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...

Vulnerability

LDAP queries should not be vulnerable to injection attacks
Vulnerability

OS commands should not be vulnerable to command injection attacks
Vulnerability

The number and name of arguments passed to a function should match its parameters
Bug

The "open" builtin function should be called with a valid mode
Bug

Only defined names should be listed in "__all__"
Bug

Calls should not be made to non-callable values
Bug

Property getter, setter and deleter methods should have the expected number of parameters
Bug

Special methods should have an expected number of parameters
Bug

Instance and class methods should have at least one positional parameter
Bug

Boolean expressions of exceptions should not be used in "except" statements
Bug

Caught Exceptions must derive from

Endpoints should not be vulnerable to reflected cross-site scripting (XSS) attacks

Analyze your code

Vulnerability | Blocker | injection cwe sans-top25 owasp

User-provided data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Furthermore, when processing an HTTP request, a web server may copy user-provided data into the body of the HTTP response that is sent back to the user. This behavior is called a "reflection". Endpoints reflecting tainted data could allow attackers to inject code that would eventually be executed in the user's browser. This could enable a wide range of serious attacks like accessing/modifying sensitive information or impersonating other users.

Typically, the solution is one of the following:

- Validate user-provided data based on a whitelist and reject input that is not allowed.
- Sanitize user-provided data from any characters that can be used for malicious purposes.
- Encode user-provided data when it is reflected back in the HTTP response. Adjust the encoding to the output context so that, for example, HTML encoding is used for HTML content, HTML attribute encoding is used for attribute values, and JavaScript encoding is used for server-generated JavaScript.

When sanitizing or encoding data, it is recommended to only use libraries specifically designed for security purposes. Also, make sure that the library you are using is being actively maintained and is kept up-to-date with the latest discovered vulnerabilities.

Noncompliant Code Example

templates/xss_shared.html

```
<!doctype html>
<title>Hello from Flask</title>
{% if name %}
  <h1>Hello {{ name }}!</h1>
{% else %}
  <h1>Hello, World!</h1>
{% endif %}
```

xss.py

```
@xss.route('/insecure/no_template_engine_replace', methods =
def no_template_engine_replace():
    param = request.args.get('param', 'not set')

    html = open('templates/xss_shared.html').read()
    response = make_response(html.replace('{{ name }}', para
    return response
```

Compliant Solution

BaseException

Bug

Item operations should be done on objects supporting them

Bug

Raised Exceptions must derive from BaseException

Bug

Operators should be used on compatible types

Bug

```
templates/xss_shared.html

<!doctype html>
<title>Hello from Flask</title>
{% if name %}
  <h1>Hello {{ name }}!</h1>
{% else %}
  <h1>Hello, World!</h1>
{% endif %}

xss.py

@xss.route('/secure/no_template_engine_sanitized_Markup_esca
def no_template_engine_sanitized_Markup_escape():
    param = request.args.get('param', 'not set')

    param = Markup.escape(param)

    html = open('templates/xss_shared.html').read()
    response = make_response(html.replace('{{ name }}', para
    return response
```

See

- OWASP Top 10 2021 Category A3 - Injection
- 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')
- SANS Top 25 - Insecure Interaction Between Components

Available In:

sonarcloud | sonarqube Developer Edition