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...

Vulnerability

Weak SSL/TLS protocols should not be used
Vulnerability

Cipher Block Chaining IVs should be unpredictable
Vulnerability

Regular expressions should not be vulnerable to Denial of Service attacks
Vulnerability

Hashes should include an unpredictable salt
Vulnerability

Regex lookahead assertions should not be contradictory
Bug

Regex boundaries should not be used in a way that can never be matched
Bug

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

"break" and "continue" should not be used outside a loop
Bug

Break, continue and return statements should not occur in "finally" blocks
Bug

Allowing public ACLs or policies on a S3 bucket is security-sensitive
Security Hotspot

Using publicly writable directories is security-sensitive

Iterable unpacking, "for-in" loops and "yield from" should use an Iterable object

Analyze your code

Bug Blocker

For-in loops, `yield from` and iterable unpacking only work with iterable objects. In order to be iterable, an object should have either an `__iter__` method or a `__getitem__` method implementing the Sequence semantic.

Note also that iterating over an asynchronous iterable, i.e. an object having the `__aiter__` method, requires the use of `async for ... in` instead of `for ... in`.

This rule raises an issue when a non iterable object is used in a `for-in` loop, in a `yield from` or when it is unpacked.

Noncompliant Code Example

```
class Empty:
    pass

empty = Empty()

for a in empty:  # Noncompliant
    print(a)

a, b, c = empty  # Noncompliant

print(*empty)  # Noncompliant

[1, 2, 3, *empty]  # Noncompliant

# yield from
def generator():
    yield from Empty()  # Noncompliant

# async generators
async def async_generator():
    yield 1

a, *rest = async_generator()  # Noncompliant
for a in async_generator():  # Noncompliant; "async" is
    print(a)
```

Compliant Solution

```
class MyIterable:
    def __init__(self, values):
        self._values = values

    def __iter__(self):
        return iter(self._values)
```

Security Hotspot

Using clear-text protocols is security-sensitive

Security Hotspot

Expanding archive files without controlling resource consumption is security-sensitive

Security Hotspot

Signalling processes is security-sensitive

Security Hotspot

Configuring loggers is security-sensitive

Security Hotspot

```
my_iterable = MyIterable(range(10))

for a in my_iterable:
    print(a)

a, b, *c = my_iterable

print(*my_iterable)

[1, 2, 3, *my_iterable]

# yield from
def generator():
    yield from subgenerator()

def subgenerator():
    yield 1

# async generators
async def async_generator():
    yield 1

async for a in async_generator():
    print(a)
```

See

- PEP 234 - Iterators
- Python documentation - Iterator Types

Available In:

sonarlint | sonarcloud | sonarqube