PDF Protection Tool

Introduction

In modern digital environments, the security of confidential documents is paramount. This tool enhances the basic file security concepts by implementing a multi-modal solution for PDF encryption using the **pypdf** library. Beyond simple encryption, the tool integrates **secrets** module to offer automated generation of secure passwords, effectively mitigating the human risk associated with choosing weak credentials.

Working Process (Interactive & Automated Security)

The tool operates based on the user's input mode, managing file validity, password generation, and encryption in a robust sequence:

1. **Argument Handling:** The script first processes command-line arguments, determining the necessary execution mode (Interactive, Auto-Generate, or Direct Password).
2. **Interactive Selection:** If running interactively, the user is prompted to choose between **auto-generating** a strong, random password or **entering a custom password** (with length warnings).
3. **Path Validation:** The tool rigorously checks if the input PDF exists, if the input file is a PDF, and if the output path is writable.
4. **Encryption Core:** It reads the original PDF, copies all pages to a PdfWriter object, and applies **AES-256 encryption** using the final selected password.
5. **Robust Output:** Provides detailed confirmation including input/output paths, page count, and file size, ensuring the user has all necessary verification data.

Key Concepts Covered

Concept	Description
Cryptographic Security (secrets)	Uses Python's secrets module, ensuring that auto-generated passwords are cryptographically secure and suitable for high-security environments.
Interactive CLI	Implements the getpass library for secure, invisible password entry and uses conditional logic (argparse) to guide the user flow.

AES-256 Encryption	Leverages modern Advanced Encryption Standard (AES) 256-bit to ensure the encrypted file is resilient against current cracking methods.
Robust File Handling	Includes extensive validation (e.g., validate_paths()) to handle non-existent files, non-PDF files, and permission issues gracefully.
Complexity Enforcement	The generate_strong_password() function enforces the inclusion of uppercase, lowercase, digits, and symbols.

Command-Line Flags (Execution Modes)

The tool's versatility is enabled by its command-line arguments, which allow users to select their desired input method without going through the default interactive menu.

Flag	Full Argument	Mode	Purpose
(None)	(Positional Args Only)	**Interactive** (Default)	Triggers the secure, step-by-step interactive menu to choose a password (recommended for first-time use).
-a	--auto	**Auto-Generate**	Skips the interactive prompt and forces the script to immediately generate and use a strong, random 16-character password.
-p	password	**Direct/Manual**	Allows the user to supply the password directly on the command line. (Less secure due to command history risk.)

Execution Examples

The following examples illustrate the primary ways the tool can be executed from the command line:

1. Interactive Mode (Default)

Running the script without any password flags triggers the safest, step-by-step process.

Command:

python pdf_protector.py input.pdf output.pdf

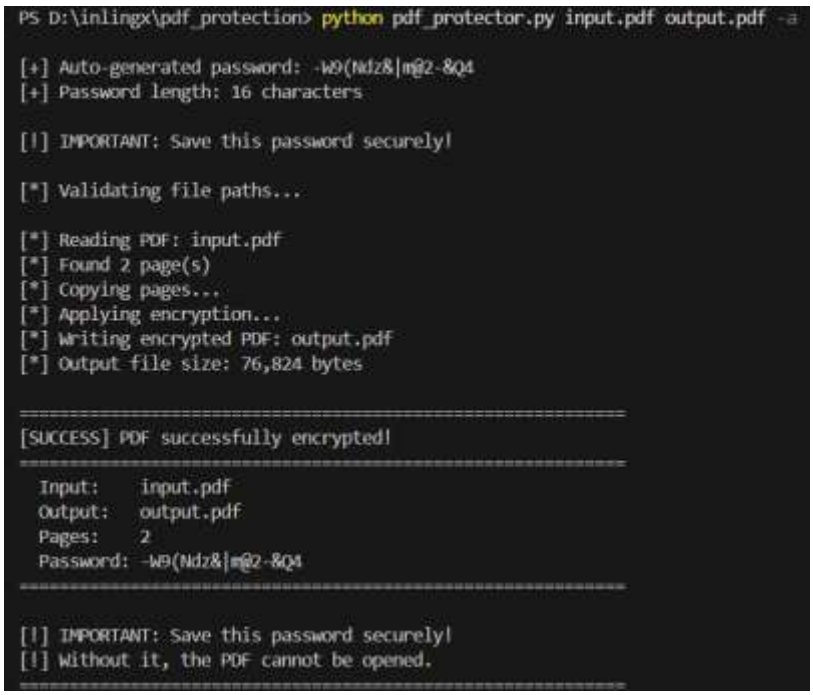
```
PS D:\inlingx\pdf_protection> python pdf_protector.py input.pdf output.pdf

============================================================
PASSWORD SELECTION
============================================================

Choose password option:
  [1] Auto-generate a strong password (Recommended)
  [2] Enter my own custom password
============================================================
```

2. Auto-Generate Mode (Scripted Security)

Using the -a flag skips prompts and generates a password using the default 16-character length.

Command:

python pdf_protector.py input.pdf output.pdf -a

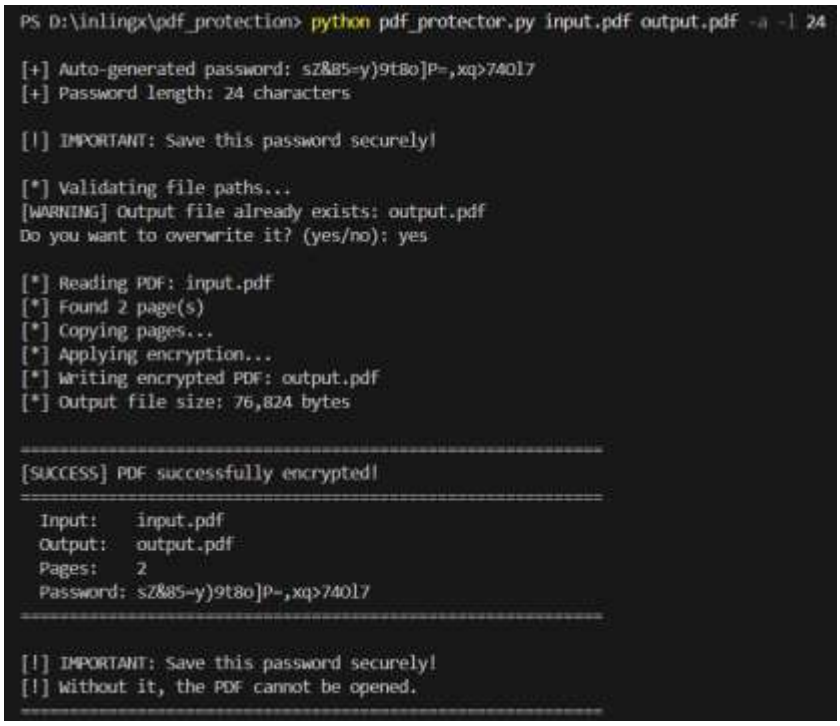


3. Auto-Generate with Custom Length

Using -a combined with -l specifies the exact length of the required strong password.

Command:

python pdf_protector.py input.pdf output.pdf -a -l 24



Code Explanation

Function	Purpose
generate_strong_password()	Creates a guaranteed strong password, checking complexity within a loop for high-entropy randomization.
interactive_password_selection()	Manages the interactive prompt, allowing the user to select auto-generation or safe manual password entry.
validate_paths()	Crucial pre-check ensuring the input file is accessible and the output location is writable, providing clear error messages.
protect_pdf()	Contains the core pypdf logic: reading pages, applying writer.encrypt(..., algorithm="AES-256"),

	and writing the output file with comprehensive success reporting.
main()	Orchestrates the entire script, parsing arguments and choosing the correct execution path (Direct, Auto, or Interactive).

Results

This revised tool is an effective solution for securing documents. By integrating an **interactive password choice** and using the **secrets** module, the project successfully elevates document security from a basic feature to a robust defense mechanism. The script's detailed error checking ensures a reliable and professional user experience, fulfilling all educational objectives related to file handling, security, and argument parsing.