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

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
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

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

Analyze your code

Bug | Critical | cwe error-handling

Using `return`, `break` or `continue` in a `finally` block suppresses the propagation of any unhandled exception which was raised in the `try`, `else` or `except` blocks. It will also ignore their return statements.

SystemExit is raised when `sys.exit()` is called. KeyboardInterrupt is raised when the user asks the program to stop by pressing interrupt keys. Both exceptions are expected to propagate up until the application stops. It is ok to catch them when a clean-up is necessary but they should be raised again immediately. They should never be ignored.

If you need to ignore every other exception you can simply catch the `Exception` class. However you should be very careful when you do this as it will ignore other important exceptions such as MemoryError

In python 2 it is possible to raise old style classes. You can use a bare `except:` statement to catch every exception. Remember to still reraise `SystemExit` and `KeyboardInterrupt`.

This rule raises an issue when a jump statement (`break`, `continue`, `return`) would force the control flow to leave a finally block.

Noncompliant Code Example

```
def find_file_which_contains(expected_content, paths):
    file = None
    for path in paths:
        try:
            # "open" will raise IsADirectoryError if the pro
            file = open(path, 'r')
            actual_content = file.read()
        except FileNotFoundError as exception:
            # This exception will never pass the "finally" b
            raise ValueError(f"'paths' should only contain e
        finally:
            file.close()
            if actual_content != expected_content:
                # Note that "continue" is allowed in a "fina
                continue  # Noncompliant. This will prevent
            else:
                return path # Noncompliant. Same as for "con
    return None

# This will return None instead of raising ValueError from t
find_file_which_contains("some content", ["file_which_does_n

# This will return None instead of raising IsADirectoryError
find_file_which_contains("some content", ["a_directory"])

import sys

while True:
    try:
```

Non-capturing groups without quantifier should not be used

Code Smell

Regular expressions should not contain empty groups

Code Smell

Regular expressions should not contain multiple spaces

Code Smell

Single-character alternations in regular expressions should be replaced with character classes

Code Smell

```
        sys.exit(1)
    except (SystemExit) as e:
        print("Exiting")
        raise
    finally:
        break  # This will prevent SystemExit from raising

def continue_whatever_happens_noncompliant():
    for i in range(10):
        try:
            raise ValueError()
        finally:
            continue  # Noncompliant
```

Compliant Solution

```
# Note that using "with open(...) as" would be better. We ke

def find_file_which_contains(expected_content, paths):
    file = None
    for path in paths:
        try:
            file = open(path, 'r')
            actual_content = file.read()
            if actual_content != expected_content:
                continue
            else:
                return path
        except FileNotFoundError as exception:
            raise ValueError(f"'paths' should only contain e
        finally:
            if file:
                file.close()
    return None

# This raises ValueError
find_file_which_contains("some content", ["file_which_does_n

# This raises IsADirectoryError
find_file_which_contains("some content", ["a_directory"])

import sys

while True:
    try:
        sys.exit(1)
    except (SystemExit) as e:
        print("Exiting")
        raise # SystemExit is re-raised

import logging

def continue_whatever_happens_compliant():
    for i in range(10):
        try:
            raise ValueError()
        except Exception:
            logging.exception("Failed")  # Ignore all "Excep
```

See

- Python documentation - the `try` statement

Available In:

sonarlint sonarcloud sonarqube