

- 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...



buckets is security-sensitive

Security Hotspot

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

Security Hotspot

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

Security Hotspot

Allowing both safe and unsafe HTTP methods is security-sensitive

Security Hotspot

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

Security Hotspot

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

Security Hotspot

Using hardcoded IP addresses is security-sensitive

Security Hotspot

Regular expression quantifiers and character classes should be used concisely

Code Smell

Character classes should be preferred over reluctant quantifiers in regular expressions

Code Smell

A subclass should not be in the same "except" statement as a parent class

Code Smell

Walrus operator should not make code confusing

Code Smell

**Non-empty statements should change control flow or have at least one side-effect**

Analyze your code

Bug Major ? cwe unused

Any statement, other than a pass, ... (ellipsis) or an empty statement (i.e. a single semicolon ";"), which has no side effect and does not result in a change of control flow will normally indicate a programming error, and therefore should be refactored.

### Noncompliant Code Example

```
a == 1 # Noncompliant; was assignment intended?
a < b # Noncompliant; have we forgotten to assign the r
```

### Exceptions

#### Strings

Some projects use string literals as comments. By default, this rule will not raise an issue on these strings. Reporting on string literals can be enabled by setting the rule parameter "reportOnStrings" to "true".

```
class MyClass:
    myattr = 42
    """This is an attribute""" # Noncompliant by default
```

### Operators

By default, this rule considers that no arithmetic operator has a side effect. Some rare projects redefine operators and add a side effect. You can list such operators in the rule parameter "ignoredOperators".


```
def process(p, beam):
    """
    Apache Beam redefines "|" and ">>" operators and thus
    Thus for Apache Beam projects "ignoredOperators" should be
    """
    p | "create" >> beam.Create() # Noncompliant by default
```

### See


- [MITRE, CWE-482](#) - Comparing instead of Assigning

Available In:


sonarlint | sonarcloud | sonarqube

 Code Smell


**Jump statements should not be redundant**

 Code Smell


**"pass" should not be used needlessly**

 Code Smell

**"except" clauses should do more than raise the same issue**

 Code Smell

**Boolean checks should not be inverted**

 Code Smell

**Unused local variables should be**

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.  
[Privacy Policy](#)