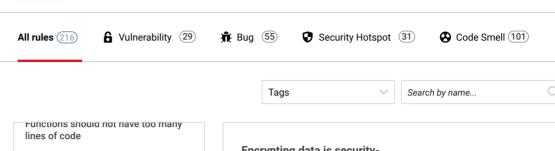


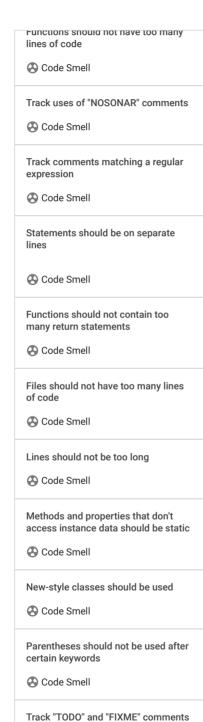




# Python static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your PYTHON code





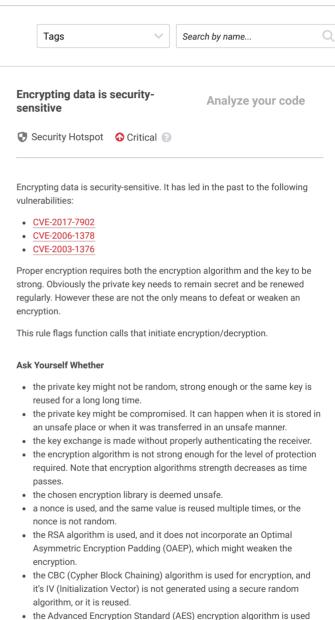
that do not contain a reference to a

Module names should comply with a

person

Code Smell

naming convention



## Recommended Secure Coding Practices

information

Generate encryption keys using secure random algorithms.

You are at risk if you answered yes to any of those questions.

When generating cryptographic keys (or key pairs), it is important to use
a key length that provides enough entropy against brute-force attacks.
 For the Blowfish algorithm the key should be at least 128 bits long, while
for the RSA algorithm it should be at least 2048 bits long.

with an unsecure mode. See the recommended practices for more

- Regenerate the keys regularly.
- Always store the keys in a safe location and transfer them only over safe channels.
- If there is an exchange of cryptographic keys, check first the identity of the receiver.

A Code Smell

Comments should not be located at the end of lines of code

Code Smell

Lines should not end with trailing whitespaces

A Code Smell

Files should contain an empty newline at the end

Code Smell

Long suffix "L" should be upper case

Code Smell

- Only use strong encryption algorithms. Check regularly that the algorithm
  is still deemed secure. It is also imperative that they are implemented
  correctly. Use only encryption libraries which are deemed secure. Do not
  define your own encryption algorithms as they will most probably have
  flaws.
- · When a nonce is used, generate it randomly every time.
- When using the RSA algorithm, incorporate an Optimal Asymmetric Encryption Padding (OAEP).
- When CBC is used for encryption, the IV must be random and unpredictable. Otherwise it exposes the encrypted value to crypto-analysis attacks like "Chosen-Plaintext Attacks". Thus a secure random algorithm should be used. An IV value should be associated to one and only one encryption cycle, because the IV's purpose is to ensure that the same plaintext encrypted twice will yield two different ciphertexts.
- The Advanced Encryption Standard (AES) encryption algorithm can be used with various modes. Galois/Counter Mode (GCM) with no padding should be preferred to the following combinations which are not secured:
  - Electronic Codebook (ECB) mode: Under a given key, any given
    plaintext block always gets encrypted to the same ciphertext block.
    Thus, it does not hide data patterns well. In some senses, it doesn't
    provide serious message confidentiality, and it is not recommended
    for use in cryptographic protocols at all.
  - Cipher Block Chaining (CBC) with PKCS#5 padding (or PKCS#7) is susceptible to padding oracle attacks.

### Sensitive Code Example

cryptography module

```
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives.ciphers.aead import
from cryptography.hazmat.primitives.asymmetric import r
from cryptography.hazmat.primitives.ciphers import Ciph

def encrypt(key):
    Fernet(key) # Sensitive
    ChaCha20Poly1305(key) # Sensitive
    AESGCM(key) # Sensitive
    AESCCM(key) # Sensitive

private_key = rsa.generate_private_key() # Sensitive

def encrypt2(algorithm, mode, backend):
    Cipher(algorithm, mode, backend) # Sensitive
```

## pynacl library

```
from nacl.public import Box
from nacl.secret import SecretBox

def public_encrypt(secret_key, public_key):
    Box(secret_key, public_key) # Sensitive

def secret_encrypt(key):
    SecretBox(key) # Sensitive
```

## See

- OWASP Top 10 2017 Category A3 Sensitive Data Exposure
- OWASP Top 10 2017 Category A6 Security Misconfiguration
- MITRE, CWE-321 Use of Hard-coded Cryptographic Key
- MITRE, CWE-322 Key Exchange without Entity Authentication
- $\bullet \;\; \underline{\text{MITRE, CWE-323}}$  Reusing a Nonce, Key Pair in Encryption
- MITRE, CWE-324 Use of a Key Past its Expiration Date
   MITRE ONLY 2015
- <u>MITRE, CWE-325</u> Missing Required Cryptographic Step
- MITRE, CWE-326 Inadequate Encryption Strength
- MITRE, CWE-327 Use of a Broken or Risky Cryptographic Algorithm
- SANS Top 25 Porous Defenses

## Deprecated

This rule is deprecated; use {rule:python:S4426}, {rule:python:S5542}, {rule:python:S5547} instead.

Available In:



sonarcloud 🔕 | sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy