josdejong / **mathjs** Public

Code | Issues 134 | Pull requests 45 | Discussions | Actions | Projects

⚠ This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.

Commit 513ab2a

josdejong committed 3 weeks ago · ✓ 8 / 8

```
fix: improve the internal setSafeProperty to not allow setting properties other than numeric indices or length on arrays
```

1 parent 24d5ee7 commit 513ab2a

4 files changed +195 -89

Top

Filter files…

- src/utils
  - customs.js
  - map.js
- test/unit-tests
  - expression
    - security.test.js
  - utils
    - customs.test.js

Search within code

src/utils/customs.js

```
@@ -1,79 +1,83 @@
import { hasOwnProperty } from './object.js'

/**
- * Get a property of a plain object
+ * Get a property of a plain object or array
 * Throws an error in case the object is not a plain object or the
 * property is not defined on the object itself
 * @param {Object} object
 * @param {string} prop
 * @return {*} Returns the property value when safe
 */
function getSafeProperty (object, prop) {
  // only allow getting safe properties of a plain object
  if (isSafeProperty(object, prop)) {
export function getSafeProperty (object, prop) {
  if (isSafeObjectProperty(object, prop) || isSafeArrayProperty(object, prop))
{
    return object[prop]
  }

  if (typeof object[prop] === 'function' && isSafeMethod(object, prop)) {
    throw new Error('Cannot access method "' + prop + '" as a property')
  if (isSafeMethod(object, prop)) {
    throw new Error(`Cannot access method "${prop}" as a property`)
  }

  if (object === null || object === undefined) {
    throw new TypeError(`Cannot access property "${prop}": object is
${object}`)
  }

  throw new Error('No access to property "' + prop + '"')
}

/**
- * Set a property on a plain object.
+ * Set a property on a plain object or array.
 * Throws an error in case the object is not a plain object or the
 * property would override an inherited property like .constructor or .toString
```

```
28  31      * @param {Object} object
29  32      * @param {string} prop
30  33      * @param {*} value
31  34      * @return {*} Returns the value
32  35      */
33      - // TODO: merge this function into access.js?
34      - function setSafeProperty (object, prop, value) {
35      -   // only allow setting safe properties of a plain object
36      -   if (isSafeProperty(object, prop)) {
    36  + export function setSafeProperty (object, prop, value) {
    37  +   if (isSafeObjectProperty(object, prop) || isSafeArrayProperty(object, prop))
        {
37  38        object[prop] = value
38  39        return value
39  40      }
40  41
41      -   throw new Error('No access to property "' + prop + '"')
    42  +   throw new Error(`No access to property "${prop}"`)
42  43    }
43  44
44  45    /**
45      -   * Test whether a property is safe to use on an object or Array.
46      -   * For example .toString and .constructor are not safe
47      -   * @param {Object | Array} object
    46  +   * Test whether a property is safe for reading and writing on an object
    47  +   * For example .constructor and .__proto__ are not safe
    48  +   * @param {Object} object
48  49      * @param {string} prop
49  50      * @return {boolean} Returns true when safe
50  51      */
51      - function isSafeProperty (object, prop) {
52      -   if (!isPlainObject(object) && !Array.isArray(object)) {
    52  + export function isSafeObjectProperty (object, prop) {
    53  +   if (!isPlainObject(object)) {
53  54        return false
54  55      }
55      -   // SAFE: whitelisted
56      -   // e.g length
57      -   if (hasOwnProperty(safeNativeProperties, prop)) {
58      -     return true
```

```
59        -   }
60        -   // UNSAFE: inherited from Object prototype
61        -   // e.g constructor
62        -   if (prop in Object.prototype) {
63        -     // 'in' is used instead of hasOwnProperty for nodejs v0.10
64        -     // which is inconsistent on root prototypes. It is safe
65        -     // here because Object.prototype is a root object
66        -     return false
67        -   }
68        -   // UNSAFE: inherited from Function prototype
69        -   // e.g call, apply
70        -   if (prop in Function.prototype) {
71        -     // 'in' is used instead of hasOwnProperty for nodejs v0.10
72        -     // which is inconsistent on root prototypes. It is safe
73        -     // here because Function.prototype is a root object
      56  +
      57  +   return !(prop in Object.prototype)
      58  + }
      59  +
      60  + /**
      61  +  * Test whether a property is safe for reading and writing on an Array
      62  +  * For example .__proto__ and .constructor are not safe
      63  +  * @param {unknown} array
      64  +  * @param {string | number} prop
      65  +  * @return {boolean} Returns true when safe
      66  +  */
      67  + export function isSafeArrayProperty (array, prop) {
      68  +   if (!Array.isArray(array)) {
74    69          return false
75    70        }
76        -   return true
      71  +
      72  +   return (
      73  +     typeof prop === 'number' ||
      74  +     (typeof prop === 'string' && isInteger(prop)) ||
      75  +     prop === 'length'
      76  +   )
      77  + }
      78  +
      79  + function isInteger (prop) {
```

```
                80 +   return /^\d+$/.test(prop)
77              81   }
78              82
79              83   /**
                     @@ -83,7 +87,7 @@ function isSafeProperty (object, prop) {
83              87    * @param {string} method
84              88    * @return {function} Returns the method when valid
85              89    */
86                 - function getSafeMethod (object, method) {
                90 + export function getSafeMethod (object, method) {
87              91     if (!isSafeMethod(object, method)) {
88              92       throw new Error('No access to method "' + method + '"')
89              93     }
                     @@ -98,22 +102,32 @@ function getSafeMethod (object, method) {
98             102    * @param {string} method
99             103    * @return {boolean} Returns true when safe, false otherwise
100            104    */
101                - function isSafeMethod (object, method) {
102                -   if (object === null || object === undefined || typeof object[method] !==
                   'function') {
               105 + export function isSafeMethod (object, method) {
               106 +   if (
               107 +     object === null ||
               108 +     object === undefined ||
               109 +     typeof object[method] !== 'function'
               110 +   ) {
103            111       return false
104            112     }
               113 +
105            114     // UNSAFE: ghosted
106            115     // e.g overridden toString
107            116     // Note that IE10 doesn't support __proto__ and we can't do this check there.
108                -   if (hasOwnProperty(object, method) &&
109                -       (Object.getPrototypeOf && (method in Object.getPrototypeOf(object)))) {
               117 +   if (
               118 +     hasOwnProperty(object, method) &&
               119 +     Object.getPrototypeOf &&
               120 +     method in Object.getPrototypeOf(object)
               121 +   ) {
110            122       return false
```

```
111  123      }
     124  +
112  125      // SAFE: whitelisted
113  126      // e.g toString
114       -    if (hasOwnProperty(safeNativeMethods, method)) {
     127  +    if (safeNativeMethods.has(method)) {
115  128        return true
116  129      }
     130  +
117  131      // UNSAFE: inherited from Object prototype
118  132      // e.g constructor
119  133      if (method in Object.prototype) {
          @@ -122,6 +136,7 @@ function isSafeMethod (object, method) {
122  136        // here because Object.prototype is a root object
123  137        return false
124  138      }
     139  +
125  140      // UNSAFE: inherited from Function prototype
126  141      // e.g call, apply
127  142      if (method in Function.prototype) {
          @@ -130,27 +145,12 @@ function isSafeMethod (object, method) {
130  145        // here because Function.prototype is a root object
131  146        return false
132  147      }
     148  +
133  149      return true
134  150    }
135  151
136       - function isPlainObject (object) {
     152  + export function isPlainObject (object) {
137  153      return typeof object === 'object' && object && object.constructor === Object
138  154    }
139  155
140       - const safeNativeProperties = {
141       -   length: true,
142       -   name: true
143       - }
144       -
145       - const safeNativeMethods = {
146       -   toString: true,
```

```
147      -   valueOf: true,
148      -   toLocaleString: true
149      - }
150      -
151      - export { getSafeProperty }
152      - export { setSafeProperty }
153      - export { isSafeProperty }
154      - export { getSafeMethod }
155      - export { isSafeMethod }
156      - export { isPlainObject }
         156 + const safeNativeMethods = new Set(['toString', 'valueOf', 'toLocaleString'])
```

src/utils/map.js

```
•••      @@ -1,4 +1,8 @@
1        - import { getSafeProperty, isSafeProperty, setSafeProperty } from './customs.js'
         1 + import {
         2 +   getSafeProperty,
         3 +   isSafeObjectProperty,
         4 +   setSafeProperty
         5 + } from './customs.js'
2        6   import { isMap, isObject } from './is.js'
3        7
4        8   /**
         @@ -32,7 +36,7 @@ export class ObjectWrappingMap {
32       36    }
33       37
34       38    has (key) {
35       -     return isSafeProperty(this.wrappedObject, key) && key in this.wrappedObject
         39 +     return isSafeObjectProperty(this.wrappedObject, key) && key in
                this.wrappedObject
36       40    }
37       41
38       42    entries () {
         @@ -46,7 +50,7 @@ export class ObjectWrappingMap {
46       50    }
47       51
48       52    delete (key) {
49       -     if (isSafeProperty(this.wrappedObject, key)) {
         53 +     if (isSafeObjectProperty(this.wrappedObject, key)) {
```

```
50  54          delete this.wrappedObject[key]
51  55        }
52  56      }
```

test/unit-tests/expression/security.test.js

```
    @@ -76,11 +76,55 @@ describe('security', function () {
76  76    it('should not allow calling Function via Object.assign', function () {
77  77      // TODO: simplify this test case, let it output console.log('hacked...')
78  78      assert.throws(function () {
79      -       math.evaluate('{}.constructor.assign(cos.constructor, {binding: cos.bind})\n' +
        79  +       math.evaluate(
        80  +         '{}.constructor.assign(cos.constructor, {binding: cos.bind})\n' +
80  81            '{}.constructor.assign(cos.constructor, {bind: null})\n' +
81  82            'f=cos.constructor.binding()("console.log(\'hacked...\')")\n' +
82      -           'f()')
83      -     }, /Error: No access to property "bind/)
        83  +           'f()'
        84  +       )
        85  +     }, /Error: No access to property "constructor/)
        86  +   })
        87  +
        88  +   it('should not allow misusing setSafeProperty to set properties on Arrays', function () {
        89  +     assert.throws(function () {
        90  +       const exploit = `
        91  +         constantNode = reviver('',{'mathjs':'ConstantNode','value':1});
        92  +
        93  +         # get a reference to a raw JavaScript array
        94  +         array = reviver('',{'mathjs':'ArrayNode'}).map().toJSON()['items'];
        95  +         array.push({'name':'a','type':{}});
        96  +
        97  +         # override params.map to hijack this.params and this.types
        98  +         array.map = f(callback)=callback({'name':{'name':'a','map':f2(callback2)=callback2({},'constructor')},'type':cos});
        99  +
       100  +         # trigger callback and get the pointer to Function.constructor
       101  +         functionAssignmentNode = reviver('',{'mathjs':'FunctionAssignmentNode','name':'a','params':array,'expr':constantNod
```

```
      e});
102   +         func = functionAssignmentNode.toJSON()['params']['type'];
103   +
104   +         getProcess = func('return process');
105   +         getProcess()`
106   +
107   +       console.log(
108   +         'Hacked! node.js version:',
109   +         math.evaluate(exploit).entries[0].version
110   +       )
111   +     }, /Error: No access to property "map"/)
112   +   })
113   +
114   +   it('should not allow getting access to constructor via DenseMatrix.get index
      argument', function () {
115   +     assert.throws(function () {
116   +       const exploit = `
117   +         m = matrix();
118   +         func =
      m.get({"length":1,"reduce":f(callback,a)=callback(cos,"constructor")});
119   +         getProcess = func("return process");
120   +         getProcess()
121   +         `
122   +
123   +       console.log(
124   +         'Hacked! node.js version:',
125   +         math.evaluate(exploit).entries[0].version
126   +       )
127   +     }, /Error: Array expected for index/)
84    128     })
85    129
86    130     it('should not allow disguising forbidden properties with unicode
      characters', function () {
```

test/unit-tests/utils/customs.test.js

```
      @@ -1,14 +1,16 @@
1     1     // test boolean utils
2     2     import assert from 'assert'
      3   + import math from '../../../src/defaultInstance.js'
```

```
3   4
4   5     import {
5   6       getSafeMethod,
6   7       getSafeProperty,
7   8       isPlainObject,
    9   +   isSafeArrayProperty,
8   10      isSafeMethod,
9       -   isSafeProperty
    11  +   isSafeObjectProperty,
    12  +   setSafeProperty
10  13    } from '../../../src/utils/customs.js'
11      - import math from '../../../src/defaultInstance.js'
12  14
13  15    describe('customs', function () {
14  16      describe('isSafeMethod', function () {
          @@ -130,55 +132,63 @@ describe('customs', function () {
130 132       })
131 133     })
132 134
133     -   describe('isSafeProperty', function () {
    135 +   describe('isSafeObjectProperty', function () {
134 136       it('should test properties on plain objects', function () {
135 137         const object = {}
136 138
137 139         /* From Object.prototype:
138 140           Object.getOwnPropertyNames(Object.prototype).forEach(
139 141             key => typeof ({})[key] !== 'function' && console.log(key))
140 142         */
141     -       assert.strictEqual(isSafeProperty(object, '__proto__'), false)
142     -       assert.strictEqual(isSafeProperty(object, 'constructor'), false)
    143 +       assert.strictEqual(isSafeObjectProperty(object, '__proto__'), false)
    144 +       assert.strictEqual(isSafeObjectProperty(object, 'constructor'), false)
143 145
144 146         /* From Function.prototype:
145 147           Object.getOwnPropertyNames(Function.prototype).forEach(
146 148             key => typeof (function () {})[key] !== 'function' &&
    console.log(key))
147 149         */
148     -       assert.strictEqual(isSafeProperty(object, 'length'), true)
```

```
149      -       assert.strictEqual(isSafeProperty(object, 'name'), true)
150      -       assert.strictEqual(isSafeProperty(object, 'arguments'), false)
151      -       assert.strictEqual(isSafeProperty(object, 'caller'), false)
         150 +   assert.strictEqual(isSafeObjectProperty(object, 'length'), true)
         151 +   assert.strictEqual(isSafeObjectProperty(object, 'name'), true)
         152 +   // assert.strictEqual(isSafeObjectProperty(object, 'arguments'), false)
         153 +   // assert.strictEqual(isSafeObjectProperty(object, 'caller'), false)
152      154
153      155        // non-existing property
154      -       assert.strictEqual(isSafeProperty(object, 'bar'), true)
         156 +   assert.strictEqual(isSafeObjectProperty(object, 'bar'), true)
155      157
156      158        // property with unicode chars
157      -       assert.strictEqual(isSafeProperty(object, 'co\u006Estructor'), false)
         159 +   assert.strictEqual(
         160 +     isSafeObjectProperty(object, 'co\u006Estructor'),
         161 +     false
         162 +   )
158      163      })
159      164
160      165      it('should test inherited properties on plain objects', function () {
161      166        const object1 = {}
162      167        const object2 = Object.create(object1)
163      168        object1.foo = true
164      169        object2.bar = true
165      -       assert.strictEqual(isSafeProperty(object2, 'foo'), true)
166      -       assert.strictEqual(isSafeProperty(object2, 'bar'), true)
167      -       assert.strictEqual(isSafeProperty(object2, '__proto__'), false)
168      -       assert.strictEqual(isSafeProperty(object2, 'constructor'), false)
         170 +   assert.strictEqual(isSafeObjectProperty(object2, 'foo'), true)
         171 +   assert.strictEqual(isSafeObjectProperty(object2, 'bar'), true)
         172 +   assert.strictEqual(isSafeObjectProperty(object2, '__proto__'), false)
         173 +   assert.strictEqual(isSafeObjectProperty(object2, 'constructor'), false)
169      174
170      175        object2.foo = true // override "foo" of object1
171      -       assert.strictEqual(isSafeProperty(object2, 'foo'), true)
172      -       assert.strictEqual(isSafeProperty(object2, 'constructor'), false)
         176 +   assert.strictEqual(isSafeObjectProperty(object2, 'foo'), true)
         177 +   assert.strictEqual(isSafeObjectProperty(object2, 'constructor'), false)
173      178      })
```

```
                179 +   })
174             180
                181 +   describe('isSafeArrayProperty', function () {
175             182       it('should test properties on an array', function () {
176             183         const array = [3, 2, 1]
177                 -       assert.strictEqual(isSafeProperty(array, 'length'), true)
178                 -       assert.strictEqual(isSafeProperty(array, 'foo'), true)
179                 -       assert.strictEqual(isSafeProperty(array, 'sort'), true)
180                 -       assert.strictEqual(isSafeProperty(array, '__proto__'), false)
181                 -       assert.strictEqual(isSafeProperty(array, 'constructor'), false)
                184 +       assert.strictEqual(isSafeArrayProperty(array, 'length'), true)
                185 +       assert.strictEqual(isSafeArrayProperty(array, 42), true)
                186 +       assert.strictEqual(isSafeArrayProperty(array, '24'), true)
                187 +
                188 +       assert.strictEqual(isSafeArrayProperty(array, 'foo'), false)
                189 +       assert.strictEqual(isSafeArrayProperty(array, 'sort'), false)
                190 +       assert.strictEqual(isSafeArrayProperty(array, '__proto__'), false)
                191 +       assert.strictEqual(isSafeArrayProperty(array, 'constructor'), false)
182             192       })
183             193     })
184             194
        @@ -195,6 +205,19 @@ describe('customs', function () {
195             205         assert.strictEqual(getSafeProperty(obj, 'username'), 'Joe')
196             206       })
197             207
                208 +     it('should return a property from an array when safe', function () {
                209 +       const array = [3, 2, 1]
                210 +       array.foo = 42
                211 +       assert.strictEqual(getSafeProperty(array, 'length'), 3)
                212 +       assert.throws(() => getSafeProperty(array, 'foo'), /No access to
                    property/)
                213 +       assert.throws(
                214 +         () => getSafeProperty(array, 'sort'),
                215 +         /Error: Cannot access method "sort" as a property/
                216 +       )
                217 +       assert.throws(() => getSafeProperty(array, '__proto__'), /No access to
                    property/)
                218 +       assert.throws(() => getSafeProperty(array, 'constructor'), /No access to
                    property "constructor"/)
                219 +     })
```

```
      220 +
198   221         it('should throw an exception when a method is unsafe', function () {
199   222           assert.throws(() => {
200   223             getSafeProperty(Function, 'constructor')
          @@ -208,6 +231,41 @@ describe('customs', function () {
208   231         })
209   232       })
210   233
      234 +   describe('setSafeProperty', function () {
      235 +     it('should set a property on an object when safe', function () {
      236 +       const obj = {}
      237 +       setSafeProperty(obj, 'foo', 42)
      238 +       assert.deepStrictEqual(obj, { foo: 42 })
      239 +
      240 +       assert.throws(() => {
      241 +         setSafeProperty(obj, 'constructor', 42)
      242 +       }, /No access to property "constructor"/)
      243 +     })
      244 +
      245 +     it('should set a property on an object when safe (2)', function () {
      246 +       const obj = {}
      247 +       setSafeProperty(obj, '42', 'fortytwo')
      248 +       assert.deepStrictEqual(obj, { 42: 'fortytwo' })
      249 +     })
      250 +
      251 +     it('should set a property on an array when safe', function () {
      252 +       const arr = []
      253 +       setSafeProperty(arr, 0, 'zero')
      254 +       setSafeProperty(arr, '1', 'one')
      255 +       setSafeProperty(arr, '2', 'two')
      256 +       assert.deepStrictEqual(arr, ['zero', 'one', 'two'])
      257 +
      258 +       setSafeProperty(arr, 'length', 10)
      259 +       assert.deepStrictEqual(arr.length, 10)
      260 +     })
      261 +
      262 +     it('should not allow setting a non-numeric property on an array', function
            () {
      263 +       assert.throws(() => {
      264 +         setSafeProperty([], 'map', () => {})
```

```
265 +       }, /No access to property "map"/)
266 +     })
267 +   })
268 +
211 269     it('should distinguish plain objects', function () {
212 270       const a = {}
213 271       const b = Object.create(a)
```

Comments 0

Please sign in to comment.