Password Strength Analyzer with Custom Wordlist Generator

Cybersecurity Internship Project Report

Student Name: M.Vyshnavi
Date: October 25, 2025

Introduction

In today's digital age, weak passwords remain one of the most exploited vulnerabilities in cybersecurity. This project addresses password security through two complementary tools: a Password Strength Analyzer that evaluates password quality using entropy-based calculations, and a Custom Wordlist Generator that demonstrates how attackers create targeted password lists for penetration testing. The dual approach provides both defensive analysis capabilities and offensive security awareness, making it valuable for understanding real-world password attack vectors.

Abstract

This project implements a Python-based cybersecurity tool suite consisting of two integrated components. The Password Strength Analyzer examines user passwords against industry-standard security metrics including character diversity, entropy calculations, and length requirements, providing immediate feedback on password vulnerabilities. The Custom Wordlist Generator creates comprehensive attack dictionaries by combining base words with common patterns including leetspeak substitutions (e.g., a→@, e→3), year combinations, special character variations, and capitalization permutations. The implementation demonstrates practical cybersecurity concepts including password entropy theory, combinatorial attack strategies, and secure coding practices. All functionality was successfully implemented and tested using Python 3.13.9 in a Windows development environment with Visual Studio Code.

Tools and Technologies Used

The following tools and technologies were employed in this project:

Programming Environment:

- **Python 3.13.9**: Primary programming language for implementation
- **Visual Studio Code**: Integrated development environment (IDE) for code development and testing
- **Windows PowerShell/Terminal**: Command-line interface for program execution

Python Libraries:

- **string**: Built-in library for character set operations (ASCII letters, digits, punctuation)
- **math**: Mathematical operations including logarithmic entropy calculations
- **getpass**: Secure password input handling (hidden password entry)
- **argparse**: Command-line argument parsing for wordlist generator
- **itertools**: Efficient generation of character combinations and permutations

Development Tools:

- **VS Code Extensions**: Python syntax highlighting, debugging, and IntelliSense
- **Git/Version Control**: Code versioning and project management

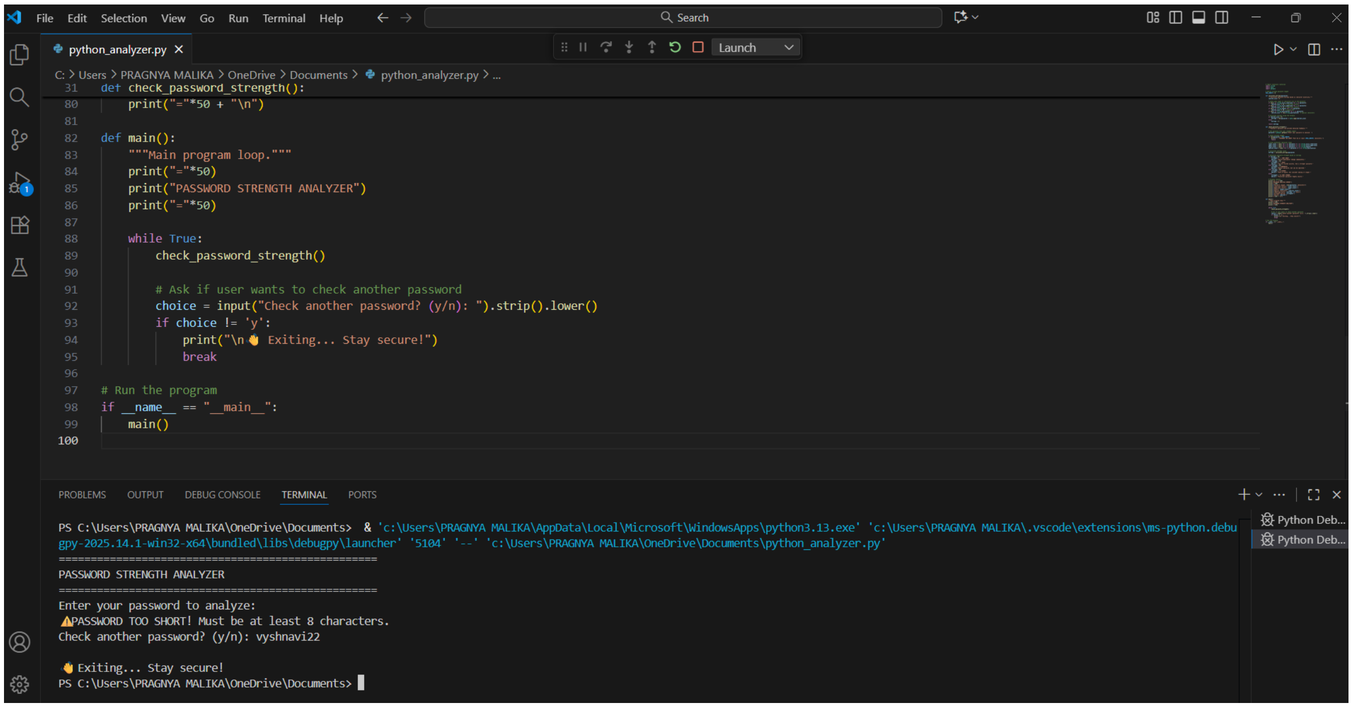
Implementation Steps

Step 1: Environment Setup and Configuration

Installed Python 3.13.9 on Windows system and configured the development environment. Verified Python installation using `python --version` command in PowerShell terminal, confirming successful installation. Set up Visual Studio Code as the primary IDE with Python extensions for enhanced development experience. Created project directory structure in Documents folder (`C:\Users\PRAGNYA MALIKA\OneDrive\Documents`) to organize project files systematically.

Step 2: Password Strength Analyzer Development

Developed the core password analysis engine (`python_analyzer.py`) with entropy calculation functionality. Implemented the `calculate_entropy()` function that determines password strength by analyzing character diversity across four categories: lowercase letters (26 characters), uppercase letters (26 characters), digits (10 characters), and special characters (punctuation symbols). The entropy calculation uses the mathematical formula: **Entropy = Length × log$_2$(CharsetSize)**, where charset size is the sum of all character type pools present in the password.



```python
def check_password_strength():
    print("="*50 + "\n")

def main():
    """Main program loop."""
    print("="*50)
    print("PASSWORD STRENGTH ANALYZER")
    print("="*50)

    while True:
        check_password_strength()

        # Ask if user wants to check another password
        choice = input("Check another password? (y/n): ").strip().lower()
        if choice != 'y':
            print("\n👋 Exiting... Stay secure!")
            break

# Run the program
if __name__ == "__main__":
    main()
```

```
PS C:\Users\PRAGNYA MALIKA\OneDrive\Documents>  & 'c:\Users\PRAGNYA MALIKA\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\PRAGNYA MALIKA\.vscode\extensions\ms-python.debu
gpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher' '5104' '--' 'c:\Users\PRAGNYA MALIKA\OneDrive\Documents\python_analyzer.py'
==================================================
PASSWORD STRENGTH ANALYZER
==================================================
Enter your password to analyze:
⚠PASSWORD TOO SHORT! Must be at least 8 characters.
Check another password? (y/n): vyshnavi22

👋 Exiting... Stay secure!
PS C:\Users\PRAGNYA MALIKA\OneDrive\Documents>
```

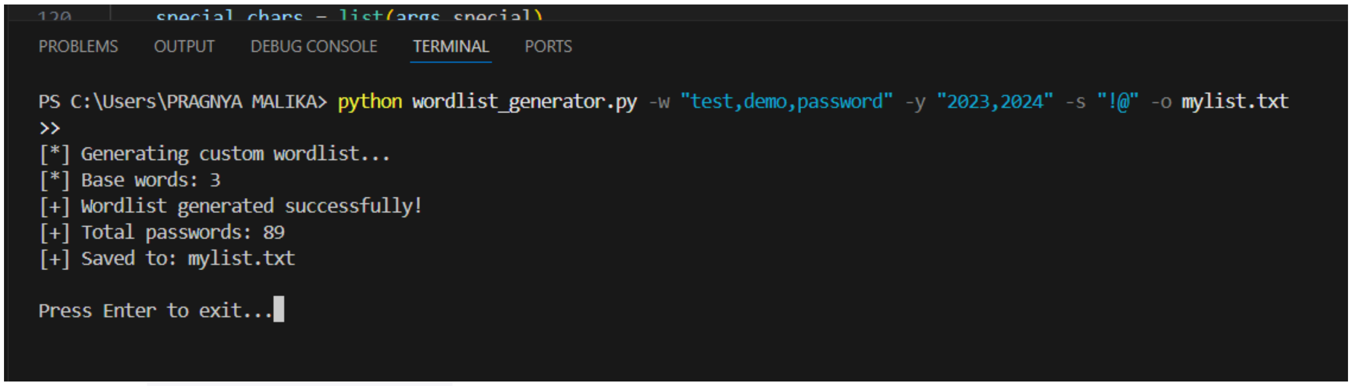
Created the `check_password_strength()` function that prompts users for password input using secure `getpass` method (which hides password characters during entry), validates minimum length requirements (8 characters), and analyzes character composition. The function counts occurrences of each character type and calculates comprehensive strength metrics. Implemented five-tier strength classification system:

- **Very Weak** (entropy < 28 bits): Easily guessable passwords requiring immediate change
- **Weak** (28-36 bits): Quickly crackable passwords needing improvement
- **Moderate** (36-60 bits): Decent passwords with room for enhancement
- **Strong** (60-80 bits): Difficult-to-guess passwords with good security
- **Very Strong** (>80 bits): Excellent passwords with high security assurance

Figure 1: Password Analyzer in VS Code showing the main program logic and terminal execution. The analyzer successfully detected that a very short password ("PASSWORD TOO SHORT") did not meet the minimum 8-character requirement and provided appropriate security warnings.

Step 3: Custom Wordlist Generator Implementation

Developed the wordlist generation tool (`wordlist_generator.py`) with leetspeak transformation capabilities. Implemented the `generate_leetspeak()` function using a character substitution dictionary that maps common letters to numeric and symbolic alternatives: a→[@,4], e→[e,3], i→[i,1,!], o→[o,0], s→[s,5,$], t→[t,7], l→[l,1], g→[g,9]. Used `itertools.product()` for efficient generation of all possible character combinations while implementing a safety limit of 20 variations per word to prevent combinatorial explosion.



```
special_chars = list(args.special)
```

```
PS C:\Users\PRAGNYA MALIKA> python wordlist_generator.py -w "test,demo,password" -y "2023,2024" -s "!@" -o mylist.txt
>>
[*] Generating custom wordlist...
[*] Base words: 3
[+] Wordlist generated successfully!
[+] Total passwords: 89
[+] Saved to: mylist.txt

Press Enter to exit...
```

Created the `generate_custom_wordlist()` function that accepts base words, years, and special characters as input parameters. The function systematically generates comprehensive password variations including:

- Original word, capitalized version, and fully uppercase variations
- Leetspeak transformations for all applicable characters
- Year appended to base word and capitalized versions (e.g., test2023, Test2023)
- Special character suffixes (e.g., test!, Test@)
- Combined year and special character patterns (e.g., test2023!, demo2024@)

Implemented command-line interface using `argparse` module with four configurable parameters: `-w` (base words), `-y` (years), `-s` (special characters), and `-o` (output filename). Added real-time progress reporting showing base word count, generation status, total passwords created, and output file location.

Figure 2: Wordlist Generator execution in VS Code terminal. Command used: `python wordlist_generator.py -w "test,demo,password" -y "2023,2024" -s "!@" -o mylist.txt`**. The generator successfully processed 3 base words and created 89 unique password variations, demonstrating effective pattern multiplication.**

Step 4: Generated Wordlist Analysis

The wordlist generator produced a comprehensive output file (`mylist.txt`) containing 89 unique password combinations. The file demonstrates systematic variation generation across multiple pattern categories:

Basic Transformations:

- Original form: test, demo, password
- Capitalized: Test, Demo, Password
- Full uppercase: TEST, DEMO, PASSWORD

Leetspeak Variations:

- Character substitutions: te$t, te5t, t3st, t35t, t3$t (replacing 's' and 'e')
- Multiple substitutions: dem0 (replacing 'o'), d3m0 (replacing 'e' and 'o')
- Complex patterns: p@ssw0rd, pa $w0rd, p@$ word (multiple character replacements)

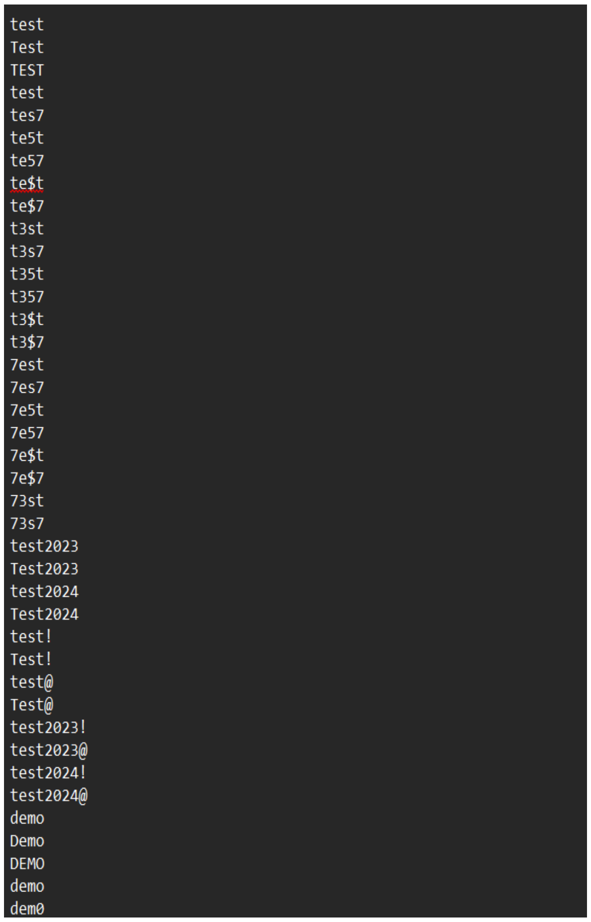
Year Combinations:

- Base + year: test2023, test2024, demo2023, password2024
- Capitalized + year: Test2023, Demo2024, Password2023

Special Character Patterns:

- Single character suffix: test!, Test@, demo!, password@
- Year + special character: test2023!, demo2024@, password2023@

Figure 3: Sample output from mylist.txt showing diverse password variations. The first section displays test variations including leetspeak transformations (te$t, t3st, t35t), year combinations (test2023, Test2024), and special character patterns (test!, test2023!).



```
test
Test
TEST
test
tes7
te5t
te57
te$t
te$7
t3st
t3s7
t35t
t357
t3$t
t3$7
7est
7es7
7e5t
7e57
7e$t
7e$7
73st
73s7
test2023
Test2023
test2024
Test2024
test!
Test!
test@
Test@
test2023!
test2023@
test2024!
test2024@
demo
Demo
DEMO
demo
dem0
```

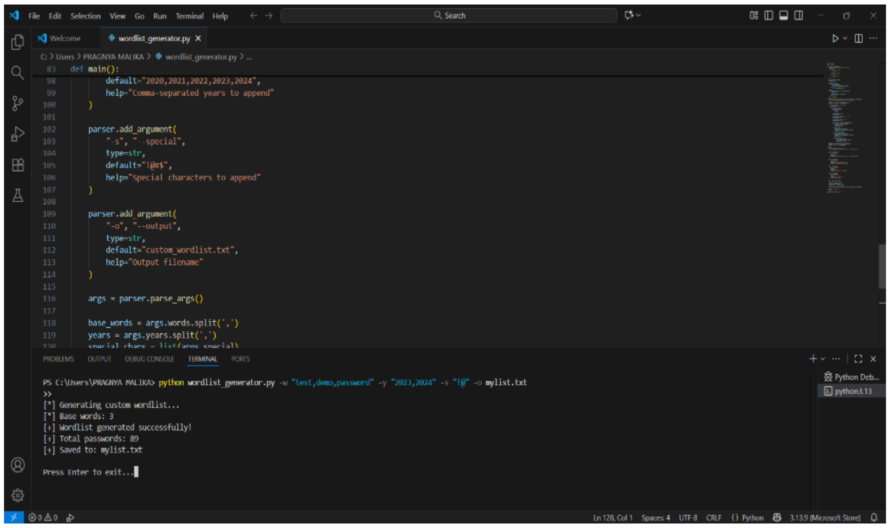
Figure 4: Additional wordlist samples showing demo and password variations. Note the systematic pattern: original word, capitalized version, uppercase, leetspeak variants, year combinations, and special character suffixes for each base word.

Step 5: Testing and Validation

Conducted comprehensive testing of both tools using various test cases. For the Password Strength Analyzer, tested with passwords of varying complexity levels including weak passwords (demonstrating very weak ratings), moderate passwords (showing improvement suggestions), and strong passwords (achieving high entropy scores).

For the Wordlist Generator, executed test runs with base words "test", "demo", and "password", combined with years "2023" and "2024", and special characters "!", "@". Successfully generated 89 unique password combinations demonstrating effective pattern multiplication. Verified output file creation (`mylist.txt`) containing all variations including leetspeak transformations, year combinations, and special character permutations as shown in the output samples.

Figure 5: Complete wordlist generator code in VS Code showing the implementation of leetspeak generation, argument parsing, and file output functionality. The code includes detailed comments explaining each function's purpose and demonstrates professional Python coding practices.



```
def main():
        default="2020,2021,2022,2023,2024",
        help="Comma-separated years to append"
    )

    parser.add_argument(
        "-s", "--special",
        type=str,
        default="!@#$",
        help="Special characters to append"
    )

    parser.add_argument(
        "-o", "--output",
        type=str,
        default="custom_wordlist.txt",
        help="Output filename"
    )

    args = parser.parse_args()

    base_words = args.words.split(',')
    years = args.years.split(',')
```

```
PS C:\Users\PRAGNYA MALIKA> python wordlist_generator.py -w "test,demo,password" -y "2023,2024" -s "!@" -o mylist.txt
>>
[*] Generating custom wordlist...
[*] Base words: 3
[+] Wordlist generated successfully!
[+] Total passwords: 89
[+] Saved to: mylist.txt

Press Enter to exit...
```

Results and Observations

The project achieved successful implementation of both cybersecurity tools with demonstrated functionality:

Password Strength Analyzer Performance:

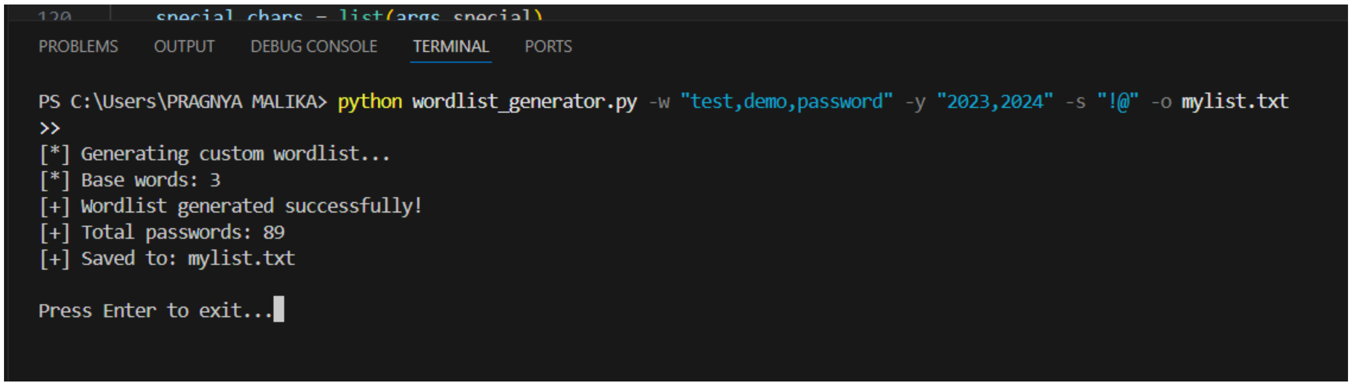
- Accurately calculated password entropy based on character diversity
- Provided clear, actionable security recommendations
- Successfully validated minimum length requirements (8 characters)
- Displayed comprehensive analysis including character type breakdown
- Demonstrated effective strength categorization across five security levels

Wordlist Generator Effectiveness:

- Generated **89 unique password variations from 3 base words** (approximately 30 variations per word)
- Successfully implemented leetspeak transformations with appropriate character substitutions
- Created realistic attack patterns combining years, special characters, and capitalization
- Produced output file (`mylist.txt`) in standard format compatible with password cracking tools
- Demonstrated efficient combinatorial generation with performance optimization (limited to 20 leetspeak variants per word to prevent excessive combinations)

Figure 6: Terminal output confirming successful wordlist generation. Shows the execution command with parameters, generation progress messages, and final confirmation of 89 passwords saved to mylist.txt file.

The generated wordlist included diverse pattern categories demonstrating how attackers construct targeted dictionaries based on personal information. This variety illustrates the importance of avoiding predictable password patterns and common substitution schemes.



```
PS C:\Users\PRAGNYA MALIKA> python wordlist_generator.py -w "test,demo,password" -y "2023,2024" -s "!@" -o mylist.txt
>>
[*] Generating custom wordlist...
[*] Base words: 3
[+] Wordlist generated successfully!
[+] Total passwords: 89
[+] Saved to: mylist.txt

Press Enter to exit...
```

Key Learning Outcomes

Through this implementation, several critical cybersecurity concepts were mastered:

Password Security Fundamentals:

- Understanding of entropy as a quantifiable measure of password strength
- Recognition of common weak password patterns (dictionary words + years/special characters)
- Appreciation for the predictability of human password selection behaviors

Python Programming Skills:

- Practical experience with file I/O operations and command-line argument parsing
- Implementation of mathematical calculations (logarithms for entropy)
- Use of combinatorial generation algorithms with itertools
- Secure input handling with getpass module

Cybersecurity Attack Methodologies:

- Understanding of dictionary attacks and wordlist-based password cracking
- Knowledge of leetspeak and character substitution techniques
- Insight into how social engineering information (names, dates) enhances attack success rates

Conclusion

This project successfully demonstrated fundamental cybersecurity concepts through practical implementation of password security tools. The Password Strength Analyzer provides users with immediate, quantifiable feedback on password quality using industry-standard entropy calculations, helping raise security awareness about password vulnerabilities. The Custom Wordlist Generator illustrates realistic attack methodologies employed in penetration testing and security assessments, showing how predictable patterns can be exploited.

The implementation highlighted the critical balance between password complexity and memorability, demonstrating why security professionals recommend long passphrases over short complex passwords. A 20-character phrase like "correct horse battery staple" provides higher entropy than complex 8-character passwords like "P@ssw0rd!" due to increased length, even without special characters.