JavaScript static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVASCRIPT code

1. HTTP responses should not be vulnerable to session fixation
   Vulnerability
2. DOM updates should not lead to open redirect vulnerabilities
   Vulnerability
3. Extracting archives should not lead to zip slip vulnerabilities
   Vulnerability
4. DOM updates should not lead to cross-site scripting (XSS) attacks
   Vulnerability
5. Dynamic code execution should not be vulnerable to injection attacks
   Vulnerability
6. NoSQL operations should not be vulnerable to injection attacks
   Vulnerability
7. HTTP request redirections should not be open to forging attacks
   Vulnerability
8. Endpoints should not be vulnerable to reflected cross-site scripting (XSS) attacks
   Vulnerability
9. Database queries should not be vulnerable to injection attacks
   Vulnerability
10. XML parsers should not be vulnerable to XXE attacks
    Vulnerability
11. I/O function calls should not be vulnerable to path injection attacks
    Vulnerability
12. OS commands should not be vulnerable to command injection attacks
    Vulnerability
13. Callbacks of array methods should have return statements
    Bug
14. Loops should not be infinite
    Bug
15. Disabling Vue.js built-in escaping is security-sensitive
    Security Hotspot
16.

	Disabling Angular built-in sanitization is security-sensitive Security Hotspot
17.	
	Hard-coded credentials are security-sensitive Security Hotspot
18.	
	Function returns should not be invariant Code Smell
19.	
	Assertions should be complete Code Smell
20.	
	Variables should be declared explicitly Code Smell
21.	
	Tests should include assertions Code Smell
22.	
	"future reserved words" should not be used as identifiers Code Smell
23.	
	Octal values should not be used Code Smell
24.	
	Switch cases should end with an unconditional "break" statement Code Smell
25.	
	"switch" statements should not contain non-case labels Code Smell
26.	
	A new session should be created during user authentication Vulnerability
27.	
	JWT should be signed and verified with strong cipher algorithms Vulnerability
28.	
	Cipher algorithms should be robust Vulnerability
29.	
	Encryption algorithms should be used with secure mode and padding scheme Vulnerability
30.	
	Server hostnames should be verified during SSL/TLS connections Vulnerability
31.	
	Server certificates should be verified during SSL/TLS connections Vulnerability
32.	
	Cryptographic keys should be robust Vulnerability
33.	

	Weak SSL/TLS protocols should not be used Vulnerability
34.	
	Origins should be verified during cross-origin communications Vulnerability
35.	
	Regular expressions should not be vulnerable to Denial of Service attacks Vulnerability
36.	
	File uploads should be restricted Vulnerability
37.	
	Function calls should not pass extra arguments Bug
38.	
	Regular expressions should be syntactically valid Bug
39.	
	Getters and setters should access the expected fields Bug
40.	
	"super()" should be invoked appropriately Bug
41.	
	"Symbol" should not be used as a constructor Bug
42.	
	Results of "in" and "instanceof" should be negated rather than operands Bug
43.	
	"in" should not be used with primitive types Bug
44.	
	A compare function should be provided when using "Array.prototype.sort()" Bug
45.	
	Jump statements should not occur in "finally" blocks Bug
46.	
	Using slow regular expressions is security-sensitive Security Hotspot
47.	
	Using publicly writable directories is security-sensitive Security Hotspot
48.	
	Using clear-text protocols is security-sensitive Security Hotspot
49.	
	Expanding archive files without controlling resource consumption is security-sensitive Security Hotspot
50.	

	Using weak hashing algorithms is security-sensitive Security Hotspot
51.	
	Disabling CSRF protections is security-sensitive Security Hotspot
52.	
	Using pseudorandom number generators (PRNGs) is security-sensitive Security Hotspot
53.	
	Dynamically executing code is security-sensitive Security Hotspot
54.	
	Equality operators should not be used in "for" loop termination conditions Code Smell
55.	
	Tests should not execute any code after "done()" is called Code Smell
56.	
	"default" clauses should be last Code Smell
57.	
	"await" should only be used with promises Code Smell
58.	
	A conditionally executed single line should be denoted by indentation Code Smell
59.	
	Conditionals should start on new lines Code Smell
60.	
	Cognitive Complexity of functions should not be too high Code Smell
61.	
	"void" should not be used Code Smell
62.	
	Loop counters should not be assigned to from within the loop body Code Smell
63.	
	"for" loop increment clauses should modify the loops' counters Code Smell
64.	
	Functions should not be empty Code Smell
65.	
	Server-side requests should not be vulnerable to forging attacks Vulnerability
66.	
	Non-empty statements should change control flow or have at least one side-effect Bug
67.	

	Regular expressions with the global flag should be used with caution Bug
68.	
	Replacement strings should reference existing regular expression groups Bug
69.	
	Regular expressions should not contain control characters Bug
70.	
	Alternation in regular expressions should not contain empty alternatives Bug
71.	
	Mocha timeout should be disabled by setting it to "0". Bug
72.	
	Unicode Grapheme Clusters should be avoided inside regex character classes Bug
73.	
	Assertions should not be given twice the same argument Bug
74.	
	Alternatives in regular expressions should be grouped when used with anchors Bug
75.	
	Promise rejections should not be caught by 'try' block Bug
76.	
	Collection elements should not be replaced unconditionally Bug
77.	
	Errors should not be created without being thrown Bug
78.	
	Collection sizes and array length comparisons should make sense Bug
79.	
	All branches in a conditional structure should not have exactly the same implementation Bug
80.	
	Destructuring patterns should not be empty Bug
81.	
	The output of functions that don't return anything should not be used Bug
82.	
	Comma and logical OR operators should not be used in switch cases Bug
83.	
	Generators should "yield" something Bug
84.	

	Attempts should not be made to update "const" variables Bug
85.	
	Strict equality operators should not be used with dissimilar types Bug
86.	
	"new" operators should be used with functions Bug
87.	
	Non-existent operators '=+', '=-' and '=!' should not be used Bug
88.	
	"NaN" should not be used in comparisons Bug
89.	
	Setters should not return values Bug
90.	
	Properties of variables with "null" or "undefined" values should not be accessed Bug
91.	
	A "for" loop update clause should move the counter in the right direction Bug
92.	
	Return values from functions without side effects should not be ignored Bug
93.	
	Special identifiers should not be bound or assigned Bug
94.	
	Values should not be uselessly incremented Bug
95.	
	Related "if/else if" statements should not have the same condition Bug
96.	
	Objects should not be created to be dropped immediately without being used Bug
97.	
	Identical expressions should not be used on both sides of a binary operator Bug
98.	
	All code should be reachable Bug
99.	
	Loops with at most one iteration should be refactored Bug
100.	
	Variables should not be self-assigned Bug
101.	

	Function argument names should be unique Bug
102.	
	Property names should not be duplicated within a class or object literal Bug
103.	
	Bitwise operators should not be used in boolean contexts Bug
104.	
	Constructing arguments of system commands from user input is security-sensitive Security Hotspot
105.	
	Allowing requests with excessive content length is security-sensitive Security Hotspot
106.	
	Statically serving hidden files is security-sensitive Security Hotspot
107.	
	Using intrusive permissions is security-sensitive Security Hotspot
108.	
	Disabling auto-escaping in template engines is security-sensitive Security Hotspot
109.	
	Using shell interpreter when executing OS commands is security-sensitive Security Hotspot
110.	
	Setting loose POSIX file permissions is security-sensitive Security Hotspot
111.	
	Formatting SQL queries is security-sensitive Security Hotspot
112.	
	Comma operator should not be used Code Smell
113.	
	Regular expressions should not contain empty groups Code Smell
114.	
	Regular expressions should not contain multiple spaces Code Smell
115.	
	Chai assertions should have only one reason to succeed Code Smell
116.	
	Single-character alternations in regular expressions should be replaced with character classes Code Smell
117.	
	Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string Code Smell

118. Tests should check which exception is thrown
Code Smell

119. Character classes in regular expressions should not contain the same character twice
Code Smell

120. Names of regular expressions named groups should be used
Code Smell

121. Regular expressions should not be too complicated
Code Smell

122. Shorthand promises should be used
Code Smell

123. Template literals should not be nested
Code Smell

124. "in" should not be used on arrays
Code Smell

125. Assignments should not be redundant
Code Smell

126. Functions should not have identical implementations
Code Smell

127. Sparse arrays should not be declared
Code Smell

128. Array-mutating methods should not be used misleadingly
Code Smell

129. Collection and array contents should be used
Code Smell

130. Functions should always return the same type
Code Smell

131. Arguments to built-in functions should match documented types
Code Smell

132. Literals should not be thrown
Code Smell

133. Functions should not be called both with and without "new"
Code Smell

134. Array indexes should be numeric
Code Smell

135. Assertion arguments should be passed in the correct order
Code Smell

136. Ternary operators should not be nested
Code Smell

137. "delete" should not be used on arrays
Code Smell

138. Variables and functions should not be redeclared
Code Smell

139. "indexOf" checks should not be for positive numbers
Code Smell

140. "arguments.caller" and "arguments.callee" should not be used
Code Smell

141. Multiline blocks should be enclosed in curly braces
Code Smell

142. Boolean expressions should not be gratuitous
Code Smell

143. Variables should be used in the blocks where they are declared
Code Smell

144. Parameters should be passed in the correct order
Code Smell

145. Two branches in a conditional structure should not have exactly the same implementation
Code Smell

146. Unused assignments should be removed
Code Smell

147. Function parameters with default values should be last
Code Smell

148. Functions should not be defined inside loops
Code Smell

149. "switch" statements should not have too many "case" clauses
Code Smell

150. Only "while", "do", "for" and "switch" statements should be labelled
Code Smell

151. Sections of code should not be commented out

Code Smell

152.

Unused function parameters should be removed
Code Smell

153.

Track uses of "FIXME" tags
Code Smell

154.

Assignments should not be made from within sub-expressions
Code Smell

155.

Labels should not be used
Code Smell

156.

Variables should not be shadowed
Code Smell

157.

Redundant pairs of parentheses should be removed
Code Smell

158.

Nested blocks of code should not be left empty
Code Smell

159.

Functions should not have too many parameters
Code Smell

160.

OS commands should not be vulnerable to argument injection attacks
Vulnerability

161.

Repeated patterns in regular expressions should not match the empty string
Bug

162.

Empty collections should not be accessed or iterated
Bug

163.

"delete" should be used only with object properties
Bug

164.

"with" statements should not be used
Bug

165.

Function parameters, caught exceptions and foreach variables' initial values should not be ignored
Bug

166.

Forwarding client IP address is security-sensitive
Security Hotspot

167.

Allowing confidential information to be logged is security-sensitive
Security Hotspot

168.

	Allowing browsers to perform DNS prefetching is security-sensitive Security Hotspot
169.	
	Disabling Certificate Transparency monitoring is security-sensitive Security Hotspot
170.	
	Disabling Strict-Transport-Security policy is security-sensitive Security Hotspot
171.	
	Disabling strict HTTP no-referrer policy is security-sensitive Security Hotspot
172.	
	Allowing browsers to sniff MIME types is security-sensitive Security Hotspot
173.	
	Disabling content security policy frame-ancestors directive is security-sensitive Security Hotspot
174.	
	Allowing mixed-content is security-sensitive Security Hotspot
175.	
	Disabling content security policy fetch directives is security-sensitive Security Hotspot
176.	
	Disabling resource integrity features is security-sensitive Security Hotspot
177.	
	Disclosing fingerprints from web application technologies is security-sensitive Security Hotspot
178.	
	Having a permissive Cross-Origin Resource Sharing policy is security-sensitive Security Hotspot
179.	
	Delivering code in production with debug features activated is security-sensitive Security Hotspot
180.	
	Creating cookies without the "HttpOnly" flag is security-sensitive Security Hotspot
181.	
	Creating cookies without the "secure" flag is security-sensitive Security Hotspot
182.	
	Using hardcoded IP addresses is security-sensitive Security Hotspot
183.	
	Regular expression quantifiers and character classes should be used concisely Code Smell
184.	
	Regular expression literals should be used when possible Code Smell
185.	

	"await" should not be used redundantly Code Smell
186.	
	"for of" should be used with Iterables Code Smell
187.	
	Imports from the same modules should be merged Code Smell
188.	
	Jump statements should not be redundant Code Smell
189.	
	Default export names and file names should match Code Smell
190.	
	The global "this" object should not be used Code Smell
191.	
	"catch" clauses should do more than rethrow Code Smell
192.	
	Boolean checks should not be inverted Code Smell
193.	
	Deprecated APIs should not be used Code Smell
194.	
	Wrapper objects should not be used for primitive types Code Smell
195.	
	Multiline string literals should not be used Code Smell
196.	
	Local variables should not be declared and then immediately returned or thrown Code Smell
197.	
	Unused local variables and functions should be removed Code Smell
198.	
	Function call arguments should not start on new lines Code Smell
199.	
	"switch" statements should have at least 3 "case" clauses Code Smell
200.	
	A "while" loop should be used instead of a "for" loop Code Smell
201.	
	Unnecessary imports should be removed Code Smell
202.	

	Return of boolean expressions should not be wrapped into an "if-then-else" statement Code Smell
203.	
	Boolean literals should not be used in comparisons Code Smell
204.	
	Extra semicolons should be removed Code Smell
205.	
	Class names should comply with a naming convention Code Smell
206.	
	Track uses of "TODO" tags Code Smell
207.	
	Web SQL databases should not be used Vulnerability
208.	
	Variables should be defined before being used Bug
209.	
	Variables declared with "var" should be declared before they are used Code Smell
210.	
	Track lack of copyright and license headers Code Smell
211.	
	Reading the Standard Input is security-sensitive Security Hotspot
212.	
	Using command line arguments is security-sensitive Security Hotspot
213.	
	Using Sockets is security-sensitive Security Hotspot
214.	
	Executing XPath expressions is security-sensitive Security Hotspot
215.	
	Encrypting data is security-sensitive Security Hotspot
216.	
	Using regular expressions is security-sensitive Security Hotspot
217.	
	Class methods should be used instead of "prototype" assignments Code Smell
218.	
	Function constructors should not be used Code Smell
219.	

	Variables should be declared with "let" or "const" Code Smell
220.	
	Unchanged variables should be marked "const" Code Smell
221.	
	Wildcard imports should not be used Code Smell
222.	
	"switch" statements should not be nested Code Smell
223.	
	Cyclomatic Complexity of functions should not be too high Code Smell
224.	
	"strict" mode should be used with caution Code Smell
225.	
	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply Code Smell
226.	
	"switch" statements should have "default" clauses Code Smell
227.	
	"if ... else if" constructs should end with "else" clauses Code Smell
228.	
	Control structures should use curly braces Code Smell
229.	
	String literals should not be duplicated Code Smell
230.	
	Expressions should not be too complex Code Smell
231.	
	Local storage should not be used Vulnerability
232.	
	Template literal placeholder syntax should not be used in regular strings Bug
233.	
	Built-in objects should not be overridden Bug
234.	
	"for...in" loops should filter properties before acting on them Bug
235.	
	Results of operations on strings should not be ignored Bug

236. Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression
Code Smell

237. "for in" should not be used with iterables
Code Smell

238. Functions should use "return" consistently
Code Smell

239. Variables and functions should not be declared in the global scope
Code Smell

240. Arithmetic operators should only have numbers as operands
Code Smell

241. Values not convertible to numbers should not be used in numeric comparisons
Code Smell

242. Arithmetic operations should not result in "NaN"
Code Smell

243. "arguments" should not be accessed directly
Code Smell

244. Comparison operators should not be used with strings
Code Smell

245. Property getters and setters should come in pairs
Code Smell

246. JavaScript parser failure
Code Smell

247. The ternary operator should not be used
Code Smell

248. "===" and "!==" should be used instead of "==" and "!="
Code Smell

249. Functions should not have too many lines of code
Code Smell

250. Track comments matching a regular expression
Code Smell

251. Statements should be on separate lines
Code Smell

252. Magic numbers should not be used

Code Smell

253.

Collapsible "if" statements should be merged
Code Smell

254.

Standard outputs should not be used directly to log anything
Code Smell

255.

Files should not have too many lines of code
Code Smell

256.

Lines should not be too long
Code Smell

257.

Debugger statements should not be used
Vulnerability

258.

"alert(...)" should not be used
Vulnerability

259.

Regular expressions using Unicode character classes or property escapes should enable the unicode flag
Bug

260.

The base should be provided to "parseInt"
Bug

261.

Function declarations should not be made within blocks
Bug

262.

Writing cookies is security-sensitive
Security Hotspot

263.

"continue" should not be used
Code Smell

264.

Trailing commas should be used
Code Smell

265.

"import" should be used to include external code
Code Smell

266.

Braces and parentheses should be used consistently with arrow functions
Code Smell

267.

Destructuring syntax should be used for assignments
Code Smell

268.

Template strings should be used instead of concatenation
Code Smell

269.

	Shorthand object properties should be grouped at the beginning or end of an object declaration Code Smell
270.	
	Object literal shorthand syntax should be used Code Smell
271.	
	Strings and non-strings should not be added Code Smell
272.	
	Object literal syntax should be used Code Smell
273.	
	"undefined" should not be assigned Code Smell
274.	
	Trailing commas should not be used Code Smell
275.	
	Array constructors should not be used Code Smell
276.	
	Quotes for string literals should be used consistently Code Smell
277.	
	Statements should end with semicolons Code Smell
278.	
	Comments should not be located at the end of lines of code Code Smell
279.	
	Loops should not contain more than a single "break" or "continue" statement Code Smell
280.	
	Variable, property and parameter names should comply with a naming convention Code Smell
281.	
	Lines should not end with trailing whitespaces Code Smell
282.	
	Files should contain an empty newline at the end Code Smell
283.	
	An open curly brace should be located at the end of a line Code Smell
284.	
	Tabulation characters should not be used Code Smell
285.	
	Function and method names should comply with a naming convention Code Smell