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...

Bug

Alternatives in regular expressions should be grouped when used with anchors

Bug

New objects should not be created only to check their identity

Bug

Collection content should not be replaced unconditionally

Bug

Exceptions should not be created without being raised

Bug

Collection sizes and array length comparisons should make sense

Bug

All branches in a conditional structure should not have exactly the same implementation

Bug

The output of functions that don't return anything should not be used

Bug

"=+" should not be used instead of "+="

Bug

Increment and decrement operators should not be used

Bug

Return values from functions without side effects should not be ignored

Bug

Related "if/else if" statements should not have the same condition

Regular expressions should not be vulnerable to Denial of Service attacks

Analyze your code

Vulnerability Critical

injection cwe owasp denial-of-service

Most of the regular expression engines use `backtracking` to try all possible execution paths of the regular expression when evaluating an input, in some cases it can cause performance issues, called `catastrophic backtracking` situations. In the worst case, the complexity of the regular expression is exponential in the size of the input, this means that a small carefully-crafted input (like 20 chars) can trigger `catastrophic backtracking` and cause a denial of service of the application. Super-linear regex complexity can lead to the same impact too with, in this case, a large carefully-crafted input (thousands chars).

It is not recommended to construct a regular expression pattern from a user-controlled input, if no other choice, sanitize the input to remove/annihilate regex metacharacters.

Noncompliant Code Example

```
from flask import request
import re

@app.route('/upload')
def upload():
    username = request.args.get('username')
    filename = request.files.get('attachment').filename

    re.search(username, filename) # Noncompliant
```

Compliant Solution

```
from flask import request
import re

@app.route('/upload')
def upload():
    username = re.escape(request.args.get('username'))
    filename = request.files.get('attachment').filename

    re.search(username, filename) # Compliant
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-400 - Uncontrolled Resource Consumption
- MITRE, CWE-1333 - Inefficient Regular Expression Complexity

🐞 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

- OWASP Regular expression Denial of Service - ReDoS

Available In:

sonarcloud sonarqube Developer Edition