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...

`str.replace` should be preferred to `re.sub`

Analyze your code

Code Smell | Critical | regex performance

An `re.sub` call always performs an evaluation of the first argument as a regular expression, even if no regular expression features were used. This has a significant performance cost and therefore should be used with care.

When `re.sub` is used, the first argument should be a real regular expression. If it’s not the case, `str.replace` does exactly the same thing as `re.sub` without the performance drawback of the regex.

This rule raises an issue for each `re.sub` used with a simple string as first argument which doesn’t contains special regex character or pattern.

Noncompliant Code Example

```
input = "Bob is a Bird... Bob is a Plane... Bob is Supe
changed = re.sub("Bob is", "It's", input) # Noncomplian
changed = re.sub("\.\.\.", ";", changed) # Noncompliant
```

Compliant Solution

```
input = "Bob is a Bird... Bob is a Plane... Bob is Supe
changed = str.replace("Bob is", "It's", input)
changed = str.replace("...", ";", changed)
```

Or, with a regex:

```
input = "Bob is a Bird... Bob is a Plane... Bob is Supe
changed = re.sub(r"\w*\sis", "It's", input)
changed = re.sub(r"\.{3}", ";", changed)
```

Available In:
sonarlint | sonarcloud | sonarqube

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