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...

LDAP connections should be authenticated
Vulnerability

Cryptographic key generation should be based on strong parameters
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

Operators should be used on compatible types

Analyze your code

Bug | Blocker

Calling an operator in python is equivalent to calling a special method (except for the identity operator `is`). Python provides a set of built-in operations. It is for example possible to add two integers: `1 + 2`. It is however not possible to add a string and an integer: `1 + "2"` and such an operation will raise a `TypeError`.

It is possible to define how an operator will behave with a custom class by defining the corresponding special method. See python documentation for a complete list of operators and their methods: arithmetic and bitwise operators, comparison operators.

For symmetrical binary operators you need to define two methods so that the order of operands doesn't matter, ex: `__add__` and `__radd__`.

This rule raises an issue when an operator is used on incompatible types. Types are considered incompatible if no built-in operations between those types exist and none of the operands has implemented the corresponding special methods.

Noncompliant Code Example

```
class Empty:
    pass

class Add:
    def __add__(self, other):
        return 42


1 + 2
1 + "2"  # Noncompliant
Empty() + 1  # Noncompliant
Add() + 1
1 + Add()  # Noncompliant
Add() + Empty()
Empty() + Add()  # Noncompliant
```

Compliant Solution

```
class Empty:
    pass

class Add:
    def __add__(self, other):
        return 42

    def __radd__(self, other):
        return 42
```

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

Security Hotspot

Using publicly writable directories is security-sensitive

Security Hotspot

Using clear-text protocols is security-sensitive

Security Hotspot

Expanding archive files without controlling resource consumption is security-sensitive

Security Hotspot

```
Add() + 1
1 + Add()
Add() + Empty()
Empty() + Add()
```

See

- Python documentation - Rich comparison methods
- Python documentation - Emulating numeric types

Available In:

sonarlint | sonarcloud | sonarqube