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...

Bug

"=+" should not be used instead of "+="

Bug

Increment and decrement operators should not be used

Bug

Return values from functions without side effects should not be ignored

Bug

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

Bug

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

All "except" blocks should be able to catch exceptions

Bug

Constructing arguments of system commands from user input is security-sensitive

Security Hotspot

Disabling auto-escaping in template engines is security-sensitive

Exceptions' "__cause__" should be either an Exception or None

Analyze your code

Bug Critical

Exception chaining enables users to see if an exception was triggered by another exception (see PEP-3134). Exceptions are chained using either of the following syntax:

- `raise NewException() from chained_exception`
- `new_exception.__cause__ = chained_exception`

It is also possible to erase a chaining by setting `new_exception.__cause__ = None` or using `except ... from None` (see PEP-409).

Chaining will fail and raise a `TypeError` if something else than `None` or a valid exception, i.e. an instance of `BaseException` or of a subclass, is provided.

Noncompliant Code Example

```
class A:
    pass

try:
    raise ValueError("orig")
except ValueError as e:
    new_exc = TypeError("new")
    new_exc.__cause__ = A()  # Noncompliant
    raise new_exc

try:
    raise ValueError("orig")
except ValueError as e:
    raise TypeError("new") from "test"  # Noncompliant
```

Compliant Solution

```
try:
    raise ValueError("orig")
except ValueError as e:
    new_exc = TypeError("new")
    new_exc.__cause__ = None  # Ok
    raise new_exc

try:
    raise ValueError("orig")
except ValueError as e:
    new_exc = TypeError("new")
    new_exc.__cause__ = e  # Ok
    raise new_exc
```

Security Hotspot

Setting loose POSIX file permissions is security-sensitive

Security Hotspot

Formatting SQL queries is security-sensitive

Security Hotspot

Character classes in regular expressions should not contain only one character

Code Smell

Superfluous curly brace quantifiers should be avoided

Code Smell

```
try:
    raise ValueError("orig")
except ValueError as e:
    raise TypeError("new") from None  # Ok

try:
    raise ValueError("orig")
except ValueError as e:
    raise TypeError("new") from e  # Ok
```

See

- PEP 3134 – Exception Chaining and Embedded Tracebacks
- PEP 409 – Suppressing exception context
- PEP 352 - Required Superclass for Exceptions
- Python documentation - Built-in Exceptions

Available In:

sonarlint | sonarcloud | sonarqube