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...

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

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
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

Server certificates should be verified during SSL/TLS connections

Analyze your code

Vulnerability | Critical | cwe privacy owasp ssl

Validation of X.509 certificates is essential to create secure SSL/TLS sessions not vulnerable to man-in-the-middle attacks.

The certificate chain validation includes these steps:

- The certificate is issued by its parent Certificate Authority or the root CA trusted by the system.
- Each CA is allowed to issue certificates.
- Each certificate in the chain is not expired.

It's not recommended to reinvent the wheel by implementing custom certificate chain validation.

TLS libraries provide built-in certificate validation functions that should be used.

Noncompliant Code Example

psf/requests library:

```
import requests

requests.request('GET', 'https://example.domain', verify=Fal
requests.get('https://example.domain', verify=False) # Nonco
```

Python ssl standard library:

```
import ssl

ctx1 = ssl._create_unverified_context() # Noncompliant: by d
ctx2 = ssl._create_stdlib_context() # Noncompliant: by defau

ctx3 = ssl.create_default_context()
ctx3.verify_mode = ssl.CERT_NONE # Noncompliant
```

pyca/pyopenssl library:

```
from OpenSSL import SSL

ctx1 = SSL.Context(SSL.TLSv1_2_METHOD) # Noncompliant: by de

ctx2 = SSL.Context(SSL.TLSv1_2_METHOD)
ctx2.set_verify(SSL.VERIFY_NONE, verify_callback) # Noncompl
```

Compliant Solution

psf/requests library:

```
import requests

requests.request('GET', 'https://example.domain', verify=Tru
```

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

```
requests.request('GET', 'https://example.domain', verify='/p
requests.get(url='https://example.domain') # by default cert
```

Python ssl standard library:

```
import ssl

ctx = ssl.create_default_context()
ctx.verify_mode = ssl.CERT_REQUIRED

ctx = ssl._create_default_https_context() # by default certi
```

pyca/pyopenssl library:

```
from OpenSSL import SSL

ctx = SSL.Context(SSL.TLSv1_2_METHOD)
ctx.set_verify(SSL.VERIFY_PEER, verify_callback) # Compliant
ctx.set_verify(SSL.VERIFY_PEER | SSL.VERIFY_FAIL_IF_NO_PEER_
ctx.set_verify(SSL.VERIFY_PEER | SSL.VERIFY_FAIL_IF_NO_PEER_
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2021 Category A5 - Security Misconfiguration
- OWASP Top 10 2021 Category A7 - Identification and Authentication Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- Mobile AppSec Verification Standard - Network Communication Requirements
- OWASP Mobile Top 10 2016 Category M3 - Insecure Communication
- MITRE, CWE-295 - Improper Certificate Validation

Available In:

sonarlint | sonarcloud | sonarqube