



Writing robust client-side code using
Modern JavaScript

or

JavaScript: the **Good**, the **Bad**,
the **Strict** and the **Secure** Parts

Tom Van Cutsem



@tvcutsem

Talk Outline

- This talk is about:
 - The JavaScript language proper
 - Language dialects and features to enable or improve security
- This talk is not about:
 - Security exploits involving JavaScript, or how to avoid specific exploits (e.g. XSS attacks)

Talk Outline

- Part I: 20 years of JavaScript
- Part II: the Good and the Bad parts
- Part III: ECMAScript 5 and Strict Mode
- Part IV: ECMAScript 6
- Part V: Caja and Secure ECMAScript (SES)

Part I: 20 years of Javascript

JavaScript's origins

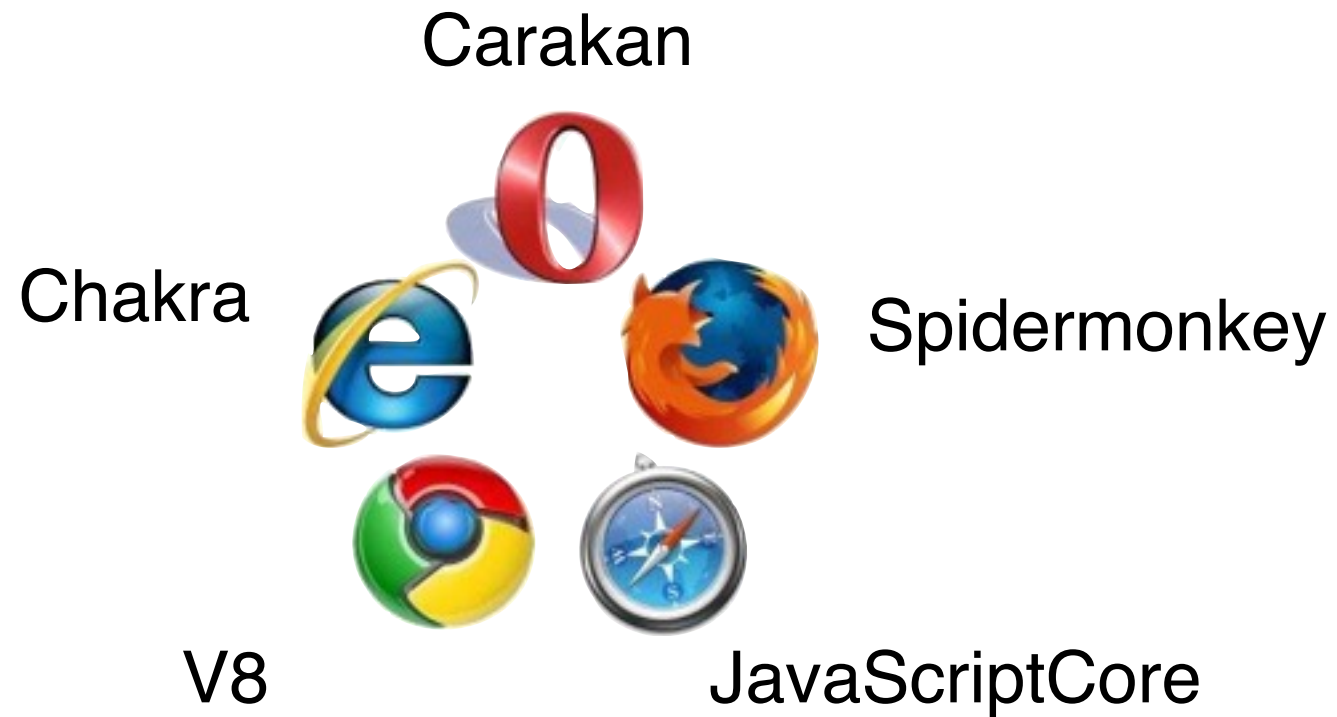
- Invented by Brendan Eich in 1995, to support client-side scripting in Netscape Navigator
- First called *LiveScript*, then *JavaScript*, then standardized as *ECMAScript*
- Microsoft “copied” JavaScript in IE JScript, “warts and all”



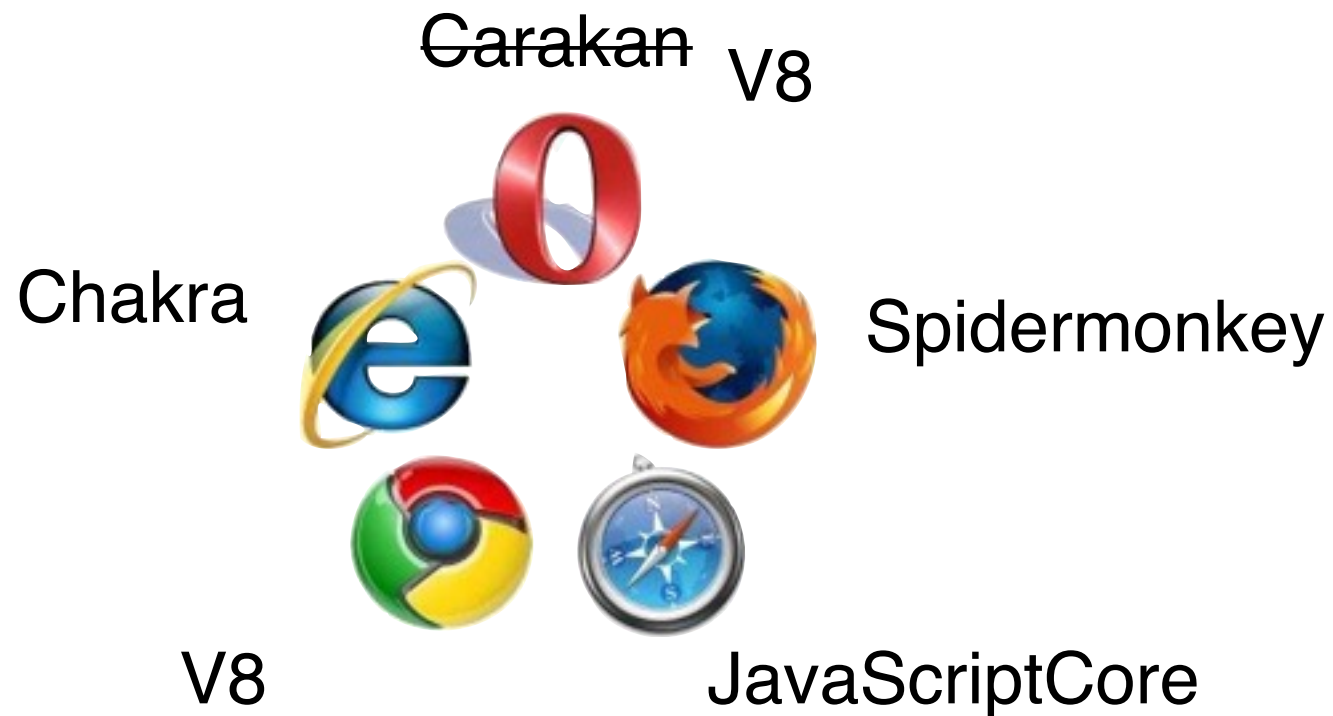
*Brendan Eich,
Inventor of JavaScript*



ECMAScript: “Standard” JavaScript



ECMAScript: “Standard” JavaScript



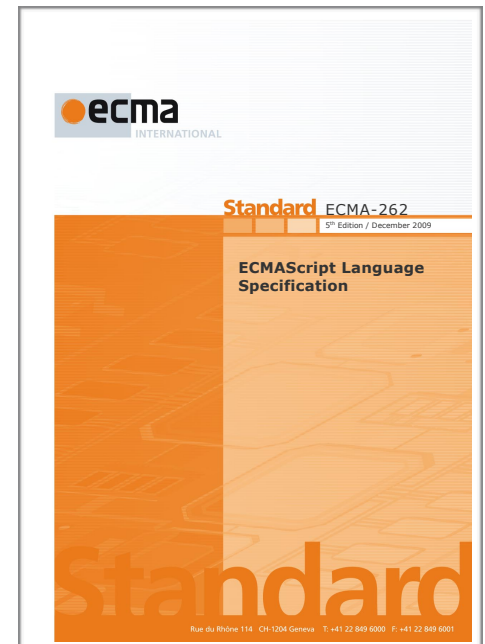
(170.000+ npm packages!)

TC39: the JavaScript “standardisation committee”

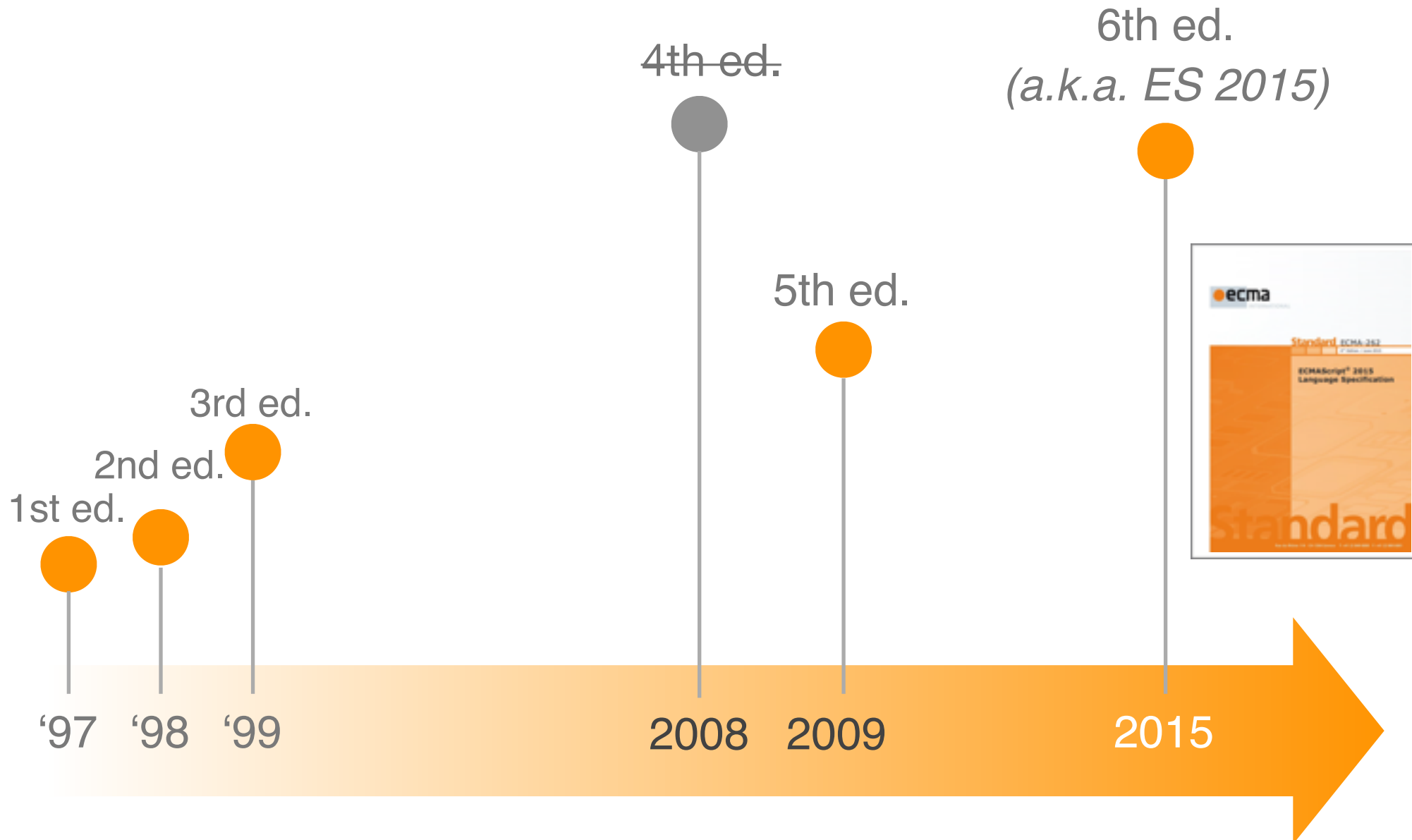
- Representatives from major Internet companies, browser vendors, web organisations, popular JS libraries and academia. Meets bi-monthly.
- Maintains the ECMA-262 specification.
- The spec is a handbook mainly intended for language implementors.



*Allen Wirfs-Brock,
ECMA-262 5th & 6th ed. editor*



A brief history of the ECMAScript spec



Part II: the **Good** and the **Bad** parts

The world's most misunderstood language



*Douglas Crockford,
Inventor of JSON*

See also: “JavaScript: The World's Most Misunderstood Programming Language” by Doug Crockford at <http://www.crockford.com/javascript/javascript.html>

Good Parts: Functions

- Functions are first-class, may capture lexical variables (closures)

```
var add = function(a,b) {  
    return a+b;  
}
```

```
add(2,3); // 5
```

```
function accumulator(s) {  
    return function(n) {  
        return s += n;  
    }  
}
```

```
var a = accumulator(0);  
a(1); // 1  
a(2); // 3
```

```
button.addEventListener('click', function (event) { ... });
```

Good Parts: Objects

- No class declaration needed, literal syntax, arbitrary nesting

```
var bob = {  
  name: "Bob",  
  dob: {  
    day: 15,  
    month: 03,  
    year: 1980  
  },  
  address: {  
    street: "Main St.",  
    number: 5,  
    zip: 94040,  
    country: "USA"  
  }  
};
```

Good Parts: combining objects and functions

- Functions can act as object constructors and methods

```
function makePoint(i,j) {  
  return {  
    x: i,  
    y: j,  
    toString: function() {  
      return '('+ this.x +','+ this.y +')';  
    }  
  };  
}  
  
var p = makePoint(2,3);  
var x = p.x;  
var s = p.toString();
```

The Good Parts



- Functions as first-class objects
- Dynamic objects with prototype-based inheritance
- Object literals
- Array literals

The Bad Parts



- Global variables (no modules)
- `with` statement (breaks lexical scoping)
- Implicit type coercion (`"" == 0`)
- No integers (all numbers are IEEE 754 double-precision floats)
- “var hoisting”: variables *appear* block-scoped but are really function-scoped
- ...

Part III: ECMAScript 5 and **Strict** Mode

ECMAScript 5 Themes

- Support for more robust programming
 - Tamper-proof objects
 - Strict mode

ECMAScript 5 Themes

- Support for more robust programming
 - **Tamper-proof objects**
 - Strict mode

Tamper-proof Objects: motivation

- Objects are *mutable* bags of properties
- Cannot protect an object from modifications by its clients
- Client code may *monkey-patch* shared objects
 - **Powerful**: allows to fix bugs or extend objects with new features
 - **Brittle**: easily leads to conflicting updates
 - **Insecure**: third-party scripts can deliberately modify shared objects

Tamper-proof Objects

```
var point =  
  { x: 0,  
    y: 0  };
```

```
Object.preventExtensions(point);  
point.z = 0; // error: can't add new properties
```

```
Object.seal(point);  
delete point.x; // error: can't delete properties
```

```
Object.freeze(point);  
point.x = 7; // error: can't assign properties
```

ECMAScript 5 Themes

- Support for more robust programming
 - Tamper-proof objects
 - **Strict mode**

Ecmascript 5 Strict mode

- Safer, more robust, subset of the language
- Why?
 - No silent errors
 - True static scoping rules
 - No global object leakage

Ecmascript 5 Strict: no silent errors

- Runtime changes (fail silently outside of strict mode, throw an exception in strict mode)

```
function f() {  
    "use strict";  
    var xfoo;  
    xFoo = 1; // error: assigning to an undeclared variable  
}
```

```
"use strict";  
var p = Object.freeze({x:0,y:0});  
delete p.x; // error: deleting a property from a frozen object
```

Ecmascript 5 Strict: true static scoping

- ECMAScript 5 non-strict is not statically scoped
- Four violations, all fixed in strict mode:
 - `with (obj) { x }` statement
 - `delete x; // may delete a statically visible var`
 - `eval('var x = 8');` // may add a statically visible var
 - Assigning to a non-existent variable creates a new global variable
`function f() { var xfoo; xFoo = 1; }`

Part IV: ECMAScript 6

ECMAScript 6

- Many new additions (too many to list here *). Big-ticket items:
 - Modules
 - Classes
 - Control flow abstractions (iterators, generators, promises)
 - Proper block scoping (let)
 - ...

* see <https://github.com/lukehoban/es6features> for an overview of ES6 features

ECMAScript 6: timeline

6th ed.
(a.k.a. ES 2015)

5th ed.

2009

June
2015



ECMAScript 6 support (february 2016)

Feature name	Current browser	Traceur	Babel + core-js ^[1]	Closure	Type-Script + core-js	es6-shim	IE 11	Edge 12 ^[3]	Edge 13 ^[3]	FF 38 ESR	FF 44	FF 45	CH 46, OP 35 ^[2]	CH 48, OP 36 ^[2]	CH 50, OP 37 ^[2]	SF 6.1, SF 7	SF 7.1, SF 8	SF 9	WK	KO 4.14 ^[4]	PJS
Optimisation																					
proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	0/2	0/2
Syntax																					
default function parameters	6/7	4/7	4/7	4/7	4/7	0/7	0/7	0/7	0/7	3/7	4/7	4/7	0/7	7/7	7/7	0/7	0/7	0/7	7/7	0/7	0/7
rest parameters	5/5	4/5	3/5	2/5	3/5	0/5	0/5	5/5	5/5	4/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5	0/5	5/5	0/5	0/5
spread (...) operator	15/15	15/15	13/15	12/15	4/15	0/15	0/15	12/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	0/15	5/15	9/15	10/15	0/15	0/15
object literal extensions	6/6	6/6	6/6	4/6	6/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	0/6	1/6	5/6	6/6	0/6	0/6
for...of loops	7/9	9/9	9/9	6/9	3/9	0/9	0/9	6/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	0/9	2/9	8/9	9/9	0/9	0/9
octal and binary literals	4/4	2/4	4/4	2/4	4/4	2/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4	4/4	4/4	0/4	0/4
template strings	5/5	4/5	4/5	3/5	3/5	0/5	0/5	4/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5	5/5	5/5	0/5	0/5
RegExp 'y' and 'u' flags	2/4	2/4	2/4	0/4	0/4	0/4	0/4	2/4	4/4	2/4	2/4	2/4	0/4	2/4	2/4	0/4	0/4	0/4	0/4	0/4	0/4
destructuring declarations	21/22	20/22	21/22	18/22	14/22	0/22	0/22	0/22	0/22	19/22	19/22	19/22	0/22	21/22	21/22	0/22	9/22	19/22	22/22	0/22	0/22
destructuring assignment	0/24	23/24	24/24	15/24	18/24	0/24	0/24	0/24	0/24	20/24	21/24	21/24	0/24	23/24	23/24	0/24	12/24	21/24	24/24	0/24	0/24
destructuring parameters	22/23	19/23	20/23	17/23	14/23	0/23	0/23	0/23	0/23	18/23	18/23	18/23	0/23	22/23	22/23	0/23	10/23	18/23	22/23	0/23	0/23
Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	0/2	0/2	2/2	2/2	0/2	1/2	1/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2	0/2	0/2
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	0/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	1/2	0/2	0/2
Bindings																					
const	8/8	6/8	6/8	6/8	6/8	0/8	0/8	8/8	8/8	8/8	8/8	8/8	5/8	8/8	8/8	1/8	1/8	1/8	8/8	2/8	1/8
let	10/10	8/10	8/10	8/10	7/10	0/10	8/10	8/10	8/10	0/10	8/10	8/10	5/10	10/10	10/10	0/10	0/10	0/10	10/10	0/10	0/10
block-level function declaration ^[12]	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	No	No	No	No	No	No
Functions																					
arrow functions	13/13	11/13	9/13	11/13	9/13	0/13	0/13	9/13	13/13	8/13	11/13	13/13	11/13	13/13	13/13	0/13	0/13	0/13	13/13	0/13	0/13
class	23/23	16/23	18/23	8/23	16/23	0/23	0/23	0/23	23/23	0/23	0/23	23/23	0/23	23/23	23/23	0/23	0/23	15/23	23/23	0/23	0/23
super	8/8	7/8	4/8	4/8	7/8	0/8	0/8	0/8	8/8	0/8	0/8	8/8	0/8	8/8	8/8	0/8	0/8	8/8	8/8	0/8	0/8
generators	21/25	23/25	22/25	18/25	0/25	0/25	0/25	0/25	25/25	20/25	21/25	23/25	19/25	21/25	22/25	0/25	0/25	0/25	25/25	0/25	0/25
Built-ins																					
typed arrays	43/46	0/46	45/46	0/46	45/46	0/46	16/46	42/46	44/46	41/46	42/46	42/46	43/46	44/46	44/46	18/46	18/46	18/46	44/46	8/46	18/46
Map	16/18	13/18	18/18	0/18	18/18	14/18	7/18	15/18	17/18	14/18	16/18	17/18	16/18	16/18	16/18	0/18	10/18	17/18	18/18	0/18	0/18
Set	16/18	13/18	18/18	0/18	18/18	14/18	7/18	15/18	17/18	14/18	16/18	17/18	16/18	16/18	16/18	0/18	10/18	17/18	18/18	0/18	0/18
WeakMap	9/10	5/10	10/10	0/10	10/10	0/10	4/10	9/10	9/10	7/10	8/10	8/10	9/10	9/10	9/10	0/10	5/10	10/10	10/10	0/10	0/10

(Source: Juriy Zaytsev (kangax)
<http://kangax.github.io/es5-compat-table/es6>)



ECMAScript 5 support (october 2015)

Feature name	Current browser	97%	42%	95%	87%	87%	87%	100%	97%	97%	92%	100%	92%	100%	95%	97%	97%
		Current browser	et5-drom	IE 9+	IE 10+	FF 21+	SF 6+	WebKit	CH 25+, OP 15+	OP 12.10	Rang 4.13	BOJAN	Stone 1.7	PhantomJS 2.0	JS	Android 4.0+	iOS 9
Object.create	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.defineProperty	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.defineProperties	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.getPrototypeOf	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.keys	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.seal	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.freeze	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.preventExtensions	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.isSealed	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.isFrozen	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.isExtensible	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.getOwnPropertyDescriptor	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.getOwnPropertyNames	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Date.prototype.toISOString	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Date.now	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Date.prototype.toJSON	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Date.parse or invalid dates	No	Yes	No	No	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Array.isArray	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
JSON.stringify	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Function.prototype.bind	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
String.prototype.trim	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.indexOf	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.lastIndexOf	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.every	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.some	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.forEach	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.map	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.filter	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.reduce	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.reduceRight	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Getter in property initializer	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Setter in property initializer	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Property access on strings	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Reserved words as property names	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Zero-width chars in identifiers	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
parseInt() ignores leading zeros	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	Yes
Immutable undefined	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Strict mode	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes

(Source: Juriy Zaytsev (kangax)
<http://kangax.github.io/es5-compat-table/es5>)



ECMAScript 6 compilers

- Compile ECMAScript 6 to ECMAScript 5
- **Babel:** focus on producing readable (as-if hand-written) ES5 code. Supports JSX as well.
- Microsoft **TypeScript:** technically not ES6 but roughly a superset of ES6. Bonus: type inference and optional static typing.



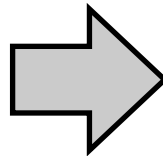
ECMAScript 6: modules

- Prior to ES6: scripts depend on global variables for linkage

Bad

```
<script>
var x = 0; // global
var myLib = {
  inc: function() {
    return ++x;
  }
};
</script>
```

```
<script>
var res = myLib.inc();
</script>
```



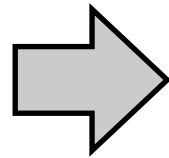
Better

```
<script>
var myLib = (function(){
  var x = 0; // local
  return {
    inc: function() {
      return ++x;
    }
  };
})();
</script>
```

ECMAScript 6: modules

- All code inside a module is implicitly opted into strict mode!

```
<script>
var x = 0; // global
var myLib = {
  inc: function() {
    return ++x;
  }
};
</script>
```



```
<script type="module"
      name="myLib">
var x = 0; // local!
export function inc() {
  return ++x;
}
</script>
```

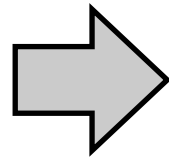
```
<script>
var res = myLib.inc();
</script>
```

```
<script type="module">
import { inc } from 'myLib';
var res = inc();
</script>
```

ECMAScript 6: modules

- All code inside a module is implicitly opted into strict mode!

```
<script>
var x = 0; // global
var myLib = {
  inc: function() {
    return ++x;
  }
};
</script>
```



```
<script type="module"
name="myLib">
var x = 0; // local!
export function inc() {
  return ++x;
}
</script>
```

```
<script>
var res = myLib.inc();
</script>
```

```
<script type="module">
import { inc } from 'myLib';
var res = inc();
</script>
```

ECMAScript 6: modules

- Dynamic module loader API (WHATWG Draft Spec *)

```
System.import("lib/math").then(function(m) {  
  alert("2π = " + m.sum(m.pi, m.pi));  
});
```

```
// create a sandboxed environment  
var loader = new Loader({  
  global: wrap(window) // replace 'console.log'  
});  
loader.eval("console.log(\"hello world!\");");
```

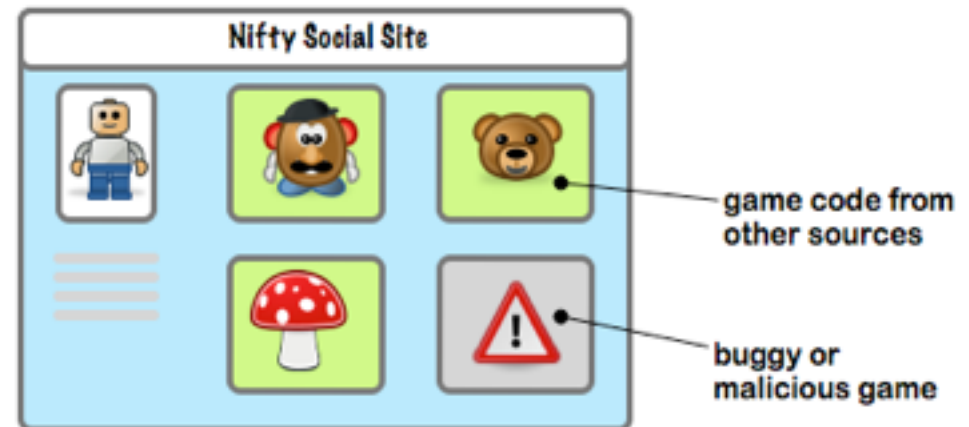
(Source: <https://babeljs.io/docs/learn-es2015/>)

* See <http://whatwg.github.io/loader/>

Part V: Caja and **Secure** ECMAScript (SES)

Caja

- Caja enables the safe embedding of third-party active content inside a single web page
- Secures Google Earth Engine, Google Sites, Google Apps Scripts
- More generally: Gadgets, Mashups:



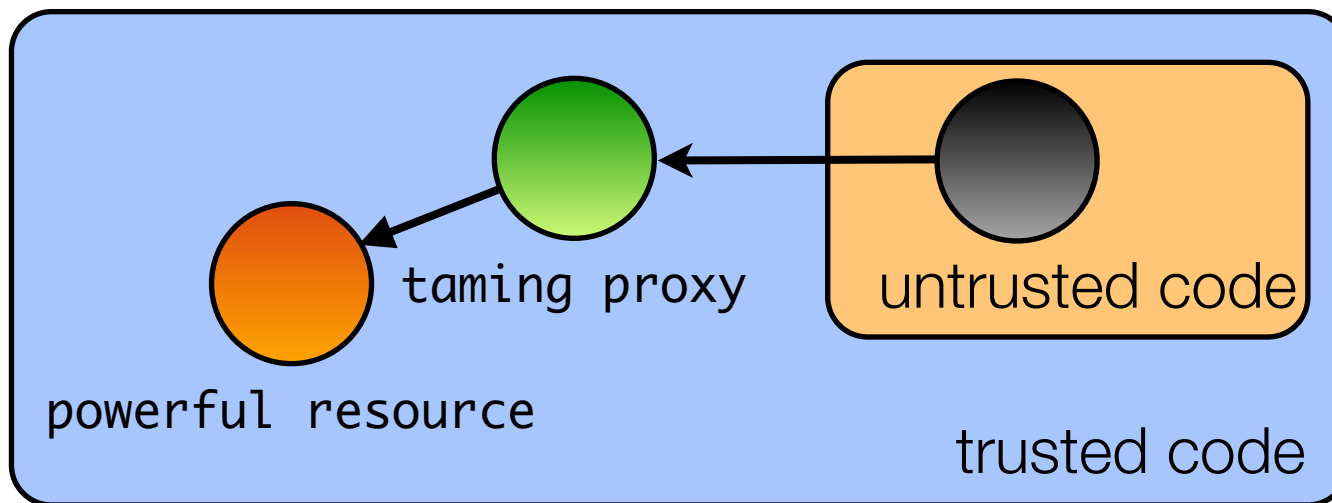
<https://developers.google.com/caja/docs/about/>

Caja

- Caja consists of:
 - A HTML and CSS sanitizer (sandbox scripts embedded in HTML/CSS)
 - A capability-secure JavaScript subset (SES)
 - A safe DOM wrapper
- SES is the portion of Caja responsible for securing JavaScript

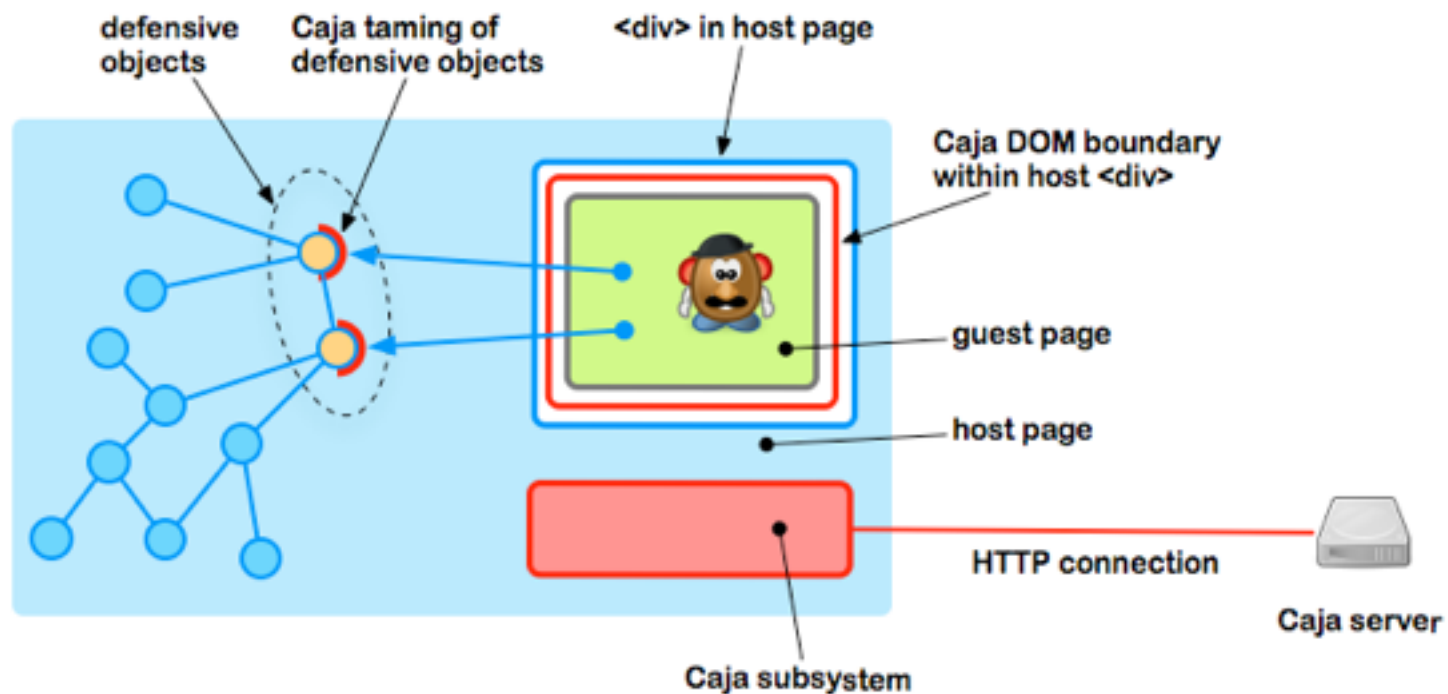
Capability-based security

- Caja uses object capabilities to express security policies
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects
- Common to wrap powerful objects with restrictive proxies (“taming”)



Caja : Taming the DOM

- Caja proxies the DOM. Untrusted content interacts with a virtual DOM, never with the real DOM.



Secure ECMAScript

- Implemented as a library on top of ES5/strict
- Include as first script, before any other JavaScript code runs:

```
<script src="startSES.js"></script>
```

Secure ECMAScript

```
<script src="startSES.js"></script>
```

- Deep-frozen global environment (incl. frozen global object)
 - Can't update properties of Object, Array, Function, Math, JSON, etc.
- Whitelisted global environment
 - No “powerful” non-standard globals (e.g. document, window, XMLHttpRequest, ...)
 - Code that spawns an SES environment may provide selective access to these
- Patches eval and Function to accept only ES5/strict code that can only name global variables on the whitelist

Secure ECMAScript

- Problem with SES as a library: slow initial page load due to transitive freezing of all standard library objects
- Draft proposal available to standardise SES as part of ES7
- One new API call: `Reflect.confine(src, globals)`
evals `src` in a new SES “realm”, with access only to standard library + its own global object containing the parameter-passed `globals`

`Reflect.confine("x + y", {x:1, y:2}) => 3`

SES enables safe mobile code!

Wrap-up

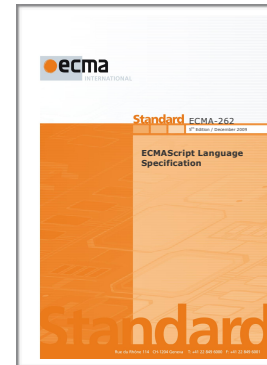
Wrap-up

ES3

ES5

ES5/strict

SES



JavaScript:

the **Good**,
the **Bad**,

the **Strict**,

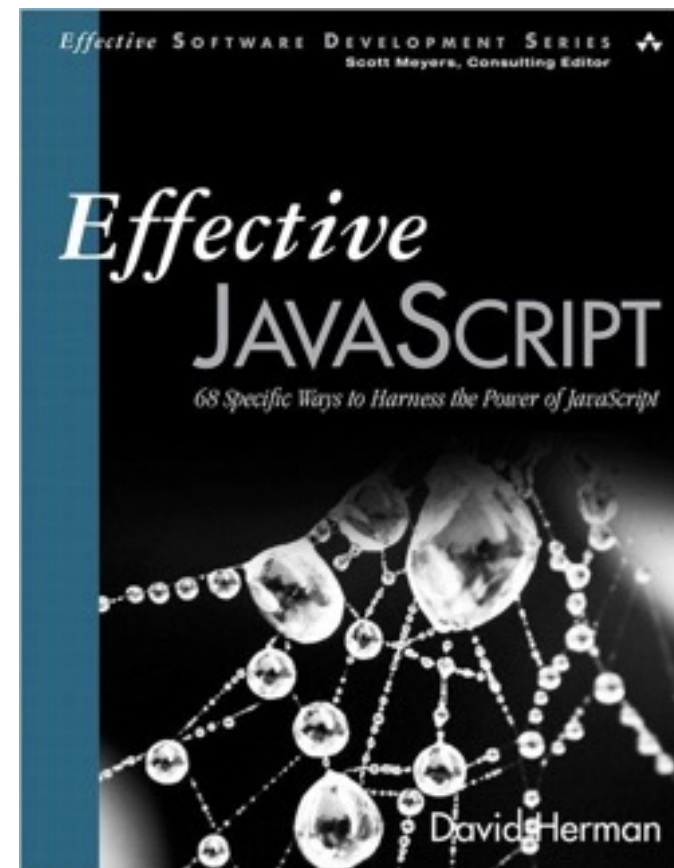
and
the **Secure** parts.

Take-home messages

- Strict mode: a saner JavaScript (opt-in in ES5)
- ES6 builds on strict mode (classes and modules)
- Secure ECMAScript (SES) builds on strict mode
- SES enables safe mobile code

References

- Warmly recommended: Doug Crockford on JavaScript
<http://goo.gl/FGxmM> (YouTube playlist)



References

- ECMAScript 5:
 - “Changes to JavaScript Part 1: EcmaScript 5” (Mark S. Miller, Waldemar Horwat, Mike Samuel), Google Tech Talk (May 2009)
 - “Secure Mashups in ECMAScript 5” (Mark S. Miller), QCon 2012 Talk <http://www.infoq.com/presentations/Secure-Mashups-in-ECMAScript-5>
- Caja: <https://developers.google.com/caja>
- SES: <https://github.com/FUDCo/ses-realm> and <https://github.com/google/caja/wiki/SES>
- HTML templating with template strings: <http://www.2ality.com/2015/01/template-strings-html.html>
- ES6 latest developments: <https://github.com/tc39> and the es-discuss@mozilla.org mailing list.
ES6 Modules: <http://www.2ality.com/2014/09/es6-modules-final.html>
<https://github.com/ModuleLoader/es6-module-loader> and <http://jsmodules.io/>
ES6 Proxies: <http://www.2ality.com/2014/12/es6-proxies.html>
R. Mark Volkmann: “Using ES6 Today!”: <http://sett.ociweb.com/sett/settApr2014.html>



Thanks for listening!

Writing robust client-side code using
Modern JavaScript

or

JavaScript: the **Good**, the **Bad**,
the **Strict** and the **Secure** Parts

Tom Van Cutsem



@tvcutsem