



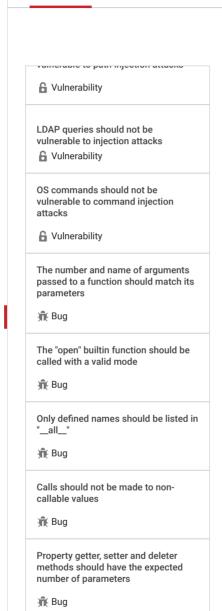
- **RPG**
- 1 Ruby
- Scala
- Swift
- Terraform
- Text
- **TypeScript**
- T-SQL
- **VB.NET**
- VB₆
- XML



Python static code analysis

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





Special methods should have an

expected number of parameters

Instance and class methods should

Boolean expressions of exceptions

Caught Exceptions must derive from

should not be used in "except"

have at least one positional parameter

Bug

📆 Bug

statements

📆 Bug

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

Tags

Analyze your code

Search by name...

❸ 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
- 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.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



Item operations should be done on objects supporting them



Raised Exceptions must derive from BaseException



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.pv
@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 | Develope Edition

© 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