SONAR JS RULES

BUGS

1)NaN should not be used in comparison

NaN is not equal to anything, itself also. Hence NaN should never be used for comparison.

2)Function calls should not pass extra arguments

In JavaScript function calls can have any number of arguments. But if we pass extra arguments, they will automatically be ignored.

3) "in" should not be used with primitive types

The keyword "in" helps us to know if a particular property is present or not. But using "in" with primitive types raises TypeError

4) Result of "in" and "instanceOf" should be negated but not the operands

!myVar in myList looks for negation of myVar but not for the negation of the result. Hence it is correct to write it as !(myVar in myList). The same is with instanceOf()

5) "new" operator should be used with functions only

The new keyword is used only with constructor functions else it'll raise TypeError

6) Bitwise operators &, | should not be confused with Logical/ Boolean operators &&, ||

7) Calls should not be made to non callable values

Only callable values like functions etc should be called.

For example,

foo=1

foo() // is not acceptable and raises TypeError

8) All conditional blocks should be reachable

If we know that some conditional block will never get executed, there would be no point in writing it there.

For example,

a=false

```
if(a)
{
	//this is meaningless
}
```

9) Generators should yield something
If there is no yield statement then the iterator returned will be null and there is no point of writing the generator.

10) “delete” should be used only with the object properties
If “delete” is used with other then object properties, it may or may not give the desired result.

VULNERABILITIES

1) Debugger statements must be removed
Debugger statements are like breakpoints in the code and should be avoided while production as they might halt or break the execution

2) alert should not be used
Using alert () is not good while production as it might expose some important information to attackers.Hence it should be used only while testing.

3) Console logging should not be used
The above reasons for alert apply here. It wouldn’t be appropriate if there are many console.logs in the production code.

4) Local storage should not be used
Using local storage may speed up apps but the data is not encrypted here, hence it might expose some sensitive information.

5)Cross-document messaging domains should be carefully restricted

Authors should not use the wildcard keyword (*) in the targetOrigin argument in messages that contain any confidential information, as otherwise there is no way to guarantee that the message is only delivered to the recipient to which it was intended.

6)Untrusted content should not be included
Including content in your site from an untrusted source can expose your users to attackers and even compromise your own site. For that reason, this rule raises an issue for each non-relative URL.

7)Web SQL databases should not be used
The Web SQL Database standard never saw the light of day. It was first formulated, then deprecated by the W3C and was only implemented in some browsers. (It is not supported in Firefox or IE.) Further, the use of a Web SQL Database poses security concerns, since you only need its name to access such a database.

CODE SMELLS

1)Array indices should be numeric
Associative arrays do allow named indices also, but if we want names indices, then using objects will be a much better option.

2)Arguments to built-in functions should match documented types
The JS documentation has functions and tells us the type of the arguments it takes. The type should match with the documented one, else we might get unexpected results.

3)Function parameters with default values should be last
If default values are not stored at the ending, users can't take advantage of them as, they will still have to specify them so that they can assign values to the arguments after them.
For example,

Function func(a=1, b)
To call the above function, we need to again pass value for a also.
Hence it is recommended to put it like func(b,a=1)

4)Dead stores should be removed
dead store happens when a local variable is assigned a value that is not read by any subsequent instruction or when an object property is assigned a value that is not subsequently used. Calculating or retrieving a value only to then overwrite it or throw it away, could indicate a serious error in the code. Even if it's not an error, it is at best a waste of resources. Therefore all calculated values should be used.

5)"strict" mode should be used with caution
Even though it may be a good practice to enforce JavaScript strict mode, doing so could result in unexpected behaviors on browsers that do not support it yet. Using this feature should therefore be done with caution and with full knowledge of the potential consequences on browsers that do not support it.

6)Control structures should use curly braces
Even though there is only one statement in the control structure, braces should be used because in the future, if any statements are required to add into the control structure, and if they are added and the fact that braces are not present is not noticed then this might cause serious problems and might be difficult to detect.

7)"===" and "!==" should be used instead of "==" and "!="
The == and != operators do type coercion before comparing values. This is bad because it can mask type errors. For example, it evaluates ' \t\r\n' == 0 as true.
It is best to always use the side-effect-less === and !== operators instead.

8)Functions should always return the same type
Unlike strongly typed languages, JavaScript does not enforce a return type on a function. This means that different paths through a function can return different types of values, which can be very confusing to the user and significantly harder to maintain.

9)An open curly brace should be located at the end of a line
Shared naming conventions allow teams to collaborate effectively. This rule raises an issue when an open curly brace is not placed at the end of a line of code. This also helps in reducing merge conflicts.

10)"undefined" should not be assigned
Variables which are not initialized will be given undefined. If we assign undefined to a variable, then it becomes difficult to differentiate between variable whose value is assigned as undefined and variable which has not been initialized.