Secure coding practices

Author: Vedang Joshi

Published: October 26, 2022 · 5 min read

Hello Everyone! we all have come across tweets like these - Security vulnerabilities being detected and exploited. What causes security vulnerabilities in the first places ? Can it be prevented ? and most importantly, what do we learn from these ?

Security vulnerabilities occur when a program does something it isn't meant to do - A bug. It was there all along, maybe in the code you didn't even know existed. For hackers, it is an opportunity and they would surely take advantage of it. Here, I would discuss some best practices that avoid common kinds of security vulnerabilities.

Common pitfalls

Dynamically executed code

If a program is taking input from the user and executing it without any sanitization, it is like serving the integrity of the application on a silver platter to the bad actors. In the case of python, the use of eval statements is highly discouraged. These types of attacks are commonly addressed as injection attacks. An attack could inject SQL statement, HTML, or Javascript which if executed on the server side gives the attackers full control. Log4j was an example of code injection. XCKD sums this up in a humorous comic.

Developers need to make sure that there is no room for user input to get executed. Here user input can be input from the end user or even from another developer.

SQL Injection can be avoided by using an ORM instead of writing raw queries. Mostly all popular ORMs come with built in sanitization features.

Major Web frameworks like Django and Flask also come with their sanitization functions. This prevents potentially malicious HTML inputs.

Other than this, Static application security testing (SAST) tools can also detect lines of code that execute a user input and flags them. Using a SAST check either on the local system or in the CI pipeline is recommended.

Hardcoding credentials

While building an application, it is a common requirement to fetch some data from an external API. It may be some payment gateway integration, cloud provider, or any other API. Generally, these APIs are authenticated and we need to specify the authentication key in the headers. While programming, developers often hardcode the credentials in the code itself. This becomes a serious issue when this code is pushed to a git repository because now the keys are visible to everybody who has access to the git repo. This problem amplifies when this git repo is public. Which means the keys are exposed to the entire world.

Not only this practice is a blunder in security but it hinders the developer's productivity too. Once the code becomes a part of a container image and is pushed to production, it becomes difficult to track and replace the credentials if needed.

It is quite easy to avoid this

1. Passing the credentials as env variables solves this problem. Developers can create a .env file if the number of variables is large. This .env file should never be checked out to git.
2. Implementing secret scanning in pre-commit hooks. This can make sure no credentials are hardcoded before every commit. Similarly, secrets scanning can be added as a CI stage as well.

Parsing

Developers often need to deal with XML and YAML files. The first step in using any of these files is to parse them. There are a few common attacks through these parsers.

- XML parsing
  Python XML standard library warns the users that using the etree, DOM, xmlrpc is not secure. An attacker can create a DOS-style attack. This attack references a piece of data in the same file and doing it over and over increases the RAM usage exponentially. Thus crashing the server.
  An attacker can also take advantage of an XML parser trying to fetch and parse resources from an external URL. This can be used to circumvent firewalls.
  The use of defuseXML is recommended as a fix.
- YAML parsing
  Using the `yaml.load` method can be used to make system calls, thus leaving you wide open to attack.
  The use of `yaml.safe_load` is recommended as a fix.

General Tips

Updating dependencies

Your python program surely uses one or the other libraries from PyPI. These libraries may contain some security vulnerabilities. If the library is maintained then these security vulnerabilities would be fixed in no time, once it is fixed it is our responsibility to update our dependencies to the proper version. Developers may find things breaking when a version upgrade on a dependency is done, but it is better than having a known vulnerability in your software.

It is recommended that developers subscribe to the GitHub releases of the packages they use in their code and update the dependencies whenever a stable version is released.

Trusting opensource

No doubt, we all love open-source software, but while installing a package from PyPI for your project, we need to always verify that the library is well-maintained. Installing unknown or outdated packages can introduce security risks. Install either the popular libraries only or review the library before installing.

That's all from my side.

Thank You

Security Python DevOps