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...

JWT should be signed and verified
Vulnerability

Cipher algorithms should be robust
Vulnerability

Encryption algorithms should be used with secure mode and padding scheme
Vulnerability

Server hostnames should be verified during SSL/TLS connections
Vulnerability

Insecure temporary file creation methods should not be used
Vulnerability

Server certificates should be verified during SSL/TLS connections
Vulnerability

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

Caught Exceptions must derive from BaseException

Analyze your code

Bug | Blocker | python3

In Python 3, attempting to catch in an `except` statement an object which does not derive from BaseException will raise a `TypeError`. In Python 2 it is possible to raise old-style classes but this shouldn't be done anymore in order to be compatible with Python 3.

In order to catch multiple exceptions in an `except` statement, a tuple of exception classes should be provided.

If you are about to create a custom Exception class, note that custom exceptions should inherit from `Exception`, not `BaseException`. `Exception` allows people to catch all exceptions except the ones explicitly asking the interpreter to stop, such as `KeyboardInterrupt` and `GeneratorExit` which is not an error. See PEP 352 for more information.

This rule raises an issue when the expression used in an `except` statement is not a class deriving from `BaseException` nor a tuple of such classes.

Noncompliant Code Example

```
class CustomException:
    """An Invalid exception class."""

try:
    "a string" * 42
except CustomException:  # Noncompliant
    print("exception")
except (None, list()):  # Noncompliant * 2
    print("exception")

try:
    "a string" * 42
except [TypeError, ValueError]:  # Noncompliant. Lists
    print("exception")
except {TypeError, ValueError}:  # Noncompliant. Sets a
    print("exception")
```

Compliant Solution

```
class MyError(Exception):
    pass

try:
    "a string" * 42
except (MyError, TypeError):
    print("exception")
```

See

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

- Python documentation - Errors and Exceptions
- Python documentation - the `try` statement
- PEP 352 - Required Superclass for Exceptions

Available In:

sonarlint | sonarcloud | sonarqube