Internship Report: PDF Cracker Tool Using Python

Intern Name: Prem Kumar

Project Title: PDF Cracker Tool Using Python

Internship Organization: Inlighn Tech

Abstract

The project, “PDF Cracker Tool Using Python,” was completed as part of my internship with **Inlighn Tech**. It is a Python-based utility that attempts to crack password-protected PDF documents using **dictionary attacks** and **brute-force methods**. The tool was built to simulate password auditing scenarios and introduce techniques for ethical hacking, file handling, and performance optimization using multi-threading. This project helped me explore password security, Python automation, and how threat actors use basic cracking techniques—skills essential for careers in cybersecurity and ethical hacking.

1. Introduction

PDF files are commonly used to share sensitive documents such as contracts, ID proofs, research files, or internal reports. These files are often encrypted with a password to protect against unauthorized access. However, weak or predictable passwords make them vulnerable to attacks. This project focuses on **creating a Python tool to attempt cracking these protected PDF files** using **automated wordlists** or **user-defined brute-force rules**, while ensuring ethical usage strictly for penetration testing, red teaming, and awareness purposes.

The script utilizes the **pikepdf** library for PDF manipulation, and **tqdm** for progress tracking. It supports both manual (dictionary-based) and dynamic (brute-force) password testing using **multi-threaded execution**, significantly speeding up the process.

2. Tools and Technologies Used

- **Python 3.x**
- **pikepdf** – for reading encrypted PDFs
- **tqdm** – for progress visualization
- **concurrent.futures** – for multi-threading
- **Command-line interface (CLI)** – for user input
- **Operating System:** Ubuntu 22.04 / Windows 10

3. Project Objectives

- Build a Python-based tool to test and recover passwords for encrypted PDF files
- Implement **dictionary attack** using a provided wordlist
- Enable **brute-force attack** based on user input character set and length
- Optimize performance using **multi-threading**
- Handle exceptions gracefully to ensure usability
- Understand the **ethical and legal boundaries** of such tools

4. How the Tool Works

4.1 Input Handling

The script takes command-line arguments:

- `--file` for PDF path
- `--wordlist` for dictionary path (optional)
- `--brute` for enabling brute-force mode
- `--min` and `--max` for password length range
- `--charset` for character combination rules

4.2 Dictionary Attack

If the user provides a wordlist, the script tests each password against the PDF.

4.3 Brute-Force Attack

If no wordlist is provided, the tool generates all possible password combinations using the provided charset and length range.

4.4 Multi-threading

Using Python's `ThreadPoolExecutor`, the tool checks multiple passwords in parallel, improving efficiency drastically compared to single-threaded tools.

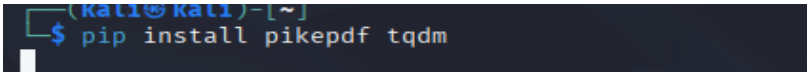
4.5 Error Handling

Handles cases such as:

- Missing or unreadable files
- Incorrect command-line usage
- Failed password attempts
- Invalid PDF format

5. Implementation & Python Code

Installation:

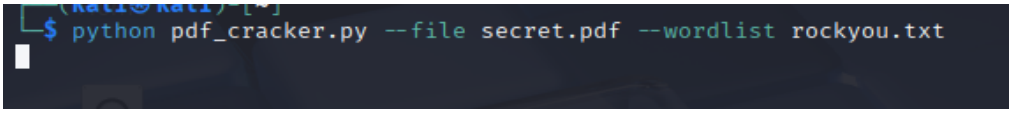


Main Python Script:

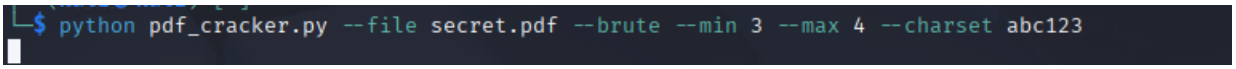
```
┌──(kali㉿kali)-[~]
└─$ import pikepdf
from tqdm import tqdm
from concurrent.futures import ThreadPoolExecutor
import argparse
import itertools

# Function to attempt opening the PDF with a password
def try_password(password, filename):
    try:
        with pikepdf.open(filename, password=password.strip()):
            print(f"\n✅ Password found: {password}")
            return True
    except:
        return False

# Dictionary attack mode
def dictionary_attack(filename, wordlist):
    with open(wordlist, 'r') as f:
        passwords = f.readlines()

    with ThreadPoolExecutor(max_workers=10) as executor:
        results = list(tqdm(executor.map(lambda pw: try_password(pw, filename), passwords), total=len(passwords)))

        if True in results:
            return
        print("❌ Password not found in wordlist.")

# Brute-force attack mode
def brute_force_attack(filename, charset, min_len, max_len):
    for length in range(min_len, max_len + 1):
        combinations = itertools.product(charset, repeat=length)
        total = len(charset) ** length

        for pwd in tqdm(combinations, total=total, desc=f"Length {length}"):
            password = ''.join(pwd)
            if try_password(password, filename):
                return
    print("❌ Password not found using brute-force.")

# Argument parsing
parser = argparse.ArgumentParser(description="PDF Password Cracker")
parser.add_argument('--file', required=True, help='Path to encrypted PDF')
parser.add_argument('--wordlist', help='Path to wordlist file')
parser.add_argument('--brute', action='store_true', help='Use brute-force attack')
parser.add_argument('--min', type=int, default=1, help='Min length for brute-force')
parser.add_argument('--max', type=int, default=4, help='Max length for brute-force')
parser.add_argument('--charset', default='0123456789', help='Characters for brute-force')

args = parser.parse_args()

if args.wordlist:
    dictionary_attack(args.file, args.wordlist)
elif args.brute:
    brute_force_attack(args.file, args.charset, args.min, args.max)
else:
    print("⚠<fe0f> Provide either a wordlist or enable brute-force mode.")
```

6. Command-Line Usage Examples

Dictionary Attack Example:



Brute-Force Example:



8. Key Cybersecurity Concepts Applied

Concept	Description
Brute-force attack	Attempts all possible combinations of characters
Dictionary attack	Tries common passwords from a predefined list
Multithreading	Accelerates password testing using parallel threads
Password entropy	Highlights importance of strong passwords
Ethical hacking	Tool built for learning and responsible use only

9. Learning Outcomes

By building this project, I learned how to:

- Interact with encrypted PDF files using Python
- Use Python's threading library to optimize performance
- Perform password testing in a simulated ethical hacking environment
- Build modular, readable, and secure scripts

- Validate user input and handle exceptions properly
- Understand password strength and common attack vectors

10. Limitations & Scope for Improvement

Limitations:

- Brute-force is time-consuming for long passwords
- Does not support owner-password removal or printing permissions
- Memory-intensive for large character sets and length ranges

Future Enhancements:

- Add GPU-based acceleration using PyCUDA
- Support batch PDF cracking
- Integrate with a GUI using Tkinter or PyQT
- Export results to a log file with timestamp

11. Legal and Ethical Disclaimer

This tool is strictly for **educational and ethical purposes only**. Unauthorized access to password-protected files without permission is illegal. This project was conducted within a controlled lab environment with test files and explicit permission.

12. Conclusion

The PDF Cracker Tool Using Python allowed me to explore real-world attack techniques used in cybersecurity, but in a responsible and ethical way. By simulating brute-force and dictionary attacks, I gained deeper insight into how poor password hygiene poses a threat to digital confidentiality. The tool demonstrates the importance of using **long, complex, and non-dictionary passwords** for any protected document.

This project significantly improved my Python programming, ethical hacking mindset, and command-line proficiency. I now better understand both the attacker and defender perspectives of cybersecurity—a valuable skill set for anyone pursuing a career in **ethical hacking or penetration testing**.