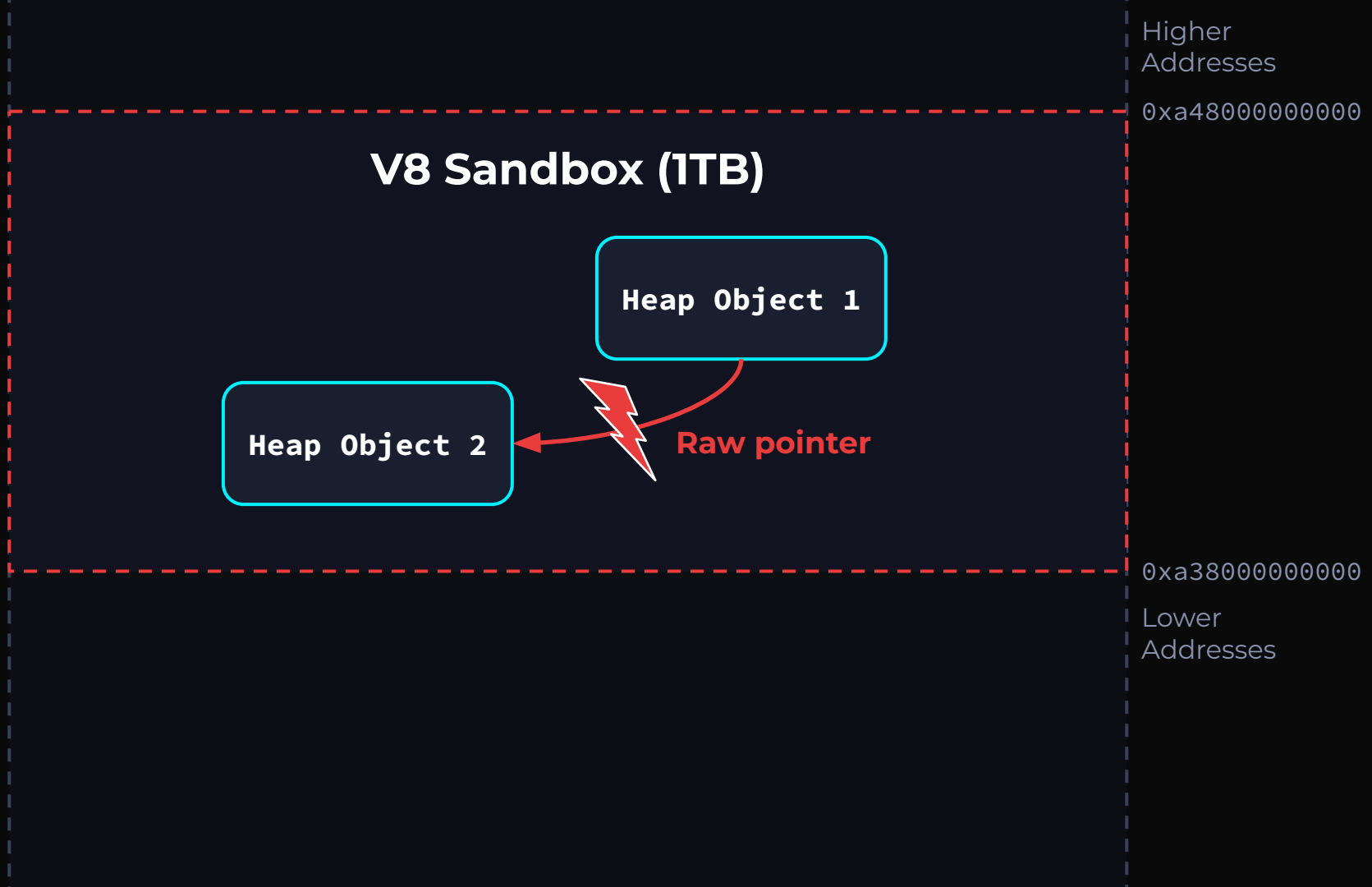
- Accept that bugs will happen and that memory will be corrupted
- Limit which memory can be corrupted

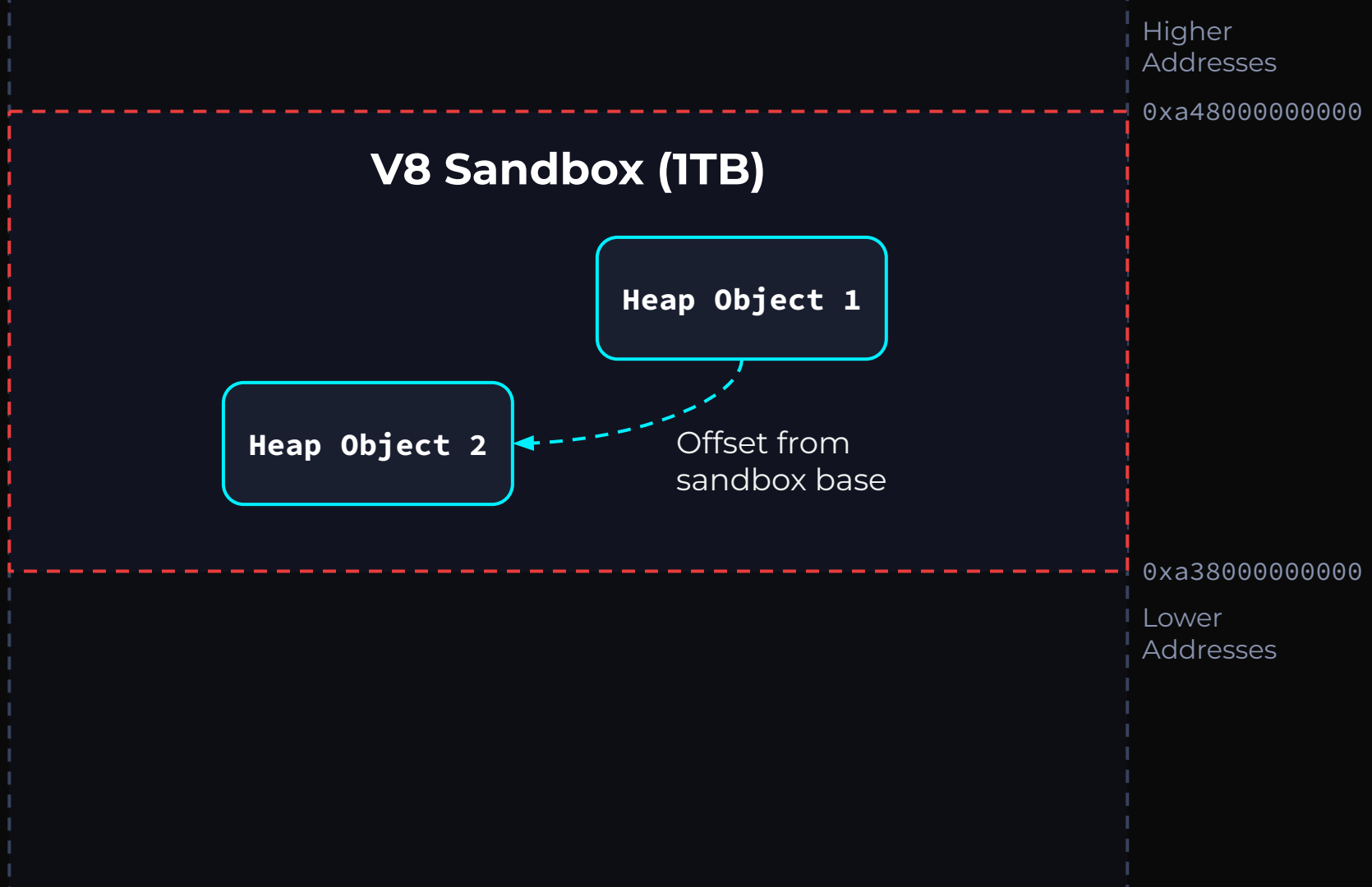
Result: An in-process sandbox

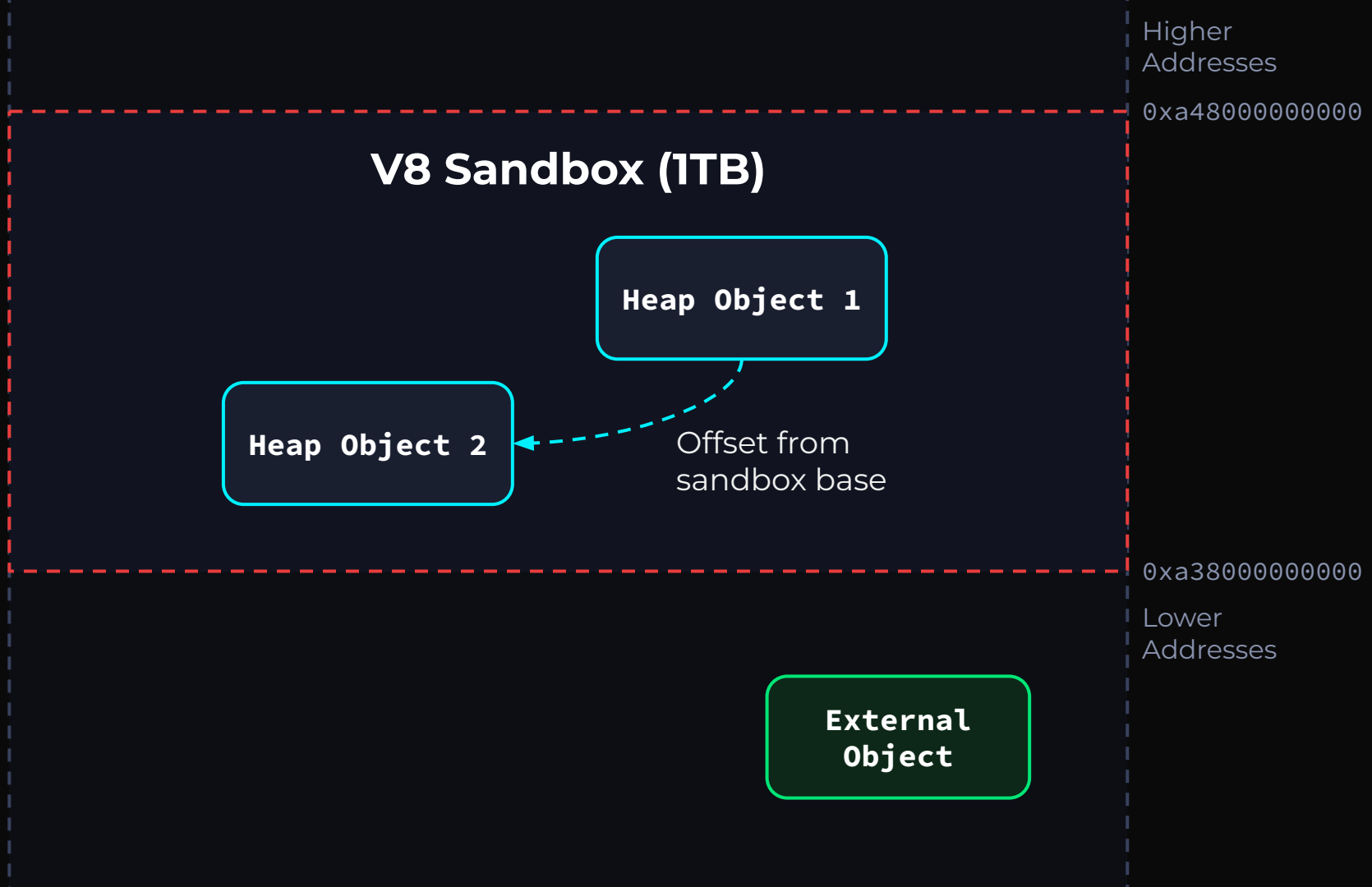


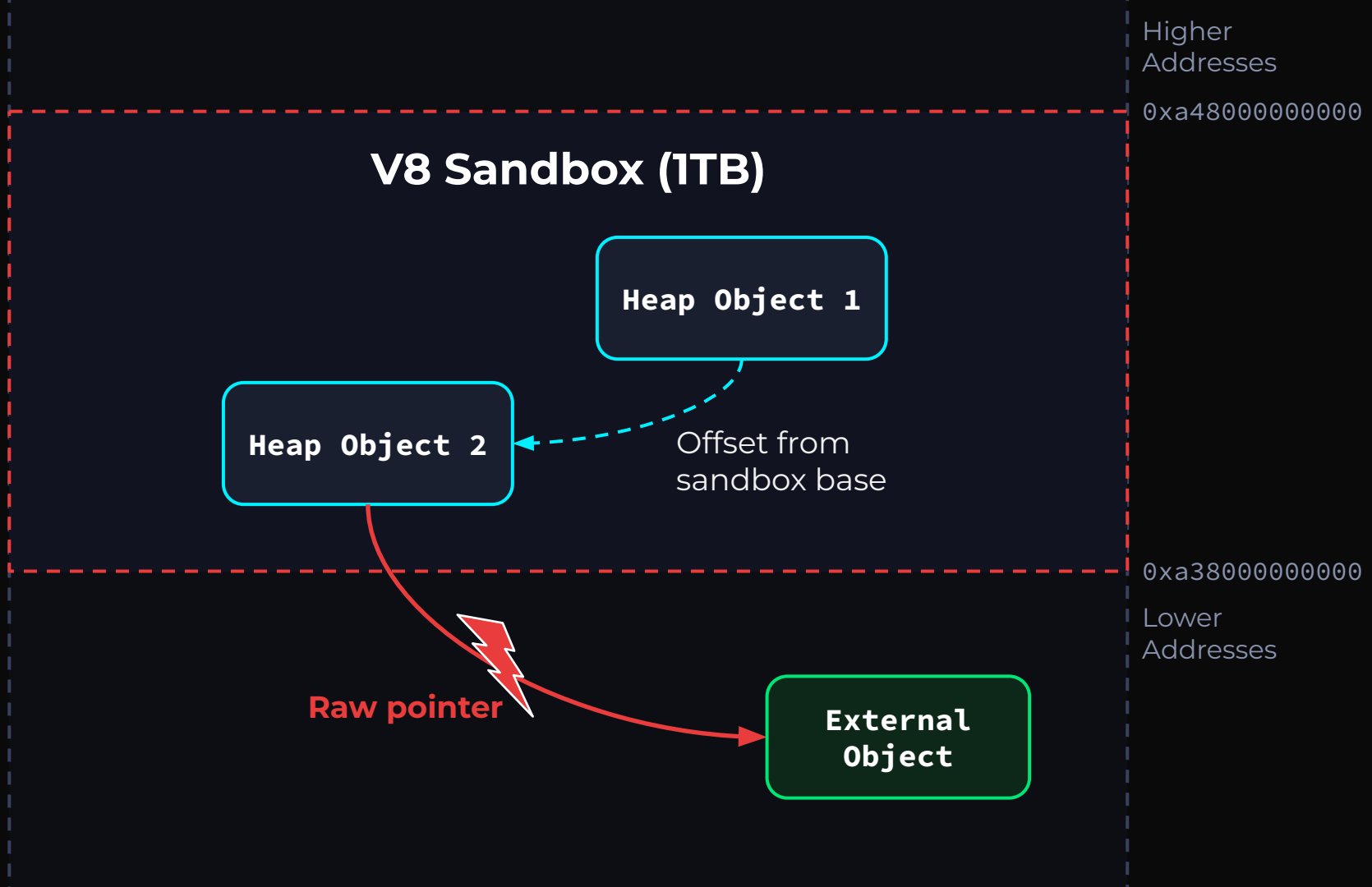

Higher
Addresses
0xa48000000000
V8 Sandbox (1TB)
0xa38000000000
Lower
Addresses

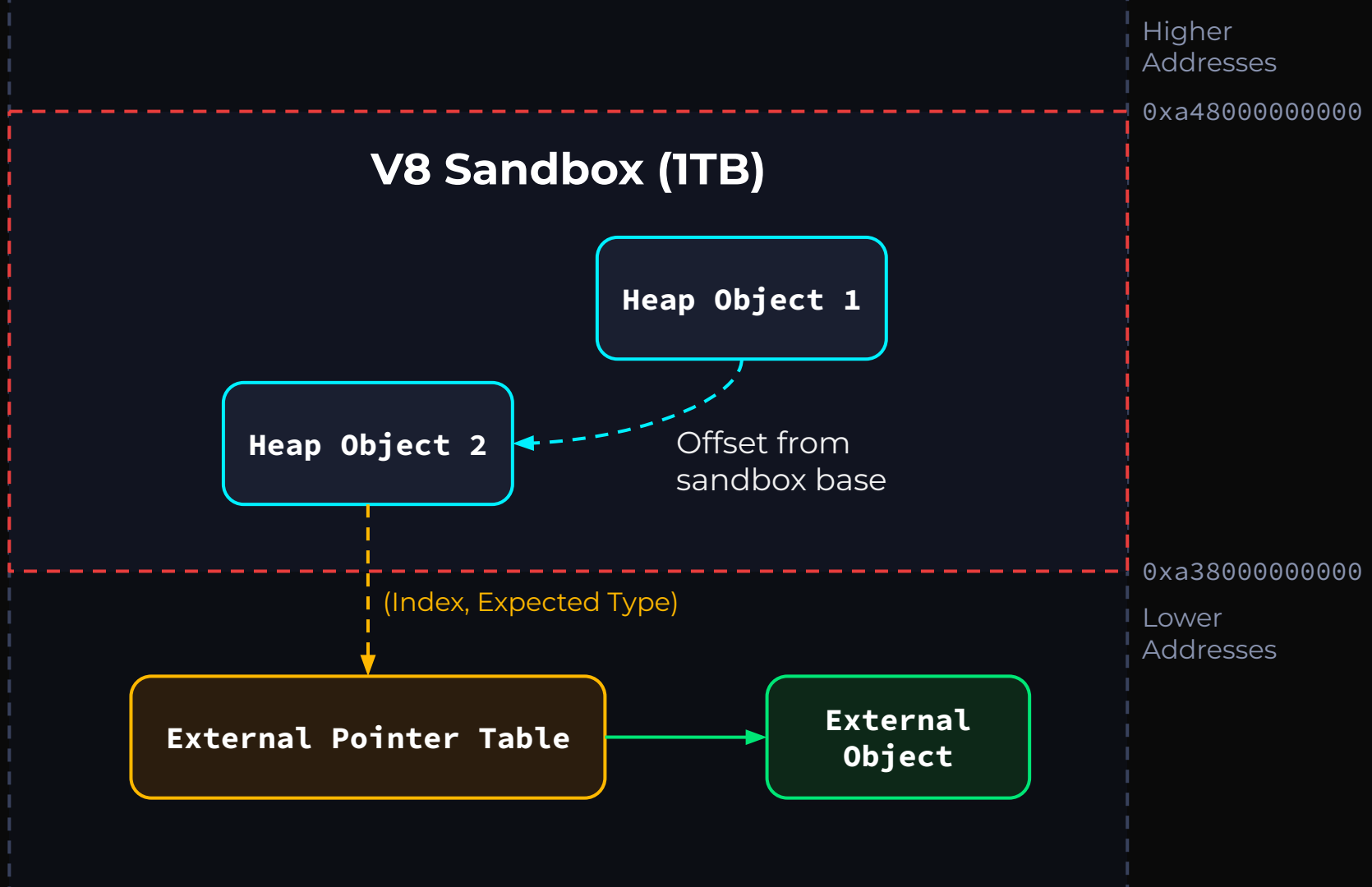

Higher
Addresses
0xa48000000000
V8 Sandbox (1TB)
Heap Object 1
Heap Object 2
0xa38000000000
Lower
Addresses


Higher Addresses
0xa48000000000
V8 Sandbox (1TB)
Heap Object 1
Heap Object 2
Raw pointer
0xa38000000000
Lower Addresses


Higher Addresses
0xa48000000000
V8 Sandbox (1TB)
Heap Object 1
Heap Object 2
Offset from sandbox base
0xa38000000000
Lower Addresses


Higher Addresses
0xa48000000000
V8 Sandbox (1TB)
Heap Object 1
Heap Object 2
Offset from sandbox base
0xa38000000000
Lower Addresses
External Object


Higher Addresses
0xa48000000000
V8 Sandbox (1TB)
Heap Object 1
Heap Object 2
Offset from sandbox base
0xa38000000000
Lower Addresses
Raw pointer
External Object


Higher Addresses
0xa48000000000
V8 Sandbox (1TB)
Heap Object 1
Heap Object 2
Offset from sandbox base
0xa38000000000
(Index, Expected Type)
Lower Addresses
External Pointer Table
External Object



Basically: ban all raw pointers!

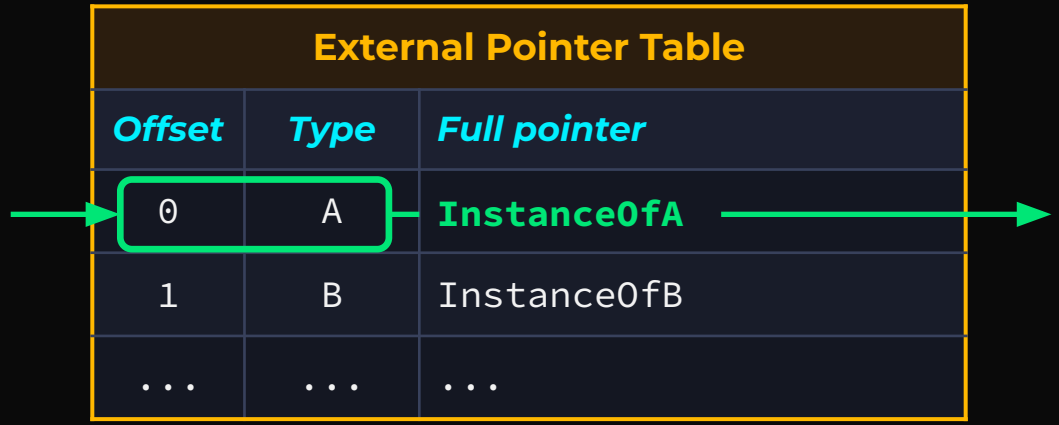
External Pointer Table

- Tables that map an access of *(offset, expected type)* to a full pointer
- Access with wrong expect type leads to hard crash

External Pointer Table		
Offset	***Type***	***Full pointer***
`0`	`A`	`InstanceOfA`
`1`	`B`	`InstanceOfB`
`...`	`...`	`...`

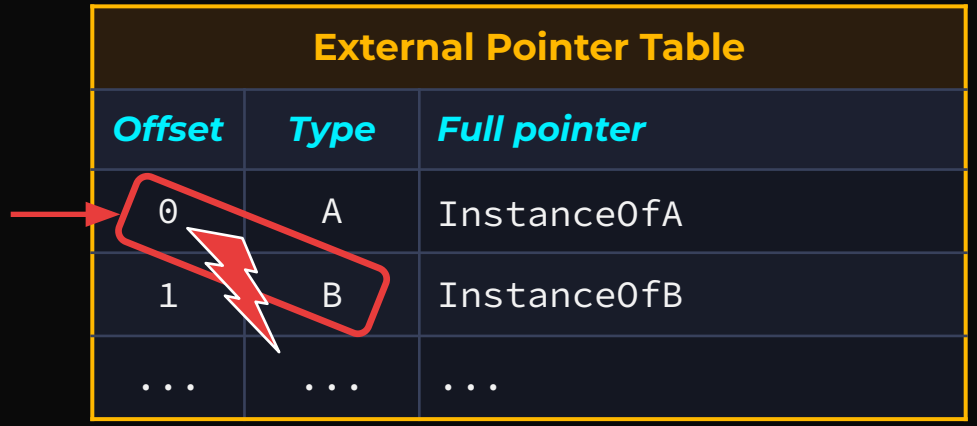
External Pointer Table

- Tables that map an access of *(offset, expected type)* to a full pointer
- Access with wrong *expect type* leads to hard crash



External Pointer Table

- Tables that map an access of *(offset, expected type)* to a full pointer
- Access with wrong *expect type* leads to hard crash



External Pointer Table: Code

- Tables that map an access of ***(offset, expected type)*** to a full pointer
- Access with wrong ***expect type*** leads to **hard crash**

```
template <ExternalPointerTagRange tag_range>
Address ReadExternalPointerField(v8::Isolate* isolate, Address heap_object_ptr, int offset) {
  Address* table = GetExternalPointerTable(isolate);
  ExternalPointerHandle handle = ReadHandle(heap_object_ptr, offset);
  ExternalPointerTag actual_tag = ExtractTagForEntry(table, handle);
  if (tag_range.Contains(actual_tag)) [[likely]] {
    entry &= kExternalPointerPayloadMask;
  } else {
    entry = 0;
  }
  return entry;
}
```

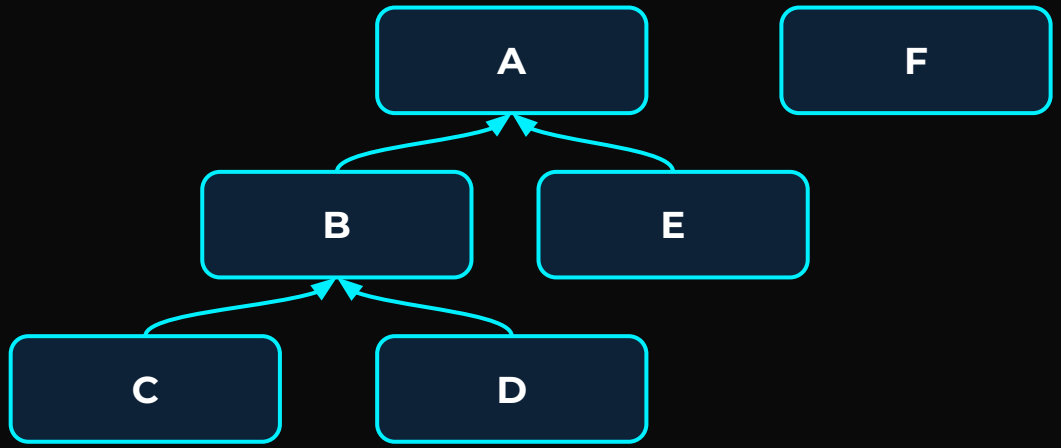
External Pointer Table: Code

- Tables that map an access of ***(offset, expected type)*** to a full pointer
- Access with wrong ***expect type*** leads to **hard crash**

```
template <ExternalPointerTagRange tag_range>
Address ReadExternalPointerField(v8::Isolate* isolate, Address heap_object_ptr, int offset) {
  Address* table = GetExternalPointerTable(isolate);
  ExternalPointerHandle handle = ReadHandle(heap_object_ptr, offset);
  ExternalPointerTag actual_tag = ExtractTagForEntry(table, handle);
  if (tag_range.Contains(actual_tag)) [[likely]] {
    entry &= kExternalPointerPayloadMask;
  } else {
    entry = 0;
  }
  return entry;
}
```

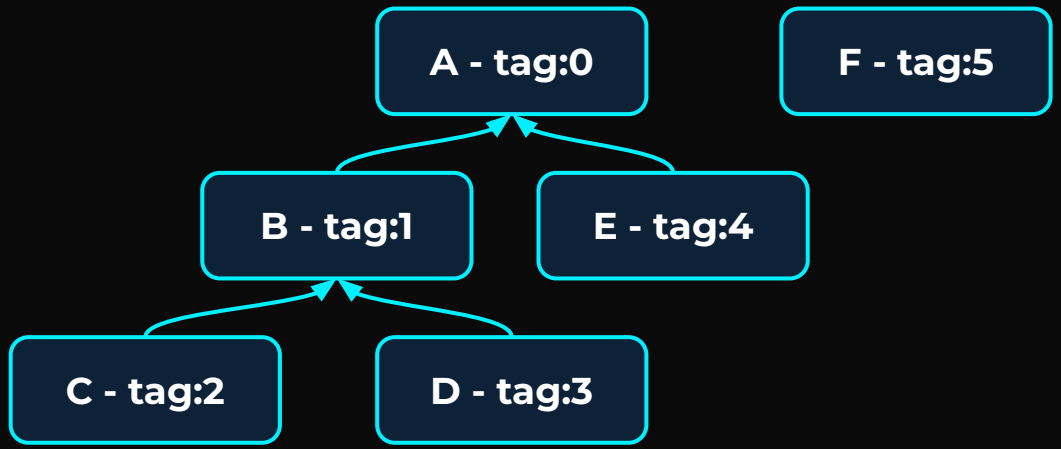
External Pointer Table: Managing type tags

- Fine-grained tags provide **type safety**
- Subtyping based on **range checks**



External Pointer Table: Managing type tags

- Fine-grained tags provide **type safety**
- Subtyping based on **range checks**



Type	Tag	Type tag range
A	0	[0, 4]
B	1	[1, 3]
C	2	[2, 2]
D	3	[3, 3]
E	4	[4, 4]
F	5	[5, 5]

V8 Sandbox: A reality check

External Pointer Tables are security critical... and surprisingly tricky to get right and maintain as a boundary in the long run!

Some examples of breakage in practice:

1. **Managing type tags**
2. **Maintaining proper tag checks**
3. **Managing lifetime of native objects**

Managing type tags: Covering the 99%

- Auto-generated type tag ranges compiled from **WebIDL spec files**
- Covers more than **1500 types**

```
class CORE_EXPORT V8Blob final : public bindings::V8InterfaceBridge<V8Blob, Blob> {
  static constexpr v8::CppHeapPointerTag kThisTag = static_cast<v8::CppHeapPointerTag>(860);
  static constexpr v8::CppHeapPointerTag kMaxSubclassTag = static_cast<v8::CppHeapPointerTag>(861);
  static constexpr v8::CppHeapPointerTagRange kTagRange = v8::CppHeapPointerTagRange(kThisTag, kMaxSubclassTag);
} // https://w3c.github.io/FileAPI/#blob-section

class CORE_EXPORT V8File final : public bindings::V8InterfaceBridge<V8File, File> {
 public:
  static constexpr v8::CppHeapPointerTag kThisTag = static_cast<v8::CppHeapPointerTag>(861);
  static constexpr v8::CppHeapPointerTag kMaxSubclassTag = static_cast<v8::CppHeapPointerTag>(861);
  static constexpr v8::CppHeapPointerTagRange kTagRange = v8::CppHeapPointerTagRange(kThisTag, kMaxSubclassTag);
} // https://w3c.github.io/FileAPI/#file-section
```

Managing type tags: Bugs are in the tail end

- Handful of cases that are **manually** wired up
  - Every modification requires careful review to not create compatible ranges
  - **Recall:** Ranges map to type hierarchies
  - **Bugs:** Overlapping ranges, misplaced tags, shared tags for different types (historical)

```
enum WrappablePointerTag : uint16_t {
  kFirstPointerTag = static_cast<uint16_t>(kGinWrappableTagRange.first),
  kAccessibilityControllerBindings =
      kFirstPointerTag,          // content::AccessibilityControllerBindings
  kAPIBindingBridge,             // extensions::APIBindingBridge
  kAPIBindingJSUtil,             // extensions::APIBindingJSUtil
  kAutomationPosition,           // ui::AutomationPosition
  kCallbackHolderBase,           // gin::internal::CallbackHolderBase
  kChromeSetting,                // extensions::ChromeSetting
  kContentSetting,               // extensions::ContentSetting
  kDomAutomationController,      // content::DomAutomationController
  kEventEmitter,                 // extensions::EventEmitter
  kEventSenderBindings,          // content::EventSenderBindings
  kGamepadControllerBindings,    // content::GameControllerBindings
  kGCController,                 // content::GCController
  // ...
}
```

Maintaining proper tag checks

Recall

```
template <ExternalPointerTagRange tag_range>
Address ReadExternalPointerField(v8::Isolate* isolate, Address heap_object_ptr, int offset) {
  Address* table = GetExternalPointerTable(isolate);
  ExternalPointerHandle handle = ReadHandle(heap_object_ptr, offset);
  ExternalPointerTag actual_tag = ExtractTagForEntry(table, handle);
  if (tag_range.Contains(actual_tag)) [[likely]] {
    entry &= kExternalPointerPayloadMask;
  } else {
    entry = 0;
  }
  return entry;
}
```

Maintaining proper tag checks

Recall

```
template <ExternalPointerTagRange tag_range>
Address ReadExternalPointerField(v8::Isolate* isolate, Address heap_object_ptr, int offset) {
  Address* table = GetExternalPointerTable(isolate);
  ExternalPointerHandle handle = ReadHandle(heap_object_ptr, offset);
  ExternalPointerTag actual_tag = ExtractTagForEntry(table, handle);
  if (tag_range.Contains(actual_tag)) [[likely]] {
    entry &= kExternalPointerPayloadMask;
  } else {
    entry = 0;
  }
  return entry;
}
```

Wrong type merely returns **nullptr**. Inlined in 1000s of callsites, which must remain small in terms of binary size. (**nullptr** deref is a **safe crash** for Chromium)

Maintaining proper tag checks

- **Theory:**
  - Crash on resolving a pointer with a **wrong tag**
  - Various tagging schemes used over the years
  - **Simple range checks** proved to be the most versatile

Maintaining proper tag checks: Issue 500771385

“This flag makes null dereference a **defined trap (noreturn)** rather than undefined behavior. Ironically, the compiler now exploits the defined trap semantics: in the majority of generated binding callbacks, the unwrapped pointer is immediately dereferenced for a single member function call, with no other observable side effects on the null path. Since the null dereference is a guaranteed noreturn trap, the compiler concludes that the `entry = 0` branch always leads to a trap — **making the entire tag check dead code. It removes it.**”

Maintaining proper tag checks: Issue 500771385

“

Since the null dereference is a guaranteed noreturn trap, the compiler concludes that the `entry = 0` branch always leads to a trap — **making the entire tag check dead code. It removes it.**”

Wat!?

Maintaining proper tag checks: Issue 500771385

“

Since the null dereference is a guaranteed noreturn trap, the compiler concludes that the `entry = 0` branch always leads to a trap — **making the entire tag check dead code. It removes it.**”

Wat!?

```
## V8 sandbox violation detected!
The sandbox violation was a *read* access which is technically not a sandbox violation. This requires
manual investigation.

"ctx_handle=0x80140 dm_handle=0x80180"
"Swapped. Calling beginPath+moveTo+stroke..."
Received signal 11 SEGV_MAPERR 414141420018
```

They do have a working escape though…

Maintaining proper tag checks: Issue 500771385

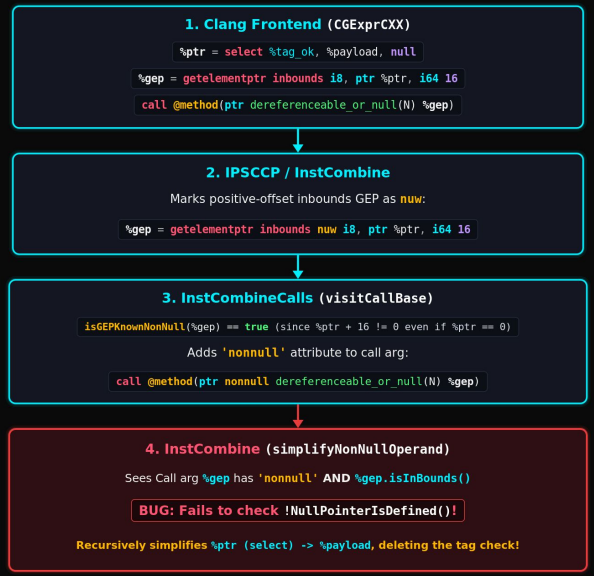
- **First instinct:** Reporter has a **wrong explanation**
- Tag check still optimized away
- Happened in early 2026:
  - AIs were already used to massively find bugs
  - Worked only so-and-so on fully diagnosing issues
- **Stop-gap:** Add `volatile` annotations and move on

Maintaining proper tag checks: Issue 500771385

- **Fast forward:** “This would be cool to present.” ⇒ **Need to understand this**
- AI is now really good at crossing various layers

Maintaining proper tag checks: Issue 500771385

- **Fast forward:** “This would be cool to present.” ⇒ **Need to understand this**
- AI is now really good at crossing various layers
- **Bug in llvm:** https://github.com/llvm/llvm-project/issues/227002



Maintaining proper tag checks: Issue 500771385

- With `-fno-delete-null-pointer-checks` llvm **“sometimes”** optimizes away the `0 assignment`
- **“Sometimes”:** When it needs inner object offsets.
- **Kudos to llvm:** Fixed within 2 days of reporting

Managing lifetime of native objects: Theory

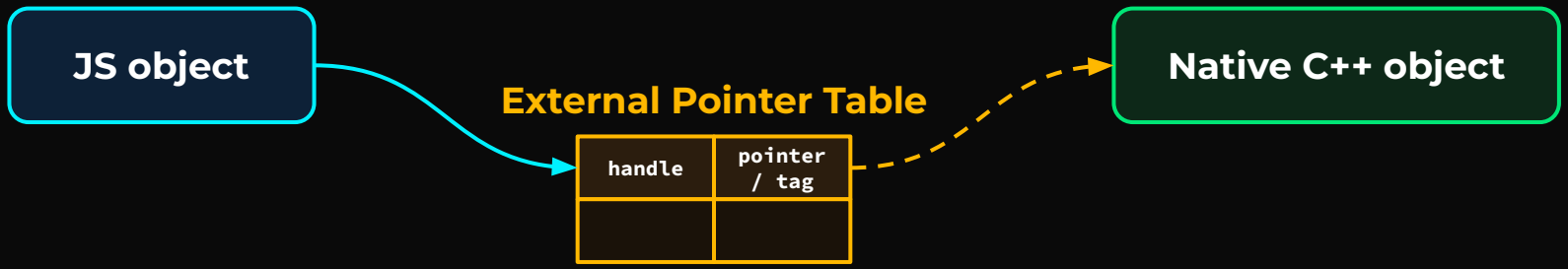
- Tables are interfaces like **file descriptors**
- Table entries have **markbits** and are known to **GC**
- Table entries just get **GCed** when the corresponding handle is **unreachable**

Managing lifetime of native objects: Practice

- Table entries often **own** the native objects they refer to
- **Custom lifetime management** all over the place

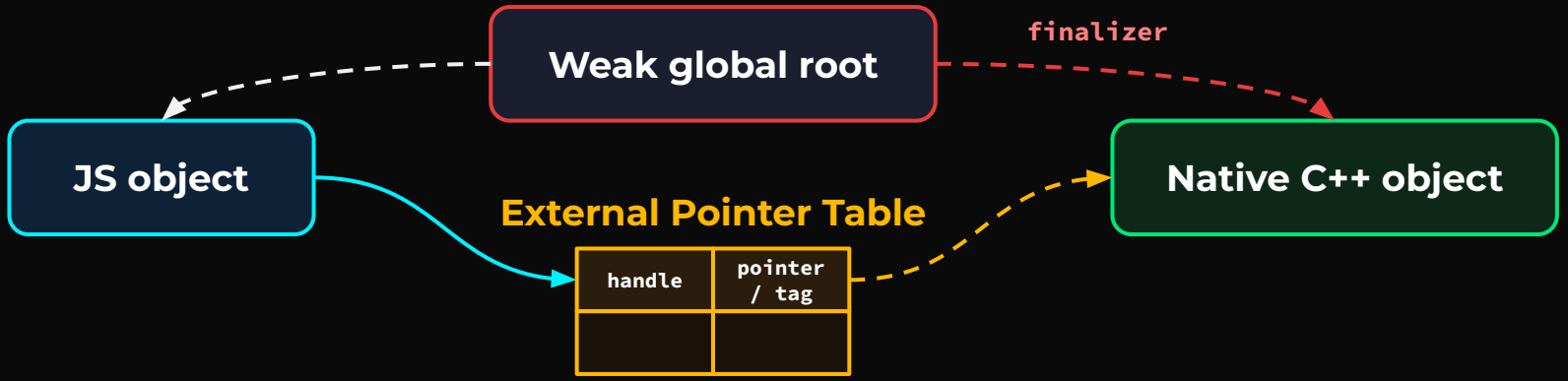
Managing lifetime of native objects: `Managed` type

- The type exists for a long time
- Attempt to have **native C++ objects** behind the **pointer tables**



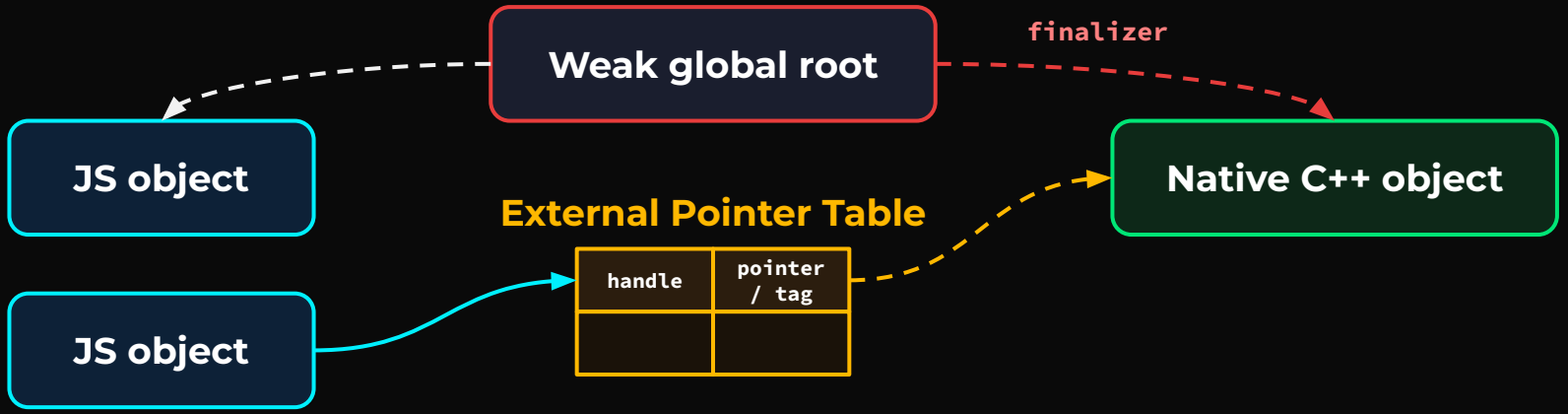
Lifetime issues: Managed

- The type exists for a long time
- Attempt to have **native C++ objects** behind the **pointer tables**



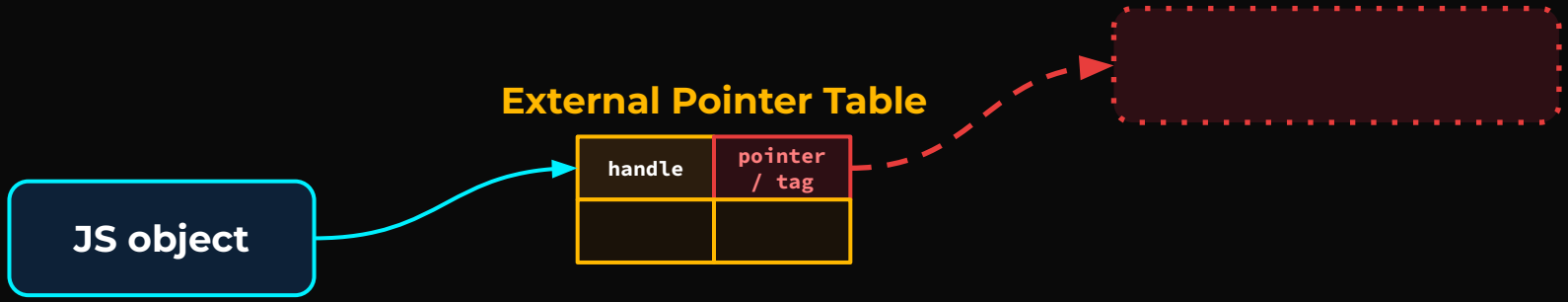
Lifetime issues: Managed

- The type exists for a long time
- Attempt to have **native C++ objects** behind the **pointer tables**



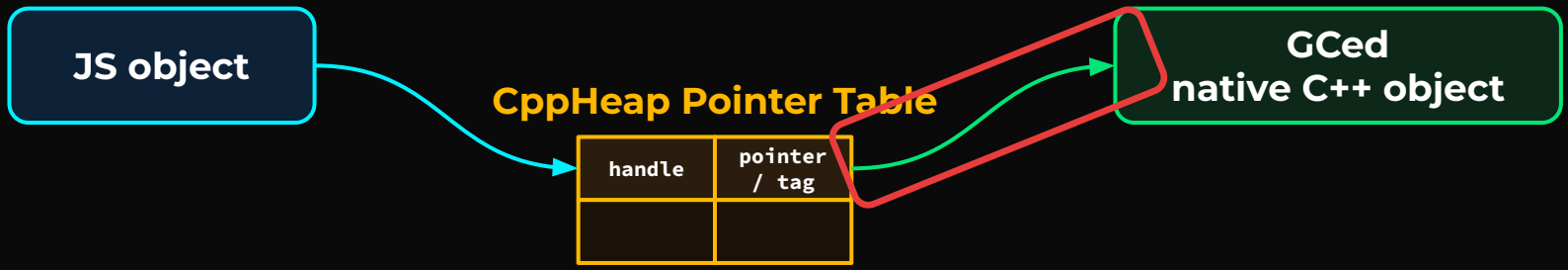
Lifetime issues: Managed

- The type exists for a long time
- Attempt to have **native C++ objects** behind the **pointer tables**



Managing lifetime of native objects: CppGCManaged

- **Solution:** Have a table that is properly garbage collected => **Oilpan**




Future of Sandbox

n-days are a problem

- As seen in **v8CTF**, n-days are now easy to exploit
- However, n-day attacker needs one bug per security boundary ***at the same time***

⇒ More security boundaries help here!

On top of that: Many sandbox issues are one-off easy fixes in old code that has not been upgrade to a sandbox world

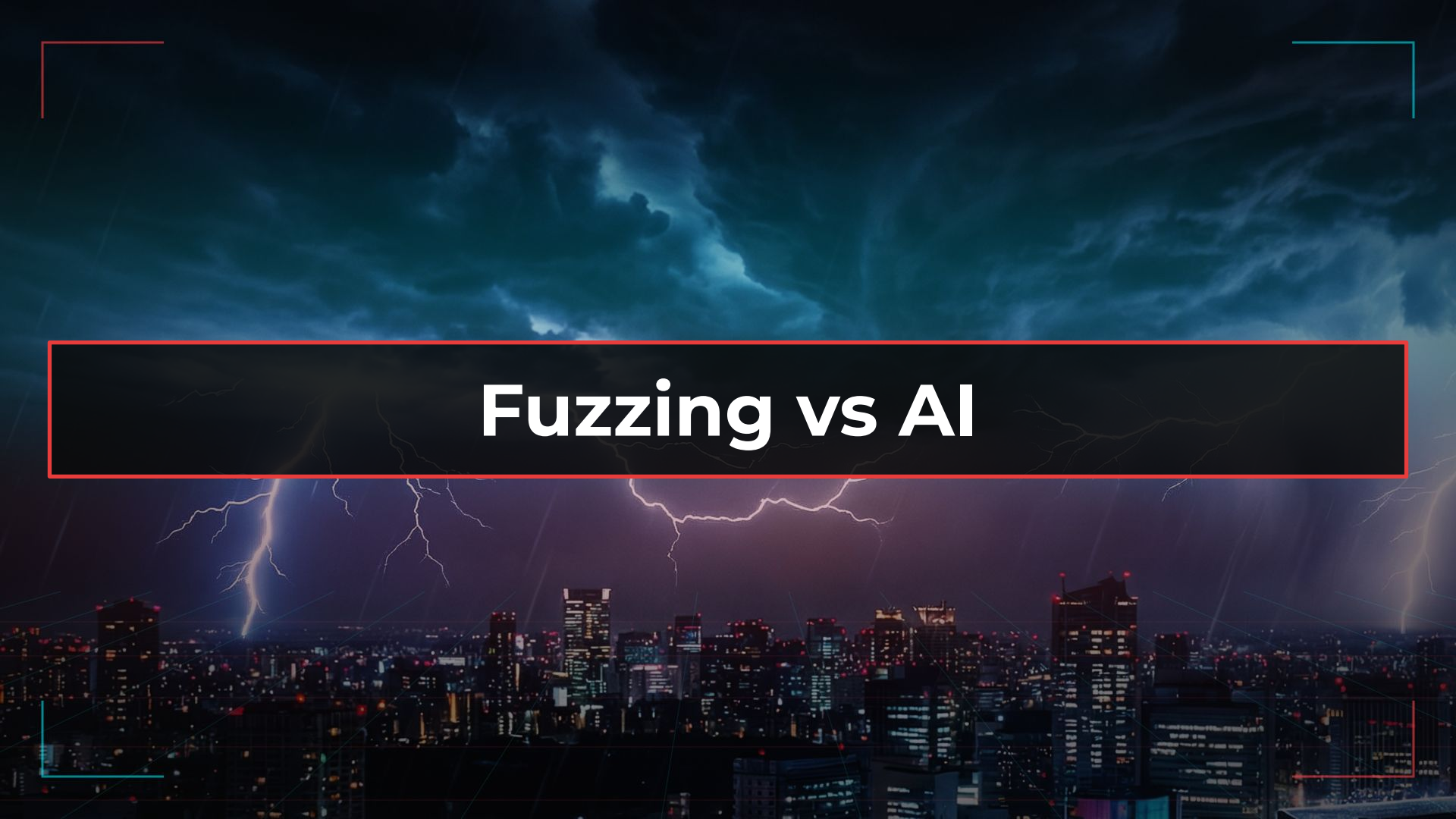
The next step: V8 Sandbox 1.0

So far, Sandbox was considered best effort

Changes:

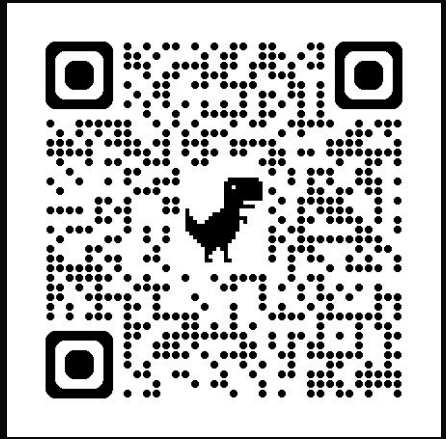
- We will be upgrading the classification (e.g. severity and impact) for sandbox bugs in the near future
- This leads to **stricter SLOs** and also automatically creates **backmerges**
  ⇒ Should make n-days harder

⇒ Sandbox now considered a security boundary!


Fuzzing vs AI

Fuzzing vs. AI

blog post



- Historically, V8 has relied heavily on fuzzing to find bugs automatically
- With the latest generation of LLMs, this is changing!
- ***Q: How do fuzzers compare to AI agents for bug finding?***

From Naptime to Big Sleep: Using Large Language Models To Catch Vulnerabilities In Real-World Code

2024-NOV-01 the Big Sleep team

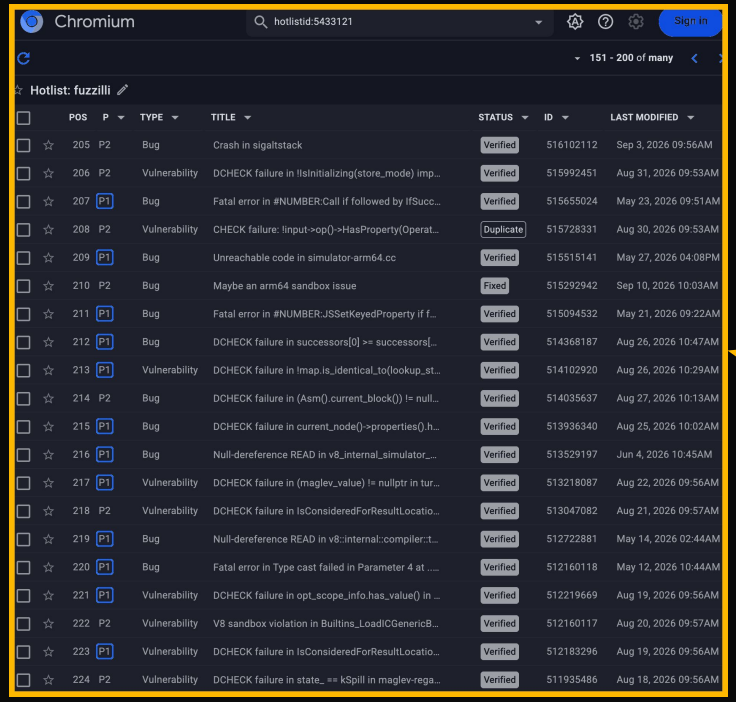
Vs.

README Contributing Apache-2.0 license

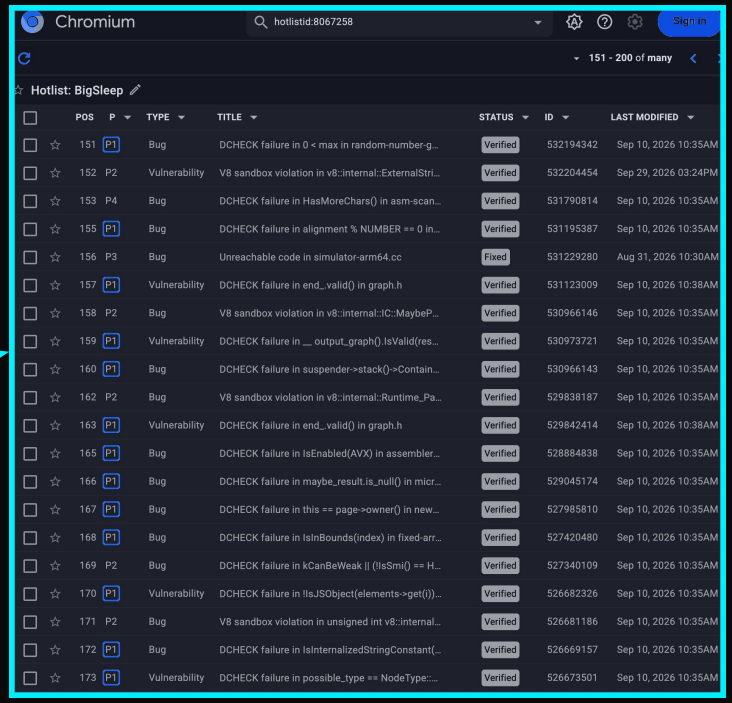
Fuzzilli

A (coverage-)guided fuzzer for dynamic language interpreters based on a custom intermediate language ("FuzzIL") which can be mutated and translated to JavaScript.

Real-world Statistics



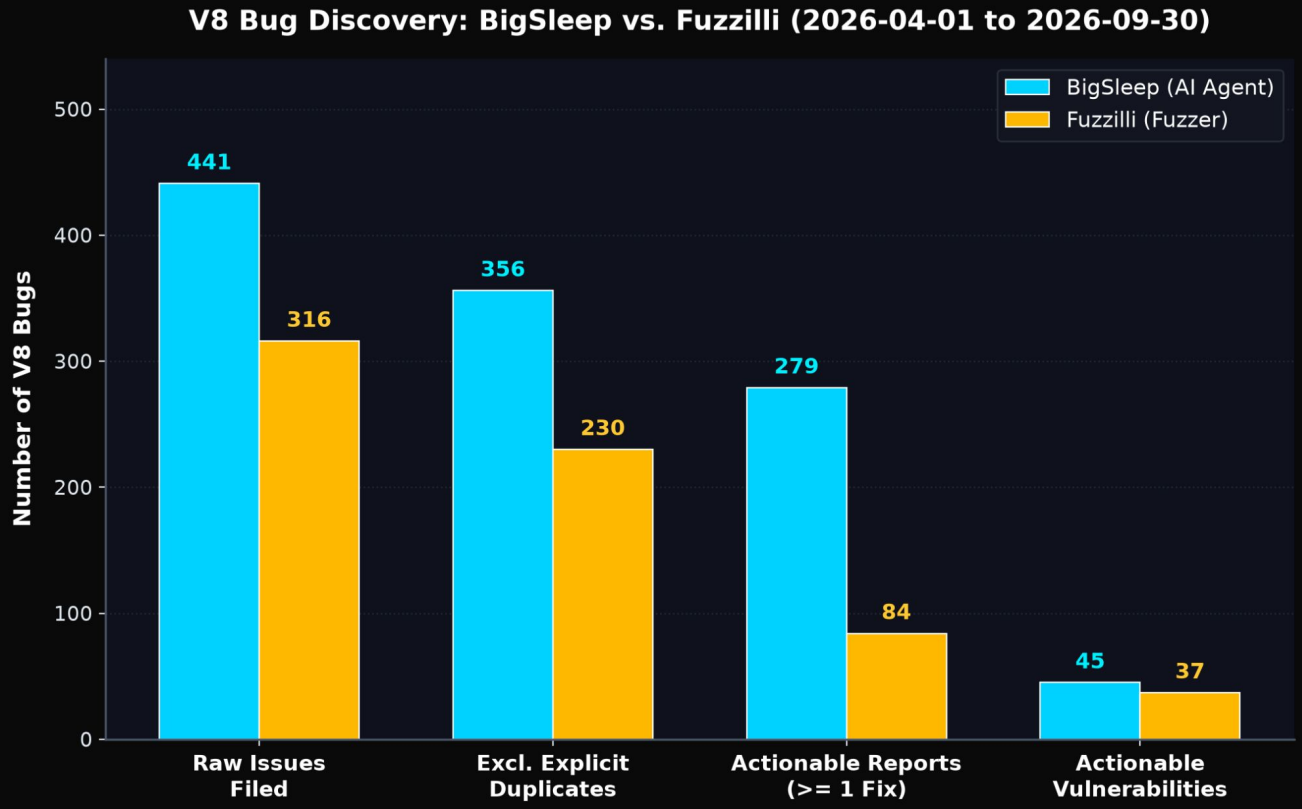
https://issues.chromium.org/hotlists/5433121



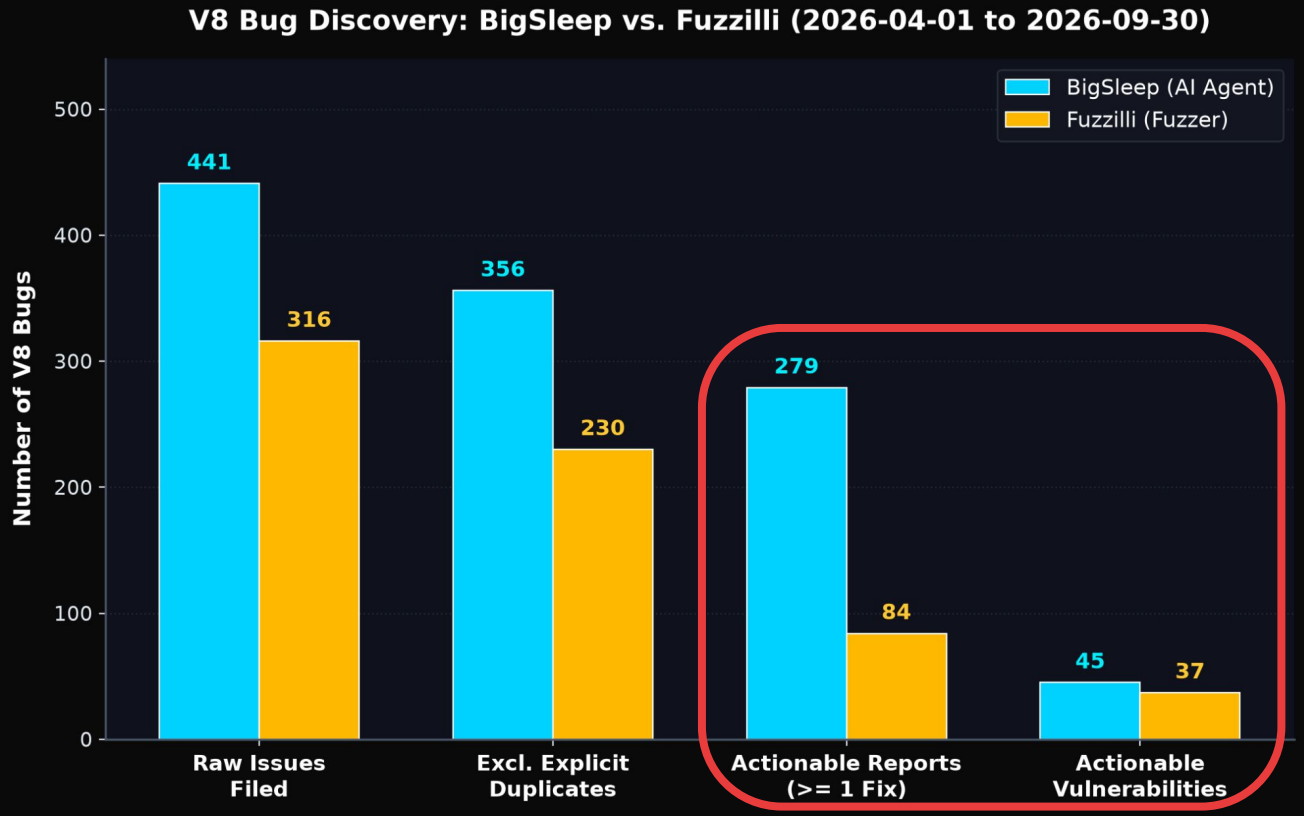
https://issues.chromium.org/issues?q=hotlistid:8067258

Bugs found by Big Sleep and Fuzzilli in Chrome/V8

Real-world statistics, April - September 2026



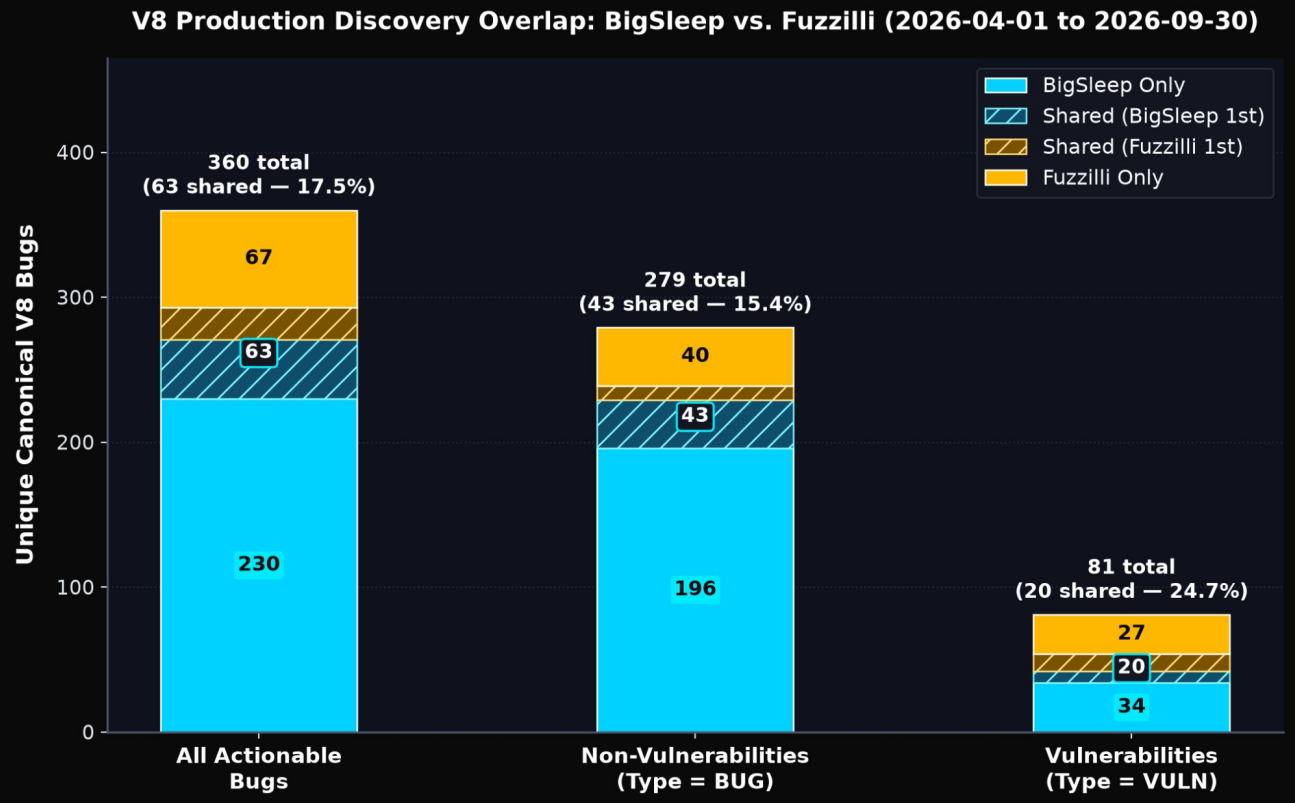
Real-world statistics, April - September 2026



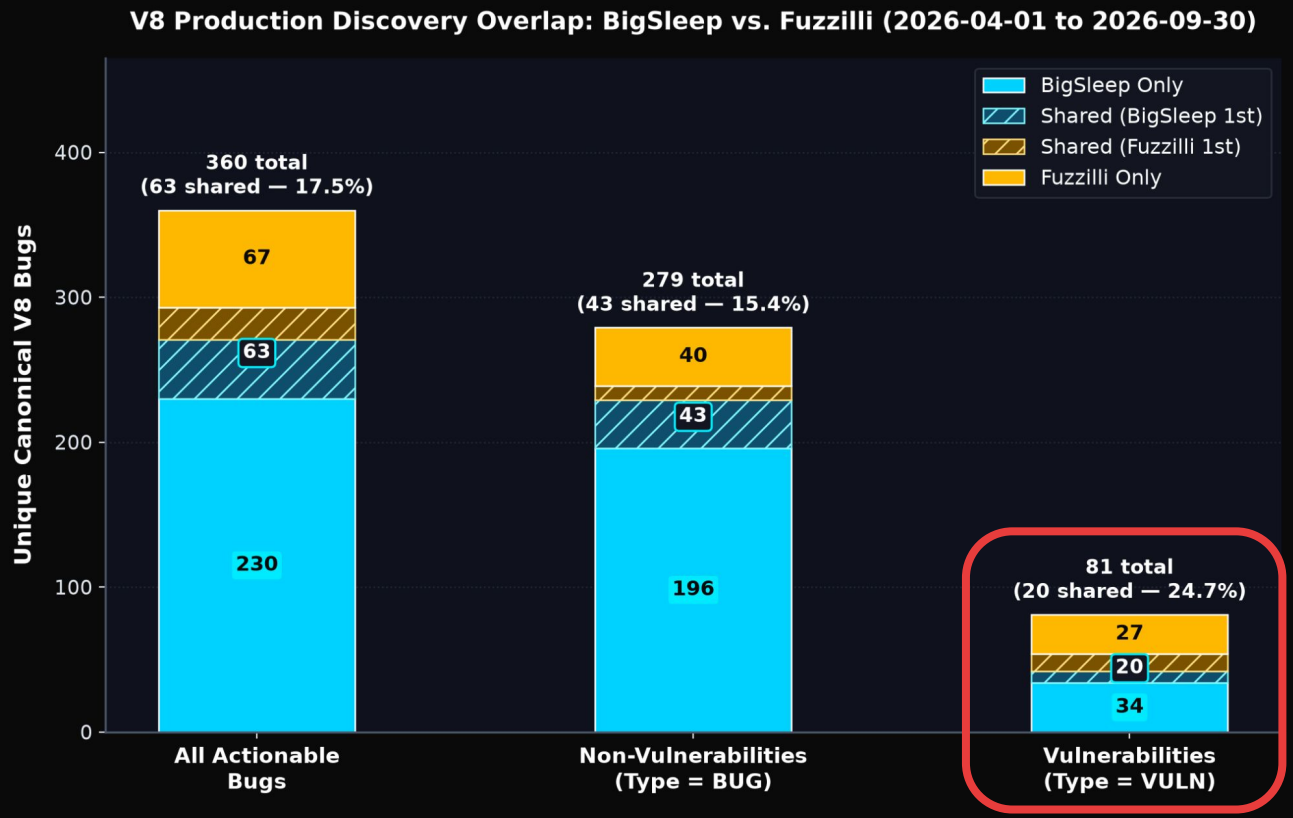
Question 2: Overlap

- **Question:** what is the overlap between both systems
  - I.e. bugs found by both of them vs. bugs only found by one of the two
- **Approach:**
  - **Part 1:** again look at bug tracker data

Bug finding overlap - in “production”



Bug finding overlap - in “production”



Question 2: Overlap

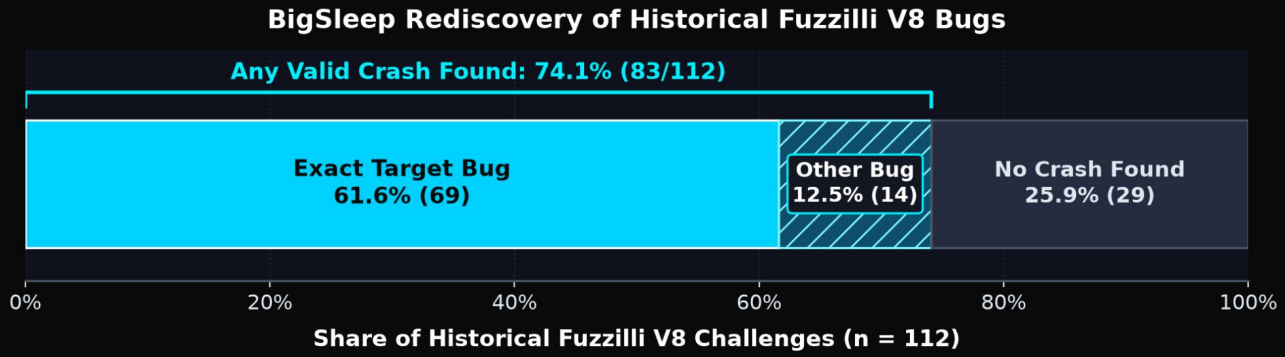
- **Question:** what is the overlap between both systems
  - I.e. bugs found by both of them vs. bugs only found by one of the two
- **Approach:**
  - Part 1: again look at bug tracker data
  - **Part 2: run dedicated experiment!** 🧪

Overlap Experiment

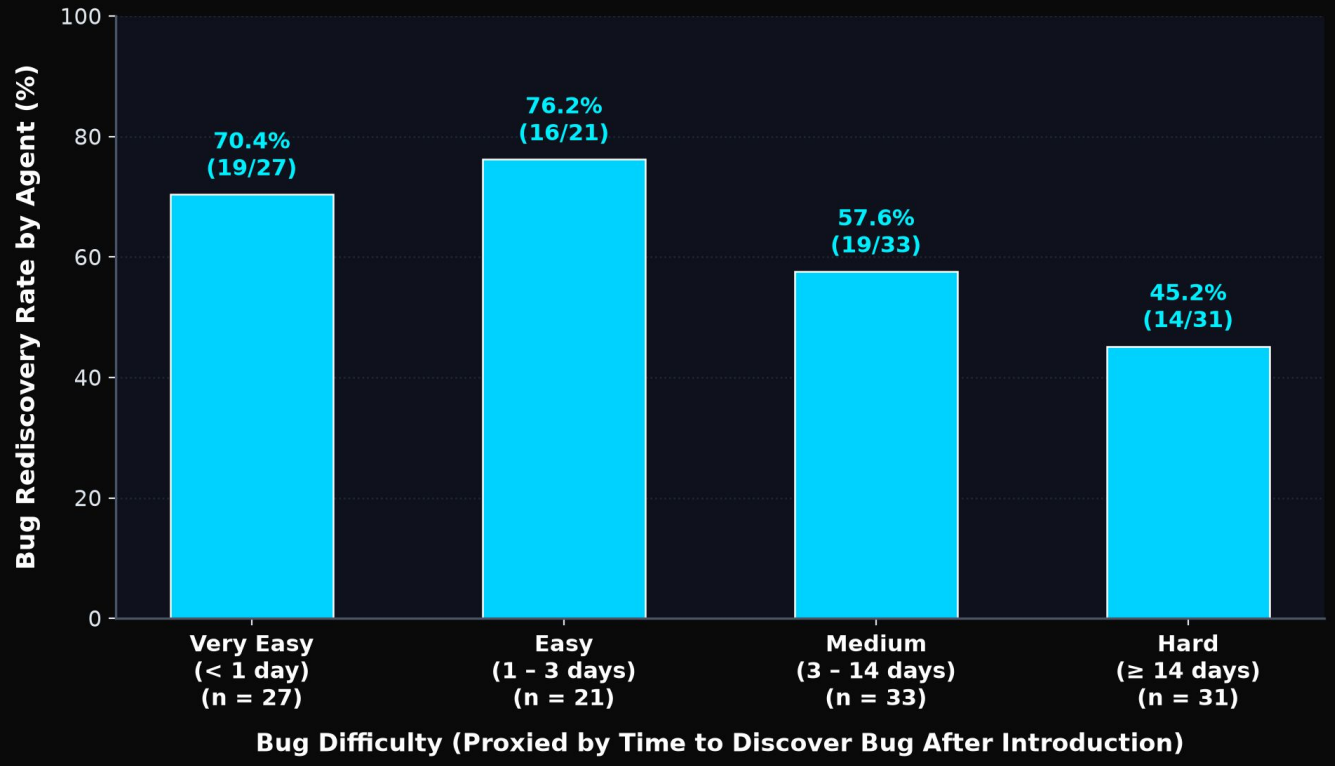
- **Idea:** attempt to re-discover historic **Fuzzilli** bugs with **BigSleep**
- **Approach:**
  - Crawl public Fuzzilli bugs, filter out those that don’t reproduce in a standard environment
  - Identify introducing commit for all bugs
  - Attempt to re-discover bugs from these commits with BigSleep

Overlap Experiment

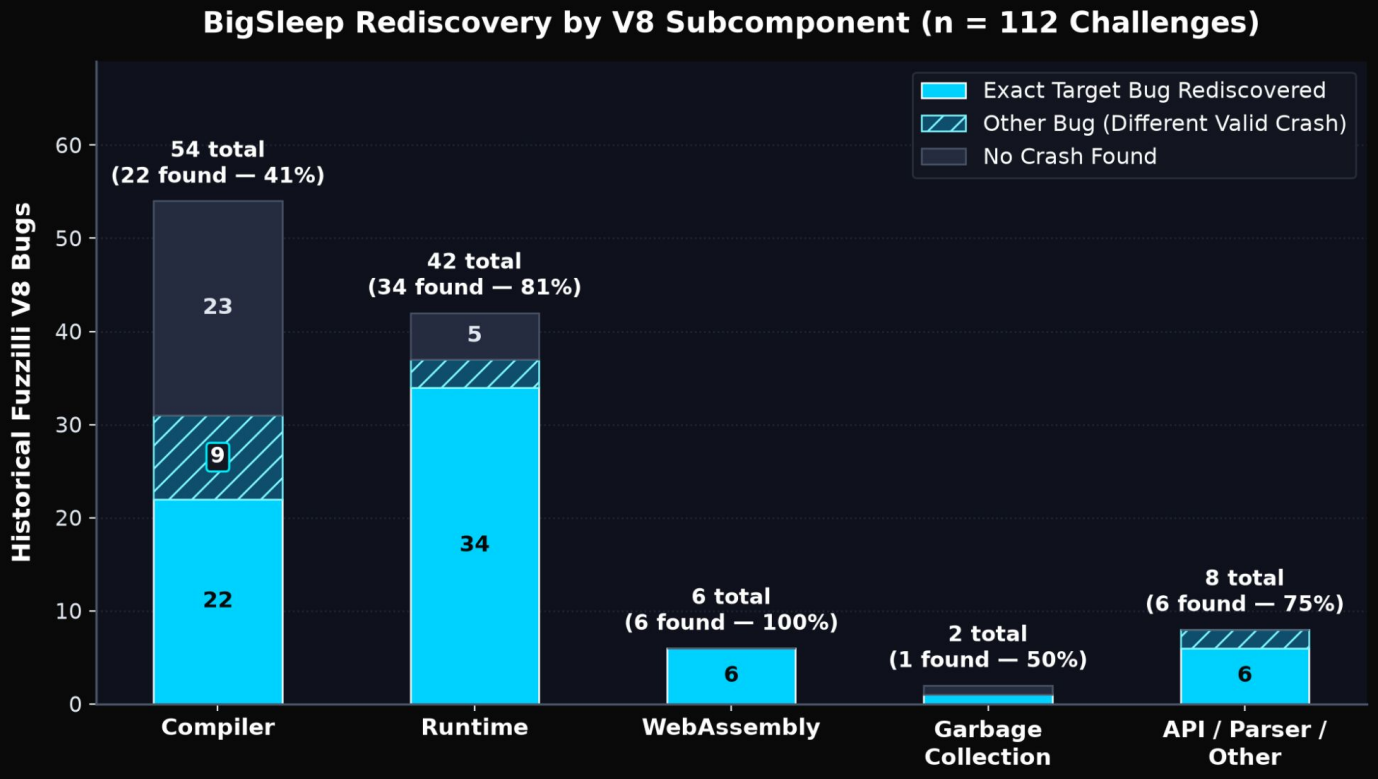
- **Idea:** attempt to re-discover historic **Fuzzilli** bugs with **BigSleep**
- **Approach:**
  - Crawl public Fuzzilli bugs, filter out those that don’t reproduce in a standard environment
  - Identify introducing commit for all bugs
  - Attempt to re-discover bugs from these commits with BigSleep



Rediscovery rate isn’t uniformly distributed…

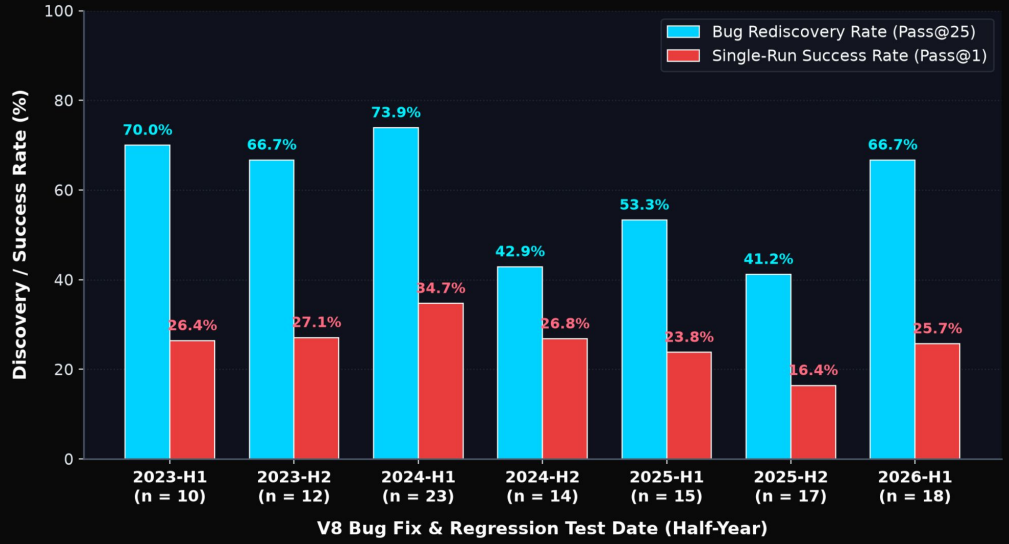


Rediscovery rate isn’t uniformly distributed...



Memorization?

- **Question:** did the model just remember the bugs?
- **Approach:** spot-checking trajectories + difficulty analysis
- Still can't 100% rule out a (small) effect...




Bug Examples

Issue 443765373 (Big Sleep)

- Exception handler is encoded as offset in bytecode
- Offset is stored as **28 bit integer (256 MB)**
- Max. bytecode size: **512 MB**
- => Huge bytecode array will cause truncation of offset
- **=> Arbitrary bytecode execution**

```
function simple() {
  try {
    throw 42;
  } catch (e) {
    return e;
  }
}


[generated bytecode for function: simple]
         0x2ad5001000e4 @    0 : 1b ff f9  Mov <context>, r0
         0x2ad5001000e7 @    3 : 0d 2a     LdaSmi [42]
         0x2ad5001000e9 @    5 : b5        Throw
         0x2ad5001000ea @    6 : d1        Star1
         0x2ad5001000eb @    7 : 8d f8 00  CreateCatchContext r1, [0]
         0x2ad5001000ee @   10 : d2        ...
         ...
         0x2ad5001000f7 @   19 : b7        Return
Handler Table (size = 16)
   from   to       hdlr (prediction,   data)
  (   3,   6)  ->     6 (prediction=1, data=0)
```

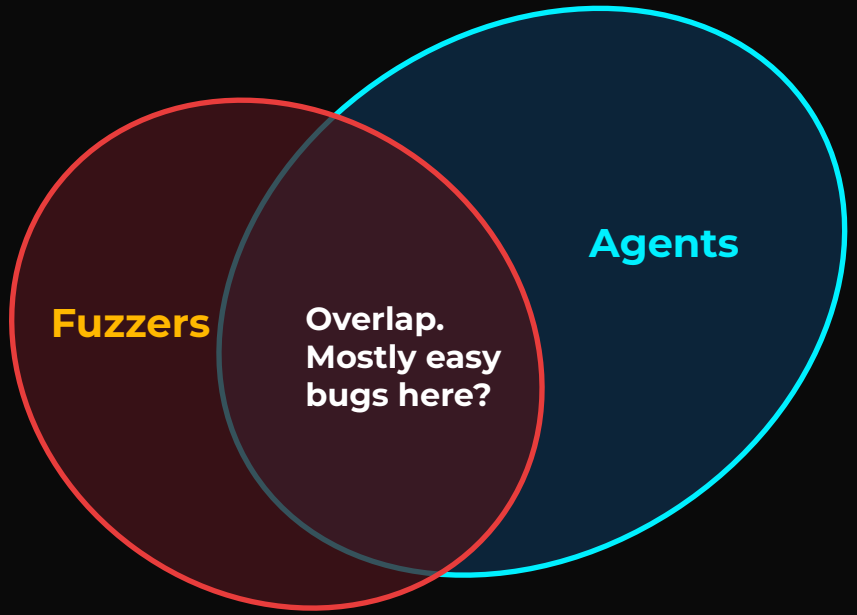
Issue 40076141 (Fuzzilli)

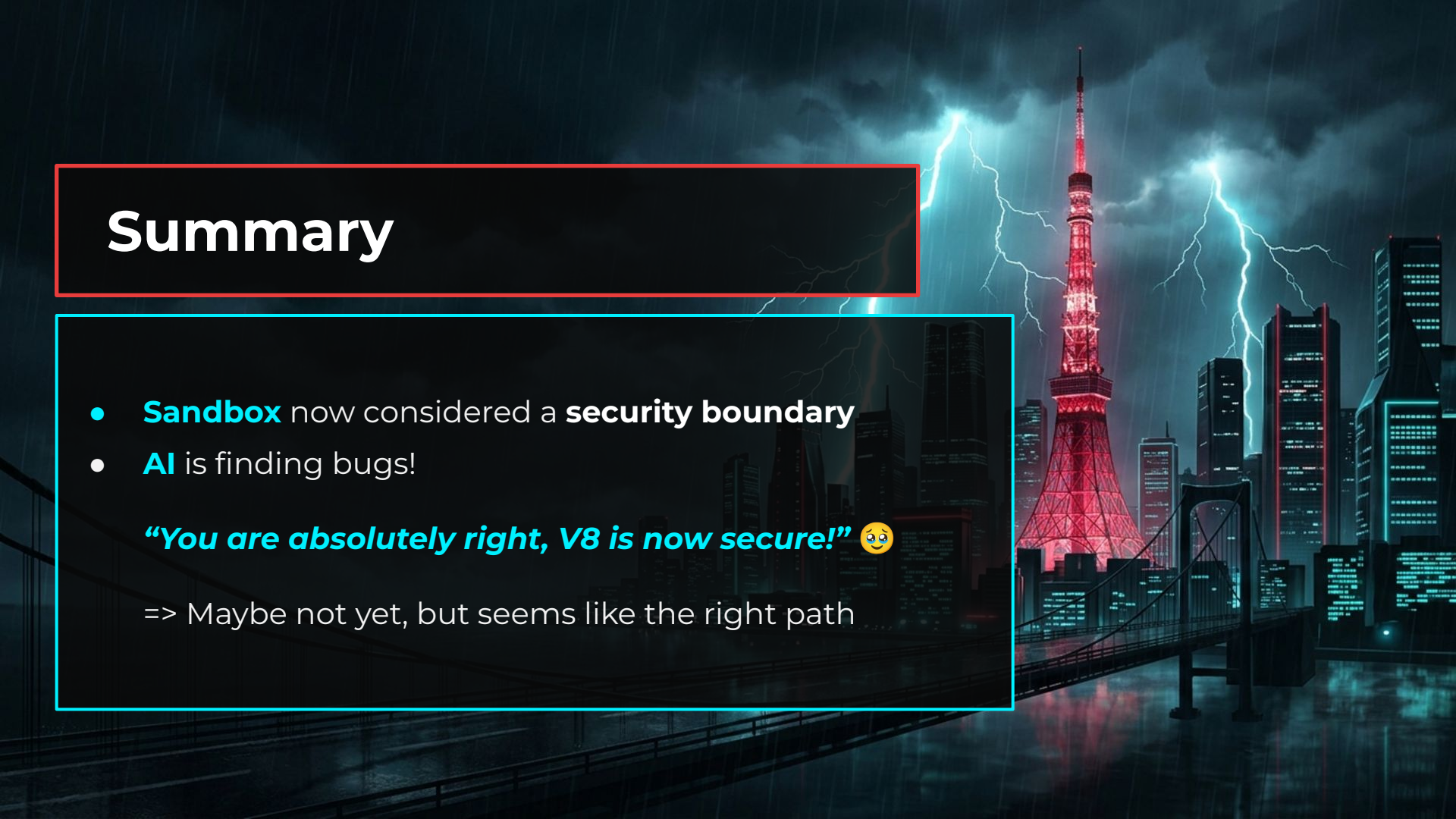
- Bad interaction between two separate optimizations in Maglev compiler
- One optimization makes assumptions that the other later violates
- Each optimization is correct ***in isolation***
  - => Difficult to spot when looking at just one part of the source code
  - => How to find the other parts?
  - **=> BigSleep did not rediscover this bug**

```
function f0() {
    let v2 = 1;
    for (let i3 = v2; i3 >= v2; i3 += v2) {
        v2 = v2 && i3;
    }
}
f0();
```

Fuzzing vs. AI: Summary

- Both approaches find unique bugs
- Overlap seems somewhat concentrated on “easy” bugs
- Fuzzers struggle with specific bug types, agents maybe more with the setup and prompts
- Agentic bug finding is very new but already contributes about as much as existing fuzzers
- **=> Expecting this to grow further**





Summary

- **Sandbox** now considered a **security boundary**
- **AI** is finding bugs!

“You are absolutely right, V8 is now secure!” 🥹

=> Maybe not yet, but seems like the right path