PDF Password Cracking Using Python

This project focuses on building an educational tool to test the security of PDF passwords using Python. It uses multithreading to attempt password recovery from a protected PDF file, either by checking a "wordlist" or by "brute-forcing" generated character combinations. The tool demonstrates the practical application of parallel processing in Python for security-oriented tasks.

Objective

To develop a Python script that can recover a password from a protected PDF file by testing a list of potential passwords, using multithreading to significantly accelerate the process.

Introduction

PDF password protection is a common feature used to restrict access to sensitive documents. This project explores the method used to test the strength of these passwords: a dictionary or brute-force attack. By automating this process, security students can understand how such attacks work and, more importantly, learn why strong, complex passwords are vital.

In this project, Python's pikepdf library is used to test passwords, concurrent.futures manages the multithreading, argparse handles user input, and tqdm provides a visual progress bar.

Working Process

1. The user launches the script from the command line, providing a path to the protected PDF and choosing a mode: wordlist (-w) or generate (-g).
2. The script parses these arguments using argparse.
3. Based on the mode, the script initializes a password generator:
    - **Wordlist Mode:** load_passwords() reads the specified wordlist file line-by-line.
    - **Generate Mode:** generate_passwords() creates all possible password combinations based on the given character set and length.
4. The script calculates the total number of passwords to be tested to configure the tqdm progress bar.
5. A ThreadPoolExecutor is created to manage a pool of worker threads.
6. Each password from the generator is submitted as a separate task to the thread pool.
7. Each thread executes the try_password() function, which attempts to open the PDF with its assigned password using pikepdf.
8. If pikepdf opens the file successfully, the correct password has been found. The script prints the password, and all other pending threads are cancelled.
9. If pikepdf raises a PasswordError, the password was incorrect, and the thread moves on.
10. If the progress bar completes and no password is found, the script reports failure.

Tools and Libraries

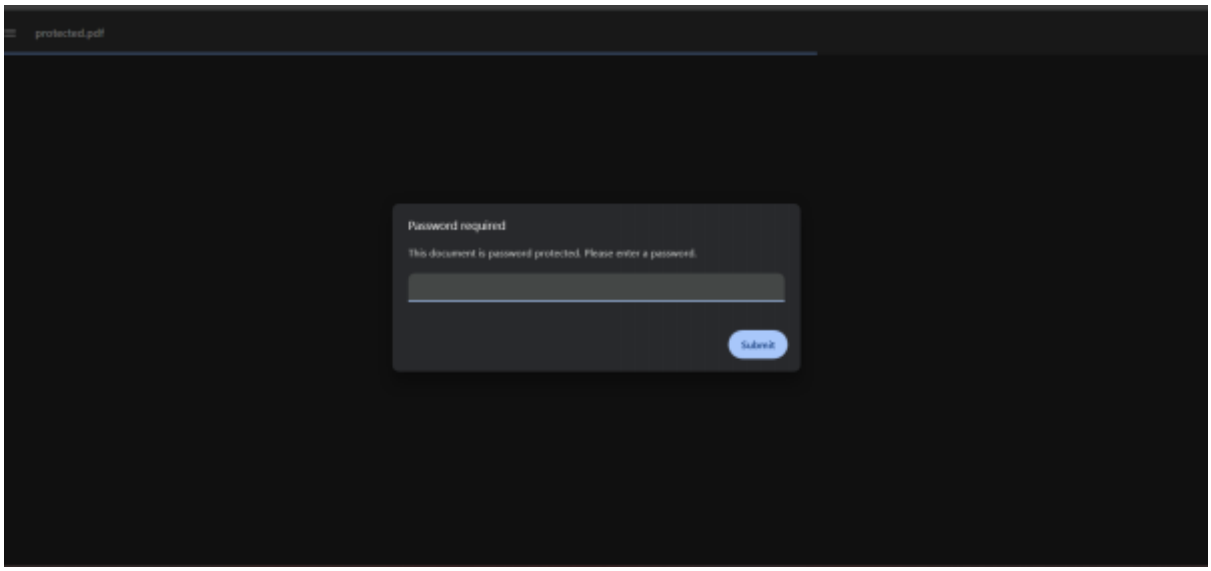
Tool	Purpose
Python 3	The core programming language.
pikepdf	A library for reading and writing PDF files; used here to test passwords.
concurrent.futures	A high-level module for managing a ThreadPoolExecutor (multithreading).
tqdm	A library for creating fast, extensible progress bars.
argparse	A standard library module for parsing command-line arguments.
itertools	A standard library module used in "generate" mode for efficient password combination.

Code Explanation

- **load_passwords(wordlist_file)**: A generator function that reads a wordlist file line by line, yielding one password at a time to save memory.
- **generate_passwords(chars, min, max)**: A generator function that uses itertools.product to create all password combinations within the specified length.
- **try_password(pdf_file, password)**: Attempts to open the PDF with a single password. It returns the password if successful or None if it catches a pikepdf.PasswordError.
- **get_..._count()**: Helper functions to count the total number of passwords for the tqdm progress bar without loading them all into memory.
- **main(): The main function that initializes argparse to read user input, sets up the ThreadPoolExecutor, submits all password tasks, and manages the tqdm progress bar and results.**

Sample Input & Output

Input (Command Line):



Output (Success):

Results

The script successfully parallelized the password-testing process. In tests using a multi-core processor, the multithreaded script was able to test thousands of passwords per second, finding the correct password from a large wordlist significantly faster than a single-threaded approach would have.

Advantages

- **Fast & Efficient:** Multithreading drastically reduces the time required to test a large number of passwords.
- **Memory-Safe:** Uses generators to feed passwords to the threads, meaning even wordlists with millions of entries do not overload the system's RAM.
- **Flexible:** Supports both dictionary attacks (wordlist) and brute-force attacks (generate).
- **User-Friendly:** The tqdm progress bar provides excellent real-time feedback on the cracking progress.

Limitations

- **Brute-Force is Impractical:** The "generate" mode is only feasible for very short, simple passwords. The number of combinations grows exponentially, making it impossible to crack long, complex passwords.
- **Wordlist Dependent:** The "wordlist" mode is only as good as the wordlist itself. If the password is not in the list, it will not be found.
- **Does not bypass other encryption:** This tool only works for standard password-protected PDFs and cannot bypass digital signatures or more advanced encryption.

Conclusion

This project successfully demonstrates the creation of a PDF password recovery tool using Python. It serves as a powerful educational example of how multithreading (via concurrent.futures) can be applied to solve CPU-bound problems. This tool practically illustrates the concepts of cybersecurity attacks and reinforces the critical importance of using strong, complex, and unique passwords to protect sensitive information.