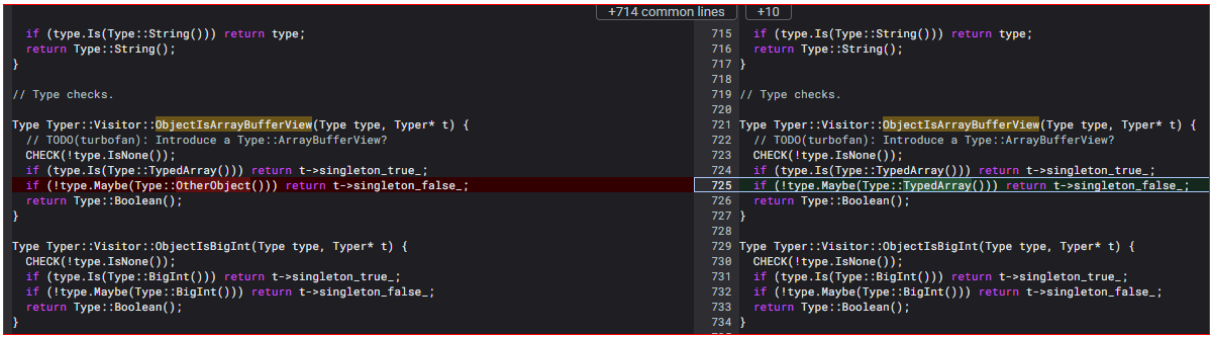
취약점 분석 보고서

결과

코드 이해는 완료하였으나, 결과적으로 취약점이 어떻게 발생 될 수 있는진 알아내지 못하였음.

새롭게 배운 점	C++의 클래스와 메소드 등 코드의 기본 구성이 어떻게 이루어졌는지 얕게나마 알게됨
	소스 트레일과 독시젠의 사용 편의성을 이해하였음
	V8의 바이너리 데이터 사용 방식과 타입 검증을 하는 방식을 얕게나마 이해하게 됨

Patch Diff



Code Anaysis

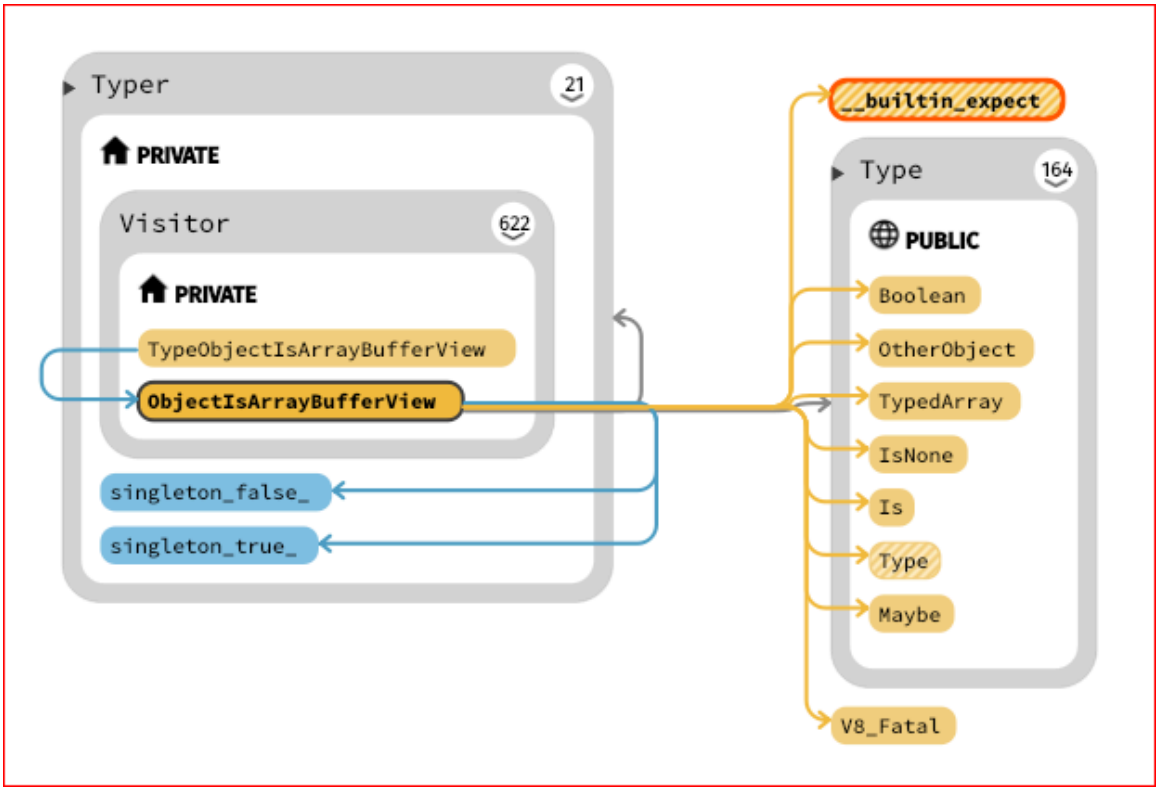
Typer::Visitor

PRIVATE으로 선언된 class, Typer 내부에 Visitor가 존재

```
class V8_EXPORT_PRIVATE Typer {
 public:
  enum Flag : uint8_t {
    kNoFlags = 0,
    kThisIsReceiver = 1u << 0,       // Parameter this is an Object.
    kNewTargetIsReceiver = 1u << 1,  // Parameter new.target is an Object.
  };
  using Flags = base::Flags<Flag>;

  Typer(JSHeapBroker* broker, Flags flags, TFGraph* graph,
        TickCounter* tick_counter);
  ~Typer();
  Typer(const Typer&) = delete;
  Typer& operator=(const Typer&) = delete;

  void Run();
  // TODO(bmeurer,jarin): Remove this once we have a notion of "roots" on
  // TFGraph.
  void Run(const ZoneVector<Node*>& roots,
           LoopVariableOptimizer* induction_vars);

 private:
  class Visitor;
  class Decorator;
```

ObjectIsArraryBufferView

Visitor 클래스의 멤버 함수



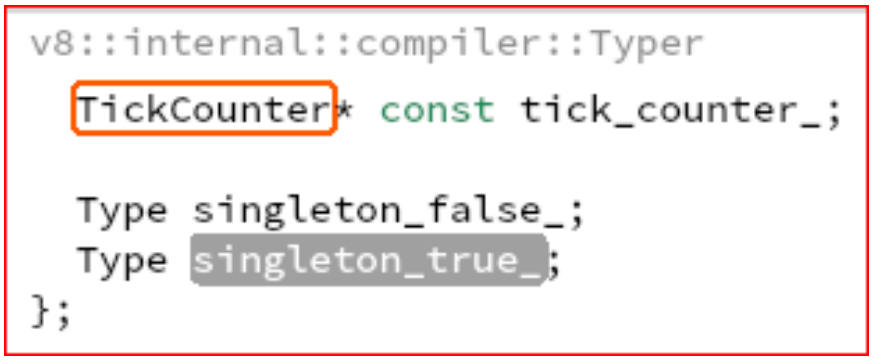
if (type.Is(Type::TypedArray())) return t -> singleton_true_; ,

if (!type.Maybe(Type::OtherObject())) return t -> singleton_false_;

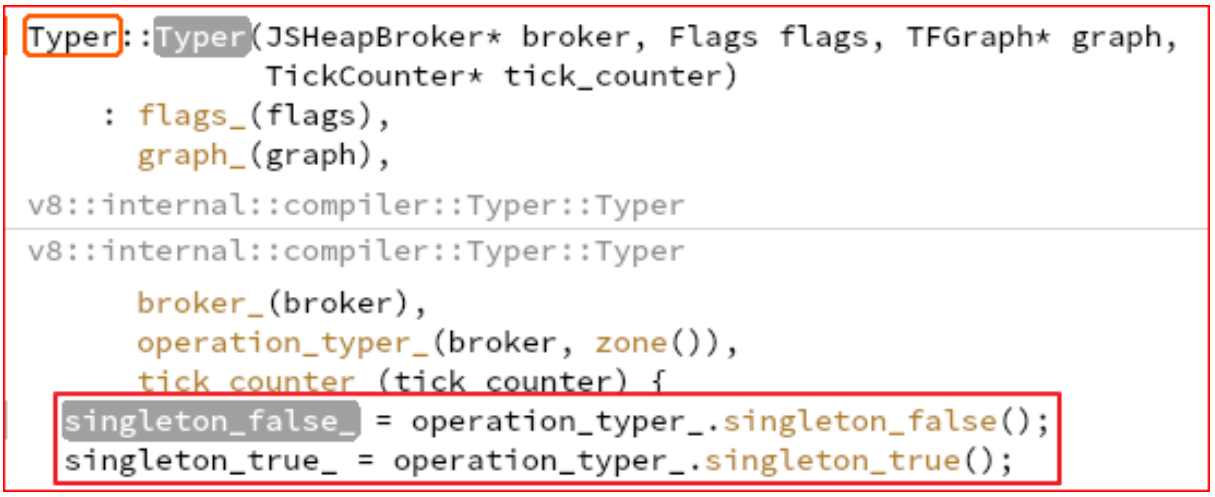
타입 헤더에 Is, Maybe 두 멤버 함수가 선언되어 있으며, Is는 인자로 받은 TypedArray() 의 비트셋과 비교하게 된다. ObjectIsArrayBufferView(Type type, ...) 에서 넘겨받은 this(&type)→payload_와 TypedArray() → payload_ 를 비교하게 된다.

```
v8::internal::compiler::Type
  bool IsNone() const { return payload_ == None().payload_; }
  bool IsInvalid() const { return payload_ == uint64_t{0}; }

  bool Is(Type that) const {
    return payload_ == that.payload_ || this->SlowIs(that);
  }
  bool Maybe(Type that) const;
  bool Equals(Type that) const { return this->Is(that) && that.Is(*this); }

  // Inspection.
  bool IsBitset() const { return payload_ & uint64_t{1}; }
```

비교 결과는 bool 형태로 True, False를 반환하게 되며, True 일 시 Typer 클래스에 정의된 singleton_ture_;를 반환하게 된다. (이 때 singleton_true()의 정확한 기능을 알지 못하였음.)



```
v8::internal::compiler::Typer
  TickCounter* const tick_counter_;

  Type singleton_false_;
  Type singleton_true_;
};
```



```
Typer::Typer(JSHeapBroker* broker, Flags flags, TFGraph* graph,
             TickCounter* tick_counter)
    : flags_(flags),
      graph_(graph),
v8::internal::compiler::Typer::Typer
v8::internal::compiler::Typer::Typer
      broker_(broker),
      operation_typer_(broker, zone()),
      tick_counter_(tick_counter) {
  singleton_false_ = operation_typer_.singleton_false();
  singleton_true_ = operation_typer_.singleton_true();
```

Maybe 멤버 함수의 경우 OtherObject() 타입과 비교하게 된다.

◆ Maybe()

bool v8::internal::compiler::Type::Maybe (Type that) const

Definition at line 638 of file turbofan-types.cc.

```
638                                    {
639   DisallowGarbageCollection no_gc;
640
641   if (BitsetType::IsNone(this->BitsetLub() & that.BitsetLub())) return false;
642
643   // (T1 \/ ... \/ Tn) overlaps T  if  (T1 overlaps T) \/ ... \/ (Tn overlaps T)
644   if (this->IsUnion()) {
645     for (int i = 0, n = this->AsUnion()->Length(); i < n; ++i) {
646       if (this->AsUnion()->Get(i).Maybe(that)) return true;
647     }
648     return false;
649   }
650
651   // T overlaps (T1 \/ ... \/ Tn)  if  (T overlaps T1) \/ ... \/ (T overlaps Tn)
652   if (that.IsUnion()) {
653     for (int i = 0, n = that.AsUnion()->Length(); i < n; ++i) {
654       if (this->Maybe(that.AsUnion()->Get(i))) return true;
655     }
656     return false;
657   }
658
659   if (this->IsBitset() && that.IsBitset()) return true;
660
661   if (this->IsRange()) {
662     if (that.IsRange()) {
663       return Overlap(this->AsRange(), that.AsRange());
664     }
665     if (that.IsBitset()) {
666       bitset number_bits = BitsetType::NumberBits(that.AsBitset());
667       if (number_bits == BitsetType::kNone) {
668         return false;
669       }
670       double min = std::max(BitsetType::Min(number_bits), this->Min());
671       double max = std::min(BitsetType::Max(number_bits), this->Max());
672       return min <= max;
673     }
674   }
675   if (that.IsRange()) {
676     return that.Maybe(*this);  // This case is handled above.
677   }
678
679   if (this->IsBitset() || that.IsBitset()) return true;
680
681   return this->SimplyEquals(that);
682 }
```

OtherObject 타입은 아래와 같이 여러 타입들과 함께
PROPER_ATOMIC_BITSET_TYPE(Low list) 매크로로 선언되어 있다.

```
#define PROPER_ATOMIC_BITSET_TYPE_LOW_LIST(V) \
  V(Negative31,                uint64_t{1} << 6)   \
  V(Null,                      uint64_t{1} << 7)   \
  V(Undefined,                 uint64_t{1} << 8)   \
  V(Boolean,                   uint64_t{1} << 9)   \
  V(Unsigned30,                uint64_t{1} << 10)  \
  V(MinusZero,                 uint64_t{1} << 11)  \
  V(NaN,                       uint64_t{1} << 12)  \
  V(Symbol,                    uint64_t{1} << 13)  \
  V(InternalizedString,        uint64_t{1} << 14)  \
  V(OtherCallable,             uint64_t{1} << 15)  \
  V(OtherObject,               uint64_t{1} << 16)  \
  V(OtherUndetectable,         uint64_t{1} << 17)  \
  V(CallableProxy,             uint64_t{1} << 18)  \
  V(OtherProxy,                uint64_t{1} << 19)  \
  V(CallableFunction,          uint64_t{1} << 20)  \
  V(ClassConstructor,          uint64_t{1} << 21)  \
  V(BoundFunction,             uint64_t{1} << 22)  \
  V(OtherInternal,             uint64_t{1} << 23)  \
  V(ExternalPointer,           uint64_t{1} << 24)  \
  V(Array,                     uint64_t{1} << 25)  \
  V(UnsignedBigInt63,          uint64_t{1} << 26)  \
  V(OtherUnsignedBigInt64,     uint64_t{1} << 27)  \
  V(NegativeBigInt63,          uint64_t{1} << 28)  \
  V(OtherBigInt,               uint64_t{1} << 29)  \
  V(WasmObject,                uint64_t{1} << 30)  \
  V(SandboxedPointer,          uint64_t{1} << 31)
```

위 매크로는 아래와 같이 #define BITSET_TYPE_LIST(V)에 의해 내부 정의가 된다.

```
#define BITSET_TYPE_LIST(V)    \
  INTERNAL_BITSET_TYPE_LIST(V) \
  PROPER_BITSET_TYPE_LIST(V)
```

이렇게 내부 정의된 매크로의 파라미터로 DECLARE_TYPE이 들어가게 되고 type에 k가 붙고

kOtherObject = (uint64_t{1} << 16)가 만들어진다.

```
class V8_EXPORT_PRIVATE BitsetType {
 public:
  using bitset = uint64_t;  // Internal

  enum : bitset {
#define DECLARE_TYPE(type, value) k##type = (value),
    BITSET_TYPE_LIST(DECLARE_TYPE)
```

kOtherObject = (uint64_t{1} << 16)는 에 의해 static 팩토리 함수로 생성된다. (주석에 왜 콘스터럭터라고 부르는진 이해 못하였음)

```
// Constructors.
#define DEFINE_TYPE_CONSTRUCTOR(type, value) \
  static Type type() { return NewBitset(BitsetType::k##type); }
  PROPER_BITSET_TYPE_LIST(DEFINE_TYPE_CONSTRUCTOR)
```

다시 Maybe() 멤버 함수로 돌아와서 OtherObject() 팩토리 함수와 비교하는 코드를 보겠다.

BitsetType::IsNone은 해당 비트가 0인지 아닌지 검사하는 헬퍼 함수다.

that 파라미터로 받은 멤버 함수 OtherObject의 BitsetLub은 16번째 비트만 1이며, &this에 16번째 비트가 1인 경우 & 연산으로 IsNone에 0이 아닌 값을 전달하게 된다. 결국 OtherObject 타입 비트가 존재하게 되면 true를 반환하고 다음 조건 검사로 넘어간다.

```
// Maybe에 OtherObject와 비교하는 구문
if (BitsetType::IsNone(this->BitsetLub() & that.BitsetLub()))
  return false;
```

```
// IsNone 헬퍼 함수 코드 부분
static bool IsNone(bitset bits) { return bits == kNone; }
```

아래 코드는 IsUnion 타입인지 검사하고 만약 IsUnion 일 시 UnionType의 하위 타입을 비교하고, 겹치는 타입이 없다면 false로 넘어가게 된다.

```
// this 가 유니언이면, 구성 요소 중 하나라도 that 과 겹치면 true
if (this->IsUnion()) {
  for (int i = 0, n = this->AsUnion()->Length(); i < n; ++i) {
    if (this->AsUnion()->Get(i).Maybe(that)) return true;
  }
  return false;
}

// 이 경우 that은 Union이 아니므로 넘어감
if (that.IsUnion()) {
  for (int i = 0, n = that.AsUnion()->Length(); i < n; ++i) {
    if (this->Maybe(that.AsUnion()->Get(i))) return true;
  }
  return false;
}
```

```
// IsUnion의 코드 부분으로 Kind가 kUnion인지를 검사
bool IsUnion() const { return IsKind(TypeBase::kUnion); }

// IsKind의 코드 부분으로 return base->kind() == kind; 에서 kUnion이 kind의 kUni
bool IsKind(TypeBase::Kind kind) const {
  if (IsBitset()) return false;
  const TypeBase* base = ToTypeBase();
  return base->kind() == kind;
}
```

아래의 코드는 비트셋과 레인지 타입 가능성 등을 조사한다. 여러 경우의 수가 존재하지만 that이 비트셋 타입일 경우 실행 가능한 조건은 아래 세 가지가 존재한다.

```
if (this->IsBitset() && that.IsBitset()) return true;
if (that.IsBitset()) { ... }
if (this->IsBitset() || that.IsBitset()) return true;;
```

```
// OtherObject는 비트셋 타입이므로 만약 this가 비트셋이면 true가 된다.
if (this->IsBitset() && that.IsBitset()) return true;

// this가 Range 타입인지를 검사
if (this->IsRange()) {

  // that은 Range 타입이 될 수 없으므로 건너뜀
  if (that.IsRange()) {
    return Overlap(this->AsRange(), that.AsRange());
  }

  // that은 비트셋 타입이므로 만약 this가 Range 타입에 해당된다면 아래 코드가 실행됨
  if (that.IsBitset()) {

    // that의 AsBitset()은 kOtherobject의 비트셋이 되고 NumberBits에선 kNone이
    bitset number_bits = BitsetType::NumberBits(that.AsBitset());
    if (number_bits == BitsetType::kNone) {
    // 결과적으로 false가 된다.
      return false;
    }

    double min = std::max(BitsetType::Min(number_bits), this->Min());
    double max = std::min(BitsetType::Max(number_bits), this->Max());
    return min <= max;
  }
}
// 이 경우도 that이 IsRange가 될 수 없으므로 건너뜀
```

```
if (that.IsRange()) {
  return that.Maybe(*this);  // This case is handled above.
}

// that은 OtherObject 타입으로 비트셋 타입에 해당되므로 결과적으로 true를 반환
if (this->IsBitset() || that.IsBitset()) return true;

// 만약 두 타입이 완벽히 같다면 true를 반환 그러나 위에 조건때문에 여기까지 올 수 없
return this->SimplyEquals(that);
```

취약점 분석

취약점은 Arraybufferview의 타입을 검사하는 함수에서 발생된다.

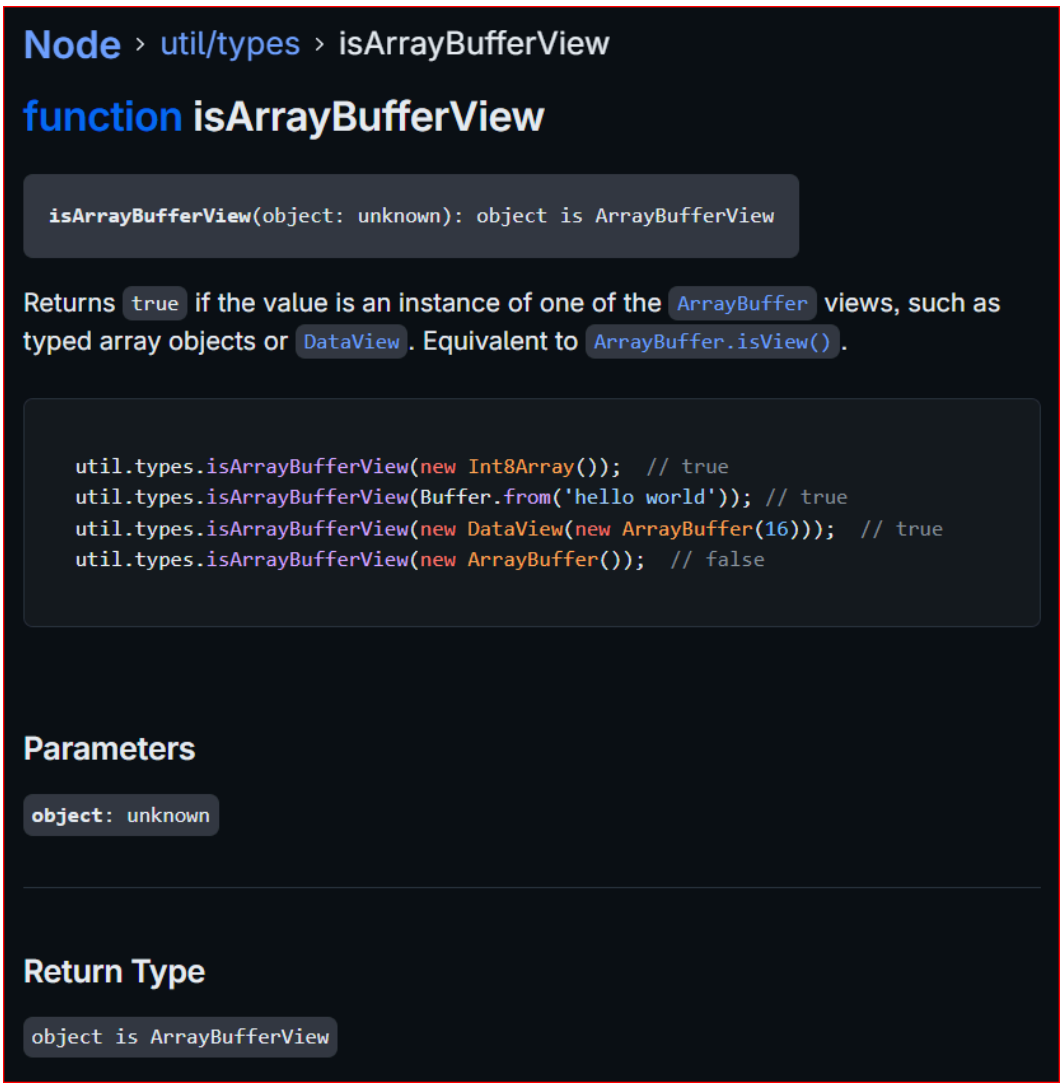
> 💡 취약점 시나리오
>
> if(!type.Maybe(Type::OtherObject())) 에서 TypedArray가 아닌 다른 타입을 넣을 수 있다면 이미 확보된 ArrayBuffer에 초과된 사이즈의 타입을 넣게 되면서 Memory Corruption이 발생 될 수 있다는 가정 하에 아래와 같은 조사를 시작한다. (이는 선생님의 힌트를 이용한 가정이다.)

우선 type.Maybe의 조건이 되려면 Bitset과 Range 타입만 아니면 모든 타입이 true가 되므로 버그를 일으킬 가능성이 높다. 그렇다면 ObjectIsArrayBufferView는 어떻게 실행되는지 알아야 한다.

v8-value.h

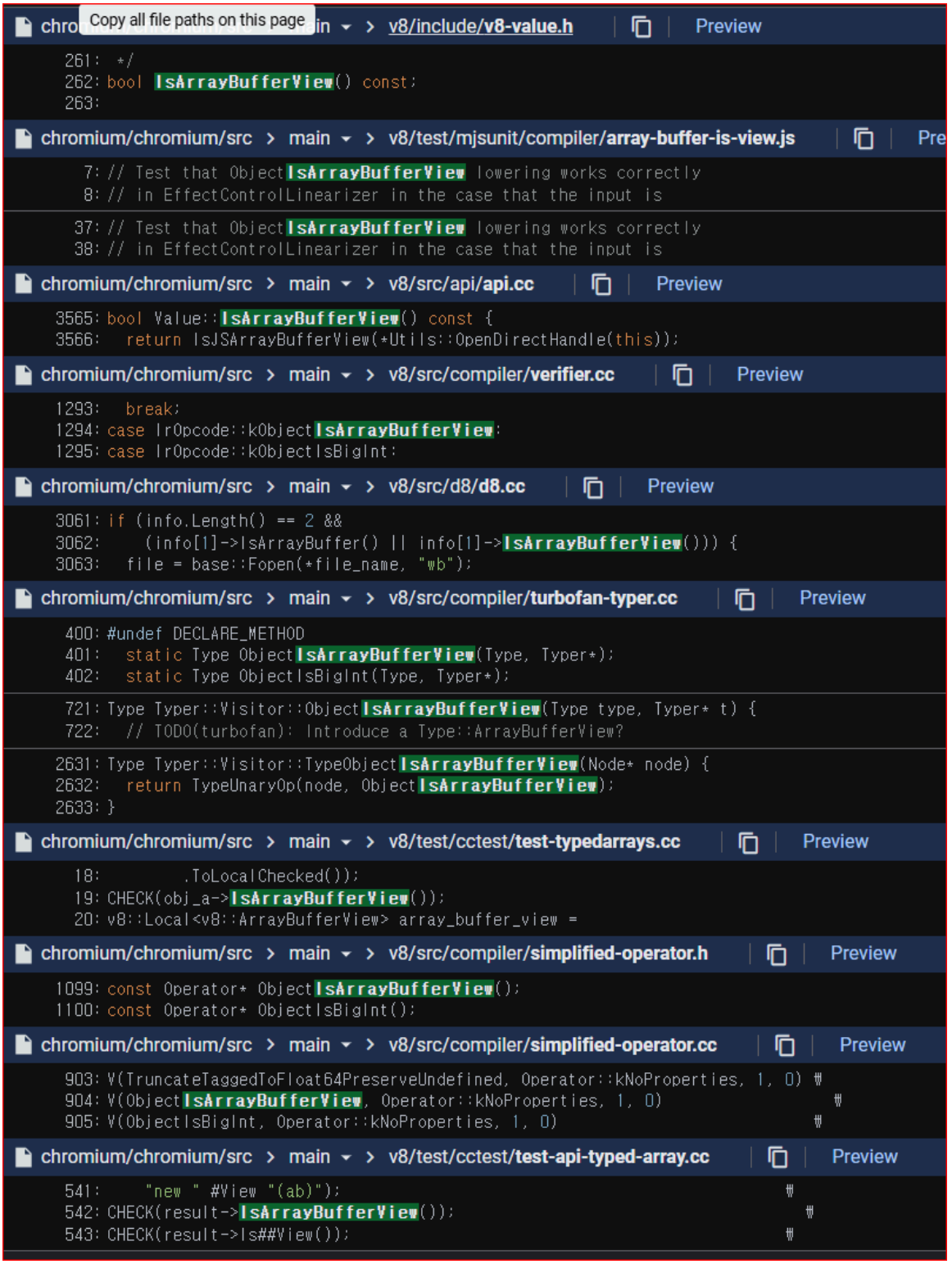
```
251    */
252   bool IsWeakRef() const;
253
254   /**
255    * Returns true if this value is an ArrayBuffer.
256    */
257   bool IsArrayBuffer() const;
258
259   /**
260    * Returns true if this value is an ArrayBufferView.
261    */
262   bool IsArrayBufferView() const;
263
264   /**
265    * Returns true if this value is one of TypedArrays.
266    */
267   bool IsTypedArray() const;
268
269   /**
270    * Returns true if this value is an Uint8Array.
271    */
272   bool IsUint8Array() const;
273
274   /**
275    * Returns true if this value is an Uint8ClampedArray.
276    */
```

IsArrayBufferView()의 실행 시점부터 따라 가보기로 한다. IsArrayBufferView() 함수의 DOCS를 참고해보면 아래와 같다.



IsArrayBufferView()의 리턴 타입이 ObjectIsArrayBufferView()가 된다. 즉, 이 함수에 특정 타입을 넣으면 취약한 메소드를 거쳐서 true와 false를 반환한다. 그러나 이 메소드는 커럽션을 유발할 방법이 전혀 보이지 않는다.

그러나 한 가지의 단서를 얻었다. 그건 ObjectIsArrayBufferView()로 유효한 타입인지를 검증하는 메소드를 잘 찾는다면 유의미한 결과가 나올 수 있다. → 크로미움 코드 서치에 IsArrayBufferView로 서치하였으나, 그러한건 보이지 않았다.



그러다 문득 생각이 들었다. Arraybuffer() 메소드 자체가 이 view 메소드로 타입 검증을 하게 된다면 ?

실제로 그럴듯해 보이는 코드 루틴을 찾았다.

v8-array-buffer.h

```
447    * should be at least as large as the maximum on-heap size of a TypedArray,
448    * was defined in gn with `typed_array_max_size_in_heap`. The default value is
449    * 64 bytes.
450    */
451   v8::MemorySpan<uint8_t> GetContents(v8::MemorySpan<uint8_t> storage);
452
453   /**
454    * Returns true if ArrayBufferView's backing ArrayBuffer has already been
455    * allocated.
456    */
457   bool HasBuffer() const;
458
459   V8_INLINE static ArrayBufferView* Cast(Value* value) {
460 #ifdef V8_ENABLE_CHECKS
461     CheckCast(value);
462 #endif
463     return static_cast<ArrayBufferView*>(value);
464   }
465
466   static constexpr int kInternalFieldCount =
467       V8_ARRAY_BUFFER_VIEW_INTERNAL_FIELD_COUNT;
468   static const int kEmbedderFieldCount = kInternalFieldCount;
469
470  private:
471   ArrayBufferView();
472   static void CheckCast(Value* obj);
473 };
474
475 /**
476  * An instance of DataView constructor (ES6 draft 15.13.7).
477  */
478 class V8_EXPORT DataView : public ArrayBufferView {
479  public:
480   static Local<DataView> New(Local<ArrayBuffer> array_buffer,
481                              size_t byte_offset, size_t length);
482   static Local<DataView> New(Local<SharedArrayBuffer> shared_array_buffer,
483                              size_t byte_offset, size_t length);
484   V8_INLINE static DataView* Cast(Value* value) {
485 #ifdef V8_ENABLE_CHECKS
486     CheckCast(value);
487 #endif
488     return static_cast<DataView*>(value);
489   }
```

Arraybuffer.isView() 라는 게 있었다. 사실 위 DOCS에도 쓰여 있었는데, 눈에 보이지 않았었다. 그런데 크로미움 코드 레퍼런스에는 isView가 보이지 않는다. 정적 메소드라면 이 안에 있어야 할 것 같은데 헤더 파일이라 구현체가 안보이는건가..?

isView() 사용하기

JS

```
ArrayBuffer.isView(); // false
ArrayBuffer.isView([]); // false
ArrayBuffer.isView({}); // false
ArrayBuffer.isView(null); // false
ArrayBuffer.isView(undefined); // false
ArrayBuffer.isView(new ArrayBuffer(10)); // false

ArrayBuffer.isView(new Uint8Array()); // true
ArrayBuffer.isView(new Float32Array()); // true
ArrayBuffer.isView(new Int8Array(10).subarray(0, 3)); // true

const buffer = new ArrayBuffer(2);
const dv = new DataView(buffer);
ArrayBuffer.isView(dv); // true
```

ArrayBuffer - JavaScript | MDN

이 정적 메소드는 js 상에서만 보인다. 그러나 .cc에 구현되어 있어야 할텐데, 어떻게 호출되고 있는지 확인을 못하였다.

```
// tracking is going to pick it up.
(function() {
  function foo(x) {
    return ArrayBuffer.isView({x}.x);
  }
```