



Password Strength Analyzer with Entropy Calculation

Step 1: Set Up Python Environment

1. Ensure Python3 and Pip are Installed

- Kali Linux typically comes with Python3 pre-installed, but let's verify it:

```
python3 --version  
pip3 --version
```

- If `pip3` is not installed, install it using:

```
sudo apt update
```

```
sudo apt install python3-pip
```

2. Install Required Libraries

- For this project, we'll use `math` and `re` (regular expressions), which are built-in Python libraries, so you don't need to install them separately.
- However, if you plan to expand your project with advanced statistical functions, install `numpy` as well:

```
pip3 install numpy
```

```
(root@kali)-[/home/theconsoler]
# sudo apt update
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [20.3 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [49.4 MB]
Get:4 http://kali.download/kali kali-rolling/contrib amd64 Packages [112 kB]
Get:5 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [274 kB]
Get:6 http://kali.download/kali kali-rolling/non-free amd64 Packages [197 kB]
Get:7 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [876 kB]
Get:8 http://kali.download/kali kali-rolling/non-free-firmware amd64 Packages [10.6 kB]
Get:9 http://kali.download/kali kali-rolling/non-free-firmware amd64 Contents (deb) [23.1 kB]
Fetched 71.2 MB in 42s (1699 kB/s)
2369 packages can be upgraded. Run 'apt list --upgradable' to see them.

(root@kali)-[/home/theconsoler]
# sudo apt install python3-pip
Upgrading:
  libjs-sphinxdoc  python3-pip  python3-pip-whl

Summary:
  Upgrading: 3, Installing: 0, Removing: 0, Not Upgrading: 2366
  Download size: 3100 kB
  Freed space: 45.1 kB

Continue? [Y/n] y
Get:1 http://kali.download/kali kali-rolling/main amd64 libjs-sphinxdoc all 7.4.7-4 [158 kB]
Get:2 http://http.kali.org/kali kali-rolling/main amd64 python3-pip all 24.3.1+dfsg-1 [1441 kB]
Get:3 http://http.kali.org/kali kali-rolling/main amd64 python3-pip-whl all 24.3.1+dfsg-1 [1501 kB]
Fetched 3100 kB in 4s (748 kB/s)
(Reading database ... 504954 files and directories currently installed.)
Preparing to unpack .../libjs-sphinxdoc_7.4.7-4_all.deb ...
Unpacking libjs-sphinxdoc (7.4.7-4) over (7.3.7-3) ...
Preparing to unpack .../python3-pip_24.3.1+dfsg-1_all.deb ...
Unpacking python3-pip (24.3.1+dfsg-1) over (24.1.1+dfsg-1) ...
Preparing to unpack .../python3-pip-whl_24.3.1+dfsg-1_all.deb ...
Unpacking python3-pip-whl (24.3.1+dfsg-1) over (24.1.1+dfsg-1) ...
Setting up python3-pip-whl (24.3.1+dfsg-1) ...
Setting up python3-pip (24.3.1+dfsg-1) ...
Setting up libjs-sphinxdoc (7.4.7-4) ...
```

```
(root@kali)-[/home/theconsoler]
# pip install numpy
Requirement already satisfied: numpy in /usr/lib/python3/dist-packages (1.26.4)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting be
/pip.pypa.io/warnings/venv. Use the --root-user-action option if you know what you are doin

(root@kali)-[/home/theconsoler]
# nano password_analyzer.py
```

NUMPY IS ALREADY INSTALLED.

Step 2: Create the Python Script

1. Create a New Python File

- Use a text editor like `nano` to create a file called `password_analyzer.py` in your project folder:

```
nano password_analyzer.py
```

2. Structure Your Script

In the file, we'll add different functions to analyze the password's strength by calculating entropy, detecting patterns, checking against common words, and generating recommendations.

Step 3: Write the Entropy Calculation Function

1. Add Imports at the Top of Your Script

```
import math
import re
```

2. Define the Entropy Calculation Function

- Entropy is a measure of randomness and helps estimate the strength of a password. We'll calculate entropy based on the character set and password length.

```
def calculate_entropy(password):
    charset_size = 0

    # Determine charset size based on characters used in the password
    if re.search(r'[a-z]', password):
        charset_size += 26 # Lowercase letters
    if re.search(r'[A-Z]', password):
        charset_size += 26 # Uppercase letters
    if re.search(r'[0-9]', password):
        charset_size += 10 # Digits
    if re.search(r'^a-zA-Z0-9', password):
        charset_size += 32 # Special characters

    # Entropy calculation
    entropy = math.log2(charset_size ** len(password)) if charset_size else 0
    return entropy
```

3. Save and Test the Entropy Function

- Save the file with `Ctrl + O`, press `Enter`, then exit with `Ctrl + X`.
- Test it by running Python interactively:

```
python3
```

- Import and test the function:

```
from password_analyzer import calculate_entropy
print(calculate_entropy("TestPassword123!")) # Replace
```

```
with any sample password
```

- You should see an entropy value.

Step 4: Write Pattern Detection Function

In the `password_analyzer.py` file, add a function to check for common patterns like repeated or sequential characters.

```
def check_patterns(password):
    patterns = []

    if re.search(r'(\1)\1\1', password): # Check for three co
nsecutive identical characters
        patterns.append("Repeated characters")

    if re.search(r'(012|123|234|345|456|567|678|789)', passwo
rd): # Simple sequence
        patterns.append("Sequential numbers")

    if re.search(r'(abc|bcd|cde|def)', password, re.IGNORECAS
E): # Simple alpha sequence
        patterns.append("Sequential letters")

    return patterns
```

Step 5: Create Dictionary Check Function

We'll add a function that flags dictionary words or commonly used passwords. You can create a set of common passwords in the script or use a wordlist file.

```
def dictionary_check(password, common_passwords):
```

```
lower_password = password.lower()
if lower_password in common_passwords:
    return "Password is too common"
return None
```

Step 6: Add Recommendations Function

This function will provide feedback based on entropy, patterns, and dictionary checks.

```
def generate_recommendations(entropy, patterns, dictionary_flag):
    recommendations = []

    if entropy < 40:
        recommendations.append("Increase the password length
or add more character types.")

    if patterns:
        recommendations.append("Avoid common patterns: " + ",
".join(patterns))

    if dictionary_flag:
        recommendations.append("Avoid common or dictionary words.")

    if not recommendations:
        recommendations.append("Your password is strong!")

    return recommendations
```

Step 7: Combine All Functions in a Main Analysis Function

Now we'll write the main function that ties everything together and outputs the results.

```
def analyze_password(password):
    common_passwords = {"password", "123456", "qwerty", "admin", "letmein"} # Add more as needed

    # Entropy calculation
    entropy = calculate_entropy(password)
    print(f"Entropy: {entropy:.2f} bits")

    # Pattern checks
    patterns = check_patterns(password)
    print(f"Detected Patterns: {patterns if patterns else 'None'}")

    # Dictionary check
    dictionary_flag = dictionary_check(password, common_passwords)
    if dictionary_flag:
        print(dictionary_flag)

    # Recommendations
    recommendations = generate_recommendations(entropy, patterns, dictionary_flag)
    print("Recommendations:")
    for rec in recommendations:
        print(f" - {rec}")
```

Step 8: Finalize and Test the Password Analyzer

1. Add the Main Execution Code

At the bottom of `password_analyzer.py`, add code to take input from the user and call the `analyze_password` function.

```
if __name__ == "__main__":  
    password = input("Enter a password to analyze: ")  
    analyze_password(password)
```

2. Run the Script

- Save and exit `password_analyzer.py` (using `Ctrl + O`, `Enter`, and `Ctrl + X`).
- Run the script from the terminal:

```
python3 password_analyzer.py
```

- Enter various passwords to see the output. The script should display entropy, detected patterns, and recommendations.

Step 9: Expand and Improve (Optional)

Here are a few ideas for enhancing the project:

- **Use External Dictionary:** Use a wordlist like `rockyou.txt` located at `/usr/share/wordlists/rockyou.txt` for a more comprehensive dictionary check.

```
with open("/usr/share/wordlists/rockyou.txt") as f:  
    common_passwords = set(word.strip().lower() for word i  
n f)
```

- **Estimate Time to Crack:** Based on entropy, estimate the time to crack the password using various attack methods.
- **Build a GUI:** If you're interested, use Tkinter to create a graphical interface.


```
(root@kali)-[/home/theconsoler]
# nano password_analyzer.py

(root@kali)-[/home/theconsoler]
# python3 password_analyzer.py

Enter a password to analyze: Brother@004
Entropy: 72.10 bits
Detected Patterns: None
Recommendations:
- Your password is strong!
```

PASSWORD - Brother@004

RESULT - Your Password is strong !

BUILDING GUI FOR PASSWORD ANALYZER WITH TKINTER

Building a GUI with Tkinter for your Password Strength Analyzer is a great idea! This will make the tool more user-friendly and accessible. I'll guide you step-by-step, following a similar process to what we did before. Tkinter comes pre-installed with Python, so you won't need additional installations for this part.

Step 1: Install Tkinter (if needed)

On Kali Linux, Tkinter may already be installed. If it's not, install it using the following command:

```
sudo apt update
sudo apt install python3-tk
```

Verify that it's installed by running:

```
python3 -m tkinter
```

If the Tkinter GUI window opens, you're good to go.

```
(root@kali) ~/home/theconsole
# sudo apt install python3-tk
The following packages were automatically installed and are no longer required:
libverbs-providers libcephfs2 libgfrpc0 libglusterfs0 libpython3.11-dev librdmacm164 python3.11 python3.11-minimal
libboost-testram1.83.0 libgfaio0 libgfrdr0 liblibverbs1 librados2 python3-lib2to3 python3.11-dev samba-vfs-modules
Use 'sudo apt autoremove' to remove them.

Upgrading:
blueman libpython3-dev libpython3.12t64 libtdb1 onboard-data python3-dev python3-minimal python3-setuptools python3.12 samba-common-bin tdb-tools
libpgm1.4t64 libpython3-stdlib libsmclient0 libwbclient0 openssl python3-gpg python3-nassl python3-talloc python3.12-minimal samba-libs winexe
libpgm1.4t64 libpython3.12-minimal libssl3t64 onboard python3 python3-gpg python3-ldb python3-pkg-resources python3-tdb samba samba-vfs-modules winexe
libldb2 libpython3.12-stdlib libtalloc2 onboard-common python3-arc4 python3-lib2to3 python3-samba python3-tk samba-common smbcclient

Installing dependencies:
libassuan9 libnss-winbind libpam-winbind libpython3.12-dev openssl-provider-legacy python3.12-dev python3.12-tk python3.13-tk samba-ad-dc samba-ad-provision samba-dsdb-modules winbind

Suggested packages:
tix bind9 bind9utils ldb-tools ntp | chrony

REMOVING:
python3-distutils

Summary:
Upgrading: 42, Installing: 12, Removing: 1, Not Upgrading: 2324
Download size: 332 kB / 41.2 MB
Space needed: 54.6 MB / 6310 MB available
```

Step 2: Update the `password_analyzer.py` Script for GUI Integration

Let's modify your existing script to add Tkinter functionality. This will involve creating a GUI layout with input fields, buttons, and areas to display results.

1. Open Your Python Script

Open the `password_analyzer.py` file in a text editor:

```
nano password_analyzer.py
```

2. Import Tkinter and Update Imports

At the top of your script, add the `tkinter` import and rename any previous imports if necessary:

```
import math
import re
import tkinter as tk
```

```
from tkinter import messagebox
```

Step 3: Design the GUI Layout

Here, we'll define a basic layout that includes:

- An entry field for the password
- A button to analyze the password
- A text area to display the entropy, patterns, and recommendations

Add the following code after the import statements:

```
def create_gui():
    # Create the main window
    window = tk.Tk()
    window.title("Password Strength Analyzer")
    window.geometry("400x400") # Width x Height

    # Label for Password Entry
    password_label = tk.Label(window, text="Enter Password:")
    password_label.pack(pady=10)

    # Password Entry Field
    password_entry = tk.Entry(window, show="*", width=30)
    password_entry.pack(pady=5)

    # Result Area
    result_text = tk.Text(window, height=15, width=45, wrap="word", state="disabled")
    result_text.pack(pady=10)

    # Analyze Button
    analyze_button = tk.Button(window, text="Analyze Password", command=lambda: analyze_password_gui(password_entry, result_text))
```

```
lt_text))
    analyze_button.pack(pady=10)

    # Run the GUI main loop
    window.mainloop()
```

Step 4: Modify `analyze_password` Function for GUI Output

Now, let's adapt the `analyze_password` function to output results to the `result_text` widget in the GUI.

Add this new function below `create_gui`:

```
def analyze_password_gui(password_entry, result_text):
    # Clear previous results
    result_text.config(state="normal")
    result_text.delete(1.0, tk.END)

    # Get password from entry field
    password = password_entry.get()

    # Sample set of common passwords (expand as needed)
    common_passwords = {"password", "123456", "qwerty", "admin", "letmein"}

    # Calculate Entropy
    entropy = calculate_entropy(password)
    result_text.insert(tk.END, f"Entropy: {entropy:.2f} bits\n\n")

    # Pattern Check
    patterns = check_patterns(password)
    result_text.insert(tk.END, f"Detected Patterns: {patterns if patterns else 'None'}\n\n")
```

```

    # Dictionary Check
    dictionary_flag = dictionary_check(password, common_passwords)
    if dictionary_flag:
        result_text.insert(tk.END, f"{dictionary_flag}\n\n")

    # Recommendations
    recommendations = generate_recommendations(entropy, patterns, dictionary_flag)
    result_text.insert(tk.END, "Recommendations:\n")
    for rec in recommendations:
        result_text.insert(tk.END, f" - {rec}\n")

    # Disable editing in result text box
    result_text.config(state="disabled")

```

Step 5: Connect GUI Functions to Main Functionality

At the bottom of your script, replace the command-line interface section with a call to `create_gui()`. This will open the GUI when the script is run.

```

if __name__ == "__main__":
    create_gui()

```

```

GNU nano 8.1
import math
import re
import tkinter as tk
from tkinter import messagebox

def calculate_entropy(password):
    charset_size = 0

    # Determine charset size based on characters used in the password
    if re.search(r'[a-z]', password):
        charset_size += 26 # Lowercase letters
    if re.search(r'[A-Z]', password):
        charset_size += 26 # Uppercase letters
    if re.search(r'[0-9]', password):
        charset_size += 10 # Digits
    if re.search(r'^a-zA-Z0-9', password):
        charset_size += 32 # Special characters

    # Entropy calculation
    entropy = math.log2(charset_size ** len(password)) if charset_size else 0
    return entropy

def check_patterns(password):
    patterns = []
    if re.search(r'(\.)\1\1', password): # Check for three consecutive identical characters
        patterns.append("Repeated characters")
    if re.search(r'(012|123|234|345|456|567|678|789)', password): # Simple sequence
        patterns.append("Sequential numbers")
    if re.search(r'(abc|bcd|cde|def)', password, re.IGNORECASE): # Simple alpha sequence
        patterns.append("Sequential letters")
    return patterns

def dictionary_check(password, common_passwords):
    lower_password = password.lower()
    if lower_password in common_passwords:
        return "Password is too common"
    return None

def generate_recommendations(entropy, patterns, dictionary_flag):
    recommendations = []
    if entropy < 40:
        recommendations.append("Increase the password length or add more character types.")
    if patterns:
        recommendations.append("Avoid common patterns: " + ", ".join(patterns))
    if dictionary_flag:
        recommendations.append("Avoid common or dictionary words.")
    if not recommendations:
        recommendations.append("Your password is strong!")
    return recommendations

def analyze_password(password):
    common_passwords = {"password", "123456", "qwerty", "admin", "letmein"} # Add more as needed
    # Entropy calculation
    entropy = calculate_entropy(password)
    print(f"Entropy: {entropy:.2f} bits")

    # Pattern checks
    patterns = check_patterns(password)
    print(f"Detected Patterns: {patterns if patterns else 'None'}")

```

```

# Dictionary check
dictionary_flag = dictionary_check(password, common_passwords)
if dictionary_flag:
    print(dictionary_flag)
# Recommendations
recommendations = generate_recommendations(entropy, patterns, dictionary_flag)
print("Recommendations:")
for rec in recommendations:
    print(f" - {rec}")

if __name__ == "__main__":
    password = input("Enter a password to analyze: ")
    analyze_password(password)

def create_gui():
    # Create the main window
    window = tk.Tk()
    window.title("Password Strength Analyzer")
    window.geometry("400x400") # Width x Height
    # Label for Password Entry
    password_label = tk.Label(window, text="Enter Password:")
    password_label.pack(pady=10)
    # Password Entry Field
    password_entry = tk.Entry(window, show="*", width=30)
    password_entry.pack(pady=5)
    # Result Area
    result_text = tk.Text(window, height=15, width=45, wrap="word", state="disabled")
    result_text.pack(pady=10)
    # Analyze Button
    analyze_button = tk.Button(window, text="Analyze Password", command=lambda: analyze_password_gui(password_entry, result_text))
    analyze_button.pack(pady=10)
    # Run the GUI main loop
    window.mainloop()

def analyze_password_gui(password_entry, result_text):
    # Clear previous results
    result_text.config(state="normal")
    result_text.delete(1.0, tk.END)
    # Get password from entry field
    password = password_entry.get()
    # Sample set of common passwords (expand as needed)
    common_passwords = {"password", "123456", "qwerty", "admin", "letmein"}
    # Calculate Entropy
    entropy = calculate_entropy(password)
    result_text.insert(tk.END, f"Entropy: {entropy:.2f} bits\n\n")
    # Pattern Check
    patterns = check_patterns(password)
    result_text.insert(tk.END, f"Detected Patterns: {patterns if patterns else 'None'}\n\n")
    # Dictionary Check
    dictionary_flag = dictionary_check(password, common_passwords)
    if dictionary_flag:
        result_text.insert(tk.END, f"{dictionary_flag}\n\n")
    # Recommendations
    recommendations = generate_recommendations(entropy, patterns, dictionary_flag)
    result_text.insert(tk.END, "Recommendations:\n")
    for rec in recommendations:
        result_text.insert(tk.END, f" - {rec}\n")
    # Disable editing in result text box
    result_text.config(state="disabled")

```

Step 6: Run and Test the GUI

1. Save and exit the file in the terminal (with `Ctrl + O`, `Enter`, then `Ctrl + X`).
2. Run your script:

```
python3 password_analyzer.py
```

You should see a GUI window open with the following elements:

- A field where you can enter a password.
- A button to analyze the password.
- A text area displaying the entropy, patterns, and recommendations.

Step 7: Test Your GUI Application

Try entering different passwords to see how the results update in the text area. The GUI should display:

- **Entropy:** The calculated entropy value in bits.
- **Detected Patterns:** Any patterns like repeated or sequential characters.
- **Recommendations:** Tips on how to make the password stronger.

```

(root@kali)~/theconsoler
# python3 password_analyzer.py

File "/home/theconsoler/password_analyzer.py", line 103
common_passwords = {"password", "123456", "qwerty", "admin", "letmein"
                    ^
SyntaxError: '{' was never closed

(root@kali)~/theconsoler
# nano password_analyzer.py

(root@kali)~/theconsoler
# python3 password_analyzer.py

Enter a password to analyze: Brother@007
Entropy: 72.10 bits
Detected Patterns: None
Recommendations:
- Your password is strong!

(root@kali)~/theconsoler
# nano password_analyzer.py

(root@kali)~/theconsoler
# python3 password_analyzer.py

Enter a password to analyze: Brother@007
Entropy: 72.10 bits
Detected Patterns: None
Recommendations:
- Your password is strong!

```

The GUI application, titled "Password Strength Analyzer", has a window with a title bar containing standard Linux window controls. Inside the window, there is a label "Enter Password:" above a text input field. The input field contains the password "Brother@007" represented by asterisks. Below the input field is a large text area displaying the analysis results: "Entropy: 72.10 bits", "Detected Patterns: None", and "Recommendations: - Your password is strong!". At the bottom of the window is a button labeled "Analyze Password".

This completes your password analyzer with a GUI.