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...

return anything should not be used
Bug

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

Generators should "yield" something
Bug

"new" operators should be used with functions
Bug

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

"NaN" should not be used in comparisons
Bug

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

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

Special identifiers should not be bound or assigned
Bug

Values should not be uselessly incremented
Bug

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

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

Using weak hashing algorithms is security-sensitive

Analyze your code

Security Hotspot Critical cwe spring owasp sans-top25

Cryptographic hash algorithms such as `MD2`, `MD4`, `MD5`, `MD6`, `HAVAL-128`, `HMAC-MD5`, `DSA` (which uses `SHA-1`), `RIPEMD`, `RIPEMD-128`, `RIPEMD-160`, `HMACRIPEMD160` and `SHA-1` are no longer considered secure, because it is possible to have `collisions` (little computational effort is enough to find two or more different inputs that produce the same hash).

Ask Yourself Whether

The hashed value is used in a security context like:

- User-password storage.
- Security token generation (used to confirm e-mail when registering on a website, reset password, etc ...).
- To compute some message integrity.

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

Recommended Secure Coding Practices

Safer alternatives, such as `SHA-256`, `SHA-512`, `SHA-3` are recommended, and for password hashing, it's even better to use algorithms that do not compute too "quickly", like `bcrypt`, `scrypt`, `argon2` or `pbkdf2` because it slows down `brute force attacks`.

Sensitive Code Example

```
const crypto = require("crypto");

const hash = crypto.createHash('sha1'); // Sensitive
```

Compliant Solution

```
const crypto = require("crypto");

const hash = crypto.createHash('sha512'); // Compliant
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- Mobile AppSec Verification Standard - Cryptography Requirements
- OWASP Mobile Top 10 2016 Category M5 - Insufficient Cryptography
- MITRE, CWE-1240 - Use of a Risky Cryptographic Primitive
- SANS Top 25 - Porous Defenses

Available In:
sonarcloud sonarqube

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

Bug

All code should be reachable

Bug

Loops with at most one iteration should be refactored

Bug

Variables should not be self-assigned

Bug