Writing robust client-side code using Modern JavaScript
or
JavaScript: the Good, the Bad, the Strict and the Secure Parts

Tom Van Cutsem
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

1st ed. '97
2nd ed. '98
3rd ed. '99
4th ed. 2008
5th ed. 2009
Part II: the **Good** and the **Bad** parts
The world’s most misunderstood language

Good Parts: Functions

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

```javascript
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

```javascript
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

```javascript
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

```javascript
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)

```javascript
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
    ```javascript
    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 support (February 2016)

(Source: Juriy Zaytsev (kangax)
ECMAScript 5 support (October 2015)

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

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

### Better
```html
<script>
var x = 0; // local
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!

```javascript
<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!

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

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

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

```html
<scrip
type="module">
import { inc } from 'myLib';
var res = inc();
</script>
```
ECMAScript 6: modules

• Dynamic module loader API (WHATWG Draft Spec *)

```javascript
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.

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

- 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

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

  \[
  \text{Reflect.confine(“x + y”, \{x:1, y:2\}) } \Rightarrow \text{ 3}
  \]

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)

• 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 Proxies: http://www.2ality.com/2014/12/es6-proxies.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