

- 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... 🔍

Functions should not have too many lines of code

Code Smell

Track uses of "NOSONAR" comments

Code Smell

Track comments matching a regular expression

Code Smell

Statements should be on separate lines

Code Smell

Functions should not contain too many return statements

Code Smell

Files should not have too many lines of code

Code Smell

Lines should not be too long

Code Smell

Methods and properties that don't access instance data should be static

Code Smell

New-style classes should be used

Code Smell

Parentheses should not be used after certain keywords

Code Smell

Track "TODO" and "FIXME" comments that do not contain a reference to a person

Code Smell

Module names should comply with a naming convention

Code Smell

Using regular expressions is security-sensitive

Analyze your code

Security Hotspot Critical ⓘ

Using regular expressions is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2017-16021
- CVE-2018-13863

Evaluating regular expressions against input strings is potentially an extremely CPU-intensive task. Specially crafted regular expressions such as (a+)+s will take several seconds to evaluate the input string aaaaaaaaaaaaaaaaaaaaaaaaaaaaaabs. The problem is that with every additional a character added to the input, the time required to evaluate the regex doubles. However, the equivalent regular expression, a+s (without grouping) is efficiently evaluated in milliseconds and scales linearly with the input size.

Evaluating such regular expressions opens the door to Regular expression Denial of Service (ReDoS) attacks. In the context of a web application, attackers can force the web server to spend all of its resources evaluating regular expressions thereby making the service inaccessible to genuine users.

This rule flags any execution of a hardcoded regular expression which has at least 3 characters and at least two instances of any of the following characters: *+{.

Example: (a+)*

Ask Yourself Whether

- the executed regular expression is sensitive and a user can provide a string which will be analyzed by this regular expression.
- your regular expression engine performance decrease with specially crafted inputs and regular expressions.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

Check whether your regular expression engine (the algorithm executing your regular expression) has any known vulnerabilities. Search for vulnerability reports mentioning the one engine you're are using.

Use if possible a library which is not vulnerable to Redos Attacks such as Google Re2.

Remember also that a ReDos attack is possible if a user-provided regular expression is executed. This rule won't detect this kind of injection.


Sensitive Code Example

Django


```
from django.core.validators import RegexValidator
from django.urls import re_path

RegexValidator('(a*)*b') # Sensitive
```


Comments should not be located at the end of lines of code

 Code Smell


Lines should not end with trailing whitespaces

 Code Smell

Files should contain an empty newline at the end

 Code Smell

Long suffix "L" should be upper case

 Code Smell

```
def define_http_endpoint(view):
    re_path(r'^(a*)*b/$', view) # Sensitive
```

re module

```
import re
from re import compile, match, search, fullmatch, split, findall

input = 'input string'
replacement = 'replacement'

re.compile('(a*)*b') # Sensitive
re.match('(a*)*b', input) # Sensitive
re.search('(a*)*b', input) # Sensitive
re.fullmatch('(a*)*b', input) # Sensitive
re.split('(a*)*b', input) # Sensitive
re.findall('(a*)*b', input) # Sensitive
re.finditer('(a*)*b', input) # Sensitive
re.sub('(a*)*b', replacement, input) # Sensitive
re.subn('(a*)*b', replacement, input) # Sensitive
```

regex module

```
import regex
from regex import compile, match, search, fullmatch, split, findall

input = 'input string'
replacement = 'replacement'

regex.subf('(a*)*b', replacement, input) # Sensitive
regex.subfn('(a*)*b', replacement, input) # Sensitive
regex.splititer('(a*)*b', input) # Sensitive

regex.compile('(a*)*b') # Sensitive
regex.match('(a*)*b', input) # Sensitive
regex.search('(a*)*b', input) # Sensitive
regex.fullmatch('(a*)*b', input) # Sensitive
regex.split('(a*)*b', input) # Sensitive
regex.findall('(a*)*b', input) # Sensitive
regex.finditer('(a*)*b', input) # Sensitive
regex.sub('(a*)*b', replacement, input) # Sensitive
regex.subn('(a*)*b', replacement, input) # Sensitive
```

Exceptions

Some corner-case regular expressions will not raise an issue even though they might be vulnerable. For example: `(a|aa)+`, `(a|a?)+`.

It is a good idea to test your regular expression if it has the same pattern on both side of a `"|"`.

See

- [OWASP Top 10 2017 Category A1](#) - Injection
- [MITRE, CWE-624](#) - Executable Regular Expression Error
- OWASP Regular expression Denial of Service - ReDoS

Deprecated

This rule is deprecated; use `{rule:pythonsecurity:S2631}` instead.

Available In:

sonarcloud  **sonarqube** 