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...

be of an expected type
Code Smell

`str.replace` should be preferred to `re.sub`
Code Smell

Unread "private" attributes should be removed
Code Smell

Cognitive Complexity of functions should not be too high
Code Smell

The first argument to class methods should follow the naming convention
Code Smell

Method overrides should not change contracts
Code Smell

Wildcard imports should not be used
Code Smell

String literals should not be duplicated
Code Smell

Functions and methods should not be empty
Code Smell

Server-side requests should not be vulnerable to forging attacks
Vulnerability

Non-empty statements should change control flow or have at least one side-effect
Bug

Replacement strings should reference existing regular expression groups
Bug

Encryption algorithms should be used with secure mode and padding scheme

Analyze your code

Vulnerability | Critical | cwe privacy owasp sans-top25

Encryption operation mode and the padding scheme should be chosen appropriately to guarantee data confidentiality, integrity and authenticity:

- For block cipher encryption algorithms (like AES):
  - The GCM (Galois Counter Mode) mode which works internally with zero/no padding scheme, is recommended, as it is designed to provide both data authenticity (integrity) and confidentiality. Other similar modes are CCM, CWC, EAX, IAPM and OCB.
  - The CBC (Cipher Block Chaining) mode by itself provides only data confidentiality, it's recommended to use it along with Message Authentication Code or similar to achieve data authenticity (integrity) too and thus to prevent padding oracle attacks.
  - The ECB (Electronic Codebook) mode doesn't provide serious message confidentiality: under a given key any given plaintext block always gets encrypted to the same ciphertext block. This mode should not be used.
- For RSA encryption algorithm, the recommended padding scheme is OAEP.

Noncompliant Code Example

pycryptodomex library:

```
from Cryptodome.Cipher import AES, PKCS1_OAEP,  PKCS1_v1_5
from Cryptodome.Random import get_random_bytes
from Cryptodome.PublicKey import RSA

# Example for a symmetric cipher: AES
AES.new(key, AES.MODE_ECB)  # Noncompliant
AES.new(key, AES.MODE_CBC)  # Noncompliant

# Example for a asymmetric cipher: RSA
cipher = PKCS1_v1_5.new(key) # Noncompliant
```

pyca library:

```
import os
from cryptography.hazmat.primitives.ciphers import Cipher, a
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa, p
from cryptography.hazmat.primitives import hashes

# Example for a symmetric cipher: AES
aes = Cipher(algorithms.AES(key), modes.CBC(iv), backend=def
aes = Cipher(algorithms.AES(key), modes.ECB(), backend=defau

# Example for a asymmetric cipher: RSA
ciphertext = public_key.encrypt(
  message,
  padding.PKCS1v15() # Noncompliant
)

plaintext = private_key.decrypt(
```

Alternation in regular expressions should not contain empty alternatives

Bug

Unicode Grapheme Clusters should be avoided inside regex character classes

Bug

Regex alternatives should not be redundant

Bug

Alternatives in regular expressions should be grouped when used with anchors

```
  ciphertext,
  padding.PKCS1v15() # Noncompliant
)
```

pydes library:

```
# For DES cipher
des = pyDes.des('ChangeIt') # Noncompliant
des = pyDes.des('ChangeIt', pyDes.CBC, "\0\0\0\0\0\0\0\0", p
des = pyDes.des('ChangeIt', pyDes.ECB, "\0\0\0\0\0\0\0\0", p
```

pycrypto library is not maintained and therefore should not be used:

```
# https://pycrypto.readthedocs.io/en/latest/
from Crypto.Cipher import *
from Crypto.Random import get_random_bytes
from Crypto.Util import Counter
from Crypto.PublicKey import RSA

# Example for a symmetric cipher: AES
AES.new(key, AES.MODE_ECB)  # Noncompliant
AES.new(key, AES.MODE_CBC, IV=iv)  # Noncompliant

# Example for a asymmetric cipher: RSA
cipher = PKCS1_v1_5.new(key) # Noncompliant
```

Compliant Solution

pycryptodomex library:

```
from Cryptodome.Cipher import AES
from Cryptodome.Random import get_random_bytes
from Cryptodome.PublicKey import RSA

# AES is the recommended symmetric cipher with GCM mode
AES.new(key, AES.MODE_GCM)  # Compliant

# RSA is the recommended asymmetric cipher with OAEP padding
cipher = PKCS1_OAEP.new(key) # Compliant
```

pyca library:

```
import os
from cryptography.hazmat.primitives.ciphers import Cipher, a
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa, p
from cryptography.hazmat.primitives import hashes

# AES is the recommended symmetric cipher with GCM mode
aes = Cipher(algorithms.AES(key), modes.GCM(iv), backend=def

# RSA is the recommended asymmetric cipher with OAEP padding
ciphertext = public_key.encrypt(
  message,
  padding.OAEP( # Compliant
    mgf=padding.MGF1(algorithm=hashes.SHA256()),
    algorithm=hashes.SHA256(),
    label=None
  )
)

plaintext = private_key.decrypt(
  ciphertext,
  padding.OAEP( # Compliant
    mgf=padding.MGF1(algorithm=hashes.SHA256()),
    algorithm=hashes.SHA256(),
    label=None
  )
)
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- MITRE, CWE-327 - Use of a Broken or Risky Cryptographic Algorithm
- SANS Top 25 - Porous Defenses

Available In:

sonarlint | sonarcloud | sonarqube