Password Strength Analyzer with Custom Wordlist Generator — Detailed Step-by-Step Report

Objective

• Analyze password strength using Python (zxcvbn).

• Generate custom wordlists for testing password security using user-provided clues and transformations.

Prerequisites (Step 0)

• Windows PC (Windows 10/11 recommended).

• Internet connection to download Python and packages.

• Basic familiarity with Command Prompt (CMD) and editing files with a text editor (Notepad).

Step 1: Install Python

1. Go to https://www.python.org/downloads/ and download the latest Python 3 release.

2. Run the installer and IMPORTANTLY, check the box “Add Python to PATH” before clicking Install Now.

3. After installation, verify in CMD:

```
python --version
```

Expected output example: Python 3.13.7

Step 2: Install Required Python Libraries

Open Command Prompt (Win+R → cmd) and run the following commands one by one:

- pip install zxcvbn (for password strength analysis)

- pip install nltk (for text processing)
- pip install argparse
- pip install tk

Note: argparse and tkinter come bundled with standard Python distributions. On some Windows systems tkinter may require special installs; most installers include it.

Step 3: Create Project Folder

1. Create a folder, e.g., on Desktop: C:\Users\<you>\Desktop\PasswordAnalyzer

2. Save all project files in this folder: password_analyzer.py, wordlist_utils.py, gui_password_analyzer.py

Step 4: Create the Python Scripts (Detailed Code + Explanations)

Below are three files with full code and explanations. Copy each into its own .py file inside your project folder.

A) password_analyzer.py (CLI)

```
import argparse

from zxcvbn import zxcvbn

from wordlist_utils import generate_custom_wordlist

def analyze_password(pw):

    """Analyze the strength of the given password."""

    res = zxcvbn(pw)

    print(f"\nPassword: {pw}")

    print(f"Score (0-4): {res['score']}")

    print(f"Estimated crack time: {res['crack_times_display'].get('offline_slow_hashing_1e4_per_second', 'N/A')}")

    warning = res.get('feedback', {}).get('warning', '')

    suggestions = res.get('feedback', {}).get('suggestions', [])

    if warning:
```

```
        print(" ⚠️ Warning:", warning)
    if suggestions:
        print(" 💡 Suggestions:", ', '.join(suggestions))
def save_wordlist(words, outfile):
    """Save generated custom wordlist to a file."""
    with open(outfile, 'w', encoding='utf-8') as f:
        for w in words:
            f.write(w + '\n')
    print(f"\n ✅ Wordlist saved to '{outfile}' (total {len(words)} entries)")


if __name__ == "__main__":
    # ---------------- Argument Setup ----------------
    parser = argparse.ArgumentParser(
        description="Password Strength Analyzer & Custom Wordlist Generator (clean output)"
    )
    parser.add_argument(
        '--password', '-p',
        required=True,
        help="Password to analyze (example: MyPass123)"
    )
    parser.add_argument(
        '--clues', '-c',
        nargs='+',
        required=False,
        help="Custom clues (example: name pet year). If a clue contains spaces, wrap it in quotes."
    )
    parser.add_argument(
        '--out', '-o',
        default='custom_wordlist.txt',
```

```
    help="Output wordlist filename (default: custom_wordlist.txt)"
)
parser.add_argument(
    '--max',
    type=int,
    default=500,
    help="Maximum number of words to generate (default: 500)"
)
args = parser.parse_args()
# Ensure clues is always a list
clues = args.clues if args.clues else []
# ---------------- Run Analyzer ----------------
analyze_password(args.password)
# Generate wordlist with leet and year variants disabled for cleaner output
raw_words = generate_custom_wordlist(clues, include_leet=True, include_years=True, max_items=args.max)
# Deduplicate and sort for neat, alphabetical output
clean_words = sorted(set(raw_words))
save_wordlist(clean_words, args.out)
```

Let's break it down **section by section** so you understand exactly how it works 👇

🧩 1. Imports

```
import argparse
from zxcvbn import zxcvbn
from wordlist_utils import generate_custom_wordlist
```

Explanation:

- **argparse** → handles command-line arguments (like --password, --clues, etc.).
- **zxcvbn** → analyzes password strength (developed by Dropbox).
- **generate_custom_wordlist** → imported from another file wordlist_utils.py, which generates variations of given clues (e.g., names, pets, dates) for wordlist creation.

🧠 2. analyze_password(pw) Function

```
def analyze_password(pw):
    """Analyze the strength of the given password."""
    res = zxcvbn(pw)
    print(f"\nPassword: {pw}")
    print(f"Score (0-4): {res['score']}")
    print(f"Estimated crack time: {res['crack_times_display'].get('offline_slow_hashing_1e4_per_second', 'N/A')}")
    warning = res.get('feedback', {}).get('warning', '')
    suggestions = res.get('feedback', {}).get('suggestions', [])
    if warning:
        print("⚠️ Warning:", warning)
    if suggestions:
        print("💡 Suggestions:", ', '.join(suggestions))
```

Purpose:

Analyzes how strong or weak a given password is.

Line-by-line:

- res = zxcvbn(pw) → runs the password through the zxcvbn algorithm and returns a **dictionary** with strength info.
- res['score'] → gives a score **from 0 to 4**:
  - 0 → Very Weak
  - 1 → Weak
  - 2 → Fair
  - 3 → Good
  - 4 → Strong
- res['crack_times_display']['offline_slow_hashing_1e4_per_second'] → estimates how long a hacker might take to crack it offline.

- res['feedback'] → gives **warnings** and **suggestions** (e.g., “Avoid common words”).
- The code neatly prints:
    - Password being analyzed
    - Strength score
    - Estimated crack time
    - Optional warnings or suggestions (if any)

🗄️ 3. save_wordlist(words, outfile) Function

```
def save_wordlist(words, outfile):
    """Save generated custom wordlist to a file."""
    with open(outfile, 'w', encoding='utf-8') as f:
        for w in words:
            f.write(w + '\n')
    print(f"\n✅ Wordlist saved to '{outfile}' (total {len(words)} entries)")
```

Purpose:

Saves the generated custom wordlist (from clues) into a .txt file.

Steps:

1. Opens the file in write mode with UTF-8 encoding.
2. Writes each generated word (variation) to a new line.
3. Prints confirmation — showing filename and total entries.

⚙️ 4. Argument Parser (argparse)

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="Password Strength Analyzer & Custom Wordlist Generator (clean output)"
    )
```

Creates a CLI (Command Line Interface) for the tool.

Arguments Defined:

Argument	Short	Description	Required	Default
--password	-p	Password to analyze	✅ Yes	–
--clues	-c	List of clues (like name, pet, year)	❌ Optional	None
--out	-o	Output file name for wordlist	❌ Optional	custom_wordlist.txt
--max	–	Max number of words to generate	❌ Optional	500

Example command:

python password_analyzer.py --password MyPass123 --clues Adarsh Dog 2001 --out mylist.txt

🔁 5. Clue Handling

clues = args.clues if args.clues else []

If the user doesn't provide any --clues, it ensures clues becomes an empty list instead of None (prevents errors).

🚀 6. Running the Analyzer

analyze_password(args.password)

Runs the password strength check using the zxcvbn library.

🧩 7. Generate and Save Wordlist

raw_words = generate_custom_wordlist(clues, include_leet=True, include_years=True, max_items=args.max)

clean_words = sorted(set(raw_words))

save_wordlist(clean_words, args.out)

Steps:

1. **generate_custom_wordlist(...)** — creates variations of the given clues:
    - Adds lowercase, uppercase, reversed, leetspeak (a→4, s→$, etc.), and year combinations (like Dog2001).
2. **set()** — removes duplicates.
3. **sorted()** — sorts the words alphabetically.
4. **save_wordlist()** — writes them to the .txt file.

🧪 Example Run

python password_analyzer.py --password MyPass123 --clues Adarsh Dog 2001 --out mywordlist.txt --max 100

Output Example:

Password: MyPass123

Score (0-4): 3

Estimated crack time: 3 hours

💡 Suggestions: Add another word or symbol for more strength

✅ Wordlist saved to 'mywordlist.txt' (total 98 entries)

🧠 Summary

Feature	Description
Purpose	Tests password strength & generates a personalized wordlist.
Input	Password + Optional clues (like name, pet, birth year).
Output	Password analysis result + Wordlist file.
Libraries Used	argparse, zxcvbn, wordlist_utils.
Extras	Handles missing clues gracefully, limits max words, prints clean messages.

B) wordlist_utils.py (Wordlist Generator)

```
import itertools

import datetime

# Leetspeak substitutions

LEET_MAP = {

    'a':['a','4'], 'b':['b','8'], 'e':['e','3'], 'i':['i','1','!'],

    'l':['l','1','|'], 'o':['o','0'], 's':['s','$','5'], 't':['t','7']

}

def leetspeak_variants(s, max_variants=500):

    s = s.lower()

    pools = [LEET_MAP.get(ch,[ch]) for ch in s]

    variants = set()

    for tup in itertools.islice(itertools.product(*pools), max_variants):

        variants.add(''.join(tup))

    return list(variants)


def append_years(base_list, max_append=10):

    years = [str(y) for y in range(1990, datetime.datetime.now().year+1)]

    out = set(base_list)

    for b in base_list:

        for y in years[-max_append:]:

            out.add(b+y)

            out.add(y+b)

    return list(out)

def combine_tokens(tokens, max_len=3):

    tokens = list(dict.fromkeys([t for t in tokens if t]))
```

```
    out = set()
    for r in range(1, min(max_len, len(tokens))+1):
        for combo in itertools.permutations(tokens, r):
            for sep in ['', '.', '_', '-']:
                out.add(sep.join(combo))
    return list(out)


def generate_custom_wordlist(clues, include_leet=True, include_years=True, max_items=5000):
    base = []
    for c in clues:
        c = c.strip()
        if not c: continue
        base += [c, c.lower(), c.capitalize()]
    variants = set(base)
    if include_leet:
        for b in base:
            for v in leetspeak_variants(b, max_variants=30):
                variants.add(v)
    combined = combine_tokens(list(variants), max_len=3)
    variants.update(combined)
    final = set(variants)
    if include_years:
        final.update(append_years(list(final), max_append=10))
    # Add common passwords
    COMMON = ['password', '123456', 'welcome', 'admin', 'qwerty']
    for c in COMMON:
```

```
        final.add(c)
        final.add(c+'123')
        final.add('123'+c)
    return list(final)[:max_items]
```

🧩 wordlist_utils.py — Explanation

This file creates **custom password wordlists** using user clues (like name, pet, birth year, etc.) with variations like leetspeak, years, and combinations.

⚙️ 1. Imports and Leet Map

```
import itertools
import datetime
LEET_MAP = {
    'a':['a','4'], 'b':['b','8'], 'e':['e','3'], 'i':['i','1','!'],
    'l':['l','1','|'], 'o':['o','0'], 's':['s','$','5'], 't':['t','7']
}
```

📘 **Purpose:**

- itertools → used for permutations & combinations
- datetime → used to get current year
- LEET_MAP → defines how letters are replaced in *leet (1337) speak*

🧠 Example:
password → p4ssw0rd, leet → l33t

🔡 2. Leetspeak Variants

```
def leetspeak_variants(s, max_variants=500):
    s = s.lower()
    pools = [LEET_MAP.get(ch,[ch]) for ch in s]
```

```
    variants = set()
    for tup in itertools.islice(itertools.product(*pools), max_variants):
        variants.add(''.join(tup))
    return list(variants)
```

📘 **Purpose:**
Creates alternative spellings using **leet substitutions**.

🔁 **Process:**

s = "dog"

↓

['d'], ['o', '0'], ['g']

↓

dog, d0g

⚙️ Uses:

- itertools.product → generates all combinations
- max_variants → limits number to avoid overload

🗓️ **3. Append Years**

```
def append_years(base_list, max_append=10):
    years = [str(y) for y in range(1990, datetime.datetime.now().year+1)]
    out = set(base_list)
    for b in base_list:
        for y in years[-max_append:]:
            out.add(b+y)
            out.add(y+b)
    return list(out)
```

📘 **Purpose:**
Adds recent years to each clue for realism (like passwords with years).

🧠 Example:

dog → dog2022, 2022dog

🗓️ Adds last **10 years** (e.g., 2016–2025).

🔀 4. Combine Tokens

```
def combine_tokens(tokens, max_len=3):
    tokens = list(dict.fromkeys([t for t in tokens if t]))
    out = set()
    for r in range(1, min(max_len, len(tokens))+1):
        for combo in itertools.permutations(tokens, r):
            for sep in ['', '.', '_', '-']:
                out.add(sep.join(combo))
    return list(out)
```

📘 **Purpose:**
Joins different clues together using separators.

🧩 **Example:**

["adarsh", "dog"] → adarshdog, adarsh_dog, dog-adarsh

⚙️ itertools.permutations() → changes order
🔠 Separators → "", ".", "_", "-"

🧠 5. Generate Custom Wordlist

```
def generate_custom_wordlist(clues, include_leet=True, include_years=True, max_items=5000):
    base = []
    for c in clues:
        c = c.strip()
        if not c: continue
```

```
        base += [c, c.lower(), c.capitalize()]
    variants = set(base)
    if include_leet:
        for b in base:
            for v in leetspeak_variants(b, max_variants=30):
                variants.add(v)
    combined = combine_tokens(list(variants), max_len=3)
    variants.update(combined)
    final = set(variants)
    if include_years:
        final.update(append_years(list(final), max_append=10))
    COMMON = ['password', '123456', 'welcome', 'admin', 'qwerty']
    for c in COMMON:
        final.add(c)
        final.add(c+'123')
        final.add('123'+c)
    return list(final)[:max_items]
```

📘 **Purpose:**
The main function — combines all steps to generate a large **custom wordlist**.

🔁 **Steps Inside:**

1. Take clues (user words like “adarsh”, “dog”).
2. Add case variations → adarsh, Adarsh
3. Add leetspeak versions → 4d4rsh, d0g
4. Combine tokens → AdarshDog, Dog_Adarsh
5. Append recent years → Adarsh2025
6. Add common passwords → password123, qwerty
7. Limit to max_items results.

📤 **Output:**
A final list of up to 5000 unique password candidates.

📜 **Example Flow**

🧩 Clues: ["Adarsh", "Dog"]

➡ Step-by-step generation:

Base: adarsh, dog

↓

Leetspeak: 4d4rsh, d0g

↓

Combinations: adarshdog, dog-adarsh

↓

Years: adarsh2025, 2025dog

↓

Common: password, qwerty, admin123

📂 **Final Wordlist Sample:**

adarsh

dog

4d4rsh

d0g

adarshdog

dog-adarsh

adarsh2025

password

qwerty123

✅ **Summary**

Step	Function	Purpose
1	leetspeak_variants()	Creates 1337-style versions
2	append_years()	Adds years to words
3	combine_tokens()	Combines multiple clues
4	generate_custom_wordlist()	Brings all together + adds common passwords

C) gui_password_analyzer.py (Tkinter GUI)

```
import tkinter as tk

from tkinter import messagebox, filedialog

from zxcvbn import zxcvbn

from wordlist_utils import generate_custom_wordlist  # import your wordlist generator


# ----------------- Password Analysis -----------------

def analyze_password(pw):

    res = zxcvbn(pw)

    score = res.get("score")

    crack = res.get("crack_times_display", {}).get("offline_slow_hashing_1e4_per_second",
"N/A")

    warning = res.get("feedback", {}).get("warning", "")

    suggestions = res.get("feedback", {}).get("suggestions", [])

    summary_lines = [

        f"Score (0-4): {score}",

        f"Estimated crack time: {crack}"

    ]
```

```
    if warning:
        summary_lines.append(f"Warning: {warning}")
    if suggestions:
        summary_lines.append("Suggestions:")
        for s in suggestions:
            summary_lines.append(" - " + s)
    return "\n".join(summary_lines)

# ----------------- Run Analyzer -----------------
def run_analyzer():
    pwd = entry_password.get()
    clues_text = entry_inputs.get().strip()
    clues = [c.strip() for c in clues_text.split(',')] if clues_text else []
if not pwd:
        messagebox.showinfo("Input required", "Please enter a password.")
        return
    # Analyze password
    result_text = analyze_password(pwd)

    # Ask where to save
    outpath = filedialog.asksaveasfilename(
        defaultextension=".txt",
        filetypes=[("Text files","*.txt")],
        initialfile="custom_wordlist.txt"
    )
    if not outpath:
```

```
        return
    MAX_WORDS = 500
    words = generate_custom_wordlist(clues)
    written = 0
    with open(outpath, "w", encoding="utf-8") as f:
        if isinstance(words, (list, tuple)):
            to_write = words[:MAX_WORDS]
            for w in to_write:
                f.write(w + "\n")
            written = len(to_write)
        else:
            for w in words:
                if written >= MAX_WORDS:
                    break
                f.write(w + "\n")
                written += 1
    note = ""
    if written >= MAX_WORDS:
        note = f"\n\nNote: output limited to the first {MAX_WORDS} entries."
    messagebox.showinfo(
        "Analysis Complete",
        f"{result_text}\n\nWordlist saved to:\n{outpath}\nEntries written: {written}{note}"
    )
# ----------------- Build GUI -----------------
root = tk.Tk()
root.title("Password Analyzer & Wordlist Generator")
```

```
# Password label & entry
tk.Label(root, text="Password:").pack(pady=(10,0))
entry_password = tk.Entry(root, show="*")
entry_password.pack(pady=(0,10))

# Show/hide password checkbox
show_var = tk.BooleanVar()
def toggle_password():
    if show_var.get():
        entry_password.config(show="")
    else:
        entry_password.config(show="*")

tk.Checkbutton(root, text="Show password", variable=show_var,
command=toggle_password).pack()

# Clues label & entry
tk.Label(root, text="Custom inputs (comma separated):").pack()
entry_inputs = tk.Entry(root)
entry_inputs.pack(pady=(0,10))

# Run button
tk.Button(root, text="Run Analyzer", command=run_analyzer).pack(pady=(5,10))
# Start GUI loop
root.mainloop()
```

💻 GUI Password Analyzer & Wordlist Generator — Explanation

📐 1️⃣ Imports & Setup

```
import tkinter as tk

from tkinter import messagebox, filedialog

from zxcvbn import zxcvbn

from wordlist_utils import generate_custom_wordlist
```

🧩 **Purpose:**

- tkinter → Builds the GUI window
- messagebox → Displays pop-up alerts
- filedialog → Lets users choose where to save files
- zxcvbn → Evaluates password strength
- generate_custom_wordlist() → Generates custom word variations

📦 **Diagram:**

```
+-------------------------------+

| User Interface (Tkinter)     |

|-------------------------------|

| zxcvbn → Password Strength   |

| wordlist_utils → Wordlist    |

+-------------------------------+
```

🧠 2️⃣ Password Analysis Function

```
def analyze_password(pw):

    res = zxcvbn(pw)

    score = res.get("score")
```

```
    crack = res.get("crack_times_display", {}).get("offline_slow_hashing_1e4_per_second", "N/A")
```

🔍 **Purpose:**
Analyzes how strong or weak a password is using zxcvbn.

📊 **It displays:**

- **Score (0–4)** — 0 = Very Weak, 4 = Very Strong
- **Estimated crack time** — Approximate time to guess password
- **Warnings & suggestions** — Helpful tips to strengthen passwords

📈 **Example Output:**

Score (0-4): 3

Estimated crack time: 2 days

Suggestions:

- Use uncommon words

- Add more characters

⚙️ 3️⃣ **Main Logic — run_analyzer()**

```
def run_analyzer():

    pwd = entry_password.get()

    clues_text = entry_inputs.get().strip()
```

🧩 **Purpose:** Runs when the **"Run Analyzer"** button is clicked.

📋 **Steps:**

Step	Action	Functionality
1️⃣	Get password & clues	Takes user input from GUI
2️⃣	Analyze password	Calls analyze_password()

Step	Action	Functionality
3️⃣	Save file	Opens “Save As” dialog to choose .txt path
4️⃣	Generate wordlist	Uses generate_custom_wordlist() from clues
5️⃣	Write to file	Saves up to **500** generated words
6️⃣	Show popup	Displays final analysis and file details

📁 **Example message:**

✅ Password strength analyzed!

Wordlist saved to: C:\Users\Adarsh\custom_wordlist.txt

Entries written: 500

🪟 4️⃣ GUI Design

```
root = tk.Tk()
root.title("Password Analyzer & Wordlist Generator")
```

🎨 **Elements in the window:**

GUI Element	Purpose
Label	Displays text like “Password:”
Entry	Lets user type password or clues
Checkbutton	Toggles “Show Password” visibility
Button	Runs the analysis

🖼️ **Layout (Sketch):**

```
+-----------------------------------------+
| 🔐 Password: [************] [✓] Show     |
| 🧩 Custom Inputs: [adarsh, dog, 2001]    |
|                                          |
| [ Run Analyzer ]                         |
+-----------------------------------------+
| 💬 Popup: "Analysis Complete!"           |
+-----------------------------------------+
```

👁️ 5️⃣ Show / Hide Password Function

```
def toggle_password():
    if show_var.get():
        entry_password.config(show="")
    else:
        entry_password.config(show="*")
```

👀 **Feature:**
Lets users toggle password visibility.

- Checked → shows password
- Unchecked → hides password (shows *)

🪟 6️⃣ Running the GUI

```
root.mainloop()
```

🌀 Keeps the GUI **open and active** until the user closes it.

📜 7️⃣ Program Flow Summary

Step	User Action	What Happens
1️⃣	Enter password	Captured by entry_password
2️⃣	Add clues	Captured by entry_inputs
3️⃣	Click Run	run_analyzer() executes
4️⃣	Analyze password	analyze_password() returns strength
5️⃣	Generate list	generate_custom_wordlist() creates variations
6️⃣	Save and display	File saved + popup summary

✅ Example Popup Result

Score (0-4): 4

Estimated crack time: 5 months

Suggestions:

- Avoid common patterns.

Wordlist saved to:

C:\Users\Adarsh\Documents\custom_wordlist.txt

Entries written: 500

Step 5: How to Run (Examples & Expected Output)

Open Command Prompt and navigate to the project folder, e.g.:

- cd Desktop\PasswordAnalyzer
- CLI examples:

1) With clues:

- python password_analyzer.py --password MyPass123 --clues Adarsh Dog 2001 --out mywordlist.txt

2) Without clues (clues optional):

- python password_analyzer.py --password MyPass123 --out mywordlist.txt

- GUI: Run gui_password_analyzer.py, enter password and comma-separated clues, click Run, choose save location.

Sample Password Checks & Results

Password	Score	Crack time example	Notes
password123	1	seconds	Very weak/common
12345678	1	seconds	Very weak/common
summer2023	2	minutes	Weak: word+year
hello1234	2	minutes	Weak: common pattern
Sunny!Day2023	3	hours	Good length, symbol
BlueSky#89	3	hours	Good mix
Schein@4567FR	4	months	Strong: varied chars
vG9$kT2!pLmQ	4	years	Very strong: random-like

Step 6: Troubleshooting & Common Issues

1) TypeError: 'NoneType' object is not iterable

- Cause: args.clues was None. Fix: ensure you convert to list after parsing:
- clues = args.clues if args.clues else []

2) GUI shows **** while typing password

- Cause: Tkinter Entry uses show='*' to mask input. Fix: either remove show or add toggle checkbox (code provided).

3) zxcvbn not installed or import error

- Fix: pip install zxcvbn-python and ensure you're using the same Python interpreter as pip.

4) Very large wordlist causing memory issues

- Fix: use generator-based production or pass max_items to limit output; in GUI we limit to 500 by default.