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 should not have too many lines of code
Code Smell

Track uses of "NOSONAR" comments
Code Smell

Track comments matching a regular expression
Code Smell

Statements should be on separate lines
Code Smell

Functions should not contain too many return statements
Code Smell

Files should not have too many lines of code
Code Smell

Lines should not be too long
Code Smell

Methods and properties that don't access instance data should be static
Code Smell

New-style classes should be used
Code Smell

Parentheses should not be used after certain keywords
Code Smell

Track "TODO" and "FIXME" comments that do not contain a reference to a person
Code Smell

Module names should comply with a naming convention

Dynamically executing code is security-sensitive

Analyze your code

Security Hotspot | Critical | cwe owasp

Executing code dynamically is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2017-9807
- CVE-2017-9802

Some APIs enable the execution of dynamic code by providing it as strings at runtime. These APIs might be useful in some very specific meta-programming use-cases. However most of the time their use is frowned upon because they also increase the risk of maliciously Injected Code. Such attacks can either run on the server or in the client (example: XSS attack) and have a huge impact on an application's security.

This rule marks for review each occurrence of such dynamic code execution. This rule does not detect code injections. It only highlights the use of APIs which should be used sparingly and very carefully.

Ask Yourself Whether

- the executed code may come from an untrusted source and hasn't been sanitized.
- you really need to run code dynamically.

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

Recommended Secure Coding Practices

Regarding the execution of unknown code, the best solution is to not run code provided by an untrusted source. If you really need to do it, run the code in a sandboxed environment. Use jails, firewalls and whatever means your operating system and programming language provide (example: Security Managers in java, iframes and same-origin policy for javascript in a web browser).

Do not try to create a blacklist of dangerous code. It is impossible to cover all attacks that way.

Avoid using dynamic code APIs whenever possible. Hard-coded code is always safer.

Sensitive Code Example

```
import os

value = input()
command = 'os.system("%s")' % value

def evaluate(command, file, mode):
    eval(command)  # Sensitive.

eval(command)  # Sensitive. Dynamic code
```

```
def execute(code, file, mode):
    exec(code)  # Sensitive.
    exec(compile(code, file, mode))  # Sensitive.

exec(command)  # Sensitive.
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-95 - Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')

Deprecated

This rule is deprecated, and will eventually be removed.

Available In:

sonarcloud | sonarqube

Code Smell

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

Code Smell

Lines should not end with trailing whitespaces

Code Smell

Files should contain an empty newline at the end

Code Smell

Long suffix "L" should be upper case

Code Smell

© 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