

- 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

Disabling auto-escaping in template engines is security-sensitive

Analyze your code

Security Hotspot

Major

cwe owasp

To reduce the risk of cross-site scripting attacks, templating systems, such as Twig, Django, Smarty, Groovy's `template engine`, allow configuration of automatic variable escaping before rendering templates. When escape occurs, characters that make sense to the browser (eg: `<a>`) will be transformed/replaced with escaped/sanitized values (eg: `<a>`).

Auto-escaping is not a magic feature to annihilate all cross-site scripting attacks, it depends on [the strategy applied](#) and the context, for example a "html auto-escaping" strategy (which only transforms html characters into [html entities](#)) will not be relevant when variables are used in a [html attribute](#) because `'` character is not escaped and thus an attack as below is possible:

```
<a href="{{ myLink }}">link</a> // myLink = javascript:
<a href="javascript:alert(document.cookie)">link</a> //
```

Ask Yourself Whether

- Templates are used to render web content and
 - dynamic variables in templates come from untrusted locations or are user-controlled inputs
 - there is no local mechanism in place to sanitize or validate the inputs.

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

Recommended Secure Coding Practices

Enable auto-escaping by default and continue to review the use of inputs in order to be sure that the chosen auto-escaping strategy is the right one.

Sensitive Code Example

```
from jinja2 import Environment


env = Environment() # Sensitive: New Jinja2 Environment
env = Environment(autoescape=False) # Sensitive:
```

Compliant Solution


```
from jinja2 import Environment
env = Environment(autoescape=True) # Compliant
```

See


- [OWASP Top 10 2021 Category A3](#) - Injection

 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

- [OWASP Cheat Sheet](#) - XSS Prevention Cheat Sheet
- [OWASP Top 10 2017 Category A7](#) - Cross-Site Scripting (XSS)
- [MITRE, CWE-79](#) - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Available In:

sonardcloud  | 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](#)