snyk

Python Security Best Practices Cheat Sheet

www.snyk.io

1. Use Python 3

What version of Python are you using? The end of Python 2 is near—January 1, 2020. If you are using Python 2 past that date, you leave yourself open to any emerging security vulnerabilities.

2. Scan your code with Bandit

Bandit is an open source security scanner for Python code. It can be run locally or as part of your CI/CD pipeline.

3. Use Pipenv for environment and dependency management

Deterministic builds are important for predictable behavior in production. However, pinning your dependencies to achieve this leaves you open to security vulnerabilities. Pipenv helps you manage your environment and dependencies in a predictable, secure way.

4. Watch your import statements

Python import statements are flexible but can be exploited. Implicit relative imports (deprecated as of Python 3) leave your code vulnerable to malicious code execution. Whatever import method you use, remember be sure you trust the module--importing executes code!

5. Download packages with care

Assume that there are malicious packages available on PyPI. When installing, be sure to spell the package name correctly--you don't want to install a malicious package that is named for a common misspelling of a popular package.

6. Handle requests safely

Understand how the requests library utilizes certain security practices so you can get the most out of them. Keep your version of certifi up to date—the requests library uses it to verify certificates.

7. Be careful with string formatting

There are multiple ways to format strings in Python. When formatting strings from user input, extra care is needed to avoid things like injection or code execution. The Template class in the string module is a more secure way to format strings with user input.

8. Be careful with string formatting

Understand the different types of open source licenses and adhere to their terms. Be wary of any project that does not have a license; you may not like the terms of the license they eventually adopt. Over 10% of packages on PyPI fall into this category.

9. Deserialize selectively

Do not deserialize data from an untrusted source. Python's pickle module allows for this using pickle.load. Deserializing from an untrusted source can result in arbitrary code execution.

10. Keep up-to-date on vulnerabilities

Just because your app is secure today does not mean it will be secure tomorrow. Stay up to date on new vulnerabilities by using the Pipenv safety package or consider trying Snyk's tools, which can alert you when a new vulnerability is found in a package that you are using.