sonar RULES

Products

- 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

Instance and class methods should have at least one positional parameter
Bug

Boolean expressions of exceptions should not be used in "except" statements
Bug

Caught Exceptions must derive from BaseException
Bug

Item operations should be done on objects supporting them
Bug

Raised Exceptions must derive from BaseException
Bug

Operators should be used on compatible types
Bug

Function arguments should be passed only once
Bug

Iterable unpacking, "for-in" loops and "yield from" should use an Iterable object
Bug

Variables, classes and functions should be defined before being used
Bug

Identity operators should not be used with dissimilar types
Bug

Only strings should be listed in

I/O function calls should not be vulnerable to path injection attacks

Analyze your code

Vulnerability Blocker injection cwe owasp sans-top25

User-provided data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Constructing file system paths directly from tainted data could enable an attacker to inject specially crafted values, such as `'../'`, that change the initial path and, when accessed, resolve to a path on the filesystem where the user should normally not have access.

A successful attack might give an attacker the ability to read, modify, or delete sensitive information from the file system and sometimes even execute arbitrary operating system commands. This is often referred to as a "path traversal" or "directory traversal" attack.

The mitigation strategy should be based on the whitelisting of allowed paths or characters.

Noncompliant Code Example

```
from flask import request, send_file

@app.route('/download')
def download():
    file = request.args['file']
    return send_file("static/%s" % file, as_attachment=
```

Compliant Solution

```
from flask import request, send_from_directory

@app.route('/download')
def download():
    file = request.args['file']
    return send_from_directory('static', file) # Compli
```

See

- OWASP Top 10 2021 Category A1 - Broken Access Control
- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- OWASP Top 10 2017 Category A5 - Broken Access Control
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-22 - Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- MITRE, CWE-99 - Improper Control of Resource Identifiers ('Resource Injection')

__all__

Bug

"__init__" should not return a value

Bug

"yield" and "return" should not be used outside functions

Bug

String formatting should not lead to runtime errors

Bug

Recursion should not be infinite

Bug

- MITRE, CWE-641 - Improper Restriction of Names for Files and Other Resources
- SANS Top 25 - Risky Resource Management

Available In:

sonarcloud | sonarqube Developer Edition