Python security best practices

Emerging-Topics-in-Cyber-Security

S.D.Sachie Karunathilake

Contents

Python security best practices

Video link :
https://drive.google.com/file/d/1cv9bYGjS246wUfgKZcBVoNlpfPW_VRPQ/view?usp=sharing

1. Use the most recent version of Python

In December 2008, ython 3 was released, but some people still use older Python versions for their projects. One concern is that Python 2.7 is not the same safety upgrade as Python 3 and earlier. For example, Python 3 has improved input methods and exception chaining. You can use the inputs to execute code in Python 2.7, which was written in Python 3.

Python 2.7 will also loose support in 2020, so you will finally have to do something if you are too close to Python 2.7 to scale up to Python 3. The more programs you use with older Python versions, the more hassle it will be to upgrade all of them in future.

Check python.org for the latest update if you want details about the latest version of Python. And if you do not know which Python version you are using now, run the following on your local machine to check:

```
python --version
```

1. Use a virtual environment

Using a virtual environment for each project, rather than globally downloading packages on your computer. In one project, it doesn't affect others if a malicious code package dependency is installed. The packages of each project are individually separated.

By creating a distinct folder for packages used within the given project, Virtualenv supports an isolated Python environment.

The following are installed:

```
pip install virtualenv
```

Activate this in the project location:

```
venv\Scripts\activate.bat
```

2. Set debug = false

Debug is specified by default in new projects for certain Python framework projects, such as Django. This could be helpful for developing our code bugs, but it's not so helpful if the project is deployed to live on a publicly accessible server. Publicly displaying errors in our code could show a security vulnerability that is being exploited.

Thus always set the following when deployed live:

```
Debug = false
```

3. Never commit anything with a password

If most developers use GitHub, double-check that a file, reading or URL with your password has not been committed. Once the password is committed to GitHub or to a similar service, it's still in a log or database to find for anyone. For eg, in May 2019, a hacker stored hundreds of passwords in GitHub repositories in plain text and requested a lending of 0.1 Bitcoin each.

Consciously stop inserting the source code with passwords or API keys.

4. Look out for poisoned packages

A programming language is just as powerful for most programmers as its libraries. Python has many beautiful libraries which can be easily installed via Pip.

Make sure you use legal and modified packages to double-check. Packages of malicious code can be built both for Python and Node.js. Make sure that the names for each kit are exactly correct. '00Seven' is a whole kit rather than '000Seven.'

In addition to double checking the name of the package, tools like Sqreen can also be used to monitor malicious code packages and search for legitimate problems or obsolete version packages in your application. Look at its Application Security Management solution to improve app visibility and secure the applications from vulnerability-induced attacks.

5. Check import paths

Three types of Python import paths are available: absolute, relative, and implicit.

An implied route does not specify the address of the package. So somewhere on your machine, the software uses a module of the same name. You may have a malicious code package installed. In Python packages, there were a variety of Trojan horse instances, specifically PyPi, of malicious code. In addition, for one year, any have not been observed.

To avoid such uncertainty, use an absolute course. We know clearly the right package to use only with the full address of the package and that malicious code is verified. That's the best way.

```
from safe_package import safe_module
```

A relative path indicates the location of the module relative to the current folder.

```
from ..some_package import less_danger
```

6. Protect against SQL injections

So how does anyone insert SQL into a database? Some people bot on a server that kills millions of websites that are poorly programmed in the expectation that sufficient people will click their affiliates.

7. Use pycryptodome for cryptography

You stop using pycrypt for your cryptography toolkit, as highlighted in a previous post. There has been a flaw, and a security update to address the problem has not been published since then. Currently, in years, the project has not been revised.

But this is cool. This is cool. Instead of using pycryptodome:

```
pip install pycryptodome
```

9. Use Bandit

For each Python project, install the Bandit package. Bandit scans the code for famous vulnerabilities including the YAML problems. It classifies the security risk from low to high and informs you which lines of code the problem is causing.

```
pip install bandit
bandit path/project.py
```

10. Keep your servers up to date

Often possible hazards are not connected to code but to servers. You need to verify that all of your software is modified and Python code compliant. Random human error will kill the planned work that has taken years. So maintain up-to-date security and information management systems.

Reference

1. Roche, D. and Roche, D., 2020. *Top 10 Python Security Best Practices - Sqreen Blog*. [online] Sqreen Blog. Available at: <https://blog.sqreen.com/top-10-python-security-best-practices/> [Accessed 29 November 2020].
2. Snyk. 2020. *Python Security Best Practices Cheat Sheet* **|** *Snyk*. [online] Available at: <https://snyk.io/blog/python-security-best-practices-cheat-sheet/> [Accessed 29 November 2020].
3. Medium. 2020. *Python — Secure Coding Guidelines*. [online] Available at: <https://medium.com/@felsen88/python-secure-coding-guidelines-73c7ce1db86c> [Accessed 29 November 2020].
4. Python, S., Huszagh, A., Kane, J. and B, B., 2020. *Secure Coding Guidelines For Python*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/32279010/secure-coding-guidelines-for-python> [Accessed 29 November 2020].
5. Sqreen Blog. 2020. *Developer Security Best Practices: How To Validate Input Data In Python*. [online] Available at: <https://blog.sqreen.com/developer-security-best-practices-how-to-validate-input-data-in-python/> [Accessed 29 November 2020].