HW 3: Code Review
Eastern Washington University
CYBR 487: Secure Software Engineerin

Date: January 24, 2026
Written by: Kyle Costlow and Calvin Ruch

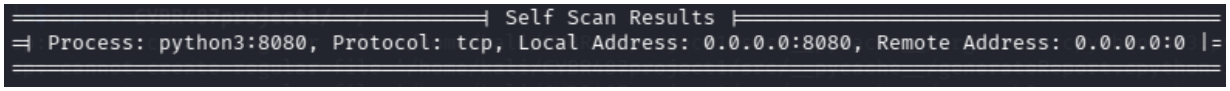
Homework 3 Code Review

Code Functionality

Does the code work as intended?

Self scan works nicely. We may want more verbose information in the future, but in its current state it meets the feature requirements.

The report generator works very well and is easy to extend if additional report types are needed. One thing we noticed is that the top line of the self-scan results takes up unnecessary space, but this is a small issue and should be easy to fix.



Argument parsing uses flags to control what scan takes place (for example, `-ss` for self scan or `-ps` for port scan).

The `-vv` option does not have real functionality behind it yet, but it can be used later for technical users that do not need extra explanation about what is being output.

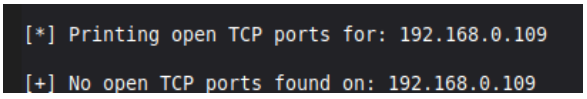
Are edge cases handled appropriately?

Many edge cases in the argument parser are handled well, including accepting multiple forms of the same answer so the program can continue operating under bad input conditions. However, there is less handling for newer arguments (`hostid`, `scanType`, and `maxhops`). If the program receives an unexpected value, it may throw a runtime error. Improving this would reduce the risk of errors during critical program operations.

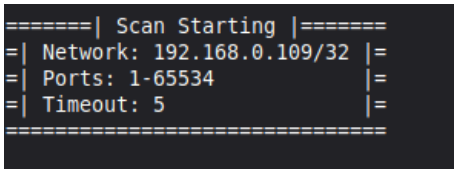
Are there logical errors or bugs?

All of the following bugs and issues have been fixed (see screenshots). The fixes were made in PR #53, with each fix committed separately in the Code-Review branch.

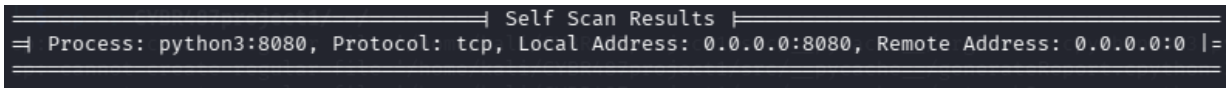
- Port scan output printed even when nothing was returned.



- When scanning a large range of ports, the pre-scan listing of targets broke. This may need to return a string that the report generator can display safely.



- Self-scan reporting included an extra line before the report started.



Code Design and Architecture

Does the code adhere to SOLID principles?

The code somewhat follows the Single Responsibility Principle, but a few classes were made more dynamic so they can be used in different ways. The code mostly follows the Open/Closed Principle, but in some cases it has been easier to modify existing functions and refactor other parts around them. This also suggests that Interface Segregation has not been fully developed yet, since some functions have been expanded to increase functionality.

Is the code modular and reusable?

The code is very modular, with functionality organized into classes following OOP principles. Docstrings exist for every function and class, which should make the codebase reusable and easier to maintain.

Are there any superfluous dependencies?

Before the code review, there were multiple unused imports across several files, including one duplicated import that was only used in one place. These were reviewed and removed during the code review.

Does the code align with the system's overall architecture?

Yes. Features are separated into classes unless they are tightly intertwined, so most classes have a single, clear use case.

Code Readability and Maintainability

Is the code clear and precise?

The code is generally clear. Some functions are complex, but documentation and docstrings help explain what the code does.

Do the names make sense?

Names make sense for the most part. There may be a few arbitrary variable names, but the overall flow of the code usually makes the purpose of each variable clear.

Are the brain benders explained?

The code does not appear to be overly complex. Any complexity is mostly due to PEP 8 formatting requirements that split long expressions across multiple lines.

Performance and Efficiency

Is the code a speed demon or a slowpoke?

When scanning a large number of ports or a large network, runtime can increase significantly. This is definitely a slowpoke for large scan ranges and should be addressed in the next sprint.

Is the code a memory hog?

There may be better approaches than using a dictionary to hold reporting data, and it could be beneficial to store some scan data in a temporary file. That said, on a machine with two processors and 2048 MB of memory, the program ran without issue and performed well.

Are we using the right tools for the job?

For the next feature, we plan to implement a graph data structure to represent connections. This may free up memory currently stored in dictionaries and enable faster lookups and more efficient operations.

Security

Is the code Fort Knox or a welcome mat for hackers?

This code was not written with the primary goal of resisting attackers, but it has few dependencies and none of the dependencies handle sensitive data. There are also no external databases or services attached, so the overall attack surface is smaller than it would be for a typical networked application.

Are we playing it safe with user input?

There is a lot of user input checking and sanitization. The program may still throw a runtime error if it encounters input it does not expect, but there is no current input path that is used without some form of validation.

Are secrets actually secret?

There are no secrets that would cause harm if leaked. The project does not use external APIs, credentials, or authentication mechanisms.

Error Handling and Logging

Is the code ready for Murphy's Law?

The code works well in isolated environments when everything is set up correctly. However, if someone does not read the help output, README, or documentation, they could still misconfigure the tool. Input sanitization and runtime errors will only go so far.

Can we play detective if things go south?

Yes. Since the codebase was built feature-by-feature, it is easy to identify where issues occur. The

reporting and output provide enough information to troubleshoot failures and determine what went wrong.

Testing

Are the new features battle-tested?

We tested the features on a cluster of VMs with internal routing, and the traceroute feature worked well in that environment. Self scan was also tested against web servers running on random ports. Unit testing is not comprehensive, but we have performed extensive system testing across different distributions and environments.

Documentation

Is the code telling its story well?

The report generator is called throughout the main program to describe which scan is about to run before it runs. There is also full documentation describing what the program does, and the codebase docstrings fill in details where anything may be unclear.

Coding Standards and Best Practices

Is the code playing by our rules?

The code follows the standards we set prior to starting the project, including OOP design, PEP 8 styling, and docstrings for all functions.

Version Control

Are we telling a good story with our commits?

We have made a strong effort to use meaningful commit messages. Under deadlines, some commit messages are less descriptive and may include more changes than ideal, but overall the commit history explains what was changed and why.