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...

Deserialization should not be vulnerable to injection 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. Deserialization based on data supplied by the user could result in two types of attacks:

- Remote code execution attacks, where the structure of the serialized data is changed to modify the behavior of the object being unserialized.
- Parameter tampering attacks, where data is modified to escalate privileges or change for example quantity or price of products.

The best way to protect against deserialization attacks is probably to challenge the use of the deserialization mechanism in the application. They are cases were the use of deserialization mechanism was not justified and created breaches (CVE-2017-9785).

If the use of deserialization mechanisms is valid in your context, the problem could be mitigated in any of the following ways:

- Since the pickle module is not secure, never use it to deserialize untrusted data.
- Only use the PyYAML module with the default safe loader.
- Instead of using a native data interchange format, use a safe, standard format such as untyped JSON or structured data approaches such as Google Protocol Buffers.
- To ensure integrity is not compromised, add a digital signature (HMAC) to the serialized data that is validated before deserialization (this is only valid if the client doesn't need to modify the serialized data)
- As a last resort, restrict deserialization to be possible only to specific, whitelisted classes.

Noncompliant Code Example

```
from flask import request
import pickle
import yaml

@app.route('/pickle')
def pickle_loads():
    file = request.files['pickle']
    pickle.load(file) # Noncompliant; Never use pickle modul

@app.route('/yaml')
def yaml_load():
    data = request.GET.get("data")
    yaml.load(data, Loader=yaml.Loader) # Noncompliant; Avoi
```

Compliant Solution

```
from flask import request
import yaml

@app.route('/yaml')
def yaml_load():
```

Bug

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

Bug

Caught Exceptions must derive from BaseException

Bug

Item operations should be done on objects supporting them

Bug

```
    data = request.GET.get("data")
    yaml.load(data) # Compliant;  Prefer using yaml.load wit
```

See

- OWASP Top 10 2021 Category A8 - Software and Data Integrity Failures
- OWASP Top 10 2017 Category A8 - Insecure Deserialization
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-502 - Deserialization of Untrusted Data
- SANS Top 25 - Risky Resource Management

Available In:

sonarcloud | sonarqube Developer Edition