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...

Nested blocks of code should not be left empty
Code Smell

Functions, methods and lambdas should not have too many parameters
Code Smell

Collapsible "if" statements should be merged
Code Smell

Logging should not be vulnerable to injection attacks
Vulnerability

Repeated patterns in regular expressions should not match the empty string
Bug

Function parameters initial values should not be ignored
Bug

Disabling versioning of S3 buckets is security-sensitive
Security Hotspot

Disabling server-side encryption of S3 buckets is security-sensitive
Security Hotspot

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive
Security Hotspot

Delivering code in production with debug features activated is security-sensitive
Security Hotspot

Allowing both safe and unsafe HTTP methods is security-sensitive

Wildcard imports should not be used

Analyze your code

Code Smell | Critical | pitfall

Importing every public name from a module using a wildcard (`from mymodule import *`) is a bad idea because:

- It could lead to conflicts between names defined locally and the ones imported.
- It reduces code readability as developers will have a hard time knowing where names come from.
- It clutters the local namespace, which makes debugging more difficult.

Remember that imported names can change when you update your dependencies. A wildcard import which works today might be broken tomorrow.

There are two ways to avoid a wildcard import:

- Replace it with `import mymodule` and access module members as `mymodule.myfunction`. If the module name is too long, alias it to a shorter name. Example: `import pandas as pd`
- List every imported name. If necessary import statements can be split on multiple lines using parentheses (preferred solution) or backslashes.

Noncompliant Code Example

```
from math import *  # Noncompliant
def exp(x):
    pass
print(exp(0))   # "None" will be printed
```

Compliant Solution

```
import math
def exp(x):
    pass
print(math.exp(0))   # "1.0" will be printed
```

Or

```
from math import exp as m_exp
def exp(x):
    pass
print(m_exp(0))   # "1.0" will be printed
```

Exceptions

No issue will be raised in `__init__.py` files. Wildcard imports are a common way of populating these modules.

methods is security-sensitive

Security Hotspot

Creating cookies without the "HttpOnly" flag is security-sensitive

Security Hotspot

Creating cookies without the "secure" flag is security-sensitive

Security Hotspot

Using hardcoded IP addresses is security-sensitive

Security Hotspot

Regular expression quantifiers and character classes should be used concisely

No issue will be raised in modules doing only imports. Local modules are sometimes created as a proxy for third-party modules.

```
# file: mylibrary/pyplot.py
try:
    from guiqwt.pyplot import *  # Ok
except Exception:
    from matplotlib.pyplot import *  # Ok
```

Just keep in mind that wildcard imports might still create issues in these cases. It's always better to import only what you need.

See

- Python documentation - The import statement

Available In:

sonarlint | sonarcloud | sonarqube