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...

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
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

Server-side requests should not be vulnerable to forging attacks

Analyze your code

Vulnerability   Major   injection cwe sans-top25 owasp

User-supplied data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Performing requests from user-controlled data could allow attackers to make arbitrary requests on the internal network or to change their original meaning and thus to retrieve or delete sensitive information.

The problem could be mitigated in any of the following ways:

- Validate the user-provided data, such as the URL and headers, used to construct the request.
- Redesign the application to not send requests based on user-provided data.

Noncompliant Code Example

```
from flask import request
import urllib

@app.route('/proxy')
def proxy():
    url = request.args["url"]
    return urllib.request.urlopen(url).read() # Noncomp
```

Compliant Solution

```
from flask import request
import urllib

DOMAINS_WHITELIST = ['domain1.com', 'domain2.com']

@app.route('/proxy')
def proxy():
    url = request.args["url"]
    if urllib.parse.urlparse(url).hostname in DOMAINS_W
        return urllib.request.urlopen(url).read()
```

See

- OWASP Top 10 2021 Category A10 - Server-Side Request Forgery (SSRF)
- OWASP Attack Category - Server Side Request Forgery
- OWASP Top 10 2017 Category A5 - Broken Access Control
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-641 - Improper Restriction of Names for Files and Other Resources
- MITRE, CWE-918 - Server-Side Request Forgery (SSRF)

Regular expression quantifiers and character classes should be used concisely

Code Smell

Character classes should be preferred over reluctant quantifiers in regular expressions

Code Smell

A subclass should not be in the same "except" statement as a parent class

Code Smell

Walrus operator should not make code confusing

Code Smell

- SANS Top 25 - Risky Resource Management

Available In:

sonarcloud sonarqube Developer Edition