TypeScript static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your TYPESCRIPT 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.
   Disabling Vue.js built-in escaping is security-sensitive
   Security Hotspot
14.
   Disabling Angular built-in sanitization is security-sensitive
   Security Hotspot
15.
   Hard-coded credentials are security-sensitive
   Security Hotspot
16.

Function returns should not be invariant
Code Smell

17.

Assertions should be complete
Code Smell

18.

Tests should include assertions
Code Smell

19.

Octal values should not be used
Code Smell

20.

Switch cases should end with an unconditional "break" statement
Code Smell

21.

"switch" statements should not contain non-case labels
Code Smell

22.

A new session should be created during user authentication
Vulnerability

23.

JWT should be signed and verified with strong cipher algorithms
Vulnerability

24.

Cipher algorithms should be robust
Vulnerability

25.

Encryption algorithms should be used with secure mode and padding scheme
Vulnerability

26.

Server hostnames should be verified during SSL/TLS connections
Vulnerability

27.

Server certificates should be verified during SSL/TLS connections
Vulnerability

28.

Cryptographic keys should be robust
Vulnerability

29.

Weak SSL/TLS protocols should not be used
Vulnerability

30.

Origins should be verified during cross-origin communications
Vulnerability

31.

Regular expressions should not be vulnerable to Denial of Service attacks
Vulnerability

32.

File uploads should be restricted
Vulnerability

33.

Regular expressions should be syntactically valid
Bug

34.

Types without members, 'any' and 'never' should not be used in type intersections
Bug

35.

Getters and setters should access the expected fields
Bug

36.

"super()" should be invoked appropriately
Bug

37.

Results of "in" and "instanceof" should be negated rather than operands
Bug

38.

A compare function should be provided when using "Array.prototype.sort()"
Bug

39.

Jump statements should not occur in "finally" blocks
Bug

40.

Using slow regular expressions is security-sensitive
Security Hotspot

41.

Using publicly writable directories is security-sensitive
Security Hotspot

42.

Using clear-text protocols is security-sensitive
Security Hotspot

43.

Expanding archive files without controlling resource consumption is security-sensitive
Security Hotspot

44.

Using weak hashing algorithms is security-sensitive
Security Hotspot

45.

Disabling CSRF protections is security-sensitive
Security Hotspot

46.

Using pseudorandom number generators (PRNGs) is security-sensitive
Security Hotspot

47.

Dynamically executing code is security-sensitive
Security Hotspot

48.

Equality operators should not be used in "for" loop termination conditions
Code Smell

49.

Tests should not execute any code after "done()" is called
Code Smell

50.

	Union and intersection types should not be defined with duplicated elements Code Smell
51.	
	"default" clauses should be last Code Smell
52.	
	"await" should only be used with promises Code Smell
53.	
	A conditionally executed single line should be denoted by indentation Code Smell
54.	
	Conditionals should start on new lines Code Smell
55.	
	Cognitive Complexity of functions should not be too high Code Smell
56.	
	"void" should not be used Code Smell
57.	
	Loop counters should not be assigned to from within the loop body Code Smell
58.	
	"for" loop increment clauses should modify the loops' counters Code Smell
59.	
	Functions should not be empty Code Smell
60.	
	Server-side requests should not be vulnerable to forging attacks Vulnerability
61.	
	Non-empty statements should change control flow or have at least one side-effect Bug
62.	
	Regular expressions with the global flag should be used with caution Bug
63.	
	Replacement strings should reference existing regular expression groups Bug
64.	
	Regular expressions should not contain control characters Bug
65.	
	Alternation in regular expressions should not contain empty alternatives Bug
66.	
	Mocha timeout should be disabled by setting it to "0". Bug
67.	

Unicode Grapheme Clusters should be avoided inside regex character classes
Bug

68.

Assertions should not be given twice the same argument
Bug

69.

Alternatives in regular expressions should be grouped when used with anchors
Bug

70.

Promise rejections should not be caught by 'try' block
Bug

71.

Collection elements should not be replaced unconditionally
Bug

72.

Constructors should not be declared inside interfaces
Bug

73.

Errors should not be created without being thrown
Bug

74.

Collection sizes and array length comparisons should make sense
Bug

75.

All branches in a conditional structure should not have exactly the same implementation
Bug

76.

Destructuring patterns should not be empty
Bug

77.

The output of functions that don't return anything should not be used
Bug

78.

Comma and logical OR operators should not be used in switch cases
Bug

79.

Generators should "yield" something
Bug

80.

"new" operators should be used with functions
Bug

81.

Non-existent operators '=+', '=-' and '=!' should not be used
Bug

82.

"NaN" should not be used in comparisons
Bug

83.

A "for" loop update clause should move the counter in the right direction
Bug

84.

Return values from functions without side effects should not be ignored
Bug

85.

Special identifiers should not be bound or assigned
Bug

86.

Values should not be uselessly incremented
Bug

87.

Related "if/else if" statements should not have the same condition
Bug

88.

Objects should not be created to be dropped immediately without being used
Bug

89.

Identical expressions should not be used on both sides of a binary operator
Bug

90.

All code should be reachable
Bug

91.

Loops with at most one iteration should be refactored
Bug

92.

Variables should not be self-assigned
Bug

93.

Bitwise operators should not be used in boolean contexts
Bug

94.

Constructing arguments of system commands from user input is security-sensitive
Security Hotspot

95.

Allowing requests with excessive content length is security-sensitive
Security Hotspot

96.

Statically serving hidden files is security-sensitive
Security Hotspot

97.

Using intrusive permissions is security-sensitive
Security Hotspot

98.

Disabling auto-escaping in template engines is security-sensitive
Security Hotspot

99.

Using shell interpreter when executing OS commands is security-sensitive
Security Hotspot

100.

Setting loose POSIX file permissions is security-sensitive
Security Hotspot

101.

Formatting SQL queries is security-sensitive
Security Hotspot

102.

Comma operator should not be used
Code Smell

103.

Regular expressions should not contain empty groups
Code Smell

104.

Regular expressions should not contain multiple spaces
Code Smell

105.

Chai assertions should have only one reason to succeed
Code Smell

106.

Single-character alternations in regular expressions should be replaced with character classes
Code Smell

107.

Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string
Code Smell

108.

Tests should check which exception is thrown
Code Smell

109.

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

110.

Names of regular expressions named groups should be used
Code Smell

111.

Regular expressions should not be too complicated
Code Smell

112.

Optional property declarations should not use both '?' and 'undefined' syntax
Code Smell

113.

Shorthand promises should be used
Code Smell

114.

Template literals should not be nested
Code Smell

115.

"undefined" should not be passed as the value of optional parameters
Code Smell

116.

"in" should not be used on arrays
Code Smell

117.

Assignments should not be redundant
Code Smell

118.
Functions should not have identical implementations
Code Smell

119.
Sparse arrays should not be declared
Code Smell

120.
Array-mutating methods should not be used misleadingly
Code Smell

121.
Collection and array contents should be used
Code Smell

122.
Literals should not be thrown
Code Smell

123.
Array indexes should be numeric
Code Smell

124.
Assertion arguments should be passed in the correct order
Code Smell

125.
Ternary operators should not be nested
Code Smell

126.
"delete" should not be used on arrays
Code Smell

127.
Variables and functions should not be redeclared
Code Smell

128.
"indexOf" checks should not be for positive numbers
Code Smell

129.
"arguments.caller" and "arguments.callee" should not be used
Code Smell

130.
Multiline blocks should be enclosed in curly braces
Code Smell

131.
Boolean expressions should not be gratuitous
Code Smell

132.
Variables should be used in the blocks where they are declared
Code Smell

133.
Parameters should be passed in the correct order
Code Smell

134.
Two branches in a conditional structure should not have exactly the same implementation

Code Smell

135.

Unused assignments should be removed
Code Smell

136.

Function parameters with default values should be last
Code Smell

137.

Functions should not be defined inside loops
Code Smell

138.

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

139.

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

140.

Sections of code should not be commented out
Code Smell

141.

Unused function parameters should be removed
Code Smell

142.

Track uses of "FIXME" tags
Code Smell

143.

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

144.

Labels should not be used
Code Smell

145.

Variables should not be shadowed
Code Smell

146.

Redundant pairs of parentheses should be removed
Code Smell

147.

Nested blocks of code should not be left empty
Code Smell

148.

Functions should not have too many parameters
Code Smell

149.

OS commands should not be vulnerable to argument injection attacks
Vulnerability

150.

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

151.

Empty collections should not be accessed or iterated

Bug

152.

"delete" should be used only with object properties
Bug

153.

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

154.

Forwarding client IP address is security-sensitive
Security Hotspot

155.

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

156.

Allowing browsers to perform DNS prefetching is security-sensitive
Security Hotspot

157.

Disabling Certificate Transparency monitoring is security-sensitive
Security Hotspot

158.

Disabling Strict-Transport-Security policy is security-sensitive
Security Hotspot

159.

Disabling strict HTTP no-referrer policy is security-sensitive
Security Hotspot

160.

Allowing browsers to sniff MIME types is security-sensitive
Security Hotspot

161.

Disabling content security policy frame-ancestors directive is security-sensitive
Security Hotspot

162.

Allowing mixed-content is security-sensitive
Security Hotspot

163.

Disabling content security policy fetch directives is security-sensitive
Security Hotspot

164.

Disabling resource integrity features is security-sensitive
Security Hotspot

165.

Disclosing fingerprints from web application technologies is security-sensitive
Security Hotspot

166.

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive
Security Hotspot

167.

Delivering code in production with debug features activated is security-sensitive
Security Hotspot

168.

Creating cookies without the "HttpOnly" flag is security-sensitive
Security Hotspot

169.

Creating cookies without the "secure" flag is security-sensitive
Security Hotspot

170.

Using hardcoded IP addresses is security-sensitive
Security Hotspot

171.

Regular expression quantifiers and character classes should be used concisely
Code Smell

172.

Regular expression literals should be used when possible
Code Smell

173.

"await" should not be used redundantly
Code Smell

174.

Redundant casts and non-null assertions should be avoided
Code Smell

175.

Type aliases should be used
Code Smell

176.

Type guards should be used
Code Smell

177.

"module" should not be used
Code Smell

178.

"for of" should be used with Iterables
Code Smell

179.

Imports from the same modules should be merged
Code Smell

180.

Jump statements should not be redundant
Code Smell

181.

Default export names and file names should match
Code Smell

182.

The global "this" object should not be used
Code Smell

183.

"catch" clauses should do more than rethrow
Code Smell

184.

Boolean checks should not be inverted
Code Smell

185.

Deprecated APIs should not be used
Code Smell

186.

Wrapper objects should not be used for primitive types
Code Smell

187.

Multiline string literals should not be used
Code Smell

188.

Local variables should not be declared and then immediately returned or thrown
Code Smell

189.

Function call arguments should not start on new lines
Code Smell

190.

"switch" statements should have at least 3 "case" clauses
Code Smell

191.

A "while" loop should be used instead of a "for" loop
Code Smell

192.

Unnecessary imports should be removed
Code Smell

193.

Boolean literals should not be used in comparisons
Code Smell

194.

Extra semicolons should be removed
Code Smell

195.

Class names should comply with a naming convention
Code Smell

196.

Track uses of "TODO" tags
Code Smell

197.

Web SQL databases should not be used
Vulnerability

198.

Variables declared with "var" should be declared before they are used
Code Smell

199.

Track lack of copyright and license headers
Code Smell

200.

Reading the Standard Input is security-sensitive
Security Hotspot

201.

Using command line arguments is security-sensitive
Security Hotspot

202.

Using Sockets is security-sensitive
Security Hotspot

203.
Executing XPath expressions is security-sensitive
Security Hotspot

204.
Encrypting data is security-sensitive
Security Hotspot

205.
Using regular expressions is security-sensitive
Security Hotspot

206.
Class methods should be used instead of "prototype" assignments
Code Smell

207.
Variables should be declared with "let" or "const"
Code Smell

208.
Unchanged variables should be marked "const"
Code Smell

209.
Wildcard imports should not be used
Code Smell

210.
"switch" statements should not be nested
Code Smell

211.
Cyclomatic Complexity of functions should not be too high
Code Smell

212.
"strict" mode should be used with caution
Code Smell

213.
Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply
Code Smell

214.
"switch" statements should have "default" clauses
Code Smell

215.
"if ... else if" constructs should end with "else" clauses
Code Smell

216.
Control structures should use curly braces
Code Smell

217.
String literals should not be duplicated
Code Smell

218.
Expressions should not be too complex
Code Smell

219.

Template literal placeholder syntax should not be used in regular strings
Bug

220.

Built-in objects should not be overridden
Bug

221.

"for...in" loops should filter properties before acting on them
Bug

222.

Results of operations on strings should not be ignored
Bug

223.

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

224.

Optional boolean parameters should have default value
Code Smell

225.

Union types should not have too many elements
Code Smell

226.

Dependencies should be explicit
Code Smell

227.

"this" should not be assigned to variables
Code Smell

228.

The "any" type should not be used
Code Smell

229.

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

230.

Functions should use "return" consistently
Code Smell

231.

"arguments" should not be accessed directly
Code Smell

232.

Comparison operators should not be used with strings
Code Smell

233.

Private properties that are only assigned in the constructor or at declaration should be "readonly"
Code Smell

234.

Property getters and setters should come in pairs
Code Smell

235.

JavaScript parser failure
Code Smell

236.

The ternary operator should not be used
Code Smell

237.

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

238.

Functions should not have too many lines of code
Code Smell

239.

Track comments matching a regular expression
Code Smell

240.

Statements should be on separate lines
Code Smell

241.

Magic numbers should not be used
Code Smell

242.

Collapsible "if" statements should be merged
Code Smell

243.

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

244.

Files should not have too many lines of code
Code Smell

245.

Lines should not be too long
Code Smell

246.

Debugger statements should not be used
Vulnerability

247.

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

248.

The base should be provided to "parseInt"
Bug

249.

Function declarations should not be made within blocks
Bug

250.

Writing cookies is security-sensitive
Security Hotspot

251.

"continue" should not be used
Code Smell

252.
Primitive return types should be used
Code Smell

253.
Default type parameters should be omitted
Code Smell

254.
Type assertions should use "as"
Code Smell

255.
Method overloads should be grouped together
Code Smell

256.
Interfaces should not be empty
Code Smell

257.
Trailing commas should be used
Code Smell

258.
"import" should be used to include external code
Code Smell

259.
Braces and parentheses should be used consistently with arrow functions
Code Smell

260.
Destructuring syntax should be used for assignments
Code Smell

261.
Template strings should be used instead of concatenation
Code Smell

262.
Shorthand object properties should be grouped at the beginning or end of an object declaration
Code Smell

263.
Object literal shorthand syntax should be used
Code Smell

264.
Strings and non-strings should not be added
Code Smell

265.
Primitive types should be omitted from initialized or defaulted declarations
Code Smell

266.
Non-null assertions should not be used
Code Smell

267.
"undefined" should not be assigned
Code Smell

268.
Trailing commas should not be used

Code Smell

269.

Array constructors should not be used
Code Smell

270.

Quotes for string literals should be used consistently
Code Smell

271.

Statements should end with semicolons
Code Smell

272.

Comments should not be located at the end of lines of code
Code Smell

273.

Loops should not contain more than a single "break" or "continue" statement
Code Smell

274.

Variable, property and parameter names should comply with a naming convention
Code Smell

275.

Lines should not end with trailing whitespaces
Code Smell

276.

Files should contain an empty newline at the end
Code Smell

277.

An open curly brace should be located at the end of a line
Code Smell

278.

Tabulation characters should not be used
Code Smell

279.

Function and method names should comply with a naming convention
Code Smell