



CVE Binary Tool: Improve concurrency and Input Support

17.03.2020

About Me

Name: Niraj Kamdar

Major: B.Tech in Information and Communication Technology (ICT)

Degree level: 3rd-year undergraduate

Graduating year: 2021

GitHub: [Niraj-Kamdar](#)

IRC: nirajkamdar

Address: [D-312, DA-IICT, Near infocity, Gandhinagar, Gujarat, India](#)

Timezone: Kolkata, West Bengal (GMT+5:30)

Resume: [My resume](#)

Overview

I want to work on improving the output interface of the CVE Binary Tool. I really liked the idea of scanning vulnerabilities in binary files and I have been driven to work on this tool because I myself have developed several binary apps and It would be good if I know about vulnerabilities and severity it contains due to dependencies so that I can choose appropriate dependencies which reduces security risks.

Code contribution

[ADD] [unittest for package: sqlite3 and version: 3.30.1](#)

Added a simple unittest for sqlite3 version: 3.30.1 and for that I have created test-sqlite-3.30.1.c which produces a fake binary for testing purposes.

[FIX] [csv2cve does not output CVE severity](#)

Added vendor, product, version and severity information along with cve in csv2cve console output.

[FIX] [test_skips, test_runs and test_unknown in test_cli.py are not working on windows](#)

Above error was due to a mismatch in output depending on os. I have fixed it by adding a pattern in Assertion which satisfies both linux and windows.

[FIX] [Fix usage of subprocess in extractor.py](#)

Replaced call of subprocess from fstring to array.

[ADD] [Added codecov to Github Actions file](#)

Added codecov to GitHub actions with every build specific flag. Now we can see the coverage report directly inside PR.

[FIX] [test_scanner accidentally storing previous cves](#)

List of CVEs were getting stored between tests and results were getting passed due to this. Now, long_test will use a new scanner object to overcome this problem.

[FIX] [fixed test_string, test_extract and test_checker on Windows](#)

Fixed rpm extractor in windows which wasn't extracting the cpio file and also fixed the problem with line ending (\r\n) in Windows for test_string.

[ADD] [Added gstreamer checker](#)

Added new gstreamer checker with long test cases for both ubuntu and centos.

[ADD] [Added harfbuzz checker](#)

Added new harfbuzz checker with long test cases for both ubuntu and centos.

[ADD] [Added binutils checker](#)

Fixed problem in signature of binutils checker.

Project information

Sub-org name: CVE Binary Tool

Project Abstract

1. Refactoring `cve-bin-tool` into independent submodules:

Refactor `cli.py` into three separate modules: 1) `cli.py`, 2) `version_scanner.py`, and 3) `cve_scanner.py`.

2. Make `cve-bin-tool` as concurrent as possible:

Use `asyncio` for IO bound tasks like extracting archives, downloading NVD datasets etc. Use `concurrent.futures` for CPU bound tasks like scanning for version string in input file with each checker's version pattern regex.

3. Allow users to scan CVEs from various input format:

We already have a `csv2cve` option to get CVEs from the csv formatted input. It would be nice to extend this functionality for other formats like JSON and HTML.

4. Allow users to divide CVEs in different categories and specify severity:

Add an option while generating user specified formats like json or csv to add each individual CVE in one of the four categories [**Not yet investigated**, **Confirmed**, **Mitigated** or **Ignored**] and also allow the user to specify severity for that CVE.

5. Internationalization of `cve-bin-tool`:

Add an argument option to select language or set language in config file. Also translate `readme.md`, `contributing.md` into multiple languages.

Detailed Description

First Goal will involve refactoring the **cli.py** file into various modules. Currently, **cli.py** became this huge file which is scanning files and extracting versions and finding CVEs from database and performing other operations as specified in arguments passed by the user.

I am proposing to divide cli.py into 3 modules:

1. **cli.py** - which only contains main functions and calls appropriate module as per user arguments
2. **version_scanner.py** - which will contain scanner class and other helper functions like `extract_and_scan`. New scanner module won't need **cvedb** because It won't be performing database lookup for finding CVEs. **version_scanner.py** only takes the path as an argument and returns a dictionary of package, vendor and versions.
3. **cve_scanner.py** - which will take package, vendor and versions as arguments and returns `all_cves`. This module will use the **cvedb** module to perform database lookup.

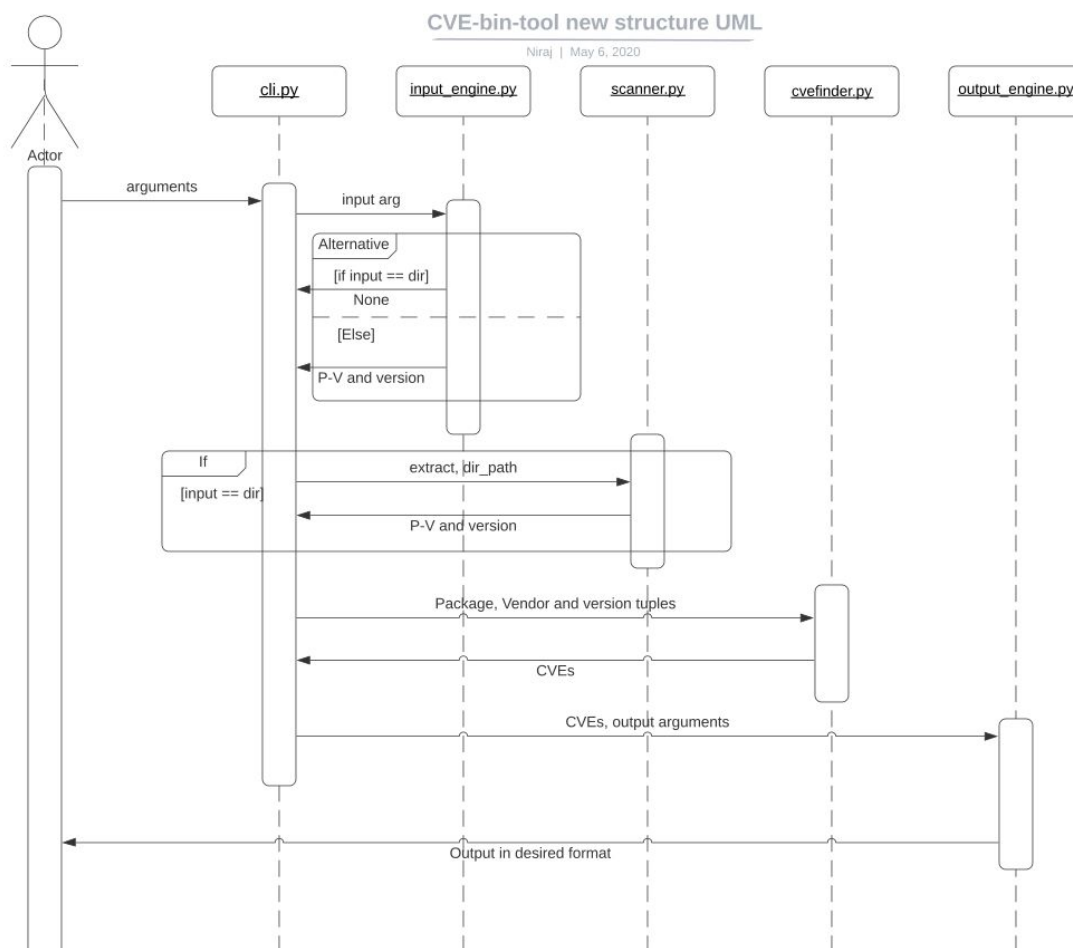
Question - Dividing **cli.py** into **cli.py** and **scanner.py** makes sense but why do we need to divide functionality of **scanner** into **version_scanner** and **cve_scanner**?

Reason for this is the introduction of a general `input_engine` which will take a csv, json or html file and make a dictionary containing package, vendor, versions and other user specified triage values. So, when we use `input_engine`, `scanner.py` function can be skipped completely and we can use `cve_finder` functions on the dictionary provided by `input_engine` directly.


Dividing modules like this will make each module independent of one another. Integration only happens in `cli.py`. `cli.py` will import each of these modules and use it according to arguments provided by the user.

Second Goal will involve refactoring code such that it can be run as concurrently as possible. **Extractor.py** currently uses many subprocess calls. We can use `asyncio`'s `subprocess` function to make it run asynchronously. I will use **aiofiles** to make **strings.py** and **input_engine.py** asynchronous. Since scanning for the version pattern using every checker is a CPU bound task, I am going to use a process pool to leverage multiple cores. I will use **aiosqlite** to make database operations asynchronous. I am going to use **aiohttp** or **parfive** module to download nvd datasets concurrently.

Third Goal will involve adding *input_engine* to handle various input and triage data. Currently, *csv2cve.py* is a separate module which contains code from parsing data from CSV to finding CVEs from database and showing it. We can implement more general *input_engine* module which will parse every data format supported by *cve_bin_tool* (csv, json and html) and create dictionary of package, vendor, versions and other user specified triage values and give it to *cve_scanner* which will find all CVEs and also handles triage data.



Above, UML diagram explains general flow from user calling *cli.py* with arguments to *output_engine* displaying output to the user.



Forth Goal will involve taking user feedback about vulnerabilities. Add an option while generating user specified formats like json or csv to add each individual CVE in one of the four categories [**Not yet investigated**, **Confirmed**, **Mitigated** or **Ignored**] and also allow the user to specify severity for that CVE. This user specified data will be stored as a separate column in csv or as a separate attribute in json so that we don't have to maintain a separate structure. We will only use these triage attributes while displaying output. We can take these attributes interactively with the help of CLI tools like [PyInquirer](#) or [clint2](#).

Fifth Goal will involve internationalization of the cve-bin-tool. We can do this by following this blog: [A complete guide to i18n in Python](#) mentioned by [@terriko](#). We can also convert our documentation in different languages using these i18n tools. Once we setup locale directory for different languages, we can provide an option to set output language. We have to take care that vendors and products don't get translated into the target language. Translated output will be the same as above figures just output language will be changed.

Milestones

I. Community Bonding (May 4 - May 31)

- Actively bonding with the community through [Gitter channel](#).
- Fix existing bugs, help merging pending PRs and closing issues.
- Discussing the proposed idea with mentors and other members of the organization
- Finalizing all the deliverables with the mentor and adding/removing extra work wherever needed
- Creating Milestones on GitHub projects using Kanban template.

II. Week 1 (June 1 - June 7)

- Refactoring cli.py into cli.py and scanner.py.
- Remove compiler dependency from test_scanner.

III. Week 2 (June 8 - June 14)

- Refactor cvedb into cve_scanner and cvedb
- Refactor cvedb test and write cve_scanner test.

IV. Week 3 (June 15 - June 21)

- Make extractor module asynchronous.\
- Rewrite Unittest of extractor to support asyncio.

V. Week 4 (June 22 - June 28)

- Use aiohttp to download nvd dataset concurrently.
- Make cve_scanner, cvedb and version_signature asynchronous using aiosqlite module.

VI. Week 5 (June 29 - July 5)

- Complete remaining tasks from Week 4
- Write unit test for cvedb, cve_scanner and version_signature.
- *First Evaluation Period* from **June 29** to **July 3**

VII. Week 6 (July 6 - July 12)

- Use aiofiles to concurrent strings module
- Use a process pool to parallelly execute checkers.

VIII. Week 7 (July 13 - July 19)

- Make version_scanner fully asynchronous.
- Test all new asynchronous packages.

IX. Week 8 (July 20 - July 26)

- Create asynchronous json and csv parsers in the input_engine module.
- Write unit tests for these two functions.

X. Week 9 (July 27 - August 2)

- Create a HTML parser that parses previously generated html data which may contain triage data.
- Write a unit test for this new html parser.

XI. Week 10 (August 3 - August 9)

- Preparing for internationalization of CVE Binary Tool.
- Mark every string in the source code we want to have translation.

XII. Week 11 (August 10 - August 16)

- Add options for extracting messages and compiling catalogue in setup script
- Write unit-tests for the above functionality.

XIII. Week 12 (August 17 - August 23)

- Parallelize pytest as much as possible.
- Fix small remaining issues
- Make sure everything works fine in Windows.

XIV. Final week (August 24 - August 31)

- Document all code, refactor code wherever needed, wrap up any remaining work and *submit final code for evaluation.*

Stretch Goals

I would like to work on automating the checker creation process after GSoC completes or my major goals get completed before GSoC deadline. This will include the following three functionality.

1. Package downloader which downloads packages for the different distro
2. Package Analyser which analyzes strings of these packages and tries to guess the best version pattern string and guess_contains string from the packages according to preconfigured rules.
3. Test module generator which generates modules and files necessary for testing this checker.

This will make new checker creation very easy because we just have to provide the vendor package pair of the checker we want to implement. I have already written a few scripts for the same. It can be found in my github repo [cve-bin-tool-helper-script](#).

Other commitments

From mid-July onwards various companies will be visiting my college for placement drive, so from mid-July onwards I may not be available on some days due to interviews other than this I will be devoting 7-9 hours daily for this project.

Why me?

Other than being a Python developer for over 2 years, and a passionate open source contributor, I believe that I am well suited for this project, because I have already worked on a CLI project where I have used [PrettyTable](#) for nice looking tabular output and [PyInquirer](#) for interactive CLI interface. You can see this project [here](#).

I have also done web development using Flask, that's why I am very familiar with Jinja templating. So, I will be able to create nice looking HTML easily.

I am also good with project management. Currently, I am managing a project named [Question-paper-generator](#) which is a Website written in Flask with Jinja templating.

Since I have worked on many Python projects which had many overlapping requirements with the CVE Binary Tool, I believe I would be the ideal candidate to work on the CVE Binary Tool project.