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

Deserialization should not be vulnerable to injection attacks
Vulnerability

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

Database queries should not be vulnerable to injection attacks
Vulnerability

XML parsers should not be vulnerable to XXE attacks
Vulnerability

A secure password should be used when connecting to a database
Vulnerability

XPath expressions should not be vulnerable to injection attacks
Vulnerability

I/O function calls should not be vulnerable to path injection attacks
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

NoSQL operations should not be vulnerable to injection attacks

Analyze your code

Vulnerability | Blocker | injection cwe owasp sans-top25

User-provided data such as URL parameters and POST body-content should always be considered untrusted and tainted.

Applications that perform NoSQL operations based on tainted data can be exploited similarly to regular SQL injection bugs. Depending on the code, the same risks exist as with SQL injections: The attacker aims to access sensitive information or compromise data integrity. Attacks may involve the injection of query operators, JavaScript code, or string operations.

This problem can be mitigated by using an Object Document Mapper (ODM) library or by validating user-supplied data based on its size or allowed characters.

Noncompliant Code Example

For DynamoDB, when `FilterExpression`, `ProjectionExpression` or `KeyConditionExpression` parameter is influenced by user-controlled values, unexpected NoSQL operations may be executed:

```
DYNAMO_CLIENT = boto3.client('dynamodb', config=config)

DYNAMO_CLIENT.scan(
    FilterExpression= username + " = :u AND password =
    ExpressionAttributeValues={
        ":u": { 'S': username },
        ":p": { 'S': password }
     },
    ProjectionExpression="username, password",
    TableName="users"
) # Noncompliant
```

Compliant Solution

For DynamoDB, `FilterExpression`, `ProjectionExpression` and `KeyConditionExpression` parameters should not be influenced by user-controlled values:

```
DYNAMO_CLIENT = boto3.client('dynamodb', config=config)

DYNAMO_CLIENT.scan(
    FilterExpression= "username = :u AND password = :p"
    ExpressionAttributeValues={
        ":u": { 'S': username },
        ":p": { 'S': password }
     },
    ProjectionExpression="username, password",
```

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

```
    TableName="users"
)
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-943 - Improper Neutralization of Special Elements in Data Query Logic
- SANS Top 25 - Insecure Interaction Between Components

Available In:

sonarcloud | sonarqube Developer Edition