Secure Coding Review

• Select a programming language and application to audit. • Perform a code review to identify security vulnerabilities. • Use tools like static analyzers or manual inspection methods. • Provide recommendations and best practices for secure coding. • Document findings and suggest remediation steps for safer code.

1. Choose language and application

To get started easily, consider:

- Language: Python 3.x

- Framework / App: a Flask-type micro-web app (to-do list, blog, contact form...)

This choice gives you an ecosystem of consolidated tools and a contained code base.

2. Prepare the audit environment

I checked if I can use this code and scanned it using automatic tools.

git clone https://github.com/sreecodeslayer/todo-flask.git

A simple todo app to learn Python Web development micro framework and also CRUD operations, Login/Signup User account handling by sreecodeslayer

Topics

angularjs mongo todo sweetalert font-awesome mongoengine python-flask

```bash

# 1. Clone the repository

git clone https://github.com/sreecodeslayer/todo-flask.git

cd todo-flask

# 2. Create and activate the virtualenv
```

python3 -m venv venv

source venv/bin/activate

Install static analysis tools:

- Bandit (`pip install bandit`)

Install dependencies

pip install -requirements.txt

- Flake8 (`pip install flake8`)

- Safety or pip-audit (`pip install safety`)

3. Automatic analysis

Perform a “first pass” with the scanners right away:

```bash
bandit -r.

flake8 .

safety check
```

Note each alert:

- Hard coded secrets

- System commands from user input

- Unparameterized SQL queries

4. Run Bandit on the entire project

-r:recursive; -lll:show all severities; -fjson:output to JSON

bandit -r. -lll -fjson > bandit-reportj.json

5. Analyze the results

- Open bandit-report.json with an editor or:

jq '.' bandit-report.json # if you have jq installed

6. YML and Report Bandit

Open bandit.yml and copy-paste this content:

yaml

#bandit.yml

----

1) Exclude directories/files from scanning

exclude:

- tests/ # test folder

- migrations/ # migration files

#2)Ignore specific rules (Bandit codes)

skip:

B101: # use of assert

B303: # pickle insecure

#3)Set the minimum severity to report

minimum_severity: MEDIUM

#4)(Optional) Configure minimum confidence

minimum_confidence: MEDIUM

Explanation:

- exclude: directories/files you don't want to be analyzed •
skip: single Bandit (Bxxx) rules to ignore •
minimum_severity: will only report issues of level ÿ MEDIUM •
minimum_confidence: (optional) filters low confidence results
•

Launch Bandit using the config file In the terminal, still from root, run:

bandit -r. -c bandit.yml

Where:

- -r.starts the scan recursively • -c bandit.yml
indicates to use the custom configuration file

Code scanned:

Total lines of code: 171

Total lines skipped (#nosec): 0

Run metrics:

Total issues (by severity):

Undefined: 0

Low: 2

Medium: 1

High: 1

Total issues (by confidence):

Undefined: 0

Low: 0

Medium: 4

High: 0

Files skipped (0):

(Optional) Export the report to HTML or TXT Bandit doesn't do this automatically via config, but you can add command line options: bash

#HTML report

bandit -r. -c bandit.yml -f html -o bandit-report.html

Extra tips:

- Integrate Bandit into CI(GitHub Actions, GitLab CI) to run analytics on every push.

- Combine with `flake8` and `safety check` to cover style, known vulnerabilities and dependency risks.

For manual inspection:

Review the most critical parts, using this checklist:

- Sensitive configurations: `DEBUG=True`, SECRET_KEY in clear text

- Authentication/Authorization: Are there routes open to anyone?

- Input validation: Are all fields sanitized?

- CSRF protection: Do form POSTs include tokens?

- SQL Injection: Do all queries use placeholder parameters?

- XSS: escape output in Jinja2 template?

- Error handling: do not expose stack traces in production

- Cryptography: Password hashing with bcrypt/Argon2, not MD5/SHA1

Document the results

Metrics:

Total lines of code: 171
Total lines skipped (#nosec): 0

ID Test	Vulnerabilities	severity	Line number	Description
B201	CWE-94	HIGH	170	A Flask app appears to run with debug=True, which exposes the Debugger tool and allows the execution of arbitration code.
B104	CWE-605	MEDIUM	170	Possible binding to all interfaces.
B105	CWE-259	LOW	12	Possible hardcoded password: ''
B105	CWE-259	LOW	13	Possible hardcoded password: 'I am I being w@tched? Damn yes!'

Recommendations and best practices

- Shift-left: integrate Bandit/Flake8 scans into CI(GitHub Actions, GitLab CI…)

- Dependency management: run `safety check` or `pip-audit` regularly

- Secure Defaults: disable DEBUG in production; set Secure and HttpOnly cookies

- Peerreview: Add OWASP Top 10 rubrics to team code reviews

- Logging & Monitoring: Do not log PII, use WAF/IDS in production

Final report

Executive summary: objectives, scope, tools used

Goals:

- identify vulnerabilities in the code, in order to make it more secure. This is a job of prevention.
- understand how a Secure Coding Review works
- tools: Kali Linux, Bandit, Git, GitHub

```
ÿ(venv)ÿ(minaÿkali)-[~/todo-flask]
 ÿÿ$ jq '.' bandit-report.json
{
  "errors": [],
  "generated_at": "2025-06-29T13:01:30Z",
  "metrics": {
          "./app.py": {
          "CONFIDENCE.HIGH": 0,
          "CONFIDENCE.LOW": 0,
          "CONFIDENCE.MEDIUM": 2,
          "CONFIDENCE.UNDEFINED": 0,
          "SEVERITY.HIGH": 1,
          "SEVERITY.LOW": 0,
          "SEVERITY.MEDIUM": 1,
          "SEVERITY.UNDEFINED": 0,
          "loc": 123,
          "nosec": 0,
          "skipped_tests": 0
          },
          "./models.py": {
          "CONFIDENCE.HIGH": 0,
          "CONFIDENCE.LOW": 0,
          "CONFIDENCE.MEDIUM": 0,
          "CONFIDENCE.UNDEFINED": 0,
          "SEVERITY.HIGH": 0,
          "SEVERITY.LOW": 0,
```

"SEVERITY.MEDIUM": 0,

"SEVERITY.UNDEFINED": 0,

"loc": 35,

"nosec": 0,

"skipped_tests": 0

},

"./settings.py": {

"CONFIDENCE.HIGH": 0,

"CONFIDENCE.LOW": 0,

"CONFIDENCE.MEDIUM": 2,

"CONFIDENCE.UNDEFINED": 0,

"SEVERITY.HIGH": 0,

"SEVERITY.LOW": 2,

"SEVERITY.MEDIUM": 0,

"SEVERITY.UNDEFINED": 0,

"loc": 13,

"nosec": 0,

"skipped_tests": 0

},

"_totals": {

"CONFIDENCE.HIGH": 0,

"CONFIDENCE.LOW": 0,

"CONFIDENCE.MEDIUM": 4,

"CONFIDENCE.UNDEFINED": 0,

"SEVERITY.HIGH": 1,

"SEVERITY.LOW": 2,

"SEVERITY.MEDIUM": 1,

```
            "SEVERITY.UNDEFINED": 0,
            "loc": 171,
            "nosec": 0,
            "skipped_tests": 0
            }
 },
 "results": [
            {
            "code": "169 if __name__ == '__main__':\n170 \tapp.run(debug=True, threaded=True,
host='0.0.0.0')\n",
            "col_oset": 1,
            "end_col_oset": 51,
            "filename": "./app.py",
            "issue_confidence": "MEDIUM",
            "issue_cwe": {
            "id": 94,
            "link": "https://cwe.mitre.org/data/definitions/94.html"
            },
            "issue_severity": "HIGH",
            "issue_text": "A Flask app appears to be run with debug=True, which exposes the Werkzeug
debugger and allows the execution of arbitrary code.",
            "line_number": 170,
            "line_range": [
            170
            ],
            "more_info": "https://bandit.readthedocs.io/en/1.8.5/plugins/b201_flask_debug_true.html",
            "test_id": "B201",
```

```
            "test_name": "flask_debug_true"
            }
    ]
}
```