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

Python static code analysis

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

All rules 216 | Vulnerability 29 | Bug 55 | Security Hotspot 31 | Code Smell 101

Tags

Search by name...

controlling resource consumption is security-sensitive
Security Hotspot

Signalling processes is security-sensitive
Security Hotspot

Configuring loggers is security-sensitive
Security Hotspot

Using weak hashing algorithms is security-sensitive
Security Hotspot

Disabling CSRF protections is security-sensitive
Security Hotspot

Using non-standard cryptographic algorithms is security-sensitive
Security Hotspot

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

Constants should not be used as conditions
Code Smell

"SystemExit" should be re-raised
Code Smell

Bare "raise" statements should only be used in "except" blocks
Code Smell

Comparison to None should not be constant
Code Smell

"self" should be the first argument to

Silly equality checks should not be made

Analyze your code

Bug Blocker unused

In some cases a comparison with operators ==, or != will always return True or always return False. When this happens, the comparison and all its dependent code can simply be removed. This includes:

- comparing unrelated builtin types such as string and integer.
- comparing class instances which do not implement __eq__ or __ne__ to an object of a different type (builtin or from an unrelated class which also doesn't implement __eq__ or __ne__).

Noncompliant Code Example

```
foo = 1 == "1"  # Noncompliant. Always False.

foo = 1 != "1"  # Noncompliant. Always True.

class A:
    pass

myvar = A() == 1  # Noncompliant. Always False.
myvar = A() != 1  # Noncompliant. Always True.
```

Compliant Solution

```
foo = 1 == int("1")

foo = str(1) != "1"

class Eq:
    def __eq__(self, other):
        return True

myvar = Eq() == 1
myvar = 1 == Eq()
myvar = Eq() != 1  # Ok. "__ne__" calls "__eq__" by def
myvar = 1 != Eq()
```

Available In:

sonarlint sonarcloud sonarqube

"self" should be the first argument to instance methods

Code Smell

Function parameters' default values should not be modified or assigned

Code Smell

Some special methods should return "NotImplemented" instead of raising "NotImplementedError"

Code Smell

Custom Exception classes should inherit from "Exception" or one of its subclasses

Code Smell

Bare "raise" statements should not be