**Final Project: Password Strength Analyzer**

Isaias Barrios

National University

Professor Sean Ayersman MS, CISSP

June 26, 2025

1. **Introduction**

For my final project I chose to build a Password Strength Analyzer that would evaluate

the strength of passwords against common attack patterns and that suggested improvements. This project will show my understanding of the course material I gained in Security Automation. I decided to work independently on this project to challenge myself. I created and organized this project using Visual Studio Code, testing the code in a virtual environment, then pushing it to GitHub. Throughout the process I used AI tools like ChatGPT, GitHub Copilot, andGrammarly. These tools were extremely useful that helped me troubleshoot, refine, and organize this project's workflow. My reflection will explain the technical and personal insights and experiences I gained through this project.

1. **Setting Up the Environment**

The first step in my project was to create an isolated environment to make everything consistent and to avoid dependency conflicts. I did this by creating a dedicated folder in Visual Studio Code where all of my files related to the password analyzer would live in. Then I initialized a virtual environment by running, python -m venv venv, in the terminal. Then I activated it in the command prompt by using, venv\Scripts\activate.bat. Even though this was not necessary, I felt that this was important in keeping the project self contained and would be more professional.

Once the virtual environment was active, I installed the rich library using, pip install rich, to enhance the command-line interface with color-coded output. I did this because it makes the command like applications visually appealing and presents data in a more readable way (Will McGugan, n.d). I tracked the installed package by generating a requirements.txt file using, pip freeze > requirements.txt. This will allow anyone who clones the repository to install the same environment with just one command. I believe that taking the time to structure and isolate the environment from the beginning made the rest of the process much smoother and more organized.

1. **Development Process**
   1. **Core Script - password\_analyzer.py**

The password strength analyzer was written in a single file called

password\_analyzer.py. I began by importing the necessary modules: re for pattern matching, math for calculating entropy, getpass to securely capture input from the terminal, and rich.print for the improved output format.

The first function I used was calculate\_entropy() which would estimate how strong a password is by calculating its Shannon entropy. In the context of password strength entropy refers to how unpredictable or random a password is, and Shannon entropy quantifies this randomness by estimating how many bits of information are needed to represent the password (Bernstein, n.d) . In short, a low entropy would entail a short password and a high entropy would entail a longer and stronger password. The formula I used would give me a numeric value showing how difficult it would be to guess a password based on its length and character variety.

Then I implemented basic\_checks(), which would evaluate if a password meets five criteria which are: minimum length of 12 characters, inclusion of one uppercase letter, inclusion of one lower case letter, one digit, and one special character. Then I built the alayze() function. This would run the entropy check, evaluate each policy, and then print the results in a color coded format. It would also get suggestions from the get\_suggestions() module and show improvement tips if it was needed. Lastly I added a CLI entry point in order to make the tool executable directly from the terminal.

* 1. **Modular Suggestions - suggestions.py**

I created a separate file named suggestions.py in order to keep the main script organized. This module would handle the user feedback based on the password analysis results. I started by defining a set of commonly used weak passwords like “123456”, “password”, and “admin” (NordPass, 2023). The main function, get\_suggestions(), would receive three arguments. It would receive the password entered by the user, the results from the basic check, and the entropy score. Then based on these inputs the functions would build a list of specific suggestions. For example if an entered password was shorter than 12 characters it would return a tip saying, “use at least 12 characters for better security”.

A very useful feature was the entropy threshold logic. If a password’s entropy falls below a certain level, the function would recommend increasing the character variety so the password would be more resistant to a brute force attack. This would make sure that a user isn't just told that their password is weak, but they also know how to fix it. I believe that keeping suggestions.py modular was a good idea. I was able to write and test the feedback logic separately which made the main analyzer script cleaner and easier to follow.

1. **Testing and Debugging**

Once the core functionality was working, I then built a dedicated file named test\_passwrod\_analyzer.py using Pythons built in unit test module. This would let me verify automatically that these key functions would behave as expected across different scenarios. I had three main test cases. The first test was to make sure that the entropy of “abcd” was lower than the entropy of “abcd1234”. It would also check that a more complex password like “All2@6b#3!” would produce a high enough entropy score. I had originally set the expected threshold at 50 bits, but when the test failed I had to recalculate the entropy formula. For the password “All2@6b#3!” The entropy was about 37.1 bits which was lower than expected but I felt was still a pretty strong password. I then adjusted the threshold to 25 bits which would make the test more realistic without having to compromise security expectations.

The other tests focused on the basic\_checks() function. One test ensured that a password that met the five criteria passed every check and the other validated that a weak password would correctly fail most of them. The script’s logic was accurate and the feedback given to the user was based on testable rules. And also, unit testing helped me catch errors early and also gave me a better understanding of how to test security related logic.

1. **Project Organization & Collaboration Tools**

Keeping this project organized was very important to me which was why I divided the

project into separate files. The password\_analyzer.py is the core functionality, suggestions.py for improvement tips, test\_password\_analyzer.py for automated testing, requirements.txt to track dependencies, and the README.md to document the setup instructions, purpose and usage. I initially encountered an issue when trying to push the project using thee Visual Studio Code GUI but after some trouble shooting and deleting a corrupted .Git folder I was successfully able to push it to GitHub using git init, git remote add, and git push.

1. **Use of AI Tools**

I used three different AI tools throughout this project. ChatGPT was a very useful tool throughout my project. ChatGPT helped me brainstorm the structure of the project, debug code issues, and it helped me understand the logic behind password entropy. When I ran into a test failure because of the unrealistic entropy expectations, I asked ChatGPT to better understand the formula and how to revise the test.

I also used GitHub Copilot inside of Visual Studio Code. It was helpful when writing test cases, formatting regular expressions, and generating the inline comments. Copilot’s autocomplete function saved me a lot of time and reduced repetitive coding tasks as well.

The third AI tool I used was Grammarly. Grammarly proofread my README.md file and my final reflection. The use of these AI tools did not replace my own work, but they helped guide and organize my project tremendously.

1. **Challenges and Lessons Learned**

As mentioned before, one of the main challenges I encountered was related to the entropy values. At first I thought that a complex password would exceed an entropy threshold of 50 bits. When I ran the test I was surprised to see it fail. I reviewed my code and realized that my expectation was flawed. I had to research how entropy actually works and recalibrate the test to match real-world mathematical outcomes. This Debugging process taught me how to think critically about why a test might fail.

Another challenge involved version control. When I was trying to push the repository to GitHub using the Visual Studio Code GUI, I ran into different issues with branches and remote configuration. I failed to solve the issue through the interface so I decided to start over from the command line. This solved my issue and also helped me gain confidence using Git from the terminal.

1. **Final Thoughts**

Building the Password Strength Analyzer from scratch helped me understand and reinforce what I have learned throughout this course. This was the first time I had to independently manage a security automation project. Everything from the environment setup, to testing and troubleshooting, and GitHub deployment made me more confident and made me use tools like unittest, manage virtual environments, and work with version control systems. I also learned the importance of maintaining well documented and organized code so it can easily be shared with others. This process also showed me how the use of AI tools can enhance learning and productivity and has to be used responsibly. It was great for brainstorming and refining logic, but I still needed to understand every piece of code I wrote. Security should never be an afterthought and with automation, a simple tool like this can help enforce stronger and safer security habits.

**References**

Bernstein, M. (n.d.). *Understanding entropy: From information theory to passwords*. Retrieved

June 22, 2025, from<https://mbernste.github.io/posts/sourcecoding/>

NordPass. (2023). *Top 200 most common passwords*. Retrieved June 22, 2025, from

<https://nordpass.com/most-common-passwords-list/>

Will McGugan. (n.d.). *Rich: Python library for rich text and beautiful formatting in the terminal.*

Textualize. Retrieved June 22, 2025, from <https://rich.readthedocs.io/en/stable/introduction.html>