sonar RULES Products

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- C#
- CSS
- Flex
- Go
- HTML
- Java
- JavaScript
- Kotlin
- Objective C
- PHP
- PL/I
- PL/SQL
- Python
- RPG
- Ruby
- Scala
- Swift
- Terraform
- Text
- TypeScript
- T-SQL
- VB.NET
- VB6
- XML

TypeScript static code analysis

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

All rules 279 | Vulnerability 27 | Bug 51 | Security Hotspot 43 | Code Smell 158 | Quick Fix 50

Tags | Search by name...

added
Code Smell

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

Non-null assertions should not be used
Code Smell

"undefined" should not be assigned
Code Smell

Trailing commas should not be used
Code Smell

Array constructors should not be used
Code Smell

Quotes for string literals should be used consistently
Code Smell

Statements should end with semicolons
Code Smell

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

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

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

Lines should not end with trailing whitespaces
Code Smell

Allowing confidential information to be logged is security-sensitive

Analyze your code

Security Hotspot Minor cwe privacy owasp

Log management is an important topic, especially for the security of a web application, to ensure user activity, including potential attackers, is recorded and available for an analyst to understand what's happened on the web application in case of malicious activities.

Retention of specific logs for a defined period of time is often necessary to comply with regulations such as GDPR, PCI DSS and others. However, to protect user's privacy, certain informations are forbidden or strongly discouraged from being logged, such as user passwords or credit card numbers, which obviously should not be stored or at least not in clear text.

Ask Yourself Whether

In a production environment:

- The web application uses confidential information and logs a significant amount of data.
- Logs are externalized to SIEM or Big Data repositories.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

Loggers should be configured with a list of confidential, personal information that will be hidden/masked or removed from logs.

Sensitive Code Example

With Signale log management framework the code is sensitive when an empty list of secrets is defined:

```
const { Signale } = require('signale');

const CREDIT_CARD_NUMBERS = fetchFromWebForm()
// here we suppose the credit card numbers are retrieved som

const options = {
  secrets: []         // empty list of secrets
};

const logger = new Signale(options); // Sensitive

CREDIT_CARD_NUMBERS.forEach(function(CREDIT_CARD_NUMBER) {
  logger.log('The customer ordered products with the credit
});
```

Compliant Solution

With Signale log management framework it is possible to define a list of secrets that will be hidden in logs:

Files should contain an empty newline at the end

Code Smell

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

Code Smell

Tabulation characters should not be used

Code Smell

Function and method names should comply with a naming convention

Code Smell

```
const { Signale } = require('signale');

const CREDIT_CARD_NUMBERS = fetchFromWebForm()
// here we suppose the credit card numbers are retrieved som

const options = {
  secrets: ["([0-9]{4}-?)+"]
};

const logger = new Signale(options); // Compliant

CREDIT_CARD_NUMBERS.forEach(function(CREDIT_CARD_NUMBER) {
  logger.log('The customer ordered products with the credit
});
```

See

- OWASP Top 10 2021 Category A9 - Security Logging and Monitoring Failures
- MITRE, CWE-532 - Insertion of Sensitive Information into Log File
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure

Available In:

sonarcloud sonarqube