PassDefender ToolKit - Complete Code Documentation

Overview

PassDefender ToolKit is a Python-based cybersecurity GUI application that analyzes password strength and generates custom wordlists. It combines the `zxcvbn` library for password analysis with custom algorithms for wordlist generation.

Part 1: Imports & Setup

Code Snippet:

```
import tkinter as tk
from tkinter import messagebox, filedialog
from zxcvbn import zxcvbn
import itertools
```

What It Does:

- tkinter: GUI framework for creating windows, buttons, text fields
- messagebox: Pop-up notifications (errors, success messages)
- filedialog: File save dialog for exporting wordlists
- zxcvbn: Industry-standard password strength library
- itertools: Creates permutations and combinations for wordlist generation

Part 2: Colors & Styling

Code Snippet:

```
BG_MAIN      = "#201133"    # Dark purple background
ACCENT_PURP  = "#6c41a1"    # Purple accents
ACCENT_BLUE  = "#1defff"    # Neon blue for titles
ACCENT_GRN   = "#30e67a"    # Neon green for buttons/success
WARN_COLOR   = "#da277a"    # Pink/red for warnings
OK_COLOR     = "#ffd157"    # Yellow for fair/ok status
STRONG_COLOR = "#7aecb8"    # Teal for strong passwords
FONT_MAIN    = ("Consolas", 12)
FONT_LG      = ("Cascadia Code", 18, "bold")
```

What It Does:

- Defines all colors for the dark-themed "hacker" aesthetic
- Purple = cybersecurity vibe, Neon green/blue = terminal feel
- Ensures consistent styling across all GUI elements

Part 3: Password Strength Analyzer

3.1 Main Analysis Function

Code Snippet:

```
def analyze_password():
    pwd = password_entry.get()  # Get password from input field
    if not pwd:
        messagebox.showerror("Error", "Please enter a password.")
        return

    result = zxcvbn(pwd)  # Run zxcvbn analysis
    score = result["score"]  # Get strength score (0-4)
    guesses = result["guesses"]  # How many guesses to crack
    feedback = result.get("feedback", {})  # Get feedback dict
    crack_time = result["crack_times_display"]["offline_fast_hashing_1e10_per_second"]
```

What It Does:

1. Retrieves password from GUI input field
2. Passes it to `zxcvbn()` which returns comprehensive analysis
3. Extracts key info: score (0-4), guesses required, estimated crack time
4. The `score` determines: Very Weak (0), Weak (1), Fair (2), Good (3), Strong (4)

3.2 Custom Issue Detection

Code Snippet:

```
issues = []
if score <= 1:  # For very weak/weak passwords
    if len(pwd) < 8:
        issues.append("Password is too short (<8 chars).")
    if pwd.lower() in common_words:
        issues.append("Password is a common word.")
    if pwd.isdigit():
        issues.append("Password only contains numbers.")
    if pwd.islower():
        issues.append("No uppercase letters present.")
    if pwd.isalpha():
        issues.append("No digits or special characters used.")
elif score <= 2:  # For fair passwords
    if not any(c.isupper() for c in pwd):
        issues.append("Try adding uppercase letters.")
    if not any(c.isdigit() for c in pwd):
        issues.append("Consider adding numbers.")
    if not any(not c.isalnum() for c in pwd):
        issues.append("Try adding special characters (!@# etc).")
```

What It Does:

- For very weak passwords (score 0-1): Check for obvious problems
  - Is it too short? (less than 8 chars)
  - Is it a common word from our list?
  - Is it only numbers or only letters?
  - Does it lack uppercase, digits, or symbols?
- For fair passwords (score 2): Give improvement suggestions
- Each issue is added to the `issues` list and displayed to user

3.3 Display Results in GUI

Code Snippet:

```
strength_labels = [
    ("Very Weak", WARN_COLOR),
    ("Weak", WARN_COLOR),
    ("Fair", OK_COLOR),
    ("Good", STRONG_COLOR),
    ("Strong", ACCENT_GRN)
]
label, color = strength_labels[score]

strength_label.config(text=f"Strength: {label}  [Score: {score}/4, Guesses: {guesses:,}]",
fg=color)

details_text.config(state="normal")  # Make text box editable temporarily
details_text.delete(1.0, tk.END)  # Clear previous text
details_text.insert(tk.END, f"Estimated crack time: {crack_time}\n", "main")

if issues:
    details_text.insert(tk.END, "\nIssues:\n", "warn")
    for iss in issues:
        details_text.insert(tk.END, f" • {iss}\n", "warn")

if suggestions:
    details_text.insert(tk.END, "\nSuggestions:\n", "ok")
    for sug in suggestions:
        details_text.insert(tk.END, f" • {sug}\n", "ok")

details_text.config(state="disabled")  # Lock text box after display
```

What It Does:

- Maps score (0-4) to label ("Very Weak" → "Strong") and color
- Displays strength with color-coded GUI label
- Shows crack time estimate in detail text box
- Lists all issues in pink/red (warnings)
- Lists all suggestions in yellow/green (actionable advice)
- Locks text box to prevent user editing

3.4 Show/Hide Password Toggle

Code Snippet:

```
def toggle_password():
    if password_entry.cget('show') == '*':  # If currently hidden
        password_entry.config(show='')  # Show as plain text
        show_button.config(text='Hide')
    else:  # If currently visible
        password_entry.config(show='*')  # Hide as asterisks
        show_button.config(text='Show')
```

What It Does:

- Toggles password visibility
- When `show='*'`, input shows asterisks (hidden)
- When `show=''`, input shows actual characters (visible)
- Button text changes between "Show" and "Hide"

Part 4: Custom Wordlist Generator

4.1 Generate Wordlist Function

Code Snippet:

```
def generate_wordlist():
    name = name_entry.get().strip().lower()
    pet = pet_entry.get().strip().lower()
    year = year_entry.get().strip()
    place = place_entry.get().strip().lower()
    number = number_entry.get().strip()

    base = [w for w in [name, pet, year, place, number] if w]
    # base = ['john', 'sparky', '2007', 'paris', '14']

    if not base:
        messagebox.showerror("Error", "Enter at least one field to generate a wordlist.")
        return

    wordlist = set(base)  # Start with base words
```

What It Does:

1. Retrieves all five input fields and converts to lowercase
2. Creates `base` list with only non-empty values
3. Initializes `wordlist` set with these base words
4. Sets prevent duplicate entries automatically

4.2 Apply Transformation Patterns

Code Snippet:

```
suffixes = ["", "123", "!", "?", "@", "_", "2025", "2024"]
leet_map = {"a": "4", "e": "3", "i": "1", "o": "0", "s": "5"}

for w in base:
    # Add base word with all suffixes
    for suf in suffixes:
        wordlist.add(w + suf)              # john, john123, john!
        wordlist.add(w.capitalize() + suf) # John, John123, John!
        wordlist.add(w.upper() + suf)      # JOHN, JOHN123, JOHN!

    # Leetspeak transformation
    leet = "".join(leet_map.get(c, c) for c in w)
    # 'john' → 'j0hn', 'paris' → 'p4r15'
    wordlist.add(leet)
    wordlist.add(leet.capitalize())

    # Reverse pattern
    wordlist.add(w[::-1])  # 'john' → 'nhoj'
```

What It Does:

Suffixes: Appends common passwords (123, !, 2024, etc.)
- `john + 123 = john123`
- `john + ! = john!`

Case Variations: Creates multiple cases
- `john`, `John`, `JOHN` (mimics user password patterns)

Leetspeak: Replaces letters with numbers
- `a → 4, e → 3, i → 1, o → 0, s → 5`
- `paris` → `p4r15`

Reversed: Flips strings
- `john` → `nhoj`

4.3 Permutations & Combinations

Code Snippet:

```
for n in range(2, min(4, len(base)+1)):
    # n=2 means pairs, n=3 means triplets
    for combo in itertools.permutations(base, n):
        # ('john', 'sparky') → all orderings
        joined = "".join(combo)  # 'johnsparky', 'sparkyjohn'
        wordlist.add(joined)
        wordlist.add(joined[::-1])  # reversed combo

        for suf in suffixes:
            wordlist.add(joined + suf)  # 'johnsparky123'
```

What It Does:

- Permutations: All possible orderings of 2-3 words
  - If base = ['john', 'sparky'], creates:
    - john + sparky = johnsparky
    - sparky + john = sparkyjohn
- Each combo also gets reversed and all suffixes applied
- Result: Hundreds of combinations from just 5 inputs!

Example Output (partial):

```
john, john123, john!, John, JOHN, j0hn, nhoj,
sparky, sparky123, sparky!, Sparky, SPARKY, sp4rky, yrkaps,
2007, 2007123, 2007!,
paris, paris123, paris!, Paris, PARIS, p4r15, sirap,
johnsparky, johnsparky123, sparkyjohn, sparkyjohn123, ...
[Total: 150+ combinations]
```

4.4 Display Preview

Code Snippet:

```
preview_text.config(state="normal")
preview_text.delete(1.0, tk.END)
```

```
for word in sorted(wordlist):  # Sort alphabetically
    preview_text.insert(tk.END, word + "\n")

preview_text.insert(tk.END, f"\nTotal combinations: {len(wordlist)}\n", "ok")
preview_text.config(state="disabled")
```

What It Does:

- Enables the read-only text box temporarily
- Clears previous preview
- Displays all generated words, sorted and line-by-line
- Shows total count (e.g., "Total combinations: 156")
- Locks text box again

4.5 Export Wordlist

Code Snippet:

```
def export_wordlist():
    preview_text.config(state="normal")
    contents = preview_text.get(1.0, tk.END).strip().split('\n')
    preview_text.config(state="disabled")

    if len(contents) < 2:
        messagebox.showerror("Error", "No wordlist generated yet!")
        return

    filepath = filedialog.asksaveasfilename(defaultextension=".txt", title="Save Wordlist")
    if filepath:
        with open(filepath, "w") as f:
            for word in contents:
                if word and not word.startswith('Total combinations'):
                    f.write(word + "\n")
        messagebox.showinfo("Saved", f"Wordlist saved to:\n{filepath}")
```

What It Does:

1. Extracts all text from preview box
2. Checks if wordlist exists (>2 lines)
3. Opens file save dialog
4. Writes each word to `.txt` file (one per line)
5. Filters out metadata like "Total combinations: 156"
6. Shows success message with file path

Output File Format (example):
```
john
john123
john!
John
JOHN
j0hn
nhoj
sparky
sparky123
...
```

Part 5: GUI Setup & Layout

Code Snippet:

```
root = tk.Tk()
root.title("PassDefender ToolKit")
root.config(bg=BG_MAIN)
root.geometry("900x550")  # Width x Height

# Password Analyzer Frame
pw_frame = tk.LabelFrame(root, text="Password Strength Analyzer",
                    font=FONT_LG, bg=BG_MAIN, fg=ACCENT_GRN, bd=2)
pw_frame.pack(fill="x", padx=30, pady=10)

# Password Entry with Show/Hide
tk.Label(pw_frame, text="Enter Password:", font=FONT_MAIN, bg=BG_MAIN,
fg=ACCENT_PURP).grid(row=0, column=0)
password_entry = tk.Entry(pw_frame, show="*", font=FONT_MAIN, width=38,
bg="#381847", fg=ACCENT_GRN)
password_entry.grid(row=0, column=1)

show_button = tk.Button(pw_frame, text="Show", command=toggle_password,
bg=ACCENT_BLUE, fg=BG_MAIN)
show_button.grid(row=0, column=2)
```

What It Does:

- Creates main window (900x550 pixels)
- Creates frame for password analyzer section
- Adds password input field with show/hide button
- Uses grid layout for precise positioning (row, column)
- Applies custom colors and fonts

Part 6: Common Words List

Code Snippet:

```
common_words = {"password", "qwerty", "letmein", "admin", "welcome",
                "iloveyou", "monkey", "dragon", "sunshine", "princess"}
```

What It Does:

- Stores frequently used weak passwords
- Used in issue detection: `if pwd.lower() in common_words`
- Helps identify if user entered a dictionary/common password
- Can be expanded with more known weak passwords

Part 7: Main Loop

Code Snippet:

```
root.mainloop()
```
```` ``` ````

What It Does:

- Starts the GUI event loop
- Keeps the window open and responsive
- Listens for user interactions (button clicks, text entry)
- Continues until user closes the window

Data Flow Diagram

USER INPUT
   ↓
[Password Entry] → analyze_password() → zxcvbn() → Score + Feedback
   ↓
[Custom Checks] → Identify Issues (length, common, chars)
   ↓
[Display Results] → Color-coded strength + issues + suggestions
   ↓
GUI OUTPUT (Real-time feedback)
````

```
[5 Input Fields] → generate_wordlist() → Base words
   ↓
[8+ Patterns] → Case, Leet, Suffix, Reverse, Permutations
   ↓
[Wordlist Set] → 100-200+ combinations
   ↓
[Preview Box] → Scrollable, sorted display
   ↓
[Export] → Save as .txt file
   ↓
FILE OUTPUT (Compatible with cracking tools)
```

Key Algorithms Explained

<u>Algorithm 1: Permutation Generation</u>

```
from itertools import permutations

base = ['john', 'sparky']
for combo in permutations(base, 2):
    # (john, sparky), (sparky, john)
    result = "".join(combo)  # johnsparky, sparkyjohn
```

Why?
Attackers try common name combinations. If they know your name and pet, they'll try both orderings.

<u>Algorithm 2: Leetspeak Transformation</u>

```
leet_map = {"a": "4", "e": "3", "i": "1", "o": "0", "s": "5"}
word = "paris"
leet_word = "".join(leet_map.get(c, c) for c in word)
# Result: "p4r15"
```

Why?
Many users think `P4ssw0rd` is strong, but it's predictable. Wordlist must include these common substitutions.

Algorithm 3: Set-Based Deduplication

```
wordlist = set()  # Automatically removes duplicates
wordlist.add("john")
wordlist.add("john")  # Ignored (already exists)
wordlist.add("john123")
# Result: {"john", "john123"} — no duplicates!
```

Why?
Without sets, we'd generate "john" multiple times through different patterns, wasting storage.

Security & Ethical Considerations

What It Does Well:

- Educational tool for learning password security
- Authorized penetration testing support
- Local processing (no data sent anywhere)
- Wordlists for ethical hacking scenarios

Important Notes:
- Use only for authorized testing
- Never use wordlists without permission
- Consider password strength in real-world scenarios
- zxcvbn provides realistic entropy analysis

Performance Notes

- Password Analysis: < 1 second (zxcvbn is highly optimized)
- Wordlist Generation: 0.5-2 seconds for 150+ combinations
- GUI Responsiveness: All operations are instant for user experience
- Memory: ~1-2 MB for typical wordlists

Conclusion

PassDefender ToolKit demonstrates:
1. Real-world password analysis using industry libraries
2. Algorithmic thinking with pattern generation & permutations
3. GUI design with responsive, themed interfaces
4. Practical cybersecurity concepts in an educational context

The code is modular, well-commented, and easily extensible for future features!

-Navya Nandika