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

New objects should not be created only to check their identity
Bug

Collection content should not be replaced unconditionally
Bug

Exceptions should not be created without being raised
Bug

Collection sizes and array length comparisons should make sense
Bug

All branches in a conditional structure should not have exactly the same implementation
Bug

The output of functions that don't return anything should not be used
Bug

"=+" should not be used instead of "+="
Bug

Increment and decrement operators should not be used
Bug

Return values from functions without side effects should not be ignored
Bug

Related "if/else if" statements should not have the same condition
Bug

Identical expressions should not be used on both sides of a binary operator
Bug

Hashes should include an unpredictable salt

Analyze your code

Vulnerability | Critical | cwe sans-top25 owasp

In cryptography, a "salt" is an extra piece of data which is included when hashing a password. This makes `rainbow-table attacks` more difficult. Using a cryptographic hash function without an unpredictable salt increases the likelihood that an attacker could successfully find the hash value in databases of precomputed hashes (called `rainbow-tables`).

This rule raises an issue when a hashing function which has been specifically designed for hashing passwords, such as `PBKDF2`, is used with a non-random, reused or too short salt value. It does not raise an issue on base hashing algorithms such as `sha1` or `md5` as they should not be used to hash passwords.

Recommended Secure Coding Practices

- Use hashing functions generating their own secure salt or generate a secure random value of at least 16 bytes.
- The salt should be unique by user password.

Noncompliant Code Example

hashlib

```
import crypt
from hashlib import pbkdf2_hmac

hash = pbkdf2_hmac('sha256', password, b'D8VxSmTZt2E2YV454mk
```

crypt

```
hash = crypt.crypt(password)         # Noncompliant: salt is
```

Compliant Solution

hashlib

```
import crypt
from hashlib import pbkdf2_hmac

salt = os.urandom(32)
hash = pbkdf2_hmac('sha256', password, salt, 100000)    # Co
```

crypt

```
salt = crypt.mksalt(crypt.METHOD_SHA256)
hash = crypt.crypt(password, salt)         # Compliant
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures

All code should be reachable

Bug

Loops with at most one iteration should be refactored

Bug

Variables should not be self-assigned

Bug

All "except" blocks should be able to catch exceptions

Bug

- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- MITRE, CWE-759 - Use of a One-Way Hash without a Salt
- MITRE, CWE-760 - Use of a One-Way Hash with a Predictable Salt
- SANS Top 25 - Porous Defenses

Available In:

sonarlint | sonarcloud | sonarqube