Google Summer of Code

Improving code coverage and Implementing fuzzing

CVE Binary Tool

Python Software Foundation

About Me

Name	Yashu Garg
Github	@yashugarg
Email	garg.y2001@gmail.com
Timezone	IST (GMT +5:30)
LinkedIn	/yashugarg

Education and Background

College	Cluster Innovation Centre
University	University of Delhi
Location	New Delhi, India
Program	Bachelor of Technology
Major	Information Technology and Mathematical Innovations
Expected Graduation	June 2023

I am a pre-final year engineering student. I have experience working with Python and other scripting languages. I have developed automated bots and done statistical analysis using python for the following projects:

1. Creating a Discord Chatbot in Python
2. Learning Data Science with Python

Contributions in cve-bin-tool:

Issues:

Issue	Issue Link	Status
Plan to make the tool support more archive types to extract and test.	#1555	Closed, Resolved

Code Submissions:

Contribution	Issue	Pull Request	Status
New checker: gnome librsvg This was my first contribution to the project where I added the Gnome librsvg vulnerability checker.	#1494	#1533	Closed, Merged
New checker: libseccomp After the first contribution, I moved to add a checker with more complex file name patterns and signatures.	#1493	#1556	Closed, Merged
Modified format_checkers to add checker name to dictionary allow Wrote a script to automatically add words from checker names to the "allow" dictionary for spelling checks	#1567	#1571	Closed, Merged
Added extractor support for more archives Enabled the tool to extract package distribution files from more operating systems like FreeBSD (.pkg) and Arch Linux (.pkg.tar.zst)	#1555	#1580	Closed, Merged

Project Information

Organisation

Python Software Foundation

Sub-Organisation

CVE Binary Tool

Abstract

- Getting the test coverage to over 95%, which is currently hovering around 80% coverage according to codecov: https://codecov.io/gh/intel/cve-bin-tool.
- Using fuzzers on some file inputs to find new bugs and fix them. Optimising code and adding tests to increase the general robustness of the code.

Detailed Description,

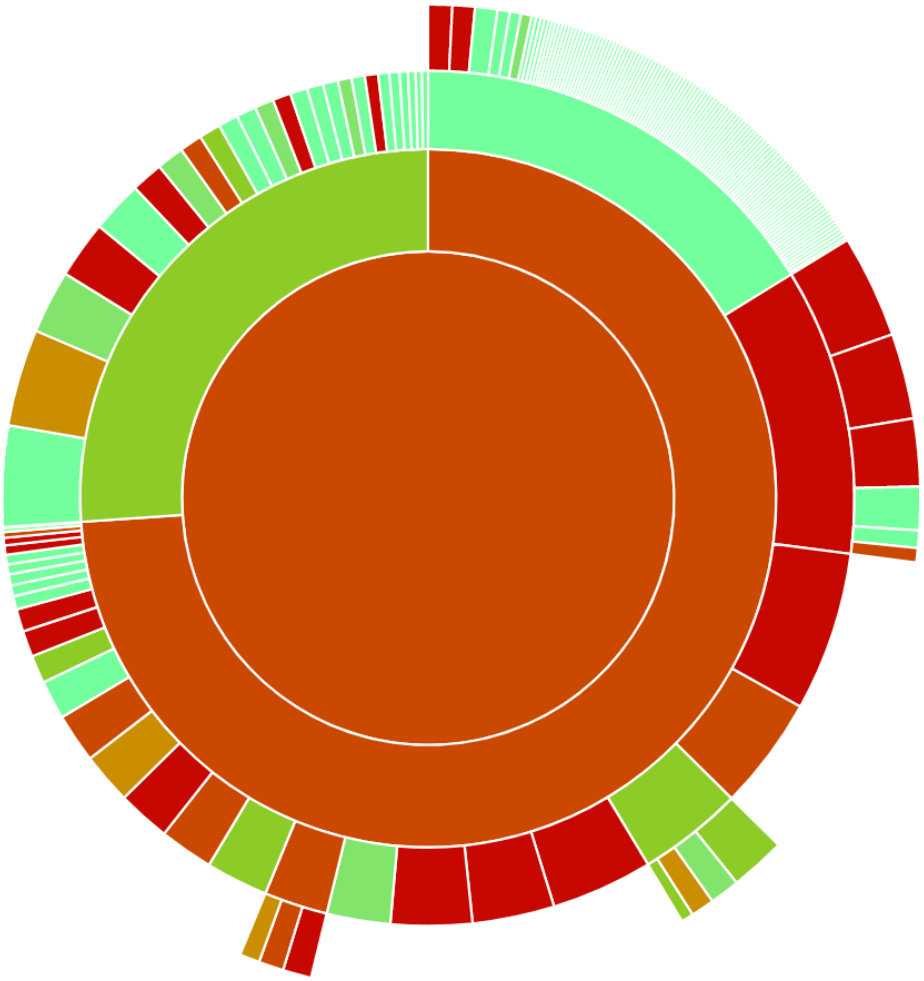
The CVE Binary Tool helps users determine known vulnerabilities in their system. The tool currently uses data from the National Vulnerability Database (NVD) list of Common Vulnerabilities and Exposures (CVEs), but additional data sources are being worked upon constantly.
I added two more to the list of over 100 checkers that focus on common vulnerable and open-source components scanned through to determine packages included in the scanned software.

Phase 1: Increasing code test coverage

Code coverage is a measurement used to express which lines of code were executed by a test suite. Three primary terms used to describe each line executed are the following.

- **Hit** indicates that the source code was executed by the test suite.
- **Partial** indicates that the source code was not fully executed by the test suite; there are remaining branches that were not executed.
- **Miss** indicates that the source code was not executed by the test suite.

The code coverage ranges from 75% to 85% according to codecov, with over 1000 misses. These misses include much code that is obsolete and functions with no proper tests.



Graph showing test coverage in different modules of the tool
Source: https://app.codecov.io/gh/intel/cve-bin-tool as of 11 April 2022.

For this part of the proposed project, I plan on removing and refactoring the code that is no longer needed. This includes analysis of different modules to understand and increase the general robustness of the functions.

Most of the modules with fewer lines than average are either total hits or misses. The partial execution is rather low, making it a good entry point to start the project.

Many modules don't have tests implemented and have 0% coverage. Hence, I propose to add test cases for the same from scratch without much past work to refactor, making the process fairly easy.

To make the overall average of code coverage cross 95%, I plan to begin by pushing the coverage of the ***cve_bin_tool*** suite to over 90% and the ***test*** suite to nearly 100%.

I also plan to improve the test harness and CI infrastructure of the project. A few checkers fail to test successfully on CI while working perfectly fine locally (Glibc being one example). Also, there's something wrong with Windows tests, so I plan to improve the CI for long tests on Windows as a stretch goal for this phase.

To summarise, I would carry out the following steps to implement the first half of the project:

- Removing/Refactoring code that is unoptimized or no longer required.
- Writing tests for the code paths that have not been covered yet.
- Improving the test harness and CI infrastructure of the project.

Stretch Goal

I noticed that the SBOM module doesn't require much work on code coverage because it is already around 90%, thus allowing me to work parallelly on the issues related to the module as a stretch goal while writing tests for other sections of the tool.
This goal includes adding more information to the SBOMs, possibly things that are not generally included in the final package (Reference Issue #1383).
It might also include adding a new output mode. Probably SBOM report to make it both user and machine-friendly for better integration into the tool to help get updates on binaries that weren't changing. (Reference Issue #475)

Phase 2: Implementing fuzzers

During the execution of the second part of the project, after the 95% test coverage, I plan to improve the robustness of current code by using fuzzers on some of our file inputs to find new bugs, then fix them and add test cases.
Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. A fuzzer is a program that injects automatically semi-random data into a program/stack and detects bugs. The data-generation part is made of generators, and vulnerability identification relies on debugging tools.
Two of the most popular tools to implement fuzzing in python libraries are pythonfuzz and Google's Atheris. While these tools are apt for a thorough implementation of the entire project, libraries like xmlfuzzer and PyJFuzz can be helpful for specific file type inputs.
The tool I plan to refer to the most is Fuzzer by Lachlan, which can be used to implement fuzzing for all input file types like JSON, CSV, and XML, thus reducing redundancy and making the process quick and tidy.

Weekly Timeline

Pre GSoC (Upto May 20)

- Make more contributions through issues and features to further my understanding of the codebase.
- Brush up on all the necessary topics and libraries in the proposed project.
- Get more familiar with the test methodology of the project.

Community Bonding (May 20 - June 12)

- Communicate with the mentors on different ideas for the testing and get input on how it can be integrated in a user-friendly way.
- Communicate with other selected applicants about their projects and how we can help each other.
- Keep working on issues and start exploring possible alternatives for fuzzers.

Phase 1 of 2 (Improving Code Coverage)

Week 1 (June 13 - June 20)

- Start refactoring dead or unoptimized code by going through files with high partial hits.

Week 2 (June 20 - June 27)

- Continue removing/refactoring unused code.
- Start writing new tests for incomplete modules.

Week 3 (June 27 - July 4)

- Fix any bugs caused during refactoring.
- Continue writing tests for different modules.

Week 4 (July 4 - July 11)

- Communicate with fellow candidates and developers to ensure all the new code is well tested and covered.
- Start writing tests for modules with 0% coverage.

Week 5 (July 11 - July 18)

- Finish writing tests.
- Start looking for possible optimizations in the CI interface and test harnesses.
- Fix CI test fails for checkers with complex file patterns.

Week 6 (July 18 - July 25).

- Keep working on improving long tests on Windows.
- Finish up phase 1 of the project with documentation.

Phase 2 of 2 (Implementing fuzzers)

Week 7 (July 25 - August 1)

- Phase 1 submission and evaluation.
- Discuss and start implementing ideas for fuzzing.

Week 8 (August 1 - August 8)

- Start implementing fuzzers for XML file inputs on SBOM code.

Week 9 (August 8 - August 15)

- Finish working on the SBOM module.
- Start with JSON/CSV fuzzers for report code.

Week 10 (August 15 - August 22)

- Finish up implementing fuzzing for the project.

Week 11 (August 22 - August 29)

- Work on fixing all the new and reported issues.
- Add new test cases to suit the changes made.

Week 12 (August 29 - September 5)

- Keep fixing new issues and adding new test cases.
- Wrap things up for phase 2 of the project with documentation.

Final Week (September 5 - September 12)

- Prepare a final summary and organise the work into a presentable form.
- Submission of the work for final mentor evaluation.

Other Commitments

The GSoC timeline aligns with my availability, therefore providing me the opportunity to work throughout the summer with focus and absolute conviction.
I do not have any exams during the GSoC contribution period.

Are you applying for other projects in GSoC?

No, I am only applying for this project.

Further Contributions

CVE Binary tool is a security-based python project which aligns with my interest in the field, so I will be more than happy to stay a part of the community even after the GSoC to keep contributing and learning.