INLIGHN TECH

EXPERIENCE. LEARN. THRIVE

Prerequisites for the PDF Cracker Tool Project

By: **Inlighn Tech**

Overview

The PDF Cracker Tool project requires students to develop a Python script that decrypts a password-protected PDF by either generating passwords on the fly or testing a wordlist, using multithreading for efficiency. The solution employs the pikepdf library to attempt decryption, tqdm for progress tracking, and concurrent.futures for parallel execution, with command-line arguments parsed via argparse. To write this script independently, students must master several key concepts and skills from the curriculum. These prerequisites, paired with practical Python code examples, ensure students understand how PDF cracking works and can implement the script effectively.

Prerequisite Knowledge and Skills

1. **File Input/Output (I/O) Basics**
   - **What to Know**:
     - Reading text files (e.g., wordlists) line-by-line using open() in text mode ('r').

        - Context managers (with statements) for safe file handling.
    - **How It Applies**:
        - The script reads passwords from a wordlist_file in load_passwords() to test against the PDF.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lesson 11 (IO with Basic Files).
        - "WORKING WITH DATA STRUCTURES & FILE HANDLING" Lesson 10 (Opening and Reading Files and Folders Python OS Module).
    - **Practical Prep**:
        - Practice reading a wordlist file.
    - **Python Code Block**:

```
# Read a wordlist file line-by-line
wordlist = "passwords.txt"

# Create a sample wordlist
with open(wordlist, "w") as f:
   f.write("pass1\npass2\npass3\n")

# Read it back
with open(wordlist, "r") as f:
   passwords = [line.strip() for line in f]
   print("Passwords loaded:", passwords)
```

    -
        - **Run It**: Save as file_io_test.py, run with python file_io_test.py. Creates and reads a simple wordlist, preparing for load_passwords().

2. **Functions, Generators, and Itertools**
    - **What to Know**:
        - Define functions with def and use yield for generators to produce values lazily (e.g., generate_passwords).
        - Use itertools.product() to generate combinations of characters for passwords.
    - **How It Applies**:

        - generate_passwords() creates password combinations on the fly, while load_passwords() yields passwords from a file, both as generators for efficient memory use.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lessons 17-19 (Introduction to Functions, def Keyword, Basics of Functions).
        - Note: Generators and itertools may require a brief intro, as they're not explicitly listed but implied in advanced Python use.
    - **Practical Prep**:
        - Test generators and itertools.
    - **Python Code Block**:

```
# Generate passwords with itertools and yield
import itertools

def generate_words(chars, length):
    for combo in itertools.product(chars, repeat=length):
        yield ''.join(combo)

chars = "ab"
for word in generate_words(chars, 2):
    print(word)  # Outputs: aa, ab, ba, bb
```

    -
        - **Run It**: Save as generator_test.py, run with python generator_test.py. Shows password generation, like generate_passwords().

3. **Exception Handling**
    - **What to Know**:
        - Use try/except to catch specific errors (e.g., pikepdf._core.PasswordError for wrong passwords).
        - Return values (e.g., None) to signal failure without crashing.
    - **How It Applies**:
        - try_password() attempts PDF decryption and catches incorrect password errors, returning None if unsuccessful.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lesson 21 (Errors and Exception Handling).

- **Practical Prep**:
  - Practice catching specific exceptions.
- **Python Code Block**:

```
# Handle specific errors
def divide(a, b):
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        print("Division by zero!")
        return None

print(divide(10, 2))  # Outputs: 5.0
print(divide(10, 0))  # Outputs: Division by zero! None
```

-
  - **Run It**: Save as error_test.py, run with python error_test.py. Mimics try_password()'s error handling.

4. **Command-Line Arguments with argparse**
   - **What to Know**:
     - The argparse module parses command-line arguments with options (e.g., --wordlist, --generate).
     - Define arguments, set defaults, and access them via args.attribute.
   - **How It Applies**:
     - The script uses argparse to handle inputs like pdf_file, wordlist, and password generation settings.
   - **Curriculum Source**:
     - Not explicitly listed; assumes basic Python knowledge or a brief intro (extends "SSH Cracker" Lesson 4: User Input).
   - **Practical Prep**:
     - Test argparse usage.
   - **Python Code Block**:

```
# Parse command-line arguments
import argparse
```

```
parser = argparse.ArgumentParser(description="Sample argument
parser")
parser.add_argument("name", help="Your name")
parser.add_argument("--age", type=int, default=20, help="Your age")
args = parser.parse_args()

print(f"Name: {args.name}, Age: {args.age}")
```

-
  - **Run It**: Save as args_test.py, run with python args_test.py Alice --age 25. Shows argument parsing, like the script's setup.

5. **Working with pikepdf Library**
   - **What to Know**:
     - pikepdf is a Python library for PDF manipulation; pikepdf.open() attempts to open PDFs, optionally with a password.
     - Raises PasswordError if the password is incorrect.
   - **How It Applies**:
     - try_password() uses pikepdf.open() to test passwords against the encrypted PDF.
   - **Curriculum Source**:
     - Implied in "PDF Cracker Tool (Solution)" Lessons (e.g., Lesson 1: Function to generate passwords on the fly).
     - Note: Students need pip install pikepdf.
   - **Practical Prep**:
     - Test basic pikepdf usage (requires a password-protected PDF).
   - **Python Code Block**:

```
# Test opening a PDF with pikepdf
import pikepdf

pdf_file = "protected.pdf"  # Replace with a real protected PDF
password = "wrongpass"  # Replace with a test password
try:
    with pikepdf.open(pdf_file, password=password) as pdf:
        print(f"Opened {pdf_file} with {len(pdf.pages)} pages")
except pikepdf._core.PasswordError:
```

```
print(f"Password '{password}' failed for {pdf_file}")
```

-
    - **Run It**: Install pikepdf (pip install pikepdf), use a protected PDF (create one with the PDF Protection Tool), save as pikepdf_test.py, run with python pikepdf_test.py. Introduces PDF decryption attempts.

6. **Multithreading with concurrent.futures**
   - **What to Know**:
     - concurrent.futures.ThreadPoolExecutor manages a pool of threads for parallel task execution.
     - Submit tasks with executor.submit() and retrieve results with future.result().
   - **How It Applies**:
     - decrypt_pdf() uses ThreadPoolExecutor to test multiple passwords simultaneously, speeding up cracking.
   - **Curriculum Source**:
     - Builds on "Network Scanner" Lesson 4 and "Port Scanner" Lesson 3 (threading concepts); concurrent.futures is an advanced extension.
   - **Practical Prep**:
     - Experiment with thread pools.
   - **Python Code Block**:

```
# Use ThreadPoolExecutor for parallel tasks
from concurrent.futures import ThreadPoolExecutor
import time

def slow_task(n):
    time.sleep(1)  # Simulate work
    return f"Task {n} done"

with ThreadPoolExecutor(max_workers=2) as executor:
    futures = [executor.submit(slow_task, i) for i in range(4)]
    for future in futures:
        print(future.result())
```

-
    - **Run It**: Save as threadpool_test.py, run with python threadpool_test.py. Shows parallel execution, like decrypt_pdf()'s threading.

7. **Progress Tracking with tqdm**
    - **What to Know**:
        - tqdm creates progress bars to visualize task completion.
        - Use with loops or iterators, setting total for accurate progress.
    - **How It Applies**:
        - The script wraps password attempts with tqdm to show cracking progress.
    - **Curriculum Source**:
        - Not explicitly taught; assumes a brief intro (common in Python projects).
        - Note: Students need pip install tqdm.
    - **Practical Prep**:
        - Test a progress bar.
    - **Python Code Block**:

```
# Display a progress bar with tqdm
from tqdm import tqdm
import time

for i in tqdm(range(5), desc="Processing", unit="step"):
    time.sleep(0.5)  # Simulate work
print("Done!")
```

-
    - **Run It**: Install tqdm (pip install tqdm), save as tqdm_test.py, run with python tqdm_test.py. Introduces progress tracking.

How PDF Cracking Works

- **Concept**:

  - PDF cracking attempts to decrypt a password-protected PDF by brute-forcing passwords—either from a wordlist or generated combinations—until the correct one is found.
  - This simulates real-world attacks on weak encryption to test security.
- **Script Workflow**:
  - Parse command-line arguments for PDF file, wordlist, or generation options.
  - Load or generate passwords as an iterable.
  - Use ThreadPoolExecutor to test passwords in parallel with pikepdf.
  - Track progress with tqdm and return the correct password when found.
  - Handle errors (e.g., wrong passwords) gracefully.
- **Why Multithreading?**:
  - Speeds up cracking by testing multiple passwords concurrently, critical for large wordlists or brute-force attempts.

How to Write the PDF Cracker Script

Using these prerequisites, students can build the script step-by-step:

1. **Setup**:
   - Import pikepdf, tqdm, itertools, string, concurrent.futures, and argparse.
   - Define generate_passwords() and load_passwords() as generators.
2. **Try Password**:
   - Write try_password() to test a password with pikepdf.open(), catching PasswordError.
3. **Decrypt Function**:
   - Create decrypt_pdf() with ThreadPoolExecutor, submit tasks, and use tqdm for progress.
4. **Argument Parsing**:
   - Set up argparse with required (pdf_file) and optional arguments (e.g., --wordlist, --generate).
5. **Main Execution**:

- Use if __name__ == "__main__": to parse args, select password source, and run decryption.

Notes for Students

- **Setup**: Install dependencies (pip install pikepdf tqdm), prepare a password-protected PDF (e.g., from the PDF Protection Tool), and create a small wordlist.txt (e.g., pass1, 123, test).
- **Engagement**: Run each code block to practice file reading, password generation, threading, and progress tracking. Test the final script with python script.py protected.pdf --wordlist wordlist.txt.
- **Tips**: Start with a small wordlist or short password lengths (e.g., --max_length 2) to see results quickly.

Jump to the PDF Cracker Tool Project Description if you are ready now.
