PORT SCANNER

Creating a port scanner in python

Sergiusz Pieszak
[redacted]

Contents

1. Introduction

This report provides a critical analysis of a Python-based port scanning tool's development, focusing on its importance in network security assessment. It examines the tools, libraries, and modules utilized, analyses the software development process, presents evidence of debugging and security measures, and provides recommendations and lessons learned. By encapsulating the tool's concept, implementation, and prospects, the report contributes to the discussion on secure software development and network security methods.

2. Tools, Libraries, and Modules

2.1 Libraries

After conducting extensive research, I carefully selected the following libraries for my program:

socket: In Python, the socket library is a standard library module that provides low-level networking interface. It allows you to create network connections, transmit and receive data over the network, and handle various network-related tasks. Python (no date, a)

threading: The threading library in Python provides a high-level interface for working with threads. Threads allow you to run multiple tasks simultaneously within a single process. The threading module allows you to create and maintain threads, synchronize their execution, and communicate with them. Python (no date, b)

sys: The sys module in Python is a standard library module that provides access to some variables used or maintained by the Python interpreter and to functions that interact strongly with the interpreter. Python (no date, c)

Queue: Python's queue module is a standard library module that provides thread-safe, FIFO (First-In-First-Out) data structures suitable for implementing producer-consumer scenarios in a multithreaded environment. Python (no date, d)

termcolor: The termcolor library in Python is a simple utility for outputting coloured text to the terminal. It provides a straightforward way to add coloured text and formatting to console output, making it more visually appealing and easier to read. Python (no date, e)

2.2 Functions

Choosing functions over classes in a port scanner is distinct advantages. Functions excel in procedural tasks such as port scanning, providing an efficient way to execute steps without extensive state or behaviour management. They promote modularity, allowing for code breakdown into reusable components for better organization and maintenance. In handling the straightforward tasks of port scanning, functions offer a lightweight and direct solution compared to classes, enhancing development, and improving performance. Additionally, leveraging functions facilitates a more efficient and focused approach to constructing a robust port scanner tailored to network security assessment needs. W3 Schools (no date)

2.3 Variables

The program utilizes variables to store, store, and manipulate data, which may be from user input or other external sources. Variables play diverse roles within the program.

2.4 if __name__ == "__main__"

In Python, `` `if __name__ == "__main__"` `` is a special conditional statement that checks whether the Python script is being run as the main program or if it is being imported as a module into another script. When a Python script is executed, Python sets the special variable `` `__name__` `` to `` `"__main__"` `` if the script is the main program being run. However, if the script is being imported as a module into another script, the `` `__name__` `` variable is set to the name of the module. If __name__ == "__main__": is used to define a block of code that should only be executed when the script is run directly, and not when it is imported as a module into another script. This allows for separating executable code from module definitions, allowing the script to be more modular and flexible. Breuss, M. (2022)

3. Software Development Lifecycle

In every stage of the Software Development Life Cycle (SDLC), ensuring the security of the entire application is paramount. Security considerations must be integrated into each phase to maintain a secure security posture throughout the development process. Let's examine how a secure SDLC framework can be applied to the development of a membership renewal portal according to Freeman, C. (2022):

1. Requirements Gathering: In the initial phase, stakeholders collaborate to outline new features while prioritizing security requirements within functional requirements.

2. Design: Functional requirements are incorporated into an application blueprint with a strong emphasis on security measures such as session token validation to prevent unauthorized access.

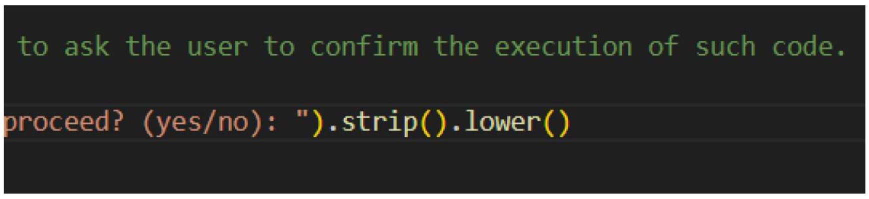
3. Development: Development efforts concentrate on adhering to secure coding practices and guidelines. Rigorous code reviews are conducted to detect and rectify vulnerabilities. Leveraging open-source components is common practice, with thorough scrutiny using Software Composition Analysis tools.

4. Verification: Comprehensive testing, such as automated security testing, is conducted to ensure compliance with design and requirements. Continuous Integration/Continuous Deployment (CI/CD) pipelines are utilized to automate testing and deployment procedures.

5. Maintenance and Enhancement: Following deployment, ongoing vigilance is essential to address any vulnerabilities identified in the production environment. This may involve patching code, revising implementations, or responding to external penetration tests and bug bounty submissions.

4. Debugging

The debugging process for this program was extremely successful, thanks to the meticulous planning and execution of the programming. The groundwork laid during the planning phase ensured that the code encountered minimal difficulties, allowing for a simpler debugging experience.

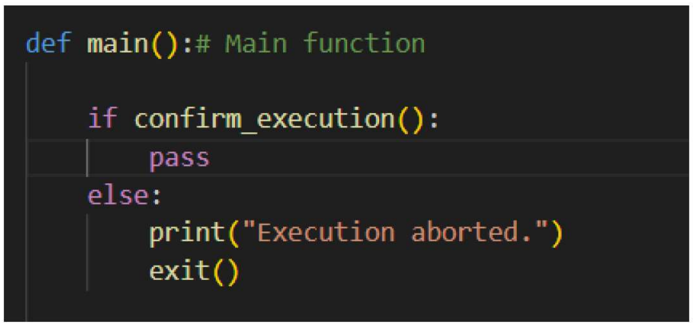


```
to ask the user to confirm the execution of such code.

proceed? (yes/no): ").strip().lower()
```

Figure 1

The application failed to recognize the user input, prompting me to take action. To ensure compatibility with variations like "YES," "YeS," or "yES," I opted to incorporate the `.strip()` and `.lower()` methods.



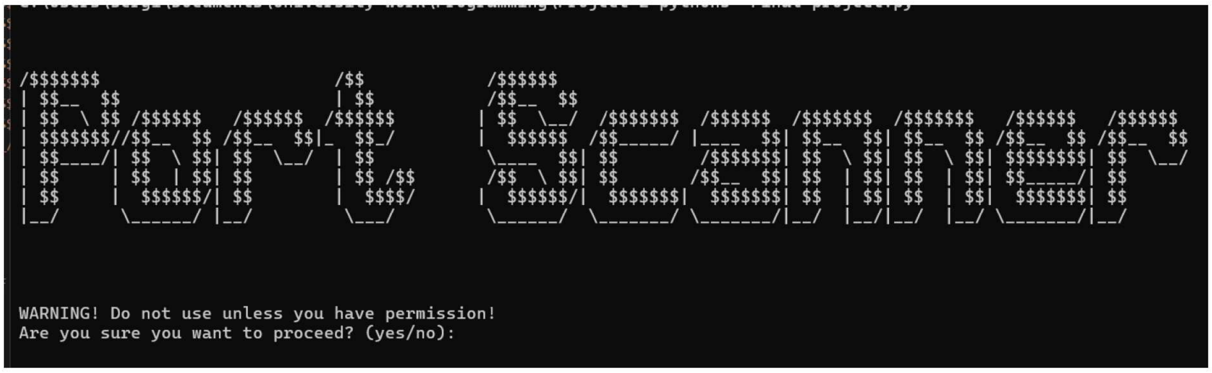
```python
def main():# Main function

    if confirm_execution():
        pass
    else:
        print("Execution aborted.")
        exit()
```

Figure 2

While integrating the `confirm_execution()` function, I encountered an issue where the "Execution aborted" section failed to exit the program. It turned out that I had forgotten to include the `exit()` function, causing the program to continue executing regardless.

5. Screenshots



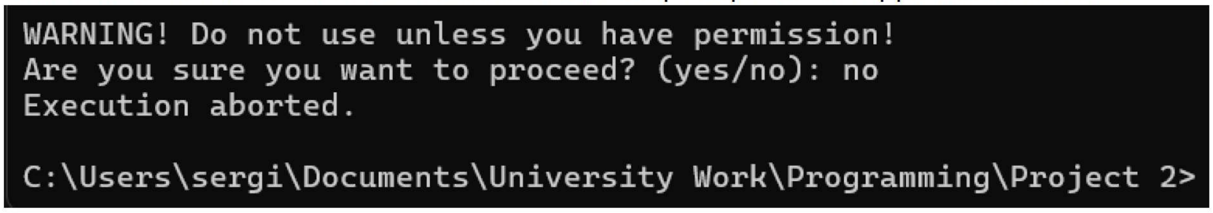
```
 /$$$$$$$                       /$$            /$$$$$$
| $$__  $$                     | $$           /$$__  $$
| $$  \ $$ /$$$$$$   /$$$$$$  /$$$$$$        | $$  \__/  /$$$$$$$  /$$$$$$  /$$$$$$$  /$$$$$$$   /$$$$$$   /$$$$$$
| $$$$$$$//$$__  $$ /$$__  $$|_  $$_/        |  $$$$$$  /$$_____/ |____  $$| $$__  $$| $$__  $$ /$$__  $$ /$$__  $$
| $$____/| $$  \ $$| $$  \__/  | $$           \____  $$| $$        /$$$$$$$| $$  \ $$| $$  \ $$| $$$$$$$$| $$  \__/
| $$     | $$  | $$| $$        | $$ /$$       /$$  \ $$| $$       /$$__  $$| $$  | $$| $$  | $$| $$_____/| $$
| $$     |  $$$$$$/| $$        |  $$$$/      |  $$$$$$/|  $$$$$$$|  $$$$$$$| $$  | $$| $$  | $$|  $$$$$$$| $$
|__/      \______/ |__/         \___/         \______/  \_______/ \_______/|__/  |__/|__/  |__/ \_______/|__/



WARNING! Do not use unless you have permission!
Are you sure you want to proceed? (yes/no):
```

Figure 3

As depicted in *Figure 3*, this serves as an introduction to the program. ASCII art was utilized to enable users to differentiate between the command prompt and the application interface.

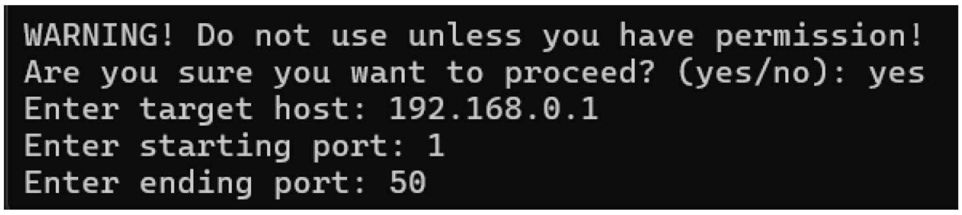


```
WARNING! Do not use unless you have permission!
Are you sure you want to proceed? (yes/no): no
Execution aborted.

C:\Users\sergi\Documents\University Work\Programming\Project 2>
```

Figure 4

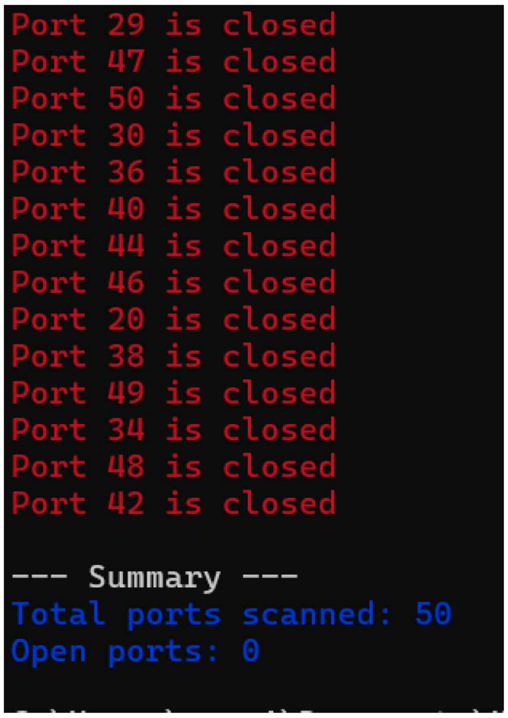
A failsafe needed implementation given the nature of a port scanner. Engaging in port scanning on a network without proper authorization is unlawful and can result in legal consequences under The Computer Misuse Act 1990 - UK Government (1990)



```
WARNING! Do not use unless you have permission!
Are you sure you want to proceed? (yes/no): yes
Enter target host: 192.168.0.1
Enter starting port: 1
Enter ending port: 50
```

Figure 5

Once the warning has been acknowledged with a "yes" response, we can proceed to inputting the data. The application will prompt for the target host, followed by determining the port range by requesting the starting and ending port numbers.

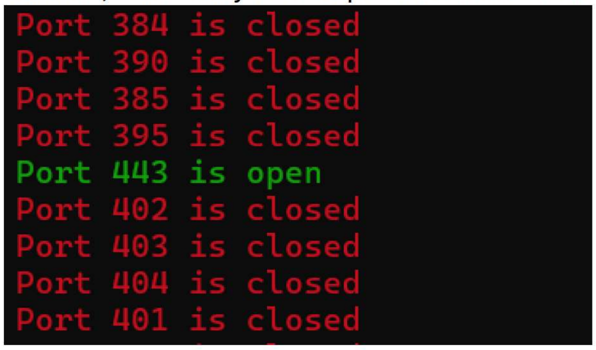


```
Port 29 is closed
Port 47 is closed
Port 50 is closed
Port 30 is closed
Port 36 is closed
Port 40 is closed
Port 44 is closed
Port 46 is closed
Port 20 is closed
Port 38 is closed
Port 49 is closed
Port 34 is closed
Port 48 is closed
Port 42 is closed

--- Summary ---
Total ports scanned: 50
Open ports: 0
```

Figure 6

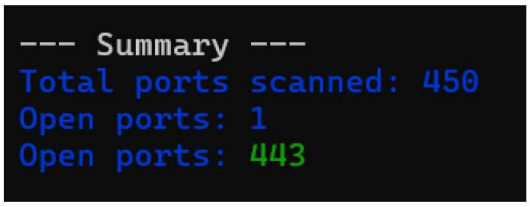
Once the scanner is configured, the scanning process commences. However, due to the threading functionality, the ports may not align sequentially. As ports are preloaded into threads, some may be completed ahead of others.



```
Port 384 is closed
Port 390 is closed
Port 385 is closed
Port 395 is closed
Port 443 is open
Port 402 is closed
Port 403 is closed
Port 404 is closed
Port 401 is closed
```

Figure 7

As depicted in *Figure 7*, an open port will be highlighted in green, enhancing the user's ability to visually identify open ports more effectively.



```
--- Summary ---
Total ports scanned: 450
Open ports: 1
Open ports: 443
```

Figure 8

At the conclusion of the scan, a summary is presented to condense the information and enhance readability, highlighting key aspects such as the number of open ports and the total ports scanned.

6. Recommendations and Lessons Learnt

Throughout the project development, numerous recommendations and insights have surfaced:

- Develop a function to export scan results as a .csv file.

- Implement a password function to enhance tool security.

- Organize the tool into a class for broader utility and easier integration.

- Introduce pre-made packages covering common ports like 80, 443, etc., for streamlined usage.

Conclusion

In summary, this report examines the development of a Python-based port scanning tool, highlighting its role in network security assessment. It examines tool selection, development processes, debugging, and security measures. Although it provides valuable insights, challenges such as the lack of specialized port lists and the inability to save results in a.csv format were identified. Despite these, the report contributes to discussions on secure software development and network security, offering recommendations for future enhancements.

References

Breuss, M. (2022) *What Does if __name__ == "__main__" Do in Python?* Available at: https://realpython.com/if-name-main-python/ (Accessed on: 17 March 2024)

Freeman, C. (2022) *Secure SDLC 101*. Available at: https://www.synopsys.com/blogs/software-security/secure-sdlc.html (Accessed on: 18 March 2024)

UK Government (1990) *Computer Misuse Act 1990*. Available at: https://www.legislation.gov.uk/ukpga/1990/18/contents (Accessed on: 17 March 2024)

Python (no date, a) *socket — Low-level networking interface.* Available at: https://docs.python.org/3/library/socket.html (Accessed on: 18 March 2024)

Python (no date, b) *threading — Thread-based parallelism*. Available at: https://docs.python.org/3/library/threading.html (Accessed on: 18 March 2024)

Python (no date, c) *sys — System-specific parameters and functions*. Available at: https://docs.python.org/3/library/sys.html (Accessed on: 18 March 2024)

Python (no date, d) *queue — A synchronized queue class*. Available at: https://docs.python.org/3/library/queue.html (Accessed on: 18 March 2024)

Python (no date, e) *termcolor 2.4.0*. Available at: https://pypi.org/project/termcolor/ (Accessed on: 18 March 2024)

W3 Schools (no date) *Python Functions*. Available at: https://www.w3schools.com/python/python_functions.asp (Accessed on: 19 March 2024)