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...

Function return types should be consistent with their type hint
Code Smell

Character classes in regular expressions should not contain the same character twice
Code Smell

Type checks shouldn't be confusing
Code Smell

Regular expressions should not be too complicated
Code Smell

Builtins should not be shadowed by local variables
Code Smell

Implicit string and byte concatenations should not be confusing
Code Smell

Identity comparisons should not be used with cached typed
Code Smell

Expressions creating sets should not have duplicate values
Code Smell

Expressions creating dictionaries should not have duplicate keys
Code Smell

Special method "__exit__" should not re-raise the provided exception
Code Smell

Unused scope-limited definitions should be removed
Code Smell

Functions and methods should not

Using pseudorandom number generators (PRNGs) is security-sensitive

Analyze your code

Security Hotspot | Critical | cwe owasp

Using pseudorandom number generators (PRNGs) is security-sensitive. For example, it has led in the past to the following vulnerabilities:

- CVE-2013-6386
- CVE-2006-3419
- CVE-2008-4102

When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information.

Ask Yourself Whether

- the code using the generated value requires it to be unpredictable. It is the case for all encryption mechanisms or when a secret value, such as a password, is hashed.
- the function you use generates a value which can be predicted (pseudo-random).
- the generated value is used multiple times.
- an attacker can access the generated value.

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

Recommended Secure Coding Practices

- Only use random number generators which are recommended by OWASP or any other trusted organization.
- Use the generated random values only once.
- You should not expose the generated random value. If you have to store it, make sure that the database or file is secure.

Sensitive Code Example

```
import random

random.getrandbits(1) # Sensitive
random.randint(0,9) # Sensitive
random.random()  # Sensitive

# the following functions are sadly used to generate salt by
random.sample(['a', 'b'], 1)  # Sensitive
random.choice(['a', 'b'])  # Sensitive
random.choices(['a', 'b'])  # Sensitive
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- Mobile AppSec Verification Standard - Cryptography Requirements
- OWASP Mobile Top 10 2016 Category M5 - Insufficient Cryptography

Functions and methods should not have identical implementations

Code Smell

Unused private nested classes should be removed

Code Smell

String formatting should be used correctly

Code Smell

Conditional expressions should not be nested

Code Smell

- MITRE, CWE-338 - Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
- MITRE, CWE-330 - Use of Insufficiently Random Values
- MITRE, CWE-326 - Inadequate Encryption Strength
- MITRE, CWE-1241 - Use of Predictable Algorithm in Random Number Generator
- Derived from FindSecBugs rule Predictable Pseudo Random Number Generator

Available In:

sonarcloud | sonarqube