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

HTTP response headers should not be vulnerable to injection attacks

Analyze your code

Vulnerability | Critical | injection

User-provided data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Applications constructing HTTP response headers based on tainted data could allow attackers to change security sensitive headers like Cross-Origin Resource Sharing headers.

Web application frameworks and servers might also allow attackers to inject new line characters in headers to craft malformed HTTP response. In this case the application would be vulnerable to a larger range of attacks like HTTP Response Splitting/Smuggling. Most of the time this type of attack is mitigated by default modern web application frameworks but there might be rare cases where older versions are still vulnerable.

As a best practice, applications that use user-provided data to construct the response header should always validate the data first. Validation should be based on a whitelist.

Noncompliant Code Example

Flask

```
from flask import Response, request
from werkzeug.datastructures import Headers

@app.route('/route')
def route():
    content_type = request.args["Content-Type"]
    response = Response()
    headers = Headers()
    headers.add("Content-Type", content_type) # Noncomp
    response.headers = headers
    return response
```

Django

```
import django.http

def route(request):
    content_type = request.GET.get("Content-Type")
    response = django.http.HttpResponse()
    response.__setitem__('Content-Type', content_type)
    return response
```

Compliant Solution

Flask

Code Smell

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

```
from flask import Response, request
from werkzeug.datastructures import Headers
import re

@app.route('/route')
def route():
    content_type = request.args["Content-Type"]
    allowed_content_types = r'application/(pdf|json|xml
    response = Response()
    headers = Headers()
    if re.match(allowed_content_types, content_type):
        headers.add("Content-Type", content_type)  # Co
    else:
        headers.add("Content-Type", "application/json")
    response.headers = headers
    return response
```

Django

```
import django.http
import re

def route(request):
    content_type = request.GET.get("Content-Type")
    allowed_content_types = r'application/(pdf|json|xml
    response = django.http.HttpResponse()
    if re.match(allowed_content_types, content_type):
        response.__setitem__('Content-Type', content_ty
    else:
        response.__setitem__('Content-Type', "applicati
    return response
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Attack Category - HTTP Response Splitting
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-113 - Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
- SANS Top 25 - Insecure Interaction Between Components

Deprecated

This rule is deprecated; use {rule:python:S5122}, {rule:python:S5146}, {rule:python:S6287} instead.

Available In:

sonarcloud | sonarqube Developer Edition