Are your JavaScript-based **protections really secure?**

Pedro Fortuna

SecAppDev 2018

jscrambler

About **Me**

PEDRO FORTUNA

CO-FOUNDER & CTO @ **JSCRAMBLER**

SECURITY, JAVASCRIPT
@PEDROFORTUNA

Agenda

1
Recap

2
JavaScript Quirks

3
Code DeObfuscation Techniques

4
Measuring Obfuscation Resilience

5
Testing & Compliance

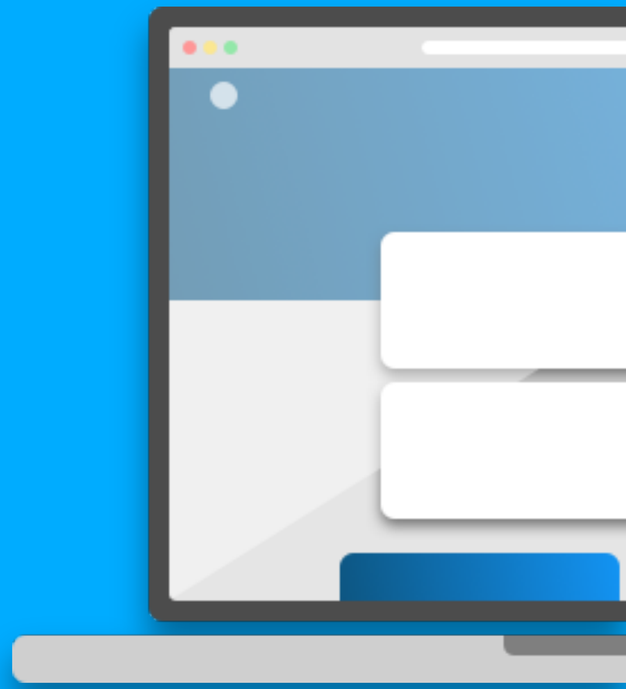
6
Conclusions

7
A Challenge

8
Q & A

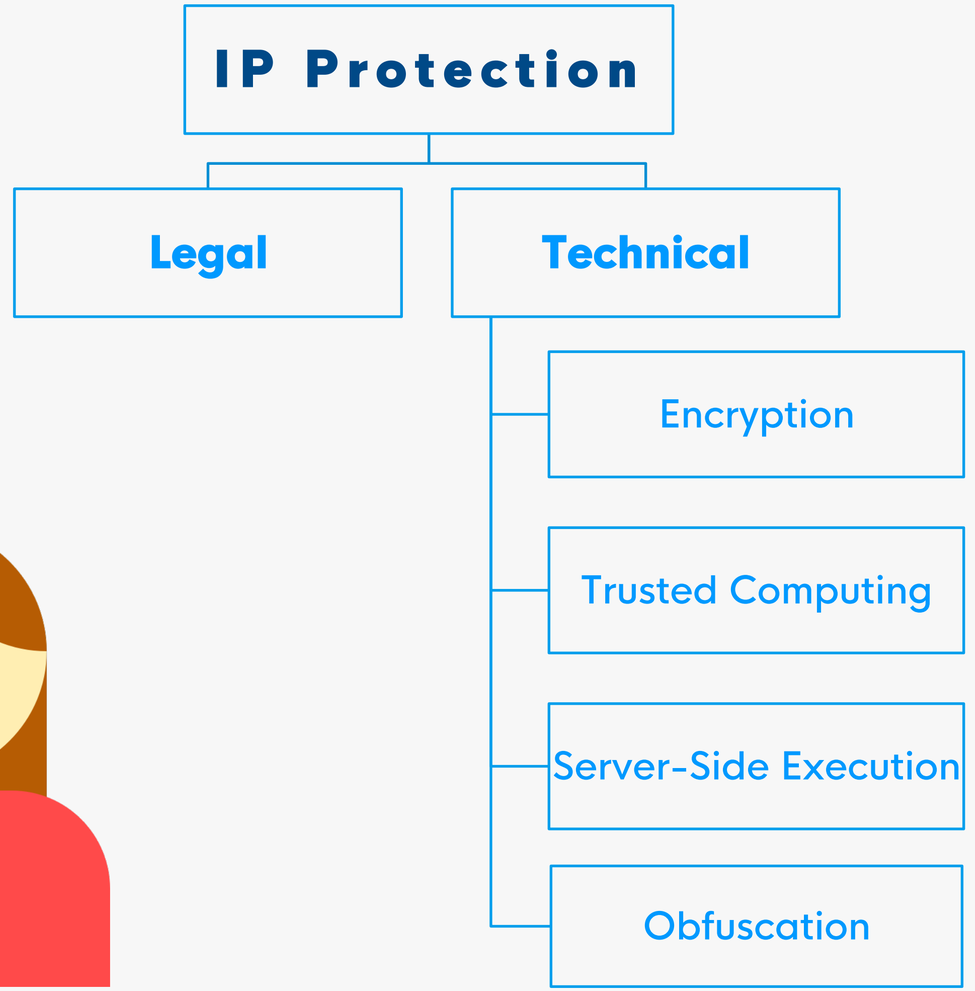
RECAP

PART 1



jscrambler

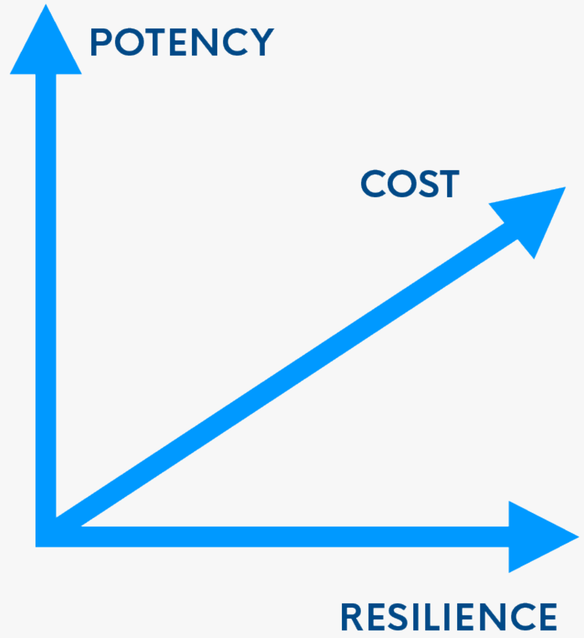
Intellectual Property **Protection**



Measuring Obfuscation

Obfuscation quality

- **Potency**
  - How much more difficult to read and understand (for a human)
- **Resilience**
  - Resistance to automated deobfuscation techniques
- **Cost**
  - Execution time & space penalty introduced



Obfuscation Potency

```
function print_exp2(x) {
    var res = x * x;
    console.log('exp2 = ' + res);
}
```

Add Predicates
Grow Program Size

Simple Optimization
Techniques

```
function print_exp2(x) {
    var res = x * x;
    if (3==2) res = x + 2;
    console.log('exp2 = ' + res);
    if (2<1) console.log('print something');
}
```

Other **Takeaways**

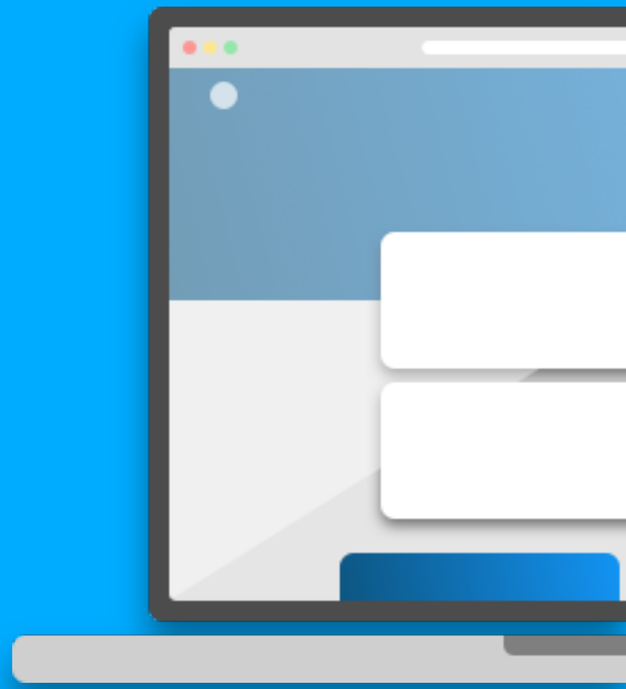
- Control Transformations > Data Transformations > Layout Transformations
- Control Flow Obfuscation combined with strong resilient Opaque predicates is essential
- Preventive Transformations
- Tamper-resistant code takes code protection resilience to the next level
  - Code Traps, Self-defending Code, etc

- Diversity and its role in making it harder to automate code tampering attacks
- Chaining effect of obfuscation transformations
- Code Transformation process

- Code Obfuscation Cost can be managed
- Success in using obfuscation requires searching for good tradeoffs for specific applications

JAVASCRIPT QUIRKS

PART 2

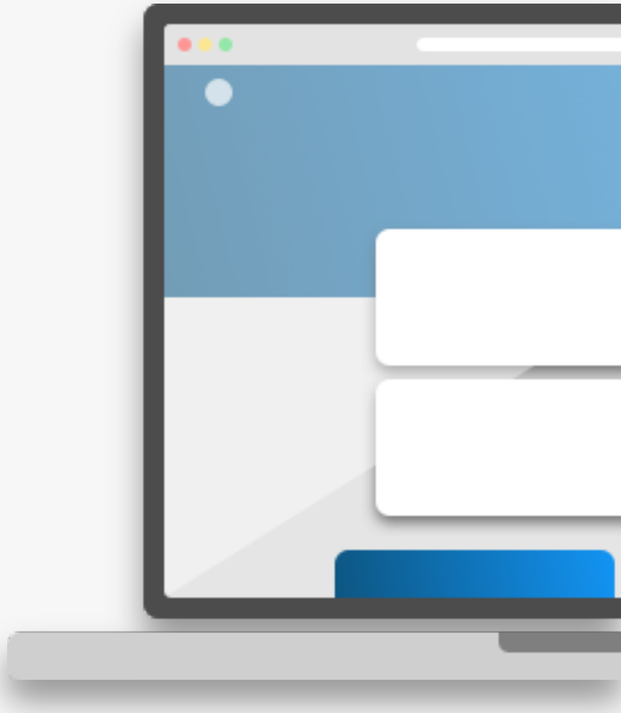


jscrambler

JavaScript **Quirks**

JavaScript has plenty quirks, some of which really unique:

- Subscript Notation
- Type coercion
- Eval functions
- Poisoning native API functions
- Proxies
- Getters, Setters



Subscript **Notation**

- Alternative to dot notation
- Useful to obfuscate object, property, and function names using string obfuscation
- Essential to non-alphanumeric obfuscation

```
// Dot notation
window.document.write("<script>alert('Dont
paste this into your console')</script>");
```

```
// Subscript-bracket notation
window["document"]["write"]
("<script>alert('I told you!')</script>");
```

Type **Coercion**

- When operands have different types
- Useful to obfuscate literals
- With Subscript Notation you can create pretty much any string you want (e.g. Non-alphanumeric JS)

```
![]
>  false

true + false
> 1

4 + “1”
> “41”

“4” + void 0
> “4undefined”

[] + {}
> "[object Object]”
```

```
_ = [] + {};
> "[object Object]"

$ = ![] + !![]
> 1

_[$]
> “o”

_[$+$]+_[$]+_[$+$]
> “bob”

([]+{})[![]+!![]+![]+!![]]+([]+{})[![]+!![]]+([]+{})[![]+!![]+![]+!![]]
> “bob”
```

Eval **Functions**

- Eval functions evaluate (parse & execute) JS code represented as strings
- Encrypted/encoded code
- Telltale indicator

```
// Prompts alert box
eval("alert(1)")

// Similar to eval but using Function constructor
Function("alert(1)")()

// Obscure way to reference Function constructor
[]["filter"]["constructor"]( "alert(1)")()
```

Poisoning **Functions**

- JavaScript “Native” API functions can be modified or completely replaced

```
// Monkey Patch

const _eval = eval;
eval = function (a) {
  console.log('Evaluating: ${a}');
  return _eval.call(this, a);
};
```

```
// Logs the input to the console and
evaluates it next

eval(“1 + 2”);
> Evaluating: 1 + 2
> 3
```

Proxies

- Introduced in ES6
- Simpler way to do MITM-like attacks to JS native functions

```
eval = new Proxy(eval, {
 apply: function (target, _this, args) {
   console.log(`Evaluating: ${args}`);
   return target.apply(_this, args);
 }
});
```

```
// Logs the input to the console and
evaluates it next
eval("1 + 2");
> Evaluating: 1 + 2
> 3
```

Getters / **Setters todo**

- Useful when you want to set dynamic behaviors when getting or setting values to an object's property.
- In the following example we've set all RegExps to evaluate a function name matching the RegExp's source (excludes everything that is not a character from the alphabet).

```
Object.defineProperty(/ /.constructor.prototype, void 0, {
 get() {
   try {
      return eval(this.source);
   } catch (e) {}
 }
});

// r: RegExp
const _ = (r) => (0, r[void 0]);
```

```
_(/alert/)(1) // Prompts alert 1

function a (b) {
  alert(b);
}
// Also prompts alert 1
_(/a/)(1)
```

Getters / **Setters**

- Slightly more sneaky

```
Object.defineProperty(/ /.constructor.prototype, void 0, {
 get() {
   try {
      return eval(this.source.replace(/[^a-zA-Z]/g, ''));
   } catch (e) {}
 }
});

// r: RegExp
const _ = (r) => (0, r[void 0]);
```

```
// And… also prompts alert 1
_(/[a-l]:\/\/e?_r{1,}\t/)(1)
```

CODE DEOBFUSCATION TECHNIQUES

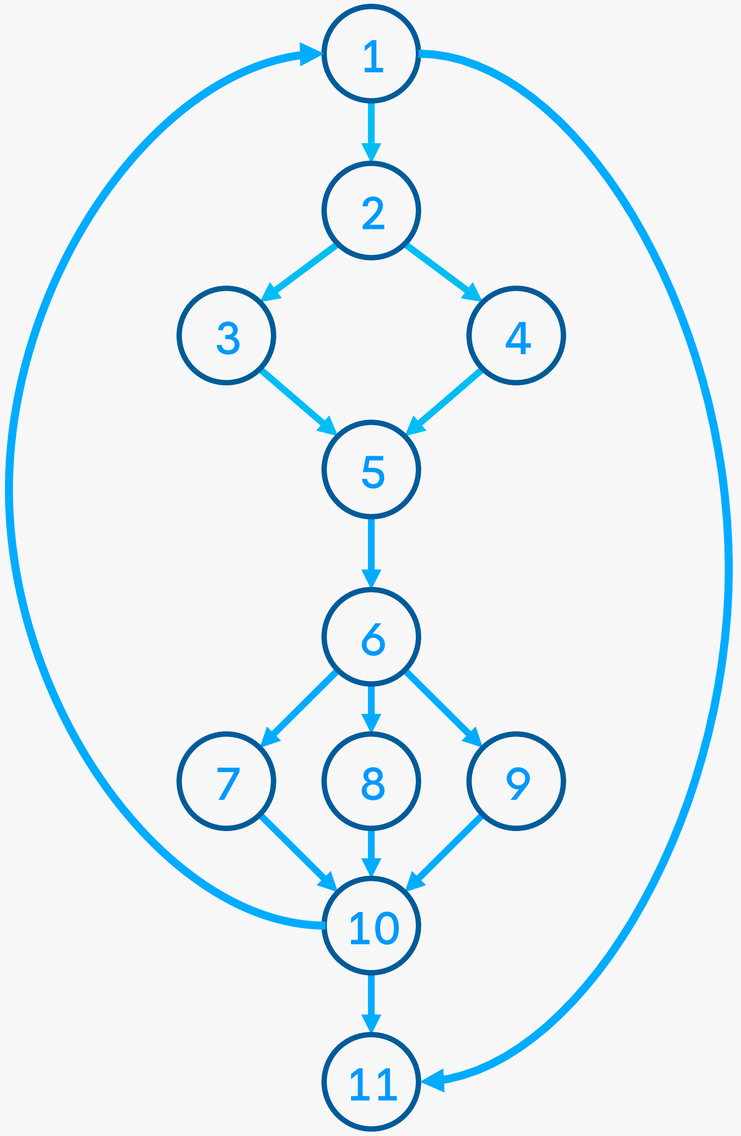
PART 3

jscrambler

Control Flow **Graph**

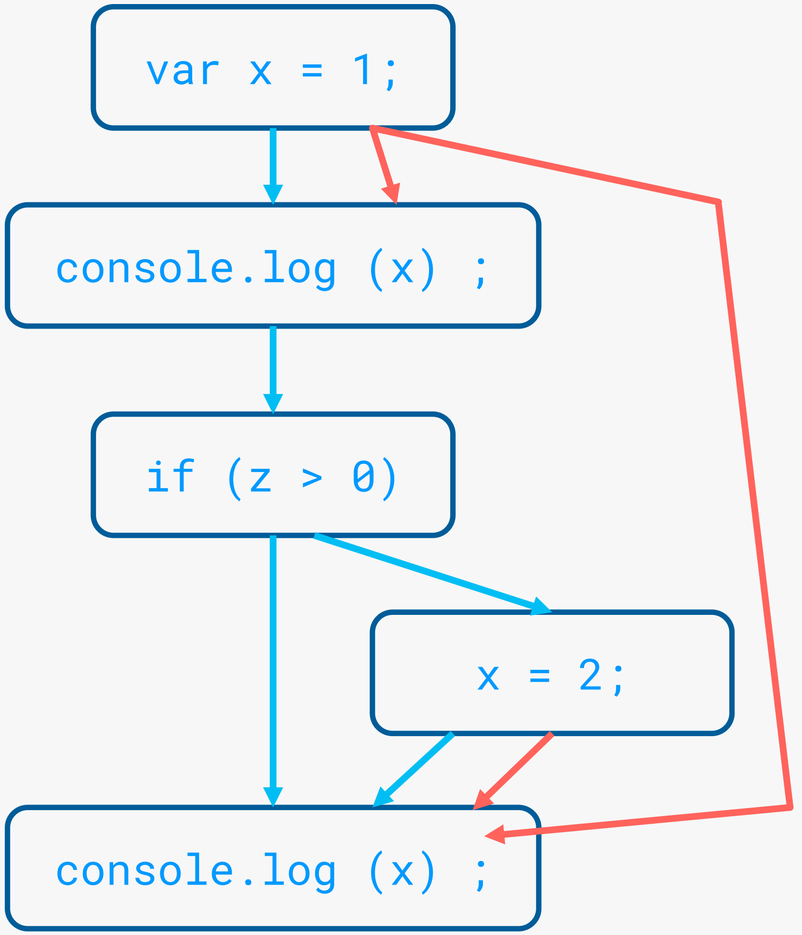
- After getting an AST, a Control Flow Graph (CFG) is produced
- Directed graph where
  - Each node represents a statement
  - Edges represent control flow

```
1) while( x < 100) {
2) if (a[x] % 2 === 0) {
3)     parity = 0;
   } else {
4)     parity = 1;
5) }
6) switch (parity) {
7)  case 0: console.log('even'); break;
8)  case 1: console.log('odd'); break;
9)  default: console.log('error');
   }
10) x++;
  }
11) console.log('finished');
```



Data Flow **Analysis**

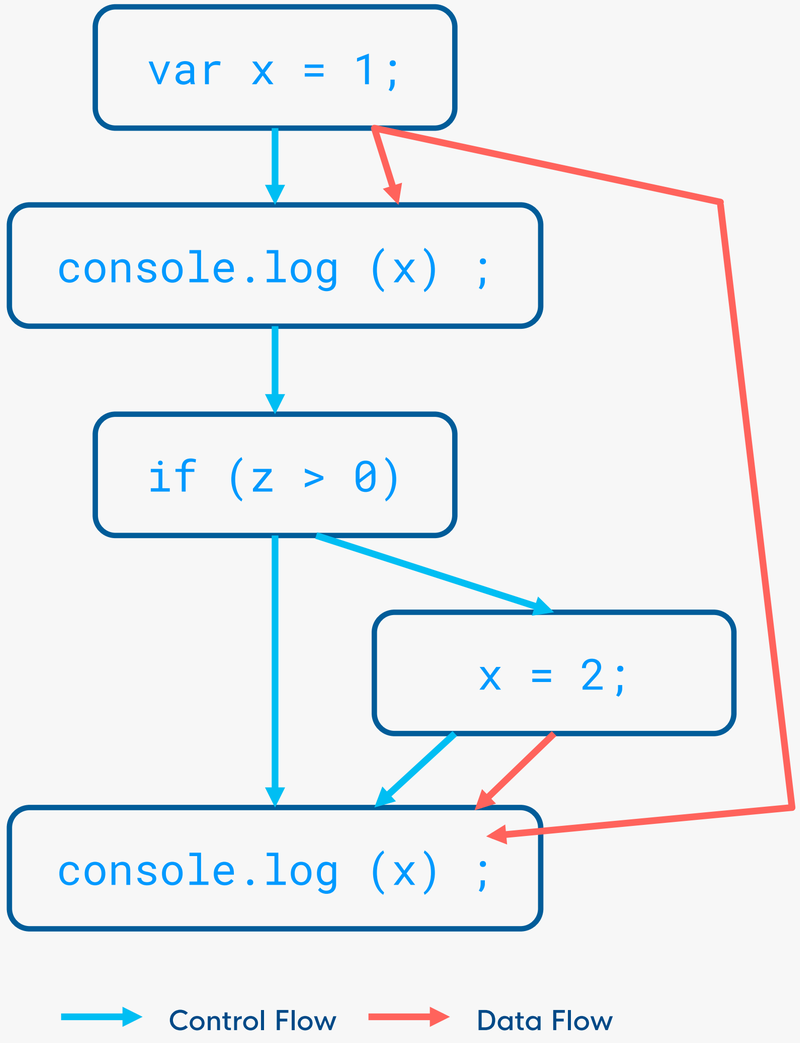
- How data flows through the program
- Operates on top of the CFG
- Determine statements that are unnecessary
- Replace variables with their constant values
- Remove dead code
- Meant for Code Optimization

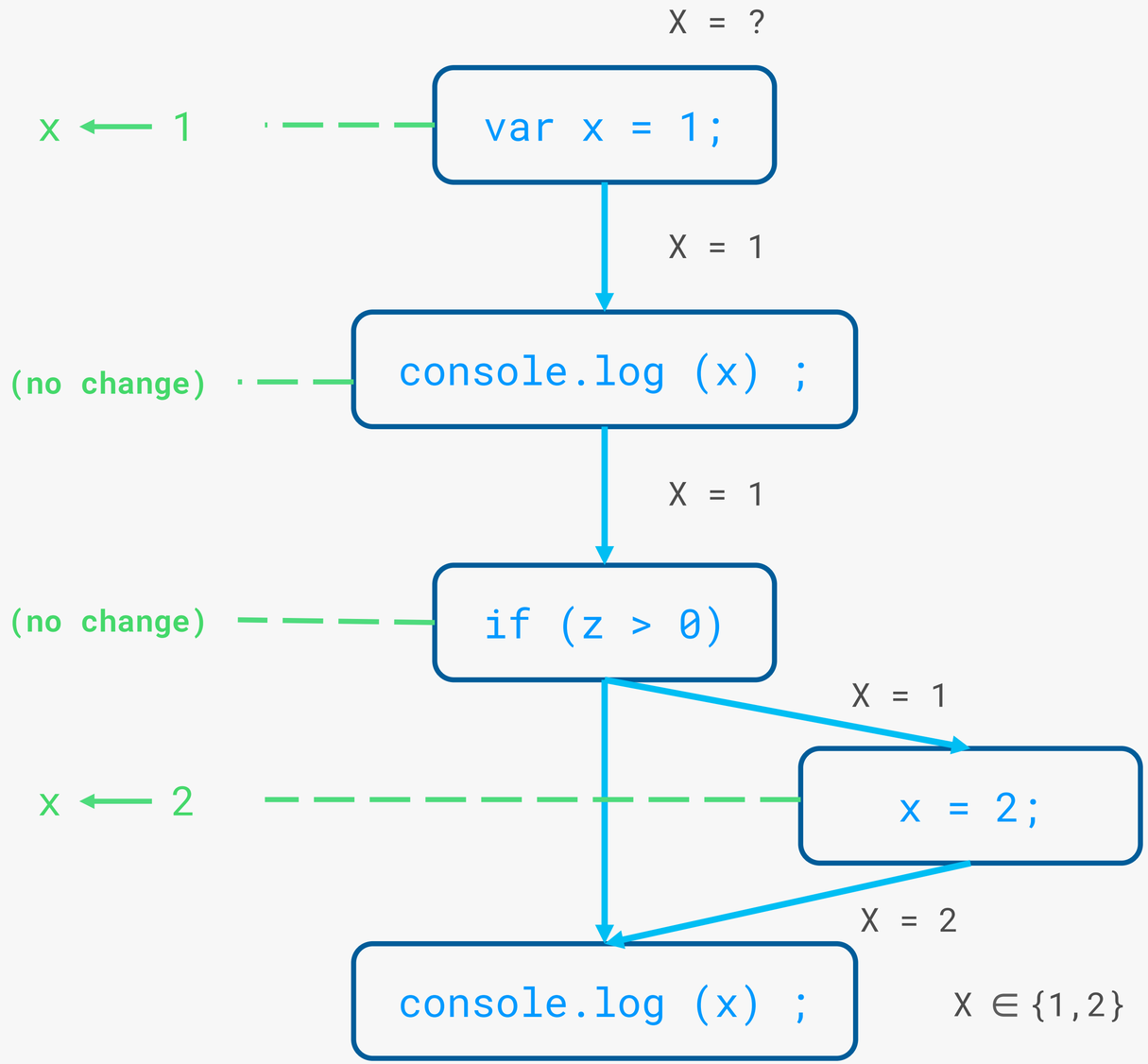


Control Flow

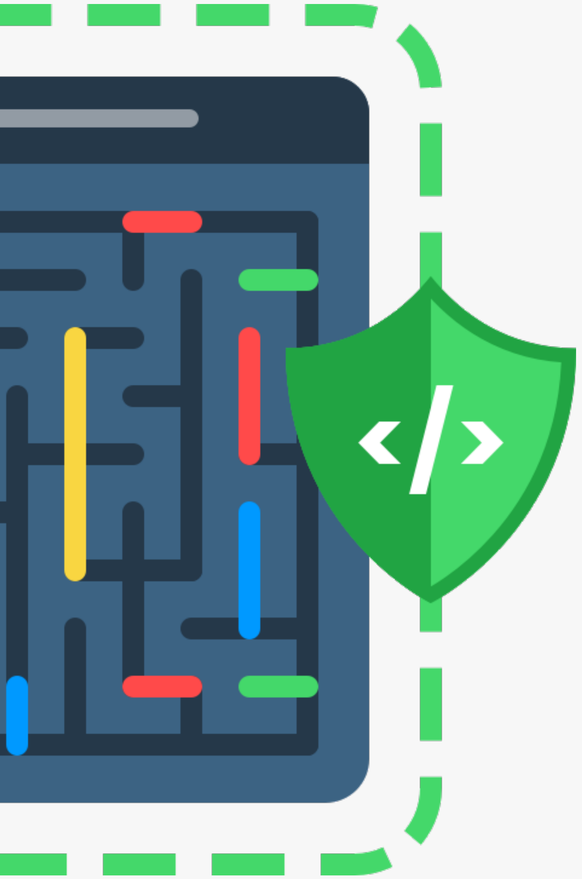
Data Flow

Data Flow Analysis vs **Control Flow Analysis**





Deobfuscation **Tools**

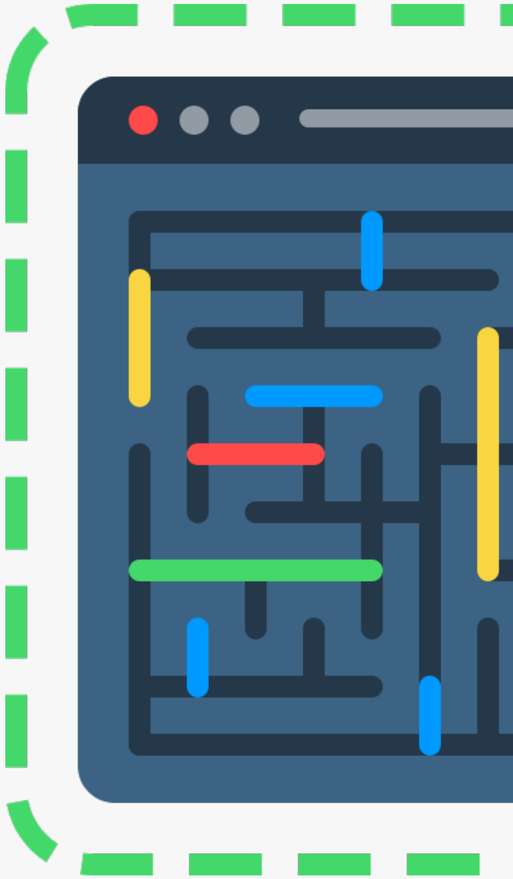


JavaScript Malware Analysis: **JStillery, JSDetox**

JavaScript DeObfuscation: **JStillery, JSDetox, JSNice**

JavaScript Optimization: **Prepack.io, Closure compiler**

JavaScript Engines: **V8, SpiderMonkey, Nodejs's VM module**



Deobfuscation **Techniques**

Static and dynamic code analysis techniques that are being used to deobfuscate obfuscated JavaScript

- **Static analysis**
  - Constant folding, Constant Propagation, Dead code elimination, Symbolic execution, etc
- **Partial Evaluation**
  - Classify expressions as static or dynamic in a program
  - Precompute all static input at compile time (i.e. deobfuscation time)
  - "Residual program" is a faster program (i.e. less computations, not necessarily less code)
- **Program Slicing**
- **Dynamic analysis**
  - Concrete execution using interpreters, Node.js VM module
- **and even Machine Learning and Statistics to predict identifier names and type annotations**
  - Conditional Random Field, Support Vector Machines, etc

Constant Folding **& Constant Propagation**

- Simplify the code statically in compile time
- Constant folding evaluates and simplifies constant expressions in compile time rather than in runtime
- Constant propagation identifies variables that hold constant values and replaces them with the actual value
- Resulting code is smaller and faster

```
var a = 5 + 2;
var b = 5 * (3 + a) - 1;
```

```
var a = 7;
var b = 49;
```

Dead Code **Elimination**

- Static code optimization that removes dead code. If statement `truthy` branch is never executed than it can be removed
- Resulting code is smaller (and has less obfuscation potency)

```
var pF = false;

// Constant propagation will replace pF with false
if (pF) {
 deadcode();
} else {
 hello();
}
```

```
var pF = false;
hello();
```

Concrete **Execution**

- Executes the code to replace it with a simpler form
- Ideally this is done with a sandboxed / restricted / virtualized environment
- Example: obfuscation usually seen in malware to hide the attack

```
function decode (a) {
 return unescape(decodeURIComponent(atob(a)));
}
eval(decode("TWFsaWNpb3VzJTI1MjBjb2Rl"));
```

```
function decode (a) {
  return unescape(decodeURIComponent(atob(a)));
}
eval('Malicious code');
```

```
function decode (a) {
 return unescape(decodeURIComponent(atob(a)));
}
eval(decode("MSs0"));  //1+4
```

```
function decode (a) {
  return unescape(decodeURIComponent(atob(a)));
}
5
```

Concrete **Execution (2)**

Another example

```
// From JStillery
function w(d) {
 var z = 23,
   u = '', c = this,
   g = 'frKvdrCode'.replace('Kvd', 'omCha'),
   f = 'cqJUjCodeAt' ['ZdPce'.replace('ZdP',
'repla')]('qJUj', 'har'),
   k = 'UNmNth' ['reQMgZnace'.replace('QMgZn',
'pl')]('UNmN', 'leng'),
   j = 'SlzbJcg'.replace('lzbJc', 'trin'),
   t = c[j], v = t[g], r, l, s;
 for (s = 0; s < d[k]; s++) {
   r = d[f](s); l = r ^ z; u += v(l);
 }
 return u;
};
w("test")
```

```
function w(d)
/*Scope Closed:false | writes:false*/
{
 var z = 23,
   u = '', c = this,
   g = 'fromCharCode',
   f = 'charCodeAt',
   k = 'length',
   j = 'String',
   t = c.String, v = t.fromCharCode, r, l, s;
 for (s = 0; s < d[k]; s++) {
   r = d[f](s); l = r ^ z; u += t.fromCharCode(l);
 }
 return u;
};
'crdc';
```

Program **Slicing**

From Wikipedia "program slicing is the computation of the set of program statements, the program slice, that may affect the values at some point of interest, referred to as a slicing criterion"

```
var i;
var sum = 0;
var product = 1;
var w = 7;
for(i = 1; i < N; ++i) {
 sum = sum + i + w;
 product = product * i;
}
console.log(sum);
console.log(product);
```

```
var i;
var sum = 0;
var w = 7;
for(i = 1; i < N; ++i) {
 sum = sum + i + w;
}
console.log(sum);
```

```
var i;
var product = 1;
for(i = 1; i < N; ++i) {
 product = product * i;
}
console.log(product);
```

Heap **Serialization**

- Optimization technique (prepack.io)
- Walks the heap in order, generating fresh straightforward JavaScript code that creates and links all objects reachable in the initialized heap

```
var arr = [];
for (var i = 0; i < 10; i++) {
 arr[i] = i * 2;
}
```

```
var i, arr;
arr = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
i = 0;
i = 1;
...
i = 8;
i = 9;
i = 10;
```

Heap **Serialization (2)**

- Another example
- A multi-dimension array with infinite number of subscripts to create opaque predicates and steps.

```
// mda.js

var createMDA = function(len, shift) {
 var mda = [];
 for (var k = 0; k < len; k++) {
   mda[(k + shift) % len] = [];
 }
 for (var i = 0; i < len; i++) {
   for (var j = len - 1; j >= 0; j--) {
     mda[i][(j + shift * i) % len] = mda[j];
   }
 }
 return mda;
};

var mda = createMDA(24, 7);
```

```
// 2_4_prepackio.js

var m;
(function() {
 var _4 =
[, , , , , , , , , , , , , , , , , , , , , , , , , , , , , ];
 _4[0] = _4;
 var _7 = [, , ,
_4, , , , , , , , , , , , , , , , , , , , , , , , , , , , ];
 _4[21] = _7;
 _7[0] = _7;
 var _a = [, , , _7, , ,
_4, , , , , , , , , , , , , , , , , , , , , , , , , , , , ];
 _4[18] = _a;
 ...
 _m[14] = _o;
 _8[23] = _o;
 _n[7] = _o;
 _o[0] = _o;
 m = [_o, _n, _m, _l, _k, _j, _i, _1, _h, _g, _f, _e, _d,
_c, _b, _a, _9, _8, _7, _6, _5, _4, _3, _2];
}());
```

Deobfuscation Tools **& Techniques**

	Constant Folding	Constant Propagation	Dead Code Elimination	Symbolic Execution	Concrete Execution	JS Interpreter / engine / emulation	Machine Learning and Statistics
JStillery						Node VM	
JSDetox							
JSNice							
Prepack.io						Interpreter	
Closure Compiler							
SpiderMonkey						Engine	
V8						Engine	

MEASURING OBFUSCATION RESILIENCE

PART 4

jscrambler

Sample #1: **Minified Code**

Using a minifier to obfuscate does is just wrong!

```
// mda.js

;(function() {
var createMDA = function(len, shift) {
 var mda = [];
 for (var k = 0; k < len; k++) {
   mda[(k + shift) % len] = [];
 }
 for (var i = 0; i < len; i++) {
   for (var j = len - 1; j >= 0; j--) {
     mda[i][(j + shift * i) % len] = mda[j];
   }
 }
 return mda;
};
var mda = createMDA(24, 7);
…
else if (ref === mda[17][21]) {
 console.log("There is no spoon.");
}
}());
```

```
// Minified with UglifyJS2
// 2_4_prepackio.js

!function(){var o=function(o,n){for(var r=[],e=0;o>e;e+
+)r[(e+n)%o]=[]
for(var t=0;o>t;t++)for(var a=o-1;a>=0;a--)r[t][(a+n*t)
%o]=r[a]
return r},n=o(24,7),r=n[15][13][22][4]
r===n[12][2]?console.log("Do not try and bend the
spoon."):r===n[17][21]&&console.log("There is no
spoon.")}()
```

```
// Deobfuscated with Prepack.io
console.log("There is no spoon.”)
```

Sample #2: **JavaScript Compressors**

Don't use JavaScript compressors/ packers either:

```
// Source code

alert(1)
```

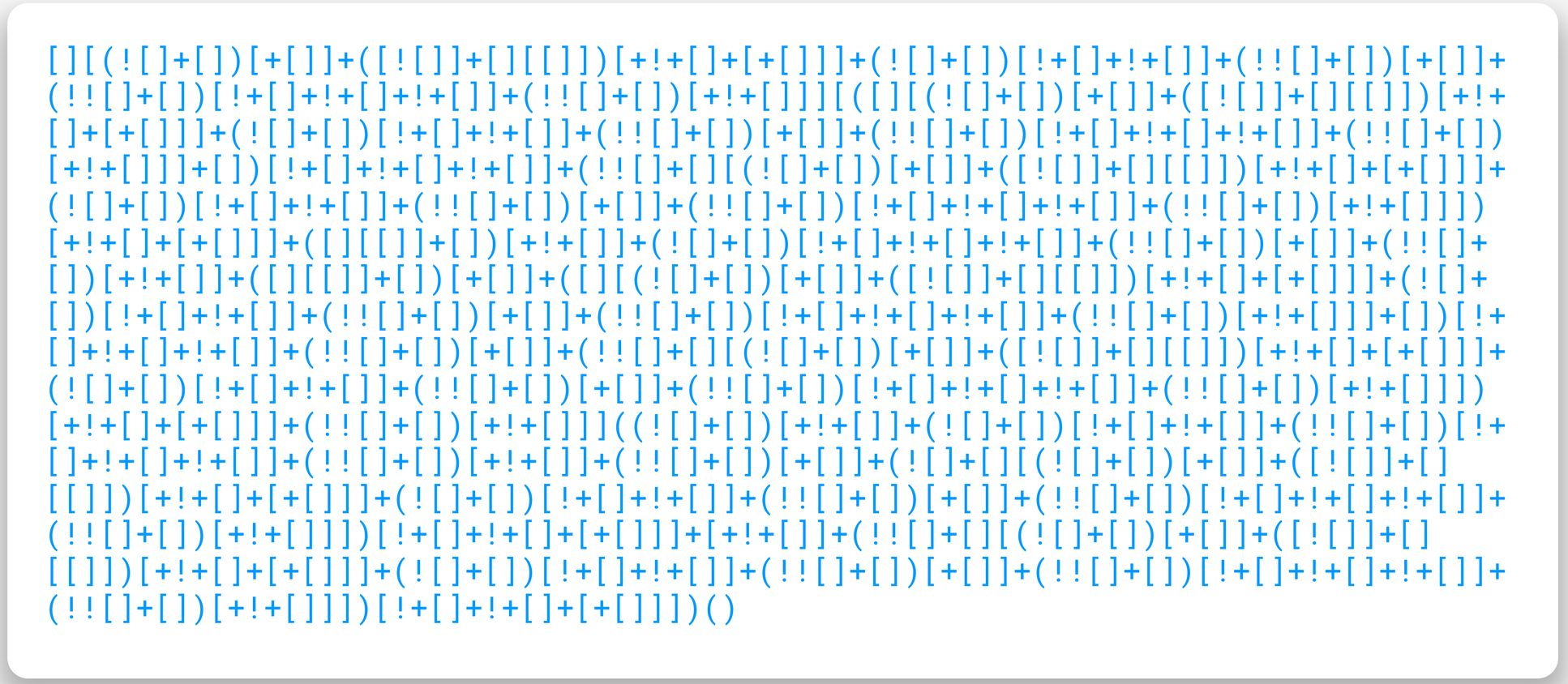
```
// Packer
eval(function(p, a, c, k, e, r) {
 e = String;
 if (!''.replace(/^/, String)) {
   while (c--) r[c] = k[c] || c;
   k = [function(e) {
     return r[e]
   }];
   e = function() {
     return '\\w+'
   };
   c = 1
 };
 while (c--)
   if (k[c]) p = p.replace(new RegExp('\\b' + e(c)
+ '\\b', 'g'), k[c]);
 return p
}('0(1)', 2, 2, 'alert|'.split('|'), 0, {}))
```

```
// Deofuscated with jsbeautifier.org

alert(1)
```

Sample #3: **JSFuck (Non-alphanumeric obfuscation)**

"JSFuck is an esoteric and educational programming style based on the atomic parts of JavaScript. It uses only six different characters to write and execute code." **- jsfuck.com**

```
// Source code

alert(1)
```



How is **that possible?**

Using type coercion and browser quirks

We can obtain alphanumeric characters indirectly

How do we get a 0 without using alphanumerics?

```
+[]              > 0
+!+[]            > 1
+!+[]+!+[]       > 2
+"1"             > 1
""+1 = "1"
```

It's easy to get any number
Type coercion to number
Type coercion to string

How do we get letters?

```
+"a"             > NaN
+"a"+""          > "NaN"
(+"a"+"")[0]     > "N"
```

Ok, but now without alphanumerics:

```
(+"a"+"")[+[]]          > "N"
```

How do we get an "a" ?

```
![]                              > false
![]+""                           > "false"
(![]+"")[1]                      > "a"
(![]+"")[+!+[]]                  > "a"
(+(![]+"")[+!+[]]+"")[+[]]       > "N"

eval( (![]+"")[+!+[]]+"lert(1)");
```

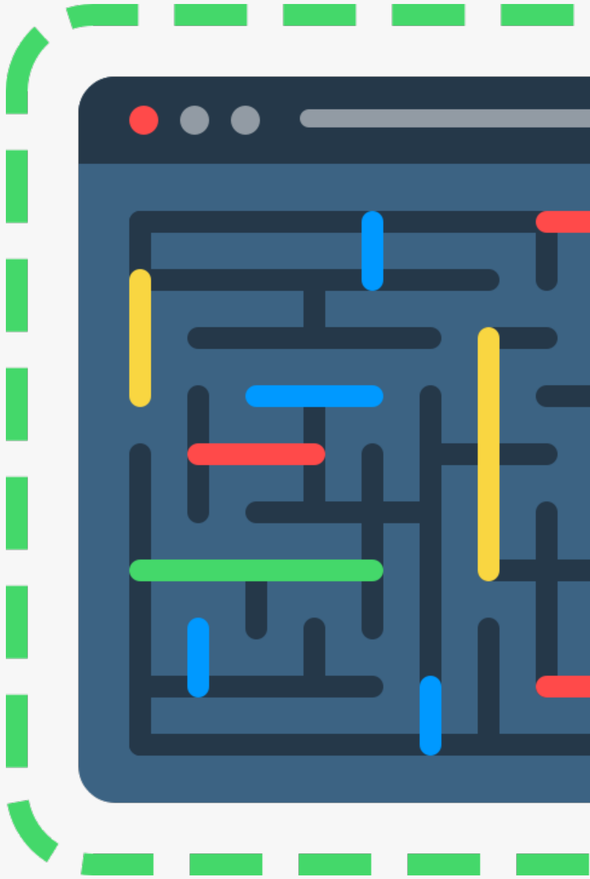
```
'a':   '("false")[1]',
'b':   '(GLOBAL+[])[2]',
'c':   '([]["filter"]+[])[3]',
'd':   '("undefined")[2]',
'e':   '("true")[3]',
'f':   '("false")[0]',
'g':   '([]+String)[14]',
'h':   '(+(17))["toString"](18)',
'i':   '("undefined")[5]',
'j':   '(GLOBAL+"")[3]',
'k':   '(+(20))["toString"](21)',
'l':   '("false")[2]',
'm':   '(Number+"")[11]',
'n':   '("undefined")[1]',
'o':   '([]["filter"]+"")[6]',
'p':   '(+(25))["toString"](30)',
'q':   '(+(26))["toString"](30)',
'r':   '("true")[1]',
's':   '("false")[3]',
't':   '("true")[0]',
'u':   '("undefined")[0]',
'v':   '(+(31))["toString"](32)',
'w':   '(+(32))["toString"](33)',
'x':   '(+(33))["toString"](34)',
'y':   '("Infinity")[7]',
'z':   '(+(35))["toString"](36)',
```

```
'A':   '(Array+"")[9]',
'B':   '(Boolean+"")[9]',
'C':   'GLOBAL["unescape"]("%"+(43)+"c")[0]',
'D':   USE_CHAR_CODE,
'E':   USE_CHAR_CODE,
'F':   '(Function+"")[9]',
'G':   USE_CHAR_CODE,
'H':   USE_CHAR_CODE,
'I':   '("Infinity")[0]',
'J':   USE_CHAR_CODE,
'K':   USE_CHAR_CODE,
'L':   USE_CHAR_CODE,
'M':   USE_CHAR_CODE,
'N':   '("NaN")[0]',
'O':   USE_CHAR_CODE,
'P':   USE_CHAR_CODE,
'Q':   USE_CHAR_CODE,
'R':   USE_CHAR_CODE,
'S':   '(String+[])[9]',
'T':   USE_CHAR_CODE,
'U':   USE_CHAR_CODE,
'V':   USE_CHAR_CODE,
'W':   USE_CHAR_CODE,
'X':   USE_CHAR_CODE,
'Y':   USE_CHAR_CODE,
'Z':   USE_CHAR_CODE,
```

```
' ':   '([]["filter"]+[])[8]',
'!':   USE_CHAR_CODE,
'"':   '("")["link"]()[8]',
'#':   USE_CHAR_CODE,
'$':   USE_CHAR_CODE,
'%':   'GLOBAL["escape"]("<")[0]',
'&':   USE_CHAR_CODE,
'\'':  'GLOBAL["unescape"]("%"+(27)+"c")[0]',
'(':   '([]["filter"]+"")[15]',
')':   '([]["filter"]+"")[16]',
'*':   USE_CHAR_CODE,
'+':   USE_CHAR_CODE,
',':   USE_CHAR_CODE,
'-':   USE_CHAR_CODE,
'.':   USE_CHAR_CODE,
'/':   '("")["sub"]()[6]',
':':   'GLOBAL["Date"]()[21]',
';':   USE_CHAR_CODE,
'<':   '("")["sub"]()[0]',
'=':   '("")["fontcolor"]()[11]',
'>':   '("")["sub"]()[10]',
'?':   USE_CHAR_CODE,
'@':   USE_CHAR_CODE,
'[':   USE_CHAR_CODE,
'\\':  'GLOBAL["unescape"]("%"+(5)+"c")[0]',
']':   USE_CHAR_CODE,
'^':   USE_CHAR_CODE,
'_':   USE_CHAR_CODE,
'`':   USE_CHAR_CODE,
'{':   '([]["filter"]+"")[18]',
'|':   USE_CHAR_CODE,
```

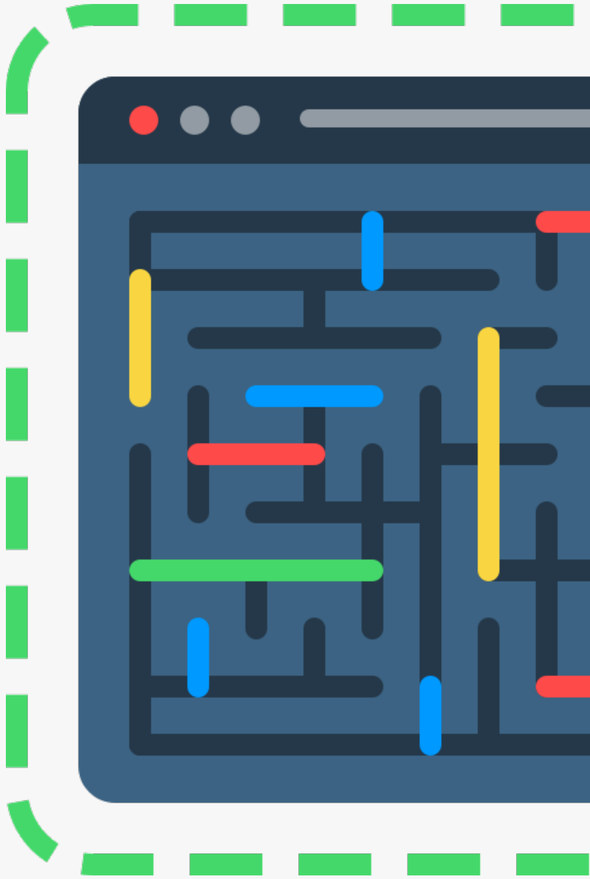
Wait... **What about the eval?**

- **eval() is not the only way to eval() !**
- **You have 4 or 5 methods more**
- **Examples**
  - `Function("alert(1)")()`
  - `Array.constructor(alert(1))()`
  - `[]["sort"]["constructor"]("alert(1)")()`
    - Subscript notation
    - Strings !



Non alphanumeric **Obfuscation**

- **100% potent**
- **0% stealthy**
- **High execution cost**
  - eval is slower
  - File is much larger => slower loading times
- **May not work in all browsers**
- **What about resilience ?**
  - Unfortunately, none!



Sample #3: **JSFuck (2)**

// Source code

```
alert(1)
```

```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+
(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+[][[]])[+!+
[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])
[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+
(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])
[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+
[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+
[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+
[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+
(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])
[+!+[]+[+[]]]+(!![]+[])[+!+[]]]((![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])[!+
[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+
(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])
[[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+
(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]]+[+!+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])
[[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+
(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]])()
```

// Deobfuscated with Prepack

```
ReferenceError: alert is not defined
```

// Deobfuscated with JStillery

```
[].filter.constructor('alert(1)')();
```

Sample #4: **Javascript2img**

```
// Source code
alert(1)
```

```
vc167ceaa2357dd0f98caf9a7514f1ea5=[function(v7c646b50cc88f596bb622e66bb6a6617)
{return"0159a99ed28b0581890608d24ada9decc48741970ca1f991bf30dd786c7d46f3c4c6bd7e"},function(v7c646b50cc88f
596bb622e66bb6a6617){return
v4a30630c14033738ffde5e5aee6844aa.createElement(v7c646b50cc88f596bb622e66bb6a6617)},function(v7c646b50cc88
f596bb622e66bb6a6617){return
v7c646b50cc88f596bb622e66bb6a6617[0].getContext(v7c646b50cc88f596bb622e66bb6a6617[1])},function(v7c646b50c
c88f596bb622e66bb6a6617){return
v7c646b50cc88f596bb622e66bb6a6617[0].text=v7c646b50cc88f596bb622e66bb6a6617[1]},function(v7c646b50cc88f596
bb622e66bb6a6617){return null},function(v7c646b50cc88f596bb622e66bb6a6617)
{"ff1eb8bd6cb17940ab78c0eeecf66268772f206117131af045e227576bc98bfec16f75d2"},function(v7c646b50cc88f596bb6
22e66bb6a6617)
{return"8d8ebea7b076c177ecd21e2d0d7e52fa74c48f3c0df27f78a9339990332cf02c05677579"},function(v7c646b50cc88f
596bb622e66bb6a6617){v7c646b50cc88f596bb622e66bb6a6617.style.display="none";return
d1794f6c2a974ee78c23dbf=vc167ceaa2357dd0f98caf9a7514f1ea5[4]
(v24fcc9be09909ff7ccd7c146dfa2d493);v1341746e3d58a8c0d7ce43b877b6beb1=vc167ceaa2357dd0f98caf9a7514f1ea5[4]
(v24fcc9be09909ff7ccd7c146dfa2d493);v7c646b50cc88f596bb622e66bb6a6617=vc167ceaa2357dd0f98caf9a7514f1ea5[4]
(v24fcc9be09909ff7ccd7c146dfa2d493);v7c646b50cc88f596bb622e66bb6a6617=vc167ceaa2357dd0f98caf9a7514f1ea5[4]
(v76b299dd277bfb2f877ea6683dd95905);vc167ceaa2357dd0f98caf9a7514f1ea5=vc167ceaa2357dd0f98caf9a7514f1ea5[4]
(v24fcc9be09909ff7ccd7c146dfa2d493)});v44d1c5a5e59e331a1d7d5855d5314fe1=vc167ceaa2357dd0f98caf9a7514f1ea5[
9](vc167ceaa2357dd0f98caf9a7514f1ea5[11](7)()+vbca103416d1794f6c2a974ee78c23dbf[9]);
```

```
// Deofuscated with PoisonJS
alert(1)

// continues...
```

Sample #4: **Javascript2img (2)**

- Each line is the result of a monkey patched function that has executed and logged to the console
- The last line is the input code, everything else is boilerplate added by the obfuscator

```
Log: return unescape(decodeURIComponent(window.atob(v7c646b50cc88f596bb622e66bb6a6617)))
Log: return document
Log: return v4a30630c14033738ffde5e5aee6844aa.getElementById(v7c646b50cc88f596bb622e66bb6a6617);
Log: return new Image();
Log: return '';
Log: return 'canvas';
Log: return 'none';
Log: return '2d';
Log: return String.fromCharCode(v7c646b50cc88f596bb622e66bb6a6617);
Log: for (ve324453514c7a3860e25f62a14b2a43a = v1341746e3d58a8c0d7ce43b877b6beb1[2]; ve324453514c7a3860e25f62a14b2a43a <
v624188683b10d830edc64001e2ffd806.data.length; ve324453514c7a3860e25f62a14b2a43a += 4) vbe62785667f315dd97269f898bad0fe6
+= (v624188683b10d830edc64001e2ffd806.data[ve324453514c7a3860e25f62a14b2a43a] != v1341746e3d58a8c0d7ce43b877b6beb1[1]) ?
v28cde49dfe1e98499bf428e006ab8f11(v624188683b10d830edc64001e2ffd806.data[ve324453514c7a3860e25f62a14b2a43a]) :
vbca103416d1794f6c2a974ee78c23dbf[4];
vbe62785667f315dd97269f898bad0fe6 = vbe62785667f315dd97269f898bad0fe6.trim();
Log: alert(1)
```

Control Flow **Obfuscation**

Control-flow obfuscation techniques can have a hard time deterring interpreters that do control-flow analysis and are able to remove dead code and control-flow obfuscation overhead

```
// mda.js
;(function() {
var createMDA = function(len, shift) {
 var mda = [];
 for (var k = 0; k < len; k++) {
   mda[(k + shift) % len] = [];
 }
 for (var i = 0; i < len; i++) {
   for (var j = len - 1; j >= 0; j--) {
     mda[i][(j + shift * i) % len] = mda[j];
   }
 }
 return mda;
};
var mda = createMDA(24, 7);
…
else if (ref === mda[17][21]) {
 console.log("There is no spoon.");
}
}());
```

```
// 5_2_cfo.js
… var createMDA = function (len, shift) {
    var o = 2;
    while (o !== 10) {
      switch (o) {
      case 13:
        j--;
        o = 6;
        break;
      case 14:
        mda[i][(j + shift * i) % len] = mda[j];
        o = 13;
        break;
      case 6:
        o = j >= 0 ? 14 : 12;
        break;
      case 9:
        …
```

Control Flow **Obfuscation (2)**

Prepack.io was able to de-obfuscate this. JStillery wasn't.

// 5_2_jstillery.js

```
…
var createMDA = function (len, shift) {
    var o = 2;
    while (o !== 10) {
      switch (o) {
      case 13:
         j--;
         o = 6;
         break;
      case 14:
         mda[i][(j + shift * i) % len] = mda[j];
         o = 13;
         break;
      case 6:
         o = j >= 0 ? 14 : 12;
         break;
      case 9:
         …
```

// 5_2_prepackio.js

```
var O3ffff;
O3ffff = 2;
console.log("There is no spoon.");
O3ffff = 1;
```

Deter Deobfuscation **Techniques**

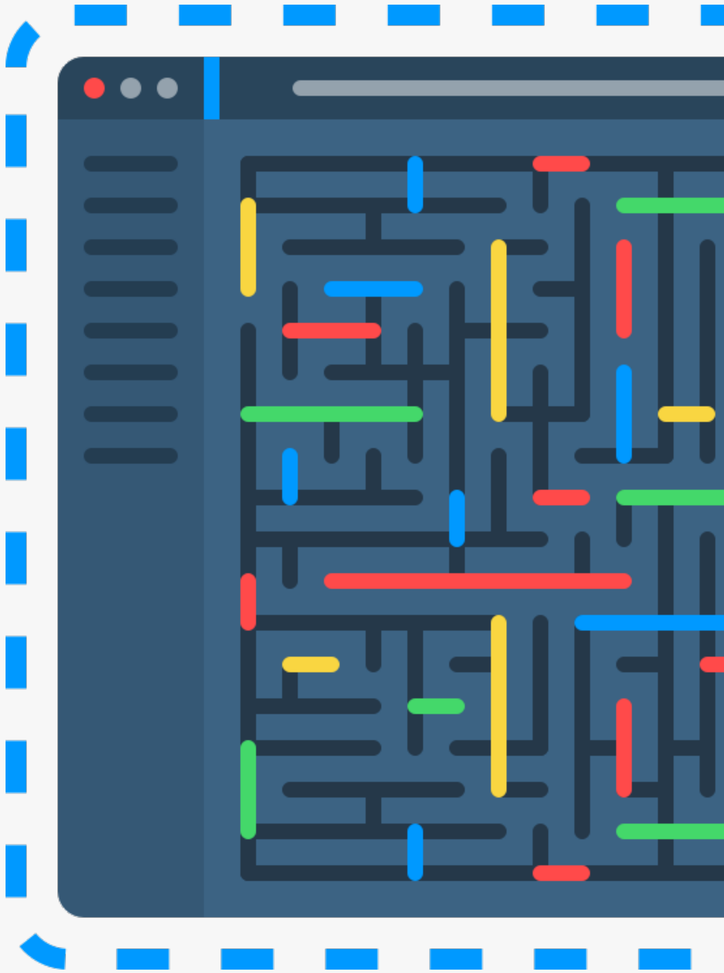
In Deobfuscation Time:

Concrete Execution/Partial Evaluation

- Detect debugging and limited environments
- Detect code tampering

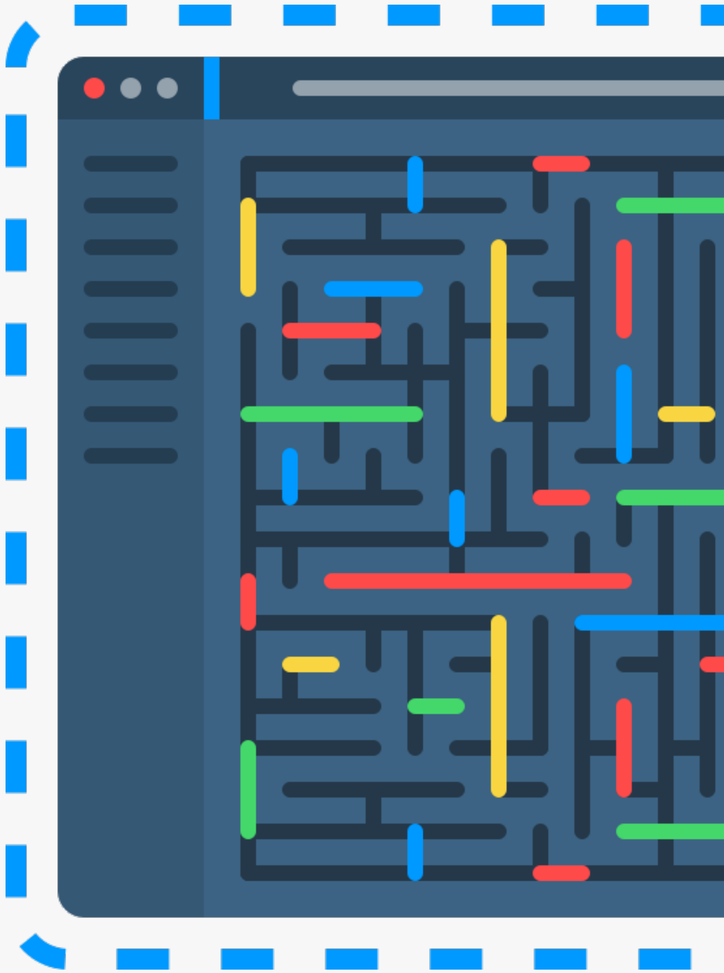
Statically

- Use global or interprocedural dependencies (instead of local)
- Use different reference aliases to the same objects/values
- Create a lot of variables to make Data Flow Analysis more expensive



Detect Interpreters / **Emulators**

- **Check tampering and make the control-flow depend on it**
- **Detect limited environments: no DOM, no WebGL, Nodejs's VM module**
    - Just exit
    - Return alternative / dead code
    - Keep process busy / drain resources



Detect Interpreters / **Emulators (2)**

Detect emulators and interpreters:

- DOM objects and properties: `document.location`, `navigator.plugins`
- WebGL computations: make the next execution depend on result of a WebGL computation

```
// Enumerate functions, objects, and properties
available in `this`

for (var i in this) {}
```

```
// Document object is not present in prepack.io

var i;
i = "self";
i = "window";
i = "setTimeout";
i = "clearTimeout";
i = "setInterval";
i = "clearInterval";
i = "i";
```

Detect Interpreters / **Emulators (3)**

Simple examples to explain how to detect interpreters/emulators

```
(function() {
var b;

// Look for document in `this`
for (var i in this) {
   if (i === 'document') b = this[i];
}

 if (b + '' !== '[object HTMLDocument]') {
   return 'Emulator detected.';
 }

 return 'Running on a Browser';
}());
```

```
// Keep process busy just draining resources
(function fn () {
if (detected) {
   while (true) {
      setTimeout(fn, 0);
   }
}
 // never reaches this part of the code
}());
```

Detect **Tampering**

Detect code tampering in run time using JavaScript introspection capabilities

Use hash digest to validate integrity

- Not necessarily embedded in the source code
- Can be computed in run time and used to decipher the next function to call
- Next function call depends on the previous hash digest
- Only has access to the function name/reference if no tampering has been detected

Detect minification or beautifiers

- Detecting unnecessary white spaces and newlines
- Detecting minifiers' renaming using negative pattern matching
- Whitespaces inside functions are used to produce data used by the application

Runtime Application **Self Protection**

This time let's try to combine advanced obfuscation with RASP.

code

```
// 8_rasp.js
```

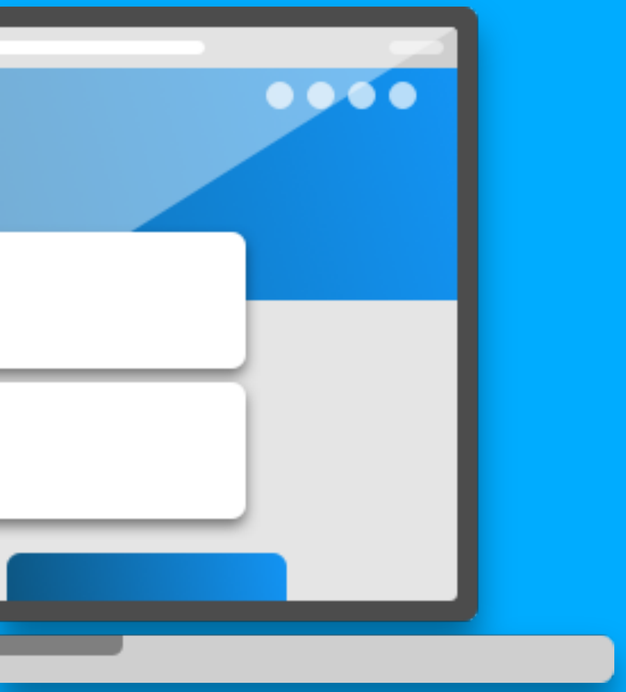
runtime

```
// interpreter is just stuck running
```

Runtime Application **Self Protection (2)**

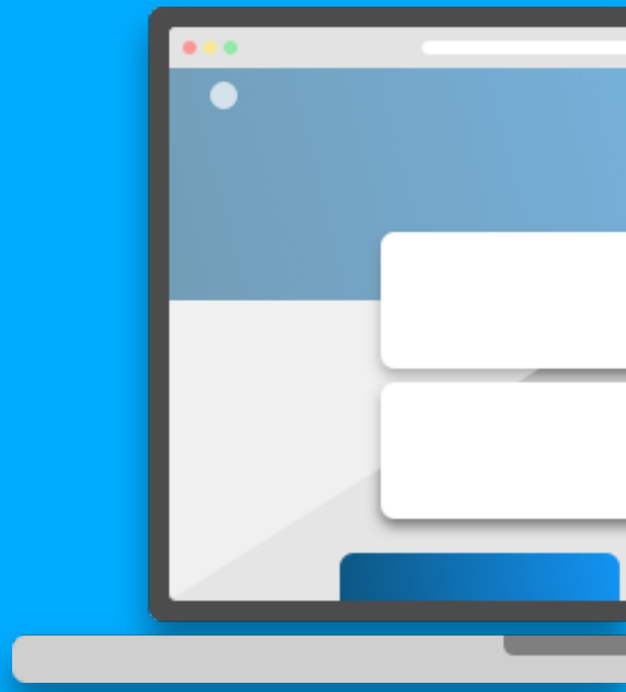
Additional logic is added to the application so it is able to protect itself

- **Detect limited environments**
- **Detect debuggers and tampering made by optimization/human**
- **Block the application to a specific environment: Node, Browser, Domain**
- **Run countermeasures**
  - Just exit
  - Return dead code
  - Redirect to another path
  - Burn resources
  - Redirect to another site
  - Send a message to a server



TESTING & COMPLIANCE

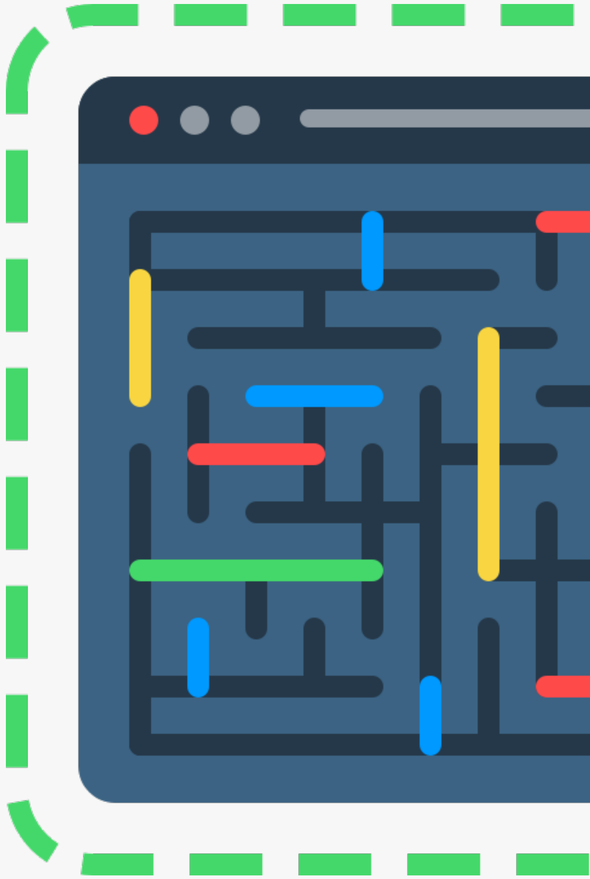
PART 5

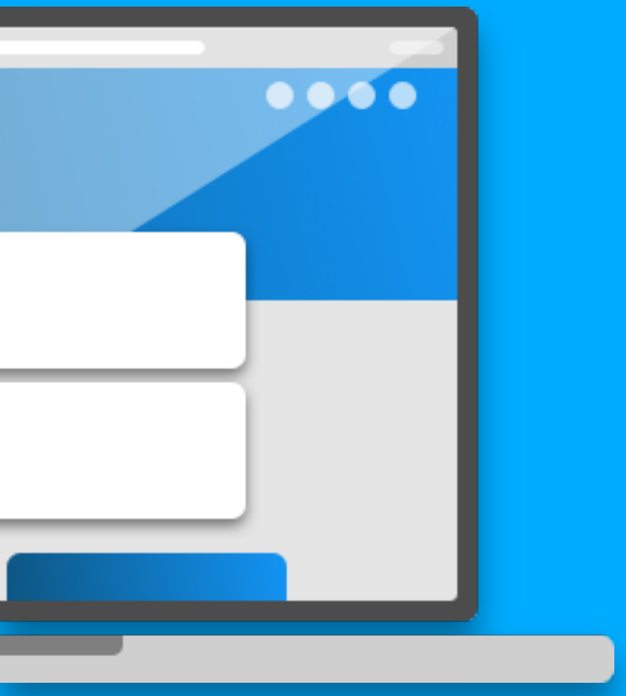


jscrambler

Functional, Performance, **and Compatibility Testing**

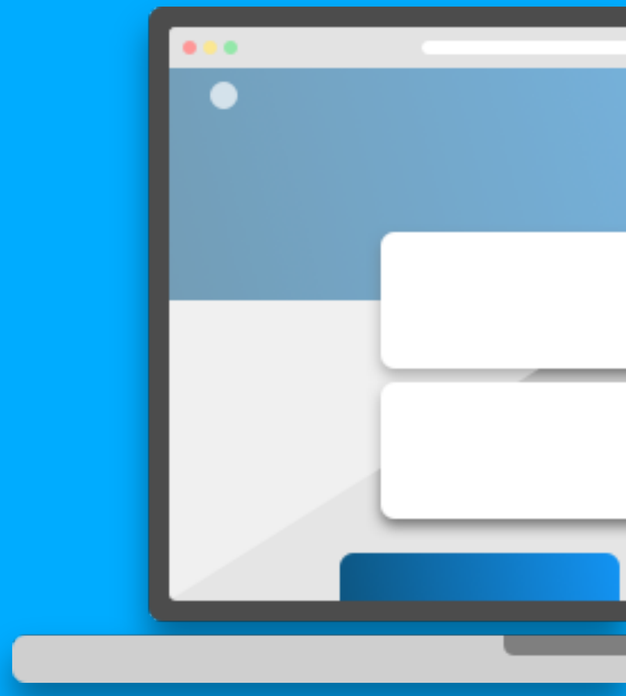
- **Language specifications (ES5, ES6, ES7)**
- **All Browsers (and versions)**
- **JavaScript Frameworks and Libraries**
  - Run framework/library/application unit tests
  - Transform framework/library/application
  - Run transformed framework/library/application unit tests
  - Unit tests results must match
- **Run-time performance (FPS, Dropped Frames, CPU, RAM)**
- **Energy consumption on mobile devices**
- **File size growth**





CONCLUSIONS

PART 6



jscrambler

Conclusions

JS features & quirks are exceptionally useful to create obscurity

… as well as to deobfuscate code

JavaScript deobfuscation tools got pretty sophisticated and can revert most obfuscation (and minification/compression) tools you can find

But advanced obfuscation together with preventive transformations (e.g. anti-tampering) can withstand both static and dynamic reverse engineering attacks

Conclusions Continued

The goal is to

Prevent deobfuscaton tools from being effective

Precluding the automation of reverse engineering attacks

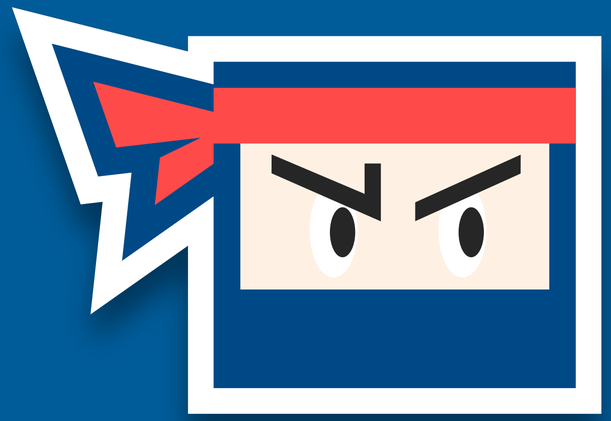
Force the human to fallback to manual reverse engineering

Professional obfuscation tool doesn't care only about obscurity

Resilience, cost, widespread compliance are key to a successful hardened code

Keeping up to date with latest JS specs, browser versions, JS libraries, etc

Keeping up to date with automated deobfuscation tools



NINJA CHALLENGE

https://ninjachallenge.jscrambler.com

Difficulty Level: **NOVICE**

THANK YOU!

@pedrofortuna

jscrambler